A Perception Pipeline for an Autonomous Formula Student Vehicle

Diogo Laranjeiro Amaro Serrão Morgado

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Dr. Pedro Daniel dos Santos Miraldo
Prof. Pedro Manuel Urbano de Almeida Lima

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Dr. Pedro Daniel dos Santos Miraldo
Member of the Committee: Dr. Plinio Moreno López

October, 2021

A Perception Pipeline for an Autonomous Formula Student Vehicle

Diogo Laranjeiro Amaro Serrão Morgado

October, 2021

Declaration:

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the *Universidade de Lisboa*.

Declaração:

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Abstract

In an effort to increase the overall road safety and reduce car accidents caused by human error, the automotive industry has been investing in the development of autonomous vehicles. These need to accomplish all the tasks a human would while driving a common vehicle, which includes being able to perceive and comprehend the environment around the vehicle, in real-time and using a combination of high-tech distance sensors and cameras. Following the trends of the automotive industry, Formula Student, one of the Europe's most established educational engineering competition, introduced the Driverless category, which challenges students to build a high-performance autonomous race car. In such competitions, the tracks through which the autonomous car should navigate are unknown and delimited by different colored cones.

This thesis presents a perception pipeline for a Formula Student car that exploits the best features of each available perception sensor in order to identify the cones and, using custom Convolutional Neural Networks, classify their color. Furthermore, due to the simplicity of the classifiers used and in order to mitigate possible misclassifications, a Nearest Neighbor based cone tracking algorithm that takes into account the color of previous observations, was developed. To test the pipeline's performance, an ablation study was conducted on the methods used to identify the cones and, both the classifiers and the tracking module, were evaluated individually. Results showed that the proposed perception pipeline is able to accurately detect and classify the cones in real-time, even when running the pipeline on CPU, something that proved not to be possible using heavy deep learning object detectors.

Keywords: cone detection, sensor fusion, classification, perception, convolutional neural networks, tracking, autonomous driving, Formula Student.

Resumo

Numa tentativa de aumentar a segurança rodoviária e reduzir os acidentes causados por erro humano, a indústria automóvel tem vindo a investir no desenvolvimento de veículos autónomos. Estes necessitam de realizar todas as tarefas que um ser humano realizaria ao conduzir um veículo comum, o que engloba ser capaz de observar e compreender o ambiente à volta do veículo, em tempo real e utilizando uma combinação de sensores de alta tecnologia, de forma a planear uma trajectória em conformidade. Seguindo a tendência da indústria automóvel, a *Formula Student*, uma das mais estabelecidas competições educativas de engenharia da Europa, introduziu a categoria de *Driverless*, que desafia os estudantes a construir um carro de corrida autónomo de alto desempenho. Nestas competições, as pistas pelas quais o carro autónomo deve navegar são desconhecidas e delimitadas por cones de diferentes cores.

Nesta tese é apresentada uma *pipeline* de percepção para um carro de *Formula Student* que explora as melhores características de cada sensor de forma a identificar os cones e, utilizando redes neuronais convolucionais personalizadas, classificar a sua cor. Para além disso, devido à simplicidade dos classificadores utilizados e de modo a mitigar possíveis classificações erradas, desenvolveu-se um algoritmo de rastreamento de cones baseado no algoritmo *Nearest Neighbor* que tem em conta a cor das observações anteriores. Para testar o desempenho da *pipeline*, realizou-se um estudo de ablação sobre os métodos utilizados para identificar os cones e avaliaram-se, indivualmente, ambos os classificadores e o módulo de rastreamento. Os resultados obtidos mostram que a *pipeline* de percepção proposta é capaz de detectar e classificar os cones com exatidão e em tempo real, mesmo correndo a pipeline apenas em CPU, algo que se mostrou não é possível utilizando detectores de objectos de *deep learning* pesados computacionalmente.

Palavras-chave: detecção de cones, fusão sensorial, classificação, percepção, redes neuronais convolucionais, rastreamento, condução autónoma, *Formula Student*.

Acknowledgements

First of all, I want to thank my supervisor Dr. Pedro Miraldo for all the availability, support and guidance provided during this eleven-month project. I also want to thank Gonçalo Pais and André Mateus for their invaluable help and tips given in our weekly meetings.

Then, I want to thank my family for all the support and conditions given to me throughout these five long years away from home.

A special thanks to all the friends I made throughout this five-year journey. They did not only helped me go through the sleepless nights spent studying and finishing projects but they also helped me to truly enjoy the academic lifestyle.

Last but not least, I want to thank the entire Autonomous Systems department from the FST team for believing in my work and for this incredible journey that we spent together building an autonomous race car.

