Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Daniela Ferreira
Pinto Dias Rato

Detection of the Navigable Road Limits by Analysis of the Accumulated Point Cloud Density

Deteção dos Limites Navegáveis da Estrada por Análise da Densidade de Nuvens de Pontos Acumulados

Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Daniela Ferreira
Pinto Dias Rato

Detection of the Navigable Road Limits by Analysis of the Accumulated Point Cloud Density

Deteção dos Limites Navegáveis da Estrada por Análise da Densidade de Nuvens de Pontos Acumulados

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtencão do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado com Agregação da Universidade de Aveiro.

O júri / The jury

Presidente / President	**Prof. Doutor Miguel Armando Riem de Oliveira** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor Luís Manuel Conde Bento** Professor Adjunto da Escola Superior de Tecnologia e Gestão de Leiria (arguente)
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado com Agregação da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Primeiro que tudo, quero agradecer ao professor Vítor Santos pelo acompanhamento, dedicação e pela enorme paixão pelo trabalho que nos incutiu ao longo destes meses. Pelas discussões sempre enriquecedoras e por arranjar sempre tempo mesmo este quando é escasso. Acima de tudo, agradeço pela motivação contangiante que consegue transmitir mesmo nos piores momentos.

Aos meus pais, por me proporcionarem esta oportunidade, pelo apoio incondicional em todas as decisões e por fazerem tudo para que nada me falte. À minha irmã Alice, pelas brincadeiras, pelos risos, e pelas conversas. Ao João, pelo apoio, motivação e principalmente pelo amor.

À Ana por estar sempre presente em todas as fases da minha vida.

Finalmente, agradeço também aos meus colegas do LAR, particularmente ao Manuel e ao Tiago, pelos bons momentos, pela entreajuda e pela troca de ideias que tanto contribuiram para esta dissertação. Um especial obrigada ao Tiago por todo o trabalho que foi feito no AtlasCar2 e no terreno.

keywords

LIDAR; Perception; Curb Detection; Road Limits; AtlasCar2; Point Cloud; Density; ROS; Gradient; Edge Detection

abstract

As part of the Atlas project, this dissertation aims to identify the navigable limits of the road by analyzing the density of accumulated point clouds, obtained through laser readings from a SICK LD-MRS sensor. This sensor, installed in front of the AtlasCar2, has the purpose of identifying obstacles at road level and from it the creation of occupation grids that delimit the navigable space of the vehicle is proposed. First, the point cloud density is converted into an occupancy density grid, normalized in each frame in relation to the maximum density. Edge detection algorithms and gradient filters are subsequently applied to the density grid, in order to detect patterns that match sudden changes in density, both positive and negative. To these grids are applied thresholds in order to remove irrelevant information. Finally, a methodology for quantitative evaluation of algorithms was also developed, using KML files to define road boundaries and, relying on the accuracy of the GPS data obtained, comparing the actual navigable space with the one obtained by the methodology for detection of road boundaries and thus evaluating the performance of the work developed. In this work, the results of the different algorithms are presented, as well as several tests taking into account the influence of grid resolution, car speed, among others. In general, the work developed meets the initially proposed objectives, being able to detect both positive and negative obstacles and being minimally robust to speed and road conditions.

palavras-chave

LIDAR; Perceção; Deteção de passeios; Limites da estrada; AtlasCar2; Nuvens de Pontos; Densidade; ROS; Gradiente; Deteção de arestas

resumo

No âmbito do projeto Atlas, esta dissertação prevê a identificação dos limites navegáveis da estrada através da análise da densidade da acumulaçao de nuvens de pontos, obtidas através de leituras laser provenientes de um sensor SICK LD-MRS. Este sensor, instalado na frente do AtlasCar2, tem como propósito a identificação de obstáculos ao nível da estrada e a partir dos seus dados prevê-se a criação de grelhas de ocupação que delimitem o espaço navegável do veículo. Em primeiro lugar, a densidade da nuvem de pontos é transformada numa grelha de densidade normalizada em cada frame em relação à densidade máxima, à qual posteriormente são aplicados algoritmos de deteção de arestas e filtros de gradiente com o objetivo de detetar padrões que correspondam a mudanças súbitas de densidade, tanto positivas como negativas. A estas grelhas são aplicados limiares de forma a eliminar informação irrelevante. Por fim, foi desenvolvida também uma metodologia de avaliação quantitativa dos algoritmos, usando ficheiros KML para deliniar limites da estrada e, contanto com a precisão dos dados de GPS obtidos, comparar o espaço navegável real com o obtido pela metodologia de deteção de limites de estrada e assim avaliar o desempenho dos algoritmos desenvolvidos. Neste trabalho são apresentados resultados dos diferentes algoritmos, bem como diversos testes tendo em conta a influência da resolução de grelha, velocidade do carro, entre outros. O trabalho desenvovido cumpre os objetivos propostos inicialmente, sendo capaz de detetar ambos obstáculos positivos e negativos e sendo minimamente robusto a velocidade e condições de estrada.

Contents

List of Tables

Intentionally blank page.

List of Figures

Acronyms

AD Autonomous Driving. 1, 3

ADAS Advanced Driver's Assistance Systems. 2, 3, 17

AI Artifitial Intelligence. 1

FN false negative. 13, 45

FP false positive. 13, 45, 49, 57

GNSS Global Navigation Satellite System. 17

GPS Global Positioning System. 17, 18, 20, 27, 39, 40, 42, 43

IMU Inertial Measurement Unit. 17, 20, 39, 42

KML Keyhole Markup Language. 14, 23, 39–41

LAR Laboratory for Automation and Robotics. 2

LIDAR Light Detection And Ranging. 1, 4, 7–10, 12, 15, 16, 20, 23, 25, 45, 61, 65, 66

OGC Open Geospatial Consortium. 39

OpenCV Open Source Computer Vision Library. 26, 31, 32

PCL Point Cloud Library. 25, 28

ROC Receiver Operating Characteristics. 45

ROS Robot Operating System. 5, 19, 20, 25, 28, 31, 39, 69

RPY roll-pitch-yaw. 20

SPAN Synchronous Position, Attitude and Navigation. 17

TN true negative. 13, 45

TP true positive. 13, 45

UPS Uninterruptible power supply. 19

UTM Universal Transverse Mercator. 42

WGS 84 World Geodetic System. 42

Chapter 1

Introduction

In today's world, there is a constant search for speed, autonomy, and efficiency. People incessantly demand solutions and technologies that give the possibility to upgrade their lifestyle and to save them time. In that line, the development of Autonomous Driving (AD) is growing at high speed and is one of the most studied topics and with faster development in the automobile industry at the present time.

AD is a complex combination of various components that can be defined as systems where perception and detection, decision making, and the manipulation of the automobile are performed by electronics and machinery, via Artifitial Intelligence (AI), instead of humans. According to SAE International, there are 5 levels to define the degree of AD of a vehicle [8]:

1. Level 0: No autonomy
2. Level 1: Driver Assistance
3. Level 2: Partial Automation
4. Level 3: Conditional Automation
5. Level 4: High Automation
6. Level 5: Full Automation

One of the first and most important fields of AD development is the perception and detection of road, that begins immediately at level 1, where classical control elements are assisting the driver in speed control and steering. Yet, to determine the navigable road limits, that is, the limits where it is safe for the vehicle to move is not an objective task and requires complex algorithms that guaranty of safety at every moment.

The proposed here is to find a solution to these problems with the usage of a 4 layers 3D Light Detection And Ranging (LIDAR) mounted on the front of a car, close to the ground. LIDAR sensors measure the distance to the closest object and with them, it is possible to withdraw a point cloud with three-dimensional points corresponding to the intersection of laser beams with the objects ahead.

1.1 Background - Atlas Project

The theme of this dissertation appears in the context of the Atlas project. This project was created by the Group of Automation and Robotics at Laboratory for Automation and Robotics (LAR), on the Department of Mechanical Engineering of the University of Aveiro in 2003. The main focus of this project was allowing the development of advanced sensing and active systems designed for implementation in automobiles and similar platforms [4].

In early years, the Atlas Project was based on mobile autonomous robots (Figure 1.1), competing in several robotics competitions and winning multiple awards.

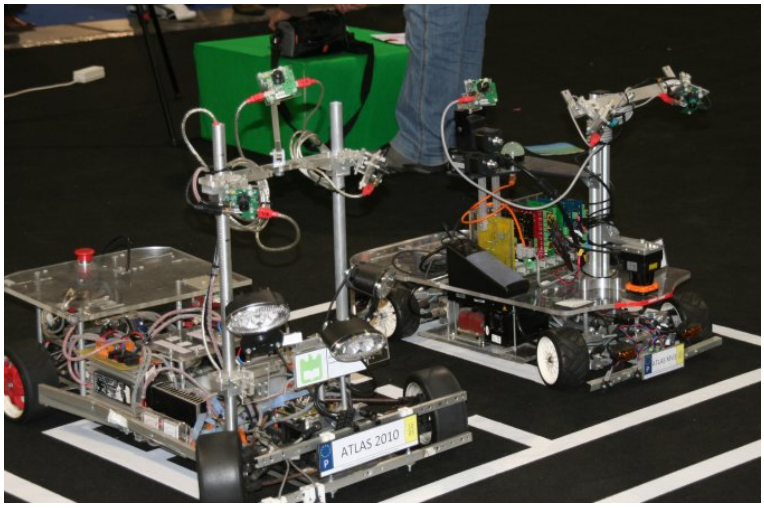


Figure 1.1: Atlascar 2010 and AtlasMV.

Later on, a Ford Escort model was equipped with several sensors and the *AtlasCar* emerged (Figure 1.2), constituting a prototype for research on Advanced Driver's Assistance Systems (ADAS) [4]. AtlasCar was able to perceive its surroundings [3, 13] and react accordingly using multiple actuators to move its mechanical components [20].

Figure 1.2: The first AtlasCar.

After severe modifications and years of wear, the *AtlasCar* was no longer suitable to the project's needs and to the evolution of technology. The project was reborn with the arrival of the *AtlasCar2* in 2015 (Figure 1.3). This new prototype is a *Mitsubishi i-MiEV* and brings a new world of possibilities to the project. Taking into account that it is electric and doesn't need gear changing, it makes the autonomous control potentially easier and allows gathering power directly from the batteries.

Figure 1.3: *AtlasCar2*.

1.2 Problem Description

With the growing development of AD, road detection and perception techniques are continuously being studied and perfected.

To achieve the goal of a fully functional prototype in ADAS, several studies were developed in previous years under the Atlas Project to find the navigable road limits hopefully allowing for navigation algorithms to know where the car can and cannot go.

In the previous *AtlasCar*, the accumulation of laser readings with the car movement proved to produced good results in real time road limits detection/reconstruction ahead of the car [23]. Laser readings can be converted into point clouds to which algorithms are applied, resulting in the identification of road curbs or similar obstacles at road level.

In that line, the general problem was to positively identify road limits, by the identification of obstacles like road curbs through the analysis of accumulated point cloud density. The road limits will be evaluated with the concept of navigable space, defined as the space within road limits, where the car can, allegedly, navigate with safety. To

accomplish that, a SICK LDMRS 4-Layer LIDAR was mounted on the front of the car. The advantage of placing the sensor in front of the car, unlike most solutions in road detection that use 360^{o} LIDARs on top of the car, is to focus more in objects at road level and to have a detailed perspective near the ground and not a general vision of the road with bigger surroundings that most times don't affect driving. In this way, road curbs, holes, and similar obstacles are identified with more detail.

Although some work was developed by T. Marques by filtering only high-density points in [12], the found solution was not robust enough and didn't produce the expected results in many situations. The focus of this dissertation is to improve the solution to work in all situations.

The general idea now proposed is, not only to take advantage of point cloud density but also of the density gradient as an edge detection tool and other edge detection filters like *Sobel* and *Canny* to identify density changes that usually define obstacles. This technique is a step ahead of density analysis taking into account that it considers both positive and negative variations of density, allowing to also identify negative obstacles.

Also planned is the development of a tool that in a semi-automated way evaluates the performance of the algorithms and the quality of the detected road limits.

1.3 Objectives

The main goal of this dissertation is to achieve real-time road detection, identifying obstacles at road level with the possibility of making decisions instantaneously. To meet this, data will be gathered from a 3D LIDAR and the innovating concept is to take advantage of the car movement to reconstruct the road directly ahead of the car. Also, to complement T. Marques's work ([12]), identify low curbs and inverted curbs is a priority.

Regarding the detection of road limits, the focus is to develop a robust algorithm, independent from the car velocity, orientation and road conditions. To this end, the influence of several factors in the achieved results was tested and modifications made to optimize the parameters to the best results possible. Different edge detection techniques were also implemented to evaluate which one would be most appropriate in the situation.

Considering this, the development of a tool to quantitatively evaluate the performance of an algorithm is also an important task to achieve, not only to compare algorithms between them and optimize the algorithm's parameters but also to evaluate the general performance of road detection. The objective here is to develop an automated program that requires the user minimal effort and automatically performs and exports calculations ready to insert in a data sheet.

Finally, this works also assumes the integration and testing of the developed methodologies in real time on board of the *AtlasCar2*.

Briefly, the purpose of this work is to:

- Develop a robust solution for road detection
- Test and integrate the solution onboard of the *AtlasCar2*
- Develop a methodology to perform quantitative evaluation of road limits.

1.4 Document Structure

This document is formed by 7 chapters:

- Chapter 1, this present one, is the introduction. In this chapter, the goal was to contextualize the problem and background and to establish goals to achieve at the end of the work;
- Chapter 2 describes the related work both at the University of Aveiro and at other institutions;
- Chapter 3 explains the used infrastructure, both hardware and software, describing features of the several sensors installed in the *AtlasCar2* and the architecture of the developed Robot Operating System (ROS) packages;
- Chapter 4 describes the methodology used to identify road limits and the navigable space;
- Chapter 5 defines the developed quantitative evaluation methodology and the integration with Google Earth;
- Chapter 6 discusses the experimental work and the influence of several external and internal factors in the performance of the algorithms;
- Chapter 7 presents a conclusion to the work described in this thesis, globally evaluating the results and discussing the future work that may improve the methodology developed.

Intentionally blank page.

Chapter 2

Related Work

As mentioned before, this thematic is studied by many people around the world and there are many solutions and different methodologies to overcome the problem. Some of the existing solutions will be described in this chapter.

When it comes to road detection and identifying the navigable road limits, there are clearly two very different approaches: camera-based methods and LIDAR-based methods. The first one is very influenced by external factors as weather conditions, etc., yet more inclusive, as the second one represents a more robust system but more difficult to work with. The ideal solution would be to incorporate both techniques and to take advantage of each one's strengths.

2.1 Detection of Road Limits

2.1.1 Work Developed at the University of Aveiro

Following the study done in previous master's thesis at the University of Aveiro in road detection using LIDAR, the majority of the work was developed by Tiago Marques in [12].

T. Marques placed the LIDAR in the front of the car, pointing to the ground, with the goal of scanning obstacles at road level and define the road limits by the detection of road curbs. The advantage of placing the sensor close to the ground is the possibility to focus only on obstacles and objects close to the ground, which is the most important feature in road limits definition.

This work takes advantage of the car movement to accumulate LIDAR scannings and obtain an accumulated point cloud. To this cloud, an algorithm is applied that scans each point and removes it if there are not enough neighbor points in a predefined radius. What this means is that it is only considered a road limit a place with high point density.

In his work, T. Marques presents two types of algorithms: one static and one dynamic, where the parameters change according to the car velocity. The last one presents more reliable results, yet it still has problems with not well-defined road curbs and mainly when there is an inverted road curb profile, meaning negative obstacles. This algorithm also presents problems with sudden changes in velocity and direction, which influences the roll and pitch of the car. This occurrence changes the distance from the sensor to the ground and also the inclination of it, which can result in very high or very low accumulation of points, depending on the situation.

For example, when the car is accelerating, the pitch of the vehicle increases and the sensor changes orientation too, making the LIDAR readings further from the car and

more apart, creating a low-density cloud, resulting in less information to analyze.

On the contrary, in deceleration the pitch decreases and the point cloud becomes closer to the ground with much higher density, which can be misleading and falsely identified as a curb.

Also, the changing of direction can create a bigger concentration of point in one side of the road, misleading the algorithms into thinking it is a road curb.

These little nuances provoke constant changes of density profiles and consequently of information which difficult the development of an algorithm that works for every case. To create a comprehensive solution is one of the most difficult challenges in improving T. Marques' algorithm.

2.1.2 Other Work

When it comes to road limits or curb detection using LIDAR there are many different approaches. Contrary to the used solution, most methodologies take advantage of 360°LIDAR on top of the car, like [27], [19], [9] and [26], allowing to detect higher objects like trees and other cars. However, creating less definition in ground level objects.

Many methods also take advantage of elevation profiles, by filtering objects according to their elevation difference and grouping the ones with similar elevations.

For example, in [6], a prediction method is used to find the height difference between two points and create an elevation map with the predicted measures. If all the points are on the ground, ΔH is 0, if not, there will be a difference of heights.

In Figure 2.1, points A, B, and C are obtained by three adjacent beams in the same circular direction. With equation 2.1, the difference of heights between each point it is calculated, considering a linear regression between point A and B and the height of C in the continuation of the calculated line.

$$\Delta H = Z_C - Z_B - (Z_B - Z_A) * \frac{d_{BC}}{d_{AB}} \tag{2.1}$$

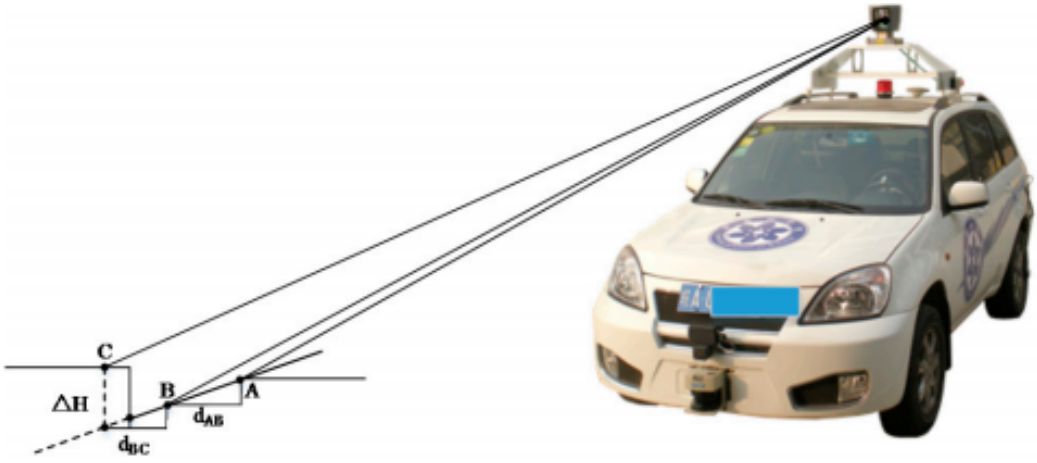


Figure 2.1: Methodology for curb detection with height difference prediction.

Also with a 360° 3D LIDAR, Zhang et al. in [27] use a sliding beam method for road segmentation and a search based method applied to detect the curbs in each frame.

The sliding beam method contains two sections: one bottom layer beam model that allows to find the beam angles, that represent the road direction in the current location of the vehicle in a beam zone inside the region of interest, and a top layer beam model, that allows to build segments of road ahead of the vehicle. After this, spatial features of curbs are defined and extracted, the features' thresholds are defined and the search based method is applied to detect curbs in each segment of the road.

For example figure 2.2, point A corresponds to a curb and point B to the road surface. For each point the following parameters will be calculated:

1. Horizontal distance between two adjacent points (δ_{xy} in Fig. 2.2)
2. Vertical distance between two adjacent (δ_z in Fig. 2.2)
3. Angle between two vectors originating from the same point (v_a and v_b in Fig. 2.2)
4. Elevation distribution in the points of the two vectors.

The algorithm will then check if the parameters are within the thresholds and make the decision if a point belongs or not to a road curb.

The method applied by Zhang et al. proves to have efficient results when compared with similar approaches.

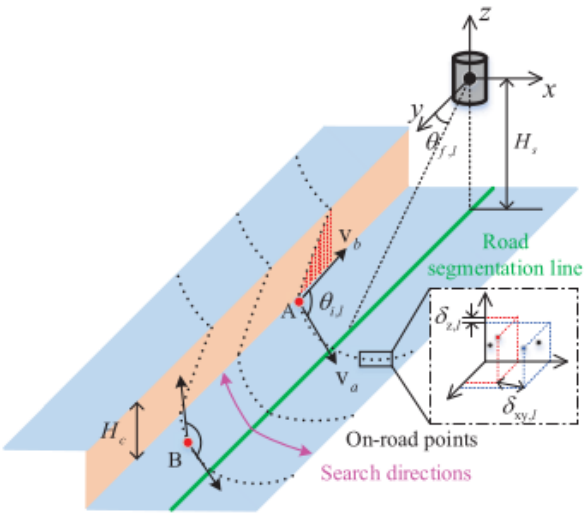
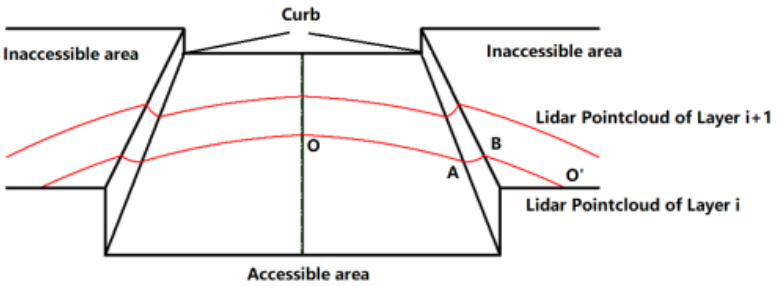
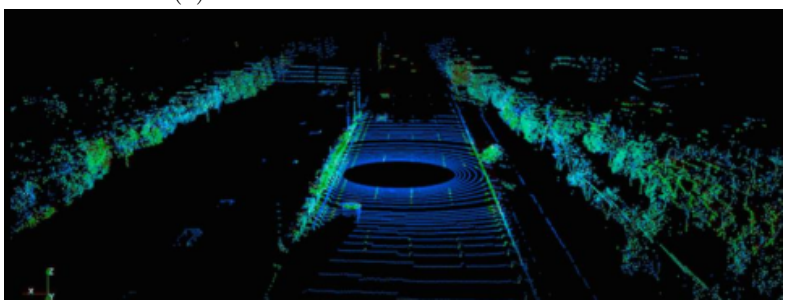


Figure 2.2: Curb scenario description.

