ETSETB – Degree in Telecommunication Technologies and Services Engineering

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

telecos BCN

Enhancing Cone Detection in the Perception System of a Formula Student Car

Bachelor's Degree Thesis
Submitted to the Faculty at the
Escola Tècnica Superior
d'Enginyeria de Telecomunicació de Barcelona
of the Universitat Politècnica de Catalunya
by

Víctor Moreno Borràs

In partial fulfillment
of the requirements for the
Degree in Telecommunication Technologies and Services Engineering

Advisor: Josep R. Casas Pla

Barcelona, June 2024

Resum

En qualsevol sistema de conducció autònoma, el sistema de percepció és una part fonamental. Entre altres tasques, aquest ha de ser capaç d'identificar i localitzar els diferents elements de l'entorn del vehicle. En el cas dels cotxes de competició autònoms de formula student, aquests elements són els cons que delimiten el circuit. Aquesta tesi proposa una evolució en el mòdul de detecció de cons de l'equip de formula student BCN emotorsport. El nou sistema de detecció consta de tres etapes: filtratge, segmentació i classificació. Els resultats obtinguts mostren una millora significativa tant en la precisió com en el rang de detecció.

Resumen

En cualquier sistema de conducción autónoma, el sistema de percepción es una parte fundamental. Entre otras tareas, este debe ser capaz de identificar y localizar los diferentes elementos del entorno del vehículo. En el caso de los coches de competición autónomos de Formula Student, estos elementos son los conos que delimitan el circuito. Esta tesis propone una evolución en el módulo de detección de conos del equipo de Formula Student BCN eMotorsport. El nuevo sistema de detección consta de tres etapas: filtrado, segmentación y clasificación. Los resultados obtenidos muestran una mejora significativa tanto en la precisión como en el rango de detección.

Summary

In any autonomous driving system, the perception system is a fundamental component. Among other tasks, it must be capable of identifying and locating the various elements in the vehicle's environment. In the case of autonomous Formula Student competition cars, these elements are the cones that define the circuit. This thesis proposes an evolution in the cone detection module of the Formula Student BCN eMotorsport team. The new detection system is divided into three stages: filtering, segmentation, and classification. The results obtained demonstrate a significant improvement in both accuracy and detection range.

Acknowledgements

Primerament, vull agrair al meu tutor, el Josep R. Casas, la seva ajuda en el desenvolupament i la redacció d'aquest treball, així com per aportar un punt de vista més pedagògic a la meva estada a l'equip.

Vull agrair a l'equip de formula student BCN eMotorsport. Sobretot a les seccions de control i electrònica, amb una menció especial al David Pereztol i l'Enrique Donada, amb qui he compartit molts bons moments. També a l'Oriol Pareras, el meu mentor durant el primer any a l'equip, li dec gran part del que he après. Vull agrair especialment als meus companys del departament de percepció: el Pol Resina, l'Òscar Medrano i el Carlos Perez. La seva ajuda ha estat essencial per a la realització d'aquest treball. Sobretot vull agrair al Carlos, per ensenyar-me més del que jo li he pogut oferir.

Agrair a tots els meus companys d'universitat, per haver fet aquests anys més lleugers i agradables. Especialment a l'Arnau Bergas, amb qui he compartit quatre anys de carrera, dos anys a l'equip i una molt bona amistat. També als meus amics i amigues de tota la vida, els quals han aconseguit fer-me desconnectar i oblidar, encara que fos per breus moments, tota la càrrega de feina. I, finalment, als meus pares i germans per tot el recolzament que m'han donat i per haver suportat pacientment la meva absència durant el temps lliure i estius en els dos anys que he estat a l'equip.

Revision history and approval record

Revision	Date	Author(s)	Description
1.0	22/02/2024	VMB	Document creation
1.1	27/05/2024	JCP	Error correction
2.0	27/05/2024	VMB	Revised after review
2.1	18/06/2024	JCP	Error correction
3.0	18/06/2024	VMB	Final version

DOCUMENT DISTRIBUTION LIST

Role	Surname(s) and Name
Student	Moreno Borràs, Víctor
Project Supervisor	Casas Pla, Josep R.

Written by:		**Reviewed and approved by:**	
Date	18/06/2024	Date	19/06/2024
Name	Víctor Moreno Borràs	Name	Josep R. Casas Pla
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

List of Tables

Abbreviations

DL Deep Learning

DV Driverless

EC Euclidean Clustering

ETSEIB Escola Tècnica Superior d'Enginyeria de Industrial de Barcelona

ETSETB Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

FS Formula Student

FSG Formula Student Germany

ikd-Tree Incremental k Dimensional Tree

IMU Inertial Measurement Unit

kd-Tree k Dimensional Tree

LiDAR Light Detection And Ranging

ML Machine Learning

PCL Point Cloud Library

RANSAC RANdom SAmple Consensus

ROS Robot Operating System

SLAM Simultaneous Localization And Mapping

CHAPTER 1

Introduction

1.1 Context

1.1.1 The competition: Formula Student

Formula Student (FS) is a prestigious international competition held annually in various countries, where university students engage in the design, manufacturing, and racing of Formula-like racecars. Formula Student Germany (FSG) is particularly renowned 1.1. The competition is governed by a stringent set of rules [6] that dictate the specifications of the cars and the regulations of the events in which they compete. It includes three static events and four manually driven dynamic events, as well as their driverless counterparts.

Figure 1.1: *Group photo of FSG 2023*

FS Driverless (DV)

Since 2017, many FS competitions such as FS Germany, FS Spain, or FS Czech added a new modality apart from combustion cars and electric cars: autonomous cars. Teams are expected to design, build and code single-seater cars that can be both driven by a human pilot and driven autonomously.

Dynamic events

The competition includes both static and dynamic events. The static events are mainly focused on the design and business plan of the car and the team. On the other hand, the dynamic events are the most exciting part of the competition, where the cars compete in different driving scenarios. The DV dynamic events are the following:

- Acceleration DV: Achieving maximum acceleration along a 75-meter straight track.
- Skidpad DV: Driving an ∞-shaped track, completing two right and two left turns.
- Autocross DV: Completing a lap on an unfamiliar enclosed track.
- Trackdrive: Completing 10 laps on an unfamiliar enclosed track.

Big Orange *(BO)* | Small Orange *(SO)* | Small Yellow *(Y)* | Small Blue *(B)*

Figure 1.2: *Cones types in FSG [7]*

In all DV events, the track is always delimited by cones, specifically by the yellow and blue ones shown in Fig. 1.2. Consequently, in any Formula Student autonomous system, it is imperative to swiftly locate the position of the cones to successfully complete the events.

As explained before, the competition follows a strict set of rules, which also dictate the points system. The dynamic events are scored mainly based on the time taken to complete the track and the time of the fastest run. The team with the lowest time gets the highest score, and the rest of the teams are scored based on their time difference from the fastest team. However, hitting the cones is also taken into account as penalty. Besides, if an event is not completed, the team is not scored. Therefore, to achieve a good overall result in the competition, it is more advantageous to have a reliable system and complete all events rather than attempting to be faster and risking hitting cones, not finishing the event, and earning no points.

1.1.2 The team: BCN eMotorsport

In 2018, the first Spanish FS Driverless team, based in Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB), was created: Driverless UPC. In its first year, the team converted an old prototype from the Escola Tècnica Superior d'Enginyeria de Industrial de Barcelona (ETSEIB) Motorsport team into the first driverless vehicle of the country that took part in a FS competition. In 2020, Driverless UPC and ETSEIB Motorsport merged under the name of BCN eMotorsport.

Figure 1.3: *At the left cat15x, 2022-23 season car, at the right cat16x, current season car*

The team is organized into 8 departments, responsible for the different systems of the car: Aerodynamics, Chassis, Vehicle Dynamics, Electronics, Powertrain, Vehicle Controls, Management, and finally Perception, where this final degree project has been carried out. The Perception department is responsible for developing and maintaining a system capable of capturing and recognizing the vehicle's environment.

1.1.3 The project: Autonomous System

Similar to the way the team is structured, the Autonomous System is divided into two subsystems: Perception and Control, as depicted in 1.4. The primary function of the Perception system is to analyze sensor data to understand and interpret the car's surroundings; in other words, it is the car's eyes. On the other hand, the Control subsystem, based on the information provided by Perception, issues the car's control commands. In summary, the goal of Perception is to determine the track limits and pass this information to Control.

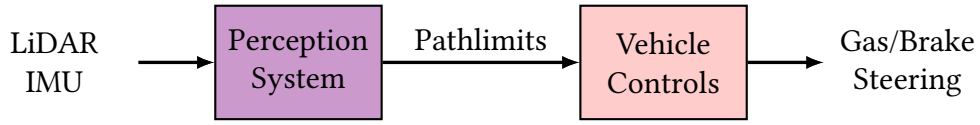


Figure 1.4: *Autonomous system pipeline*

1.2 Motivation

In this context, during the 2022-23 academic year, I joined the team as a member of the Perception department. In the following season, 2023-24, I was promoted to head of the section. Over these two years, I have primarily focused on the development and improvement of the cone detection module within the Perception system.

In my first year with the team, apart from learning the basics of the system, I was able to understand the limitations of the previous approaches and identify potential improvements. Throughout the second year, I have implemented and tested some of these improvements, and I have compared them with the previous system, the baseline. Furthermore, thanks to the experience and knowledge acquired during the degree, I have been able to turn these improvements into something real and valuable, with tangible results in the competitions.

This document reflects part of the work carried out over two years during my time with the team. Specifically, it documents the ideas and improvements I have contributed to the cone detection system compared to the previous system during my stay in the section. In this way, it aims to facilitate future generations of the team in understanding the technical direction taken, the reasons behind it, and the possible improvements remaining, thus promoting the fundamental basis upon which the team is built: the transfer of knowledge across generations of Perception members.

In summary, this document is not only a record of the changes and improvements in the detection system within the department's timeline but also aims to serve as the basis for future modules that will enhance the current system, ensuring the continued development of the section.

Figure 1.5: *Cat16x*

1.3 Work goals

1.3.1 General objective

The main objective of this project is to develop a reliable, accurate, and fast cone detection module for the autonomous system of the BCNeMotorsport team car. Additionally, the system must outperform the baseline presented in 2.3.2, be tested under real conditions, and be able to compete in the dynamic DV events of the FS competitions.

1.3.2 Specific objectives

1. Design and implement a new point cloud filtering algorithm.
2. Design and implement a new point cloud segmentation phase.
3. Train and integrate classification models into the global system.
4. Implement a validation module to obtain performance metrics for both the baseline and the proposed approach in this document.
5. Validate and compare the enitre system with the baseline.

1.4 Requirements and specifications

1.4.1 Project requirements

- The code must be simple, modular, easy to understand, and well documented so that future team members will be able to maintain the code.
- The segmentation stage must run fast and detect almost all the cones of the track.
- The classification model must be able to reduce the number of false positives from the previous stage, without reducing the total recall.
- The module must be more efficient and have better metrics than the baseline.
- The cone detection pipeline must be able to run in a test with the autonomous car.

1.4.2 Project specifications

- The output of the clustering-based segmentation phase must have a Recall > 80 % in a 20 m range.
- The output from the classification phase must have a Precision > 85 % and a Recall > 80 % in a 20 m range.
- The average inference time of the entire module must be less than <100 ms.

1.5 Work plan

As shown in 1.6, the project has not only been developed during the TFG enrollment period but also throughout the entire academic year 2023-2024, starting in October 2023. There have not been any significant changes in the initial schedule.