Contents

List of Figures **xii**

List of Tables **xiv**

Acronyms **xvii**

1 Introduction **1**

1.1 Formula Student Competition 2

1.2 Motivation 5

1.3 Contributions and Outline of the Thesis 6

2 Related Work **9**

2.1 Camera Only Approaches 9

2.2 LiDAR Only Approaches 11

2.3 LiDAR and Camera Approaches 12

3 Vehicle Setup **15**

4 Proposed Perception Pipeline **19**

4.1 Cones Detection 19

4.1.1 Pass-through Filter 20

4.1.2 Ground Removal 21

4.1.3 Euclidean Clustering 23

4.1.4 Cone Reconstruction 24

4.1.5 Cone Validation 26

4.2 Cones Classification 27

4.2.1 Sensor Fusion 28
4.2.2 Cone's Color Classification with LiDAR 32
4.2.3 Cone's Color Classification with Camera 37
4.3 Tracking 39

5 Experimental Results **43**
5.1 Metrics 43
5.2 LiDAR Classification 45
5.3 Camera Classification 49
5.4 LiDAR Processing Ablation Study 51
5.5 Tracking 53

6 Conclusion **57**

List of Figures

1.1 Formula Student Driverless Dynamic Events. 4
1.2 Specifications of the cones used in Formula Student Driverless competitions. 5

3.1 FST09e prototype at Formula Student Germany 2019. 15
3.2 Positioning and Field of View (FoV) of the perception sensors. 17

4.1 Proposed perception pipeline overview. 20
4.2 Pass-through filtration result. 21
4.3 Ground removal result. 22
4.4 Point cloud clustering result. 24
4.5 Ground removal problem. 24
4.6 Cone reconstruction result. 26
4.7 Cone validation result. 27
4.8 3D Bounding boxes created around the cone proposals. 28
4.9 Representation of the pinhole camera model. 30
4.10 Camera intrinsic calibration process. 31
4.11 Projected 2D bounding boxes and resultant crop. 32
4.12 Representation of the camera's Field of View (FoV) problem. 33
4.13 Intensity pattern of the cones. 34
4.14 LiDAR color estimation CNN architecture. 35
4.15 Input examples of the LiDAR color estimation CNN. 36
4.16 Example of the LiDAR CNN output. 37
4.17 Camera color estimation CNN architecture. 38
4.18 Example of the camera CNN output. 39
4.19 Overview of the tracking algorithm. 40

4.20 Example of the tracking results. 42

5.1 Confusion Matrix for binary classification. 44
5.2 Comparison between the number of cones detected using and without using the LiDAR CNN. 47
5.3 Example of Cone Images Before and After Cone Reconstruction. 47
5.4 LiDAR CNN Confusion Matrices. 48
5.5 Camera CNN Confusion Matrices. 51
5.6 Tracking Confusion Matrix. 54

List of Tables

5.1 Class Distribution of the LiDAR CNN Training Dataset. 45
5.2 LiDAR CNN Results. 46
5.3 LiDAR CNN Cones Reconstruction Comparison. 48
5.4 Class Distribution of the Camera CNN Training Dataset. 49
5.5 Camera CNN Average Inference Time. 50
5.6 Camera CNN Results. 50
5.7 Ablation Study Results regarding the Number of Cone Proposals. 52
5.8 Ablation Study Results regarding the Execution Time per Iteration. . . . 53
5.9 Tracking Color Corrections. 55

Acronyms

CNN Convolutional Neural Network. 6, 9, 10, 12, 13, 19, 26, 27, 30, 34, 35, 36, 37, 38, 39, 43, 45, 46, 47, 46, 47, 48, 49, 50, 51, 52, 53, 57

CPU Central Processing Unit. 6, 16, 50, 57

FoV Field of View. 11, 16, 17, 19, 20, 26, 27, 30, 32, 39, 45, 48, 51

FSD Formula Student Driverless. 2, 4, 5, 9, 15, 16

FSG Formula Student Germany. 2, 4, 15

FST Formula Student Técnico. 5, 12, 13, 15, 19, 20, 32, 57

GPU Graphical Processing Unit. 13, 16, 50

LiDAR Light Detection And Ranging. 9, 11, 12, 13, 16, 17, 19, 20, 21, 20, 21, 22, 23, 24, 26, 29, 30, 32, 33, 34, 35, 36, 39, 43, 45, 46, 47, 46, 48, 51, 52, 53, 52, 53, 57

PCL Point Cloud Library. 21, 22, 23

RANSAC Random Sample Consensus. 12, 21, 22

ReLU Rectified Linear Unit. 34, 37

ROI Region Of Interest. 12, 20, 21, 23, 26, 39

SVM Support Vector Machine. 12, 32

Chapter 1

Introduction

The automotive industry is noticeably moving more and more towards a world where human driver interaction is not required, i.e., towards the world of autonomous vehicles. Although road fatalities keep decreasing [40], there are still more than 40 000 deaths on European roads each year. Vehicle manufacturers are introducing at a rapid pace new in-vehicle safety and Advanced Driving Assistance Systems that can intervene before an accident happens. This is a much-needed development, as more than 90% of all accidents are caused by human error [39]. Risky and dangerous driver behaviors such as impaired driving, drugged driving, speeding and distraction can be reduced by reaching to higher levels of autonomy. In addition, it may help increase, or restore, the independence of the elderly and people with mobility and visual impairments.