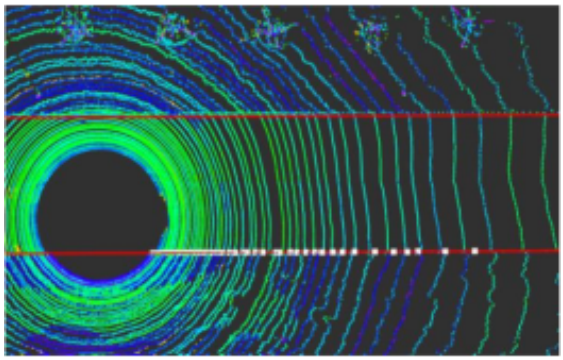
Hu et al. use an HLD-64 SE3 LIDAR with 64 laser layers and 360º range to extract road boundaries using the model shown in Figure 2.3a. First, a sliding pane method is used to extract the road boundary based on two principles: the height difference between adjacent points of same laser layer increases along the road curb and the point cloud curvature in the same laser layer changes significantly along the road curb. After the boundary extraction, Hu et al. apply a `RANSAC` algorithm to conduct straight line fitting on the boundary points collected and a Kalman and Amplitude-Limiting filters to remove abnormal line equations and smooth linear jitter, obtaining the fitting road boundary lines (Figure 2.3c).



(a) Road model used in Hu's method.



(b) Point cloud obtained in in Hu's method.



(c) Line fitting visualization in in Hu's method.

Figure 2.3: Methodology and results used in in Hu's method.

Some approaches, like the method described by Kang et al. in [11], also combines the use of *Hough Transforms* to detect curb with the use of a Kalman filter [10] to track the position of the curb. The existence of a curb is decided based on a probability threshold to accurately estimate the roadside curb position. The decision making algorithm is explained in Figure 2.4 resulting in the decision whether to follow the path or not. This approach presents a powerful tool due to combining LIDAR and advanced algorithms and decision making.

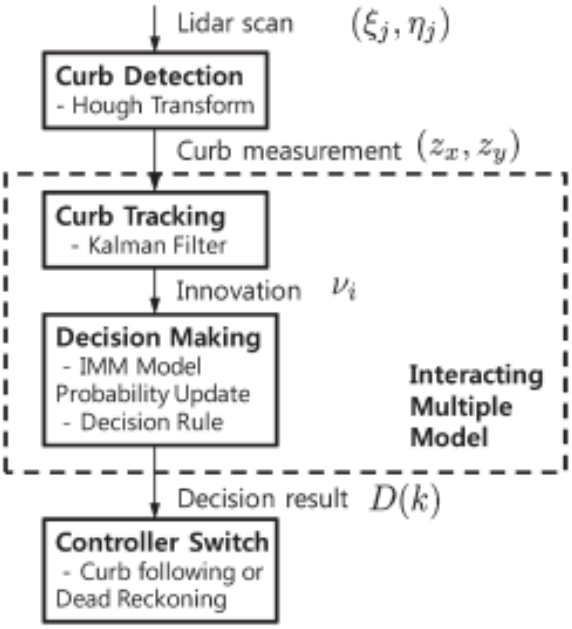
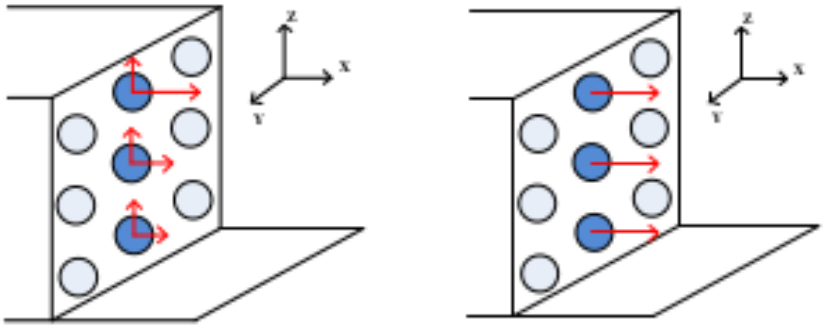


Figure 2.4: Decision making algorithm scheme.

Xu et al. in [24] use a similar approach to the one studied, also based on a density gradient of point clouds. The methodology consists of calculating the difference of density in adjacent voxels in 2D and then adding the 3rd dimension as the difference of elevation between voxels (Figure 2.5).



(a) Elevation difference. (b) Intensity difference.

Figure 2.5: The different definitions of the 3D gradient.

The road classification (Figure 2.6) is then based in three principles:

1. The voxel within the surface, corresponding to only one large gradient.
2. The voxel in the intersection of two surfaces, corresponding to more than one large gradient.

3. The voxel in the intersection of three mutually non-parallel surfaces, corresponding to three large gradients.

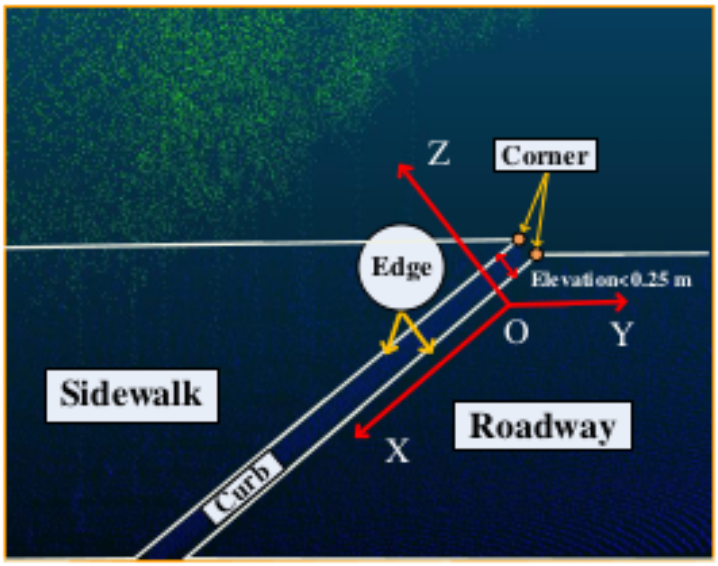
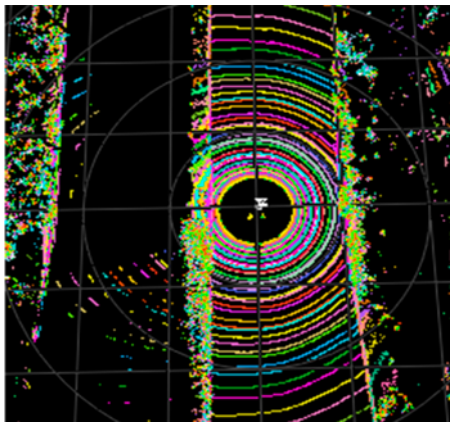


Figure 2.6: Three areas in road point clouds, namely sidewalk, roadway and curb.

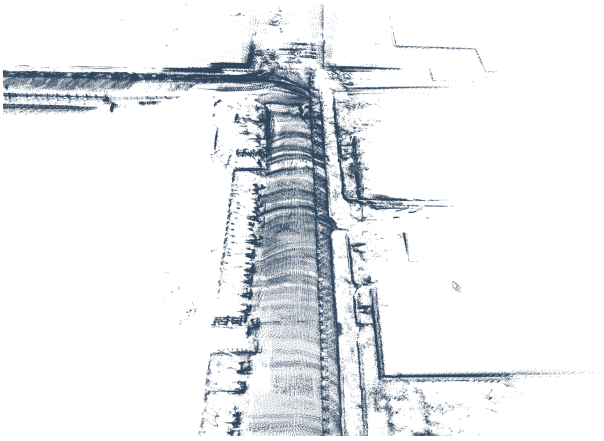
2.1.3 Critical Analysis

In relation to this work, Xu et al.'s approach is the one that comes the closest to the proposed work, by also using point cloud density and gradients. However, Xu's methodology diverges by adding the third component of gradient as elevation differences and follows a completely different path by doing road classification and identifying features as opposed to focusing on road limits.

In addition, all the described methodologies have as a base a 3D 360º LIDAR, offering a completely different perspective (Figure 2.7a), less close to the car and focusing on a wider view with bigger obstacles like tree or people. The LIDAR used in the proposed method (Figure 2.7b) only has a field of view 85º and is placed to contemplate objects close to the car and at ground level, like holes and depressions, road curbs and others.



(a) Point cloud map obtained with a velodyne.



(b) Point cloud map obtained with a SICK-LDMRS in the front of the car.

Figure 2.7: Difference between different a Velodyne and the LIDAR used in this dissertation.

2.2 Ground Truth Gathering

It is fundamental to evaluate the performance of an algorithm and quantify the "correctness" of its results. Usually statistical measures like Precision (Equation 2.2), Specificity (Equation 2.3), NPV (Equation 2.4), Recall (Equation 2.5), F-measure (Equation 2.6) and Accuracy (Equation 2.7) and are used to quantify performance of automotive applications [5]. These evaluation tools are based on relations between true positive (TP), false positive (FP), true negative (TN) and false negative (FN).

$$\text{Precision/PPV} = \frac{TP}{TP + FP} \tag{2.2}$$

$$Specificity/TNR = \frac{TN}{TN + FP} \tag{2.3}$$

$$NPV = \frac{TN}{TN + FN} \tag{2.4}$$

$$\text{Recall/Sensitivity/TPR} = \frac{TP}{TP + FN} \tag{2.5}$$

$$\text{F-measure} = (1 + \beta^2)\frac{\text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{2.6}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.7}$$

To perform this evaluation, it is necessary to obtain the ground truth of the path intended to test.

Some works, like [26] and [14], calculate road boundaries manually frame-by-frame. Although quite easy and direct, this is a time consuming and inefficient process.

Some authors also use databases of labeled curbs that can be found online. An example of this is the evaluation done in [27], yet, this is not possible for every situation and may not be easy to implement the comparison.

In [25], Yang et al. convert the detected boundaries in a Keyhole Markup Language (KML) format and imported into Google Earth. These boundaries are overlapped with Google Earth images and the result is visually checked. To perform a quantitative evaluation, road boundaries are also manually extracted and save onto data sets that correspond to the ground truth.

Chapter 3

Experimental Infrastructure

This chapter describes in detail the different tools used in this dissertation, both in the form of hardware and software. The hardware are the sensors that were used to gather data when the *Atlascar2* is navigating, and its correct the placement in the vehicle. And the software tools that serve as a bridge to gather and analyze information from the sensors and achieve the final output of road limits.

3.1 Hardware

3.1.1 SICK LD-MRS400001

The most important device in this work is the 3D LIDAR sensor. A LIDAR is a device that measures distances to obstacles through the emission of high-frequency laser beams.

In the case of SICK LD-MRS400001 (Figure 3.1), it is a 3D outdoor sensor with the possibility of 4 measuring planes. Its long range scanner allows to measure up to 250m of distance.

In table 3.1 are described as the main features of this sensor [7].

Figure 3.1: 3D LIDAR sensor Sick LD-MRS400001.

FEATURES	
Application	Outdoor
Laser class	1 (IEC 60825-1:2014, EN 60825-1:2014)
Horizontal aperture	85° (4 measuring planes), 110º (2 measuring planes)
Vertical aperture	3.2°
Scanning frequency	12.5 Hz ... 50 Hz
Angular resolution	0.125°, 0.25°, 0.5°
Working range	0.5 m ... 300 m
Scanning range	50 m
Amount of evaluated echoes	3

Table 3.1: Relevant features of Sick LD-MRS400001.

To meet the needs of this work the LIDAR was previously configured to 4 measuring planes with an aperture of 85º and 50Hz of frequency and consequentially an angular resolution on 0.5º since having a high scanning frequency is fundamental and more important than angular resolution.

Figure 3.2 shows a detailed view of the LIDAR placement in the front of the *AtlasCar2*, close to the ground.

Figure 3.2: LIDAR placement in the *AtlasCar2*.

With its current setup, the sensor is in a position that provides a unique point of view of the road. Figure 3.3 illustrates a simulation of the 4 layers of the LIDAR it is clear that the position and orientation of the beams result in a point cloud accumulation focused in the features of the road like curbs, road pins, sidewalks, among others obstacles present in the road.

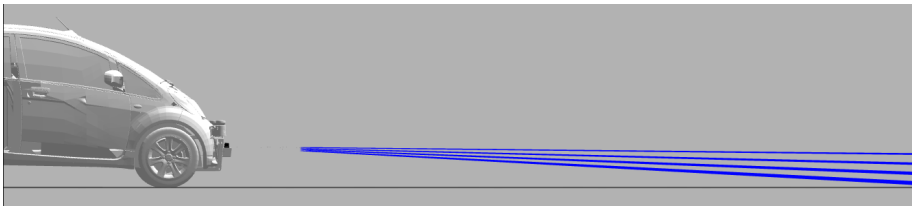


Figure 3.3: Simulation of the 4 layers LIDAR mounted on the car.

3.1.2 Novatel SPAN-IGM-A1 and Novatel GPS-702-GG Dual-Frequency Antenna

The combination of Novatel SPAN-IGM-A1 (Figure 3.4a) and Novatel GPS-702-GG (Figure 3.4b) offer a powerful positioning solution based on Synchronous Position, Attitude and Navigation (SPAN) technology, that takes advantage of both the accuracy of Global Navigation Satellite System (GNSS) and the stability of Inertial Measurement Unit (IMU) resulting in a stable solution, even when the satellite signal is blocked [16].

This combo allows for very precise global positioning and is fundamental to position the car and point clouds in the world.

(a) GNSS inertial navigation system SPAN-IGM-A1.

(b) Dual-Frequency GPS antenna Novatel GPS-702-GG.

Figure 3.4: Inertial Navigation and GNSS.

In the *AtlasCar2* the current setup uses the combination of these sensors and is configured to use Global Positioning System (GPS) signal with a frequency of 20Hz, that goes to the SPAN receiver to be processed along with IMU data, gathered at a frequency of 200Hz. Both sensors are connected to the car through RS232. This setup was configured by P. Nova and is further explained in [15].

3.1.3 AtlasCar2

As mentioned in chapter 1, *AtlasCar2* is a prototype for research on ADAS, so it is equipped with multiple devices to assist its purpose. Besides the 3D LIDAR and the GPS and IMU combo mentioned before, it also has four optoelectronic sensors Sick DT20 Hi (two in visible in figure 3.5 and two in the back of the car), that assist in the inclinometry module. In the car, there is also a PointGrey ZBR2-PGEHD-20S4C and two PointGrey

FL3-GE-28S4-C cameras to perform road perception with artificial vision by identifying road lanes, pedestrians, and other features. The car is also supplied with two SICK LMS151.

The Sick DT20 Hi are currently configured with 2.5ms response time (400Hz) and a measuring range between 200mm and 700mm and are connected to an `arduino` that internally calculates the inclinometry of the car and published the topic `/DadosInclino`.

Figure 3.5 shows the placement of each device in the body of *AtlasCar2* and Figure 3.6 shows the position of the GPS antenna in the body of *AtlasCar2*.

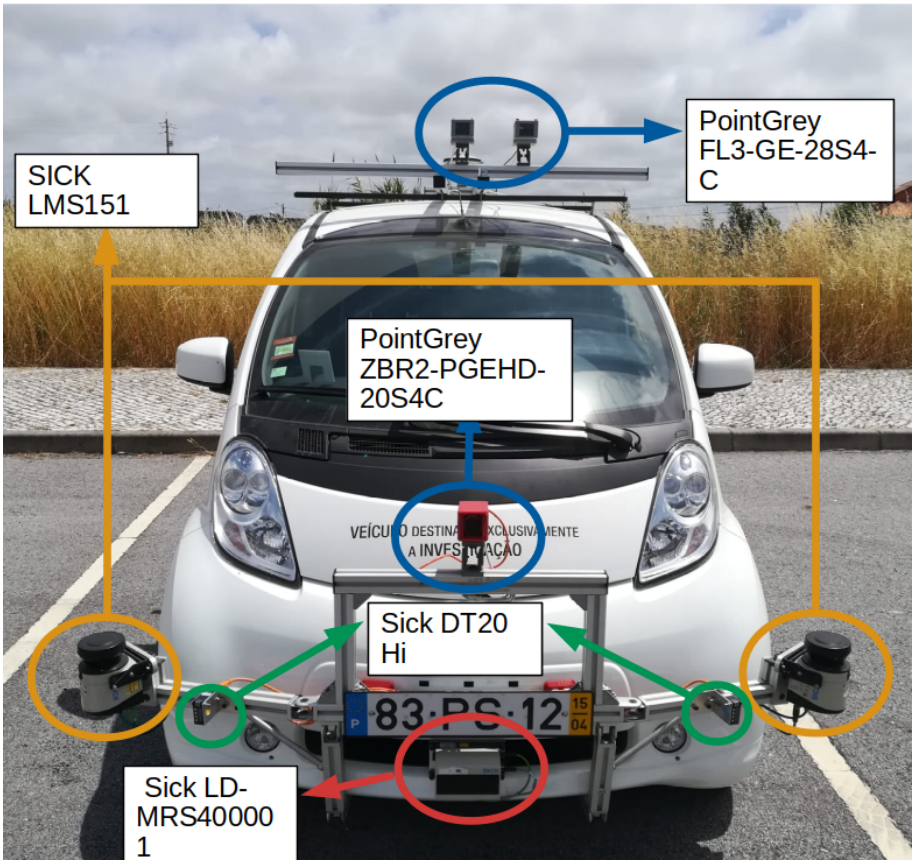


Figure 3.5: Sensor placement in *AtlasCar2*.

Figure 3.6: GPS placement in *AtlasCar2*.

The car is also equipped with an Uninterruptible power supply (UPS) unit that provides about 30 minutes of power to the computer when the car is moving.

3.2 Software

3.2.1 Robot Operating System (ROS)

Responding to the need to have an open source dedicated software for robotics, ROS was created in 2007 at Stanford University. Its main intention is to "encourage collaborative robotics software development" [22], by promoting the development of multiple institutions and for multiple robots, creating a really unique software platform directed for robotics and build by the overall community.

In a simple way, a ROS project is constituted by different modules, `nodes`, that communicate between them through `messages`, defined data structure for each `message` type. This `messages` can be published and subscribed to through `topics`, that link `nodes` between them. `Nodes` are also grouped in `packages`, that can contain several `nodes`, or just one, and are defined by a single `CMakeList.txt` file, where the specifications necessary to compile the package are defined.

Figure 3.7 [21] is shows a diagram that illustrates the functioning of ROS.

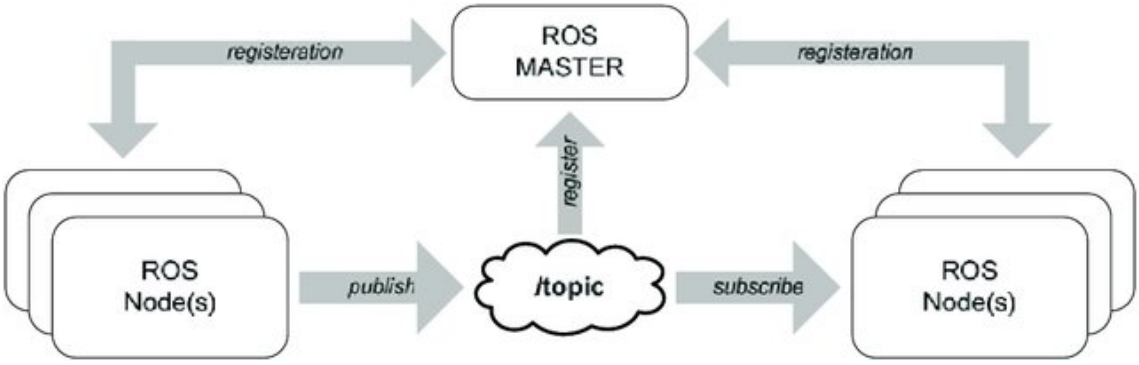


Figure 3.7: Schematics of ROS functioning.

Also very relevant to this work is a set of tools for recording and playing back ROS `topics`, `rosbags`. This tool allows to record the desired `topics`, for example, from real data from sensors, in a `bag` file and replay them from any computer. This is an essential tool to enhance the work progress once it would be unsustainable to perform real testing to each modification and tool developed. With `bag` files, the data from GPS, IMU and LIDAR can be recorded in the real environment onboard of the *AtlasCar2* and the information can be replayed in the same situation as if in real time.

Existing Architecture

As mentioned in chapter 1, the work developed in this dissertation is a continuation of previous work done in the Atlas project.

The existing architecture necessary to this work contained three main packages: `road_reconstruction`, `orientation_module` and `free_space_detection`.

In Figure 3.8 there is a scheme with the relations between the existing `nodes` and `topics` and the communications between them. In resume, belonging to `road_reconstruction` package, there a `node` with the same name. This `node` subscribes to the laser data and assembles a point cloud with the information from the 4 laser layers, also making the accumulation with the car movement with a laser assembler. It also performs the necessary operations to obtain the road limits defined in [12], in the form of a new point cloud where points represent obstacles in the road.

To the `orientation_module` package belongs the `comunic` and `frame_tf_broadcaster` `nodes` present in Figure 3.8. These `nodes` create the necessary frames (Figure 3.9) and transformations between them with the support of a calibration file them and also process the correct the inclinometry of the car, computed by an `arduino` connected to the Sick DT20 Hi.

Finally, `free_space_detection` packages uses `device_frame_publisher` to publish the correct frames for each device in the car.

