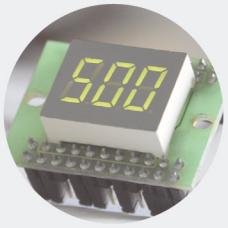
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Implementing and Tuning an Autonomous Racing Car Testbed

DANIEL GONÇALVES ALMEIDA
novembro de 2019

POLITÉCNICO
DO PORTO

ISEP

Instituto Superior de Engenharia do Porto

Implementing and Tuning an Autonomous Racing Car Testbed

Master Thesis

To obtain the degree of master at the
Instituto Superior de Engenharia do Porto
public defend on November 26 by

Daniel Gonçalves Almeida
1140452

Degree in Electronics and Computer Science - Automation and Systems
Porto, Portugal.

Supervisor:

Prof. Dr. Ricardo Severino

ISBN ++-++++-+++-+

Author email: 1140452@isep.ipp.pt

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Dr.Ricardo Severino, for giving me the opportunity to work in such enthusiastic project at CISTER and for sharing his knowledge, support, patience, supervision, availability throughout the course of this project. Thank you for always pushing me and my colleagues to achieve our objectives.

A thank you to the CISTER infrastructure and all their collaborators, from professors to staff, as they provide such outstanding conditions of work. I got to understand the dimension and prestige this institution is.

I want to thank all my friends that I have made at CISTER, from the long nights of work, to the more stressful moments, no day would pass without a laughter. Thank you all for the great moments and for giving your support and mutual help whenever it was necessary. I will never forget the days that I have spent here with you all.

A special thanks to my two colleagues Guedes and Vieira, that have become two of closest friends for the past 5 years and whose work I got to share and learn. Without you two nothing would be possible.

To the oldest and closest friends and all the colleagues I've came across during my years at ISEP, a thank you for the support and for the knowledge shared.

I would also like to thank my family for the education, support and advice they provided me along my entire life, without their sacrifice I would not be completing this thesis.

Last, but not least, I would like to thank Joana, for her love, encouragement and for keeping me on the right path during my academic years, by advising me in all my doubts, without you I would have not reached this point and would not be the person I am today. No words are enough to express my gratitude and love for you.

Abstract

Achieving safe autonomous driving is far from a vision at present days, with many examples like Uber, Google and the most famous of all Tesla, as they successfully deployed self driving cars around the world. Researchers and engineers have been putting tremendous efforts and will continue to do so in the following years into developing safe and precise control algorithms and technologies that will be included in future self driving cars.

Besides these well known autonomous car deployments, some focus has also been put into autonomous racing competitions, for example the Roborace. The fact is that although significant progress that has been made, testing on real size cars in real environments requires immense financial support, making it impossible for many research groups to enter the game.

Consequently, interesting alternatives appeared, such as the F1 Tenth, which challenges students, researchers and engineers to embrace in a low cost autonomous racing competition while developing control algorithms, that rely on sensors and strategies used in real life applications.

This thesis focus on the comparison of different control algorithms and their effectiveness, that are present in a racing aspect of the F1 Tenth competition. In this thesis, efforts were put into developing a robotic autonomous car, relying on Robot Operative System, ROS, that not only meet the specifications from the F1 Tenth rules, but also allowed to establish a testbed for different future autonomous driving research.

Keywords: Autonomous Driving, Control Algorithms, F1 Tenth, Robotic Testbed, ROS.

Resumo

Obter uma condução autónoma segura está longe de uma visão dos dias de hoje, com exemplos como a Uber, Google e o mais famoso deles todos, a Tesla, que já foram globalmente introduzidos com sucesso. Investigadores e engenheiros têm colocado um empenho tremendo e vão continuar a fazê-lo nos próximos anos, a desenvolver algoritmos de controlo precisos e seguros, bem como tecnologias que serão colocados nos carros autónomos do futuro.

Para além destes casos de sucesso bem conhecidos, algum foco tem sido colocado em competições de corridas de carros autónomos, como por exemplo o Roborace. O facto é que apesar do progresso significante que tem sido feito, fazer testes em carros reais em cenários verdadeiros, requer grande investimento financeiro, tornando impossível para muitos grupos de invetigação investir na área.

Consequentemente, apareceram alternativas relevantes, tal como o F1 Tenth, que desafia estudantes, investigadores e engenheiros a aderir a uma competição de baixos custos de corridas autónomas, enquanto desenvolvem algoritmos de controlo, que dependem de sensores e estratégias usadas em applicações reais.

Esta tese foca-se na comparação de diferentes algoritmos de controlo e na eficácia dos mesmos, que estão presentes num cenário de corrida da competição do F1 Tenth. Nesta tese, foram colocados muitos esforços para o desenvolvimento de um carro autónomo robótico, baseado em *Robot Operative System*, ROS, que não só vai de encontro às especificações do F1 Tenth, mas que também permita estabelecer uma plataforma para futuras investigações de condução autónoma.

Palavras-Chave: Condução Autónoma, Algoritmos de controlo, F1 Tenth, Plataforma Robótica, ROS.

Contents

List of Figures

List of Tables

Acronyms

ADAS	Advanced Driver Assistance Systems
AHRS	Attitude Heading Reference System
AMCL	Adaptive Monte Carlo Localization
ALV	Autonomous Land Vehicle
BARC	Berkeley Autonomous Race Car
CISTER	Research Center in Real-Time & Embedded Computing System
DARPA	Defense Advanced Research Projects Agency
EKF	Extended Kalman Filter
ESC	Electronic Speed Controller
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
ICARUS	Interest group on Cooperative Autonomous Reliable Systems
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
LIDAR	Light Detection And Ranging
MCL	Monte Carlo Localization
MIT	Massachusetts Institute of Technology
MPC	Model Predictive Controller
MPPI	Model Predictive Path Integral
OS	Operative System
OSRF	Open Source Robotics Foundation
PCB	Printed Circuit Board
PD	Proportional Derivative
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation

RC Radio Controller

ROS Robot Operative System

Rviz ROS Visualization

SAE Society of Automotive Engineers

SLAM Simultaneous Localization and Mapping

UKF Unscented Kalman Filter

USB Universal Serial Bus

1 Introduction

During the course of these last decades, the automotive industry has improved tremendously in various aspects, with the discovery of new and improved technologies, whether it is in fuel consumption efficiency, design, driving assistance and safety improvement. This last point has been a major target of interest, throughout the years, in the scientific community, since ensuring the drivers, as well as pedestrians safety, is still a great challenge. As most accidents happen due to human negligence, fatigue or deprecated safety systems, developing technologies that aid the driver and ensure road safety are a demand in current days. Although many efforts have been made, according to the World Health Organization [14], approximately 1.35 million people die each year as a result of road traffic accidents.

To address this issue, many have envisioned systems that would automatically drive a car, in the safest way possible, ensuring the protection of every being involved. Thus the creation of the self driving car, also denominated as autonomous cars. Autonomous cars have been idealized as the future of navigation in cities, where the passengers will be taken to their destination, without even a press of a pedal or a steering of a wheel. In the present days, autonomous driving is not a vision nor fictional idea, as many autonomous car have been deployed in cities around the world, with great success. However, these systems are of an enormous complexity, with much work still to be done, until fully autonomous cars driving around, with full awareness of the environment surrounding them is achieved.

As a result, the development of technologies has been rising immensely among sci-

entist and engineers. Many different research topics regarding autonomous vehicles have been explored in the latest years, whether about safety conditions, obstacle avoidance, communication between vehicles, control algorithms, among many others. Although this statement is true, it is still hard for many research groups to enter this game, as it requires immense financial support. To overpass such drawback, attentions were put into small, low-cost platforms, that support the implementation of the same approaches applied in full size autonomous cars

As a major target of interest in the scientific community, this accelerates the pursuit to achieve better and more efficient results, awakening the competitive aspect among researchers, engineers and students. From that competitive spirit the F1 Tenth competition arose, challenging participants to develop algorithms for autonomous driving, while adopting their robotic platform.

1.1 Context

In the context of robotic applications for autonomous driving, this thesis was developed at Research Center in Real-Time Embedded Computing Systems, CISTER, in connection with the SafeCop European project and the ICARUS interest group (Interest group on Cooperative Autonomous Reliable Systems).

This testbed provides an autonomous vehicle that allows the development, testing and validation of different technologies, from vision and control algorithms to embedded systems that guarantee safety for Advanced Driver Assistance Systems, ADAS. In addition, the platform follows the architecture of the F1 Tenth competition, aiming at future participation in such competitions.

1.2 Research Objectives

The pivotal objective of this thesis is to build and develop a robotic testbed to compare different control algorithms in a racing environment. The main focus, is to develop a testbed, to serve as a baseline platform, to implement, test and validate different tools, regarding ADAS and other autonomous driving components.

To attain it, a robotic platform, respecting the F1 Tenth competition rules, needs to be developed. An additional objective and motivation is the development of an autonomous racecar that aims to qualify and compete in the upcoming F1 Tenth competition.

1.3 Research contributions

The main research contributions of this thesis are:

- Implementation of a low-cost robotic testbed based on ROS, that will allow to test and validate different technologies for increased safety in autonomous vehicles.
- Implementation, evaluation and improvement of different control algorithms in terms of trajectory, time and degrees of correction.

1.4 Thesis Structure

The remainder of this thesis is organized as follows. On Chapter 2, the state of the art will be outlined, encompassing an overview of the autonomous driving panorama history and an outlook of some existing robotic platforms and their approaches to autonomous driving.

The following two chapters describe all the technologies and tools used in this thesis, as well as the system architecture, providing an overview of the components, in terms of the Hardware and Software.

The fifth chapter concerns the algorithms and strategies implemented in this thesis, while the sixth demonstrates the results achieved, of each of the methods applied.

The thesis will finish with Chapter 7 that will present the major conclusions and projections to future work related to this project.

2

State Of The Art

In the latest years tremendous work and study has been put into the development of autonomous vehicles. Having the perception of its surroundings, planning a path and controlling its movements are key factors when creating an autonomous vehicle. Depending on the type of sensors adopted, it is possible to obtain different types of perception, that can affect the complexity of the control algorithm.

In this section, the state of the art of scale autonomous vehicles platforms is presented, overviewing different testbeds developed and the implementation of their control algorithms.

2.1 Autonomous Driving

In the current state of society, we have reached a point where having a car is almost inherent in a family. With the continuous rise of the population, the number of cars per city has also risen. As consequence, the number of causalities in car accidents increased. Although modern cars provide several features that assist the driver and provide better safety conditions, compared to previous decades, most of the accidents happen due to human error. It is estimated, by the World Health Organization, that 1.35 million fatalities result from traffic accidents [14]. Therefore, to address such significant numbers, huge investments have been made over the last century to the development of autonomous vehicles.

With the introduction of autonomous vehicles, or also described as self-driving, or

driver-less cars, it is believed that 10 million lives could be saved per decade around the world, as mentioned by the author in [15]. Achieving automotive autonomy not only will bring immense improvements to the quality of life of people, but specially to road safety. However, this process is still in development, until fully automated vehicles are present in mass scale in our society.

The dream of achieving an intelligent system, capable of driving a car on its own, without human intervention, has increased greatly throughout the last decades. Many experiments were carried out, trying to implement an effective solution.

In [16], Keshav Bimbraw presents a survey of the development of autonomous vehicles from the past and current century and future approaches on this topic. A brief resume from it is exposed below, to provide the advancements of this technology throughout the years.

As mentioned in [16], the first appearance of a semi-autonomous car, dates back to the 1920's, with the appearance of the radio controlled car. The Linriccan Wonder consisted in a car with an antenna, on its back and was operated by a follower car, by sending radio impulses. Upon receiving the radio commands, the on board electronics actuated on the car electric motors and the cars direction was controlled.

Fast forward in time, it was presented in 1953 a small scale car, controlled by wires disposed in a certain pattern in a laboratory and in 1958, the idea was reproduced in a larger scale, into a highway and were able to detect the presence and velocity of a metallic vehicle and guide it [16].

After years of experiments in highways with guided systems, [16] in the 1980s, it was designed in the Bundeswehr University, Munich, Germany, a vision guided car that achieved 63 km/h on the streets. The Defense Advanced Research Projects Agency of the U.S. Department of Defense, DARPA, contributed with the ALV, Autonomous Land Vehicle, which made use of computer vision, LIDAR to achieve the first road following robotic vehicle. Alternatively, an off-road map and sensor based navigation on the ALV was implemented by the HRL Laboratories.

Although, semi-autonomously [16], in 1991 it was presented 2 robot vehicles that drove more than a thousand kilometers, with traffic addition and reaching up to 130 km/h. Also, it was demonstrated other features such as lane changing and convoy driving. In 1995, an autonomous S-Class Mercedes Benz reached 95% autonomous driving, on almost 1600 Km, with resource to computer vision and microprocessors that could react in real time.

An approach with neural networks was undertook in 1995, called the Navlab project. Although it reach 98% autonomous driving, on a 5000 km route, the car was semi-autonomous. The neural networks were only applied on the steering control,

while the acceleration and braking were done manually. By 1996, it was launched the Argo Project that applied an algorithm, with resource to camera vision, to follow lane marks on a normal highway. This car, ended up being fully autonomous 94% of the test it carried out [16].

By the arrival of the millennium, projects with military purpose surged from the US government. These Demo projects, were capable of roaming autonomously through aggressive terrain, while avoiding obstacles and also demonstrate real time control system [16].

In more recent years, with the improvement of technology and the fast growing interest on driver-less cars, more investment was made and thus, more projects related to autonomous car surged. Developments in this area, revealed successful implementations like the VisLab Intercontinental Autonomous Challenge, in 2010, which consisted in a autonomous trip form Italy to China. The 3 month journey, with a level of autonomy, proved to be possible in future years to safely transport goods, neglecting human intervention. During the same year and further on, another project was carried out, with an Audi TTS, focusing on safety functions in autonomous driving. Resourcing to laser scanners and other sensors, the project aimed to prevent accidents from distracted drivers. Other projects were also conducted, with the intention of integrating safety systems, in a urban and highway environments, while relying in image processing and state of the art sensors. By 2014, it was showcased by Toyota and Nissan their proposals of autonomous vehicles and presented effective navigation and solid reference for future cars, as they set a base to test numerous scenarios. Being the LIDAR one of the most important sensors in autonomous navigation, Navya introduced to an electric shuttle that uses 4 of them, in conjunction with optical cameras to generate real-time map of its surroundings and navigate at slow speeds.

Although tremendous improvements have been achieved in regards to autonomous driving during all these years, much work still needs to be done, to attain the best and safest autonomous cars. Emphasising on the safety aspect, as humans lives are and will be held by robotic cars, that think on their own and constant flaws are detected. In 2016, this was proved, as the first fatal accident with an autonomous car occurred, due to sensors not distinguishing a white truck. Although it is proven that self driving cars will prevail in comparison to human driving, since they react faster and can sense and adapt to situations a human driver can not, this accident has proven that machines are not a perfect system and have flaws.

Nowadays, with much investments from big automotive companies, more environmental friendly cars are put into the market, that provide the drivers with highly

autonomous functions that aid the driver or give the possibility of fully autonomous driving. Aspects such as, self-parking, lane keeping, collision avoidance, cruise control and various other driver assist functions are now part of these generation of cars. It is only by 2035, that it is expected that most of the cars driving around our cities will not require any human action, as they will be fully autonomous.

To understand the concept of autonomous vehicles the international automotive organization SAE International (Society of Automotive Engineers) specified 6 levels of driving automation [1]. This levels determine how autonomous a vehicle is and the requirements and precautions of the human driver.

Level 0, No automation. This level affirms that no automation is present, where driving only relies in human actions in every situation and the system only provides warnings or momentary assist. An example of it is lane departing warnings and is mostly present in a old generation of cars.

Level 1, Driver assistance. This level is responsible for assisting the driver with a specific task. Examples of it are lane centering or cruise control, since the system is capable of taking control of an action, like steering or braking.

Level 2, Partial automation. The system can accomplish two or more automatic task, working along side to assist driver, however it is not yet considered an autonomous car and the driver is required to maintain the car under control and take the necessary actions if necessary. The car performs functions such as accelerating or decelerating and steering at the same time. Examples of are same of level 1, however they work along side and the driver is responsible to supervise.

Level 3, Conditional automation. Up from this level, the one driving is the car. The system is responsible for the normal actions in driving and only requires the person in the driver seat to take measures, if the system requires assistance or an emergency circumstance appears. That being said, although the car is mostly driven automatically, the driver must at all time analyze traffic and road conditions and be alert and ready to take control when the system demands. For instance, these features can be found in autonomous highway driving or in traffic congestion.

Level 4, High automation. The vehicle is able to monitor the environment and can operate fully autonomously in many different driving scenarios. Besides, contrary to the previous level, the system is capable of protecting its passengers from accidents, even if it required and the driver does not properly intervenes on time. If the system finds that not all conditions are met, then the driver must assume control, until autonomy can be defined again.

Level 5, Full automation. Equal to level 4, however the system car drive the car in all driving scenarios and conditions. It can replicate human actions and may

exceed at it. All emergency situations are dealt autonomously and never requires the human to take actions or to supervise. No pedals and steering are necessary to be installed.

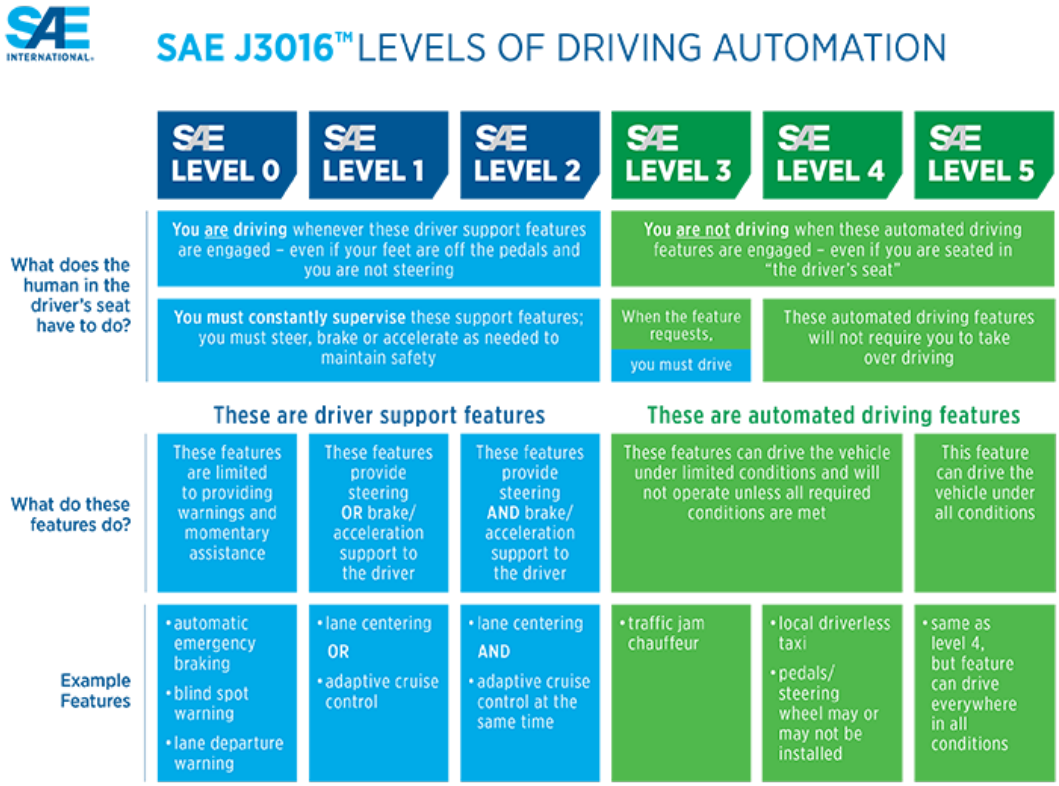


Figure 2.1: *Graphical summary of SAE 6 levels of driving automation[1].*

To summarize, as it can be seen in Figure 2.1, one can affirm that from level 0 to 2, the driver is the one who is in full control of the car, it needs to be completely aware of every situation on the road and is responsible for the safety of its passengers and other drivers on the road. From level 3 to level 5, the system is the one responsible from almost to all controls of the car. The driver in the passenger seat must analyze all conditions and take control of the car if necessary, however it needs to be prepared to take action in emergency situations.

Given these points, it is also required to understand the foundations of an autonomous vehicle system. To safely navigate and reach every destination, it is required by the car to perceive the environment, apply the necessary control and be always alert to unexpected situations. Figure 2.2 represents the conceptual structure of an autonomous vehicle system. In [17] and [2] this architecture is studied and clarified along the descriptions of each part below.

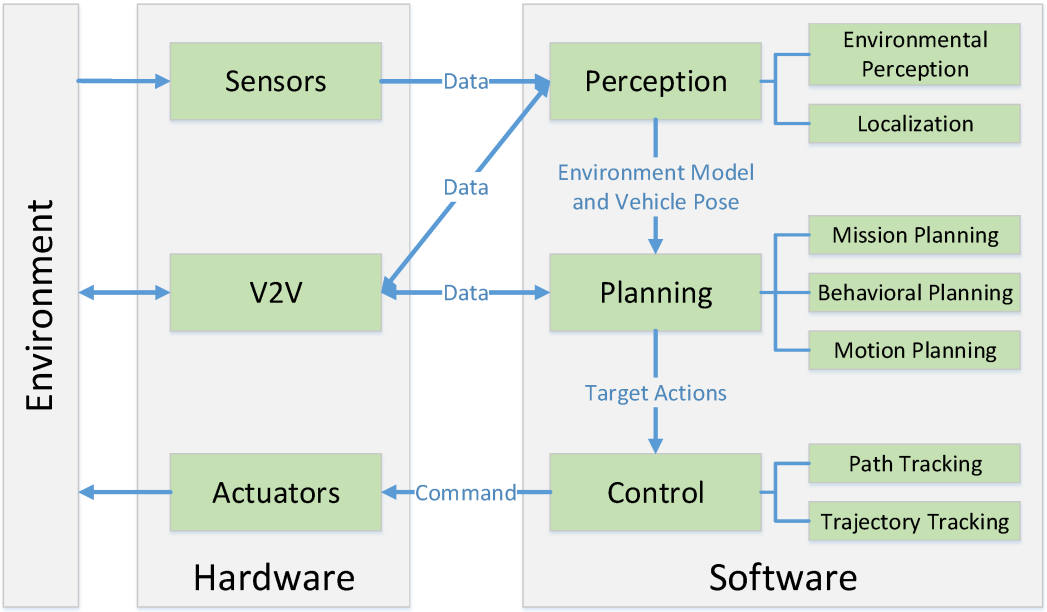


Figure 2.2: *Conceptual diagram of an autonomous vehicle system[2].*

The perception refers to all the information derived from sensor input, that are present in the car. This sensors can vary and each is responsible to understand the environment surrounding the car in a different way. In [18] an overview of recent technologies applied in current and future autonomous vehicles. This sensor range from ultrasound sensors to long and short range finders, that are responsible for detecting obstacles and assist in small tasks such as parking assist. Image processing, is responsible to a large extent of perception. Through stereo cameras, the system is capable of detecting and perceive other cars, people, obstacles and the road layout to provide enough information for a correct path planning. The Lidar, is a faster spinning approach to range finders, that permit to acquire a cloud of points from reflected light on objects, in a surrounding area. Although at present time, Lidars are still very costly, with such sensor at disposal, it is possible to map a static environment and detect different moving or stationary obstacles, such as pedestrians and cars.