The increased safety in addition to an improved comfort promised by autonomous vehicles leads the automotive industry to use a big portion of their investments for the development of autonomous driving. When taking the human out of the equation, the vehicle must be able to respond to unknown situations in order to ensure a reliable system. Therefore, it needs to accomplish all the tasks a human would while driving a common vehicle. This includes perceiving the environment, estimating its position, predicting what other road users will do, planning a trajectory accordingly and giving the correct acceleration and steering input to follow that trajectory. The investments made by the industry led us to the current state-of-the-art in autonomous driving, with several companies already having their autonomous vehicle prototypes being tested on real-life scenarios without any human interference.

A key part when dealing with any kind of autonomous driving problem is being able

to perceive and comprehend the environment around the vehicle, in real-time and using a combination of high-tech distance sensors and cameras, combined with state-of-the-art perception algorithms. This is still a very challenging task in autonomous vehicles due to the extremely low acceptable error rate, as it is a crucial task to ensure the safe and reliable operation of the vehicle by being the source of information used by the path-planning and decision-making algorithms that dictate what the vehicle should do or where it should go.

A crucial aspect for an autonomous vehicle to reach its full autonomous capabilities is the ability to operate the vehicle close to its limits of handling, *e.g.*, by performing emergency/avoidance maneuvers or by driving on bad road conditions such as those of a very slippery surface road. When it comes to new technologies in the automotive industry, racing has often played a key role in fuelling innovation and pushing cars to the limit of what is possible. Autonomous racing has proven to be an essential platform to develop, test and validate new technologies under challenging conditions. It provides a unique opportunity to test autonomous driving software such as redundant perception pipelines, failure detection algorithms and control in challenging conditions and on closed tracks, which mitigates the risks of human injury. Competitions, by themselves, allow researchers and developers to test the applicability of different solutions and their robustness. Racing competitions, in particular, present additional challenges related to computational speeds, power consumption, and sensing.

1.1 Formula Student Competition

Formula Student[1] is the Europe's most established educational automotive engineering competition that challenges university students from worldwide top universities to design, manufacture, build and test electric and combustion race cars following a strict set of rules [13] that prioritize safety. These students are then challenged to race their cars and compete against other teams in international competitions. However, the competition is not won solely by the team with the fastest car, but rather by the team with the best overall package of construction, performance and financial and sales planning. Backed by the automotive industry and by high-profile engineers, Formula Student Germany (FSG), one of the most reputable competition organizers, introduced, in a con-

[1] `https://www.formulastudent.de`. Accessible on July 30, 2021.