Very briefly, the `map` frame is placed in the first GPS coordinates received by the program, then there is a transformation to the `ground` frame, clamped to the ground, directly below the GPS antenna. The `car_center` frame is placed on the center of the vehicle to allow roll-pitch-yaw (RPY) calculations, as seen in figure 3.11. The `moving_axis` frame is placed on the front of the car, following the car movement, and finally, `lms151_E` corresponds to the frame of the left SICK LMS151 and serves as a reference to all the

other sensors' frames.

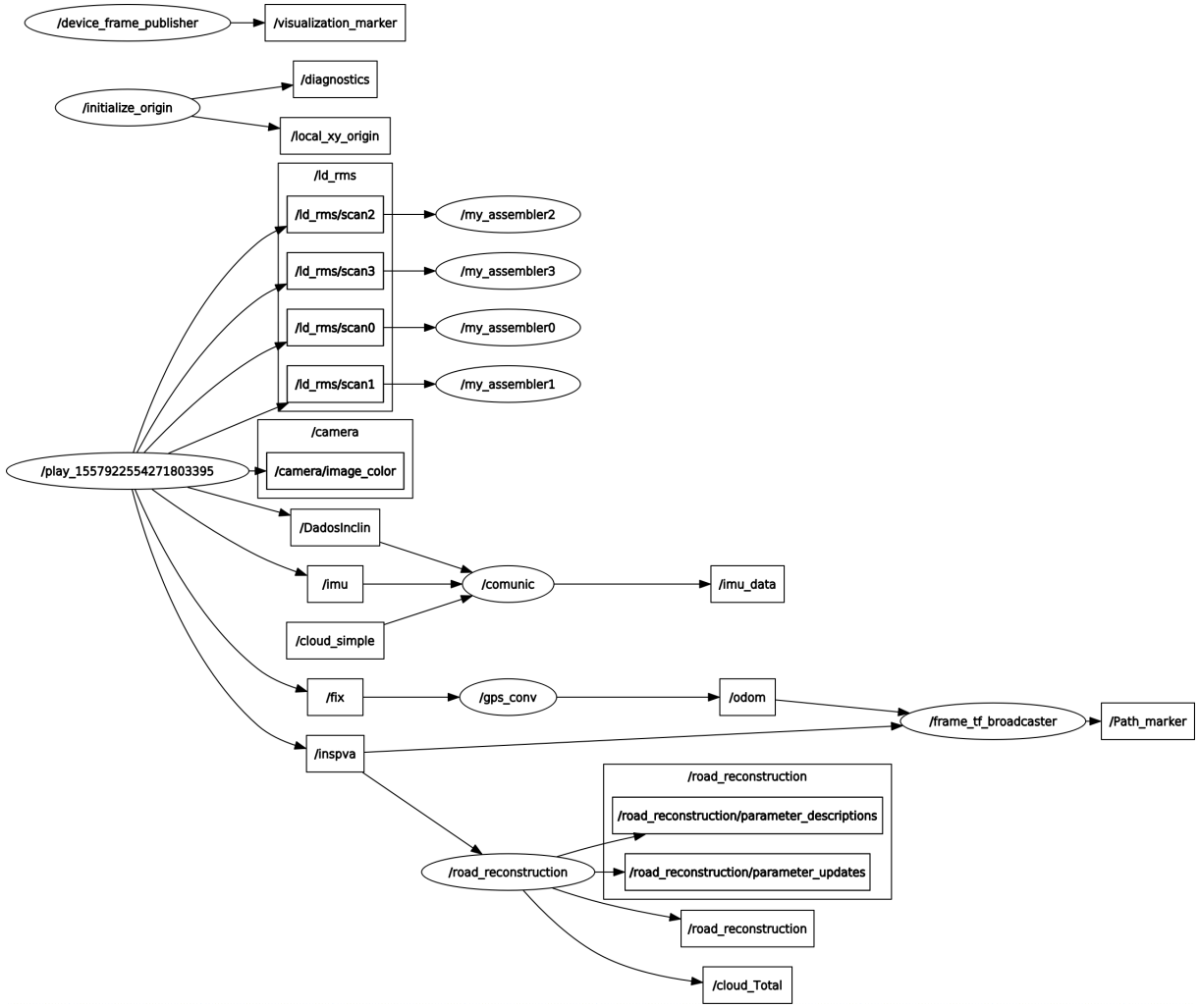


Figure 3.8: Node graph of existing architecture.

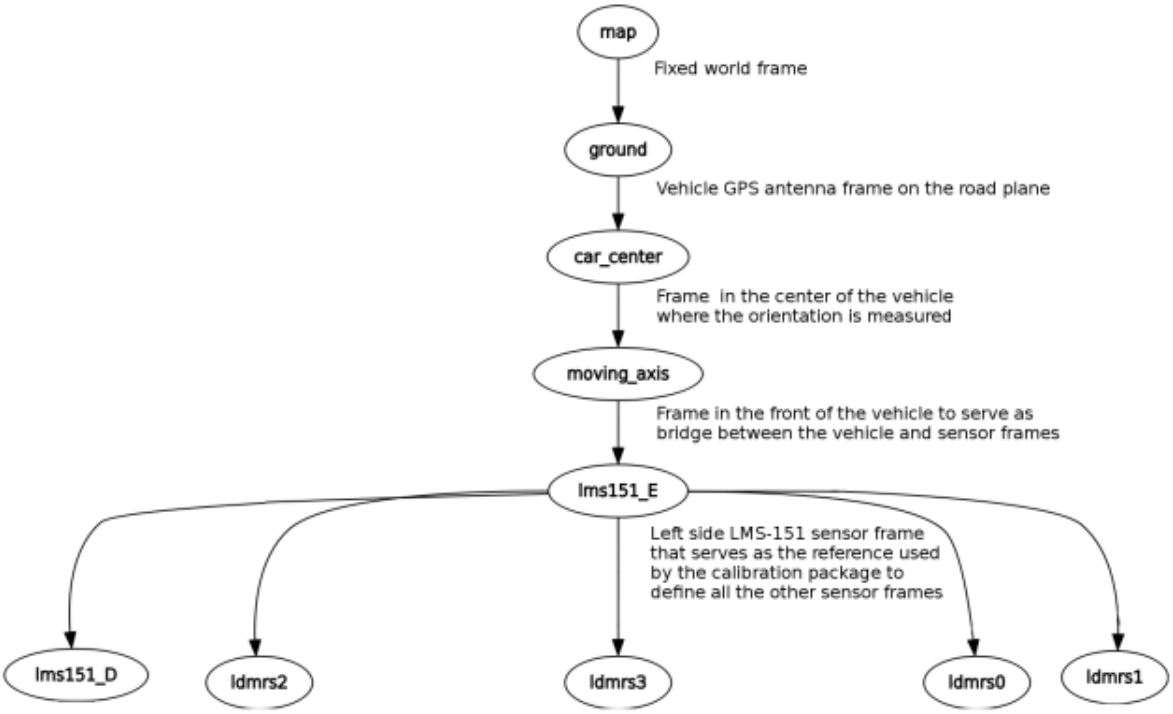


Figure 3.9: Frame tree of *AtlasCar2*.

Figure 3.10 [12] shows a visualization in `rviz` of the most relevant frames described in Figure 3.9 in their correct placement relatively to each other.

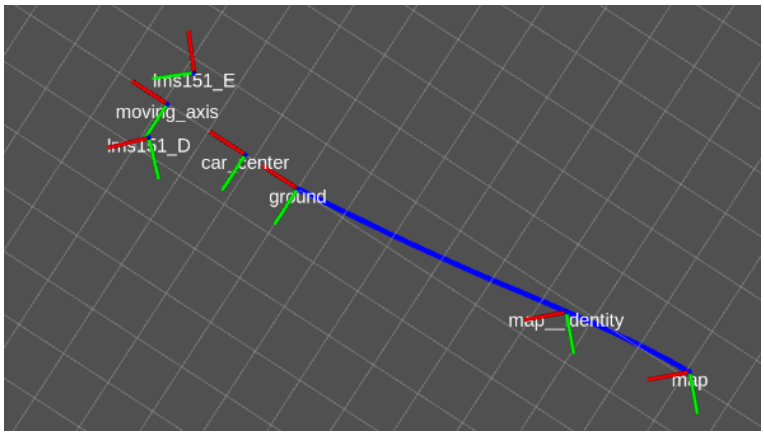


Figure 3.10: Frame tree visualization of *AtlasCar2* with `rviz`.

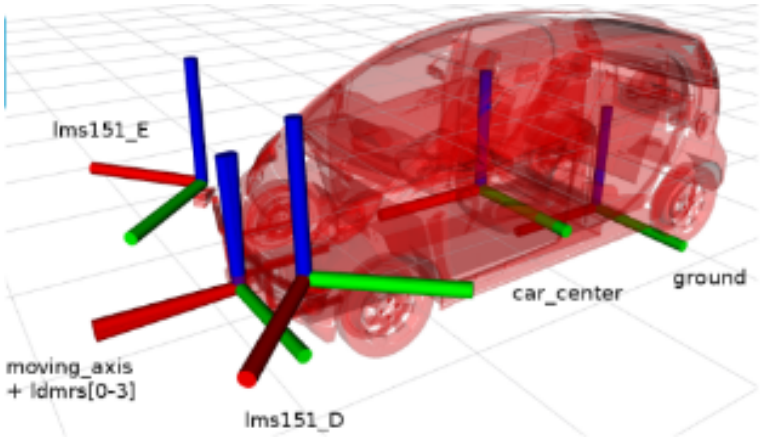


Figure 3.11: Frame tree visualization overlapped with *AtlasCar2*.

Final Architecture

For the development of this work, two new packages there were added, `negative_obstacles` and `eval_api`. Each package, as shown in Figure 3.12, has one `node` with the same name. The first one concerns the identification of road limits as the second one concerns the development of a ground truth and quantitative evaluation of the detected limits and `negative_obstacles` has also a node exclusively to perform the point cloud accumulation and assembling.

The `road_reconstruction` node developed by T. Marques, due to high computational weight processing point clouds, was publishing the accumulated clouds at a much slower rate than desired (0.5 Hz to 1 Hz), considering that the LIDAR is configured to output data at 50 Hz. This slow publishing rate caused problems reconstructing the environment and resulted in a bad accumulation, specially at higher car speeds. To solve that problem, a newly separated `node` was created in the `negative_obstacles` packages to only perform the assembling and accumulation of laser reading. Also, the cloud accumulation buffer was decreased from 150 to 100, thus increasing the frame rate in about 34 times.

Briefly, the `negative_obstacles node` subscribes to the accumulated cloud and performs the operations required for the different edge detections, publishing the correspondent `Occupancy Grid topics` with the results.

One of this `topics` at choice is subscribed by the `eval_api node` to perform the evaluation of that same edge detector. This `node` subscribes to `/inspva topic` to obtain the coordinates of the car at every frame, saves the coordinates in a KML file and also reads the drawn road limits to create a ground truth and perform calculations. In the end of the program the statistical indicators for each frame of evaluation are saved in a CSV file.

The development of this packages will be further explained in chapters 4 and 5, respectively.

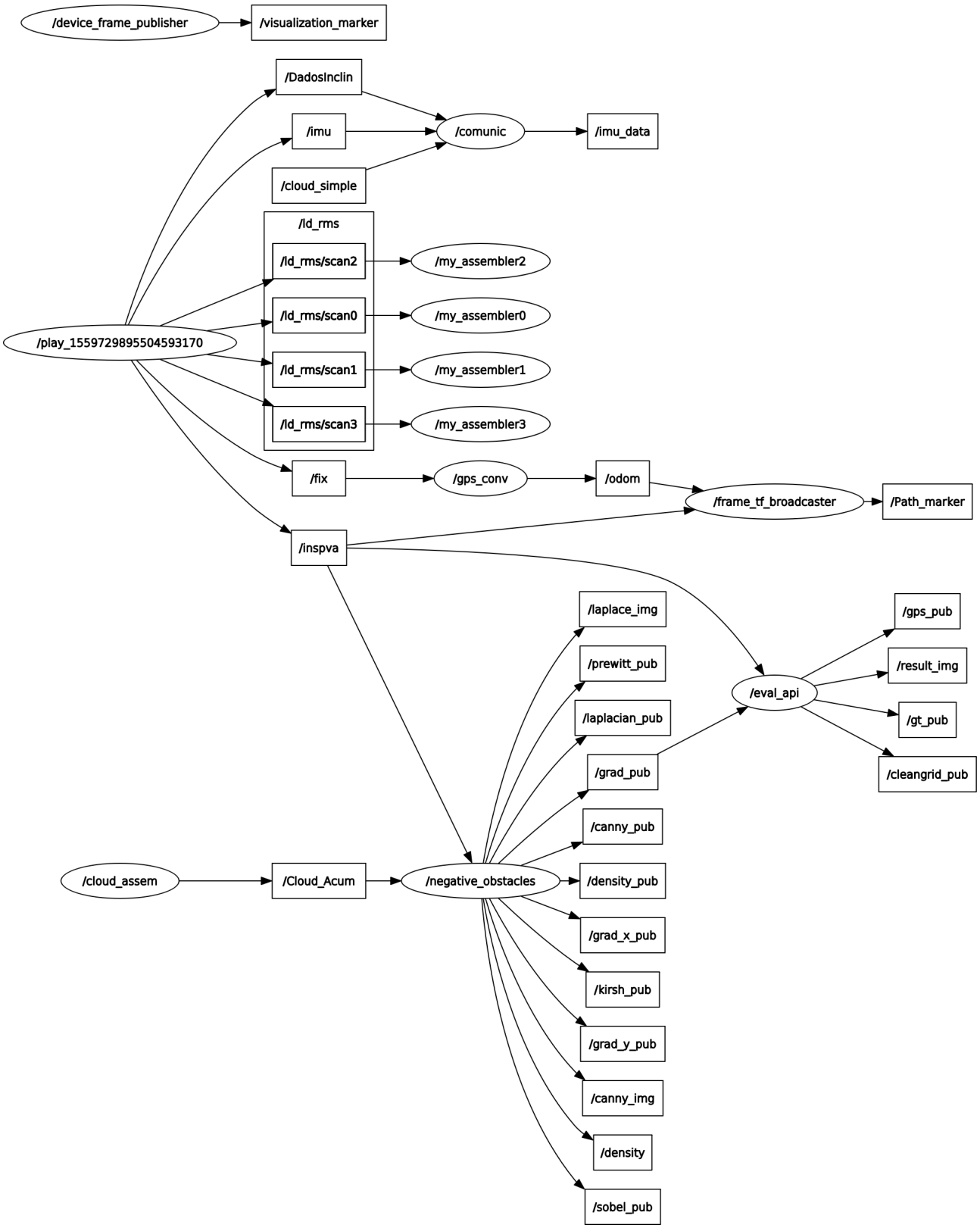


Figure 3.12: Node graph of final architecture.

3.2.2 Other Libraries

PCL

The Point Cloud Library (PCL) is an open-source library providing countless algorithms for 2D and 3D image and point cloud processing (Figure 3.13 [18]). Created in 2010, it is a well known C++ library used around the world mainly in perception and artificial vision in robotics. PCL's main advantage is the possibility to treat and analyze data from LIDAR and similar sensors like kinetics and grouping sensor data in point clouds, data structures that represent multi-dimensional points [18].

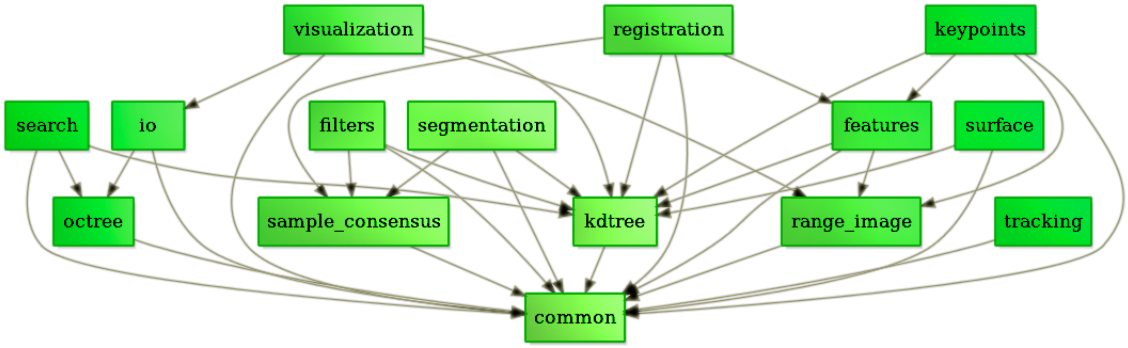


Figure 3.13: PCL tree of libraries.

PCL can be integrated with ROS in a package named `pcl_ros` that allows to perform operations within ROS environment like transformations between different frames.

In the line of this work, PCL is a fundamental tool to gather and compute data from the laser sensors. Figure 3.14 shows a clear example of this in data gathered in a path around the University of Aveiro. This image was obtained with the accumulation of laser readings through the complete route around the University.

Figure 3.14: Example of laser readings accumulation using PCL.

OpenCV

Open Source Computer Vision Library (OpenCV) is an open source library built for computer vision and machine learning. With interfaces for C++, Matlab and Python and supporting Linux, Windows, MacOS, and Android, is one of the most universal and complete libraries for computer vision. It has more than 2500 algorithms, performing recognition, identification, classification, tracking, etc [17].

OpenCV is a powerful tool for image processing and has interesting features in the integration of grid maps with images, allowing conversions from grid maps to images and vice versa. These features were of extreme importance, giving the possibility to apply complex edge detection filters (Figure 3.15) available in OpenCV libraries in density grids collected from laser data.

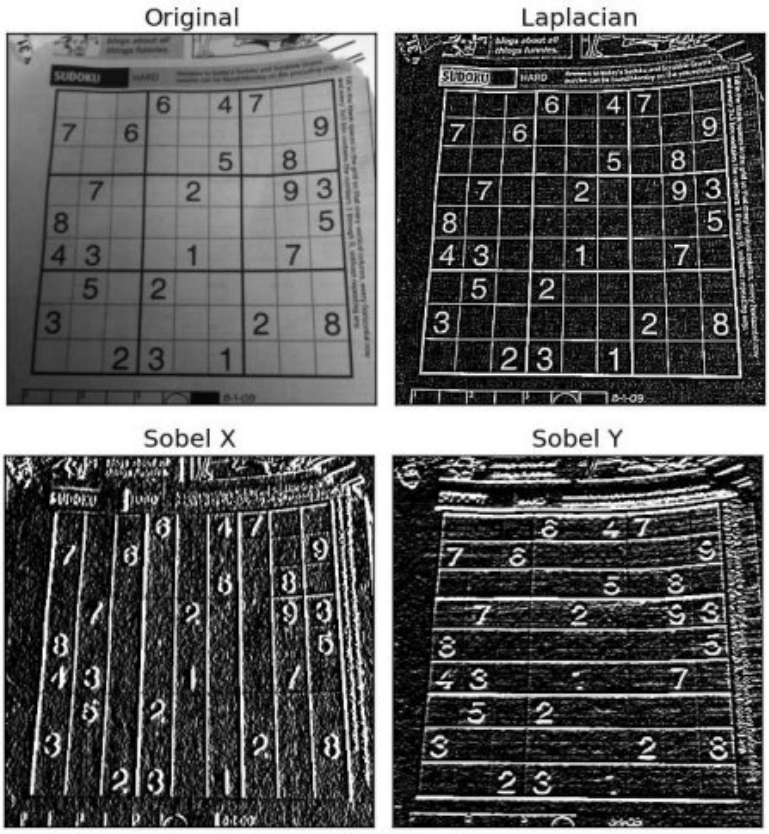


Figure 3.15: Example of image processing toolbox in edge detection.

Chapter 4

Detection of Road Limits

This dissertation's main goal was defined since the begin as the establishment of the navigable road limits. Logically, there are endless different approaches that can lead to road limits detection, but through the years, in the Atlas Project, the accumulation of laser readings (i.e. point clouds) with the car movement proved to be an interesting and less studied approach [23]. This accumulation, further studied by T. Marques [12], allows instant decision making, identifying road obstacles on time.

The definition of a new and effective methodology for road limits detection is the core of this dissertation. As the title suggests, the idea was to take advantage of point cloud density to detect, not only positive obstacles, but also negative, as depressions, and inverted curbs.

Although other authors, as mentioned before in Chapter 2, have also applied gradient filters, the method proposed in this dissertation takes it a step further by applying edge detection techniques to point clouds.

4.1 Approach

Taking advantage of the car movement to create a point cloud accumulator and mapping the road as the car moves by placing the point clouds in the world frame resorting to GPS data is the core idea of this work. The assumption is to use this mapped point cloud to define a rule that allows detecting the road limits in real time.

In his approach, T. Marques limited the algorithm to eliminate points in low-density zones, under the assumption that road limits are defined as high-density zones [12]. Although presenting good results in general, this method failed in identifying negative obstacles, such as inverted curbs, that are exactly defined as "shadow" zones with zero point cloud density (Figure 4.1 [14]). So, the challenge is how to identify empty zones as well as high-density zones.

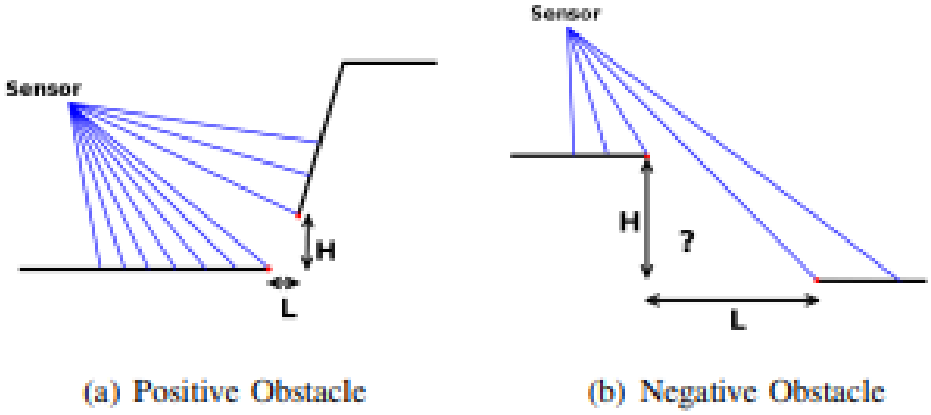


Figure 4.1: Positive vs negative obstacle.