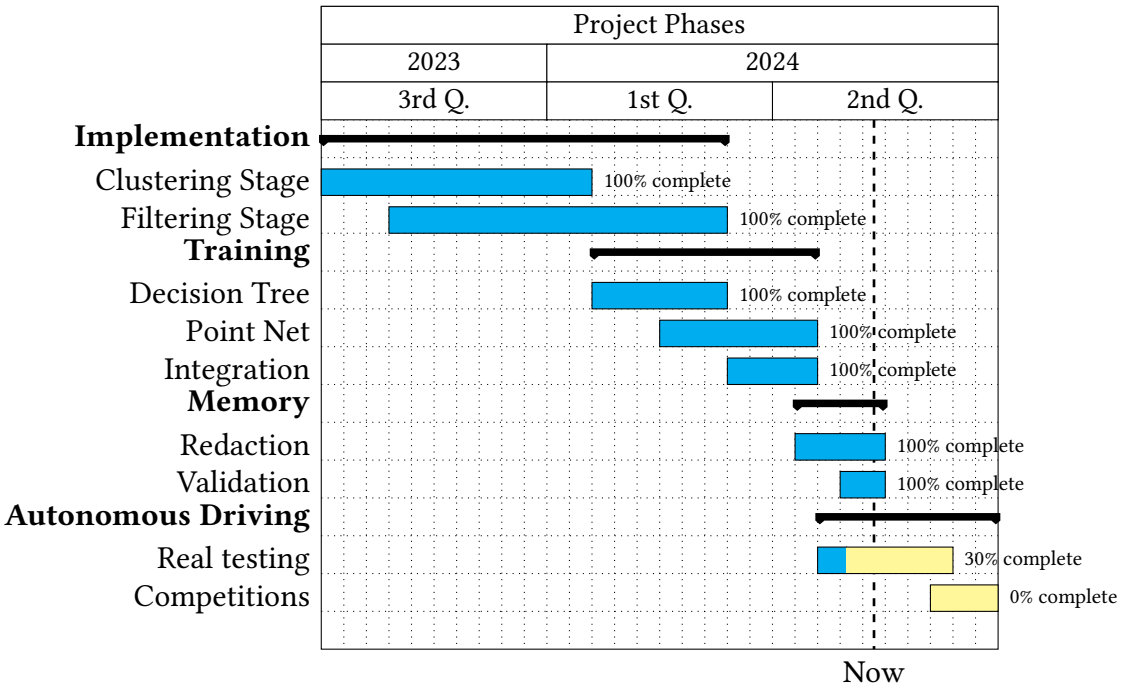


Figure 1.6: *Gantt diagram of the project*

1.6 Document structure

From this point on, the document is structured into three main sections. Firstly, in chapter 2, the concepts and ideas that will allow the reader to understand the foundations upon which the project is based are introduced. Specifically, certain useful concepts are defined. Furthermore, the complete perception system is presented in detail, and both the approaches taken by other FS teams to the problem and the previous approach used by the team are explained.

Secondly, we have the main chapter of the document, chapter 3. Here, we can find in detail, first the formulation and description of the problem, followed by the overall pipeline structure proposed. Then, a breakdown of all the subparts of the module is explained point by point.

Last but not least, in chapter 4, the results of the project are presented and discussed, comparing them with the baseline.

CHAPTER 2

Prelude to Methodology

2.1 Definitions

2.1.1 Point Cloud

A point cloud, denoted by $\mathcal{P}$, is a set of data points representing the external surface of an object or scene in a coordinate system. Formally, a point cloud is an unordered set in the $\mathbb{R}^3$ space that can be mathematically defined as:

$$\mathcal{P} = \{\boldsymbol{p}_i \mid \boldsymbol{p}_i \in \mathbb{R}^3, i = 1, 2, \ldots, n\} \tag{2.1}$$

where $\boldsymbol{p}_i = (x_i, y_i, z_i)$ are the Cartesian coordinates of the i-th, point and n is the number of points such that $n = |\mathcal{P}|$. In some cases, point clouds may exist in higher-dimensional spaces, where an additional dimension is used to encode other attributes such as intensity, RGB or point normals.

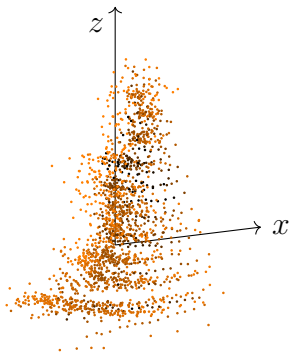


Figure 2.1: *Cone point cloud example*

Point clouds, unlike structured images, lack inherent spatial relationships due to their unordered nature, making analysis challenging. In an image, the data structure itself inherently has "spatial" relationships between pixels, i.e., the adjacency information between pixels is intrinsic to the image itself. Conversely, in a point cloud, the i-th point is not necessarily close to $\boldsymbol{p}_{(i+1)}$ or to $\boldsymbol{p}_{(i-1)}$.

2.1.2 Tree data structures

To address this, various data structures, such as Octrees and kd-Trees, are employed to organize point cloud data spatially, enabling efficient neighbor searches and proximity-based operations. In this project, a kd-Tree has been used for segmenting the point cloud into clusters based on radial searches.

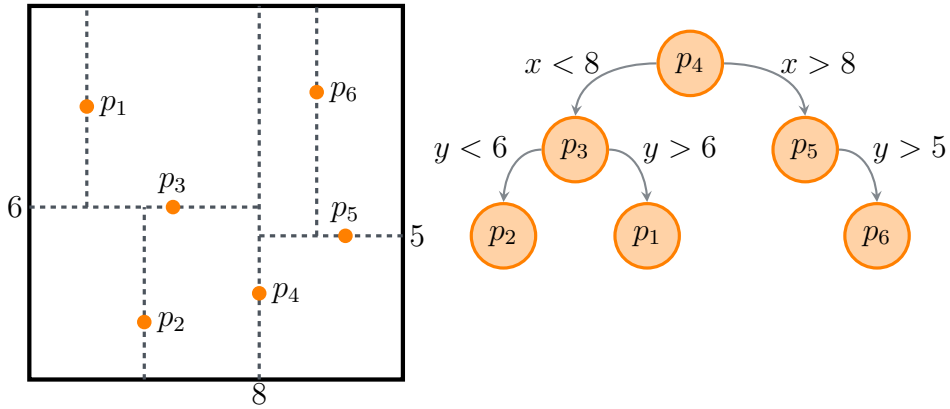


Figure 2.2: *kd-Tree: left shows point partitioning, right shows tree structure*

A k Dimensional Tree (kd-Tree), introduced by Bentley in 1975 [1], is a hierarchical data structure used for organizing points in a k-dimensional space. It is particularly useful for efficient multidimensional searches, such as nearest neighbor searches and range queries. Constructing a kd-Tree involves recursively partitioning points in space to form a binary tree. Starting with an axis selection, the process identifies a median point along the axis to split the space, dividing points into two subsets based on their positions relative to this splitting line. This splitting continues recursively for each child node, with a new axis chosen each time. An ilustrative example of a kd-Tree created from a point cloud with $n = 6$ points such that $\boldsymbol{p_i} = (x_i, y_i)$ can be seen in Fig. 2.2.

In addition to the kd-Tree, a variant known as the Incremental k Dimensional Tree (ikd-Tree) [3] is also employed. Specifically, the ikd-Tree is used in the retrieval of clusters from the original map 3.3.2. The fact that the points map is updated each time a new frame is received makes the ikd-Tree a suitable choice for this task, as it is capable of efficiently updating a kd-Tree without the need to rebuild a new one from scratch.

2.1.3 Statistical descriptors

Another useful mathematical tool used in this project is the statistical descriptors. In order to classify a set of points, such as the grid-wise or the cluster-wise classification, it is necessary to featurize the full information available and condense it into a set of features $\mathbf{D}$. In this section, the ones used in this project are explained:

$$\mathbf{D} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\kappa}\} \tag{2.2}$$

where $\boldsymbol{\mu}$ is the mean, $\boldsymbol{\sigma}$ is the standard deviation, $\boldsymbol{\gamma}$ is the skewness, and $\boldsymbol{\kappa}$ is the kurtosis.

Given a set of points $\mathcal{P} \in \mathbb{R}^{n\times 3}$, where n is the number of points, the mean $\boldsymbol{\mu}$ and the standard deviation $\boldsymbol{\sigma}$ both in $\mathbb{R}^3$ are defined as:

$$\boldsymbol{\mu} = \mathbb{E}[\mathcal{P}] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{p}_i \qquad \boldsymbol{\sigma}^2 = \mathbb{E}[(\mathcal{P} - \boldsymbol{\mu})^2] = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{p}_i - \boldsymbol{\mu})^2 \tag{2.3}$$

These descriptors provide information about the central tendency and dispersion of the set of points in the space. On the other hand, the skewness $\boldsymbol{\gamma}$ and the kurtosis $\boldsymbol{\kappa}$ in $\mathbb{R}^3$, are used to characterize the shape of the distribution. They are defined as:

$$\boldsymbol{\gamma} = \mathbb{E}\left[\left(\frac{\mathcal{P} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}\right)^3\right] = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathbf{p}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}\right)^3 \tag{2.4}$$

$$\boldsymbol{\kappa} = \mathbb{E}\left[\left(\frac{\mathcal{P} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}\right)^4\right] = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathbf{p}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}\right)^4 - 3 \tag{2.5}$$

As depicted in 2.3, skewness measures the asymmetry of the distribution, while kurtosis measures the peakedness or the flatness of the distribution. Upon a more detailed analysis of the interpretability of these values, we can affirm the following: Firstly, for null values of γ and κ, the set of points resembles a normal distribution. Conversely, positive values of γ yield a positively skewed distribution, meaning the tail of the distribution is longer on the right side. In contrast, negative values result in the opposite. On the other hand, positive values of κ indicate a peaked distribution (leptokurtic), whereas negative values suggest a flat distribution (platykurtic).

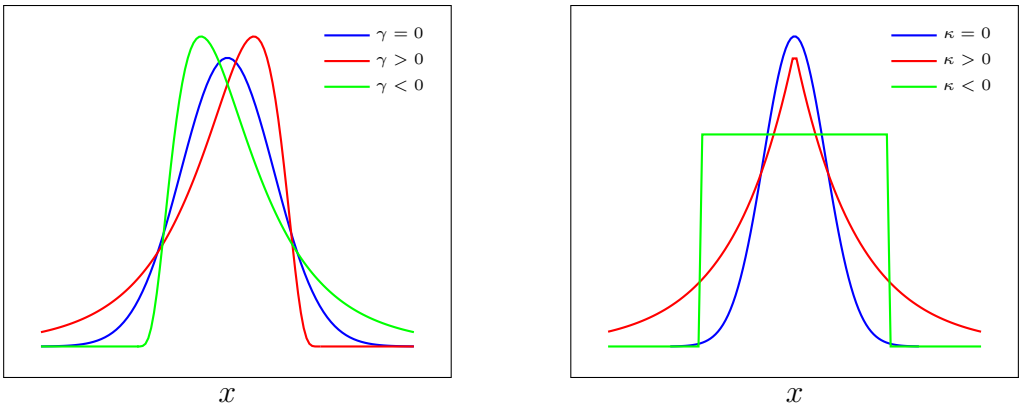


Figure 2.3: *Skewness and Kurtosis examples*

Computing these descriptors for a large number of points can be computationally expensive. To address this, the Welford algorithm and its generalizations are used. The Welford algorithm is a method for computing the central moments of a set of points in a numerically stable way. It allows the computation of the mean, variance, skewness, and kurtosis online, that is, without storing all the points.

2.2 Perception system

To precisely understand the cone detection module, it is necessary to know its context within the Perception system. Below, all the parts of the system are explained in detail.

2.2.1 Sensors

The Perception System uses a Light Detection And Ranging (LiDAR) and an Inertial Measurement Unit (IMU). A LiDAR is a device that measures the distance to a target object by timing the return of reflected laser light and typically transmits multiple rotating laser beams. The LiDAR used by the team, a Velodyne VLP-32C, has 32 laser beams rotating $\approx$ 10 Hz. An example of a frame provided by it is shown in Fig. 2.4. On the other hand, IMUs are electronic devices that combine accelerometers, gyroscopes, and magnetometers to measure forces, angular rates, and the car's body orientation.

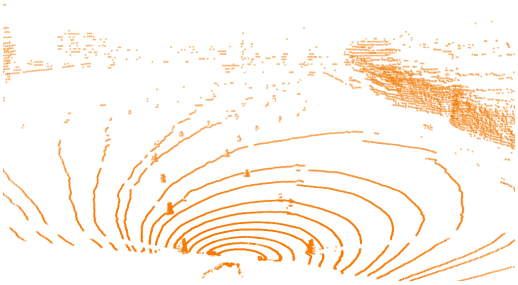


Figure 2.4: *Velodyne VLP-32C frame*

2.2.2 Pipeline

For context, the perception system takes the data received from the sensors as input and provides the track limits as output. The perception system can be broken down into three main stages: SLAM, Cone Detection, and Track Limits Detection, as shown in Fig. 2.5. Each of these stages is explained in detail below.