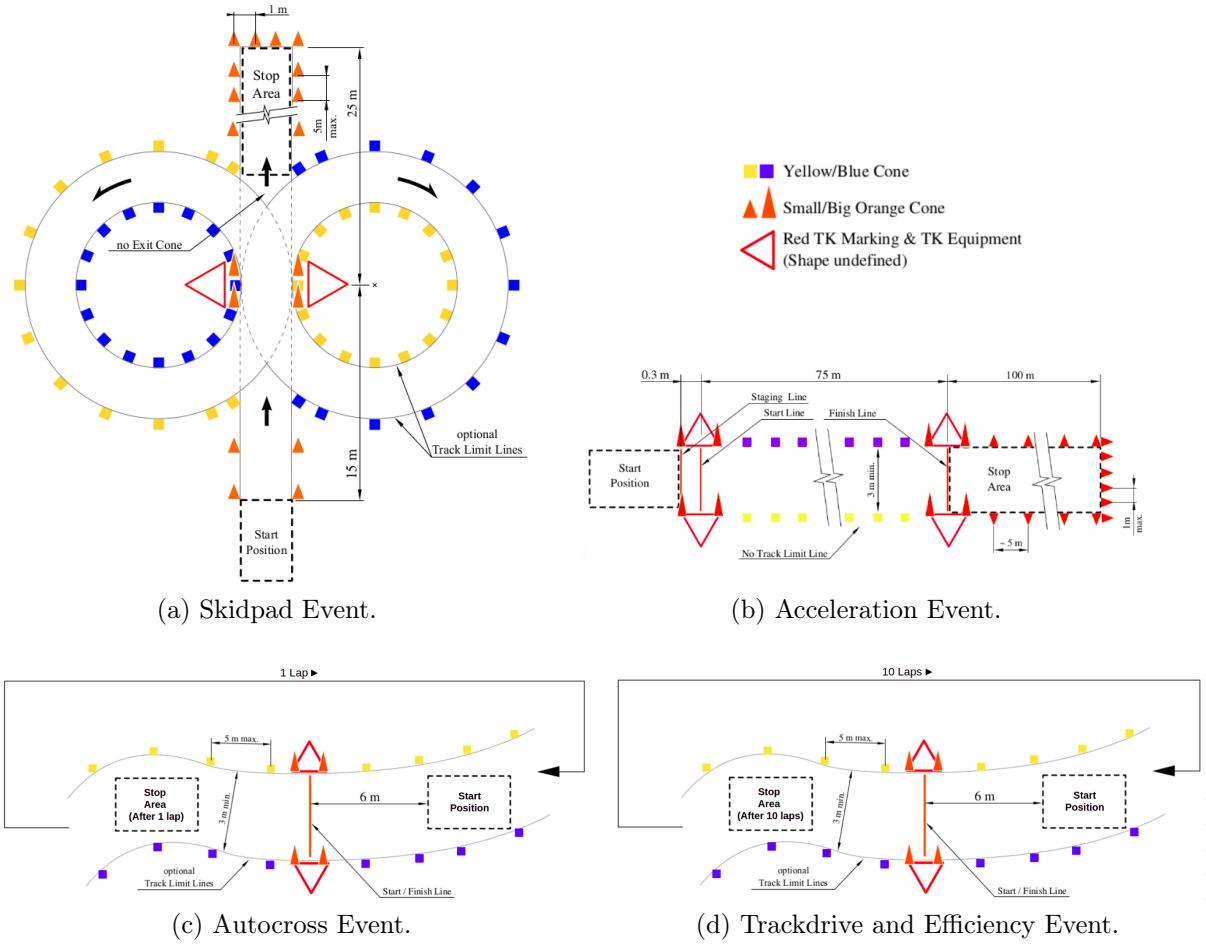
stant effort to follow the trends of the automotive industry world, a new competition class, the Formula Student Driverless (FSD) class. This is a new competition class that challenges students to build a high-performance autonomous race car. In this class, the autonomous student-developed prototypes must be able to compete in different events fully autonomously without any human interaction and without prior knowledge of the track's layout. FSD is remarkably unique by allowing the students to design their cars across the entire stack, *i.e.*, from the car's chassis and electronics layout to high-level perception and motion planning algorithms using state-of-the-art sensors and computing hardware.

The Formula Student competitions take place in famous racetracks around the world and follow a strict set of regulations established by the Formula Society of Automotive Engineers[2] (FSAE) that must be met by every participating prototype. The competitions can be divided into two sets of events: static events and dynamic events. The static events evaluate the technical aspects of the design, conception and sustainability of the prototype, and include an Engineering Design event, Cost and Manufacturing and a Business Plan Presentation that evaluates the team's ability to develop and deliver a comprehensive business model which demonstrates that their product – a prototype race car – could become a rewarding business opportunity that creates a monetary profit. On the other hand, in the dynamic events, the students are able to show the actual racing capabilities of their developed prototype. However, even before the race cars can go on track, a technical inspection puts the students to the test by evaluating their prototype and by making sure that the mechanical, electrical, and autonomous aspects of their prototype are in accordance with the rules, especially those related to safety. The dynamic events comprise a total of 5 events, represented in Fig. 1.1. These events are:

- **Skid Pad (Fig. 1.1(a)):** the car must complete, as fast as possible, two full laps on a track that is delimited by two pairs of concentric circles in an eight pattern figure. Here, mostly the lateral accelerations of the car are tested;

- **Acceleration (Fig. 1.1(b)):** the car must accelerate as fast as possible on a track that consists of a straight line with a length of 75 meters. Here, mostly the longitudinal acceleration is tested;

[2] `https://www.fsaeonline.com/`. Accessible on July 30, 2021.



(a) Skidpad Event.

(b) Acceleration Event.

(c) Autocross Event.

(d) Trackdrive and Efficiency Event.

Figure 1.1: Formula Student Dynamic Events for the Driverless Class vehicles. Figures taken from [14].

- **Autocross (Fig. 1.1(c)):** the car must complete one full lap on an unknown closed-loop track with a length of 200 to 500 meters. The track is composed of several corners and straights, which tests the handling capabilities of the car;
- **Trackdrive and Efficiency (Fig. 1.1(d)):** the car must be able to complete ten consecutive laps around the same track used in the autocross event. Here, the consistency and robustness of the prototype are tested. Besides that, the energy efficiency of the car is also evaluated by measuring the energy spent during the trackdrive event.