Since the beginning, the density gradient seemed the right direction to go, as it contemplates both positive and negative changes in density. Initially, the definition of the medium road plane through PCL's `ransac` techniques was considered but soon it became clear that simply eliminating the z coordinate of each point and obtaining a 2D flattened point cloud in the XY plane was enough.

As the work evolved, the idea became to create a two-dimensional grid attached to the car frame: first, a density grid with data corresponding to the number of points in each predefined square, and then grids with the results of density gradient and other different edge detection techniques.

As grids follow the car movement, real-time visualization of the road limits became possible.

4.2 Development of a Density Grid

The concept of an occupancy grid is to represent a map of the environment as an evenly spaced grid with each cell representing the probability of the presence of an obstacle at that location in the environment.

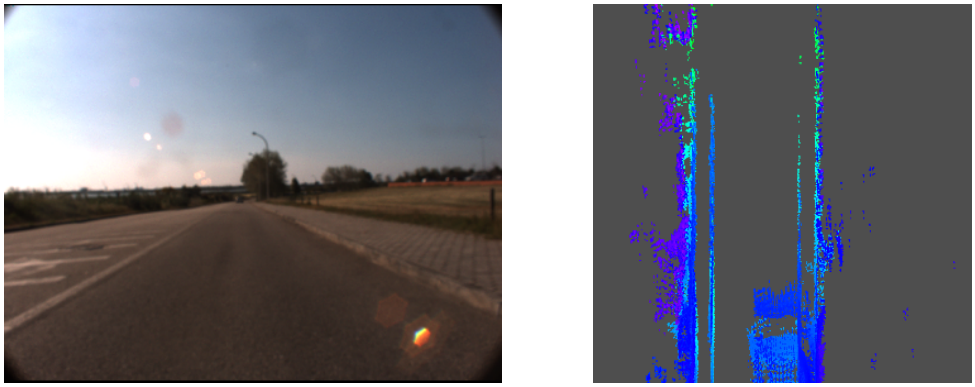
In ROS environment, occupancy grids are a part of navigation messages, *nav_msgs*, and are defined by:

- A *header*, containing the frame id and time stamp
- An *info*, containing the resolution, height and width of the grid and its position relative to frame id
- Cell *data*, a raw major vector containing the data relative to each grid cell.

The main idea was to convert the flattened point cloud (Fig. 4.2b) into a density grid (Fig. 4.3) with the density of the point cloud in each grid cell, allowing to perform more complex operations with a significantly lower computational effort. The density grid was built under the following principles:

1. The density inside each cell is equal to the number of points within the coordinates of that cell.
2. The altitude component is not relevant, thus only x and y components were considered.
3. The grid base frame is `moving_axis` (Figure 3.9 for further understanding) and consequentially the point cloud must be transformed from the `map` frame to the target frame.
4. Once there was no advantage in considering the information behind the car front, the grid was defined with 40 m ahead of the car and 20 m to each side of the car, making a grid with a final size of 40×40m.

Note that, as shown in Figure 4.3, the x-axis is along with the car movement and y-axis to the side of the car.



(a) Camera view. (b) Point cloud of the road.

Figure 4.2: Camera image and correspondent accumulated point cloud.

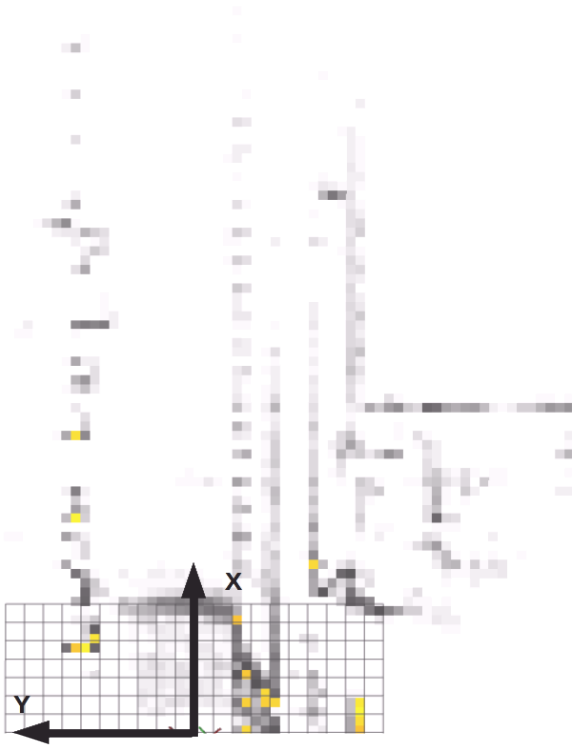
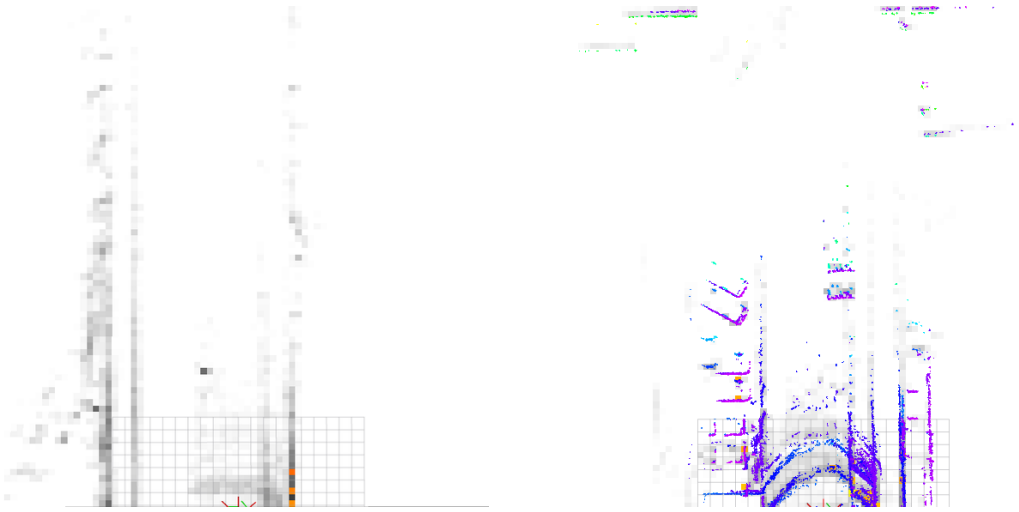


Figure 4.3: *moving_ axis* frame orientation and positioning.

Although the influence of grid resolution in the detection of road limits is discussed further on (Section 6.2.3), the base resolution is considered 0.4 m/cell. Also, Fig. 4.2a shows a camera image from the camera installed in the car in the same frame of the point cloud and density grid visible in Fig 4.2b. In this image it is possible to visualize the real environment and the road curbs detected.

Figure 4.4a shows a density grid in a straight road situation. Darker zones represent higher density while lighter zones represent lower density, with white as zero density. In Figure 4.4b there is a representation of the density grid overlapped with the point cloud that originated it.



(a) Density grid.

(b) Density grid overlapped with the correspondent point cloud.

Figure 4.4: Density grid visualization.

To develop this grid, the `negative_obstacles` package was created with its corresponding *node*. In this *node*, the accumulated point cloud `topic`, created by a cloud accumulator, is subscribed and with `pcl_ros` it is transformed to the `moving_axis` frame. Then, for each point of the point cloud, the cell coordinates (line and column) to which it belongs are calculated. To convert the matrix coordinates to a row-major vector indexes, the equation $index = line + column \times number \quad of \quad columns$ is used.

A `nav_msgs::OccupancyGrid` according to the described proceedings was created with the density information and published in the correspondent `topic`. With ROS visualization tools, `rviz` or `mapviz`, the grid can be visualized along with the car movement.

In order to standardize the density values in each frame and minimize the impact of car velocity and variations in the RPY values, the density occupancy grid values are normalized to the maximum density value in each frame, returning values from 0 to 100.

4.3 Density Gradient

As an answer to the question of how to identify negative obstacles in the accumulated point clouds, the use of gradients appears as a tool to identify both positive and negative changes in density, that define, respectively, positive and negative obstacles.

These gradients are applied in the form of bi dimensional filters. For each pixel location (x,y) in the image or grid, its neighborhood is considered and used to compute the response as a weighted sum of pixel values. The computed response is stored in a new image or grid at the same location pixel coordinate.

Logically, the more complex the filter, the more difficult the implementation and computational effort, and more complex filters were out of reach in computational complexity of manual implementation. The solution was to use `GridMapRosConverter` package to easily convert the density grid in an OpenCV image and open a new set of edge detection tools in the image processing toolbox. This conversion has two steps:

convert the occupancy grid into a `GridMap`, and convert the grid map to a `cv::Mat`, an OpenCV image, with a `CV_8UC1` encoding (8-bit unsigned type with one channel).

To this image, the 2D filters are applied with algorithms in the OpenCV image processing libraries and converted back into new occupancy grids through the same process.

After applying each edge detection operator, the images must be converted to absolute values since both positive and negative gradients are relevant.

A big advantage in this process is the possibility to apply a custom threshold when converting the image back to a grid and eliminate low-interest regions in the edge detection process.

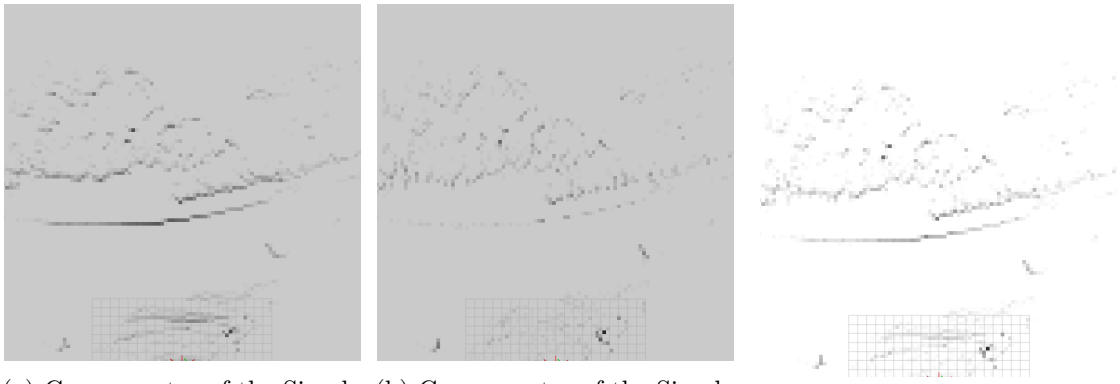
So, with the density occupancy grid defined, the next step was to create new grids with the horizontal and vertical gradient, and gradient magnitude. The grids were created separately in order to evaluate the most efficient in this particular case.

A grid with the gradient direction was also developed but didn't lead to any significant result and it was discarded.

To calculate the horizontal (G_y) and vertical (G_x) gradients, the filters used were, respectively, $\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

Figure 4.5 represents the behavior of each Gradient component in the presence of a frontal obstacle. As expected, G_x is the main responsible for the identification of these obstacles as there is no density change to the sides. Antagonistically, in Figure 4.6, G_y identifies the side curbs and G_x doesn't detect significant modifications. Figures 4.5c and 4.6c show the gradient magnitude, calculated by equation 4.1, where the approximation is implemented to decrease computational effort. As expected, the Gradient magnitude performs well in both situations and is the most appropriate to identify all types of objects.

$$\left\|\vec{G}\right\| = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y| \tag{4.1}$$

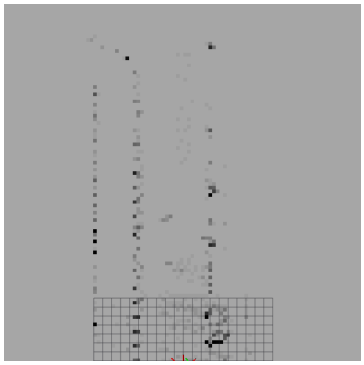


(a) Component x of the Simple Gradient.

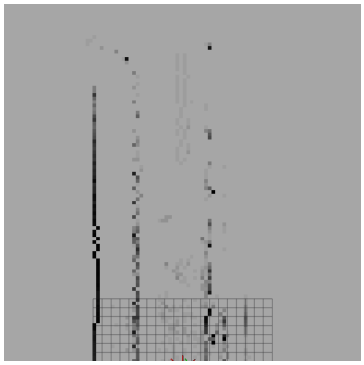
(b) Component y of the Simple Gradient.

(c) Gradient magnitude

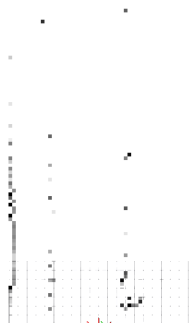
Figure 4.5: Simple Gradient grids in a frontal obstacle situation.



(a) Component x of gradient.



(b) Component y of gradient.



(c) Gradient magnitude

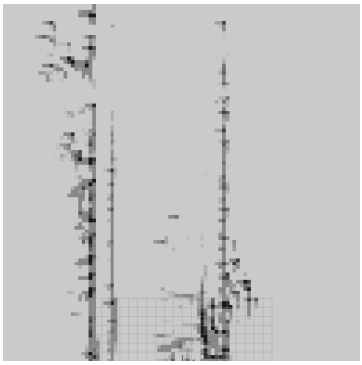
Figure 4.6: Gradient grids in a curve situation.

4.4 Edge Detection Algorithms

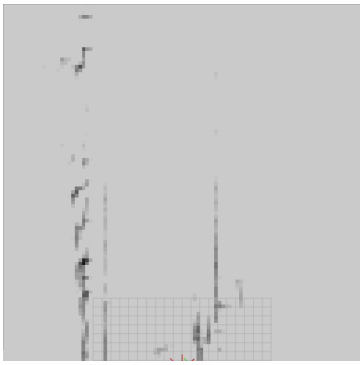
After calculating the Simple Gradient, the transition to more complex filters like *Prewitt* and *Kirsch* filters' was the natural way to go.

A *Prewitt* filter is defined by the following operators: $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ and $G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$. The magnitude of the filter in each pixel, in this particular case, cell, is then again defined by Equation 4.1. In Figure 4.7b

Although more sensible to noise *Kirsch* edge detector was calculated through a similar process and is defined by $F_y = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix}$ and $F_x = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix}$, considering only 2 of the 8 possible directions of this operator. Figure 4.7a shows the result of edge detection using *Kirsch*.



(a) *Kirsch* gradient magnitude grid.



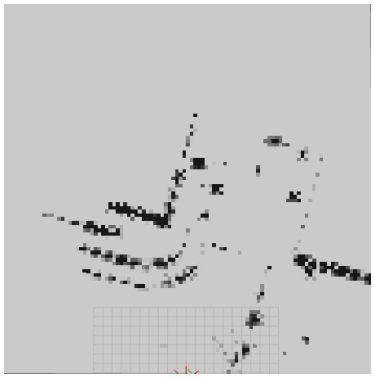
(b) *Prewitt* gradient magnitude grid.

Figure 4.7: Edge detection grids visualization.

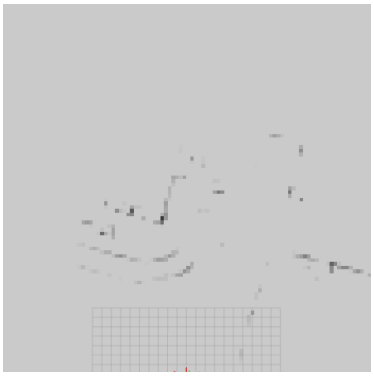
When using more complex edge operators already predefined in the OpenCV image

processing toolbox, as *Sobel*, *Canny* and *Laplace*, in general three parameters need to be given as input: the kernel size, odd number that defines the size of the filter, set to 3, the scale, if necessary to scale the output, and *Delta*, an optional value that is added to the results prior to storing them in a matrix, both as 0.

Figure 4.8 and Figure 4.9 show the performance of algorithms in both a curved and a straight road situation. Visually, the *Laplacian* operator seems to have the most well defined and smoothest detection and works well in both situations. The results obtained with *Canny* are good but the algorithm is very sensible and doesn't result well in every situation.



(a) *Laplacian* filter.

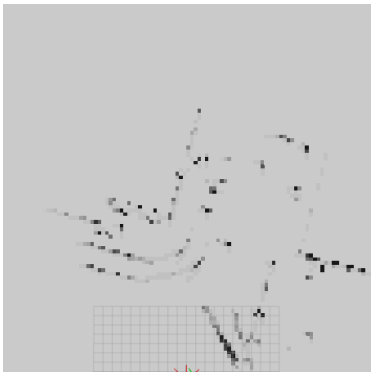


(b) *Prewitt* filter.

(c) *Sobel* operator.

(d) *Canny* filter.

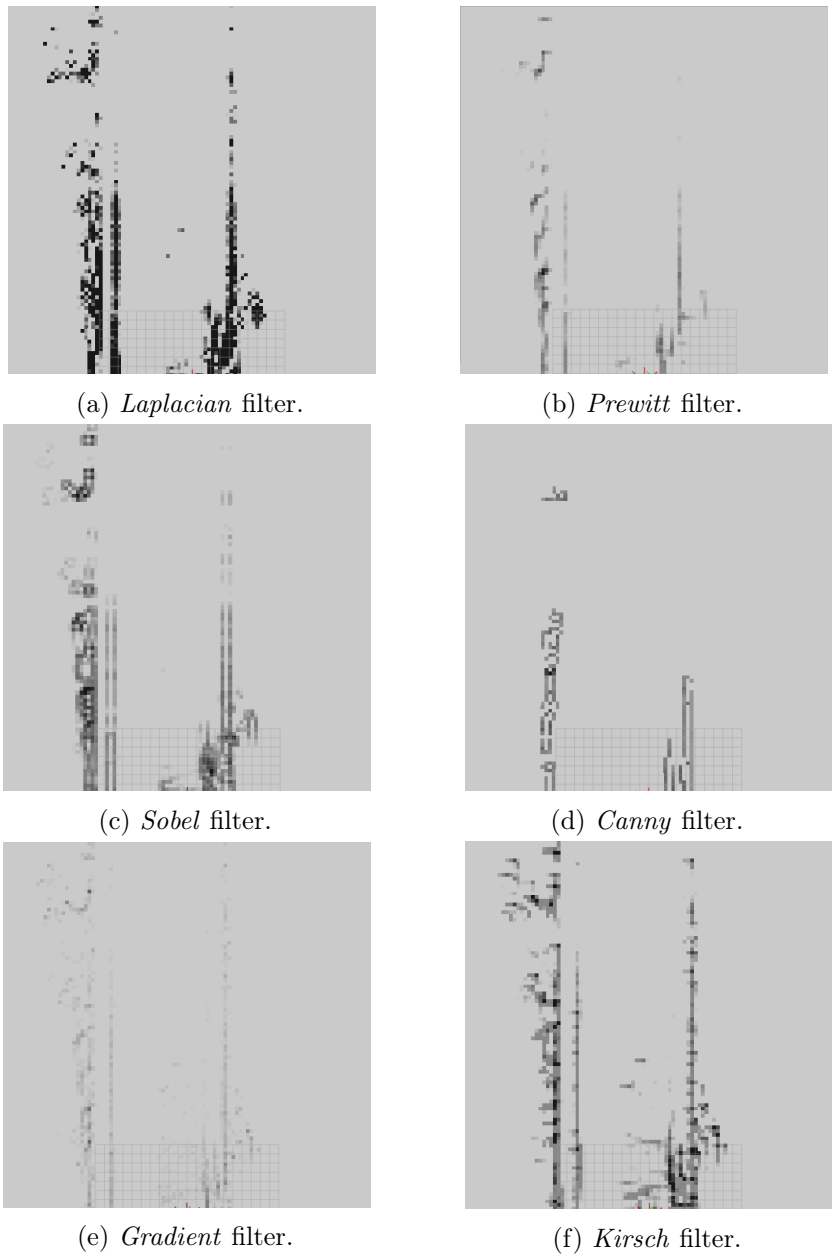
(e) *Gradient* operator.



(f) *Kirsch* filter.

Figure 4.8: Edge detection grids in a curve situation.

In straight road situations as shown in Figure 4.9, the *Laplacian* operator presents again more complete and fine results, namely on the right side of the road, where the other algorithms fail. Comparing it with other detection mentioned above, *Kirsch* filters detect obstacles with poor precision, and Gradient filter identifies fewer obstacles than the rest as only x and y directions are considered in contrast to other filters.



(a) *Laplacian* filter. (b) *Prewitt* filter.

(c) *Sobel* filter. (d) *Canny* filter.

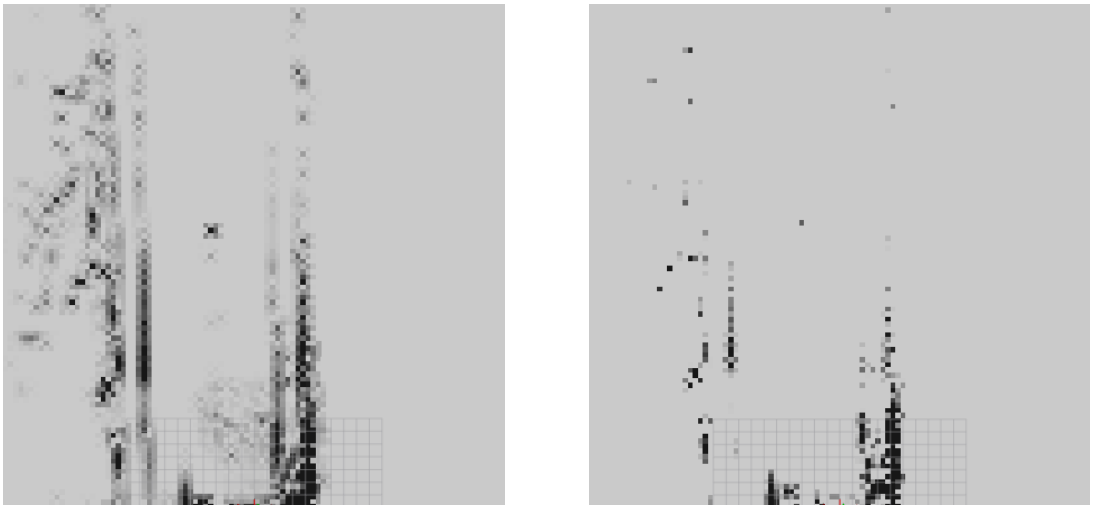
(e) *Gradient* filter. (f) *Kirsch* filter.