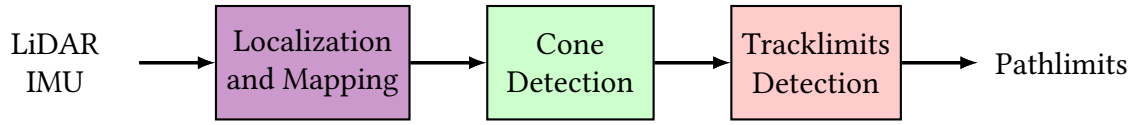


Figure 2.5: *Perception system pipeline*

Simultaneous Localization And Mapping (SLAM)

For localization, LIMO-Velo, an odometry algorithm developed by Andreu Huguet [9], a former team member, is used. Mixing LiDAR and IMU data, LIMO-Velo estimates car position precisely and allows the accumulation of the LiDAR frames. This creates a global point cloud map, such as shown in 2.6. The main advantage of this approach is the increase in range, since the global map has a much higher density.

Cone Detection

Subsequently, cone detection is performed on the obtained global map. This stage is the one studied in this document. As will be explained in detail later, this module performs cone detection using a three-step strategy: filtering, segmentation and classification. In this way, the set of cones on the circuit is obtained.

Track Boundary Detection

Once a set of cones has been detected, the next step in Perception is to obtain the track boundaries, which means classifying the cones into right cones and left cones, as well as assigning an order in the cone list. Currently, the track boundaries are obtained by a Delaunay triangulation-based algorithm developed by Oriol Gorriz, a former team member. It uses Delaunay triangulation and a limited-heuristic-pondered tree search [8].

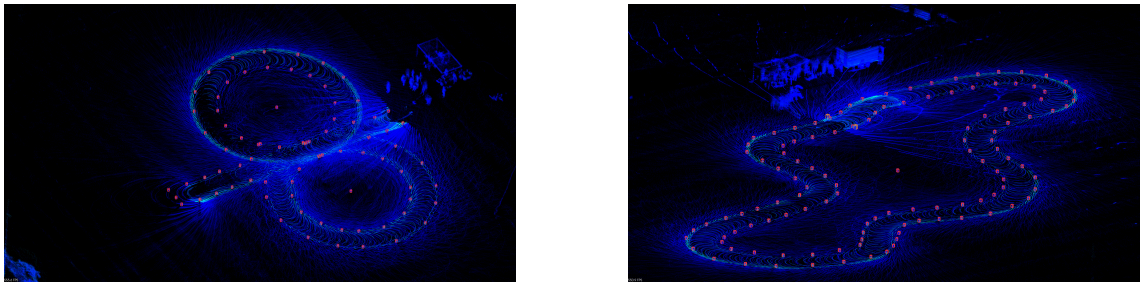


Figure 2.6: *Point cloud map created by the SLAM module, left: Skidpad, right: Autocross*

2.2.3 Framework

All of the car's code is built upon Robot Operating System (ROS). ROS is an open-source framework that provides a set of tools, libraries, and conventions to create robotics software. In our case, ROS is used to communicate between the different modules of the car and sensors. Additionally, the majority of the code that runs on the car is written in C++. For point cloud processing, the Point Cloud Library (PCL) [17] is used. PCL is an open-source library that provides a set of tools for processing 2D/3D point clouds. Finally, for the cones classification, two Python libraries have been used: scikit-learn and PyTorch.

2.3 State of the art and foundations

The cone detection module is a fundamental part of the perception system of any FS autonomous car. However, this type of module is highly specific to this particular application. Consequently, there is limited literature available for comparison with the system. In this section, we aim to discuss both the approaches of other FS teams and the previous approaches adopted by our team.

2.3.1 Formula Student environment

Because FS is a competition, there's limited information available about other teams' approaches. However, some teams have published papers. There are two main types of systems in the FS environment: camera-LiDAR based and LiDAR-only based. In the first approach, both cameras and LiDAR detect the cones separately and later the proposals are fused. In the second approach, the LiDAR is fully responsible for all the detections.

Regarding the LiDAR-only strategy, in most teams, the approach taken is quite similar. Generally, they split the cone detection into three main stages: ground filtering, clustering, and classification. Nevertheless, as explained in the literature of other teams such as AMZ [10] or KIT [14], the majority of the teams work frame by frame. That is because using a global map is not common in the FS environment. Key ideas to highlight from the other teams are that they just filter ground points, mainly using RANSAC, and cone classification is based on geometric heuristic rules.

2.3.2 Previous approaches

In the team's timeline, there is a clear before and after the development of the SLAM module in 2022. This achievement enabled the transition from frame-by-frame detection to accumulating frames into a global map. In the past, two different approaches were taken: LiDAR processing (the baseline) and PCL filtering. Evidently, this work has gathered ideas from the two previous approaches, as well as introducing new proposals.

LIDAR processing: 2020 approach

Before the SLAM module was developed, the cone detection was performed frame by frame. This was the case of the cone detection developed during the 2019-2020 season by the ex-member of the team Arnau Roche. As can be seen in his final master thesis [16], he built a cone detection based on PCL. Specifically, the module first filtered the ground through RANSAC, segmented the cones using Euclidean Clustering and classified them using heuristics. Also, a cluster reconstruction was made before the classification.

PCL filtering: 2022 approach

Once the SLAM module was developed, a new approach was taken by an ex-member of the team, Pau Biosca. Taking advantage of the global map, 2D grid based detection was implemented. The main idea was to project the point cloud into the XY plane and divide it into a grid. Then, through convolutions and heuristics the cones were detected for each cell. However, this approach was not successful as it had a lot of problems, such as losing 3D information.

Due to the fact that the PCL Filtering module didn't work as expected, the department decided to create a new module based on Arnau's approach, but taking advantage of the global map. The result of this is what is explained in this document. Furthermore, as LiDAR processing is more similar to the one developed in this project and the one that had the best results in the team's history, it has been taken as the baseline.

CHAPTER 3

Cone Detection Module

3.1 Conceptual basis

3.1.1 Problem definition

The main goal of the cone detection module is to equip the autonomous vehicle with the capability to accurately perceive and localize all the track cones within their environment.

Mathematically, we can define the target of the module as a set of points representing the real centroids of the cones that delimit the circuit as $\mathbf{\Lambda} = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_n\}$, where $\Lambda_i = (x_i, y_i, z_i) \in \mathbb{R}^3$. The objective is to obtain the set of n cones $\mathbf{\Lambda}$, with the highest precision and in the shortest time possible, so that the car does not leave the track and completes laps as quickly as possible.

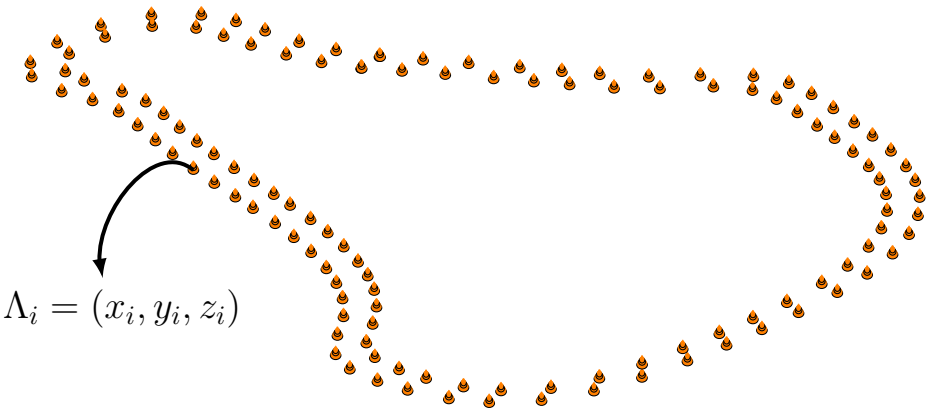


Figure 3.1: *Map of cones*

To obtain the set of cones $\mathbf{\Lambda}$, the module must filter, accumulate, and process LiDAR frames following a series of steps, described conceptually below. The output of the module will be a list of cone proposals, denoted as $\hat{\mathbf{\Lambda}}$, that ideally should be as close as possible to the real set of cones $\mathbf{\Lambda}$, in the shortest time possible.

3.1.2 Pipeline structure

Let $\mathcal{P}_t$ denote a LiDAR frame representing a point cloud collected at instant t. The cone detection module has this frame as input and a list of cone proposals $\hat{\mathbf{\Lambda}}$ as output. The pipeline of the module is divided into three stages: filtering, clustering and classification.

Firstly, $\mathcal{P}_t$ is divided into two subsets through a filtering process, where $\mathcal{F}_t$ and $\mathcal{R}_t$ represent the filtered and removed points, respectively and such that $\mathcal{P}_t = \mathcal{F}_t \cup \mathcal{R}_t$. Then, the filtered point cloud $\mathcal{F}_t$ is stored in a buffer $\mathcal{B}$ containing filtered frames from t to $t - k$. This phase runs whenever a new frame is received, as shown in Fig. 3.2.

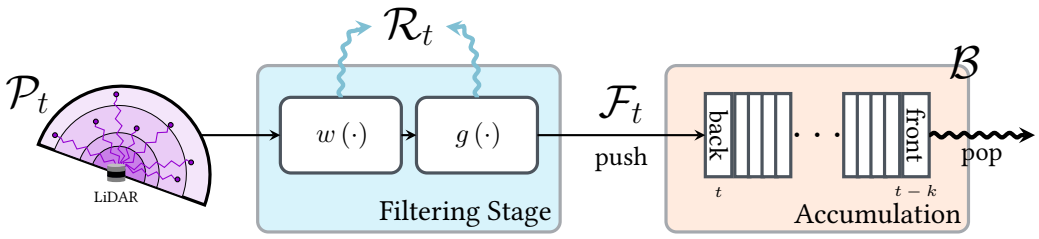


Figure 3.2: *Updating the buffer with the new filtered frame*

It is important to emphasize that the coordinate system, of the input point clouds is global, i.e., the coordinate system's origin is the initial position of the vehicle, instead of the sensor location. This is possible thanks to the SLAM. Consequently, the accumulation of points in the buffer presents no further issues beyond simply merging the points into a single structure without performing any transformations 3.1.2.

$$\mathcal{B} = \bigcup_{l=t-k}^{t} \mathcal{F}_l$$

The next step is to divide the point buffer $\mathcal{B}$ into subparts. That is, we segment the buffer into clusters of points in such a way that we isolate the different obstacles that make up parts of the environment. We will call each of these inividuals set of points as a cluster, denoted as $\mathcal{C}_i$ and we will denote the list of all clusters as $\boldsymbol{\mathcal{C}}$ such that $\boldsymbol{\mathcal{C}} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_c\} \subseteq \mathcal{B}$. Then, optionally each $\mathcal{C}_i$ is reconstructed from an ikd-tree in order to retrieve the lost information through the filtering process.

Once the list of clusters $\boldsymbol{\mathcal{C}}$ has been obtained, the next step is to classify each of these into two classes: cones and non-cones. Based on certain conditions, the list of clusters $\boldsymbol{\mathcal{C}}$ will either be marked as cones or discarded. The conditions a cluster has to meet in order to be considered as cone are given by a previously trained model. Then, by computing the $\boldsymbol{\mathcal{C}}_{cones}$ centroids, the list of cones proposals $\hat{\mathbf{\Lambda}}$ is obtained:

$$\hat{\Lambda}_i = \mathbb{E}[\mathcal{C}_i \mid \mathcal{C}_i \in \boldsymbol{\mathcal{C}}_{cones}]$$

Software

All the code developed has been written in C++. The code follows a modular structure and object-oriented programming paradigm. The pipeline is implemented as a series of classes, where each class represents a stage of the pipeline.

For more information about the software architecture, please refer to the GitHub repository where the code is hosted [2].