Data sensor shall then be transferred and processed by mapping and localization algorithms. Significant work has been done over the last decades to develop and improve such algorithms, in order to precisely monitor and track car movement and positioning. [19] examines different probabilistic methods used in numerous robotic applications, such as Gaussian filters, like the Extended Kalman Filter, EKF, Nonparametric filters, like the Particle Filter, occupancy grid and mapping, as well

as Monte Carlo Localization, MCL, the Simultaneous Localization and Mapping, SLAM, approach and other derivations of it, along with other approaches as well. In terms of SLAM trends in autonomous driving, [20] reviews variations of it in different experiments, while [21] explores procedures of localization techniques.

With local information of the surroundings of the car at disposal, from the sensors, it is possible understand where a vehicle is situated in a known map. While achieving accurate localization of the car, the possibilities to reach a desired destination are immense, surging the path planning and decision making aspect of the autonomous driving, that at the same time is directly correlated to the motion control of the vehicle. These methods proceed to decide the optimal path for the vehicle to safely reach its target, while at the same applying the correct actions on the car controllers to actuate upon the steering and engine. Likewise, various algorithms to achieve motion planning have been implemented over time, always taking into consideration the vehicle model. In [22] a survey of motion planning and control techniques is presented exploiting various algorithms to accurately achieve the a desired objective and practical examples, as well as path stabilization, with approaches like the Pure Pursuit and Model Predictive Controller, MPC.

2.2 Scale Autonomous Vehicles Testbeds/Platforms

2.2.1 F1 Tenth

The F1 Tenth (F1/10) Autonomous Cyber-Physical Platform [23] is an Open-Source platform that intends to provide Researchers and Students a testbed to simulate and test various approaches to autonomous driving, by using a 1 to 10 scale of a real car. Adaptive to different case of studies, this testbed provides a compact work tool, explaining how to build a car, refereeing the required components, the software necessary to install that is centered in Robot Operative System, ROS, different tutorials explaining the distinct approaches to autonomous driving and the packages necessary to easily implement it, as well as a simulation environment based on ROS-Gazebo.

By providing an easy to implement testbed, it enables the user to focus on various approaches of investigation and testing. Using its tools, one can focus on the improving localization and perception of the car, using sensors like cameras and Lidars.

Other methods like vehicle to vehicle communication can be explored, studying scenarios like approaching an "roundabout" or integration in a platoon, that have been heavily investigated in the latest years.

On focusing this thesis objective, the F1 Tenth organizes international competitions to evoke the competitive spirit around students and engineers. This competition challenges the participants on developing their own algorithms in order to encounter which team can achieve the fastest lap and most number of laps completed in the fixed time. In recent editions, a head-to-head competition was also held, challenging the participants to not only autonomously drive around the track, but to race against another car and dodge obstacles. This competition has great interest around the community as every participant has to use the same type testbed and thus making every team focus solely on the algorithms. This exalts the pursue in developing in efficient algorithms that can be applied autonomous driving.

Up until the submission of this thesis, the latest competition being held was in April 2019, in Montreal. The winners of both challenges presented a simple, but different approach in autonomous driving, which they called the "Disparity Extender" algorithm [13]. In further chapters this algorithm developed by the UNC-Chapel Hill team will be explored and explained into detail.

To conclude, as a result of the F1 Tenth testbed, it will be possible to explore this thesis purpose on comparing and validating different racing algorithms.

2.2.2 Formula PI Testbed

A different approach to an autonomous racing competition is the Formula Pi competition [4]. Created by Timothy Freeburn [24] this competition aims to attract people to the development of self-driving robots with little experience regarding hardware and software. The competition, as the name suggests it, is based around a Raspberry Pi where the model for the robot is "Monsterborg", a robotic model developed by PiBorg [3]. This model, like the one in Figure 2.3, is a small scale robot that features:

- 4x high-torque 300 RPM metal geared 37mm motors;
- 105mm / 4 inch diameter off road wheels;
- ThunderBorg - A Dual Motor controller designed to attach to a Raspberry Pi and to handle up to 5A per motor connection. It can be stackable and controls motors via Pulse Width Modulation (PWM) signals. It uses Inter-Integrated Circuit, I2C, SDK/SDA for communication;
- Can come with a Pi camera: A 8 Megapixel Camera ideal for beginners for image detection and processing.

- A 10x AA battery pack that gives a 3 hour autonomy for powering the platform.

Figure 2.3: *Monsterborg - The robotic platform used in the Formula Pi competition [3]*

The particularity of this competition is centered on the fact that the participants are not required to buy and assemble the robot, as the organizers have the all the robotic platforms, leading the software to be the main source of focus. Competitors send in their codes to the organizers, that will be put to test against other opponents during their summer and winter series. The race consists of 5 robots at a time to make a total of 23 laps around the track that can be seen in Figure 2.4.

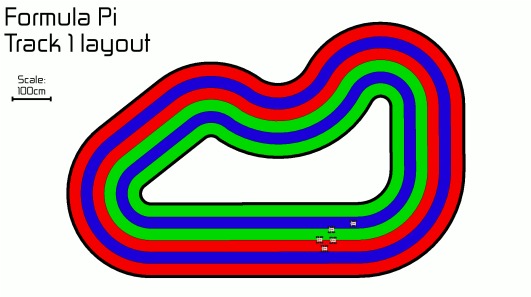


Figure 2.4: *Formula Pi race track [4].*

As mentioned before, the robot is based around a Raspberry Pi that is responsible for processing its surroundings and act upon the motor controllers. For the perception aspect, it uses only the camera as a source of information. The algorithm [25] is written in Python and uses the OpenCV libraries for image processing. Depending on the Camera processed information, the code applies the motors the necessary power, via PWM, to drive around the track.

An example code for racing, uses the track distinct colors to detect a lane depending on the colour chosen to follow. Then it detects two points in the track, the closest and farthest points and it will plot a line linking those points, indicating the target direction. In terms of steering the example relies on two Proportional Integral Derivative, PID, controllers to apply a steering control to the car. One based on the offset from the center of the track, and the other based on how far the track position changes between the two points. The speed is set by a direct PWM signal.

In conclusion, the Formula Pi is a noble competition to attract newcomers to the fields of robotics, that have appreciate a challenge, using very low-cost hardware. It also introduces to image processing with OpenCV, giving a learning aspect on how to use a camera to analyse the robot surroundings. However, the fact the its a compact and low cost platform, it utilizes inferior components compared to other autonomous racecar robots, never achieving the processing power and performance as other platforms.

2.2.3 Autorally

The Autorally project [26] is a 1:5 scale open source platform developed at Georgia Institute of Technology, in Atlanta, dedicated to aggressive autonomous driving. Focusing on autonomous driving, this compact and robust platform implements a variation of a model predictive controller, MPC, that relies on accurate dynamics models for motion prediction.

In terms of its architecture, the Autorally robot uses a 1:5 scale Radio-Controlled, RC, Truck, like the one shown in Figure 2.5, that had modifications to enclosure all the hardware component, as a protective measure. To achieve accurate measurements of the positioning of the robot, it uses a combination of 3 sensors. Hall-effect sensors and magnets in a circular pattern were used to measure the wheel speeds, and by calculating timing information between magnets, that is later translated to rotation rates. Likewise, a high precision Global Positioning System, GPS, and a Inertial Measurement Unit, IMU, are used. The GPS provides absolute precision at 20 Hz, accurate to approximately 2 cm under ideal conditions with real-time kinematic corrections from a GPS base station. For the IMU, it was used a Lord Microstrain 3DM-GX4-25 IMU provides raw acceleration and angular rate data at 200 Hz (maximum 1 kHz) and fused orientation estimates at 200 Hz (maximum 500 Hz).

To compute all information, the setup uses: Asus Z170i pro Gaming, Mini-it X; CPU: Intel i7-6700, 3.4 Ghz quad-core 65 W; RAM: 32 GB ddR4, 2133 Mhz; GPU: Nvidia GtX-750ti sc, 640 cores, 2 GB, 1176 Mhz; Memory: SSD 512 GB M.2 and 1

tB sAtA3; Wireless: 802.11ac Wi-Fi, 900 Mhz XBee, and 2.4 Ghz Rc; Power supply: Mini-Box M4-AtX, 250 W; Battery: 22.2 V, 11-Ah lithium-polymer, 244 Wh. The system runs on Ubuntu 16.04 and all the software used developed in ROS Kinetic.

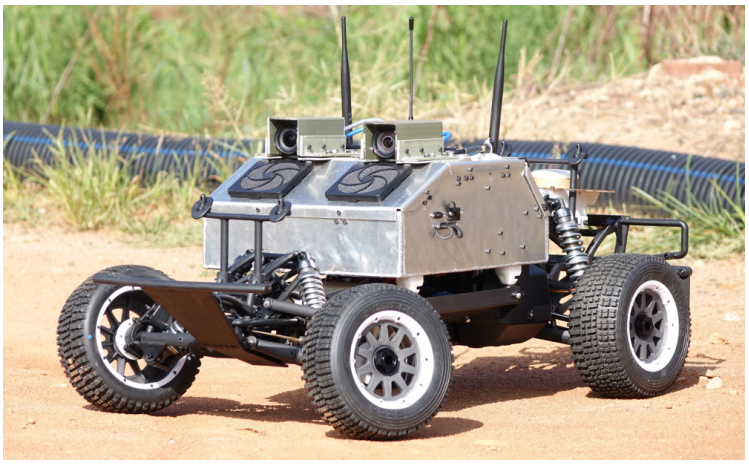


Figure 2.5: *Autorally robotic platform [5].*

In conjunction with the physical platform, a simulation environment is also explored, developed in Gazebo, that allows to carefully control environmental parameters for gathering statistical data, which requires performing repetitive or time-consuming experiments.

Covering the control aspect of this platform, a variation of MPC was adopted, the MPPI, which stands for Model Predictive Path Integral control. MPPI [27] is a sample-based, derivation free approach to model predictive control (MPC) method that can drive AutoRally up to, and beyond, the friction limits of the track. The MPC intersperses optimization and execution, firstly optimising an open-loop control sequence in a defined finite time. Then, it executes the first control sequence, sending the feedback state and the optimization process is repeated.

The Path integral optimal control framework grants a mathematical methodology to develop optimal control algorithms given on stochastic sampling trajectories. Thousands of trajectories are sampled from a importance sampling distribution are used to estimate the optimal control.

The MPPI assumes that are given the system dynamics, initial control sequence and a cost function for the given task. This algorithm, in each iteration, uses optimal control sequence from previous ones and receives the sampled trajectories and generates new sequences of control inputs. These control sequences are then propagated forward in the state space using the system dynamics, and each trajectory is evaluated according to a cost function. The estimation of the optimal control se-

quence is then updated with a cost-weighted average over the sampled trajectories. State feed-back is then introduced to begin the next iteration.

All the real time computation involved in this implementation for sampling-based MPC is to produce a large number of samples in real time and it is done in parallel, using the sampling step on a Nvidia GPU, using Nvidia CUDA architecture.

In [28] this implementation was tested in a aggressive driving scenario, proving to be successful when vehicle maneuvered around the track. In [29] the MPPI was improved to solve model-based reinforcement learning tasks using multi-layer neural networks as dynamics models.

This type of approach takes in consideration perturbations that affect the dynamics of the car model when driving around a track, providing the best control actuation's to tackle the track. Taking that into account, this methodology makes it ideal for a time-trial type of race, as results show that the robot is able perform at high speeds while achieving the best results in trajectory in time in a lap. However, this testbed could not perform in a head-to-head competition, as it lacks in the ability to avoid collision, since the pivotal point is to drive the robot around a known and static map, but never taking in consideration a dynamic obstacle that leads to a great change of direction.

2.2.4 The BARC Project

In autonomous driving and racing, researchers approach different cases of study. A specific case of study involves maneuvering a car when drifting and thus a robotic platform called the Berkeley Autonomous Race Car, BARC [30], was developed.

Equivalent to the F1/10 platform, this project is based on a 1:10 RC car and aims to create a platform that can achieve complex maneuvers in autonomous car such as drifting and obstacle avoidance. In terms of physical hardware, the car uses a brushless motor, a servo motor , an Electronic Speed Controller, ESC, and a LiPo battery to supply power to the on-board electronics. Regarding the sensors used in BARC, measurement sensors such as an IMU, camera, range finders and encoders were applied to the platform. Distinct to the F1/10, this platform does not rely on a Lidar, however resourcing on the encoders on the wheels, it can provide precise information of velocity and positing, in conjunction with the IMU. For computational processing, the platform uses an ODROID-XU4, an on-board ARM based computer and an Arduino Nano to interface with the actuators. Figure 2.6 shows the fully assembled BARC platform.

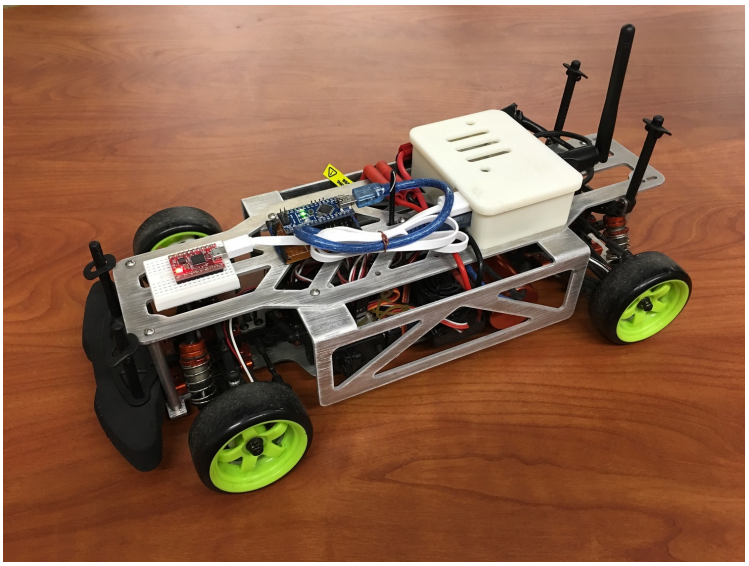


Figure 2.6: *Berkeley Autonomous Race Car platform [6]*

This project aimed to tackle the difficulty of challenging maneuvers, even for expert drivers, like drifting in corners. The authors in [31] proposed an algorithm for an autonomous corner drifting, while mixing open and closed-loop control strategy. To achieve it the system model is outlined as a six-state bicycle model with linear front and rear-wheel tire forces. Next, an explanation for the optimal path planning is presented and the control law to applied laid out, mixing the open-loop and closed-loop controller, following a rule based algorithm. In the end, the results are demonstrated both in simulation environment and experimentally on the platform.

Although, no demonstration of this platform in a racing environment, this testbed possesses the necessary components to deliver a good performance, since it has good odometry for pose estimation, being possible to implement complex algorithms such as a Model Predictive Control, MPC, or a conjunction of waypoints in a known track with a local planner, or more classic approaches like Lane detection or PIDs. In addition, the implemented corner drifting algorithm would aid in a racing scenario, however acquisition of the car position would need to altered, as the authors use a indoor GPS kit which uses ultrasonic beacons to localize the vehicle.

3

Technologies and Tools

This sections present the major technologies and tools that were adopted in order to implement the desired testbed.

3.1 Robotic development frameworks

3.1.1 ROS

The Robot Operating System (ROS) [32] framework, maintained by Willow Garage and Open Source Robotics Foundation (OSRF) since 2007, is a open-source middleware that has undergone rapid development and has been widely used to design robotics applications. With its many software frameworks it provides a variety of tools, libraries and conventions that facilitates the creation of robotic applications and further encourages the sharing and reusing codes and problem solving through the robotic community. Although ROS is not a real-time framework, it is possible to integrate it with real-time code.

ROS way of working is based on a distributed framework of processes called *Nodes* that enables executables to be individually designed and loosely coupled at runtime [33]. Each node is responsible with one task and the communication between them follows the publish/subscriber model, thus providing a very simple and clean way of connecting different software and hardware components. For example, the following diagram in Figure 3.1 represents a simple demonstration system. The ellipses rep-

resent ROS nodes, where *Publisher* publishes to the *Chatter* topic, represented by a square and the *Subscriber* node subscribes to the referred topic.

Figure 3.1: *rqt_graph example of a publisher and subscriber.*

This kind of diagrams are generated by using rqt_graph tool, which automatically generates a diagram representing the current running ROS system. On further sections, when relevant, similar diagrams will appear in order to present the running nodes and explain their relation.

As explained in [34], ROS has 3 levels of concepts, divided as it follows: The Filesystem level, The Computation Graph level, and the Community level.

The Filesystem level covers the overall resources of ROS and includes the Packages, Messages and Services. The Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package. The Messages description store the messages used in each package. Service description define the request and response data structures for services in ROS

The Computation Graph level is what establishes the communication between ROS processes that are processing data together. The fundamental concepts of this level are:

- Master [35] - The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

- Nodes [36] - Nodes are processes that perform computation. A robot control system will usually comprise many nodes, that one can be inn charge of controlling a laser range finder, another reading wheel odometry, other preforming localization and so on.

- Messages [37] - A message is a data structure, comprising typed fields and is used by nodes to pass information between each other. For example a message

can carry the values of a node that reads the scan of a LiDAR and needs to share to another node to use that distance in order to generate a local map.

- Topics [38] - Messages need to be transported trough a defined route. To do that Topics are in charge of guiding the messages via a system with publish and subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption.

- Bags [39] - When developing ans testing the necessity of reproducing the same environment is crucial. Bags, created from the tool ROS Bags, are a format that store serialized message data, generally from sensors as its received. This data is then saved in the bag and can be played back the same way as any other node.

The ROS Master is the central node in the ROS Computation Graph. It stores topics and services registration information for ROS nodes. Nodes communicate with the Master to report their registration information and to receive information about other registered nodes. The Master will also make callbacks to these nodes when this registration information changes, which allows nodes to dynamically create connections as new nodes are run.

Nodes connect to other nodes directly, being the master only responsible to store and associate information. Nodes that subscribe to a topic will request connections from nodes that publish that topic, and will establish that connection over an agreed upon connection protocol. The most common protocol used in a ROS is called TCPROS, which uses standard TCP/IP sockets.

The third and last level of ROS, the Community level is responsible for sharing and providing resources, knowledge and software with distinct communities. This level provides distributions, repositories, a wiki, Q&A site for answering dedicated ROS Questions. In this project the distribution used was ROS Kinetic Kame.

Included in the extensive list of tools ROS provides two crucial tools in the development of robotic applications, being ROS Gazebo and ROS visualization, Rviz.

3.1.2 Gazebo

Gazebo [40] is an open-source 3D robotic simulator, that allows to design robots and test them using realistic scenarios, either in indoor or outdoor environments. This simulator allows the user to simulate sensors used in real applications, for example a sonar range finder, or a stereo camera, or even inertial or kinetic style sensors, always using a realistic physics engine or optionally with noise appliance. It also allows the creation of worlds with objects with different textures and realistic rendering with lighting and shadows. Since ROS can be integrated with Gazebo, it allows the development of a ROS system in a simulated scenario, compatible with nodes, messages and its remaining properties. This way with a well implemented simulation, a developer can test and validate a robotic application before testing in a real life situation, preventing damages on the robot. Different types of robots can be simulated, starting from wheeled robots, drones, Humanoids or costume designed robots. Figure 3.2 displays a Gazebo simulation from one of the F1 Tenth tutorials.

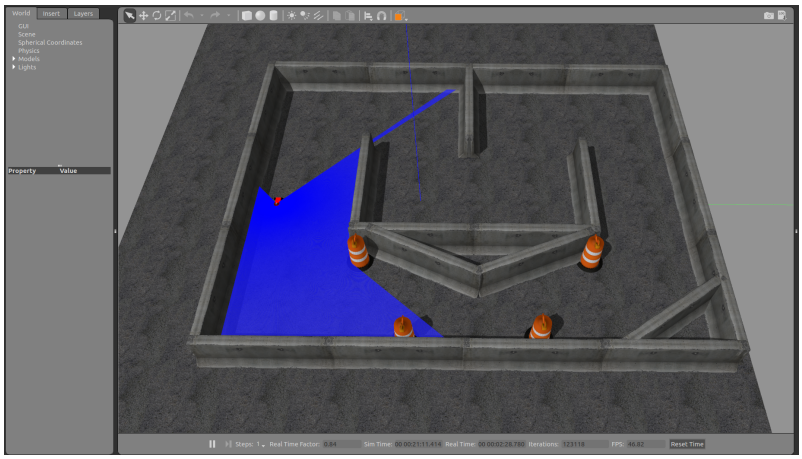


Figure 3.2: *Example Gazebo Simulation*

3.1.3 Rviz

ROS Visualization, Rviz, is a powerful 2D/3D visualization tool from ROS. Rviz provides various features for the user to visualize a robot model while tracking its movement, observed sensor information and many other aspects that are found to be relevant for the project in development. By providing this data that can be displayed from actual moment or through logged information, using ROS Bags, Rviz asserts as powerful instrument for debugging a robot application, since it allows the user to display only the information that the user requires.

Regarding sensor visualization, Rviz allows to display 3D sensor data from stereo cameras, lasers, kinetics and other 3D devices data from point clouds or depth images. In the same way, 2D sensors data can be attain from webcams or RGB cameras and laser range finders.

With ROS working and communicating with a machine running Rviz, this one will display the robots current configuration on the virtual robot model. This model interacts with sensors, through TF transforms that will receive the data from the sensors and applying to the virtual model.

Figure 3.3 shows an example of an Rviz window from one of the F1 Tenth tutorials. In this figure it can be observed the sensors, a static map and a particle filter from the estimated position.