Figure 4.9: Edge detection grids in a straight road situation.

As expected, edge filters detect a lot of useless information and the most prominent features are detected in the form of high cell magnitudes. To filter the information, applying a threshold becomes in some cases necessary so that grids only show the relevant density variations, especially on more sensible filters. The influence of threshold application

varies from filter to filter, once some filters are susceptible to different noise levels. The application of this threshold is possible when converting the OpenCV images back to Occupancy Grids, that allows to only convert from a certain user-defined value. The study of the influence of the threshold application and the optimization of the same will be studied later on in chapter 6.

Figure 4.10 shows the example of a *Laplace* grid before and after filtering. As can be seen, in Figure 4.10a has a lot of noise areas due to the filter being prone to noise and in Figure 4.10b that same noise is eliminated due to only considering cell values above 120, in this particular case.



(a) Raw *Laplace* grid.

(b) *Laplace* grid after the application of a threshold of 120 in 255.

Figure 4.10: Edge detection grids visualization.

4.5 Closest Limits to the Car

In reality, the purpose here is to find the navigable road limits, thus objects detected beyond the first obstacle at the right and left of the car are irrelevant. With that in mind, a function was developed to clean the detected edges to contemplate only the closest limits to the car (Figure 4.11).

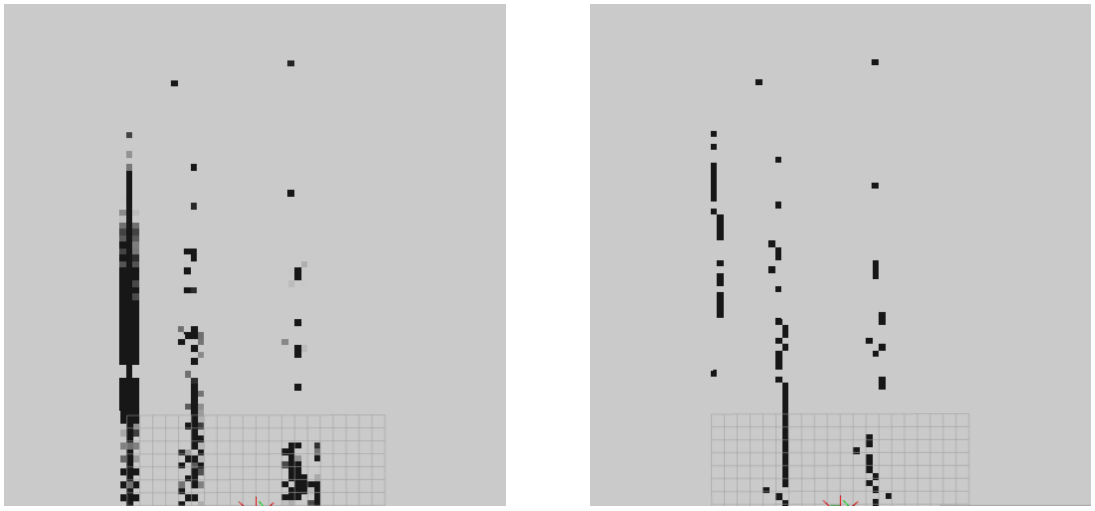
Knowing the center of the `moving_axis` frame (Figure 4.3), a matrix of zeros was created and for each line, only the first detected cells to the right and to the left of the car were passed from the considered grid's data.

Although the goal would be to obtain a full complete line corresponding to road curbs or similar features defining the road limits, that is unlikely to happen, as, the further the distance to the car, the more scattered laser readings become, and consequently there is a loss of information and lower objects may not be detected. Taking the example represented in Figure 4.11, close to the car, road curbs are more or less well identified, but approximately halfway through the grid, they start disappearing and the first detected object becomes a building at the left side of the road limits.

Despite this problem, as the point cloud accumulation is constantly happening as 50Hz,

the girds are updating at this rate and the limits are well defined at approximately 10 m to 30 m ahead of the car, this distance is enough to make instant decisions of trajectory and the badly defined limits will be defined seconds later, which makes this problem less relevant.

In addition to removing unnecessary information, this cleaned grid also allows a better quantitative evaluation (see Chapter 5).



(a) Raw grid. (b) Cleaned grid.

Figure 4.11: Example for *Laplacian* operator.

Intentionally blank page.

Chapter 5

Development of a Ground Truth Application

To evaluate the performance of an algorithm it is necessary to compare its results with ground truth correspondent to the reality. In this case, to compare the real road limits to the ones found with the described approach, a ROS `package` was created to perform a quick and automated approach to an evaluation method.

5.1 Approach

Although the initial idea was to develop a GTK application to manually draw road limits from maps and camera images frame-by-frame, this would be a time-consuming process because road limits would have to be drawn individually for each frame and the real limits would only rely on the human eye to select each cell correspondent to an obstacle. Taking that into consideration, the idea evolved to using KML files to draw the total limits on Google Earth and create a ROS package to read those files, extract the coordinate points and convert them to the correct frame, allowing to convert those limits into an Occupancy Grid, by a similar process to the one applied to point clouds.

KML files, developed by Google and maintained by Open Geospatial Consortium (OGC), are used to display geographic data in an Earth browser such as Google Earth. KML is an XML language focused on geographic visualization, including annotation of maps and images. The geographic visualization includes not only the presentation of graphical data on the globe but also the control of the user's navigation in the sense of where to go and where to look [1].

With this work in mind, one of KML the main features is the possibility to draw lines or polygons with latitude and longitude coordinates, as shown in Figure 5.1. This can be read by the program, that then compares those coordinates with the car coordinates obtain from the GPS satellite and IMU combo.

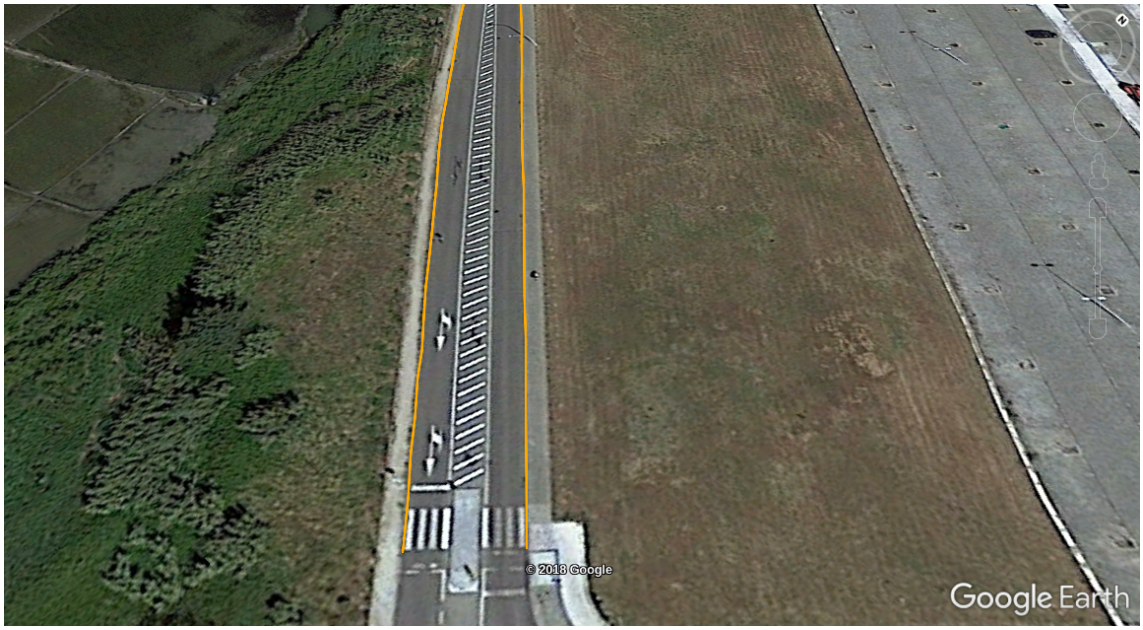


Figure 5.1: Example of road limits marking using KML file with orange lines representing road limits.

In line with this work, navigable space is defined as the space within road limits, where the car can, allegedly, navigate with safety.

Although this method relies on the quality of GPS data and has several error sources as human error drawing limits or a poor GPS calibration, the main objective is to develop a metric to, not only compare edge detection algorithms, but to easily evaluate the **navigable road limits** detection in standard situations with no unusual obstacles. It should be noted that, for the purpose of ground truth testing, the main purpose of this work is to find navigable road limits and not to detect temporary obstacles or similar barriers. With this in mind, a cell will be considered a positive if it belongs to the navigable space and a negative if it is outside the navigable space.

5.2 Integration with Google Earth

To facilitate the process of marking the ground truth, the first step was to automatically save the car coordinates for each frame in a KML file, allowing to visualize the car path in the Google Earth application (Figure 5.3). With this path, knowing the exact correct place to mark road limits becomes easier.

To this end, the GPS topics must be subscribed to obtain the car latitude and longitude in each frame. At the beginning of the program, a file is created with the correct XML headers. To that file, latitude and longitude are constantly added, with zero altitude, and each point is separated by commas. At the end of the program, the file handler is closed and the file is saved becomes readable by Google Earth. The process is explained in Figure 5.2.

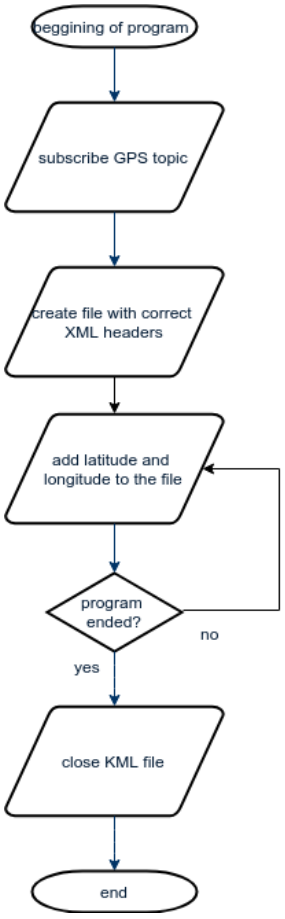


Figure 5.2: Diagram of coordinates acquisition process.

In the following code, an example of a KML file with one point is presented, with the correct headers. In Google Earth, lines can be custom to have different colors and width but, in this context, these parameters are not relevant. To draw a path (i.e. a line), the `styleURL` field must be `pathstyle`. The coordinates of the path's points are between the tab `<coordinates>` `</coordinates>` and latitude, longitude and altitude are separated by commas and each point separated by spaces. In this case, altitude is ignored and always set to zero, assuming points are clamped to the ground. Figure 5.3 shows a Google Earth image of a path around University of Aveiro in the form of a KML.

```
<?xml version='1.0' encoding='utf-8'?>
<kml xmlns='http://www.opengis.net/kml/2.2'>
<Placemark>
<description>Path to evaluate road limits precision</description>
<styleUrl>#pathstyle</styleUrl>
<LineString>
<tessellate>1</tessellate>
<coordinates>-8.6328479884000000055,40.633761692699998491,0</coordinates>
</LineString>
</Placemark>
</kml>
```

Figure 5.3: Example of the car path around University of Aveiro using a KML to upload the car coordinates to Google Earth.

Unless the car path is predefined before driving and the ground truth pre-marked, this evaluation is only possible when using of `rosbags`. In this work, `rosbags` are used to record laser and GPS data in the form of published `topics`, allowing to play back the information to later compare it with the later drawn ground truth.

The process of reading the road limits in the form of ground truth is more complex. This process involves drawing the data corresponding to the right and left side of the road. The program opens the correspondent files and identifies the coordinates within the tab `<coordinates> </coordinates>`. Each point is separated by commas, so a string vector with each point string is created and, as each coordinate is separated by spaces, the latitude and longitude are separated and place in `double` type vectors.

5.3 Coordinates Conversion

The coordinates placed in the World Geodetic System (WGS 84) frame need to be transformed to the correct car frame, `moving_axis`. This requires the following steps:

1. Convert the car latitude and longitude to the Universal Transverse Mercator (UTM) frame.
2. Convert every road limit point to the UTM frame.
3. With both points in a metric scale, calculate the difference between each point coordinates and the car coordinates.
4. Rotate the obtained coordinate to the `moving_axis` orientation by performing a z rotation correspondent to the yaw (azimuth in the GPS message) (Equation 5.1).
5. Add 2.925 m to the x coordinate of each point, correspondent to the translation from the `ground` frame (below the GPS antenna) to the `moving_axis` frame (see Figure 3.9 and 3.10 for better comprehension)

$$\begin{bmatrix} x_{correct} \\ y_{correct} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(yaw) & \sin(yaw) & 0 \\ -\sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{utm\_point} - x_{utm\_car} \\ y_{utm\_point} - y_{utm\_car} \\ 1 \end{bmatrix} \tag{5.1}$$

To create a continuous line, an interpolation function was implemented to create new points within the lines, assuming a straight line between each one.

5.4 Ground Truth Visualization

The points go through the same process of transformation into an `Occupancy Grid` as point clouds, scanning each point and checking if its coordinates are inside the grid's limits. As the coordinates are on the `moving_axis` frame, they will only appear in the grid if they are 40 m ahead of the car, thus, a variable was created to detect when the ground truth is in the grid's limits and initiate the statistical measures referred in 2.2.

Figure 5.4 shows a Google Earth image of the car path, in red, with the correspondent ground truth, in green. In the image it is visible that the combination of GPS and IMU provides very accurate data, even detecting lane changing. This accuracy is fundamental for this evaluation method to work bearing in mind that GPS data is the core of the developed method.

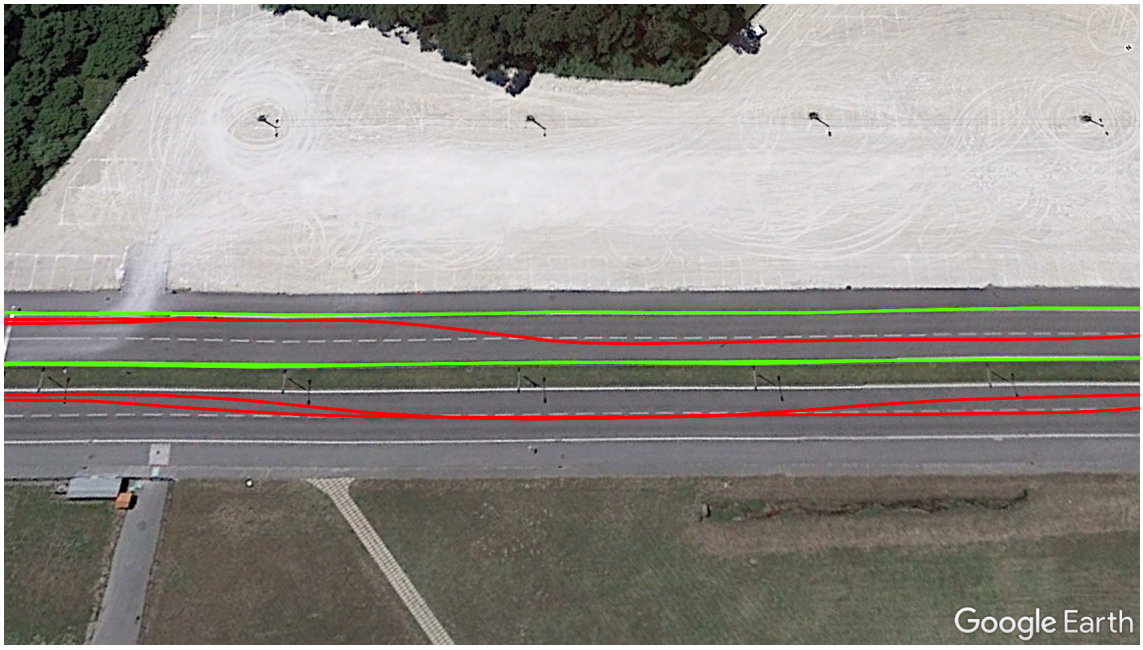
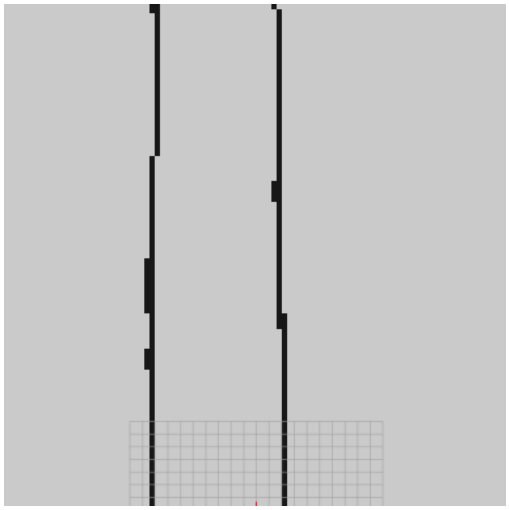
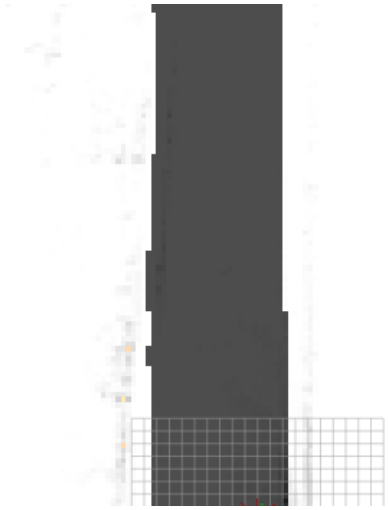


Figure 5.4: Google Earth representation of the car path, in red, and correspondent ground truth curbs, in green.

First, a ground truth grid with the right and left limits is obtained (green lines in Figure 5.5a) and then filled to comprehend the navigable space (Figure 5.5b). The same process is applied to the grid with the closest limits to the car (see Section 4.5) to allow comparison through statistical calculations. The obtained grid with the navigable space calculated with the road perception method can be observed in Figure 5.6. With a visual analysis of this last grid, it is clear that road limits are not always well defined, especially in the beginning and end of the grid, mostly due to lack of information at certain cells or false positives on the middle of the road caused by excessive accumulation in the road surface, nevertheless a road profile is visible and allows comparison with the ground truth grid calculated.



(a) Grid with a representation of a linear ground truth.



(b) Filled grid corresponding to the navigable space.

Figure 5.5: Ground truth grid.

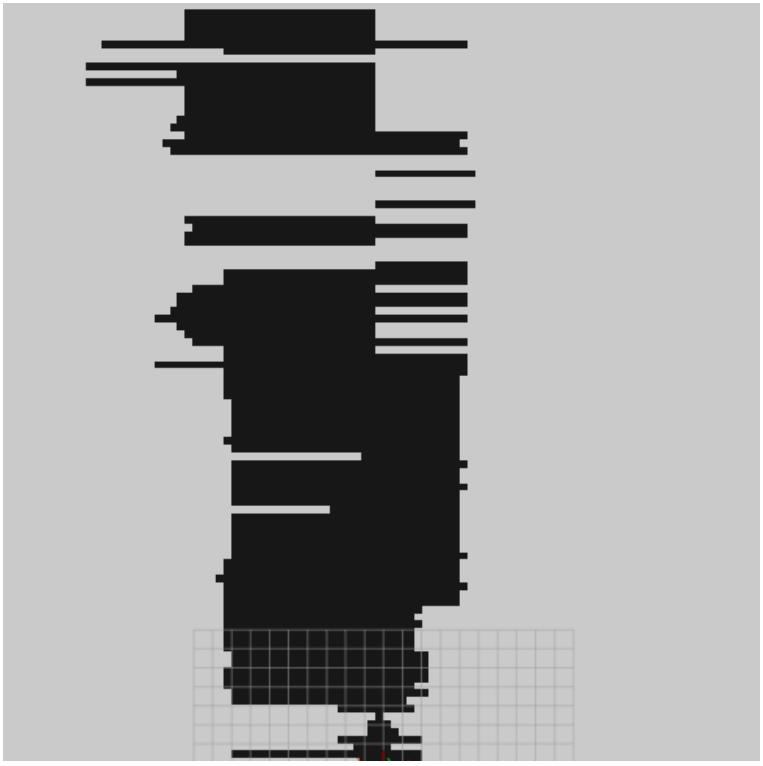


Figure 5.6: Example of navigable space grid for Laplacian edge detector.

5.5 Statistical Calculations

As mentioned before, a positive cell is considered a cell in the navigable space. So, comparing both the ground truth grid (Figure 5.5b) and the algorithm grid, the statistical measures TP, TN, FP and FN can be calculated for each frame.

To make the process more automated, a variable was created that is set when the ground truth's points are close enough to the car to appear in ground truth grid, letting the program know that it should start performing statistical calculations and saving the indicators. For each frame, measurements from Equations 2.2, 2.3, 2.5, 2.6 and 2.7 are calculate and saved in an `handle` that, in the end of the program, is exported in the form of a CSV file. This allows importing the performance data to a data sheet with the final aim to perform further calculations to evaluate and compare algorithms performance. Although the initial goal was to create Receiver Operating Characteristics (ROC) curves with the variation of studying parameters, this proved not to be possible due the characteristics of the problem not fitting the necessary profile (the Specificity does not variate).

Due to excessive accumulation directly in front of the car (Figure 5.8), the quantitative evaluation will only consider information from 10 m to 30 m from the car front, due to considering that no decisions from navigation algorithms will be taken for that distance, once 10 m are traveled in less than 1 s at normal car speed. Note that this excessive accumulation is due to alterations in the LIDAR inclination with the car movement. Moreover LIDAR is set to a covering range of only 85º, as shown in Figure 5.7, making the accumulation close to the car restricted to the area represented in blue.