3.1.3 Baseline comparison

There are five main changes between the new pipeline and the baseline approach (shown in Fig. 3.3). Firstly, the filtering stage of the baseline just filters the ground points, while the new approach filters both the walls and the ground. Additionally, the ground filtering baseline approach is just based on RANdom SAmple Consensus (RANSAC) algorithm.

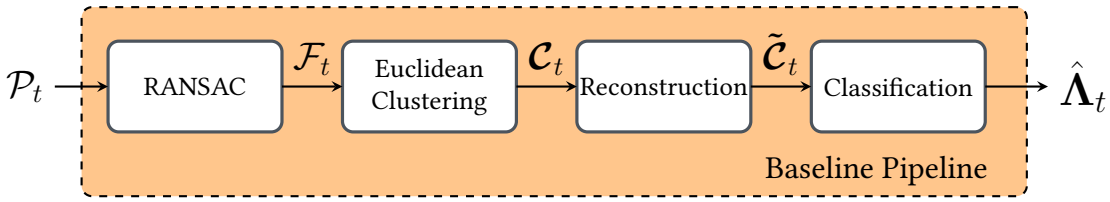


Figure 3.3: *Baseline pipeline*

When the baseline was developed, the module input was directly the LiDAR frame, as there was no SLAM algorithm in the perception system yet. Therefore, the point cloud $\mathcal{P}_t$ received had to be processed independently, without frame accumulation and a list of cone proposals $\hat{\mathbf{\Lambda}}_t$ was obtained for each frame. Furthermore, in the reconstruction stage a new OcTree had to be created for each frame, insted of efficiently using an ikd-tree.

Finally, other differences between the baseline and the approach proposed in this document are the integration of a more efficient implementation of the clustering algorithm and also a new classification approach.

In summary:

1. The filtering stage of the baseline only filters the ground points using RANSAC.
2. The baseline processes each frame independently, without accumulating them.
3. The new approach uses a more efficient implementation of clustering algorithm.
4. The baseline classification uses heuristics, while now a trained model is employed.
5. The new reconstruction stage implementation approach is more efficient and faster.

3.2 Filtering stage

The filtering stage establishes the foundation upon which the pipeline is built. The main goal is to remove the points that are not cones from $\mathcal{P}_t$, so that the subsequent stages can easily identify them. Indeed, this stage not only facilitates the tasks for following phases but also significantly reduces the overall execution time of the pipeline.

The filtering stage follows a two-step approach: walls, and ground filtering. In contrast, the baseline only filters the ground. Additionally, the ground removal is based on RANSAC, whereas in this work, besides studying RANSAC, a surface estimation based ground removal is also proposed. In this section, the new filtering pipeline is explained. Firstly, the key concepts of walls filtering are presented. Then, two different ground filtering strategies are studied. Finally, some details about the accumulation are given.

3.2.1 Walls filtering

In order to filter the walls, a rasterization of the point cloud is performed. Specifically, all the points are projected into a 2D grid, denoted as $\boldsymbol{\mathcal{M}}$, where each cell contains a set of statistical descriptors. Mathematically, the grid $\boldsymbol{\mathcal{M}}$ can be defined as an $n \times m$ matrix. Each element $\mathcal{M}_{ij}$ spans the coordinates $(i \cdot \Delta x, (i+1) \cdot \Delta x)$ in x and $(j \cdot \Delta y, (j+1) \cdot \Delta y)$ in y, where Δx and Δy are the cell sizes in the x and y directions, respectively. Each point $\mathbf{p}$ from $\mathcal{P}_t$ is assigned to a cell $\mathcal{M}_{ij}$, where the indices i and j are calculated as:

$$i = \left\lfloor \frac{p_x}{\Delta x} \right\rfloor \qquad j = \left\lfloor \frac{p_y}{\Delta y} \right\rfloor$$

Additionally, each cell $\mathcal{M}_{ij}$ contains a tuple $(\mu_z, \sigma_z, \kappa_z, n)$. These statistical descriptors are computed online, i.e., each time a new frame is received their points are assigned to the corresponding cell and the statistical descriptors are updated. This means that the grid is a dynamic structure that is updated each time a new frame is received.

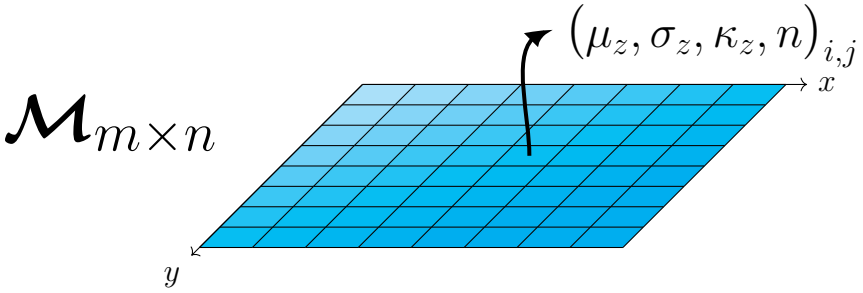


Figure 3.4: *2D Grid that contains statistical descriptors*

Once the descriptors are recomputed with the points of $\mathcal{P}_t$, each updated cell is analyzed to determine if it is a wall cell, taking into account the elevation μ_z and deviation along the z-axis σ_z. This classification creates a binary image $\mathcal{W}$ with 1s representing walls and 0s for the rest.

The probability that a cell neighboring another cell detected as a wall is also a wall is very high, due to the nature of these objects. Therefore, a morphological operator is applied to the image $\mathcal{W}$ to fill in all the gaps and surrounding areas of the cells detected as walls, marking them as walls as well. This operation expands the regions of 1s in the binary image $\boldsymbol{\mathcal{W}}$, effectively filling in gaps and extending the boundaries of detected wall cells. Specifically, the morphological operator applied is dilation, which can be defined as follows: given a binary image $\mathcal{W}$ and a structuring element B, the dilation of $\mathcal{W}$ by B is denoted as $\mathcal{W}' = \mathcal{W} \oplus B$ and is defined by:

$$\mathcal{W}' = \mathcal{W} \oplus B = \bigcup_{b \in B} \mathcal{W}_b$$

where $\mathcal{W}_b$ represents the translation of the image $\mathcal{W}$ by the vector b. In other words, $\mathcal{W}'$ is the set of all cells where the structuring element B overlaps with at least one non-zero cell of the image $\mathcal{W}$. Finally, the points of $\mathcal{P}_t$ that are in a cell marked as a wall in $\mathcal{W}'$ are removed.

3.2.2 Ground filtering

The main idea behind ground filtering is to parametrize the terrain with a model. and then filter out the points that are part of it. In this work, two different strategies are proposed and studied: RANSAC and surface estimation.

RANSAC based

As mentioned, our goal is to model the ground. In the case of RANSAC [5], the model is an ideal 3D plane, i.e., the algorithm seeks the best-fitting plane model $\pi : ax+by+cz+d = 0$, where a, b, c, and d are the parameters of the plane. The algorithm iteratively selects a random subset of points from $\mathcal{P}$, computes the plane parameters (a, b, c, d) that fit these points, and then determines how many points from the entire set $\mathcal{P}$ lie close to this plane. A point is considered part of the plane if the perpendicular distance from the point to the plane is less than a predefined threshold ϵ. The plane that has the largest number of inlier points, Fig. 3.5, is considered the best model of the ground. After identifying the best-fitting plane, the inlier points that are close to this plane are filtered out as part of the ground.

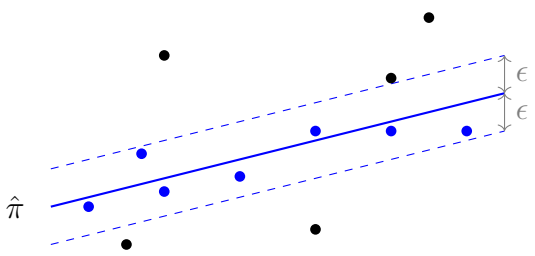


Figure 3.5: *RANSAC algorithm, in blue the inliers*

Surface estimation based

As may be evident, attempting to model the ground as an ideal plane is quite unrealistic, as it does not account for terrain irregularities or elevations. Therefore, in this work, a variant is proposed that attempts to address the problem locally. Relevant literature, such as [13] [12], suggests ground removal by dividing the cloud into patches and determining a plane for each. Similarly, [18] proposes a surface estimation akin to ours. This thesis blends these concepts with our own to achieve ground removal tailored to our context.

The goal is to model the ground as a surface, that is, a function that assigns an elevation value to each point in the XY plane. Formally, this can be defined as a scalar field $h : \mathbb{R}^2 \to \mathbb{R}$ such that $z = h(x, y)$. The objective is to estimate h using $\boldsymbol{\mathcal{M}}$, especifically its mean elevation μ_z. However, not all cells contains just ground points, as some may be formed by obstacles, such as cones. Using the mean elevation of all the cells would lead to a noisy estimation of the surface. In order to deal with this, a two-step strategy is proposed: first, the cells of $\boldsymbol{\mathcal{M}}$ that can be assured to represent the ground are detected. Then, a multivariable interpolation is performed using the obtained values.

For ground cells detection, a heuristics based strategy is used. Similarly to the walls detection, each cell $\mathcal{M}_{ij}$ is analyzed to check if it is a ground cell or not. As explained at 2.1.3 the kurtosis κ is used to determine if the shape of a certain distribution is flat (platykurtic) or not. Therefore, a simple heuristic is proposed: if the κ of a cell is less than a threshold β, with $\beta < 0$, then the cell is considered a ground cell and included in $\mathcal{G}$.

Algorithm 1 Wall and ground cells detection

Input: new frame $\mathcal{P}_t$
Output: $\mathcal{W}'$, $\mathcal{G}$
$\boldsymbol{\mathcal{M}}.update(\mathcal{P}_t)$ ▷ Update the grid with the new frame
for each $\mathcal{M}_{ij} \in \boldsymbol{\mathcal{M}}$ **do**
    if $\mu_z + \sigma_z > \alpha$ **then** $\mathcal{W}_{ij} \leftarrow 1$ ▷ Mark cell as a wall
    else if $\kappa_z < \beta$ **then** $\mathcal{G}_{ij} \leftarrow 1$ ▷ Mark cell as a ground
    else $\mathcal{W}_{ij} \leftarrow 0, \mathcal{G}_{ij} \leftarrow 0$ ▷ Mark cell as neither wall nor ground
end for
$\mathcal{W}' \leftarrow \mathcal{W} \oplus B$ ▷ Apply dilation to the binary image
return $\mathcal{W}'$, $\mathcal{G}$

Once the ground cells $\mathcal{G}$ are identified, a multivariable interpolation is performed to estimate the elevation function $h(x, y)$ using the values of the ground cells $\mathcal{G}$. To achieve this, a simple interpolation methode is used: the nearest neighbors interpolation. For each point (x, y) in the XY plane, the elevation $h(x, y)$ is interpolated based on the average elevation of its nearest neighbors in the ground cells $\mathcal{G}$. Formally, if $\mathcal{N}(x, y)$ denotes the set of nearest neighbors of (x, y), the interpolated value $h(x, y)$ is given by:

$$h(x, y) = \frac{1}{|\mathcal{N}(x, y)|} \sum_{z_j \in \mathcal{N}(x,y)} z_j$$

To smooth the interpolated elevation function and reduce the discontinuities added by the nearest neighbors method, a kernel based low-pass filter is applied. Then, similary to RANSAC, the points of the $\mathcal{P}_{\sqcup}$ frame are considered ground points if the distance between a point and the estimated elevation is less than a threshold, i.e., the point $\mathbf{p}_i = (x_i, y_i, z_i)$ is considered a ground point if $z_i - h(x_i, y_i) < \epsilon$.

As can be observed in Fig. 3.6, a visual analysis reveals that the RANSAC-based version exhibits a lot of noise. In other words, there are points that have not been classified as ground. The result of the frame accumulation is a lot of horizontal streaks of ground. On the other hand, the image on the right shows the accumulation for the surface estimation-based. In this case, the ground segmentation works much better than the RANSAC-based version.