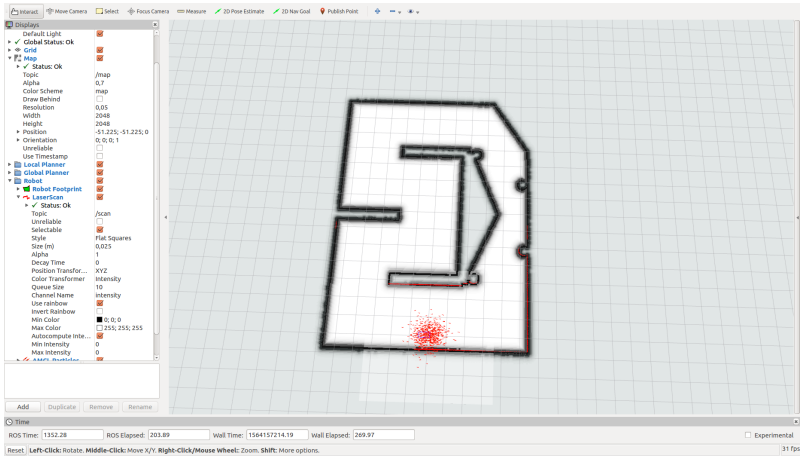


Figure 3.3: *Example Rviz window.*

3.2 Embedded computing platforms

3.2.1 NVIDIA Jetson TX2

As well as based on the F1 Tenth competition, the on-board computer that was used in this test bed was the Nvidia Jetson TX2 Developer Kit carrier board, like the one displayed in Figure 3.4. The reason for the use of it was due to its computational power, that properly meets this project needs, but also its compact format that fits in the robot. Ideal for the development of software using Linux Operative System (OS), it possesses standard connectors that provide a flexible and expandable interfaces with other modules.

Figure 3.4: *NVIDIA Jetson TX2 Developer Kit [7].*

As documented from its datasheet [41], this low-powered embedded module features an integrated 256-core NVIDIA Pascal GPU, a hex-core ARMv8 64-bit CPU complex, and 8GB of LPDDR4 memory with a 128-bit interface. It connects to 802.11 a/b/g/n/ac WLAN/Bluetooth enabled devices. This board also provides an advanced power management.

The development board offers a variety of peripherals that include:

- Gigabit Ethernet RJ45 connector;
- USB: 1x USB 2.0 Micro AB and 1x USB 3.0 Type A;
- Storage extension for a full size SD card and a SATA connector;
- Display expansion and a HDMI Type A slot;
- Expansion header that include 40-pin headers with I2C, SPI, UART, I2S, Audio clock/control and digital mic;
- GPIO Expansion header with 30-pin headers, I2S, GPIOs and digital speakers;
- Debug with JTAG connector and Serial port signals;
- User interfaces and indicators, including Leds and buttons and power supply DC jack 5.5 V - 19.6 V.

The main objective of this board is to process all the necessary information of the environment, that come from sensors connected to its peripherals and control upon the car. In addiction, although the TX2 developer kit comes with all the peripherals required, however an USB hub was necessary to add, to extend the Universal Serial Bus, USB, ports to interface with all sensors, as it only had one USB port.

3.2.2 Teensy 3.2 Micro-controller

The Teensy 3.2 [8] is a complete USB-based micro-controller development system and it features the following specifications:

- 32 bit ARM MK20DX256VLH7 Cortex-M4 (72 MHz)
- 256 kbytes RAM
- 2048 bytes EEPROM
- 34 Digital I/O
- 21 Analog Input and 1 Output with 12 bit resolution
- 12 Timers in total
- Communication: USB, I2C, SPI, Serial, CAN Bus and Digital Audio

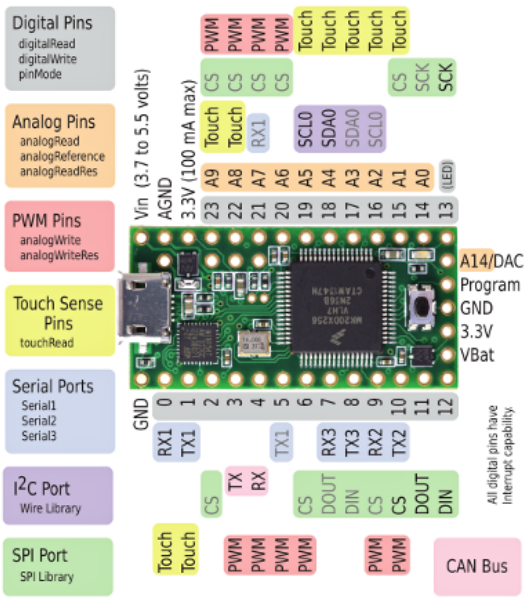


Figure 3.5: *Teensy 3.2 Micro-Controller [8].*

With a micro-controller at disposal like the Teensy, one can easily implement programs, using its compatibility with the Arduino IDE, Integrated development environment, and Teensyduino, a library for Arduino to allows to upload programs to the Teensy in use. This programs can go from reading sensors, to interfacing with other controllers and boards and thus the versatility of this board, proves to be helpful in the implementation of robotic applications.

3.3 Sensors

To obtain information about the environment and the localization of the platform, it was necessary to use sensors that gave us that information with the most accurate precision possible. The ones used and recommended were the Hokuyo LIDAR UST 10-LX [9] and the SparkFun Razor IMU [10].

3.3.1 LIDAR

The Lidar, which means Light detection and ranging, not only gives the possibility to scan the surrounding environment in a specific range, depending on the type of sensor used, but also to obtain the distance to a certain object. This device is ideal for robotic applications that required obstacle detection and localization. Its function principle is the same as a simple range finder, a light is emitted and whenever it reflects on a surface and is received by the sensor, it calculates the distance taking in consideration the time it has passed since it was emitted. Figure 3.6 displays the Hokuyo Lidar used in this testbed.

Figure 3.6: *Hokuyo LiDAR UST 10-LX [9].*

As mentioned before, the LIDAR used in this project was the Hokuyo LIDAR UST 10-LX that has the following properties: (i) measures in a wide field of view of 270 degrees; (ii) distances reach up to 30 meters; (iii) Scan speed reaches 25 m/s; (iv) Angular resolution of 0.25 degrees; (v) Accuracy is more or less 40 mm. Figure 3.7 shows its Laser scanning view. This sensor communicates through Ethernet, which

is ideal since the Ethernet port in the Nvidia Jetson is available and requires 12/24 V DC for supply voltage.

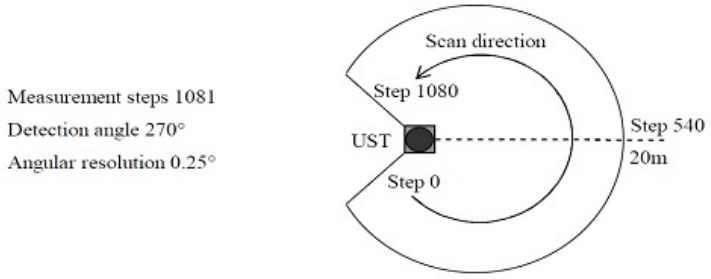


Figure 3.7: *Hokuyo LIDAR scanning view [9].*

3.3.2 IMU

Localizing the car in a known map is one the of requirements when navigating an autonomous vehicle. An inertial measurement unit, IMU, is a device that includes sensors like gyroscopes, accelerometer and sometimes magnetometer and can returns information like acceleration, angular velocity and orientation.

In this testbed the used IMU was the Sparkfun Razor IMU, represented in Figure 3.8, which comes with 3 sensors mentioned above and thus providing 9 degrees of freedom.

Figure 3.8: *Sparkfun 9d0f Razor IMU [10].*

This compact IMU sensor board features a MPU-9250 three 3-axis sensors proving acceleration, velocity, angular rotation and magnetic field vectors, along with a on-board microprocessor, Atmel's SAMD21, a 32-bit ARM Cortex-Mo+ microcontroller, that is compatible with Arduino.

3.4 Vehicle platform

In order to develop a physical test bed, a car model was necessary to be altered and adapted to have all the components placed and organized. As mentioned before, this test bed is based on the Massachusetts Institute of Technology, MIT Racecar [42], also used in the F1 Tenth competition . The car model is a Traxxas Fiesta ST Rally [11], a 1/10 scale of a real car, like the one in Figure 3.9. The versatility of the RC model, allows it to be adjusted and be built upon it, creating a well structured platform to test different scenarios.

This RC car comes with a Titan 12T Waterproof DC Motor, up to 8.4 V, a XL-5 Electronic Speed Controller (ESC), a Steering Servo, a RC Receiver and its remote controller. Besides the components that come mounted on the car, a Lipo battery is necessary to power up the car, in this build case a 2-cell 5800 maH 7.4V Traxxas Lipo battery was used.

Figure 3.9: *Traxxas RC car model used for this testbed [11].*

To support every component in the vehicle, modifications were necessary to made to the structure. That being said, some acrylic sheets were precisely cut, with reference to the CAD files from [43]. All the mounting and wiring followed an approximation of the "Build Manual" from [23]. In this build, contrary to the F1 Tenth, the Orbity Carrier board, the Power board, the Energizer power bank and

the VESC were not used. Due to it, a few adaptations were made, like changing the height of the NVIDIA Development support board to fit a different power bank, as well as drilling new holes in it, to attach it to the new base created. In addition the wiring for the power supply was redone to fit the needs of this build.

4

System Architecture

The following section will describes the system architecture, explaining how every component is connected with each other and the benefit of implementing this robotic system.

4.1 Hardware

This robotic testbed is based on the Racecar model from the F1/10 competition and it follows the architecture represented in the block diagram in Figure 4.1. Every component is connected to the on board computer, the Nvidia Jetson TX2, which will process all computation necessary. The components are either connected to it via USB, in regard to the Teensy micro-controller and the IMU, or via Ethernet, in the case of the LiDAR. Since the developer kit only has one USB an extender was required. In the LiDAR case the connection is established through Ethernet.

To power every electrical component of the robotic testbed, a source of DC power supply was necessary and for that a power bank was used. The power bank connects directly to the Nvidia Jetson developer kit and the Hokuyo Lidar, outputting 12 V DC. However this power bank does not supply energy to the cars motor, as for that a specific Lipo battery was used.

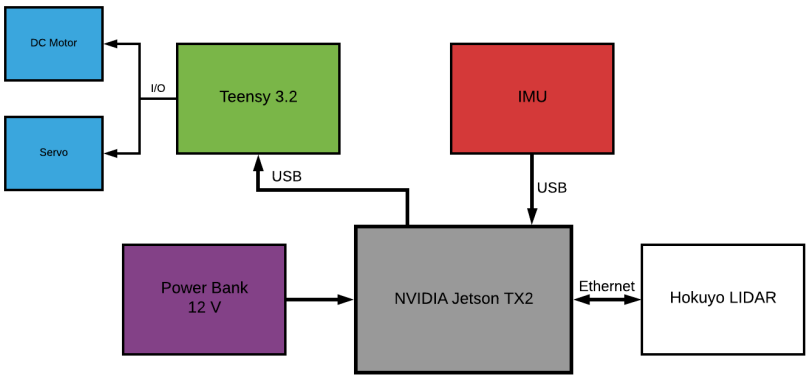


Figure 4.1: *System Architecture: Nvidia Jetson is the main computer and processes every sensor input that come from the Lidar and IMU and computes into actions in the robot platform.*

4.2 Software

In the same manner as the Hardware architecture, a software architecture will be also described. and follows the scheme presented in Figure 4.2.

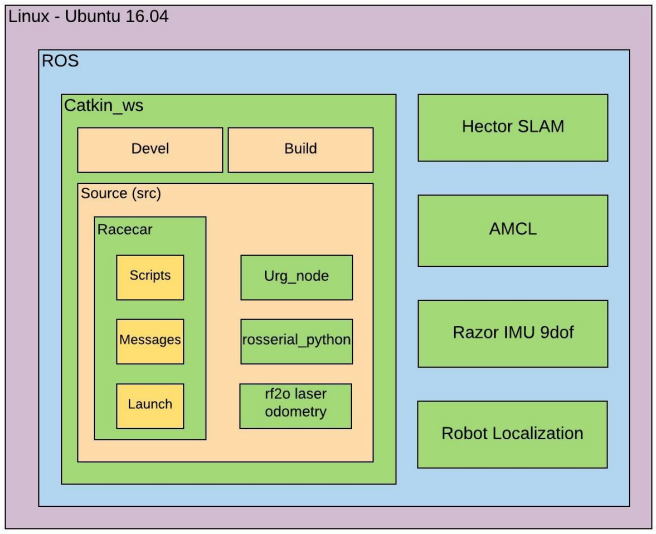


Figure 4.2: *System software architecture.*

Being the NVIDIA Jetson TX2 our on-board computer it requires a running operative system. As commonly used in robotic applications, the operative system running on the TX2 is Linux Ubuntu 16.04. Within this computer, the Robot

Operative System is installed and provides a series of packages and it includes the workspace, Catkin_ws, to develop and build our system. ROS provides various packages that enable a ready to use software, mostly relying on parameter configuration. In this implementation, the packages that were used are the Razor IMU 9dof, which allows to acquire data from the Sharp Razor IMU, the Hector SLAM that enables a static map generation. Also, the packages Robot Localization, that will provide sensor data fusion to then be applied in the Adaptive Monte-Carlo Localization (AMCL) package, where it makes possible to localize the robot in a static map.

In the Catkin_ws consist in the 3 directories, Build, Devel, Source and in this last one, is where the designed scripts will be saved, as it can be seen in Figure 4.3. It also includes the message folder, where it saves the messages types passed through topics and a launch folder, which allows to create custom files to launch different nodes and configuration files all in simultaneously. The scripts folder holds the algorithms created in python, that were implemented in the robot, for example the Talker.py script, that converts data arrival of angle and speed to Pulse Width Modulation, PWM, signals. In addition, in the source directory, it is found other packages that were locally installed. These packages were the rosserial_python, Urg_node and the rf2o_laser_odometry. The first acts as a bridge from the Jetson TX2 to the Teensy, to enable message passing. The Urg_node is responsible for reading information from the Lidar sensor and publish. To acquire odometry data from the Lidar to use in sensor fusion, it is used the rf2o_laser_odometry package.

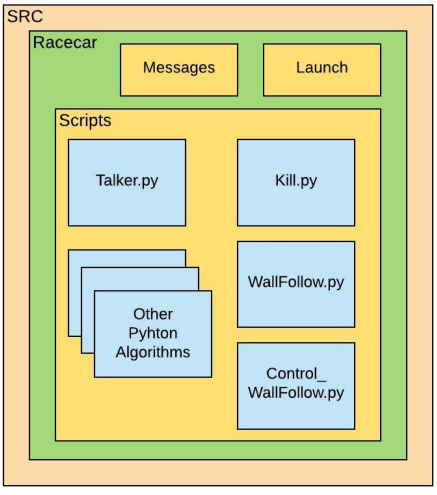


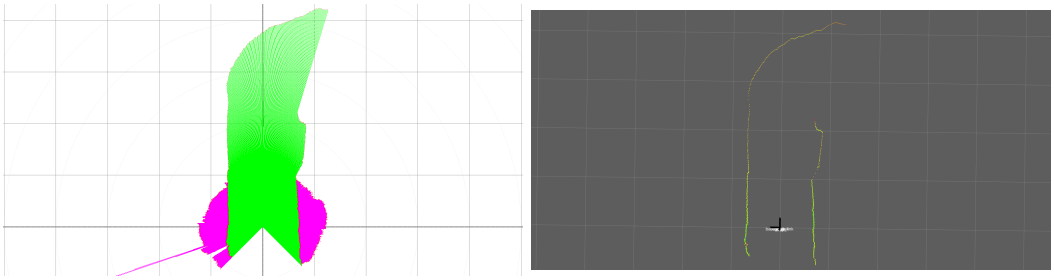
Figure 4.3: *Close up to the source directory of the workspace, catkin_ws.*

4.3 Lidar Scanning

The Lidar will provide data to the Nvidia Jetson with 2D scanning of the environment, covering a range of 270 degrees. This data will arrive to the Jetson at a frequency of 40 Hz, through Ethernet communication. Since this sensor is vastly used in robotic applications, ROS already possesses a package that supports this LIDAR model.

The setup is straight forward since another program, UrgBenri, can be used to configure the IP address of the LIDAR and very if it is working as intended. Then in our ROS work package and with the specific package installed the LIDAR was ready to use. With a Master running, it was only necessary to run the node with its IP address and the topic /scan was being published, returning measurements from the LIDAR. Using Rviz and subscribing to /scan topic, an identical visualization from UrgBenri could be seen and confirming the Lidar was working like intended. Figure 4.4 displays the scan from the the LIDAR in both UrgBenri and Rviz.

With everything functioning as expected, the values could now be used in the control algorithms that will be discussed in the next chapter.



(a) *UrgBenri* **(b)** *Rviz*

Figure 4.4: *Verification of the correct configuration of the Hokuyo Lidar using UrgBenri in (a) and operation in a ROS environment using Rviz in (b) .*

4.4 IMU interface

In [44], provides a tutorial on how to set-up this board in order to fulfill the project needs, additionally in [45] it is presented a tutorial that enables this board to publish messages in a format that ROS is capable of reading. This messages require that the data from the IMU come in a Attitude and Heading Reference System (AHRS) and to enable it a new firmware was uploaded to the Razor board. During the process of setting up the car, it is required to calibrate the sensor, since every sensor is different from each other and suffers from magnetic interferences.

With the ROS package "Razor_9dof_IMU" installed, the configuration file was updated with the calibration parameters. This package, apart from providing a node to read the IMU data publishing it in a topic, offers a visualization tool, helpful for debugging as we can observe its parameters altering. Figure 4.5 showcases the IMU working along with its visual tool, where it shows parameters like yaw, pitch, roll, linear acceleration and angular velocity.

In addition, in Figure 4.6 it can be seen the IMU topic that shows the different values being passed in a ROS sensor type of message. In further chapters, this information will prove to useful when trying to provide the robotic platform odometry information for localization.

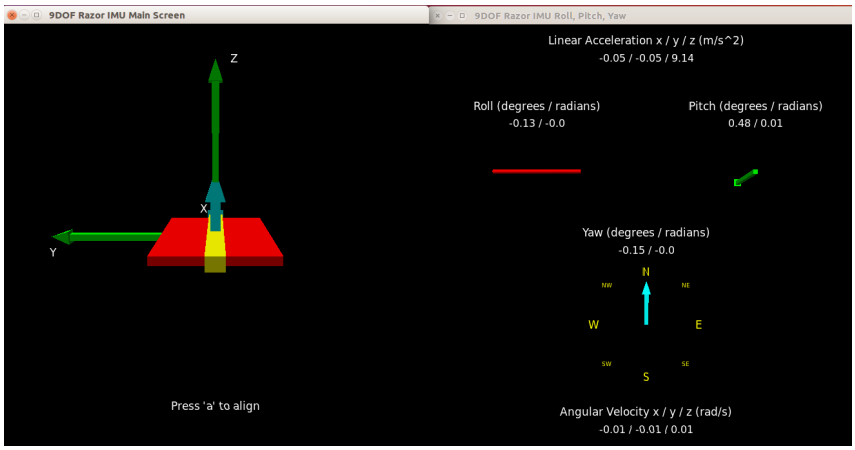


Figure 4.5: *IMU visual tool*

```
---
header:
  seq: 2425
  stamp:
    secs: 1564715705
    nsecs: 374305009
  frame_id: "base_imu_link"
orientation:
  x: 0.00493627358227
  y: -0.000132029033719
  z: -0.70833979688
  w: 0.70585433903
orientation_covariance: [0.0025, 0.0, 0.0, 0.0, 0.0025, 0.0, 0.0, 0.0, 0.0025]
angular_velocity:
  x: -0.01
  y: -0.01
  z: 0.01
angular_velocity_covariance: [0.02, 0.0, 0.0, 0.0, 0.02, 0.0, 0.0, 0.0, 0.02]
linear_acceleration:
  x: -0.09576171875
  y: 0.07124671875
  z: 9.16784390625
linear_acceleration_covariance: [0.04, 0.0, 0.0, 0.0, 0.04, 0.0, 0.0, 0.0, 0.04]
---
```

Figure 4.6: *IMU Messages*

4.5 Teensy Micro-Controller

Since the on-board computer, the NVIDIA Jetson TX2, and the RC car lack in a direct connection, an interface was required to be established. For that, a micro-controller was used, that could receive steering and speed controls from the Jetson TX2 and treat the values to insert in the servo and speed controller from the Traxxas car. Thus, the requirement of the Teensy 3.2 micro-controller.

With this micro-controller a bridge between the Nvidia Jetson and the electronic speed controller (ESC) can be established. In this build, the Teensy will be sending Pulse Width Modulation, PWM, signals to the motor and the servo, depending on the information received from the Jetson TX2. This implementation is possible since the Arduino software, which the Teensy is based on, has libraries compatible with ROS.

In addiction, this device will be connected with two switches to the car so we can decide whether we want control the car manually or autonomously. For that a custom Printed Circuit Board (PCB) was developed, like the one in Figure 4.7, which allowed to connect all the components. The board is composed by the Teensy micro-controller, two switches and pins that are associated to the DC motor, servo and the radio controller for the manual control.

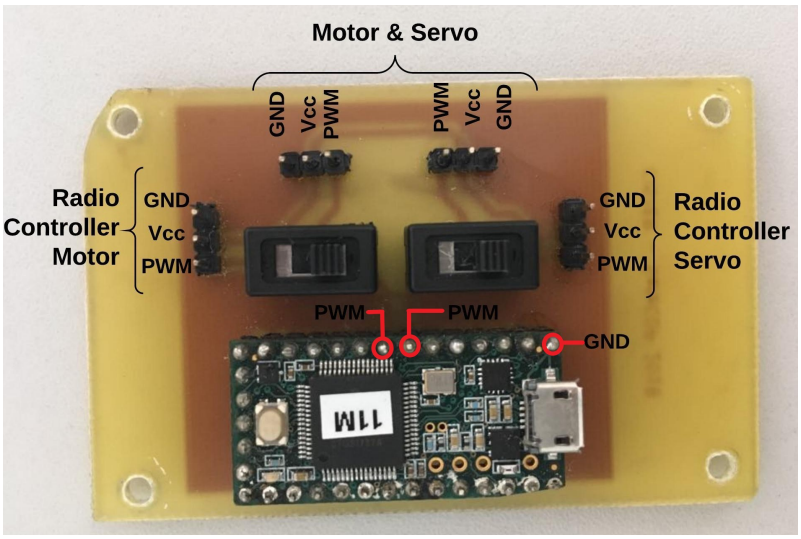


Figure 4.7: *This figure presents the custom PCB developed. The switches are respectively connected to the DC Motor and Servo.*

In order to send the information from the Jetson TX2 to the Teensy using ROS, it was used the *rosserial_python* package [46]. This python implementation automat-

ically handles the setup, publishing and subscribing for a connected rosserial-enabled device. As an example of this build, a Python script, *talker.py*, converts the received messages of steering and speed, from a control script, into valid PWM signals and publishes the topic */drive_pwm*. The conversion, relies on receiving speed values ranging from -25 to 25, where value 0 the car is stopped, and steering values with an extent from -30 to 30 degrees, which are respectively the minimum and maximum steering angle that the car possesses.