As shown in Figure 1.1, for the FSD class the track boundaries are delimited by blue

Figure 1.2: Specifications of the cones used in Formula Student Driverless competitions. Figure taken from [14].

cones on the left and yellow cones on the right, small orange cones delimit stopping zones and big orange cones mark timekeeping zones. The shape, dimensions and color pattern of these cones must always be the same, as shown in Fig. 1.2, and are regulated by the FSG rules [13]. Since the layout of the track is unknown, in order to safely navigate through the unknown environment, the perception pipeline implemented in the prototype must be able to identify the cones' position and color using the data retrieved from the available perception sensors. Having detected the cones that delimit the track, it is then necessary to compute a valid path between the track boundaries, something that is done by the path planning pipeline. Finally, in order to navigate through the track using the computed path one needs to determine a speed target and a steering input, which is performed by the control pipeline. During the dynamic events the only way of communicating with the vehicle is via the Remote Emergency System (RES), which allows to send a starting signal, the "Go Signal", and to stop the vehicle in case of emergency, activating the Emergency Brake System (EBS).

1.2 Motivation

Formula Student Técnico[3] (FST) is the team that represents Portugal and Instituto Superior Técnico (IST) in the international Formula Student competitions that it attends every year. FST has been developing prototypes for this competition since 2001, with ten prototypes developed so far and proven results in the most prestigious competitions

[3] `https://fstlisboa.com`. Accessible on July 30, 2021.

of Europe. In 2019 the team, with more than 60 members from different engineering courses, decided to, in addition to building a new fully electrical prototype, embrace a new challenge: empower the previous electric prototype (FST09e) with autonomous driving capabilities and compete in the FSD class in the summer of 2020. Initially composed of only eight students, the autonomous systems team successfully developed, in less than one year, a complete autonomous pipeline [15], comprising perception, estimation and control pipelines.

Unfortunately, due to the COVID-19 pandemic, the formula student competitions that FST was going to compete in, were canceled. Because of this, an opportunity for extensively testing the developed algorithms emerged. As this was the team's first year to develop an autonomous car, it would be expected to encounter problems during this testing phase, which turned out to be the case. Even though the different pipelines were extensively tested in simulation and using synthetic data, one pipeline that was notoriously hard to test on simulation was the perception pipeline. Due to this, several problems related to the perception pipeline and its computational expensiveness arose during the testing phase. In order for the car to be competitive, it needs to be able to navigate through the track (delimited by yellow and blue cones) at a very fast pace, which means that an accurate, robust and fast perception pipeline that takes advantage of all the available sensors must be developed.

1.3 Contributions and Outline of the Thesis

In this thesis, I propose a new perception pipeline, for an autonomous Formula Student car, that exploits the best features of each available perception sensor to generate cone proposals and, using custom CNNs, classify the image patches created by projecting the cone proposals into the camera image. Furthermore, and due to the simplicity of the CNNs used, a Nearest Neighbor based tracking algorithm is used in order to mitigate misclassifications by taking into account previous observations. The proposed perception pipeline is characterized by its high accuracy and low execution time, which allows it to be run on CPU only, increasing the efficiency score in the Formula Student competitions by taking the high power consumption GPU out of the processing unit.

In Chapter 2 a review is conducted on the methods used on the perception pipelines implemented by other Formula Student teams. In Chapter 3, it is presented an overview

of the Formula Student prototype used alongside the system integration and specifications of the perception sensors added and all the modifications done to the prototype in order to transform it into an autonomous prototype. Chapter 4 sequentially details the proposed perception pipeline, from the identification and generation of the cone proposals, in Sec. 4.1, to the estimation of the correspondent color, in Sec. 4.2, and the developed tracking algorithm, detailed in Sec. 4.3. Chapter 5 presents the experimental results obtained using real data and the results obtained from an ablation study conducted on the methods used to generate cone proposals. Finally, Chapter 6 presents the conclusions on the proposed perception pipeline as well as suggestions for future work.

Chapter 2

Related Work

In this chapter, a literature review is performed on the existing methods used in the perception pipelines implemented by Formula Student Driverless (FSD) teams. Unfortunately, since the Formula Student Driverless competition is still a recent competition, with only three competitions held so far, there is not much literature available regarding the methods used in the implementations of the teams. However, most of the available literature belongs to top-tier teams, whose implementations were considered to be, at the time, the state-of-the-art. Depending on the sensors available to the teams, different approaches can be considered: those that solely rely on either cameras (one or more), described in Sec. 2.1, or LiDARs, described in Sec. 2.2 and those that fuse the information of both sensors, described in Sec. 2.3.