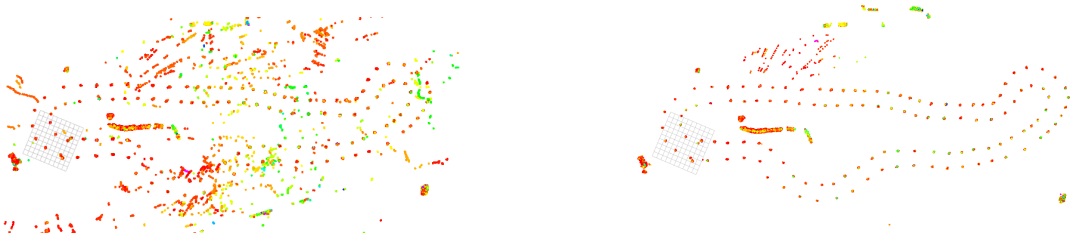


Figure 3.6: *Filtered frames accumulated RANSAC (left) and surface estimation (right)*

3.2.3 Accumulation

The density of points in a LiDAR frame decreases within the distance from the sensor. That is, the further away an object is, the fewer points it will have. So the probability of detecting a cone in a frame at a distance of 30 meters is much lower than at a distance of 10 meter. Therefore, the accumulation of frames is a key step in the pipeline, as it allows us to increase the density of points in the buffer, which in turn increases the probability of detecting cones at a distance.

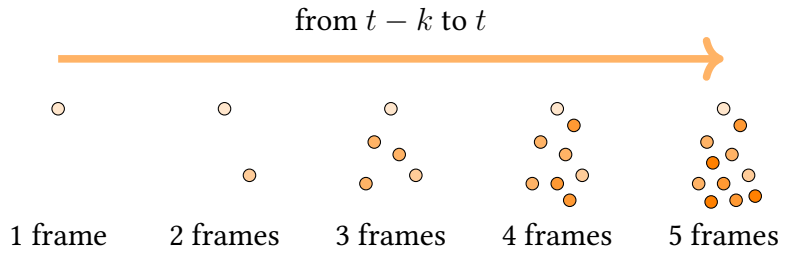


Figure 3.7: *30 meters cone in the buffer through accumulation*

However, accumulating frames increases computational cost and processing time, making it a trade-off between cost and detection probability. There are other cons of the accumulation of frames, such as the increase of the noise points from the previous stage, that leads to a higher number of false positives. So the optimal number of frames is a crucial hyperparameter that must be tuned to obtain the best results.

3.3 Segmentation stage

Once the LiDAR frames are filtered and accumulated, the subsequent stages focus solely on locating the cones on the circuit. Therefore, the goal of the segmentation stage is to divide the point buffer $\mathcal{B}$ into clusters, in order to identify the different obstacles and objects, with the cones being the a subset of these.

In contrast to the baseline, this stage differs mainly in the implementation of the EC algorithm. Moreover, similarly to the baseline, an optional sub-stage has been added to recover the initial quality of the clusters found. Below, these main differences between the baseline and the new approach implementation are explained. Furthermore, the optional clusters reconstruction is briefly explained.

3.3.1 Clustering

The next step is to segment the point buffer $\mathcal{B}$ into clusters $\mathcal{C}_i$. In order to do this, is necessary to use a clustering algorithm. Mathematically, the algorithm could be defined as a function $\mathcal{S}$ that assigns each point in the buffer to a cluster, in such a way that the points in the same cluster are more similar to each other than to those in other clusters according to a certain metric. That is to say that:

$$\mathcal{S} : \mathcal{B} \rightarrow \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_c\} \tag{3.1}$$

Here, the point buffer $\mathcal{B}$ is typically a set of n points $\boldsymbol{p}_i \in \mathbb{R}^3$. The output is a set of c clusters $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_c\}$, where each cluster $\mathcal{C}_i$ is a subset of $\mathcal{B}$, so $\mathcal{C}_i \subset \mathcal{B}$. The function $\mathcal{S}$ maps each point $p \in \mathcal{B}$ to exactly one cluster $\mathcal{C}_i$. As can be seen in Fig. 3.8 example, the points in the same cluster are close to each other according to a certain condition.

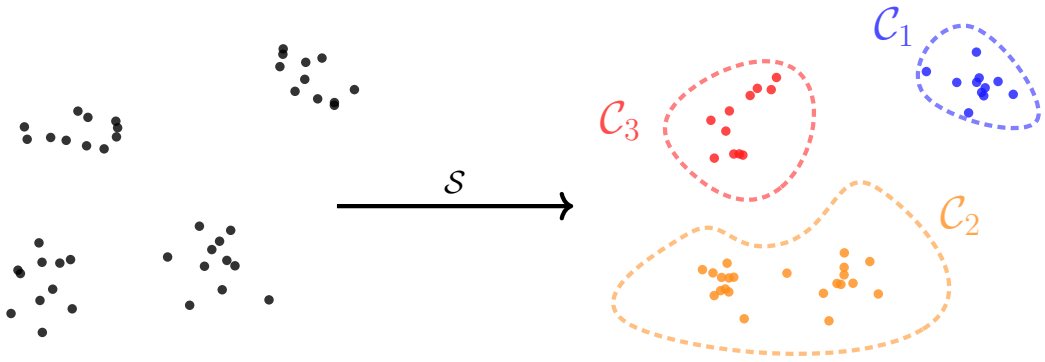


Figure 3.8: *Example of clustering in a 2D space*

There are several clustering algorithms that can be used for this purpose, such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise), K-means, or Euclidean Clustering. After testing different algorithms and observing the results, this project uses the Euclidean Clustering (EC) approach, as well as the baseline approach. The main reason for this choice is that the EC algorithm is the fastest option for our purposes.

Algorithm

Theortically, the EC algorithm can be defined as follows:

$$\mathcal{C}_i = \{\mathbf{p} \in \mathcal{B} \,|\, \exists \mathbf{q} \in C_i,\ \mathbf{q} \neq \mathbf{p},\ \|\mathbf{p} - \mathbf{q}\|_{L2} < \epsilon\} \tag{3.2}$$

This definition states that $\mathcal{C}_i$ consists of all points $\mathbf{p}$ in the buffer for which there exists at least one other point $\mathbf{q}$ in the same cluster C_i, different from $\mathbf{p}$, such that the euclidean distance between $\mathbf{p}$ and $\mathbf{q}$ is less than ϵ.

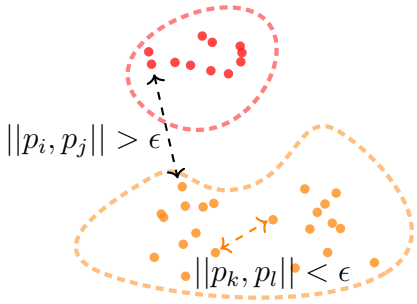


Figure 3.9: *The orange cluster is a merge of two possible clusters due to a large ϵ value*

The main hyperparameter of the EC algorithm is the distance ϵ, which determines the maximum distance between two points to be considered in the same cluster. Finding the optimal value of ϵ is a crucial step in the pipeline, so a small value will lead to a high number of clusters and a oversegmentation of the buffer, while a large value will lead an undersegmentation, like Fig. 3.9. That means that ϵ must be greater that the maximum possible distance between two points in the same cone, but smaller than the minimum distance between two different cones. Otherwise, it will oversegmentated the cone into two different clusters or will count two cones as a single one, respectively.

Another important hyperparameter is the minimum number of points that a cluster must have in order to be considered a cluster. This parameter is crucial to avoid the creation of clusters with a small number of points that could be considered as noise.

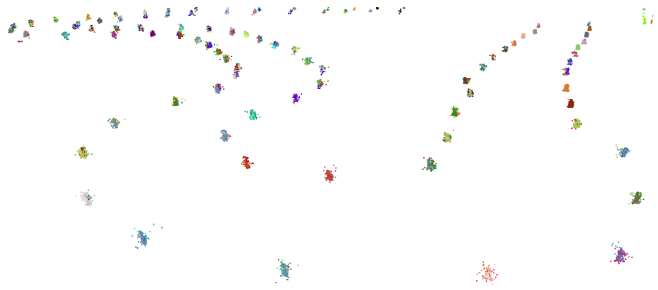


Figure 3.10: *Buffer with colored clusters found by the EC algorithm*

Implementation

As mentioned earlier, both the baseline and the new approach use the EC algorithm, however, its implementations are quite different. The baseline approach uses the PCL library implementation of the algorithm, on the other hand the project uses the implementation proposed by [4] and called Fast Euclidean Clustering, which is more efficient and faster. The implementation of EC PCL is based on a cluster-wise search, meaning that for each point, the entire cluster to which it belongs is searched. On the other hand, the implementation used in this work is based on a point-wise scheme, explained below.

Algorithm 2 Fast Euclidean Clustering

Input: buffer $\mathcal{B}$, distance ϵ
Output: labels $\boldsymbol{l}$

$\mathcal{K} \leftarrow createKdtree(\mathcal{B})$ ▷ Creating a Kd-Tree
$\boldsymbol{l} \leftarrow \text{zeros}(|\mathcal{B}|)$ ▷ Initializing labels to 0
  $l_{\text{current}} \leftarrow 1$

for each $\boldsymbol{p}_i \in \mathcal{B}$ **if** $\boldsymbol{l}_i = 0$ **do** ▷ Main loop
  $\boldsymbol{c} \leftarrow \mathcal{K}.searchradius(\boldsymbol{p}_i, \epsilon)$ ▷ If $\boldsymbol{p}$ is empty, $\boldsymbol{p}_i$ is a noise point
  $l_{\text{min}} \leftarrow \textbf{min}\ (\boldsymbol{l}, l_{\text{current}})$
  MERGINGCLUSTERS($\boldsymbol{c}, l_{\text{min}}$) ▷ Merging clusters loop
  $l_{\text{current}} + +$
end for

return $\boldsymbol{l}$

As shown in Algorithm 2, the output of EC is a list of labels $\boldsymbol{l}$ that assigns each point in the buffer to a cluster. The algorithm starts by creating a Kd-Tree from the buffer $\mathcal{B}$ and initializing the labels to 0, indicating unvisited points. Then, for each unvisited point $\boldsymbol{p}_i$ in the buffer, the algorithm searches for points within a radius ϵ from $\boldsymbol{p}_i$. From this step, the PCL implementation and the new approach differ. The PCL implementation searches the entire cluster to which $\boldsymbol{p}_i$ belongs and does not stop until all the points in the cluster are found. On the other hand, the new approach labels only the points within the radius search and then skips to the next point. Each time a point is found in the search that already belongs to a cluster, the algorithm merges the clusters.

Due to this change in approach, the execution time of the new algorithm decreases drastically. As mentioned in [4], the execution time of the point-wise scheme is 100 times faster than the cluster scheme used in the baseline code.

3.3.2 Reconstruction

Because of the filtering process, the list of clusters $\boldsymbol{C}$ does not contain the full information about the object that each cluster represents. This could affect directly the classification stage, as some important properties for distinguishing cones from other objects could be lost. For addressing this problem, a reconstruction stage like the baseline approach has been implemented. However, this stage is optional and can be disabled if the results of the classification stage are satisfactory without it.

For this purpose, an ikd-tree is used to store this information and perform neighborhood searches efficiently as it is a data structure that allows to add new points efficiently without the need to rebuild it, as it is the case of the normal kdTree (explained in 2.1.2).