Also, the accumulated information from point clouds is not well defined to correctly map the road at big distances and the further the distance, the more limits fade also due to changes in the car pitch and roll caused by accelerating, decelerating and changing direction. For all these reasons, calculations will only consider 20 m ahead of the car, meaning half the grid.

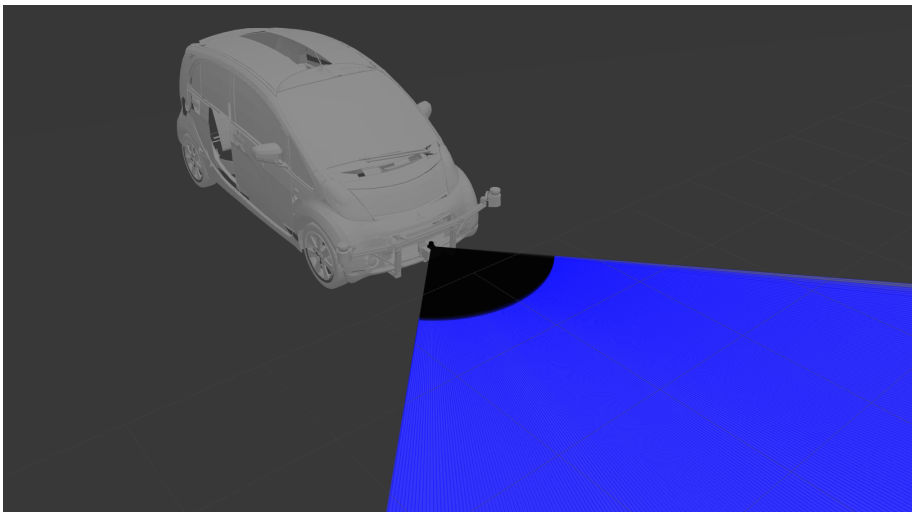


Figure 5.7: LIDAR simulation with covered range of 85º.

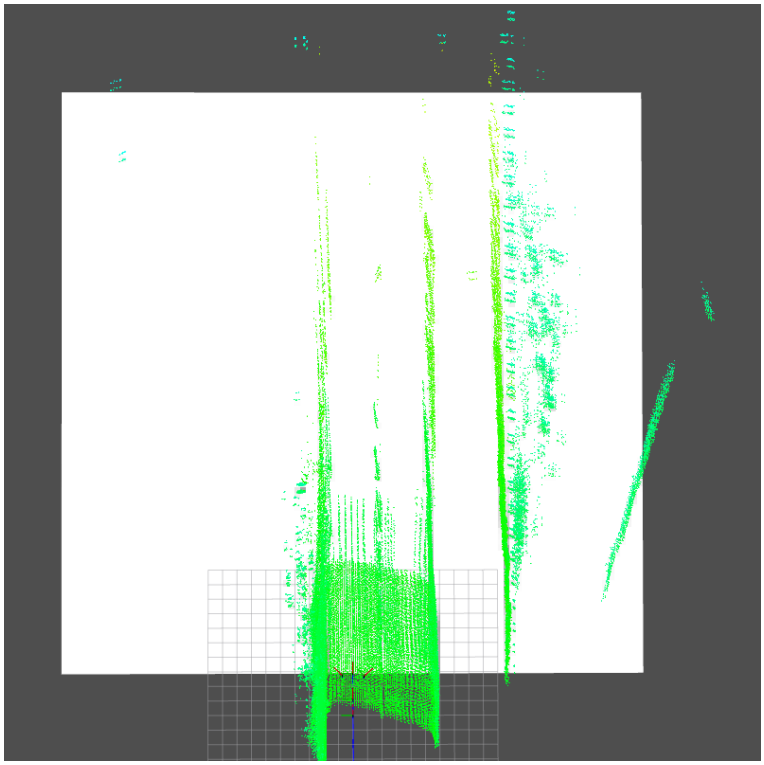


Figure 5.8: Excessive accumulation directly in front of the car.

Intentionally blank page.

Chapter 6

Tests and Results

The definition of a metric to quantify the performance of road detection is fundamental, not only to generally evaluate the results but mainly to study the influence of some parameters and optimize the algorithm to its best performance and evaluate its robustness to different conditions.

In this chapter, the influence of cell resolution, car velocity, grid thresholding, road condition, and profile and, mainly, the difference of detection performance between each algorithm.

6.1 Qualitative Evaluation

Looking at Figures 4.8 and 4.9 that show the results of all the algorithms in a straight road and in a curve situation, visually, the *Laplacian* filter seems to be the one that performs better in both situation. This filter is a very common edge detector, usually combined with a *Gaussian* filter to remove noise. In this case, removing the noise would not produce the desired result because it smooths the edges and creates false information.

The *Sobel* operator seems to produce also very efficient edge detection, slightly similar to *Canny* but the latter results in some loss of information and doesn't detect more distant edges.

The *Kirsch* operator visually works very well in a straight road, but it appears not so efficient in the curve detection. *Kirsch* operators are usually a combination of 8 multidimensional masks, covering all 8 directions, although here only two directions were considered (see section 4.4). It is also very sensitive to noise, like *Laplace* operator.

Gradient and *Prewitt* operators present more blurred results and then again *Prewitt* doesn't detect edges at longer distances.

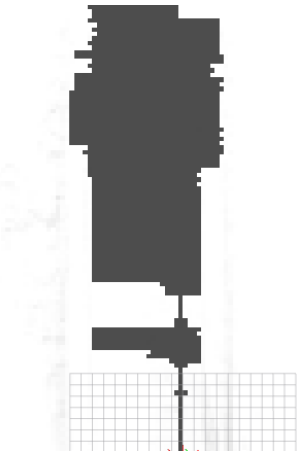
In the straight road situations, the *Laplacian* operator presents again more complete and fine results, detecting information, namely on the right side of the road, where the other algorithms fail. Comparing it with other detection mentioned above, *Kirsch* filters detect obstacles with less precision, and gradient filters identify fewer obstacles than the rest as only x and y directions are considered in contrast to other filters.

Also in Figures 6.1 there is a representation of the navigable space obtained from each edge detector in the same moment. By the analysis of these images, it can be concluded that:

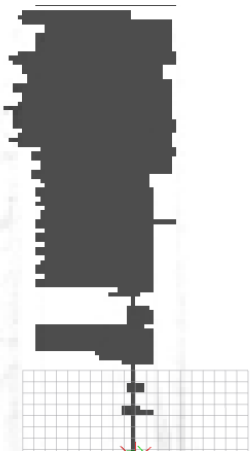
- Although being the only one that results relatively well close to the car, the *Canny*

edge detection produces poor worse results when more distant from the car, with lots of gaps in the detection.

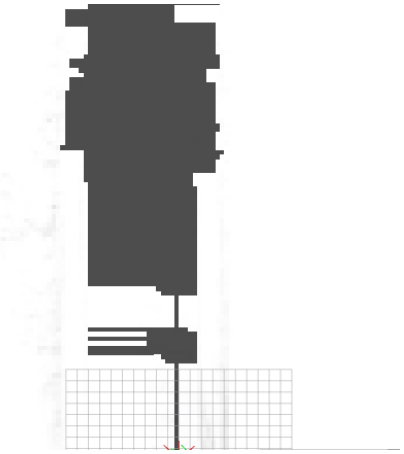
- All the edge detectors, apart from *Canny* produce poor results in the 10 m ahead of the car.
- The simple Gradient filter is the one with fewer gaps in the detection and more clear road, apart from the initial meters.
- The *Laplace* and *Prewitt* algorithms produce similar results with some gaps in the middle of the road.
- *Sobel* and *Kirsch* filters have defined road but further away from the car than the rest of the algorithms.



(a) Grid with *Laplacian* filter.

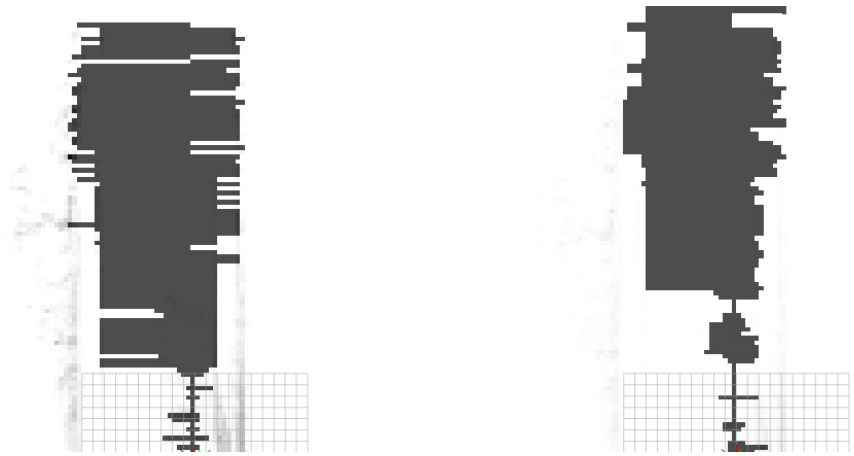


(b) Grid with *Prewitt* filter.



(c) Grid with *Sobel* filter.

(d) Grid with *Canny* filter



(e) Grid with *Gradient* filter. (f) Grid with *Kirsch* filter

Figure 6.1: Behavior of edge detection techniques to detect navigable space in a straight road situation.

6.2 Quantitative Evaluation

The quantitative evaluation of the algorithms is done using the methodology explained in Chapter 5. The calculations are performed for a frame range of 100 to 1000 frames and then the value for each parameter is calculated.

For the purpose of this work, the most relevant parameters are F-measure and Accuracy, once these include the global performance of the algorithm in terms of detection of positives and negatives. The remaining parameters are less relevant but are also taken into consideration, especially if they present particularly low values that may mean an unreliable algorithm.

For example, Precision and Specificity are not very relevant for this evaluation because even if the algorithm performs a poorly, the quantity of FP will be almost null, resulting in a good Precision and Specificity too but not reflecting well the real performance of the algorithm.

6.2.1 Algorithm's Performance

In a preliminary analysis, all algorithms were tested in the exact same straight road, using a `rosbag` file. The cell resolution was set to 0.4 m/cell and all the thresholds after the edge detection application were set to zero. The threshold application is used to eliminate the filter's lower values in order to remove some noise, mainly in filters more prone to it like *Laplace* and *Kirsch*. In this case, applying no threshold means that all the edge detection values obtained from the filtering are considered to calculate the navigable space.

Figure 6.2 shows a satellite view of the path used to perform this evaluation in Rua da Pêga. This path has about 115 m. Figure 6.3 also shows a camera image in the moment of evaluation, from the camera installed on the *AtlasCar2*.

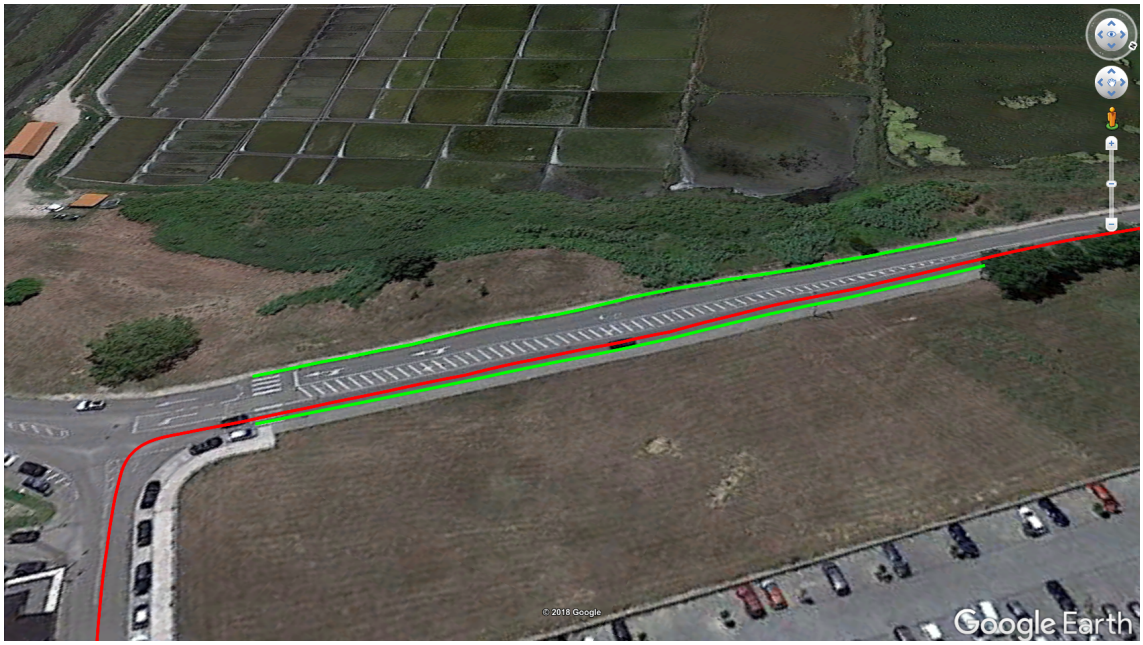


Figure 6.2: Satellite view of the path used to do the evaluation. In red, the GPS path of the car, previously calculated. In green, the drawn road limits used for evaluation.

Figure 6.3: Camera view of the path used to do the evaluation.

This test aims to do a pre-analysis of the system and compare algorithms among them for further evaluation.

Filter	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
Laplace	84.1	90.5	70.7	60.5	70.0	75.9
Gradient	83.8	86.6	84.5	83.3	83.0	84.6
Sobel	83.9	89.8	71.8	63.1	71.7	76.7
Prewitt	81.6	87.5	78.9	72.6	76.0	80.2
Kirsh	86.9	89.9	72.3	66.7	75.4	78.1
Canny	86.0	91.5	66.6	52.3	62.5	71.7

Table 6.1: Statistical indicators in each algorithm's performance with 0.4m/cell and no threshold applied (10 m to 30 m).
Note: Percentage figures.

According to the results presented in table 6.1, and considering the Accuracy and F-measure as the most important indicators of evaluation, the edge detection tool that presents the best results is clearly the *Simple Gradient*. Although very simple, it provides a robust solution by being insensitive to noise.

Also, the *Prewitt* edge detector provides good results in F-measure and Accuracy, similar to the Gradient. It is interesting to note that the filters that visually look more faded are the ones who have the best results.

Although being a very powerful and used edge detector tool one of the bests in terms of Precision, the *Canny* edge detector presents poor Sensitivity and F-measure, making the algorithm less suitable for this type of application.

6.2.2 Influence of Road Condition and Curb Profile

According to Yang et al. in [25], there are 3 curb profiles (Figure 6.4). Typically, Type I curbs are the most common and usually easier to detect and correspond to the previous testings. Type II curbs are more subtle, yet still detected by T. Marques algorithm. Type III curbs are called inverted curbs and are harder to detect and representing negative obstacles, defined by the lack of information in the shadow zone, visible as the unknown area seen in Figure 4.1 [25]. Nevertheless, the fact of using edge detection techniques as a tool to detect density changes allows to also identify the negative obstacles, defined as negative gradients as well as the positive obstacles, which were already detected by T. Marques with high-density search [12].

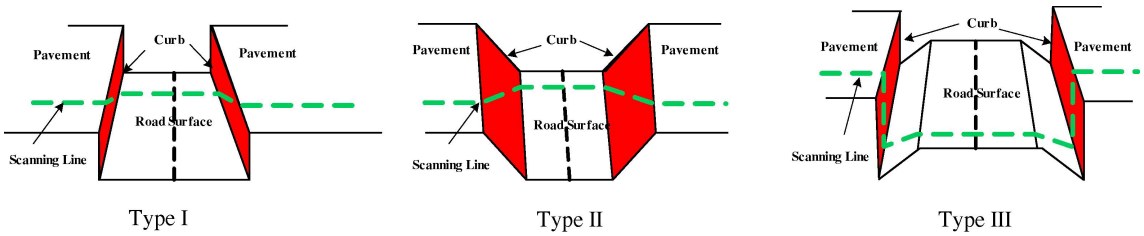


Figure 6.4: Types of curb profiles.

Figure 6.5 shows the path of about 280 m used to perform the evaluation of the algorithms in the presence of negative obstacles. Also Figure 6.6 shows a camera image in the moment of evaluation, from the camera installed on the *AtlasCar2*.

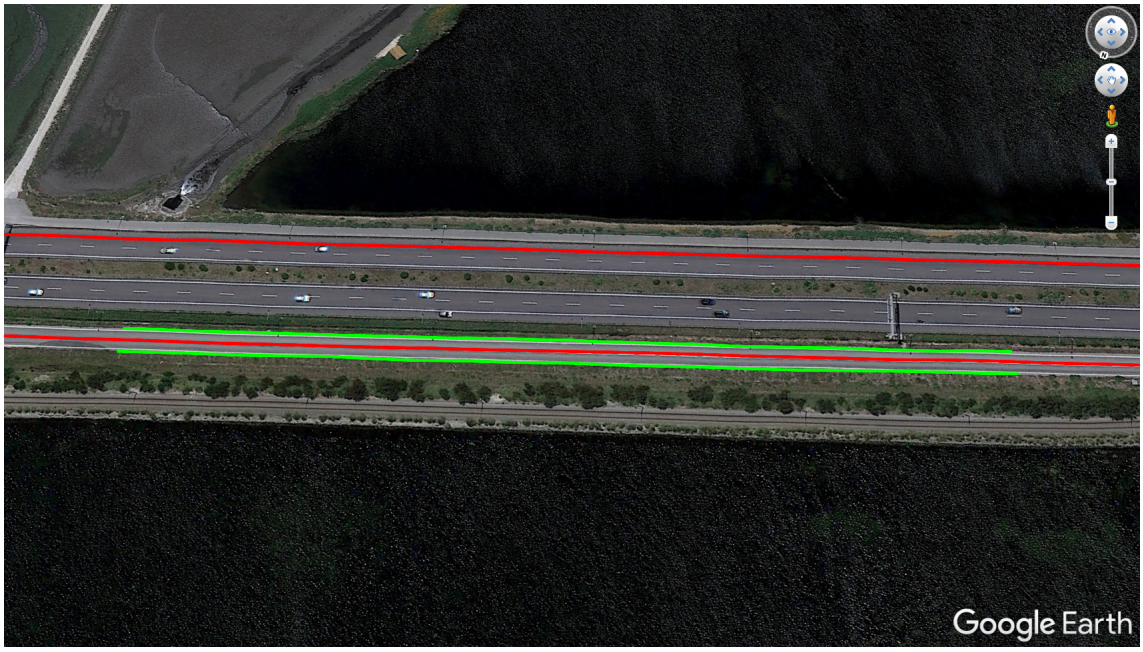


Figure 6.5: Satellite view of the path used to do the evaluation of negative curbs. In red, the GPS path of the car, previously calculated. In green, the drawn road limits used for evaluation.

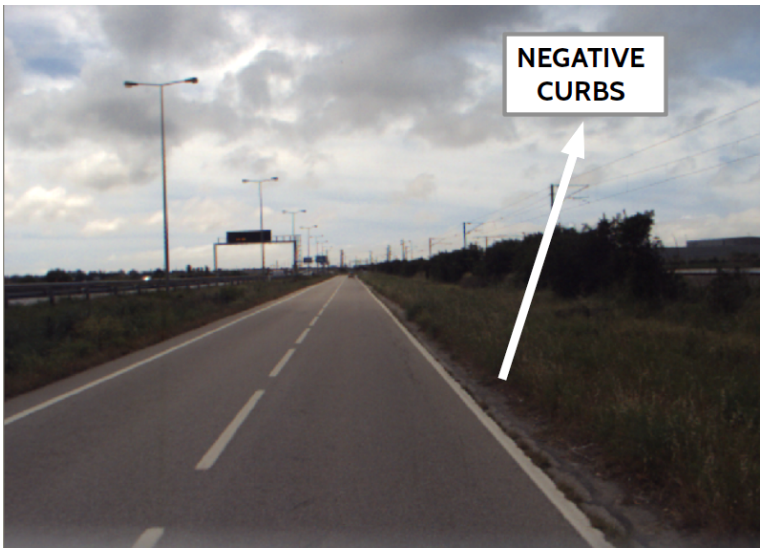


Figure 6.6: Camera view of the path used to do the evaluation of negative curbs.

To evaluate the performance of algorithms when faced with negative obstacles, a test was performed in a straight road with Type III curbs. The results obtained are presented in table 6.2.

Filter	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
Laplace	94.6	97.9	76.0	52.8	67.6	80.1
Gradient	85.0	91.8	83.6	72.0	77.9	84.0
Sobel	91.0	96.9	76.2	51.4	65.5	79.4
Prewitt	86.9	93.8	79.5	63.0	73.0	81.6
Kirsh	94.6	97.8	78.9	59.6	73.0	82.7
Canny	78.9	95.2	67.5	27.3	39.6	68.8

Table 6.2: Performance of algorithms in the presence of Type III curbs with 0.4m/cell and no threshold applied (10 m to 30 m).

When comparing the obtained results with the ones obtained with Type I curbs in the previous subsection (Subsection 6.2.1), the numbers are very similar and, in some cases, even better.

The *Canny* edge detector continues to produce poor results, with even lower F-measure and Sensitivity, proving once again not to be suitable for this application.

The *Simple Gradient* filter still produces the best results, although in general, all algorithms had a decrease of Sensitivity when comparing them with the Type I curbs, that measures the fraction of actual positives correctly identified.

6.2.3 Influence of Occupancy Grid Resolution

To evaluate the influence of the cell resolution (m/cell) in the performance of the system in the same stretch of road, test were made with different resolutions. Table 6.3 and Figure 6.7 show the obtained results. This test was performed with the *Simple Gradient* edge detector since it was the one that better identifies the situations, and also for considering that the influence of the grid resolution on the other algorithms would be similar.

This evaluation was performed on the same road shown in Figures 6.3 and 6.2 by repeating the process with different cell resolutions.