2.1 Camera Only Approaches

The methods used in the implementations that solely rely on cameras are highly dependent on the number and type of cameras used (monocular *vs.* stereo), as depth estimation on stereo cameras can be easily obtained. Regardless of the type of camera used, these implementations start by detecting the cones and estimating their correspondent color, either by using deep learning and state-of-the-art object detectors or using more classical computer vision techniques. Regarding the deep learning techniques, some teams [18, 24] chose to use well-known object detectors such as YOLOv2 [31], YOLOv3 [32] and their correspondent lightweight versions, YOLOv2-tiny [29] and YOLOv3-tiny [1], respectively, while others [37] opted to use custom object detectors, based on

the YOLOv3 architecture, to detect the cones, and a custom 7-layer CNN to estimate their correspondent color. By training these real-time and powerful object detectors to detect cones from four different classes (blue, yellow, orange and big orange), the teams are able to accurately get bounding boxes around the detected cones, along with the confidence scores for each detection.

In order to estimate the 3D position of the detected cones, there is a consensus (among the formula student teams) that, for monocular setups, using a keypoint regression alongside the Perspective-n-Point (PnP) [12] algorithm is the best approach. The known cone's shape and size are exploited to perform a keypoint regression and find specific feature points, on each of the detected bounding boxes, that match their 3D correspondences, whose locations can be measured from a reference frame. A computer vision approach [24] was initially explored in order to extract these keypoints. First, the RGB bounding box of the detected cones is converted to the LAB color space and the "b" channel, which represents the color axis from blue to yellow, is extracted. Then, an Otsu threshold [27] is used to obtain a binary image where, performing contour fitting, three vertices from the top region of the cone can be identified. These three vertices are then extended, using the known dimensions of the cone, to obtain further four more keypoints. This approach has however shown not to be robust in edge cases, which led the teams to use deep learning methods. Here, custom CNNs [9, 37] with a ResNet-based architecture [20] were developed to detect "corner-like" features on the input image – the bounding boxes of the previously detected cones. The detected keypoints are used as 2D points on the image to make correspondences with the respective points on the 3D model of a cone. Using the correspondences and the camera intrinsics, PnP is used to estimate the 3D position of every detected cone. As for stereo camera setups, the 3D position of the detected cones can be estimated using stereo matching algorithms that compute the points' depth by getting the disparity map between the two images.

Regarding the more classical computer vision techniques, two methods were tested by Zeilinger *et al.* [44] in order to detect the cones using a stereo camera. The first one performs a block-matching algorithm [3] on a depth image, which allows to extract and remove 3D planes. Since the track floor is geometrically known, it can be pre-segmented, making cone proposals appear as isolated objects in the image. These proposals are then further evaluated using classical image processing steps on the estimated location in the image plane, similar to the approach taken in [43]. However, in addition to its high

computational cost, it was found to be hard to adapt the block-matching algorithm to use the known structure of the environment. The second method is able to detect cones in both images separately by using an algorithm that computes the disparity, *i.e.*, the distance between two corresponding points in the left and right images of the stereo camera, the z-depth, and, consequently, the 3D position of the cone. In both methods, the color of the cones is heuristically estimated by evaluating the RGB pixel values of the cone's centroid, which should be within a white or a black stripe, and moving downwards until another color (blue, yellow, or orange) is found, identifying the color of the cone.

2.2 LiDAR Only Approaches

From the review conducted on the available literature regarding the perception pipeline implementations that solely rely on the LiDAR, one can conclude that all the implementations [2, 4, 18] have in common their two first 3D LiDAR processing steps. Firstly, the ground plane is removed using an adaptive ground removal algorithm [21] that adapts to changes in the inclination of the ground using a regression-based approach. This algorithm divides the LiDAR's FoV into angular segments and it splits each segment into radial bins from which a line is then regressed using the lowermost points of all the bins in a segment. Finally, all the points that are within a threshold distance to this line are classified as ground points and are removed. The second LiDAR processing step shared by these implementations is the step of clustering the point cloud. Here, the Euclidean Clustering method is used to group the point cloud into clusters by performing clustering extraction in a Euclidean sense, *i.e.*, by computing the Euclidean distance between each of the 3D points of the point cloud and evaluate whether that point corresponds to the cluster or not. Having the cones clustered out, Agarwal *et al.* [2] and Gosala *et al.* [18] apply some heuristic-based filters to estimate the likelihood of the clusters being cones, before going for the cones' color estimation step. First, the dimensions of a bounding box created around the cluster and the elements of the covariance matrix are used to filter candidates. Then, using the dimensions of the cones and the vertical and horizontal angular resolutions of the LiDAR, a rule-based filter is used to check whether the number of points in that cluster is in accordance with the expected number of points in a cone at that distance. Only the clusters that successfully go through these filters are considered to be cones and are forwarded to the

color estimation step.