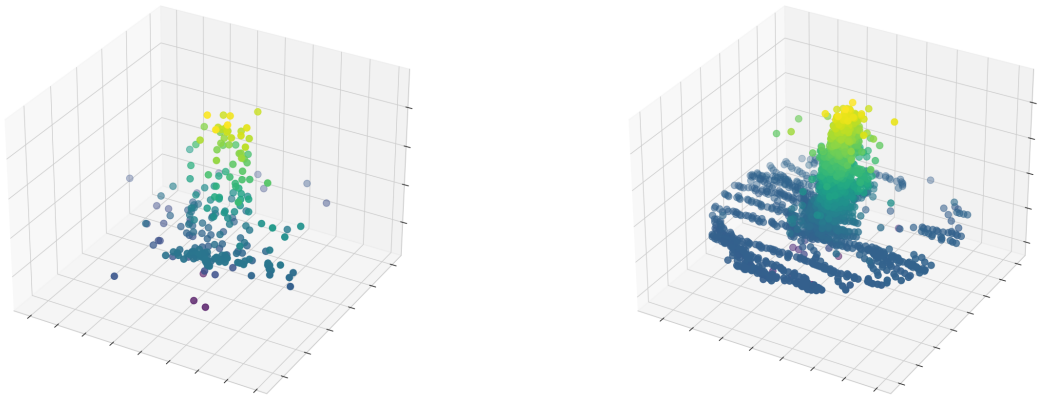


Figure 3.11: *Example reconstruction left: cluster $\mathcal{C}_i$ right: cluster recontruction $\tilde{\mathcal{C}}_i$*

The reconstruction process is based on the following steps: First, each new $\mathcal{P}_t$ frame received is added to the ikd-tree, without filtering it. For storage efficiency reasons, the points are downsampled. Then, for each cluster $\mathcal{C}_i$ found through the EC algorithm, its centroid is calculated. Consecutively, a radius search around this centroid is performed in the ikd-tree to find the cluster reconstruction $\tilde{\mathcal{C}}_i$. The search radius used is 0.4 (greater than the dimensions of an FSG cone) so that the reconstructed cluster also includes the ground area around the cone, thereby better characterizing the cone as a cluster, as can be seen in figures 3.11 and 3.12.

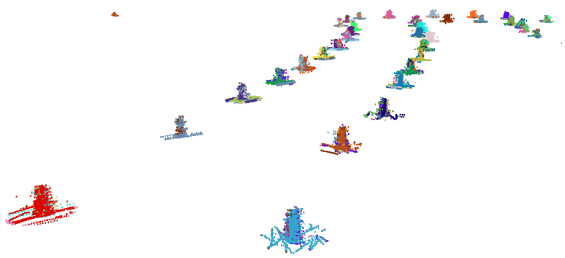


Figure 3.12: *Buffer with colored clusters found by the EC algorithm*

3.4 Classification stage

The final stage of the system is the classification, which is responsible of determining the final output of the module, the cone proposals $\hat{\mathbf{\Lambda}}$. The key idea is to classify each cluster $\mathcal{C}_i$ as a cone or non-cone through a classification model previously trained.

In the past, the baseline classification was based on predefined rules, i.e., human-defined geometric conditions. For example, a cluster was considered a cone if it met certain specific dimensions and had a particular geometric shape. The cones classification using heuristics presents a series of disadvantages and limitations. For instance, cones can be extremely variable in their appearance and location within the point cloud, making it challenging for a human to grasp the full spectrum of possibilities.

To address this problem, two different strategies have been studied with a more modern and suitable approach for the task: a machine learning-based and a deep learning-based approach. In the Machine Learning (ML) approach, clusters are still featurized, but the rules are learned directly by the model, without human intervention. Conversely, the Deep Learning (DL) approach takes this a step further by not selecting specific features, leaving the model to identify the most relevant features from the raw data on its own.

All the data used for training and testing the models was labeled by hand. The dataset structure, how it was created and more details about it are explained in A. Furthermore, the results of this particular stage are shown in the next chapter 4.1.

3.4.1 Machine Learning based

The first strategy involves using a decision tree classifier. Decision trees are a popular ML algorithm that work by recursively splitting the data into binary subsets based on the value of input features. This is done in such a way that the resulting subsets are as pure as possible, i.e., they contain clusters of the same class.

The reasons for choosing a decision tree classifier are its simplicity and interpretability. Additionally, its integration with the rest of the pipeline is straightforward. For this approach, features $\mathbf{D}$ (explained in 2.1.3) were extracted from each clusters $\mathcal{C}_i$ to train the decision tree. Additionally to $\mathbf{D}$, the number of points in the cluster and its distance to the sensor were also used as features. Regarding the hyperparameters, the maximum depth of the tree was set to 3 in order to simplfy the integration, and the rest of the hyperparameters were found through a grid search using the validation set. Another important detail to highlight is the weighting of the two classes in favor of the cone class. The justification for this is that detection is much more crucial than filtering out false positives. In other words, failing to detect a cone is far more critical than detecting a non-cone obstacle as a cone (false positive). In fact, the cone class is given five times more importance compared to the non-cone class.

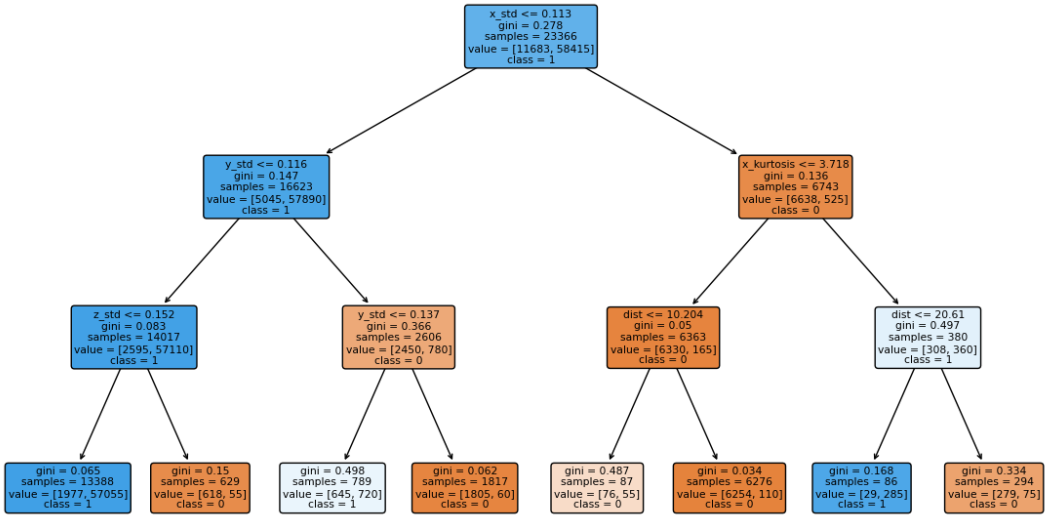


Figure 3.13: *Decision tree classifier example*

3.4.2 Deep Learning based

The second strategy involves using a deep learning model, PointNet [15]. In contrast to the decision tree, PointNet's input is the raw point cloud. Instead of extracting particular features, the models is fed with the raw data and learns the features by itself. The main advantage of this approach is that the model can learn the features that are most relevant for the task, without the need for human intervention.

However, as explained in 2.1.1, the point clouds are unordered sets of points with no inherent structure. This is a problem for most deep learning models, as they require a fixed-size input. To solve this issue, PointNet uses a symmetric function (max pooling) to handle unordered inputs. The network learns to select and encode informative points, with the final layers aggregating these into a global descriptor for shape classification or per-point labels for shape segmentation. Furthermore, the input format allows for easy application of rigid or affine transformations, as each point transforms independently.

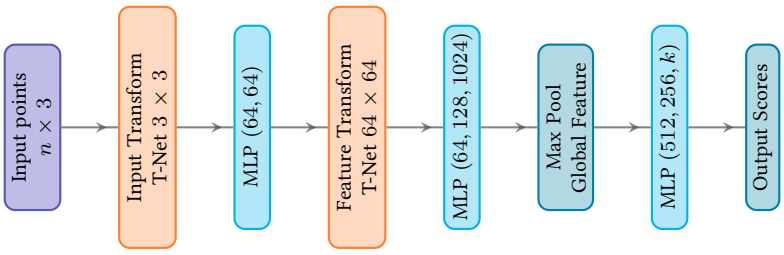


Figure 3.14: *PointNet architecture*

PointNet architecture processes point cloud data starting with input points (nx3) that undergo an input transform network. The transformed points pass through shared MLP layers (64, 64), followed by a feature transform network. The features are then processed by additional shared MLP layers (64, 128, 1024) to create an nx1024 matrix. Max pooling aggregates these into a single 1024-dimensional global feature vector. Finally, this vector is fed through MLP layers (512, 256, k) to output the classification probabilities for each class. In this case, the $k = 2$, as there are two classes: cone and non-cone.

Input data format

As can be seen in Fig. 3.15, the number of points in the clusters is highly variable. In fact, the number of points in the clusters ranges from 4 to 300 approximately and its mean is 62. This is a problem for PointNet, as it requires a fixed-size input. To solve this issue, the clusters are either upsampled or downsampled to a fixed number of points before feeding them to the model. The number of points chosen was 64 points as it is a power of 2 and a good balance between the number of points and the computational cost. For the upsampling, the points are duplicated, while for the downsampling, the points are randomly removed. Additionally, as the clusters coordinate system is global, the points are transalted to the origin setting it as the centroid of the cluster.

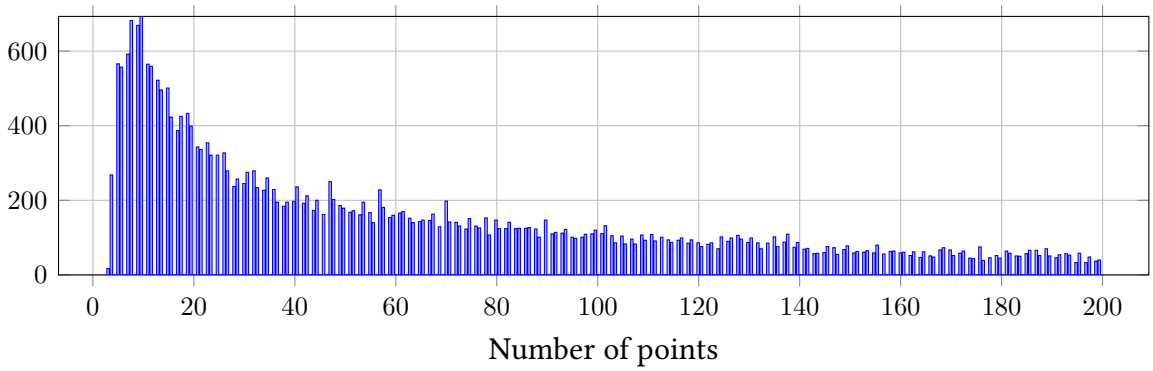


Figure 3.15: *Histogram of the number of points in the clusters*

CHAPTER 4

Results

In this chapter, the results obtained from different system configurations are presented, and an analysis is conducted to determine which of the proposed strategies is the most attractive for integration into the real autonomous vehicle. Additionally, the obtained metrics are compared with those of the baseline: Lidar Processing by Arnau Roche.

4.1 Classification Performance

First of all, the different classification approaches are compared. Additionally, an analysis of the influence of cluster reconstruction on classification is conducted.

The same data have been used for both the decision tree and PointNet. To train and validate the classification models, the number of instances in each class has been balanced to the same value. On the other hand, the test set has not been balanced to verify the model's performance under real conditions, where the number of instances per class is different. In fact, the metrics used include the F1-score, precision, and recall, as well as their respective equivalents weighted by the number of instances per class. For more information about the dataset, refer to Appendix A.

Decision Tree

In the case of the decision tree, it has been trained for four possible scenarios: ground filtering using RANSAC-based or surface-based methods, and adding the reconstruction stage to both cases. The results of the classification using the decision tree can be found in Table 4.1. The table shows the F1-score, precision, and recall, along with their respective weighted measures (marked with w). The versions with reconstruction are marked with an asterisk (*). The magnitude of the rate of instances per class used for testing the models can also be found, where class 1 represents cone and class 0 represents non-cone. For instance, if the value is 2, it means there are twice as many cone instances as non-cone instances.

Firstly, as observed, the surface-based method is less noisy than the RANSAC-based method. The ratio of instances per class shows that the number of non-cone instances is three times lower than the number of cone instances. In contrast, the RANSAC-based method has a ratio of 0.8, meaning that there are more non-cone instances than cone instances. This indicates that the decision tree has an easier task with the surface-based classification case, as there are fewer clusters to classify as non-cones.