The *rosserial_python* will bridge this topic to the Teensy program, that will subscribe to it, through serial communication. The Teensy will then proceed to apply the PWM signals to the ESC.

In addition as a safety feature, an emergency script is also running, *kill.py*, that upon pressing the delete key, completely stops the car, ignoring other messages information. In Figure 4.8 it is displayed a flowchart of the 3 algorithms described.

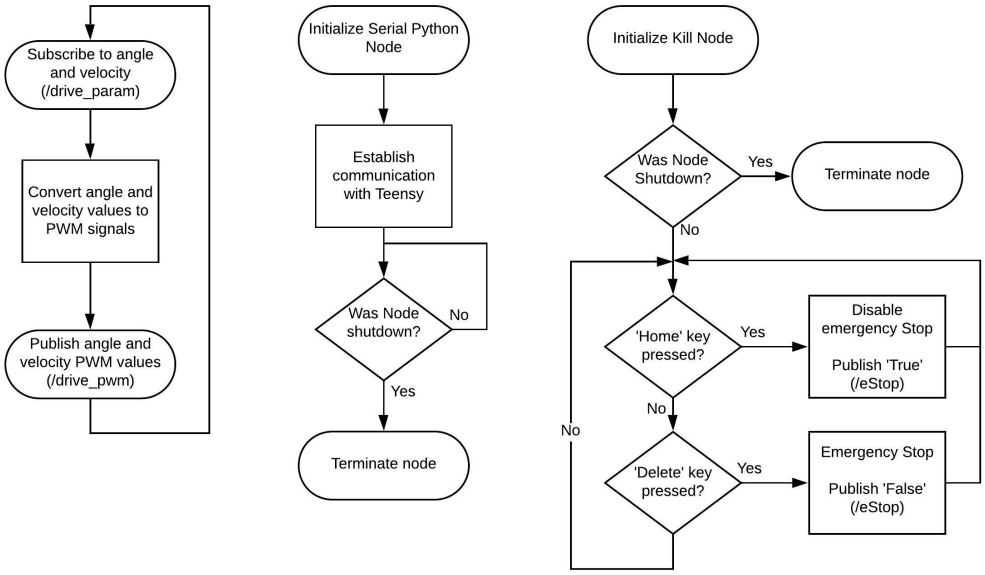


Figure 4.8: *Flowcharts of the pyhton algorithms: left - "talker.py", centre - "rosserial_pyhton node", right - "kill.py".*

5 Implementation

As previously presented in Chapter 2, an autonomous vehicle system is represented by 3 main core components, perception, planning and control. In this chapter we will present the methods used on this project to complete these 3 characteristics. We will start by demonstrating the implementations made in terms of mapping and localization, explaining their functionality. Afterwards, path and control phase are connected, explaining the different algorithms applied for the car to drive autonomously.

5.1 Mapping and Localization

One of the key aspects of autonomous vehicles is their ability to localize themselves with precision in a known environment. To accomplish it, sensor data must be acquired to perceive the car surroundings. Many approaches can be made depending on the system and sensors used. The following topics present the methods used to generate a map of a track and to localize the robot in the generated map.

5.1.1 SLAM

The Simultaneous Localization and Mapping (SLAM) algorithm provides, as the name suggests, a method for localizing the robot while generating a map of the environment. From the input perspective, it is required that they are given some

source of data, that generally come from sensors, such as LiDARs or cameras and can be complemented with other sensors, such as an IMU. SLAM derives from the problem of a robot not having a map of the environment and the necessity to obtain one. In other words, it is considered a "chicken and egg" problem, where it is necessary a map of the environment to determine the robot location, however at the same time the initial position of the robot is required to build the map. SLAM map generation is based on occupancy grids and scan matching.

Occupancy grids [19] can be defined as 2D maps or 3D worlds and in this project case, since we are working on a planar environment, the final product will be 2D map. This method is characterized for estimating the static objects in a given situation. Based on open spaces or blocked or obstacles, the occupancy grip problem turns to a binary problem. If an obstacle is found the it will be set to 1, otherwise 0 in free spaces. With this we can represent a map of an environment as an evenly spaced field of binary variables. Scan matching allow the system to adjust pose estimation between two consecutive time stamps, as it aligns the second scan with the previous one. With this, it is possible to maintain the robot pose and it updates the map simultaneously. Being the scan from the Lidar point clouds, scan matching uses the iterative closest point method, to find the transformation between the consecutive point clouds.

ROS provides different packages that facilitates the implementation of SLAM algorithms, where 3 of the most commonly used are Google Cartographer, Gmapping and Hector SLAM. Under the scope of this project, the package used was the $HectorSLAM$. The decision to adopt this package was due to the fact that it almost solely localizes the robot using its scan matching and in some situations an IMU when taking in consideration pitch and roll motions. In the remaining two, it is required a precise odometry, for example odometry provided from wheel encoders, something not implemented in this testbed, as the only source of odometry come from the IMU and Laser scan matching algorithms. Due to those restrains, Hector SLAM was chosen to build the map of the race track.

Hector SLAM [47], aims to deliver a fast online learning of occupancy grids, while requiring low computational resources, and sensor input, only relying on Lidar scans and optionally it can be combined with attitude motion from an IMU. In this SLAM approach, the authors apply a scan matching that optimizes with beam points, with the map learnt so far, thus not eliminating exhaustive pose searching or data association between endpoints. In addition, it is referred that an interpolation scheme is used, to exclude the limit in the precision of the occupancy grid map and as consequence, allow a sub-grid cell accuracy, that can be view as samples and with

that provide a continuous probability distribution.

The usage of the open source Hector SLAM package in ROS comes with other features that are described in [48]. It requires the user to provide data from the Lidar, the transformations, tf, between coordinate frames and alter the parameters values that fulfill its requirements. Figure 5.1 illustrates the transformation frame (5.1b), along with its running nodes and topics (5.1a). Figure 5.2 demonstrates the visualization of a map creation using the Rviz tool. In conjunction it is possible to observe the movement of the car, showing a correct behaviour of the localization.

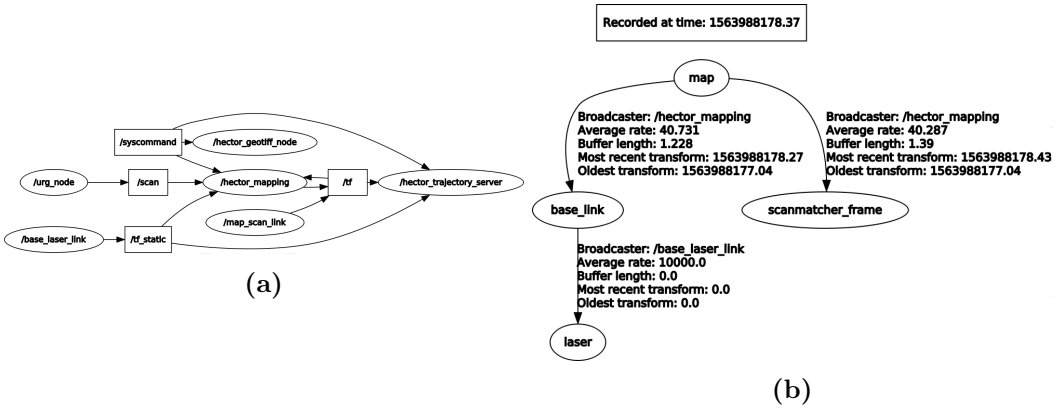


Figure 5.1: *Display of running transform frames (a) and nodes and topics (b) of Hector SLAM, using the rqt tool.*

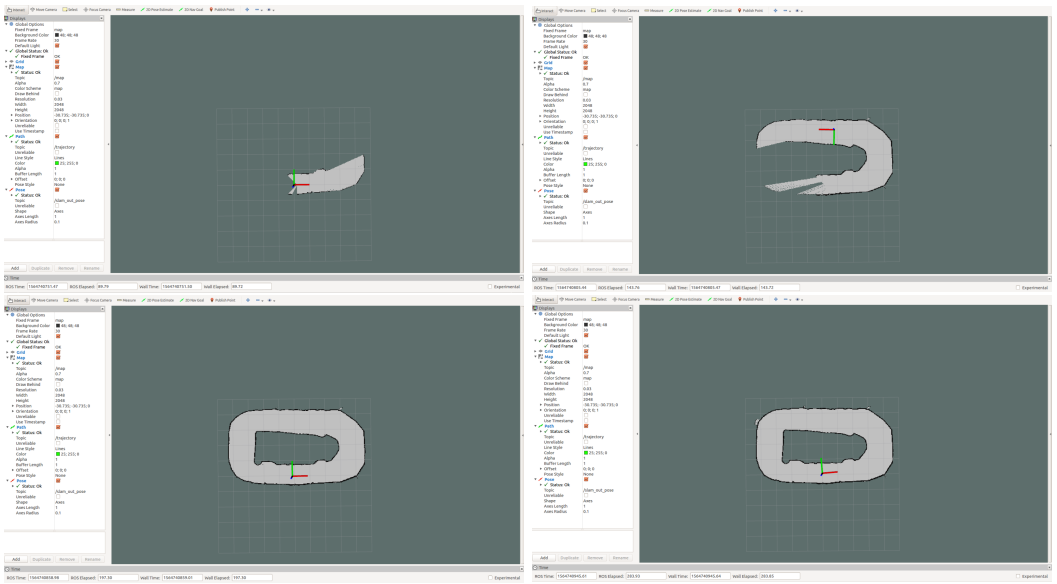


Figure 5.2: *Process of Map generation using Hector SLAM.*

5.1.2 Odometry Fusion

In many unmanned vehicles the necessity to obtain information about its position, heading, attitude, velocity and other parameters is crucial in any kind of navigation. To acquire this type of information, robotic applications require sensors, such as IMU, Wheel Encoders, GPS, Scan matching. A change in this sensors values, can provide an update of the pose of a vehicle, thus providing odometry data.

Even though these sensors can provide information about their relative position, it is frequent to observe oscillations and errors due to their precision, which gradually increases when dealt with low-cost sensors. To counter this measurements, sensor fusion was introduced and is present in numerous robotic applications, to reduce errors from various inputs. In [49], the authors expose several implementations of different types of odometry and their virtues and flaws. Generally, this fusion are based on discrete implementations of the Kalman Filter, such as the Extended Kalman Filter, EKF, the Unscented Kalman Filter, UKF, and many other as described in [49] an in [50], that although applied in vision applications can be used with other inputs.

Kalman filter is an algorithm, that upon receiving measurements, produces an optimal estimation for linear discrete-time state-space models. However, a real system will never produce a linear system, thus the necessity to implement solutions that could estimate non-linear models. With that said, solutions rely on filters like the EKF and UKF. EKF is the non-linear version of the Kalman filter and executes a linearization at each time step, while the UKF applies, what's entitled as unscented transform, to pick a minimal set of sample points around the mean, so that the filters can avoid poor performance, when the state transition and observation models are highly nonlinear [50].

Depending on the application, some parameters from the sensors might need to be discarded, tanking in consideration the more important. With the purpose of achieving that, methods were developed to fuse sensor information, outputting the odometry data.

In this testbed only 2 sensors were used that could provide useful data to be fused, so that odometry information could be acquired, being them the IMU and the LiDAR. With the intention of fusing sensor data, ROS provides a variety of options to accomplish it. The "Robot Localization" package [51] offers approaches to fuse multiple sensor sources using Kalman filters, remarkably, the Extended Kalman filter, EKF, and the Unscented Kalman Filter, UKF. In this project, it was only focused the use of the Extended Kalman Filter.

Since the robot developed in this project does not have highly precise odo-

metry sensor, it was necessary to rely on the Lidar to offer a pose estimation with enough precision. For this purpose, the odometry arrived from the Lidar, was obtained through a laser scanning algorithm to convert its values to valuable data to be used on the EKF. To fulfil this objective, the ROS package used was the "RF2O_laser_odometry" [52], which provides an estimation of planar motion from consecutive range scans, while performing dense scan alignment based on the scan gradients.

To enable the EKF, it was necessary to configure its parameter file, in order to receive the odometry data from the RF2O and the IMU. Since this state estimation node expects data from multiple sources and given the lack of sensors used, it was extracted all the information from the sensors. The parameter file, accepts the data from each input in a vector, in a Boolean format, where each position represents the following variables: X,Y,Z,roll,pitch,yaw,$\dot{X}$,$\dot{Y}$,$\dot{Z}$,$\dot{roll}$,$\dot{pitch}$,$\dot{yaw}$,$\ddot{X}$,$\ddot{Y}$,$\ddot{Z}$.

In other words, if one decides to only use the x,y position variables from a sensor to feed the EKF node, then the first two positions of the vector are set as true, while the remaining others are set as false. Taking into consideration the amount of data that the laser scan odometry and the IMU provide, their correspondent vector in the EKF configuration file can be seen below.

In addition, since this testbed only operates in a planar environment, this package provides an option,"two_d_mode",(2D mode), that ignores 3D variables, improving the odometry estimation. The "odom0_config" represents the matrix for the RF2O odometry data and the "imu0_config" the matrix from the Razor IMU. The first will accept the x and y position and the yaw rotation values, along with the linear velocity in the X-axis and angular velocity around the Z-axis. The second one, although giving more information, we only required the data from yaw rotational values, the velocity and acceleration around it, since the 2D mode is activated.

```
odom0_config:[true, true, false,
              false, false, true,
              true, false, false,
              false, false, true,
              false, false, false]

imu0_config: [false, false, false,
              false,  false,  true,
              false, false, false,
              false, false, true,
              false,  false,  true]
```

Provided all the configurations necessary, in both laser scan odometry, IMU and EKF, it was possible to achieve filtered odometry message to subsequently be used to feed the odometry data in a localization system, that will be presented immediately.

5.1.3 Adaptive Monte-Carlo Localization

Many pose estimation algorithms, can be applied in order to localize a robot in a known environment/map. In this project, it was applied the same method as in the F1 Tenth build. The Adaptive Monte-Carlo Localization, AMCL, [53] is a probabilistic localization system for robots moving in 2D. The system is part of ROS navigation stack and it functions by tracking position of the robot in a known map, using a particle filter, while following implementations from [19].

This method is an improved version of the Monte Carlo Localization, MCL [54]. The original format relies on fast sampling technique to represent the belief of the position. After movement, re-sampling is done to acquire new position. Being the goal to recursively compute at each time step the set of samples from a density probability, this method relies on particle filters.

Particle filters [19], are based on Bayes filters, and use a number of finite samples to approximate a posterior distribution of given parameters, called particles. Taking into consideration a large number of particles, the general idea is to approximate a belief, by a set of particles. Depending on the density of a region populated by samples, the likely it is for a true statement to be correct.

The MCL algorithm proceeds in two phases the prediction phase and update phase, as described in [54]. The first starts with a uniform random distribution of particles and at this given point the robot does not have any input on where it is located. In the second phase, the robot has already moved, implying a shift of the particles pose prediction, giving a new state. If something new is sensed by the robot, the particles are re-sampled. Throughout each step, the particles will converge to the respective position of the robot.

Following the same base line, the AMCL applies the same methodology [55], as it ancestor, however adjustments to the sampling are made. Though the MCL presented efficient results, it suffered from a large computational processing as the same number of samples were always taken into consideration. Such large values are required in the first phase, as the estimation is still being gathered and the convergence made to the actual position of the car. Once the particles converged, having as consequence a much smaller cluster, the number of samples is maintained, introducing unnecessary error the pose estimation.

AMCL applies a method to cover this issue, adapting the number of samples required. Its chooses a small number of samples, if the density is focused on a small subspace of the cluster and chooses a large number of samples, if the samples have to cover a major part of it. This way both the implementation and computational overhead of this approach were reduced, improving overall results [55].

As of other mechanisms for robot localization in a known map, [19] presents other methods such as Markov localization, while [56] demonstrates a fast particle filter approach. Although, different implementations towards robot localization could be made, it was decided to follow the AMCL algorithm, since all the resources to implement this solution were ready to use, enabling fast and efficient set up to acquire localization for future results.

As stated, AMCL requires a map to compute its localization, which in this case it will use the map generated with resource to Hector Slam. Coupled with the map, the system requires input from laser scans that will try to match with it, which arrive from the Lidar, as well as odometry and transform messages, with aim to output an estimation of the robot position and orientation. This implementation is important in this project, since the odometry acquired is not sufficient to localize the robot in a map.

Regarding frame transformations, the localization system follows the required conventions [57] that, consequently, upon its implementation produces a tree of coordinate frames in a similar aspect to Figure 5.3. The *map* and *odom* frames are world-fixed frames, which diverse from each other on their continuity. Whereas the first one produce a long-term global reference, which presupposes a discrete jumps overtime it eliminates drift, the other one, proceeds to offer accurate short-term local reference. The *base_link* frame describes the rigid robot base, that supports the all hardware, like the Lidar and IMU, and provides a point of reference to the robots base.

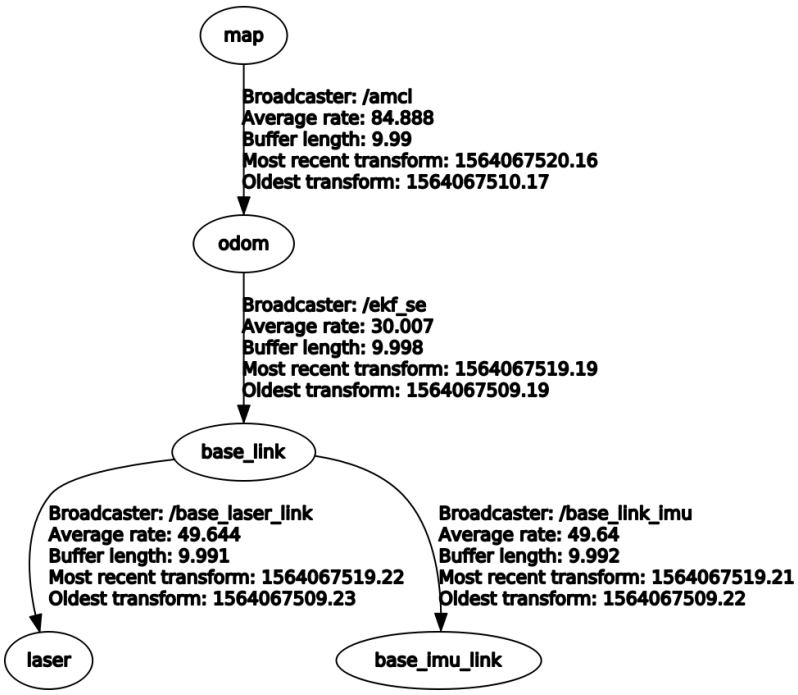


Figure 5.3: *AMCL Tree Frame, acquired using the ROS rqt tool.*

With all the configurations completed, the results of the Adaptive Monte-Carlo Localization will be presented in Chapter 6.

5.2 Control Methods

Provided data from the sensors, different algorithms can be implemented with the objective to control the car autonomously. The following introduce the robust control methods put into practice in this testbed, in order to later be compared between each other. These algorithms rely mostly on Proportional Integral Derivative, PID, controllers to, except for the last two explained, to attain a steering control.

In terms of speed control, it was decided to implement the same process in all algorithms. This was due to the difficulty to arrange a robust yet effective technique to adjust the speed along its path. Henceforth, this method below in Equation 5.1, was employed to manage the speed that is passed through PWM to the Teensy.

$$speed = min\_speed + \frac{dist}{max\_dist - min\_dist} \times (max\_speed - min\_speed) \quad (5.1)$$

As it can be seen, the equation above requires constant parameters, that are

given, respectively, by the maximum and minimum distance and speed. This parameters can be altered, in order to restrict the speed to be delivered as a PWM signal, in particular the maximum speed to be applied. The remaining parameter, *dist*, is the current distance being read by the Lidar at 90 degrees, i.e., the distance right in front of the car.

With this approach, it was possible to smoothly adjust the speed that the car was reaching, allowing to understand the limitations, in terms of speed, of each algorithm that will be implemented and thus achieving a control with steadier stability.

5.2.1 Wall Follow

This algorithm describes a simple method as a first approach to autonomous driving in a robotic testbed. Figure 5.4 demonstrates the flowcharts of this method.

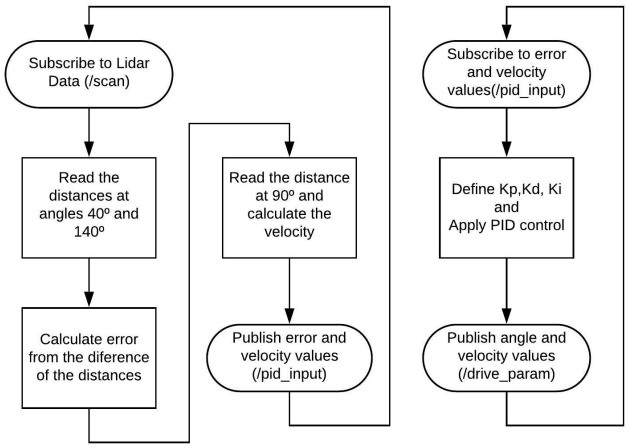


Figure 5.4: *Flowchart of the wall follow algorithm implemented. On the left is displayed the data processing algorithm and on the right the control algorithm.*

By subscribing to the scan topic, published by the node referent to the Lidar, urg_node, it is possible to obtain the distances at specific angle.

Taking it that in consideration, the distances at 40 and 140 degrees are acquired, which are both a distance to the right and left side. These angles can be adjusted, however these were the ones which gave best results.

With these distances, an error was obtained by the difference between the left and the right measure. This error, that is published to another topic, will indicate how far from the center of the track the car is and will be used as weight to control the steering angle of the servo motor.

Upon receiving the error variable, it was possible to implement a Proportional, Integral and Derivative, PID, controller, since it will automatically apply the necessary corrections to the steering angle. The PID controller runs in a separate node and subscribes to the topic that receives the error value. The applied control action is given by a PID presented as

$$Control\_error(t) = K_p * e(t) + K_I * \int e(t) + K_D * \frac{\Delta e(t)}{dt} \tag{5.2}$$

where K_p, K_I and K_D denote respectively the Proportional, Integrator and Derivative gain constants, $error(t)$ is the measured error and $Control\_error(t)$ is the output control of the system. The output is then added to the previous angle applied, thus resulting in the angle to be applied in the steering, that will publish to a new topic to be subscribed by the talker node, referenced in the prior chapter.