To estimate the color of the validated cones, all the teams chose to use different deep learning methods. As shown in Fig. 1.2, the cones used in Formula Student can be distinguishable not only by their color pattern but also by their intensity pattern. Yellow cones have a yellow-black-yellow pattern whereas blue cones have a blue-white-blue pattern, which results in differing LiDAR intensity patterns. Gosala *et al.* [18] exploits the differing LiDAR intensity patterns of the cones to develop a custom CNN whose input image corresponds to an image created by mapping the 3D bounding boxes of the validated clusters and whose pixel values store the intensities of points in the point cloud. Similarly, Agarwal *et al.* [2] use binary classification techniques by developing a shallow 1D CNN that exploits the mean intensity and number of points per ring in the validated clusters. The output of both CNNs is the color of the cone. Following a different route, and inspired by the different colors of cones that distinguish the left and right boundary of the track, Chen *et al.* [4] chose to use a custom CNN to distinguish the geometric distribution of cones and, consequently, find the color of the validated clusters.

2.3 LiDAR and Camera Approaches

A review was also conducted on implementations that follow an approach of fusing the information between the LiDAR and the camera. Here, the perception pipeline previously implemented by the Formula Student Técnico (FST) will also be reviewed. These implementations [41, 15] start by identifying the cones using methods similar to those described in Sec. 2.2, *i.e.*, methods that only rely on the LiDAR. First, the raw point cloud is filtered using either a box region to remove points that are outside it or a pass-through filter that performs a simple filtering along any specified axis, removing points that lay outside the defined range for each axis. Then, the ground points are removed from the point cloud using Random Sample Consensus (RANSAC) [8] with the assumption that the ground plane is flat. Finally, Euclidean clustering is used to cluster the point cloud into groups and to compute their correspondent centroids, which are then projected onto the image plane.

Tian *et al.*, in [41], create a Region of Interest (ROI) around the projected centroids and, from these ROI boxes, gradient features are extracted using Histogram of Oriented

Gradients (HOG) [7]. Furthermore, using the extracted HOG features, a Support Vector Machine (SVM) [6] classifier is trained to distinguish cones from other clustered objects. The RGB color space of the ROI patches is then converted to the HSV color space and the color of the cones is estimated by extracting the main color present on the ROI.

In [15], the FST team proposes using YOLOv3-tiny [1], a lightweight version of the 2D real-time object detector YOLOv3, to classify and identify the location of the cones on the camera image. Then, for each projected centroid of the cones, it is evaluated if it falls inside any of the bounding boxes that surround the detected objects. Those who do, receive the color associated with the class of the correspondent bounding box. Although this approach combines the best features of each sensor by using the position estimated by the LiDAR and the color estimated using the camera, it is a very calibration dependent approach, in the sense that a single projected point is used to evaluate if it matches any bounding box and, since the further the detected cones are, the smaller the bounding boxes will be, this matching task becomes more demanding and less forgiving for further away cones. Moreover, this approach involves using the full image to detect the cones, something that is very computationally expensive, even running, on GPU, a lightweight version of a real-time object detector, like YOLOv3-tiny.

The perception pipeline proposed in this work fits inside this last approach category, in the sense that it fuses the information between the two sensors to identify the cones and classify their color. However, the proposed perception pipeline differs by not using heavy object detectors to detect the different colored cones in the full camera image. Instead, the cones are detected using the LiDAR information and are then projected onto the camera image plane, where 2D bounding boxes are created around the cones. These 2D bounding boxes are cropped from the original image and, using custom lightweight CNNs, the resultant image patches are classified, something that is significantly less computationally expensive.

Chapter 3

Vehicle Setup

The base vehicle chosen to be transformed to compete in the Formula Student Driverless FSD class was the last prototype developed by the Formula Student Técnico (FST) team, the FST09e. This was the ninth prototype developed by the team and it is currently the most successful and reliable prototype developed and built by the team, achieving, in 2019, 9^{th} place out of 39 teams that attended what is the most prestigious and challenging Formula Student competition in the world, the Formula Student Germany (FSG), held at the famous Hockenheimring.

Figure 3.1: FST09e, the ninth prototype developed by FST, at Formula Student Germany 2019.

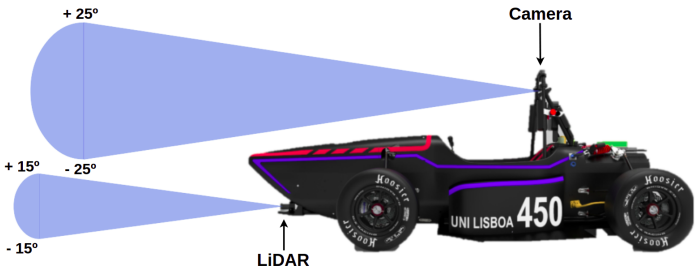
This prototype incorporates a full carbon fiber monocoque, a complete aerodynamic

package, and an 8 kW/600 V high voltage battery that supplies the four (one in each wheel) permanent magnet synchronous electric motors, producing a total peak power of 120 kW and capable of accelerating from 0-100 km/h in around 2.5 seconds.