Decision Tree	Ratio	F1	Precision	Recall	F1 (w)	Precision (w)	Recall (w)
RANSAC-based	0.8	0.877	0.803	0.967	0.88	0.89	0.88
Surface-based	3	0.941	0.952	0.932	0.91	0.91	0.91
RANSAC-based*	0.8	0.896	0.814	**0.995**	0.88	0.90	0.88
Surface-based*	3	**0.978**	**0.966**	0.990	**0.96**	**0.96**	**0.96**

Table 4.1: *Decision tree results (marked with * the configurations with reconstruction)*

As reflected in the results, both with and without reconstruction, the decision tree performs better with the surface-based method. However, this does not mean that the surface-based method is generally better than the RANSAC-based method, just that it is easier to classify the clusters as fewer non-cone instances are present.

Finally, the reconstruction stage has a positive impact on the classification, as the metrics increase in all cases. In fact, the surface-based method improves the weighted metrics by 5%. Conversely, the improvement in the RANSAC-based method is not as significant. Thus, the reconstruction stage impacts classification performance, but further analysis of its influence on overall performance is needed to determine if it is worth the computational cost.

Another conclusion can be drawn from Fig. 4.1, which shows two plots of the error rate versus the distance to the sensor. The left plot displays the error rate of the decision tree with RANSAC, while the right plot includes the reconstruction stage. In the left plot, the error rate increases with distance, whereas this trend is absent in the right plot. Thus, the reconstruction stage mitigates the distance-related error increase. A future version could employ a hybrid approach, reconstructing only distant clusters where the reconstruction stage is more beneficial.

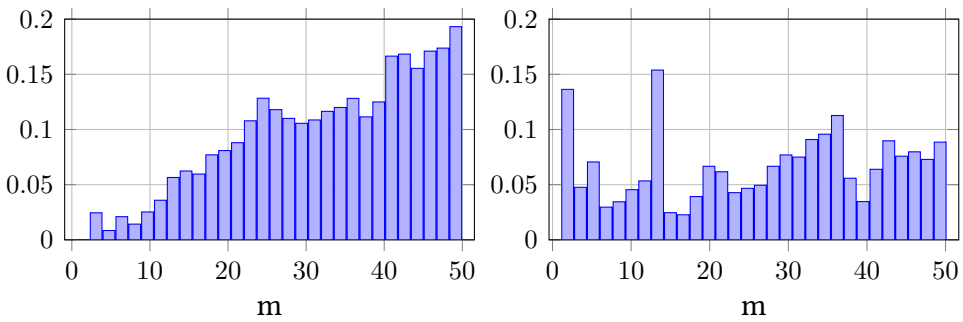


Figure 4.1: *Error rate of the decision tree classifier*

Point Net

The results of the classification using PointNet can be found in Table 4.2. As shown, only the configurations with reconstruction have been trained because it was not possible to achieve learning without it. This means that, without the reconstruction step, PointNet failed to converge and faced overfitting issues. The results indicate that the decision tree outperforms PointNet across all metrics, even when using reconstruction.

Point Net	F1	Precision	Recall	F1*	Precision*	Recall*
RANSAC-based*	0.7	0.68	0.72	0.70	0.70	0.70
Surface-based*	**0.735**	**0.72**	**0.75**	**0.72**	**0.72**	**0.72**

Table 4.2: *Point net results (marked with * the configurations with reconstruction)*

For these reasons, after training and testing the model, it was decided to discard PointNet as a candidate for the classification stage. Therefore, in the following sections, the analysis will be performed with just the integration of the decision tree in the system.

4.2 Overall Performance

In this section, the overall performance of the system is analyzed. Since a personalized analysis for the filtering stage cannot be conducted due to the lack of ground truth, its performance will be evaluated within the context of the entire system. A different test dataset was used for this evaluation. For details about the dataset, refer to AppendixA.

Execution Time

First of all, an estimation of the execution time of the different configurations and the baseline is presented in Table 4.3. As the execution time is very variable, the mean value of the execution time and an interval are given.

	Baseline	RANSAC-based	Surface-based
Without walls filtering	-	40±20 ms	30±15 ms
Without reconstruction	-	15±10 ms	10±5 ms
With reconstruction	5±3 ms	30±20 ms	20±10 ms

Table 4.3: *Mean execution time for different modules configurations*

The baseline is the fastest, followed by configurations without reconstruction, and lastly, those with reconstruction. The baseline's speed is expected since it processes each point cloud individually, while the other configurations process an accumulation of frames. However, the difference in execution time is not significant enough to disregard the benefits of frame accumulation.

Regarding reconstruction, it is evident that adding this stage significantly increases the execution time. When comparing RANSAC-based and surface-based approaches, we can observe that the surface-based approach has a lower execution time. This is due to the reduced noise rate resulting from ground filtering, which facilitates the clustering stage, thereby decreasing the overall execution time. Finally, a real influence of the walls filtering on the execution time can be seen, as it reduces the time by more than half.

Overall Anlysis

Finally, the last study of the different strategies is conducted, evaluating the detection globally. To achieve this, a study of the metrics is performed based on the detection range. The study is conducted as follows: let $\mathbf{\Lambda}$ be the set of cones on the track, the ground truth, and $\hat{\mathbf{\Lambda}}$ the set of cones proposed by the module. A study of recall and precision is conducted, conditioned to a maximum range R:

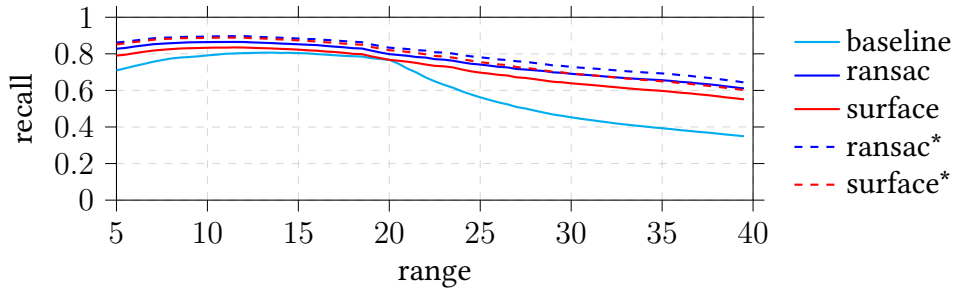
$$\text{metrics}(\mathbf{\Lambda} \approx \hat{\mathbf{\Lambda}} | r < R) \tag{4.1}$$

That is, for each $\hat{\mathbf{\Lambda}}$ received by the module, the detections are checked to determine precision and recall for a maximum range imposed. This maximum range is increased from 5 to 40 meters, and the metrics are calculated for each range. It is important to note that this study does not evaluate the accuracy of the detection in terms of centroid distance. In other words, a detection is considered a true positive if it is close enough to a real cone, regardless of the distance between the centroids.

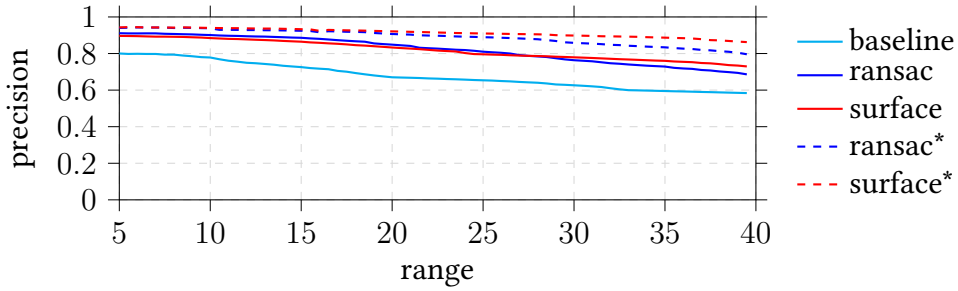
To interpret the results, a brief explanation of the metrics is provided. Recall is the ratio of true positives to the sum of true positives and false negatives, representing the percentage of cones detected by the module. Precision is the ratio of true positives to the sum of true positives and false positives, representing the percentage of correct detections made by the module. Thus, recall evaluates the ability to detect all cones, while precision evaluates the ability to avoid false detections.

Figures 4.2a and 4.2b show the recall and precision curves, respectively, for different configurations of the detection module. The x-axis represents the detection range in meters, increasing from 5 to 40 meters. The y-axis represents the recall and precision values, ranging from 0 to 1.

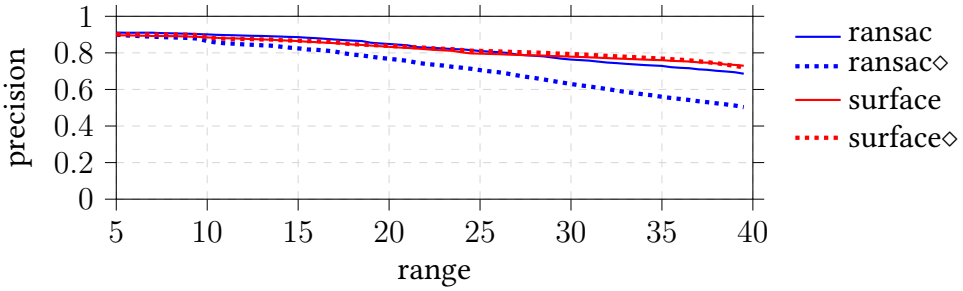
In Figure 4.2a, we observe that recall generally decreases as the detection range increases. The baseline, represented by the cyan line, shows the most significant drop in recall as the range increases, particularly beyond 20 meters. Additionally, its recall values are lower than those of the other configurations. The 'ransac' configuration (blue line) and the 'surface' configuration (red line) maintain similar recall values along the range, with a slight advantage for the 'ransac' configuration. On the other hand, the 'ransac*' (dashed blue line) and 'surface*' (dashed red line) configurations, which include reconstruction, exhibit slightly higher performance than the other configurations, especially at longer ranges where the 'surface' configuration slightly outperforms the 'ransac'.



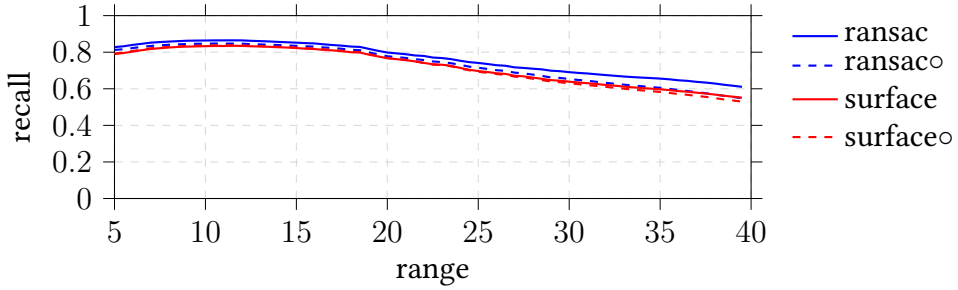
(a) *Recalls curves for different configurations (recontruction marked with * and dashed line)*



(b) *Precision curves for different configurations (recontruction marked with * and dashed line)*



(c) *Precision curves with and without classification (without marked with ◇ and dotted line)*



(d) *Recall curves with and without classification (without marked with ○ and dotted line)*

Figure 4.2: *Precision and recall curves for different configurations*

Figure 4.2b illustrates the precision curves for the same configurations. In this plot, we are evaluating the ability of the module to not detect false cones. Similar to recall, precision tends to decrease as the detection range increases. The baseline configuration again shows a more pronounced decline in precision compared to the other configurations. The 'ransac' and 'surface' configurations demonstrate better precision performance, with more balanced performance between them. The 'ransac*' and 'surface*' configurations again show better performance than the no-reconstruction options.

Lastly, Figures 4.2c and 4.2d show the precision curves for the configurations without classification and the improvement of the walls filtering, respectively. In Figure 4.2c, the influence of the classification stage on overall performance can be seen. In the dotted line, the configurations without classification (marked with $\diamond$) are shown. As mentioned before, in the case of the surface-based method (red line), the classification does not have a significant impact as the number of false positives is low. In contrast, as the RANSAC-based method (blue line) has a higher noise rate, the classification has a more significant impact, especially at longer ranges. In Figure 4.2d, the configurations without walls filtering, marked with $\circ$ and dotted lines, show that the walls filtering has no impact on the recall, i.e., we are not losing any cones despite filtering a part of the point cloud. So with filtering walls, we are reducing the execution time without losing precision.