Resolution	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.1	87.5	89.2	75.4	72.0	78.9	80.4
0.2	87.7	88.8	79.0	77.3	82.1	82.9
0.3	87.8	88.2	80.6	80.1	83.7	83.9
0.4	88.8	88.6	83.6	83.9	86.2	86.1
0.5	87.7	87.3	82.5	83.1	85.3	85.1
0.6	88.5	88.0	80.3	81.2	84.6	84.3
0.7	86.7	86.8	78.8	79.2	82.7	82.7
0.8	87.2	88.0	82.6	82.7	84.6	85.1
0.9	88.0	86.6	78.5	80.5	84.0	83.2
1.0	88.3	87.1	78.9	80.6	84.2	83.5

Table 6.3: Influence of cell resolution in statistical indicators for the density gradient with 0.4m/cell and no threshold (10 m to 30 m). Note: Percentage figures

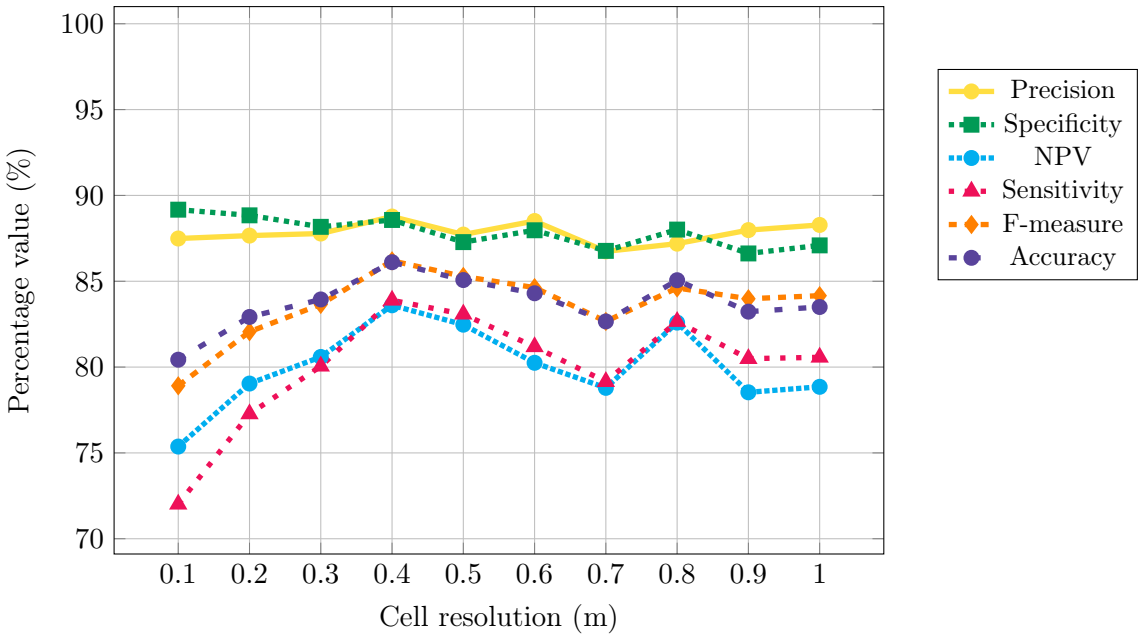


Figure 6.7: Influence of cell resolution in Statistical indicators for the density gradient with 0.4m/cell and no threshold applied (10 m to 30 m).

Through the observation of the graphic, it can be concluded that a resolution of 0.4 m/cell is the one that produces the best results. Taking into account that a pedestrian can be fitted in a square of about 0.5 m × 0.5 m, a resolution of 0.4 m is considered enough to even identify a person in the road. Logically, if there is a thin object in the road, like a road pin, this resolution can lead to the dissolution of the points in that cell and lead to the non identification of the obstacle, but there has to be a trade-off between a very thin resolution that results in worse outcome and a not so big cell that can still detect small objects.

6.2.4 Influence of Car Velocity

One of the big concerns with this work was its robustness to the different car speeds.

To evaluate the performance of road detection with car velocity, the same road was tested at different speeds, evaluating the system in about 200 m. Logically, the number of frames evaluated varies according to the car velocity since lower speed results in more time to travel the same distance and, consequently, more frames.

Figure 6.8 shows the path of about 230 m used to perform the evaluation of car speed in the performance of the Gradient detector. Also, Figure 6.9 shows a camera image in the moment of evaluation, from the camera installed on the *AtlasCar2*.

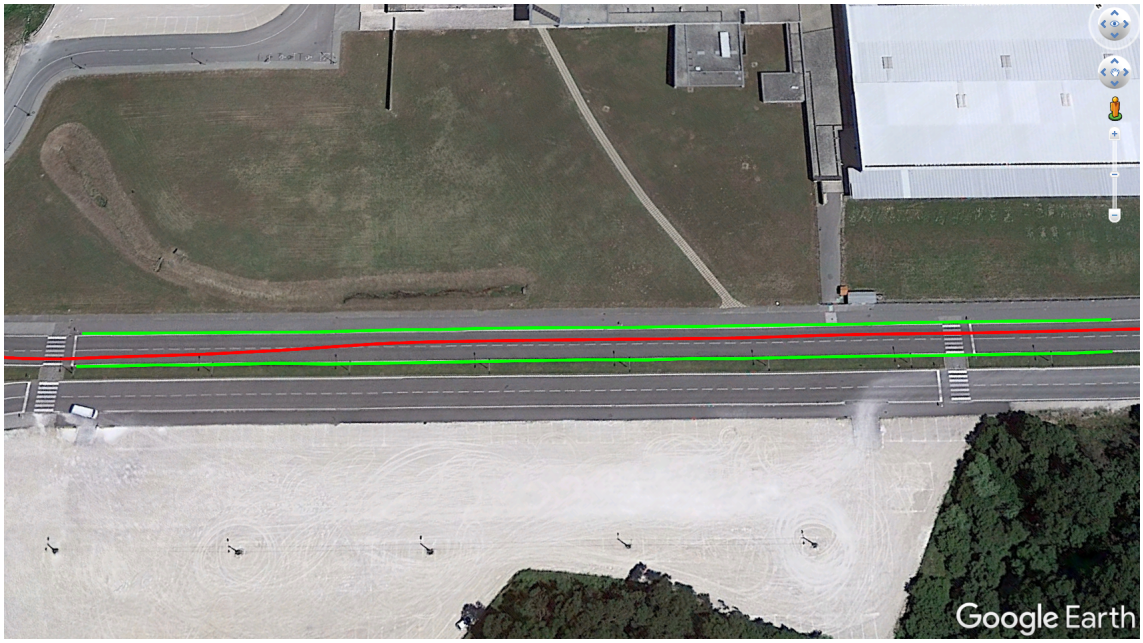


Figure 6.8: Satellite view of the path used to evaluate the influence of car speed in the performance of algorithms. In red, the GPS path of the car, previously calculated. In green, the drawn road limits used for evaluation.

Figure 6.9: Camera view of the path used to evaluate the influence of car speed in the performance of algorithms.

Table 6.4 and the graphic in Figure 6.10 show the results of the algorithm analysis by varying car speed from 20 to 60 $\mathrm{km\,h^{-1}}$.

Velocity	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
20.0	76.2	84.5	82.1	72.4	74.1	79.7
30.0	78.9	87.1	81.9	71.3	74.9	80.7
40.0	79.2	88.8	83.6	72.4	75.5	82.3
50.0	74.6	84.6	82.9	72.5	73.0	79.7
60.0	65.8	83.1	70.4	47.9	55.1	68.9

Table 6.4: Influence of car velocity ($km\,h^{-1}$) in the statistical indicators for the density gradient with 0.4 m/cell and no threshold (10 m to 30 m).
Note: Percentage figures.

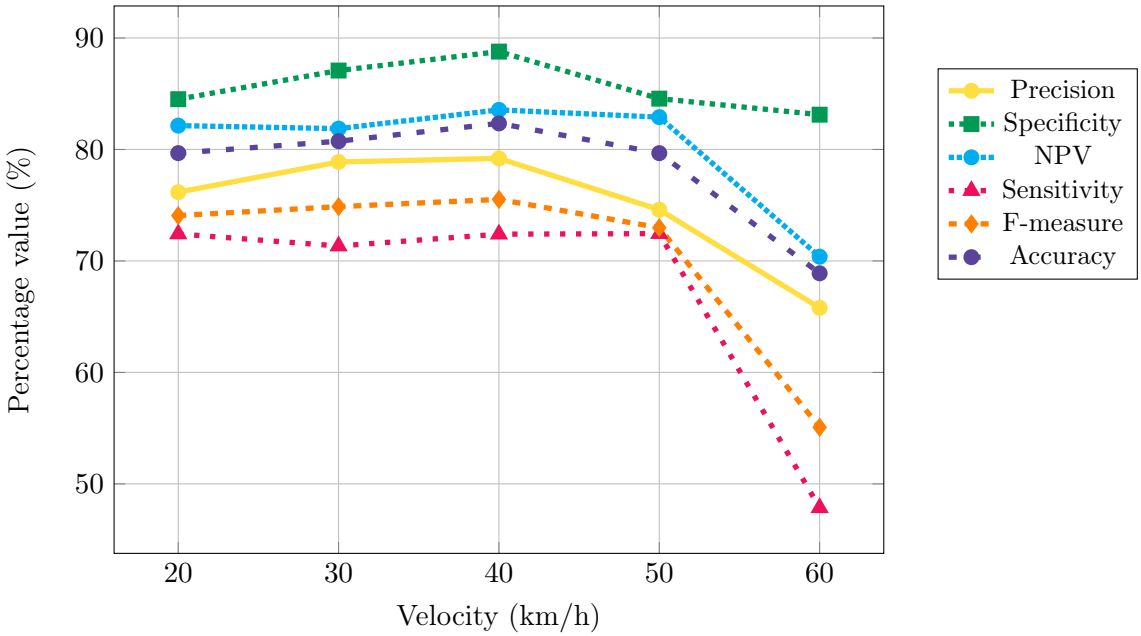


Figure 6.10: Influence of car velocity in statistical indicators for the density gradient with 0.4m/cell and no threshold (10 m to 30 m).

Through the analysis of Figure 6.10, it can be concluded that the performance maintains more or less constant until $50\,km\,h^{-1}$. After this value, the performance of the algorithm starts decreasing but still above 50%.

Note that tests in the same road could not be performed at higher velocities due to the speed limit in that zone.

6.2.5 Influence of Grid Threshold

When using the `OpenCV` toolbox to apply edge detection filters and 2D filters to `Occupancy Grids` it is possible to apply a threshold when converting the converted image back to an `Occupancy Grid` type of message. This can be interesting to eliminate some lower values that correspond to road noise or noise obtained in the filtering. Logically, the threshold value varies from filter to filter because some are much more susceptible to noise, like

Kirsch, than others, like *Gradient*. For this reason, the study was performed individually for each algorithm in order to optimize the performance of each one.

As mentioned before, as the image is of the type `CV_8UC1` it means that the threshold is within the range of 0 to 255.

This evaluation was also performed on the same road shown in Figures 6.3 and 6.2 by repeating the process with different threshold values for each algorithm.

Gradient

Table 6.5 and the graphic in Figure 6.11 show the results of the Gradient filter by varying the threshold from 0 to 175, due to the results being already conclusive without going any further.

As expected, this bi-dimensional filter is little affected by noise; thus, increasing the threshold value results in a loss of information, as seen in Figure 6.11. Also, the reason for the increasing of Specificity with the threshold is due to the loss of information, also provoking a decrease of FP, making equation 2.3 tend to 1. It is then concluded by the analysis of the results that the algorithm performs better when no threshold is applied.

Threshold	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.0	86.6	88.1	85.2	84.3	85.2	86.1
10.0	87.1	88.2	80.0	77.8	81.9	83.1
20.0	87.4	91.2	67.0	56.1	66.5	72.9
50.0	85.4	97.2	54.2	20.5	30.2	58.0
75.0	85.6	98.9	51.9	9.9	16.6	54.1
100.0	80.5	99.6	55.4	5.0	9.2	56.3
125.0	64.4	99.8	49.2	2.0	3.8	49.7
150.0	35.3	99.9	48.1	0.7	1.4	48.2
175.0	11.8	100.0	47.7	0.3	0.5	47.8

Table 6.5: Influence of algorithm threshold in Statistical indicators for the density gradient with 0.4m/cell (10 m to 30 m). Note: Percentage figures.

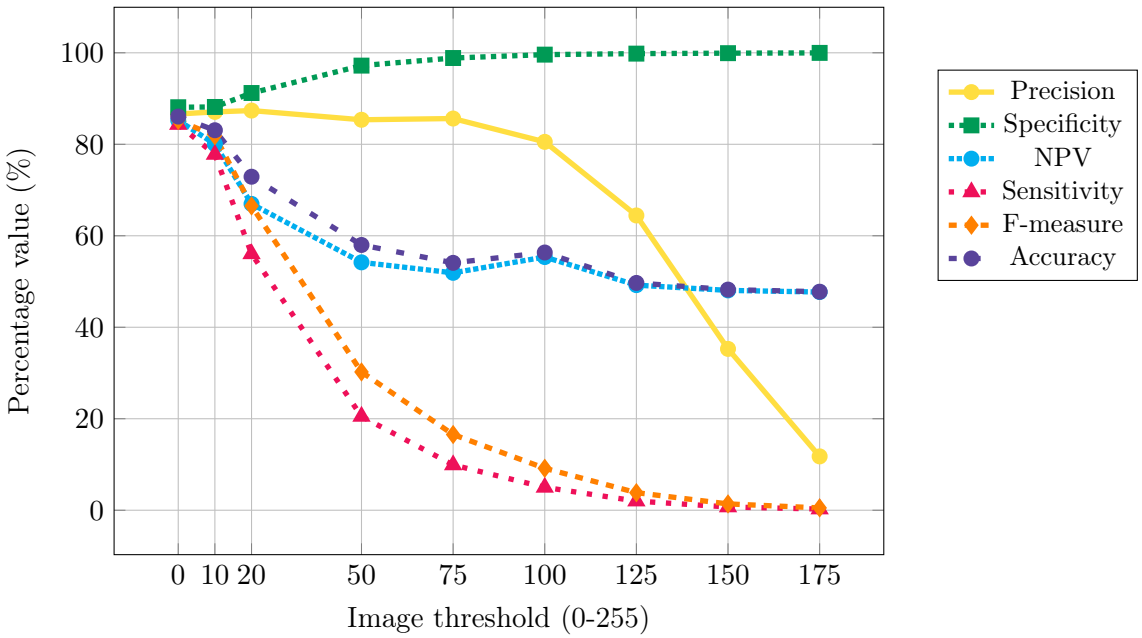


Figure 6.11: Influence of algorithm threshold in statistical indicators for the density gradient with 0.4m/cell (10 m to 30 m).

Laplace

Table 6.6 and the graphic in Figure 6.12 show the results of the *Laplace* filter by varying the threshold from 0 to 250.

As predicted, the *Laplace* filter is more susceptible to noise and therefore the use of a threshold to remove some noise highly increases the performance of the algorithm.

Also, as concluded in the qualitative evaluation in Section 6.1, the initial meters are the ones with more noise and worse detection due to the high accumulation of points directly in front of the car explained in section 5.5. This means that the evaluated section has less noise, and a low threshold is enough to achieve the best performance from these filters. It is then concluded by the analysis of Figure 6.12 that this algorithm performs the best with a threshold of 50.

Threshold	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.0	90.5	93.0	70.1	63.1	74.3	77.5
25.0	89.0	89.2	79.4	78.8	83.5	83.8
50.0	88.7	88.0	84.3	84.9	86.7	86.4
75.0	88.8	88.1	83.9	84.3	86.4	86.2
100.0	88.2	88.0	81.3	80.6	84.0	84.3
125.0	88.8	88.9	78.5	77.3	82.3	82.8
150.0	87.2	89.1	75.1	70.8	77.5	79.8
175.0	88.5	89.5	73.0	69.0	76.6	78.6
200.0	88.0	90.2	69.6	62.7	71.7	75.5
225.0	88.7	90.5	68.3	61.0	70.5	74.6
250.0	87.0	91.3	64.6	52.4	62.7	70.5

Table 6.6: Influence of algorithm threshold in Statistical indicators for the *Laplace* operator with 0.4m/cell (10 m to 30 m).
Note: Percentage figures.

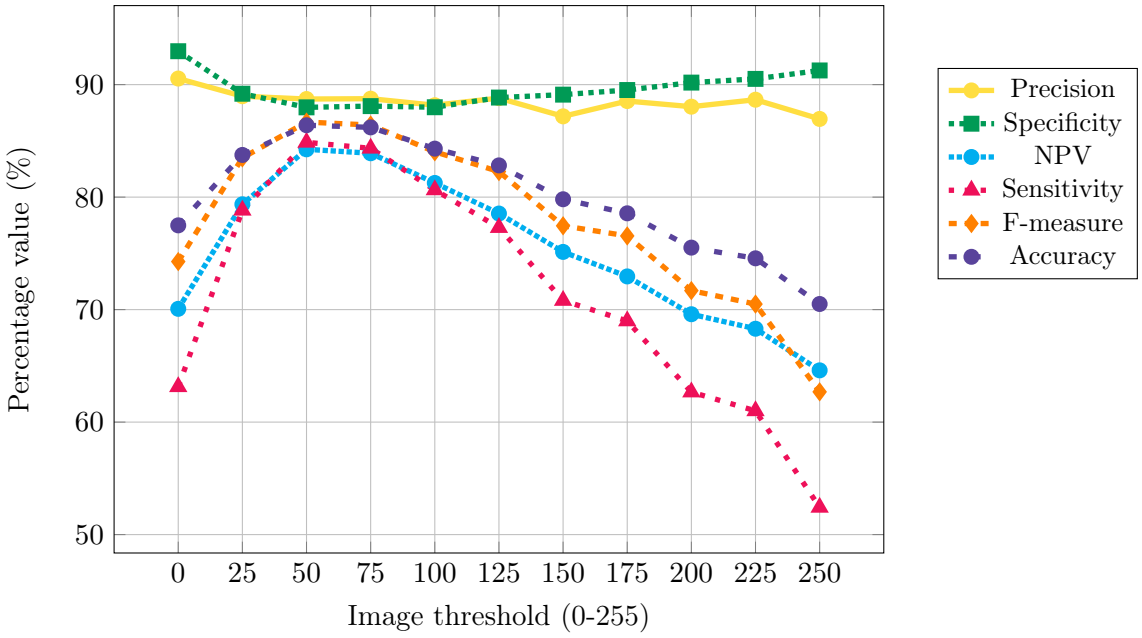


Figure 6.12: Influence of algorithm threshold in Statistical indicators for the *Laplace* operator with 0.4m/cell (10 m to 30 m).

Sobel

Table 6.7 and the graphic in Figure 6.13 show the results of the *Sobel* filter by varying the threshold from 0 to 200. By the analysis of the graphic, this filter has the best result when using a threshold of 25.

Threshold	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.0	90.4	92.5	70.1	64.3	75.1	77.7
25.0	88.0	88.0	79.9	79.6	83.5	83.7
50.0	87.5	88.5	73.8	70.4	77.4	79.0
75.0	86.8	90.6	66.7	56.4	66.9	72.8
100.0	85.4	91.9	62.1	45.5	56.7	67.6
125.0	83.3	93.5	58.1	34.9	45.5	62.9
150.0	83.3	95.3	55.2	27.1	37.1	59.6
175.0	86.1	97.3	61.8	23.0	33.7	65.3
200.0	86.1	98.5	50.8	12.3	19.3	53.4

Table 6.7: Influence of algorithm threshold in Statistical indicators for the *Sobel* operator with 0.4m/cell (10 m to 30 m). Note: Percentage figures.

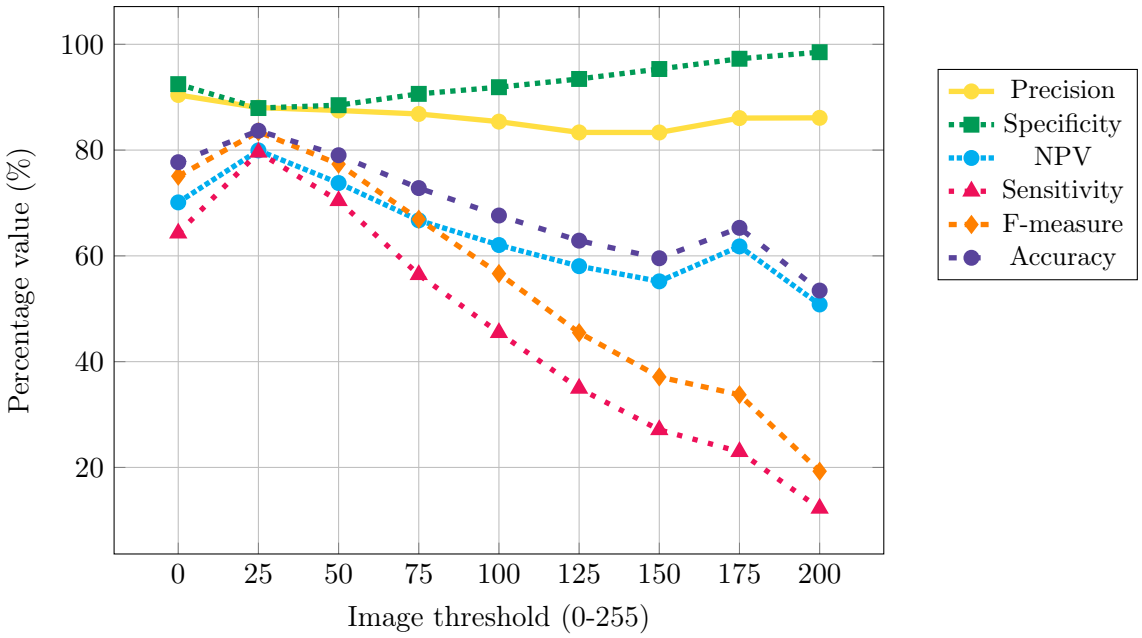


Figure 6.13: Influence of algorithm threshold in Statistical indicators for the *Sobel* operator with 0.4m/cell (10 m to 30 m).

Prewitt

Table 6.8 and the graphic in Figure 6.14 show the results of the *Prewitt* filter by varying the threshold from 0 to 175.