5.2.2 F1/10 PID

Considering the scan data from the LiDAR, the F1 Tenth crew developed an algorithm [12] that enables the car to move parallel to walls at a fixed distance while taking in consideration the orientation of the robot obtained from the distances at specific angles.

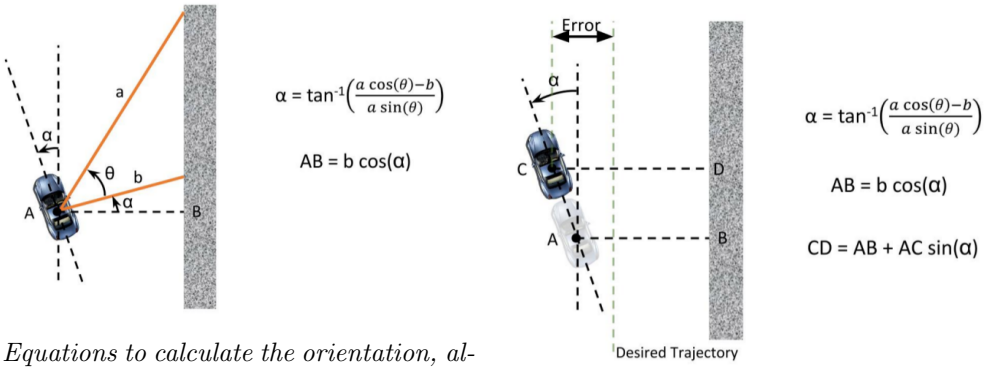
Firstly, the algorithm starts by assuming the LiDAR reads the angles from 0 to 180 degrees, being 90° the front of the car. Since the Hokuyo Lidar used in this project covers 270 degrees, it was sufficient to meet the needs of this algorithm.

Secondly, two distances are read at 0 degrees and theta degrees, where theta is an angle in between [0-70]°.

Thirdly, defining an angle alpha as the orientation of the car, it can be obtained following the geometric problem in Figure 5.5a and subsequently the distance of the car to the wall at that orientation.

Fourthly, since the robot will be performing at high speeds, at the moment of the data acquisition until actuating upon the car, the control values applied would not represent the actual controls necessary to apply due to that delay. To overcome this problem, it is added a distance to forward the car from its actual position and from that the final distance to the wall is achieved.

Lastly, an error is obtained from the difference of the desired distance to wall and the actual distance calculated. Figure 5.5b demonstrates the equations to achieve the distance from the wall.



(a) *Equations to calculate the orientation, alpha, where 'a' represents the distance read by Lidar at theta degrees, 'b' the distance at angle 0, 'AB' the current distance of the car to the wall.*

(b) *Addition of a distance compensation due to a delay in processing, AC, and recalculation of the new distance to the wall.*

Figure 5.5: *Achieving an error from a desired distance and the actual distance of the car [12].*

This procedure is applied when following the right wall of a track, however by inverting the angles to the opposite side, it is possible to obtain the error referenced to the left wall.

By obtaining an error from the previous equations, this parameter can be used in a standard Proportional, Integral, Derivative (PID) controller, in the same manner as previously described. However, in this algorithm only a Proportional Derivative, PD, controller is used, as the integral variable does not contribute to the systems stability. To deliver a better perspective of the algorithm, in Figure 5.6 it is demonstrated a flowchart of this algorithm. Regarding the control algorithm, it follows the same principle as previously shown.

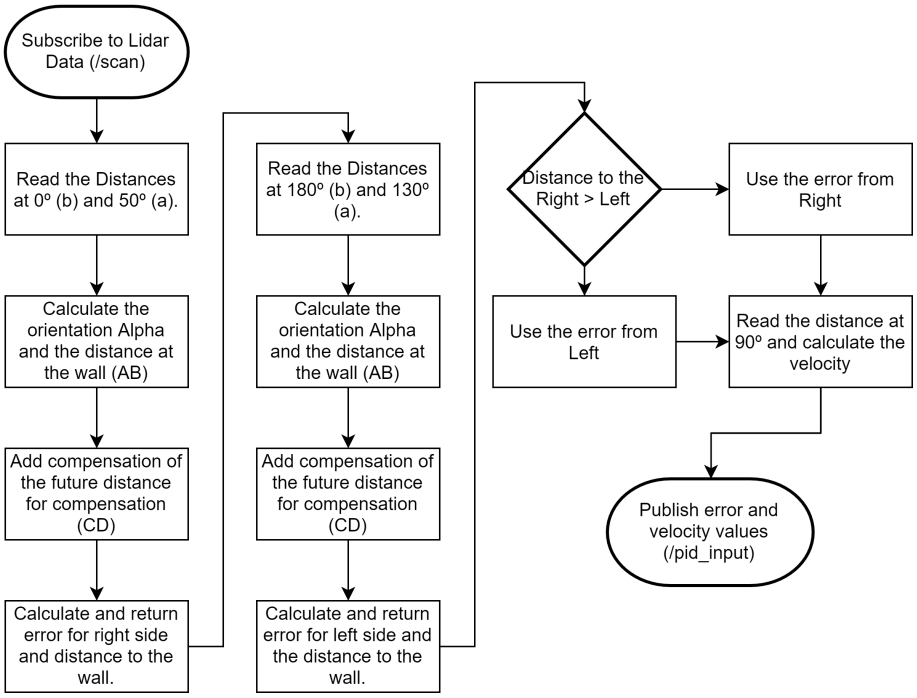


Figure 5.6: *Flowchart of an example algorithm from F1/10 implemented.*

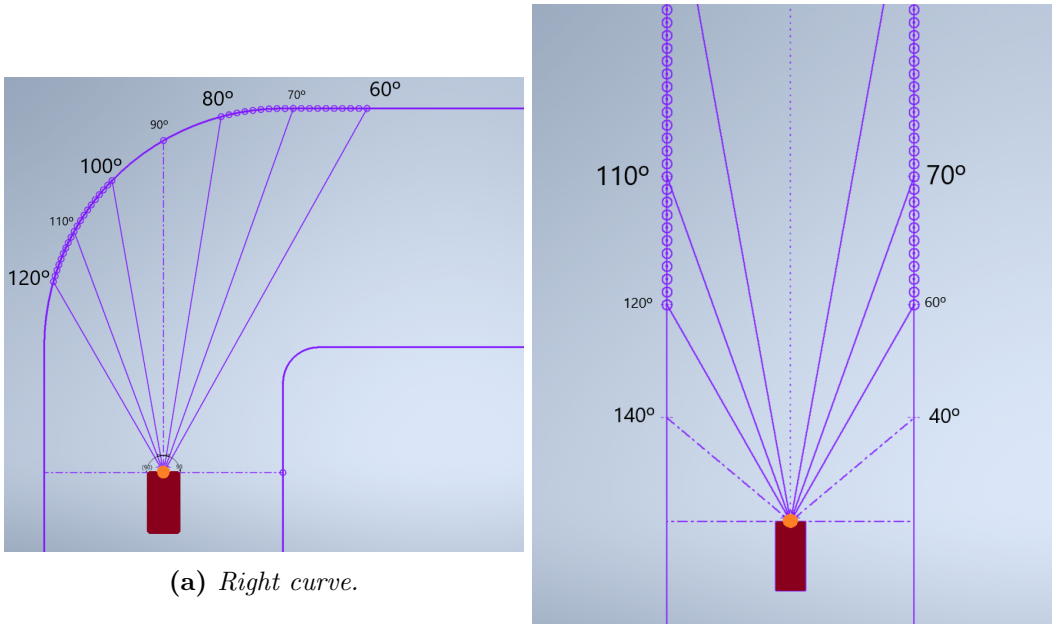
5.3 Curve or Straight line detection

When the car faces a curve, depending on the direction of it, for instance we will use a right curve situation, the distances readings from the Lidar, in clockwise, will gradually increase, since the more to the left points will be closer. Same applies in the opposite direction. In a straight and cleared line in a track, the center readings, around 90 degrees, will return much greater distances, that the ones from more lateral readings, around the 45 degrees.

Under these circumstances, founded on the PID implementation from the F1/10, a different approach was made, where a straight line or a curve in both directions could be detected, relying on the Lidar readings. To enable a change of situation encountered, specific intervals from the Lidar range were taken in consideration. After tuning these intervals, the following set of ranges proved to be ones that gave the best results.

- Angle range for straight line detection: 70° - 110°
- Angle range for Right curve detection: 60° - 80°
- Angle range for Left curve detection: 100° - 120°

As it can be seen below, both situations where the car encounters a right curve, Figure 5.7a, and a straight line, Figure 5.7b, are illustrated. It is clear that in 5.7a, the 20 points read, from 100° to 120°, are at closer distance that the ones from 60° to 80°. In 5.7b the points from 70° to 110°, that cover most of the center readings of the Lidar are at much greater distance, comparing to more lateral values. With these assumptions, we will be able to qualify when the car is present in one of the 3 possible situations.



(a) *Right curve.*

(b) *Straight Line.*

Figure 5.7: *Illustration of two different encounters in a race track. The red box represents the robotic racecar and the orange circle the Lidar, along with an exemplification of the laser beams at various angles reaching the walls of the track.*

Upon subscribing to the Lidar scan topic, the distances for each angle, in the defined interval, was compared to a fixed distance and if minor, a counter would increment, giving the number of points acquired in that range.

Subsequently, the number of points from each counter would be compared to a strict set of rules defined. In the straight line detection case, it is required for the number of points read in the front centered range, [70;110]°, to be less or equal to 5 and the number of points in both left and right ranges to be less than 15. If confirmed, this would mean the car could have an appropriate control to follow a straight line. Regarding the curve detection, depending on the direction, a number of ranges is obligatory to be greater than the other, while the number of points in the center readings, is required to be less than a large amount of values. Following the

example of the right curve previously illustrated, we confirm the number of points on the left is greater than the ones on the right, as well as the number of points on the left are greater than a certain value, while the center ranges are mostly covered. If none of the 3 situations is detected, then the previous state is kept.

When the algorithm received the value that would correspond to the encountered situation, it applies the same procedure of the following the right or left wall or both, but contrary to the side found. In other words, when a right curve was detected, it would follow the left curve, when a left curve was detected, it would follow right wall. When a straight line was detected, a simple $WallFollow$ approach was applied, with the angles for left and right being, respectively, 140° and 40°.

The process to clearly detect each situation is represented in follow chart below, Figure 5.8.

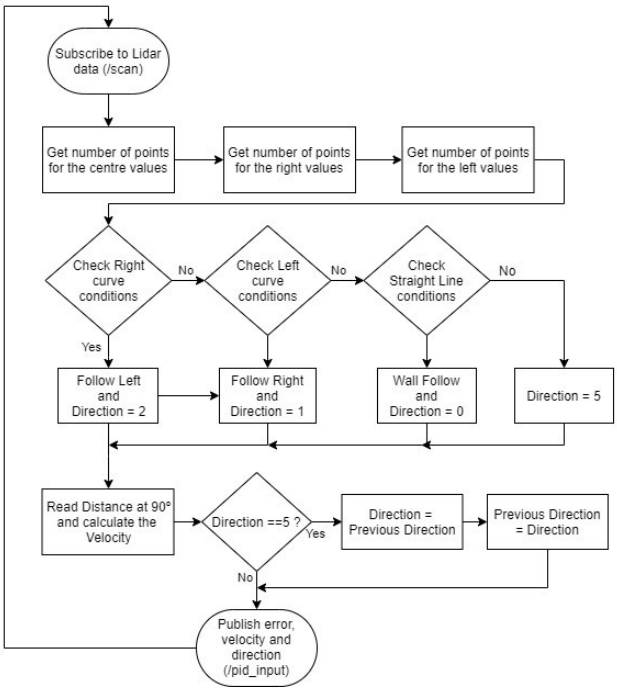


Figure 5.8: *Flowchart of the curve and line detection algorithm implemented.*

Following the same principles, the errors obtained are applied to the PID controller for the steering angle, however in this particular implementation, two different sets of constants were used. When detecting a straight line, it was only applied a Proportional controller, as the adjustments when tackling it should be as minimal as possible, in order to make the car go as straight as possible. In a curvature situation, a PID controller was applied as a more careful handling was necessary, to quickly

respond to variations. Figure 5.9 represents the flowchart of the control algorithm.

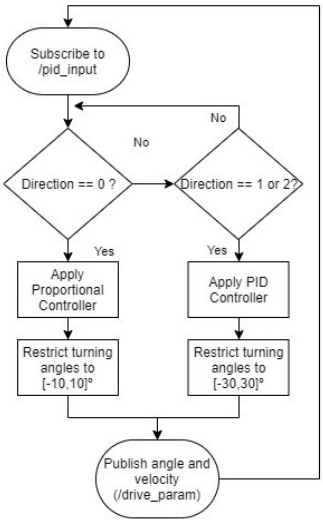


Figure 5.9: *Flowchart of the control algorithm for the curve and line detection implementation.*

5.4 Disparity Extender algorithm

As mentioned before in Chapter 2, the winning algorithm of the latest F1/10 competition was the developed by the UNC-Chapel Hill team, which introduced the "Disparity Extender" Algorithm. Since this was the winning team by a large margin and the authors provided an explanation on how the algorithm works, it was decided to replicate it as closely to the information given in [13] and put it to test.

As the Authors mentioned, this algorithm gives a more simpler and robust approach to autonomous racing, comparing to what the rest of the participants competed with. The foundation of this control is to find the longest distance that the car can safely reach while driving in a straight line, by checking the disparities provided from the Lidar readings. A disparity is defined by two subsequent points that differ from a large amount predefined. Figure 5.10 exemplifies a visual representation of a disparity.

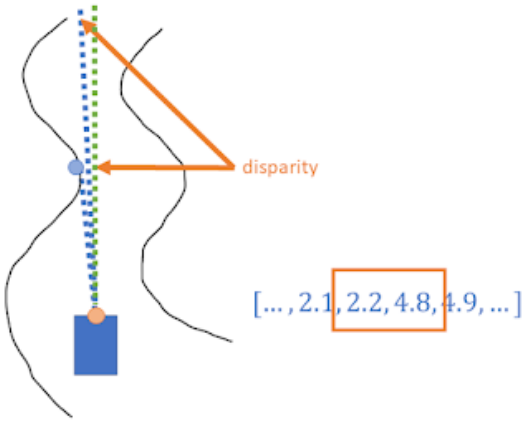


Figure 5.10: *Example of a disparity from Lidar readings [13].*

While continuously reading the distances from the Lidar, the point with the closer distance is picked and calculated the number of Lidar samples needed to cover half the width of the car and some tolerance. Next, starting from the furthest distance of the 2 points at a disparity, the number of samples in the array is overwritten with the closer distance. Continuing in the same direction, the process is repeated for every disparity until it reaches the end of the array or the number of samples calculated before is covered.

At this point, the new array only contains safely-reachable distances, like in Figure 5.11. Consequently, the sample with furthest distance from the array is used as the target for the car to drive to it and the angle necessary to apply is calculated. With this repeated process, the car can aim at the best goal possible while avoiding obstacles that may appear along the track, since they can be detectable by the disparities.

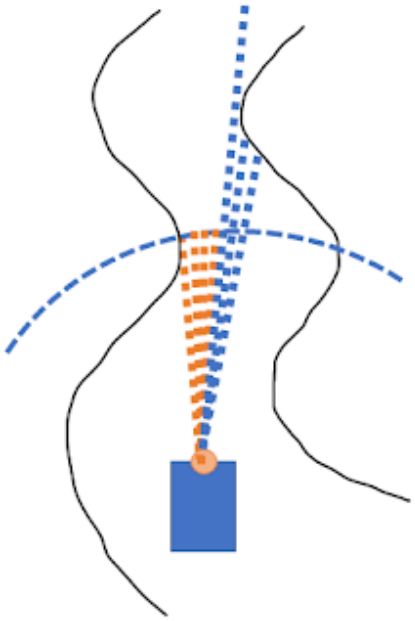


Figure 5.11: *Filtered array of distances [13].*

Concerning the speed to be applied, the team decided to only rely on the distance in front of the car. Depending on the distance read, if greater than the value established, then it applies the maximum speed and the reverse when it reaches the minimum safety distance. Between these range, the speed scales based on the distance read.

The algorithm also covers a possible problem when tackling corners. Figure 5.12 demonstrates that if no correction is applied, when facing that situation, the car would crash into the corner. To contour this error and since the Lidar readings cover the sides of the car to the back, due to the 270° field of view, a safe distance is included on the side of the car, to stop it from turning and keep going straight until it completes the curve with safety.

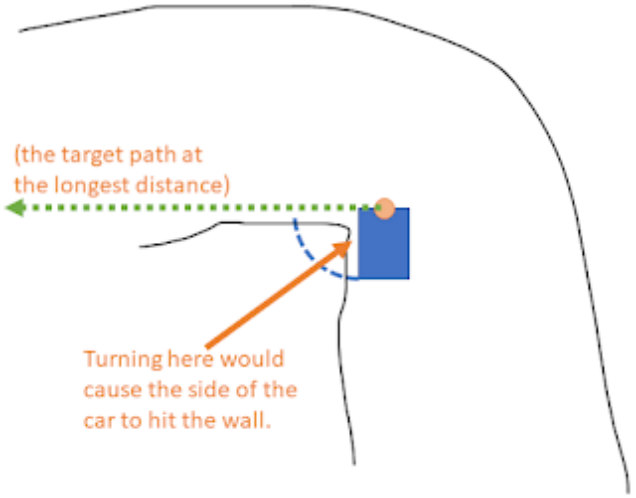


Figure 5.12: *Exemplification of a possible problem when tackling a corner [13].*

Up to this point the algorithm is assumed to never crashed even when facing obstacles along the track. However, the authors also raised an issue that would still occur under some situations. If it is assumed that the track does not uniform width, when approaching a corner in a wide portion of a track that will translate to a narrower portion, the car might reach a point where it will do an U-turn. Since the opposite corner will give a longer distance than the ones in the curve. The car will move towards it and depending on how fast it is driving, by the time it processes a new array of distances, the car might have moved far enough, never assuming the correct path to take and thus making an U-turn. In [13], this problem is explained in more detail. To produce a clear perspective of this implementation, in Figure 5.13 it is displayed an overview of the disparity algorithm in a flowchart pattern.

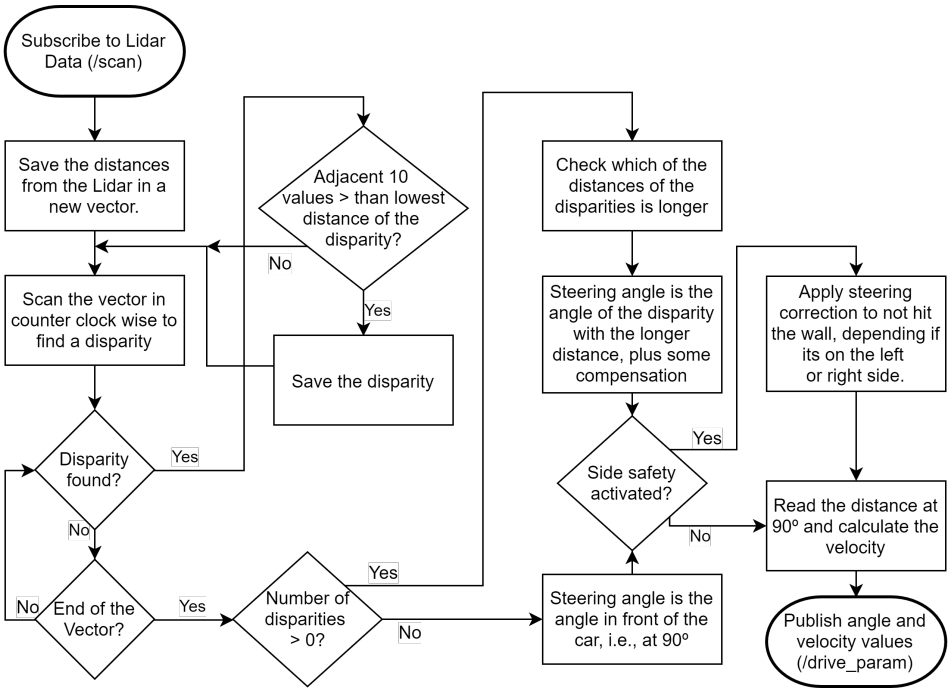


Figure 5.13: *Flowchart of the Disparity Extender algorithm implemented.*

To sum up, this algorithm addresses the control method in a simpler way, yet effective, comparing to the F1/10 approach. Without having the complex mathematical calculations it preforms well when racing in a time-trial or a head-to-head challenge. Nevertheless, as it will be seen in Chapter 6, the results could still be improved and adjusted in some aspects and thus the necessity reevaluate the disparity control. Due to it, the following is an upgraded version of this algorithm that emphasises on correcting the problems of it successor.

5.5 Upgraded Disparity

In the first place, upon testing the previous algorithm, it was evident that some corrections were required. One problem that was clear, was that the car made some steep adjustments when driving in a straight line, targeting the furthest point, until it found a disparity from the next curve or an obstacle. When it happened the car ended up applying to much steering.

Another issue encountered, was the number of angles that were being taken into consideration when collecting data for the disparities. If the robot stands in the beginning of a long hallway, the beams propagated from the Lidar will be more spread out, in comparison to the car being in the middle or at the end of it. In

other words, something that is closer to the Lidar will comprise more angles. That being said, the need cover the same number of angles along the track should deviate depending on far the disparities are situated.

The junction of this two matters, resulted in a slight oscillation in the path and as consequence a loss of time per lap. Under those circumstances, to improve the algorithm, we propose the following changes to be applied in order to overcome the issues mentioned above, while keeping the authors aim on the simplicity and robustness of the algorithm. In Figure 5.14 it is demonstrated the flowchart of this algorithm, highlighting the changes applied in comparison to previous one.