Overall, the evaluation of the detection strategies reveals that mothodes proposed in this document outperform the baseline. Additionally, others conclusions can be derived from the results. Firstly, the reconstruction stage has a positive impact on the overall performance. Especifically in the precision metric, as it enables to discard false positives cones. Additionally, the classification stage improve cosiderably the precision metric, especially in the RANSAC-based method. Finally, the walls filtering has a huge impact on the execution time without losing precision.

CHAPTER 5

Sustainability Analysis and Ethical Implications

5.1 Sustainability Matrix

5.1.1 Environmental Impact

The environmental impact of the project during its development can be assessed using the following indicator: energy consumption.

Energy consumption

Device	Power (W)	Hours	Consumption (kWh)
Server	500	4.080	23.940
Computer	50	960	308
Total	586	5.200	33.167,6

Table 5.1: *Energy Consumption*

Calculating the energy consumption of all resources involved in the project we get about **3 MWh** of energy were consumed during the development. The main source of energy consumption is clearly the use of a server, which is required to host the dataset and the Gitlab (where the code is stored). Also, all the models have been trained in the server. Its consumption is based on figure 5.1, which corresponds to the power consumption during one week of operation. This translates to a total of **393** $kgCO_2eq$ based on the conversion factor of 131 gCO_2eq/kWh.

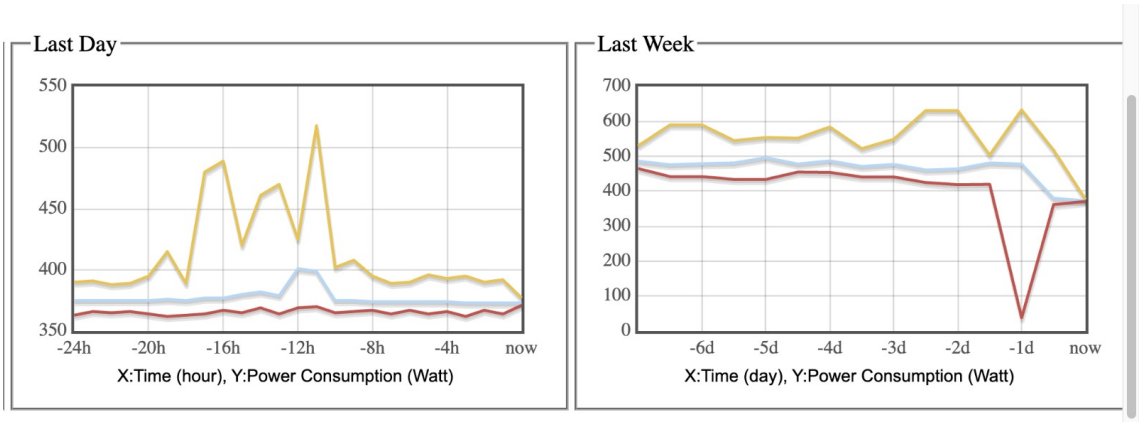


Figure 5.1: *Energy consumption of the server in the 5th of June 2024*

Risks and limitations

While the use of a server is essential for the development of the project, it poses several risks and limitations. First, the high energy consumption leads to significant CO2 emissions, contributing to the carbon footprint. Efforts to mitigate this could include optimizing the code to reduce the number of computational resources needed or utilizing renewable energy sources for power.

Another limitation is the potential for hardware failure, which could interrupt the project and lead to data loss. To address this, regular backups and redundancy measures should be in place.

5.1.2 Economic Impact

Budget analysis

The economic impact of the project has been done for the development:

Topic	Hourly cost (€)	Hours	Expected hours	Cost (€)
Junior engineer	12	960	1.040	24.000
Server	0,30	4.080	43.800	14.364
Total				38.364

Table 5.2: *Budget Analysis*

Human cost has been calculated taking as reference the salary of a full time junior engineer for the development phase and a part-time junior engineer for maintenance during the lifespan of the project. Cost of server operation is based on *Amazon Web Services* price calculator, for a similar server to the one in use.

5.1.3 Social Impact

This project has had and will have an impact mainly on BCN eMotorsport. Additionally, the development of this autonomous driving technology can have broader social implications, such as advancing the field of autonomous vehicles, which can contribute to safer roads and more efficient transportation systems. However, there are also potential negative social impacts to consider. The rise of autonomous technology could lead to job displacement in industries reliant on human drivers.

5.2 Ethical Implications

The development of autonomous driving technology raises several ethical considerations. Key concerns could include ensuring safety and reliability to prevent accidents or protecting privacy by securely handling data collected by the vehicles.

5.3 Sustainable Development Goals

The project has a direct impact on the following sustainable development goals:

- **SDG 9: Industry, Innovation and Infrastructure**. The project is a clear example of innovation in the industry, as it provides a reliable, sustainable and quality solution to a problem.
- **SDG 11: Sustainable Cities and Communities**. The project is a step towards a safer community, as it contributed to autonomous driving, which is a key factor in reducing accidents.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

The project successfully accomplished its general objectives. The proposed cone detection module surpassed the performance of the module developed by Arnau Roche, the baseline. Both range and precision were improved in all the different strategies proposed.

Other conclusions drawn from this work include improved execution time when filtering walls without losing recall. Additionally, there is a substantial improvement in ground filtering using the newly proposed surface-based approach compared to the use of RANSAC. Another conclusion from the results is that the reconstruction stage improves cluster classification. However, this benefit is not worth the longer execution time, especially for the autonomous vehicle where the real-time processing and low latency are crucial. Lastly, classifying clusters through feature extraction followed by a decision tree to distinguish between cones and non-cones is the best strategy. Conversely, PointNet has never improved the decision tree metrics, possibly due to the undefined nature of the non-cone class and its complexity and computational demands which are not as well-suited for real-time applications.

Regarding project specifications, the clustering-based segmentation phase achieved a recall rate of over 80% within a 20-meter range. The output from the classification phase exhibited precision and recall rates exceeding 85% and 80%, respectively, within the same range. Furthermore, the average inference time of the entire module was less than 100 milliseconds, meeting the specified performance criterion.

Overall, the project's successful completion demonstrates the effectiveness of the developed cone detection module, fulfilling the objectives, requirements, and specifications outlined in the project plan.

6.2 Future Directions

There are several areas for future work to further the cone detection module's performance proposed in this project.

- **Partial Reconstruction**: As demonstrated in Fig. 4.1, the classification error rate increases with distance. Thus, the reconstruction stage mitigates the distance-related error increase. A future version could employ a hybrid approach, reconstructing only distant clusters where the reconstruction stage is more beneficial.
- **Alternative Interpolation Methods**: Exploring other interpolation methods for surface estimation, such as the nearest natural neighbor method, where a distance-based weighting is performed to estimate the surface.
- **Point Pillars**: More advanced research could be conducted for end-to-end cone detection using deep learning. For example, using Point Pillars [11], a convolutional neural network for point cloud detection based on PointNet. With the existing dataset, a version of this model could be trained to directly detect cones in the LiDAR frame.

Bibliography

[1] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Commun. ACM* 18 (1975). DOI: 10.1145/361002.361007.

[2] Víctor Moreno Borràs. *EC-thor*. 2024. URL: https://github.com/victhormoreno/EC-thor.

[3] Yixi Cai, Wei Xu, and Fu Zhang. *ikd-Tree: An Incremental K-D Tree for Robotic Applications*. 2021. DOI: 10.48550/arXiv.2102.10808.

[4] *Drones | Free Full-Text | FEC: Fast Euclidean Clustering for Point Cloud Segmentation*. URL: https://www.mdpi.com/2504-446X/6/11/325.

[5] Martin A. Fischler and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Commun. ACM* 24 (1981). DOI: 10.1145/358669.358692.

[6] *FS-Rules 2024 v1.1*. 2024. URL: https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.1.pdf.

[7] *FSG24 Competition Handbook v1.1*. 2024. URL: https://www.formulastudent.de/fileadmin/user_upload/all/2024/important_docs/FSG24_Competition_Handbook_v1.1.pdf.

[8] Oriol Gorriz Viudez. "Enhancement of a Formula Student car perception system using a global 3D map". Bachelor thesis. Universitat Politècnica de Catalunya, 2022. URL: https://upcommons.upc.edu/handle/2117/376095.

[9] Andreu Huguet Segarra. "LIMO-Velo: A real-time, robust, centimeter-accurate estimator for vehicle localization and mapping under racing velocities". Bachelor thesis. Universitat Politècnica de Catalunya, 2022. URL: https://upcommons.upc.edu/handle/2117/365241.

[10] Juraj Kabzan et al. *AMZ Driverless: The Full Autonomous Racing System*. 2019. DOI: 10.48550/arXiv.1905.05150.

[11] Alex H Lang et al. "Pointpillars: Fast encoders for object detection from point clouds". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12697–12705.

[12] Seungjae Lee, Hyungtae Lim, and Hyun Myung. "Patchwork++: Fast and Robust Ground Segmentation Solving Partial Under-Segmentation Using 3D Point Cloud". In: 2022. DOI: 10.1109/IROS47612.2022.9981561.

[13] Hyungtae Lim, Minho Oh, and Hyun Myung. "Patchwork: Concentric Zone-Based Region-Wise Ground Segmentation With Ground Likelihood Estimation Using a 3D LiDAR Sensor". In: *IEEE Robotics and Automation Letters* (2021). DOI: 10.1109/LRA.2021.3093009.

[14] Sherif Nekkah et al. *The Autonomous Racing Software Stack of the KIT19d*. 2020. DOI: 10.48550/arXiv.2010.02828.

[15] Charles R. Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[16] Arnau Roche López. "LiDAR cone detection as part of a perception system in a Formula student car". Bachelor thesis. Universitat Politècnica de Catalunya, 2019. URL: https://upcommons.upc.edu/handle/2117/168711.

[17] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: 2011, pp. 1–4. URL: https://ieeexplore.ieee.org/abstract/document/5980567.

[18] Nicolai Steinke, Daniel Goehring, and Raúl Rojas. "GroundGrid: LiDAR Point Cloud Ground Segmentation and Terrain Estimation". In: *IEEE Robotics and Automation Letters* (2024), pp. 420–426. DOI: 10.1109/LRA.2023.3333233.

APPENDIX A

Database

For training, validating, and testing the decision tree and PointNet models, a self-labeled dataset has been used. It has been implemented and maintained by Carlos Perez Ruiz, a member of the perception department. It has been labeled by all the perception members using Xtreme1 software. The data is obtained from recorded rosbags from the testing with the vehicle. In these rosbags, all the raw data from both the LiDAR and the IMU is stored. With this data, the entire perception system can be executed and tested in offline conditions.

The way the data is labeled is as follows. First of all, a global map with all the frames processed by the SLAM module is saved in a .pcd file. This file is loaded in Xtreme1, and all the cones of the map are manually labeled. Then, all this data is saved in an SQL database. Approximately, more than 3000 cones have been labeled.

For training and validating the models, the number of instances in each class has been balanced to the same value. On the other hand, the test set has not been balanced to verify the model's performance under real conditions. In the table A.1 can be found all the rosbags labelled and used for the training, validating, and testing the models. All the metrics of 4.1 have been computed using this rosbags. Additionally, other rosbags have been used for the validation of the overall system. These can be found in the table A.2. All the metrics of 4.2 have been computed using these rosbags.

Run	Event	Date	Location	Type
Run 64	Autocross	2023-10-26	Montmeló	Train
Run 5	Acceleration	2023-07-26	Montmeló	Train
Run 14	Autocross	2023-12-16	Martorell	Train
Run 54	Skidpad	2023-07-24	Fòrum	Train
Run 33	Autocross	2023-07-29	Castelldefels	Train
Run 32	Trackdrive (10 laps)	2023-08-01	Fòrum	Validation
Run 2	Autocross	2023-08-03	Fòrum	Test
Run 50	Autocross	2023-10-22	Castelldefels	Test

Table A.1: *Rosbags used for train, validate and test the models*

Run	Event	Date	Location
Run 17	Trackdrive (2 laps)	2023-07-31	Fòrum
Run 20	Trackdrive (2 laps)	2023-07-31	Fòrum
Run 7	Autocross	2023-08-03	Fòrum

Table A.2: *Rosbags used for the validation of the overall system*