Very similar to the Gradient filters, the *Prewitt* filter also performs better with no threshold. Also in Section 6.1 it was concluded that these two filters, visually, were the ones that looked more faded and less susceptible to noise. The results present in the graphic confirm the visual analysis.

Again the Specificity increases with the decrease of performance for the reasons already

mentioned.

Threshold	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.0	90.6	91.6	76.0	73.7	81.2	82.2
25.0	87.9	88.5	76.5	74.2	80.1	81.1
50.0	86.7	91.2	65.0	52.7	63.8	71.1
75.0	84.8	93.4	58.0	35.7	46.9	63.1
100.0	82.7	95.4	54.5	25.6	35.6	58.7
125.0	81.8	96.7	52.2	18.6	27.4	55.6
150.0	79.8	98.8	50.0	10.5	16.9	52.4
175.0	47.8	99.4	48.9	6.2	10.5	50.3

Table 6.8: Influence of algorithm threshold in Statistical indicators for the *Prewitt* operator with 0.4m/cell (10 m to 30 m). Note: Percentage figures.

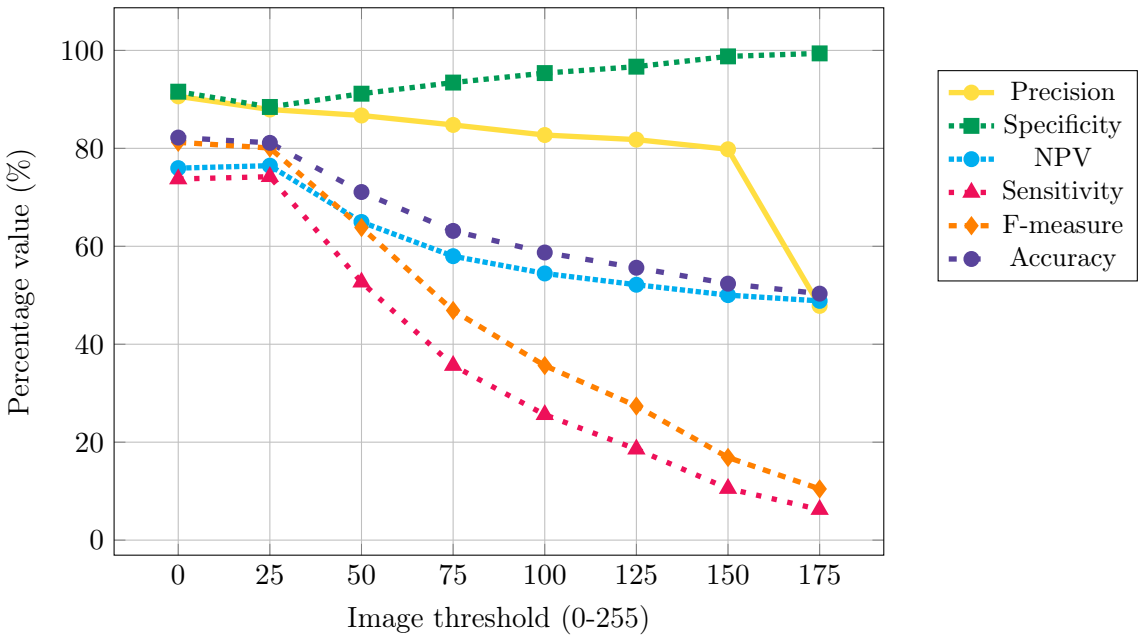


Figure 6.14: Influence of algorithm threshold in Statistical indicators for the *Prewitt* operator with 0.4m/cell (10 m to 30 m).

Canny

Table 6.9 and the graphic in Figure 6.15 show the results of the *Canny* filter by varying the threshold from 0 to 200.

As mentioned before, this filter is the one with the worst, results making the algorithm less suitable for this type of application, mainly due to distorting the results by applying a Gaussian and making operations that changes the true data obtained from the LIDAR. Nevertheless, this algorithm also performs better with when no is threshold applied.

Threshold	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.0	87.6	91.4	65.3	53.2	64.0	71.4
25.0	87.1	91.5	64.0	51.6	63.2	70.6
50.0	87.7	93.2	60.5	44.1	57.3	67.3
75.0	87.9	94.1	58.7	39.0	52.2	65.0
100.0	87.6	94.3	57.7	35.5	47.7	63.3
125.0	87.5	94.8	56.3	31.1	42.0	61.2
150.0	84.6	96.5	52.6	20.4	29.8	56.4
175.0	70.7	98.5	49.6	9.6	16.0	51.6
200.0	58.3	99.6	48.0	3.1	5.7	48.7

Table 6.9: Influence of algorithm threshold in Statistical indicators for the *Canny* operator with 0.4m/cell (10 m to 30 m). Note: Percentage figures.

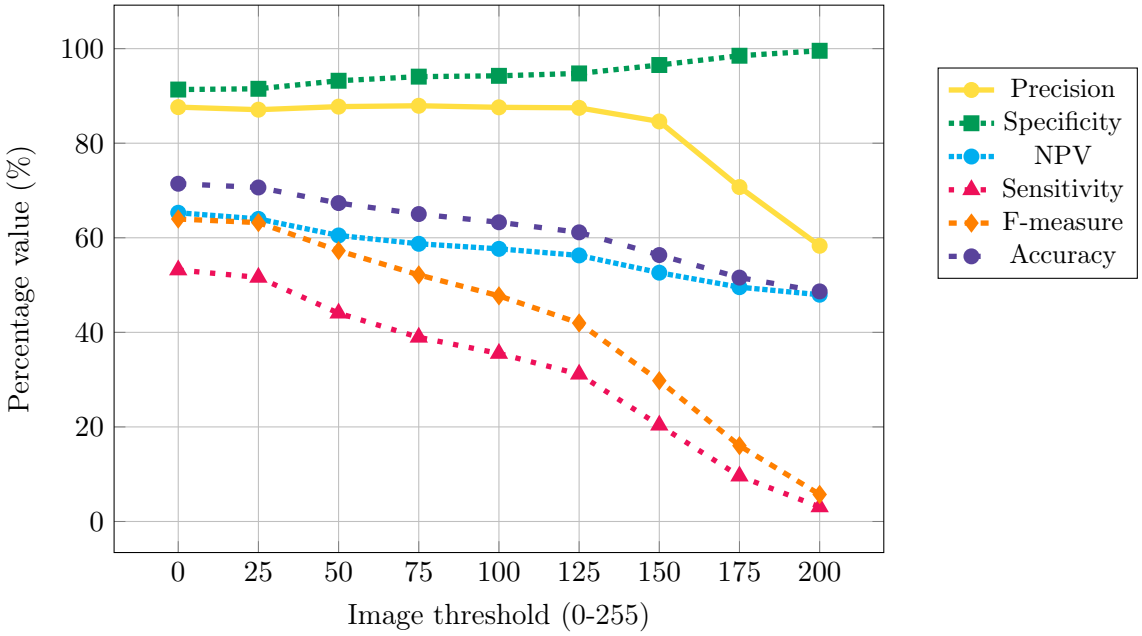


Figure 6.15: Influence of algorithm threshold in Statistical indicators for the *Canny* operator with 0.4m/cell (10 m to 30 m).

Kirsch

Table 6.10 and the graphic in Figure 6.16 show the results of the *Kirsch* filter by varying the threshold from 0 to 200.

Very similar to *Laplace*, this filter is also very susceptible to noise and presents a very similar response to the variation of the threshold, as seen in the graphic.

Also like *Laplace* it also has the best performance at a threshold of 50.

Threshold	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
0.0	89.2	90.9	71.6	67.5	76.8	78.6
25.0	87.7	87.1	82.5	83.0	85.2	85.0
50.0	87.6	86.4	84.9	85.8	86.6	86.2
75.0	87.2	86.3	82.8	83.3	85.1	84.8
100.0	86.9	86.7	79.2	78.2	82.0	82.4
125.0	86.5	87.2	75.7	72.9	78.5	79.9
150.0	88.0	91.9	66.2	55.0	66.1	72.6
175.0	87.9	94.0	60.9	42.8	55.2	67.2
200.0	87.6	95.8	56.9	32.2	44.3	62.4
225.0	87.5	96.9	54.3	24.9	36.1	59.1

Table 6.10: Influence of algorithm threshold in Statistical indicators for the *Kirsch* operator with 0.4m/cell (10 m to 30 m).Note: Percentage figures.

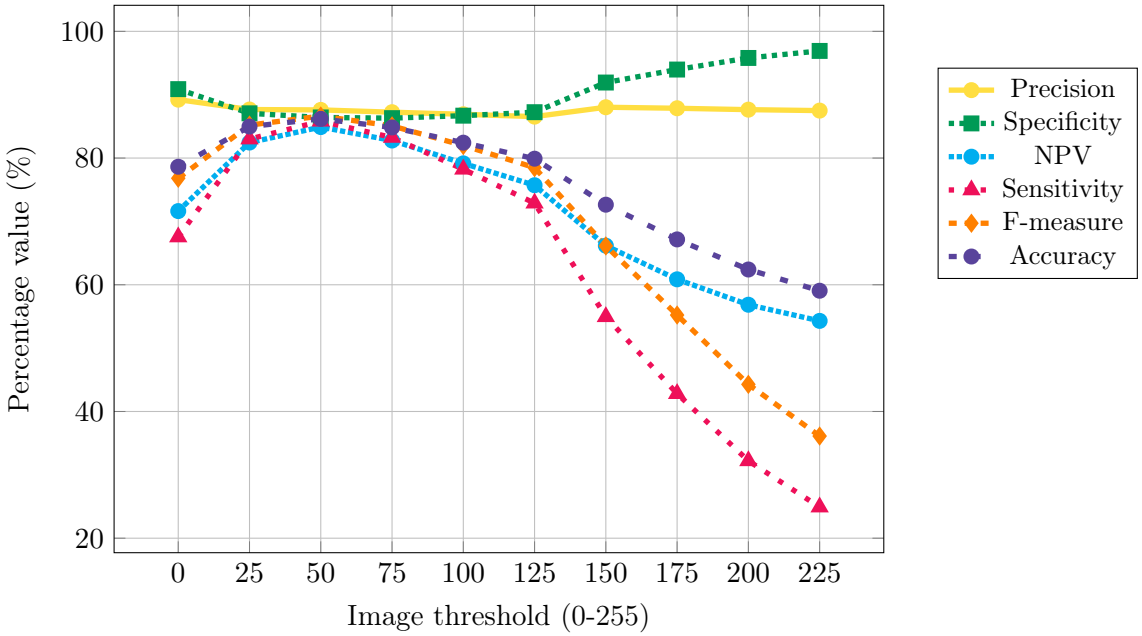


Figure 6.16: Influence of algorithm threshold in Statistical indicators for the *Kirsch* operator with 0.4m/cell (10 m to 30 m).

Summary

In conclusion, Table 6.11 presents the most adequate threshold for each algorithm. It is also very interesting to confirm that the visual analysis is done in section 6.1 is almost always confirmed by the subsequent calculations.

Filter	Laplace	Gradient	Sobel	Prewitt	Kirsch	Canny
Threshold (0-255)	50	0	25	25	50	0

Table 6.11: List of best threshold for the several algorithms.

6.2.6 Performance with improved algorithms

To conclude, Table 6.12 shows the results of all the algorithms with improved thresholds, car speed, and cell resolution.

Filter	Precision	Specificity	NPV	Sensitivity	F-measure	Accuracy
Laplace	88.9	88.1	84.8	85.6	87.1	86.8
Gradient	89.4	89.0	83.4	83.9	86.5	86.3
Sobel	87.1	87.6	80.4	79.6	83.0	83.5
Prewitt	87.6	88.5	77.2	74.4	80.0	81.4
Kirsch	86.6	86.0	84.8	85.3	85.9	85.7
Canny	87.3	91.1	67.5	56.0	66.3	73.3

Table 6.12: Result of the performance of the algorithms with the improved parameters.

By analyzing this table, it can be concluded that Gradient and the *Laplace*, very close to *Kirsch* filter produce the best results. The *Canny* edge detector, for the reasons already mentioned, is not the most suitable for this application and the *Sobel* and *Prewitt* filters, very similar to each other, although having good results fall short of the best algorithms.

Overall the work showed better than expected results with good road detection and more robustness than the previous solution.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Since the beginning, this dissertation proposed to find an effective way to correctly identify road limits and the navigable space of the road. The thesis to study was that the density of the accumulated cloud, obtained from car movement with an 85^{o} range 3D LIDAR, is a new and innovative methodology to perform road detection. Unlike most of the work done in this area, using 360^{o} Velodyne LIDARs, seeing the road at ground level offers a completely different perspective, possibly allowing to make real-time driving decisions taking into account the road limits identified.

To solve the problem, the approach used was to transform accumulated point clouds in a density occupancy grid and, originally to calculate a two-dimensional gradient and subsequently to apply edge detection algorithm to find density patterns that define obstacles. One of the biggest innovations, when compared with previous work done in the *AtlasCar2*, is the ability to detect both positive and negative obstacles due to considering density increases and decreases.

When it comes to the edge detection algorithms used, tests were made to quantify the performance of each one and evaluate the influence of several factors in the global performance of the system. The conclusions taken were that:

- The *Simple Gradient* produced the best results detecting the navigable space in all situations tested.
- The *Kirsch* and *Laplace* edge detectors also proved to produce good detection results.
- The algorithm threshold that optimizes detection is different from filter to filter due to the characteristics of the same and noise sensitivity.
- The algorithms have a stable performance up to $50\,\mathrm{km\,h^{-1}}$ and from that value the performance, although acceptable, begins to decrease.
- The cell resolution that optimizes the detection of the navigable space is $0.4\,\mathrm{m/cell}$.

Briefly, the main contributions of this work were:

- The use gradient as a tool to detect hard limits of the road in a moving car;

- Development of a method able to detect different types of curbs;
- Development of a tool to evaluate algorithms performance;
- Test and prove the efficacy of the method in real time on board of the *AtlasCar2*;
- Reduce computational time of point cloud accumulation.

Additionally, the work developed in this dissertation originated an article submitted in the Fourth Iberian Robotics Conference named "Detection of Road Limits using Gradients of the Accumulated Point Cloud Density".

Broadly, the initially defined goals have been met. This work led to an effective solution in road detection that works in low and inverted curbs, detecting positive and negative obstacles. A metric to quantitatively evaluate the detection was developed and implemented and several tests in different situations were conducted on the real environment in the *AtlasCar2*.

Although this system alone may not be enough to provide data for a navigation system, when integrated with the other sensors in the *AtlasCar2*, it provides essential navigational information of hard limits. A good solution would be to complement the detection with lane perception (soft limits) to obtain a more complete and safer occupancy grid.

The work developed along the semesters was described week by week in https://danielaratodissertation.home.blog/ and the correspondent code can be view in https://github.com/danifpdra/road_detection.

7.2 Future Work

This work leaves open a wide range of possibilities. On the one hand, road detection for navigational purposes can be largely improved by combining the work already developed in lane detection using cameras by T. Almeida in [2] and creating a multi-sensorial algorithm with the possibility to create an occupancy grid with different levels of probability according to the detected features (lane less serious, hard limits more serious, etc.). Fusing the results of several edge detection algorithms may also be interesting to obtain more complete and robust information. Also to improve the detection, finding a solution to the behavior of the accumulated point cloud in roundabouts, where the constant change of speed doesn't allow to properly define the features in the road, and temporary obstacles, that are delayed due to the accumulation buffer speed, would make the algorithm more robust and wide-ranging.

To improve the hardware in the car, adding a second LIDAR to cover a bigger range of road and setting the sensors to asynchronous times for more reliability at higher velocities.

On the other hand, concerning the evaluation methodology developed, the method could be extended to contemplate not only the limits of the road but also lanes and other features. It could also be upgraded to become more comprehensive by including other road situations like curves, roundabouts, and other sharp curves.

References

[1] Open Geospatial Consortium . KML. http://www.opengeospatial.org/standards/kml/.

[2] Tiago Almeida. Multi camera and multi algorithm architecture to visual perception onboard the atlascar2, 2019.

[3] Rui Azevedo. Sensor fusion of laser and vision in active pedestrian detection, 2014.

[4] Universidade de Aveiro. ATLAS project. http://atlas.web.ua.pt/index.html.

[5] Jannik Fritsch, Tobias Kuhnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1693–1700. IEEE, 2013.

[6] Rulin Huang, Jiajia Chen, Jian Liu, Lu Liu, Biao Yu, and Yihua Wu. A practical point cloud based road curb detection method for autonomous vehicle. 8(3):93, 2017.

[7] SICK Sensor Intellegence. LD-MRS400001. https://www.sick.com/nz/en/detection-and-ranging-solutions/3d-lidar-sensors/ld-mrs/ld-mrs400001/p/p112355.

[8] SAE International. LEVELS OF DRIVING AUTOMATION ARE DEFINED IN NEW SAE INTERNATIONAL STANDARD j3016, 2014.

[9] Jiyoung Jung and Sung-Ho Bae. Real-time road lane detection in urban areas using LiDAR data. 7(11):276, 2018.

[10] R. E. Kalman. A new approach to linear filtering and prediction problems. volume 82, pages 35–45. J. basic eng. edition, 1960.

[11] Y. Kang, C. Roh, S. Suh, and B. Song. A lidar-based decision-making method for road boundary detection using multiple kalman filters. 59(11):4360–4368, 2012.

[12] Tiago Marques. Detection of road navigability for ATLASCAR2 using LIDAR and inclinometer data, 2017.

[13] Ricardo Morais. Parametrização de algoritmos para deteção de estrada a bordo do atlascar, 2014.

[14] Ryan D Morton and Edwin Olson. Positive and negative obstacle detection using the HLD classifier. page 6.

[15] Pedro Nova. Localização e navegação global do atlascar2 usando gnss e interface com mapas on-line, 2018.

[16] NovAtel. SPAN-IGM-a1. https://www.novatel.com/products/span-gnss-inertial-systems/span-combined-systems/span-igm-a1/.

[17] OpenCV. About OpenCV. https://opencv.org/about/.

[18] PCL. About - point cloud library (PCL). http://pointclouds.org/about/.

[19] K. Peterson, J. Ziglar, and P. E. Rybski. Fast feature detection and stochastic parameter estimation of road shape using multiple LIDAR. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 612–619, 2008.

[20] Sérgio Pinho. Caixa automática e manobras especiais no atlascar, 2014.

[21] G. Priyandoko, T. Y. Ming, and M. S. H. Achmad. Mapping of unknown industrial plant using ROS-based navigation mobile robot. 257:012088, 2017.

[22] ROS. ROS.org | about ROS. http://www.ros.org/about-ros/.

[23] Pedro Salvado. Reconstrução dinâmica de mapa local para o AtlasCar, 2012.

[24] Sheng Xu, Ruisheng Wang, and Han Zheng. Road curb extraction from mobile LiDAR point clouds. 55(2):996–1009, 2017.

[25] Bisheng Yang, Lina Fang, and Jonathan Li. Semi-automated extraction and delineation of 3d roads of street scene from mobile laser scanning point clouds. 79:80–93, 2013.

[26] D. Zai, J. Li, Y. Guo, M. Cheng, Y. Lin, H. Luo, and C. Wang. 3-d road boundary extraction from mobile laser scanning data via supervoxels and graph cuts. 19(3):802–813, 2018.

[27] Y. Zhang, J. Wang, X. Wang, and J. M. Dolan. Road-segmentation-based curb detection method for self-driving via a 3d-LiDAR sensor. 19(12):3981–3991, 2018.

Appendices

Appendix A

Instructions to Install, Run and Manage Packages

A.1 Prerequisites and Required Installations

To run the developed packages it is necessary to install ROS Melodic with:

```
sudo apt install ros-melodic-desktop-full
```

Then, follow the steps in http://wiki.ros.org/melodic/Installation/Ubuntu to configured a catkin workspace.

To install PCL, please read http://www.pointclouds.org/downloads/linux.html.

Install OpenCV by running:

```
[compiler] sudo apt-get install build-essential
[required] sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
[optional] sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

Instal Eigen by running:

```
sudo apt-get install libeigen3-dev
```

It is necessary to install the following ROS packages:

- novatel-gps-driver
- pcl-ros
- grid-map*
- driver-base
- swri*

To install a package, use:

```
sudo apt install ros-melodic-$PACKAGE_NAME$
```

The last step is to clone the developed packages into the source folder within the catkin workspace with:

```
git clone https://github.com/danifpdra/road_detection
```

A.2 Compilation

To compile the packages, inside catkin_ws (`cd catkin_ws`), first do:

```
catkin_make -pkg driver_base
```

to generate the necessary messages to run the program. Then do:

```
catkin_make
```

and the packages should be ready to run.

A.3 AtlasCar2 Setup

To launch all sensors:

```
roslaunch atlas2_bringup drivers.launch
```

Make sure that the `novatel.launch` file, included in the `drivers.launch` file has the correct port names correspondent to the two Novatel ports.

To launch the inclinometry module:

```
rosrun rosserial_python serial_node.py /dev/$PORT_NAME$
```

To visualize the data and edge detection algorithms in real time run:

```
roslaunch negative_obstacles negative_obstacles.launch
```

A.4 Create and Manage `rosbags`

To record a new rosbag:

```
rosbag record /ld_rms/scan0 /ld_rms/scan1 /ld_rms/scan2 /ld_rms/scan3 /lms151_D_scan
/lms151_E_scan /top_camera/image_rect_color /fix /inspva /DadosInclin /gps
```

To play a rosbag:

```
rosbag play $BAG_NAME$ -clock -topic /DadosInclin /ld_rms/scan0 /ld_rms/scan1
/ld_rms/scan2 /ld_rms/scan3 /imu /gps /fix /inspva /top_right_camera/image_rect_color
```

For more options please read http://wiki.ros.org/rosbag.

A.4.1 Run the program with `rosbags`

First, in the `negative_obstacles.launch`, inside the `negative_obstacles` packages, uncomment the `<param name="use_sim_time" value="true"/>` line to use the simulated time. Then run the `negative_obstacles.launch` with the `roslaunch` command and then play the `rosbag` with the described command.