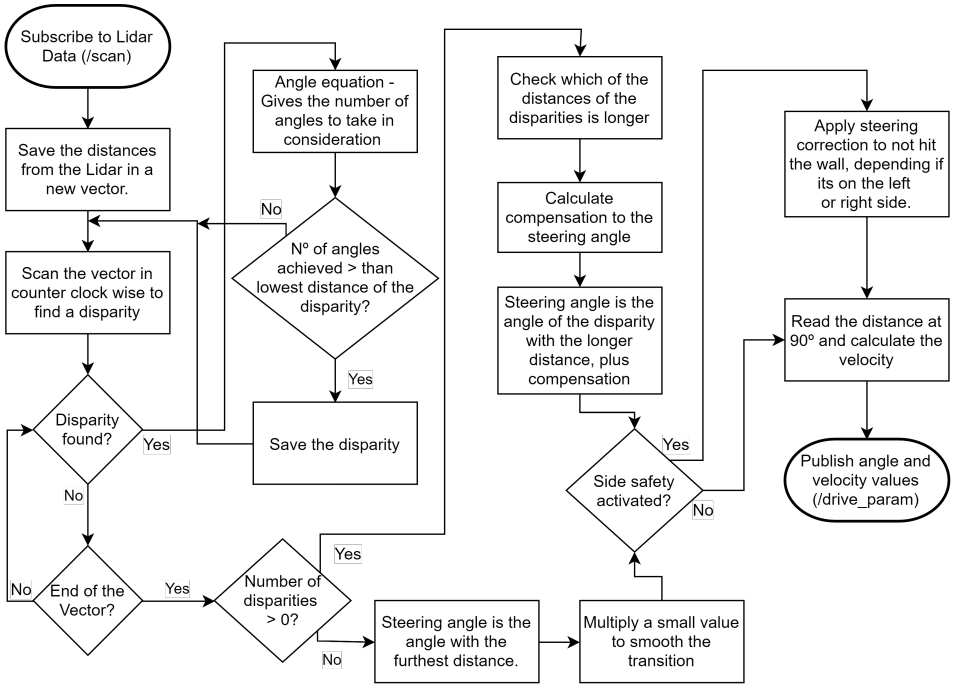


Figure 5.14: *Flowchart of the Disparity Upgraded algorithm implemented.*

Firstly, to regulate how many angles should be taken into account, it was used a linear regression, using the furthest distance of the disparity. Through trial and error, measuring the distances and checking the number of angles the Lidar would include, the optimal values reached were $0.75m$ and $5.25m$ and the angle count for both respectively were 28 and 10. As a result, Equation 5.3 was obtained through a linear regression and used to calculate how many angles would be necessary between these range of values. Anything above 28 or below 10, would be fixed at those values.

$$Angle\ Count = -4 \times distance + 31 \tag{5.3}$$

Secondly, pursuing the same approach as the previous one, a small compensation was added to prevent abrupt transitions in the steering applied. That being said, the linear regression in Equation 5.4 was achieved, based on the distance of the nearest disparity, equally through trial and error. The values were also restricted in the same manner, between 20 and 23.5 degrees. If no disparity found and the car aims at the furthest distance, then a small constant in the steering is multiplied. This constant has the objective to smooth out the values for the steering calculation, to move closer to zero.

$$Compensation = -2.8 \times distance + 26.2 \tag{5.4}$$

Granted the additional implementations, it will be possible to see in Chapter 6 the impact these have made, not only in terms of trajectory, but also in a timing performance.

6 Results

This chapter demonstrates the results achieved throughout the development of this work. First, it will be shown the results of the mapping and localization implementations in different scenarios. Next, to prove the effectiveness of the algorithms used, a comparison of the trajectories taken by each one will be demonstrated. Time will also be taken into measurement, comparing which algorithm proves to the fastest. Lastly, a summary of the results will be lifted.

6.1 Mapping and Localization Results

For the following results we will demonstrate the precision through a series of tests. For each localization algorithm, we will measure the deviation at the initial position and at 1 and 3 meters in front it. The measurements were acquired in the start the start of the algorithm, after completing 1 Lap and after 5 Laps and were repeated two times for each.

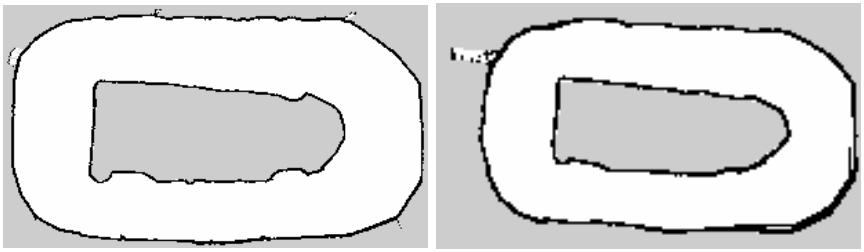
6.1.1 Mapping

As mentioned before, for the generation of the map for localizing the robot, it was used the hector mapping. To obtain the map, it was fixed in the track, what it was considered for all tests that will be realized the initial point, like it can be seen in Figure 6.1.

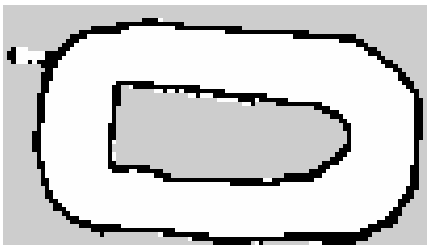
Figure 6.1: *Real initial starting point of the car on the track.*

The procedure to generate the map, consisted in manually driving the robot at a small speed around the track. The reason for acquiring the map this way, lies in the guaranteeing a correct map generation, since doing it at higher speeds or autonomously, would increase the possibilities of a bad generation, as the car would "get lost" and start giving a shifted map. Additionally, Rviz was used to debug and observe the map being created.

Different maps of the same track were generated, while changing the resolution of it, in their configuration file. The results from Figure 6.2 represent the maps generated in the oval track. As it can be seen distinct resolutions can greatly affect the map quality, as small details start to be more noticeable. However, this brought out a negative aspect, which can be noticed, being a small shift that would accumulate and mess up the borders for to be used in localization.



(a) *Resolution: 0.03* **(b)** *Resolution: 0.06*



(c) *Resolution: 0.1*

Figure 6.2: *Different map generations with Hector Slam while varying their resolution.*

Apart from the Oval track, another map was created to guarantee the correct behaviour of Hector Slam, more specifically in a wider and open space. Figure 6.3 demonstrates the map generated of the *Undergrad Lab Area* at CISTER. Overall the map was generated successfully, although few cells were badly generated, as it can be seen on the top of the map, since those locations are all windows. Due to it, the Lidar couldn't properly detect it as solid material, as the laser beams were reflected and thus such errors occur.

Figure 6.3: *Generated Map from Hector SLAM of the Undergrad Lab Area at CISTER with a 0.03 resolution.*

To demonstrate the precision of the localization aspect of Hector SLAM, we proceeded to calculate the absolute error, in meters, by comparing the values obtained from the SLAM and the actual distance measured. We started by registering the values provided from SLAM, at the initial point of the car. Then, we moved 1 meter forward from the starting point, in a straight line, an registered the values once again. Afterwards, the values were acquired, but 3 meters from the initial point. Finally, the procedure was repeated after making 1 lap around the track and 5 laps, in order to understand if error was accumulated. From our experiments, we achieved the following results.

Table 6.1: *Obtained errors with Hector SLAM, in the initial lap (Init) and after 1 and 5 laps. The errors were measured in the initial position, after 1 meter and 3 meters.*

SLAM Error			
NºLaps / Position	Init	1 Lap	5 Laps
Initial Point (0m)	0.002	0.005	0.03
1m from Intial Point	0.005	0.02	0.03
3m from Initial Point	0.01	0.005	0.07

Observing the obtained results, we can conclude that the hector slam was successfully implemented, providing precise pose estimation, with very accurate precision. However, important to notice, this values were only possible to attain while driving the robot manually, at a very slow speed. When tried to run at higher speeds, the map generation failed tremendously, leading to the algorithm to lose track of its position.

6.1.2 Odometry

Based on Chapter 5, the results of fusing odometry are represented below. In Figure 6.4, it can be observed, resorting on the Rviz tool, the update pose of the car using the fused odometry from the Lidar and IMU. The origin of the pink arrow in the image, signalizes the position of the car and the direction its pointing the heading of it. The four images in the figure below, were obtained by manually controlling the car, and represents the updates of the odometry while driving around the oval track.

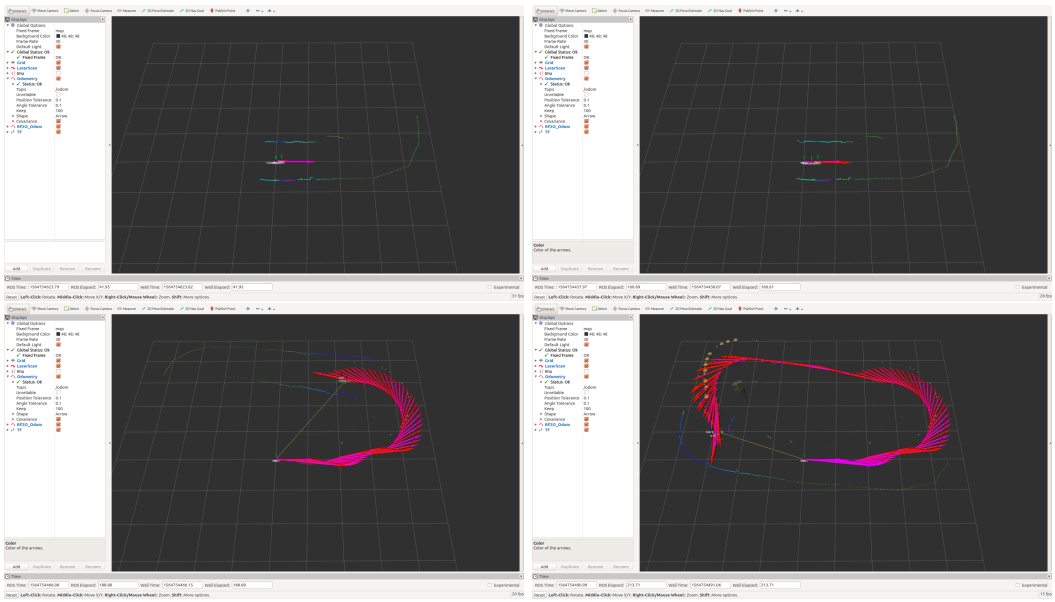


Figure 6.4: *Visualization of the pose update of the odometry, provided from the Extended Kalman Filter.*

Regarding the precision of the odometry achieved, Table 6.2 presents the absolute errors obtained, in the same scenarios used in the Hector SLAM tests. We could concluded that the odometry in the beginning provides a precise position of the car, however after 1 lap it can observed that this precision drops heavily and continues throughout it. This results are due to the fact that the sensors and methodology used in this approach, lack in stability, thus drifting apart along the course.

Table 6.2: *Obtained errors with the Extended Kalman Filter, in the initial lap (Init) and after 1 and 5 laps. The errors were measured in the initial position, after 1 meter and 3 meters.*

EKF Error			
NºLaps / Position	Init	1 Lap	5 Laps
Initial Point (0m)	0.01	0.49	2.07
1m from Intial Point	0.09	0.5715	2.545
3m from Initial Point	0.105	0.665	3.705

With this results, we could conclude that the odometry from the EKF provided a good approximation to the actual position, in the beginning of the course, as the trajectory from Rviz resembles the same of the car. However, upon completing more laps, the approach proves to be unreliable to use it alone, as significant error accumulation is noticed.

6.1.3 AMCL Results

Resulting from the acquisition of a static map from hector mapping and the updated odometry, it was possible using the Adaptive Monte-Carlo Localization, AMCL, to localize the robot in a given map. Provided all the necessary information, it can be observed in Figure 6.5 using Rviz, the particle filter generated by AMCL, that represent all the possible positions the car believes it is. As the car moves around the track, the particles can be seen agglomerating following the precise movement of the car.

Reading the values from the topic published by AMCL, */amcl_pose*, which provides the coordinates in X and Y, it could be concluded that the values obtained from the position were stable. In Table 6.3 the values of the deviation are presented, using the same methods as before for the other cases. From the values attained, we can conclude that AMCL proved to be working with expected precision, as the error values, even after 5 laps, only present, as a worst case analysed, a deviation of 7cm, in 3 meters apart from the starting point.

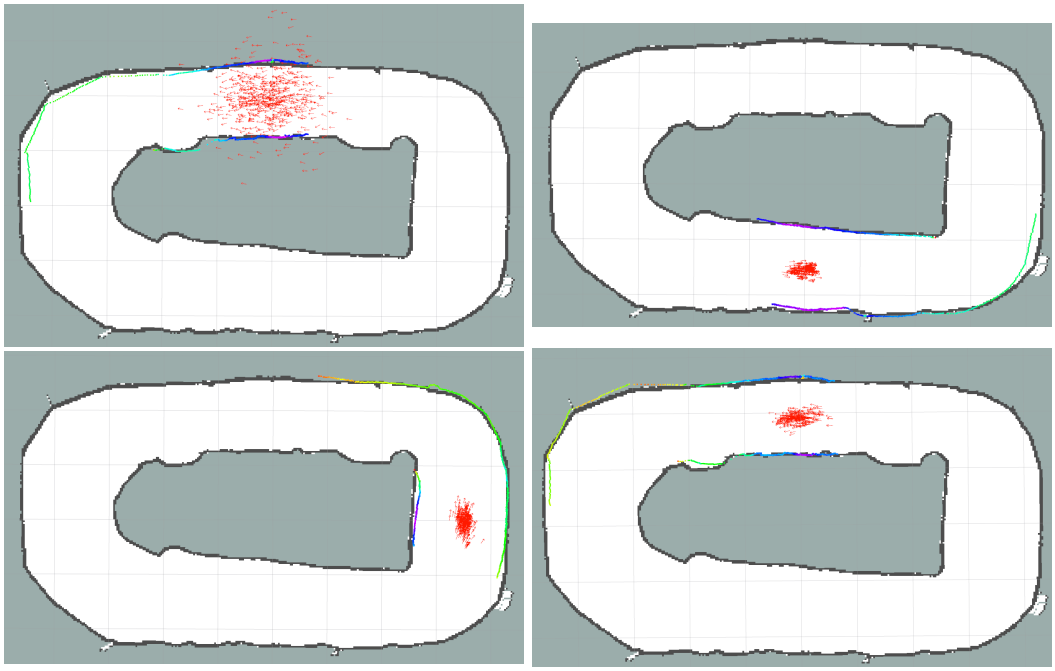


Figure 6.5: *Visualization, in Rviz, of AMCL's particle filter of the position estimation.*

Table 6.3: *Obtained errors with the Adaptive Monte-Carlo Localization, in the initial lap (Init) and after 1 and 5 laps. The errors were measured in the initial position, after 1 meter and 3 meters.*

AMCL Error			
NºLaps / Position	Init	1 Lap	5 Laps
Initial Point (0m)	0.0135	0.015	0.028
1m from Intial Point	0.055	0.01	0.035
3m from Initial Point	0.015	0.04	0.07

When driving in a more aggressive manner at higher speeds, it was possible to observe that the localization gave satisfactory results, even though at certain instances the position would give erroneous values. Despite this flaw, the algorithm was capable of getting track of its actual position after some time and it was possible to achieve good performance, as it will be observed in the following findings, regarding the paths obtained by the car when driving autonomously.

6.2 Control Algorithms

The following results show the trajectories obtained from the car when driving autonomously around a track, using different algorithms. In addition, time to complete a lap and various laps are taken into consideration as measurements of performance.

The algorithms used are the ones described in Chapter 5. To obtain the results for this experiences, it was used the combination of the Lidar and IMU odometry, EKF, alongside with AMCL to obtain a precise positioning of the car in a known map, that was acquired via Hector Slam. The position of the car is passed through the topic */amcl_pose*, which a node that we called, *waypoint_logger*, will read and save the coordinates of X and Y position, to a *.csv* file. With the saved coordinates in the *.csv* file, it was used another algorithm that proceeded to connect all consequent coordinates, until full trajectory was obtained and displayed in Rviz.

Furthermore, the procedures were all commenced in the same spot of the track, same as Figure 6.1, that is considered the initial point from the map generated, and all the laps and respective time were counted manually. For the acquisition of the path and time for multiple laps, it was stipulated a maximum number of 30 laps. Since the car could not be running indefinitely, it was necessary for it to stop at a certain number of laps, but at the same time give enough data and thus the 30 laps were the number stipulated for it. In the light of this experiments, 3 tests for each

algorithm were made and the number total of laps completed saved.

In this observations, the car was started and stopped, until it completed a full lap around the track. This experiment allows to examine the behaviour of the car, when preforming a single lap, when using different algorithms.

The results of this experiments for each algorithm are exhibited below

6.2.1 Wall Follow

The following demonstrate the results obtained using the first algorithm described earlier, the $WallFollow$. This method, uses a simplistic approach to control the steering of the car, along with a PID controller. By continuous manual adjust, the best constant values of the PID achieved were:

- Kp : 10
- Ki : 0.05
- Kd : 0.1

Additionally, the speed parameters were adjusted to obtain the fastest speed possible, without heavily compromising the stability of the control. This way the following are the values applied:

- Maximum speed = 6
- Minimum speed = 4.3
- Maximum distance = 5
- Minimum distance = 0.75

6.2.1.1 1 Lap

As it can be seen in Figure 6.6 the car took a very oscillating path like it was predicted. Since the algorithm only relies on the acquisition of the distance in two points, to be applied in a PID, it was expected to have volatile response. Henceforth the pattern of an 'S' shape the car makes when driving around the track. In terms of time, the path in the figure bellow took 11,58 seconds to preform one lap around the track. Consequently, after preforming 10 experiments, the average time was 11,67 seconds.

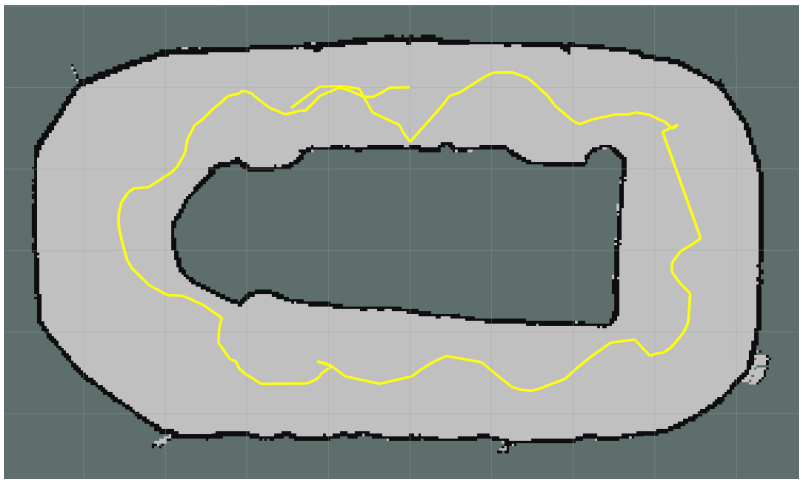


Figure 6.6: *Trajectory of one Lap using the "Wall Follow PID" algorithm.*

6.2.1.2 Multiple Laps

When put to test, this algorithm failed to complete the sixth lap as it can be seen in Figure 6.7, taking 00:58,73 (mm:ss,ms). Identical to the previous figure, the instability of this algorithm can detected, never succeeding to travel a correct and most of all safe path. The remaining test remained at 3 and 4 laps, lasting 00:38.37 and 00:48.69 (mm:ss,ms) respectively.

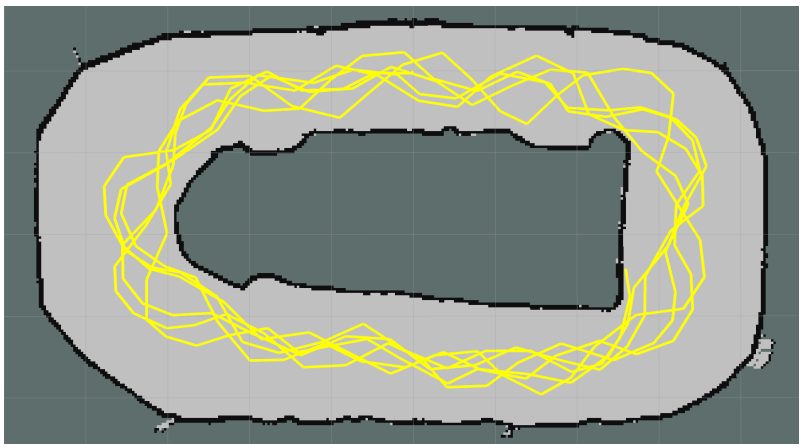


Figure 6.7: *Continuous trajectory of 5 Laps using the "F1/10 PID" algorithm.*

6.2.2 F1/10 PID

The following method, demonstrates the results of $F1/10$ PD algorithm. In the same manner, the constant parameters of the PD controller were manually adjusted and the ones achieved were:

- Kp : 3
- Kd : 25

Equally, the speed parameters were adjusted to obtain the fastest speed possible, without heavily compromising the stability of the control. This way the following are the values applied:

- Maximum speed = 8
- Minimum speed = 4.5
- Maximum distance = 5
- Minimum distance = 0.9

6.2.2.1 1 Lap

The implementation of the PID algorithm from F1 Tenth transpired a better result from the previous algorithm demonstrated, as more variables are taken into account for the control. However, tuning the PID parameters was challenging as a much higher Kp than a Kd resulted in a good response for straight lines and dreadful for curves, but when a higher Kd than a Kp was applied, the response was the opposite. In this manner, the path obtained from this algorithm can be seen in Figure 6.8, showing slightly less oscillations, although aggressive responses can be detected, much due to a bad tackle of the curve and adjustment of the PID control. Regarding time measurements, it took 9,91 seconds to complete the path and having an average of 9,85 seconds after 10 repetitions.

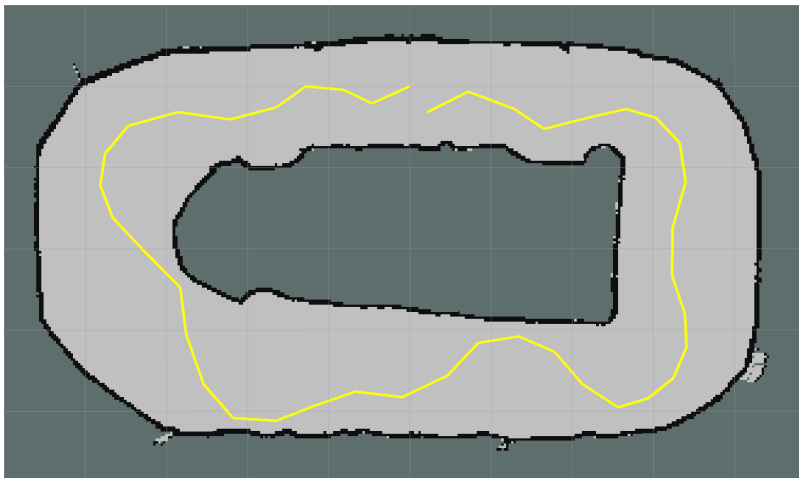


Figure 6.8: *Trajectory of one Lap using the "F1/10 PID" algorithm.*

6.2.2.2 Multiple Laps

Observing the results of the F1 Tenth PID approach, Figure 6.9, it can be concluded that this algorithm, when running more continuous laps, preforms much worse when compared to the path obtained for one lap. Through the 3 experiments, the car completed 6 laps twice in 1:09,58 and 1:03,43 respectively and 2 laps in 23,73 seconds, until it crashed.

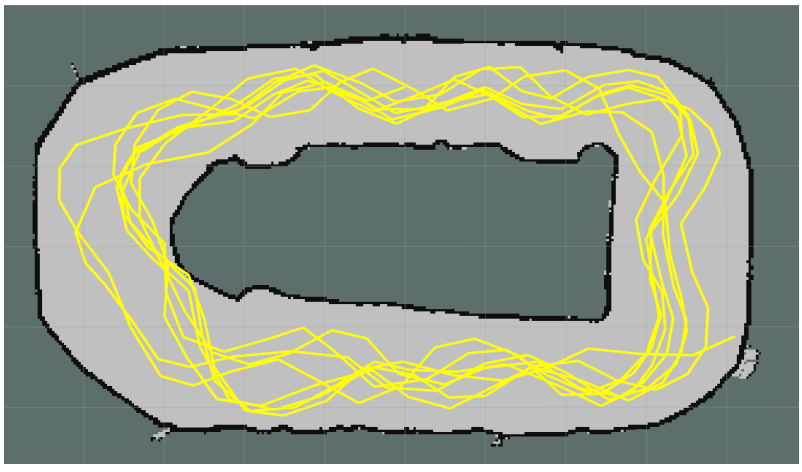


Figure 6.9: *Continuous trajectory of 6 Laps using the "F1/10 PID" algorithm.*

6.2.3 Curve and Line detection

Regarding this last PID implementation, there were 2 sets constants to be adjusted, as it was mentioned in Chapter 5. Following the same principles the values achieved were:
Straight Line:

- Kp : 1.2

Curve:

- Kp : 8
- Ki : 0.00007
- kd : 0.9

The speed parameters were adjusted to obtain the fastest speed possible, without heavily compromising the stability of the control. This way the following are the values applied:

- Maximum speed = 10
- Minimum speed = 4.5
- Maximum distance = 5
- Minimum distance = 0.9

6.2.3.1 1 Lap

As mentioned before, this algorithm is an approach to improve the PID implementation from F1 Tenth. As it identifies when the car is dealing with a curve or a straight line, this method not only proved to stabilize the cars sudden jumps, but also the general trajectory around the track. This results are demonstrated in Figure 6.10, where it is possible to notice a smother path along the straight lines of the track as well as in the curves. Following this path, the algorithm was able to complete the track in 8,08 seconds. The average of 10 experiments in the same conditions was 8,22 seconds.

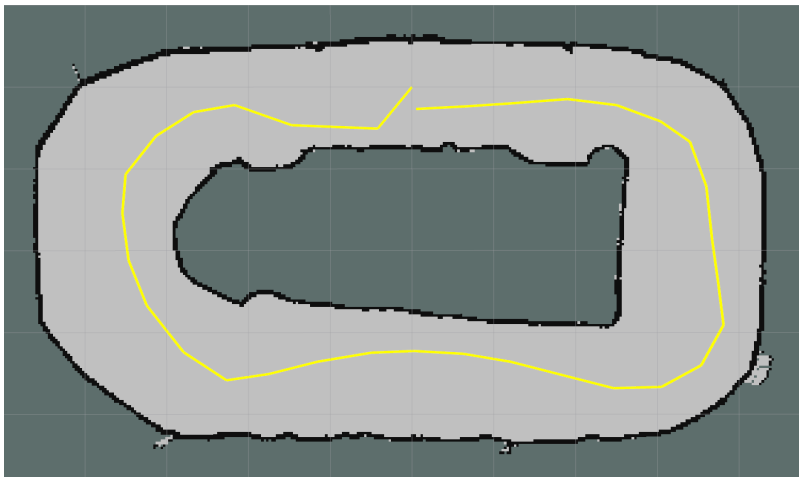


Figure 6.10: *Trajectory of one Lap using the "Curve and Line Detection PID" algorithm.*

Putting this algorithm through continuous driving, the car was able to produce the path in Figure 6.11, making 9 laps, almost reaching 10 laps, in under 1:16,85 (mm:ss,ms). This algorithm already demonstrates an improvement in all aspects comparing to the previous analysed. Not only the path is smother, but the time per lap decreased more than 1.5 seconds. However this method is still flawed, since it still crashes after some laps. Under the same conditions, 3 tests were made, resulting in 12 and 13 laps completed in a total time of 1:28,60 and 1:40,55 (mm:ss,ms) respectively.

Figure 6.11: *Continuous trajectory of 9 completed Laps using the "Curve and Line Detection PID" algorithm.*

6.2.4 Disparity

The following are the results obtained for the *Disparity* algorithm. Contrary to the previous algorithms, this one does not require adjustments with PID constants. However, some variables were fine tuned to properly meet our requirements.

With respect to the distance necessary to accept a disparity between two points, it was set as 1.5 meters and the minimum of angles necessary to cover a disparity was set to 10.

- Distance for Disparity = 1.5
- Number of points to accept a disparity = 10
- Compensation given to the steering = 20
- Lateral distanc

In terms of the speed parameters, the adjustments are as follows:

- Maximum speed = 13
- Minimum speed = 1
- Maximum distance = 5
- Minimum distance = 0.3

6.2.4.1 1 Lap

In Figure 6.12 it can be examined the route of the car when using the Disparity algorithm. As it was described in Chapter 5, the car will target a disparity in distances and plan the trajectory with accordance to it. With that granted, it is clear that car tackles the curves through the inside more aggressively, however a hard transition when coming out of the curve is detected. This is due to the algorithm not having a smooth transition when detecting a disparity. This transition breaks the straight line of the car should have, diminishing the optimization of the trajectory. As a result, it was possible to observe an improvement of the time for one lap, lasting 7,11 seconds and consequently, the average time for one lap rounds the 7,07 seconds.

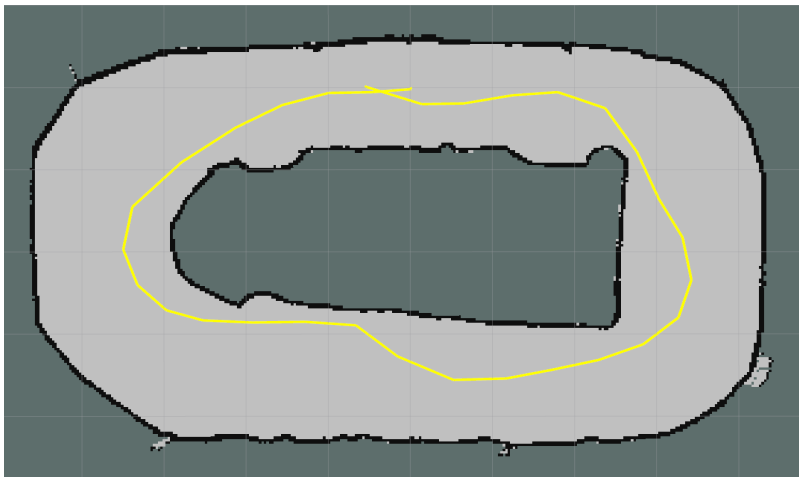


Figure 6.12: *Trajectory of one Lap using the "Disparity" algorithm.*

6.2.4.2 Multiple Laps

When facing continuous driving this algorithm reached the limit stipulated, the 30 laps. Under those circumstances, it could be assumed that the racecar would not crash and therefore proved to be the best approach so far. Figure 6.13 demonstrates the path of 30 laps which resulted in a total of 3:06,84. The outcome of the two remaining tests was 3:07,67 and 3:06,79.

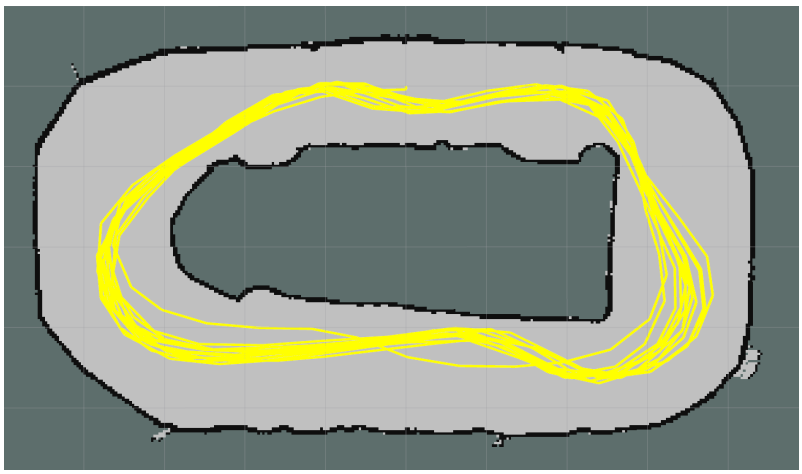


Figure 6.13: *Continuous trajectory of 30 Laps using the "Disparity" algorithm.*

6.2.5 Disparity Upgraded

The following are the results obtain for the *Disparity Upgraded* algorithm. Likewise to the proceeding algorithm, some parameters were manually tuned. With respect to the distance to accept a disparity it was set to 0.6 meters. As explained in the previous chapter, the number of angles to cover an existing object was set to be adaptable in this algorithm, so unlike the *Disparity* algorithm, no value was stipulated in this one.

In terms of the speed parameters, the adjustments are as follows. Although, in this algorithm, due to the high speed that was noticeable while doing curves, a restriction to the speed of 12 was added, when the steering angle was superior to 20 degrees.

- Maximum speed = 18
- Minimum speed = 1
- Maximum distance = 5
- Minimum distance = 0.3

6.2.5.1 1 Lap

Figure 6.14 demonstrates the path traveled by the car when using the upgraded version of the disparity algorithm. Similar to its previous version, it can be observed that the car also cuts the curves by the inside, however, when finishing it, the car leaves from the outside of it. Additionally, as mentioned in Chapter 5, the improvements made allowed to extend the limit of the speed applied to the motor. For this reason and considering the enhancement of the trajectory this algorithm was able to complete the path from the figure below in under 5,98 seconds. The 10 experiments for one lap lead to a mean time of 5,89 seconds.

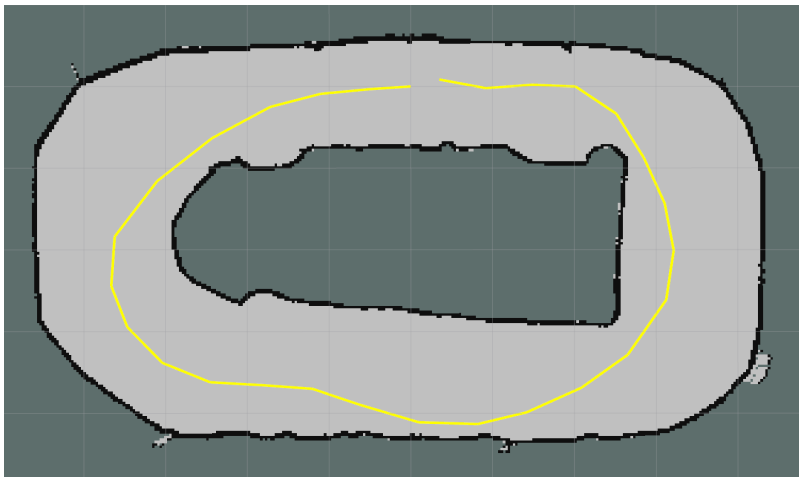


Figure 6.14: *Trajectory of one Lap using the upgraded version of the "Disparity" algorithm.*

6.2.5.2 Multiple Laps

Going under the continuous driving around the track, this algorithm preformed outstandingly. Not only was capable of never crashing, completing this way the 30 laps in every test, but also was able to reduce the time by 44 seconds, were all the results of the 3 tests were all 2:22, only changing around the milliseconds. Figure 6.15 displays the total routes of this algorithm.

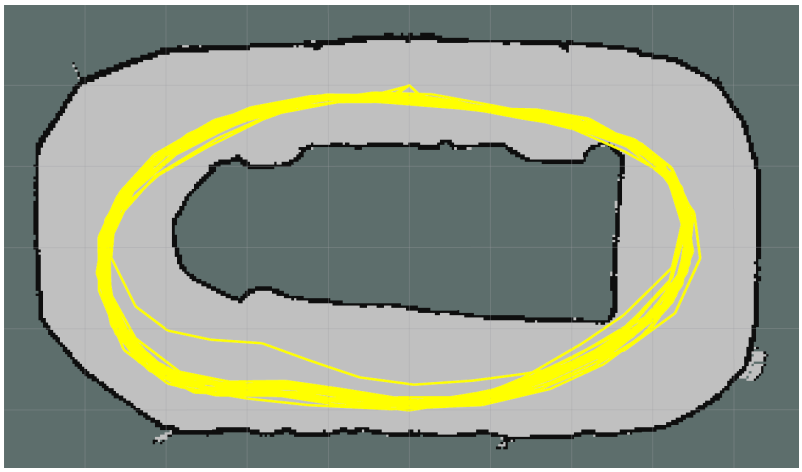


Figure 6.15: *Continuous trajectory of 30 Laps using the upgraded version of the "Disparity" algorithm.*

6.2.6 Results overview

Provided the previous results presented above, it can be clear which stood out in the end. We can conclude that the last 3 algorithms analysed, the curve and line detection PID, the *Disparity Extender* and the Upgraded version of it, yield the most stable paths. When compared to the remaining 3, it can be observed a smooth route when tackling a straight line, as well as when approaching a curve. On the other end, when compared into detail between them, the dissimilarities are clear.

As a first base of comparison, in Figure 6.16 we demonstrate a graphic displaying the average laps completed of each implemented and tested algorithm. From this perspective we can already assume that both Disparity approaches, provide better results, as they complete the most number of laps.

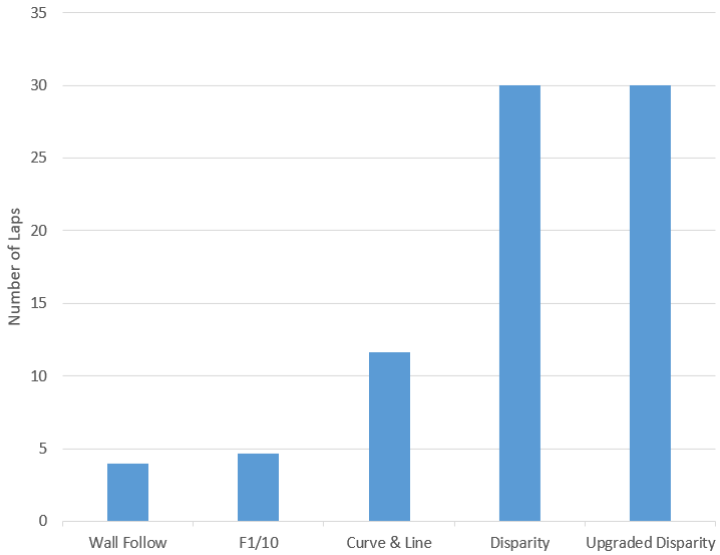


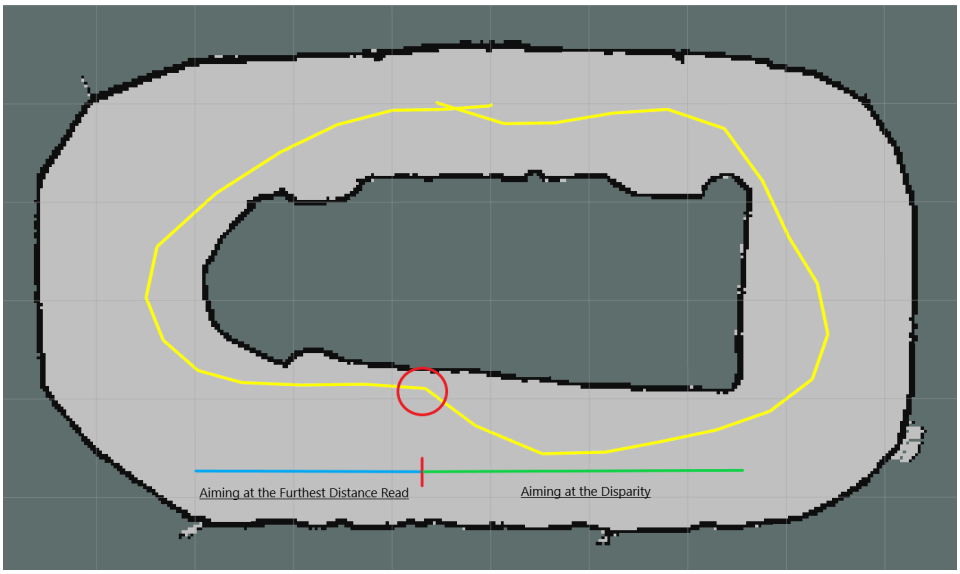
Figure 6.16: *Average number of Laps completed from each algorithm.*

The Curve and Line detection algorithm was not able to complete 30 Laps like the disparities algorithms. As it could be seen in Figure 6.11, the car ends up crashing, near the end of the 10th lap, when leaving the curve. This behaviour was due to the car approaching the curve to much on the outside, meaning a late transition from line to curve detection. As this late detection processed, the car is already driving close to the wall and at the same time, the speed control is starting to increase it, as the distance at 90 degrees is increasing as well. Having this increase of the speed, the car does not process this steering control effectively at a crucial time, ending up crashing.

In the Disparity algorithm, despite giving the insurance that it will "never"

crash, since it completed the 30 laps, some errors were detected while observing its trajectory. In both Figure 6.12 and Figure 6.13, it can be seen that the car suffers a sharp move at the middle of the straight line. Figure 6.17 points out these moments in a red circle, where this happens in both the 1 lap and 30 laps map.

This unnecessary movement, happened due to the transition in the algorithm where it had just finished a curve, entering in linear movement. During this period, a disparity was not found and the car is aiming to the furthest distance read in the array. When a new disparity is found, the car suffers a harsh steering command, since it needs to aim at new target, to enable the car to steer safely to that point. Consequently, this adjustment, makes the car lose valuable time, not only in a time-trial, but also in a head-to-head race.



(a)

(b)

Figure 6.17: *Indication of a transitional detection in the Disparity algorithm, where (a) refers to the one lap trajectory and (b) to the 30 laps.*

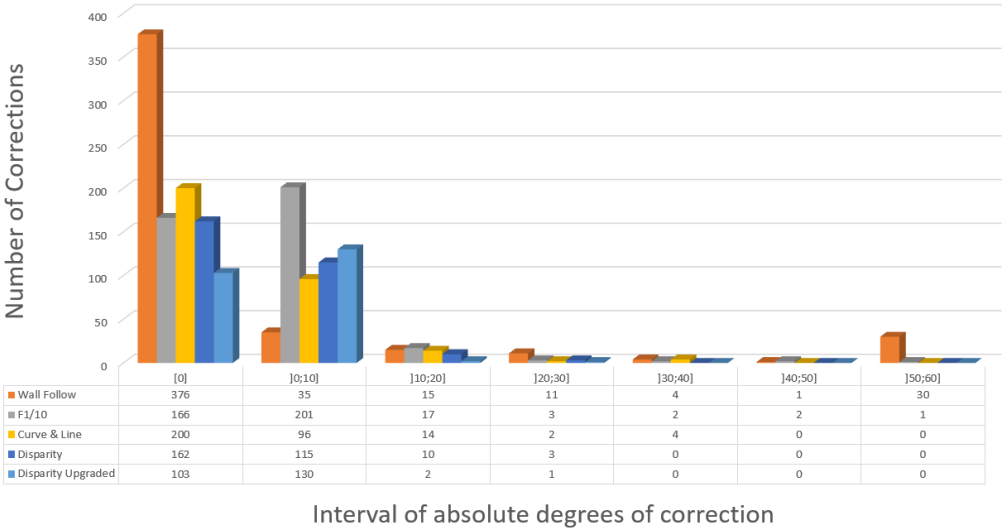
Thus, from the necessity to improve this last algorithm, surges the upgraded version of "Disparity Extender" algorithm, that corrected the previous error mentioned. As it could be seen in the previous Figure 6.14 and 6.15, the trajectory demonstrates an optimized approach to the problem outlined before. Comparatively, this method evidences similarities to the "Curve and Line Detection" algorithm, that also presents a sleeker transition from a curve to a straight line, until reaching the other curve. On the contrary to its original version, the car shown the ability to take full advantage of the straight line, since the average distances used for the speed calculus were much higher, implying a much higher speed to be applied in the motor.

To support the findings that were stated above, the degrees of correction in the steering, in one lap, were taken into consideration. Gathering each steering angle applied, we calculated the absolute difference of angles between steering corrections. With this metric we could observe how steep the corrections being made in each algorithm were, sustaining the findings on the trajectories.

In Figure 6.18 a graphic of the obtained results for each algorithm is displayed. The number of corrections are represented in defined intervals, that start in 0 alone and go up to]50,60] degrees, where 60 is the maximum correction possible, as it means the steering angle made a transition from -30° to 30°, or vice-versa.

From this graphic we could prove, besides the total number of iterations done in each algorithm, that the first two implementations required more steeper corrections, in comparison to the rest. The curve and line detection algorithm, if in one way shows smoother transitions, some abrupt corrections can still be detected.

In contrast, both disparity approaches, showed that most of the corrections are centered in]0;10] interval, however to prove the improvement from one algorithm to another, we can observe that the number of corrections from the Disparity Upgraded algorithm was reduced, comparing to the other. These reductions can be seen in the]20;30], from 3 to 1 corrections and a significantly higher reduction in the]10;20] interval, from 10 to 2 corrections. Based on these results we can affirm that the modifications applied to the disparity algorithm proved to be successful, as the number of sharper transitions applied in the steering improved and resulted in a smoother trajectory.



	[0]	]0;10]	]10;20]	]20;30]	]30;40]	]40;50]	]50;60]
Wall Follow	376	35	15	11	4	1	30
F1/10	166	201	17	3	2	2	1
Curve & Line	200	96	14	2	4	0	0
Disparity	162	115	10	3	0	0	0
Disparity Upgraded	103	130	2	1	0	0	0

Figure 6.18: *Graphical representation of the number of steering corrections of each algorithm.*

To finalize the analysis, time was also taken into consideration. The following are the results for the different times measured. For that, 10 experiments were done, timing the duration of each algorithm to complete one lap, while one starting in a static position and another while the car was already in movement, where the average time was then obtained. These results also serve to demonstrate the time loss on the startup of the car. In Figure 6.19, it is displayed the performance of the algorithms in both cases stated. In the same manner, it can be concluded that the "Upgraded Disparity" algorithm possesses the best results, achieving an average time per lap of 4.7 seconds. When compared to the "Disparity" Algorithm, which has an average of 6.27 seconds, it showed an improvement of approximately 1.57 seconds. A general observation of the times achieved by each algorithm, proved that the high oscillations of the trajectories, proved to have high impact on the duration for each lap and thus a slight decrease of the duration per lap, can be seen along the algorithms.

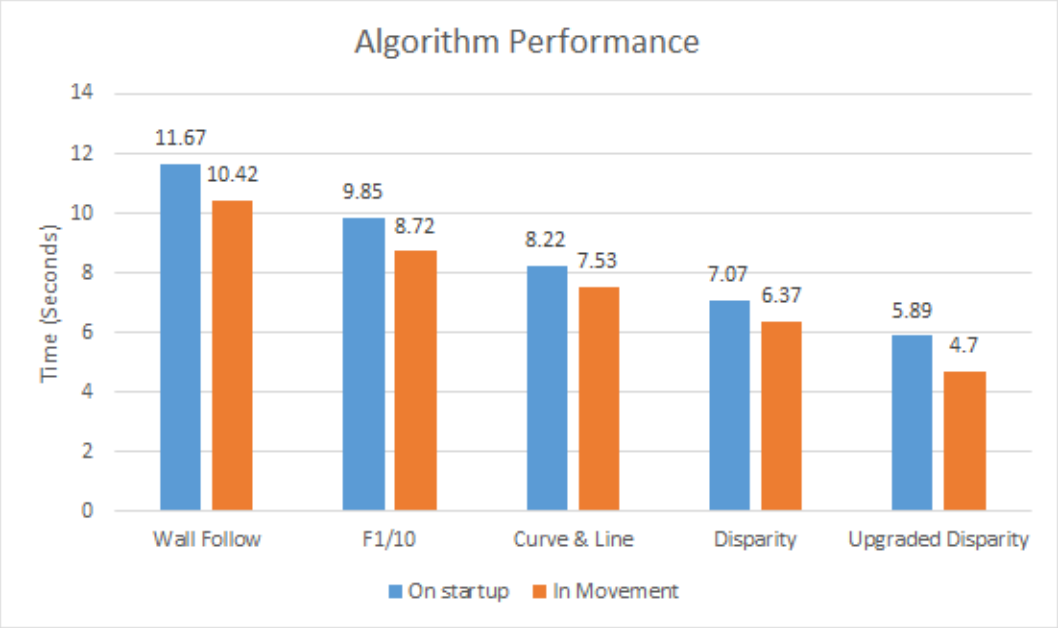


Figure 6.19: *Time comparison of each algorithm on startup and when in movement*

As a prof of concept, aside from *timetrial* results, experiments regarding obstacle avoidance were carried out using the *Disparity Upgraded* algorithm. For this specific scenario, boxes were placed in certain positions, that was thought the car would not able to dodge. In Figure 6.20, it is possible to visualize the map generated with Hector Slam in the Oval track, with the obstacles placed in it.

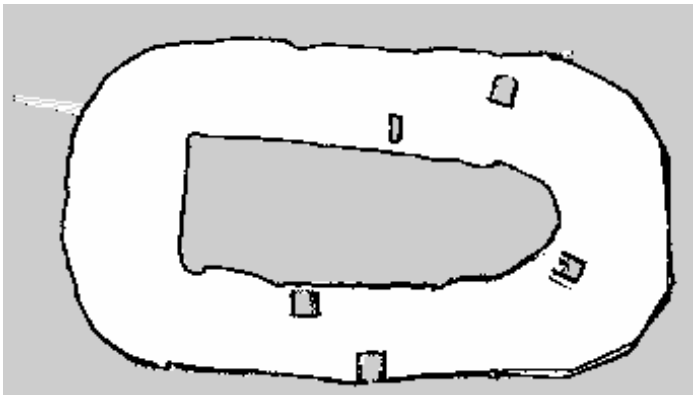


Figure 6.20: *Generated map with obstacles in Hector Slam.*

Consequently, in Figure 6.21 we can observe the path of one lap taken from the robot when faced with obstructions. As shown, the robot proceeded to adjust its

trajectory, avoiding effectively the proposed challenges. Although this may be true, this result was only possible, by reducing the PWM speed applied to a maximum of 14. When tested out with the maximum speed set to 18, like in the other tests, the car did not managed to avoid the obstacles, as it achieved such speed, making it impossible to apply the necessary corrections on time. With the purpose of accomplish obstacle avoidance, the maximum speed was reduced until a stable value was found and thus the 14 as the utmost PWM value in this algorithm. In addition in Figure 6.22 we can observe a sequence of images that demonstrate the car avoiding an obstacle.

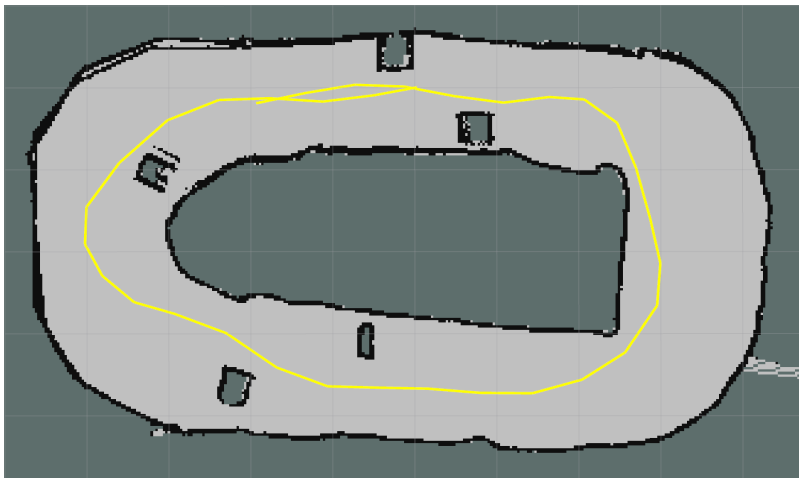


Figure 6.21: *Obstacle avoidance trajectory of one Lap using the Disparity Upgraded algorithm.*

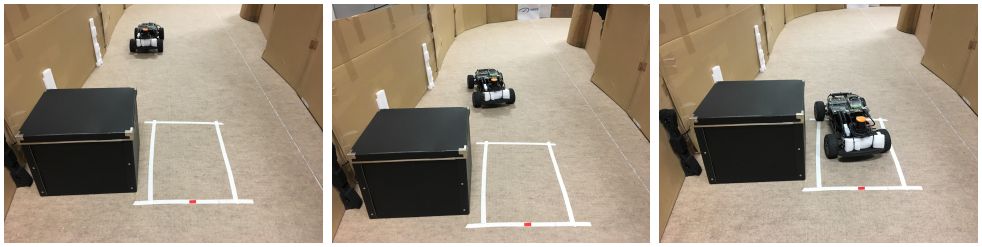


Figure 6.22: *Demonstration of the racecar avoiding an obstacle in real life.*

6.3 Concluding Remarks

In this chapter, we exposed the practical results that were obtained throughout the development of a robotic testbed for autonomous driving, demonstrating the outcome of the algorithms implemented.

A reflection upon the map generation and localization procedures were outlined, proving the possibility to acquire such data, whilst using a limited number of sensors. From the obtained results, we could conclude that the sensor fusion failed to provide useful data, for long periods of time. On the contrary, both SLAM and AMCL show that their pose estimation provided precise approximations to real life measurements. Although, the first could not function as a permanent localization for racing scenarios, as the car tends to lose its position when driving at high speeds. The AMCL, given the sensor inputs, proved to have a very successful implementation, only oscillating the values once in a while. Notably, in the acquisition of other sensors, as an example wheel encoders, we could improve the potential to attain greater and more accurate results, particularly in a localization aspect.

The performance of each implemented algorithm, previously described in Chapter 5, was obtained, taking into consideration the trajectory, time and steering angle corrections as evaluation parameters. To demonstrate it, coordinates from the car in a static map, were acquired using the Adaptive Monte-Carlo Localization, that when linked together, produced a perceivable path. On the other hand, time was measured manually, recording the duration of each lap. Observing the results, we came to the conclusion that a simpler and robust approach, from both Disparity algorithms, proved to dominate the methods that relied on PID controllers, by a wide margin. From the correction point of, it was proven that the PID algorithms applied much sharper transitions, explaining this way the trajectory attained from the other controls. In the end, the experiments proved to be important, since it has given the perspective on the difficulties that can be encountered, when applying control algorithms, in particular on PID approaches, where the probability to manually reach the optimal values are close to none.

In addition, the *Disparity Upgraded* algorithm was put to test, in track with random obstacles. The results showed that by slightly restraining the maximum speed, the car could avoid collision with obstacles, while achieving significantly good results in terms of time and trajectory.

To conclude, although one can affirm, and correctly, that a platform just like this, still has space for improvements, like adding more sensors, we can affirm that good baseline platform was reached, meaning this testbed is ready to support many diverse applications. For example, this testbed is already taking part on another research application, that is a robotic platooning testbed. In Figure 6.23, it is shown the end result of the robotic platform achieved in this thesis.

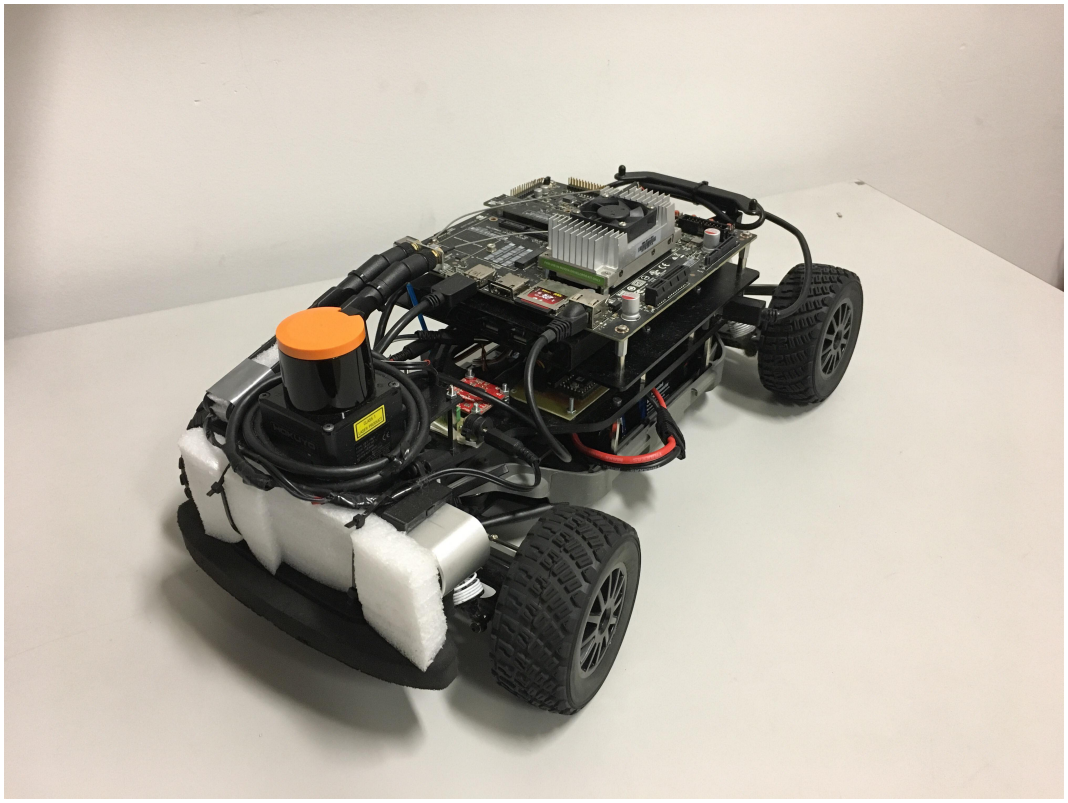


Figure 6.23: *Implemented robotic platform.*

7

Conclusions and Future Work

This thesis addressed the comparison of different algorithms for autonomous driving in a racing environment, as well as developing a robotic testbed for it, based on the Robot Operative System, ROS. In this context, although exploring different approaches, we followed the implementation provided by the F1 Tenth developers, also based on the MIT Racecar. With the robot assembled, we were able to study different approaches concerning sensing, planning and control when facing a closed lopped racetrack, while coming in contact with ROS and its various work tools and packages.

Relying on the Lidar and IMU as the sources of information to be processed, we were able to produce a perceivable static map, using the Hector Mapping tool, that later was used to achieve the robot localization in a two dimensional frame. For that a fusion of the odometry was required, to consequently be used by the Adaptive Monte-Carlo Localization, that enabled a position estimation of the autonomous vehicle in the given map. Although we achieved localization with these two sensors, we do believe it wasn't enough, as in certain situations, they failed to provide a precise estimation of position. Like previously stated, by adding more and preciser sensors, specially a wheel encoders, we expect to attain the precision we desire. That being said, we could conclude that localization achieved fulfilled our needs, enabling to acquire the position with enough precision, the demonstrate our results.

With our experimental work, we concluded that Upgraded version of the Disparity algorithm offered the best solution, by a large margin compared to the remaining

algorithms. The implementations that relied on a PID controller, on one hand proved to offer an inadequate and unreliable control, under performing in comparison to the Disparities algorithms. On the other hand, the endless search for the optimal PID constants, proved to consume much time, as they had to be manually tuned and tested, until acceptable parameters were found. Although we failed to reach such good results with the algorithms that relied on PIDs, we feel that if the track was wider, the controller would have more time to effectively respond. From this perspective, by taking extra tune into the constants and adjusting some parameters, like the speed, we assume that the results would improve, in comparison to the ones reached up until this point.

Summarizing, we confirmed, out of the algorithms examined, that the modifications applied in the original disparity algorithm, evidenced a major improvement, being considered the best algorithm to rely on a racing competition, out of the ones evaluated. To emphasize, this controller besides dominating a time-trial race, it possesses the capability of going head-to-head against a challenger, since it was proven it can dodge obstacles while driving at high speeds. To conclude, we can affirm that we have positively tuned into an autonomous racing competition .

Regarding future works, besides improving the algorithm in both speed and steering control, we aim on developing and implementing new controllers, for instance a Model Predictive Controller,MPC, pursuing the optimal algorithm for autonomous racing. To achieve this results, we believe that upgrading the robots odometry, by adding more sensors, we could effectively attain more accurate localization and thus explore alternative navigation approaches.

Bibliography

[1] "SAE International Releases Updated Visual Chart for Its âLevels of Driving Automationâ Standard for Self-Driving Vehicles,", https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles, (Accessed on 10/03/2019).

[2] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, "Perception, Planning, Control, and Coordination for Autonomous Vehicles," Machines 5 (2017).

[3] "PiBorg — MonsterBorg - The Ultimate Raspberry Pi Robot,", https://www.piborg.org/robots-1/monsterborg, (Accessed on 09/06/2019).

[4] "Formula Pi — Self-driving robot racing with the Raspberry Pi,", https://www.formulapi.com/, (Accessed on 09/06/2019).

[5] "Platform Specs AutoRally,", https://autorally.github.io/specs/, (Accessed on 09/06/2019).

[6] "About BARC,", http://www.barc-project.com/about-1, (Accessed on 09/06/2019).

[7] "945-82771-0000-000 NVIDIA® Jetson™ TX2 Developer Kit (US/CA) by NVIDIA Embedded System Development Boards and Kits Arrow.com,", https://www.arrow.com/en/products/945-82771-0000-000/nvidia, (Accessed on 09/08/2019).

[8] "Teensy 3.2,", https://www.pjrc.com/store/teensy32.html, (Accessed on 09/08/2019).

[9] "Hokuyo-USA :: UST-10LX,", https://www.hokuyo-usa.com/products/scanning-laser-rangefinders/ust-10lx, (Accessed on 09/08/2019).

[10] "SparkFun 9DoF Razor IMU M0 - SEN-14001 - SparkFun Electronics,", https://www.sparkfun.com/products/14001, (Accessed on 09/08/2019).

[11] "Traxxas Ford Fiesta ST Rally — RC Rally Car,", https://traxxas.com/products/models/electric/ford-fiesta-st-rally, (Accessed on 09/08/2019).

[12] "Code,", http://f1tenth.org/code/legacy1/code_legacy1.html#s2l3, (Accessed on 09/08/2019).

[13] N. Otterness, "The "Disparity Extender" Algorithm, and F1/Tenth,", Online at https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html, 2019, (Accessed on 09/09/2019).

[14] "WHO — World Health Organization,", https://www.who.int/, (Accessed on 10/03/2019).

[15] J. Fleetwood, "Public Health, Ethics, and Autonomous Vehicles," American Journal of Public Health **107,** 532–537 (2017), pMID: 28207327.

[16] K. Bimbraw, "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology," In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, **01,** 191–198 (2015).

[17] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and Decision-Making for Autonomous Vehicles," Annual Review of Control, Robotics, and Autonomous Systems **1,** 187–210 (2018).

[18] J. V. Brummelen, M. OâBrien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," Transportation Research Part C: Emerging Technologies **89,** 384 – 406 (2018).

[19] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)* (The MIT Press, 2005).

[20] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving," IEEE Transactions on Intelligent Vehicles **2,** 194–220 (2017).

[21] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, "A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications," IEEE Internet of Things Journal **5,** 829–846 (2018).

[22] B. Paden, M. ÄÃ¡p, S. Z. Yong, D. Yershov, and E. Frazzoli, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," IEEE Transactions on Intelligent Vehicles **1,** 33–55 (2016).

[23] M. O'Kelly *et al.*, "F1/10: An Open-Source Autonomous Cyber-Physical Platform," CoRR abs/1901.08567 (2019).

[24] "Formula Pi - Self-driving robot racing with the Raspberry Pi by Timothy Freeburn â Kickstarter,", https://www.kickstarter.com/projects/frobotics/formula-pi-self-driving-robot-racing-with-the-rasp, (Accessed on 09/06/2019).

[25] "GitHub - piborg/monster-self-drive: Easy to understand self-driving example for MonsterBorg,", https://github.com/piborg/monster-self-drive, (Accessed on 09/06/2019).

[26] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, "AutoRally: An Open Platform for Aggressive Autonomous Driving," IEEE Control Systems Magazine **39,** 26–55 (2019).

[27] G. Williams, A. Aldrich, and E. A. Theodorou, "Model Predictive Path Integral Control: From Theory to Parallel Computation," Journal of Guidance, Control, and Dynamics **40,** 344–357 (2017).

[28] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1433–1440 (2016).

[29] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721 (2017).

[30] J. Gonzales, F. Zhang, K. Li, and F. Borrelli, "Autonomous drifting with onboard sensors," In , (2016).

[31] F. R. Zhang, J. Gonzales, K. Li, and F. Borrelli, "Autonomous Drift Cornering with Mixed Open-loop and Closed-loop Control," In , (2017).

[32] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, *ICRA Workshop on Open Source Software* (2009).

[33] "ROS/Introduction - ROS Wiki,", http://wiki.ros.org/ROS/Introduction, (Accessed on 09/06/2019).

[34] "ROS/Concepts - ROS Wiki,", http://wiki.ros.org/ROS/Concepts, (Accessed on 09/06/2019).

[35] "Master - ROS Wiki,", http://wiki.ros.org/Master, (Accessed on 09/06/2019).

[36] "Nodes - ROS Wiki,", http://wiki.ros.org/Nodes, (Accessed on 09/06/2019).

[37] "Messages - ROS Wiki,", http://wiki.ros.org/Messages, (Accessed on 09/06/2019).

[38] "Topics - ROS Wiki,", http://wiki.ros.org/Topics, (Accessed on 09/06/2019).

[39] "Bags - ROS Wiki,", http://wiki.ros.org/Bags, (Accessed on 09/06/2019).

[40] "Gazebo,", http://gazebosim.org/, (Accessed on 09/06/2019).

[41] "NVIDIA Jetson TX2 Developer Kit,", https://developer.download.nvidia.com/embedded/L4T/r32-2_Release_v1.0/jetson_tx2_developer_kit_user_guide.pdf?xvNiBFTzWLbqQaTYAJGsj-uspxyumDTVswIh2WSTd1dsuS4tCjcHOEVfr5hAhxA98MgUK3cmomovx6yFQyxZyunCQx2kxBykLhw, (Accessed on 09/06/2019).

[42] "RACECAR,", https://mit-racecar.github.io/, (Accessed on 09/08/2019).

[43] "GitHub - mlab-upenn/f1tenthpublic,", https://github.com/mlab-upenn/f1tenthpublic, (Accessed on 09/14/2019).

[44] "9DoF Razor IMU M0 Hookup Guide - learn.sparkfun.com,", https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide?_ga=2.159944827.77366589.1564527235-314315670.1557151784, (Accessed on 09/14/2019).

[45] "razor_imu_9dof - ROS Wiki,", http://wiki.ros.org/razor_imu_9dof#Load_Firmware_into_Razor_Board, (Accessed on 09/14/2019).

[46] "rosserial_python - ROS Wiki,", http://wiki.ros.org/rosserial_python, (Accessed on 09/18/2019).

[47] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, (2011).

[48] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. Von Stryk, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," In , pp. 624–631 (2014).

[49] S. A. S. Mohamed, M. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, "A Survey on Odometry for Autonomous Navigation Systems," IEEE Access **7,** 97466–97486 (2019).

[50] S. Y. Chen, "Kalman Filter for Robot Vision: A Survey," IEEE Transactions on Industrial Electronics **59,** 4409–4420 (2012).

[51] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, (Springer, 2014).

[52] M. Jaimez, J. G. Monroy, and J. González-Jiménez, "Planar Odometry from a Radial Laser Scanner. A Range Flow-based Approach," In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4479–4485 (2016).

[53] "amcl - ROS Wiki,", http://wiki.ros.org/amcl, (Accessed on 09/19/2019).

[54] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, **2,** 1322–1328 vol.2 (1999).

[55] D. Fox, "KLD-Sampling: Adaptive Particle Filters," In Advances in Neural Information Processing Systems 14 (2002).

[56] C. Walsh and S. Karaman, "CDDT: Fast Approximate 2D Ray Casting for Accelerated Localization," abs/1705.01167 (2017).

[57] "REP 105 – Coordinate Frames for Mobile Platforms (ROS.org),", `https://www.ros.org/reps/rep-0105.html`, (Accessed on 09/19/2019).