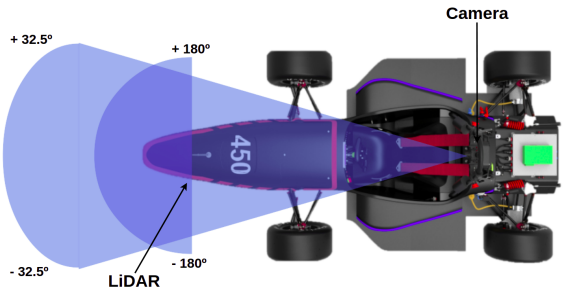
In order to meet the set of challenges imposed by the FSD competition, the prototype was equipped with a set of sensors that replaces the driver's ability to perceive the environment. The first sensor added to the car was a Light Detection and Ranging (LiDAR) sensor, which allows to accurately get the distance to obstacles by targeting them with a laser beam and measuring the time for the reflected light to return to the receiver. More specifically, the LiDAR used is a Velodyne VLP-16, which is a 16 channel LiDAR that has a 100-meter range and a 360 degrees horizontal Field of View (FoV) with a 30-degree vertical FoV angle. The LiDAR was positioned on top of the front wing of the car as it corresponds to the position that maximizes the number of point returns per cone, allowing to detect further away cones. In the context of Formula Student, the vertical resolution of the LiDAR represents the most important specification since it is what limits the number of point returns per cone, which directly relates to the distance at which cones can be perceived. A representation of the LiDAR FoV, as well as its positioning on the car, can be seen in Fig. 3.2.

The second perception sensor added to the prototype was a CMOS camera responsible for collecting rich and colorful information of the environment in the form of an RGB image, complementing the sparse information, in the form of a point cloud, retrieved by the LiDAR. The chosen camera was a Lucid Triton TRI032S, which is a 3.2 megapixels camera that, coupled with a 6 mm f/1.8 Kowa lens, offers a useful 65 degrees of FoV with a 2048×1536 resolution and a frame rate of 30 FPS. The camera is positioned on the main hoop of the car, close to where the eyes of the pilot would be, as this positioning allows to reduce the occlusion among cones and to still be able to perceive cones that are placed one behind the other (in line of sight). The positioning of the camera as well as a representation of its FoV can also be seen in Fig. 3.2.

To simulate the stimulus that a pilot feels while driving and to estimate the car's position and orientation, the prototype was also fitted with an Attitude and Heading Reference System (AHRS), which estimates the accelerations and velocities of the vehicle by fusing the information from an Inertial Measurement Unit (IMU) and from a Global Positioning System (GPS). The information retrieved by all of these sensors is then processed by the autonomous system algorithms using a powerful onboard computer



(a) Side view.



(b) Top View.

Figure 3.2: Representation of the positioning and Field of View (FoV) of the LiDAR and Camera.

equipped with an Intel Core i7-8700T central processing unit (CPU) and an NVIDIA GeForce GTX 1060 Ti graphical processing unit (GPU).

To replace the driver's hands, a DC motor, coupled with a gearbox, was attached to the steering column using a pulley system that allows turning the steering wheel according to the steering angle requested by the controller pipeline. Furthermore, as safety is a primary concern when dealing with any kind of racing vehicle, especially autonomous ones, the prototype was also fitted with a Remote Emergency System (RES), responsible for sending the "Go Signal" and, in case of emergency, for stopping the vehicle by deploying the Emergency Brake System (EBS), which should be able to stop the vehicle with an average deceleration greater than 8 m/s^2.

Chapter 4

Proposed Perception Pipeline

This chapter describes the proposed perception pipeline and how it helps solve the problems identified while using the perception pipeline previously implemented by the FST team. In Sec. 4.1 it is described the process of identifying the cones that delimit the Formula Student tracks using only the Light Detection And Ranging (LiDAR) sensor. Then, in Sec. 4.2, it is detailed how the classification of the cone's color is performed and how the sensors can be fused to help the classification task. Finally, in Sec. 4.3, it is described the implementation of a tracking algorithm that allows to mitigate misclassifications by taking into account previous observations. A complete overview of the proposed perception pipeline is shown in Fig. 4.1.

The proposed pipeline was implemented using ROS [30], which handles the communication and synchronization between modules, who were developed in C++ for its enhanced computational efficiency. Furthermore, PyTorch [28] was used to develop and train the implemented Convolutional Neural Networks (CNNs) given the ease with which it allows different neural network models to be developed and tested.

4.1 Cones Detection

This section describes the process of identifying and generating cone proposals, and their corresponding position, using only the point cloud generated by the Light Detection And Ranging (LiDAR) sensor. The generated cone proposals will later be used in the classification step described in Sec. 4.2, where it will be assessed which proposals actually represent cones and with what color. These cone proposals are generated through a

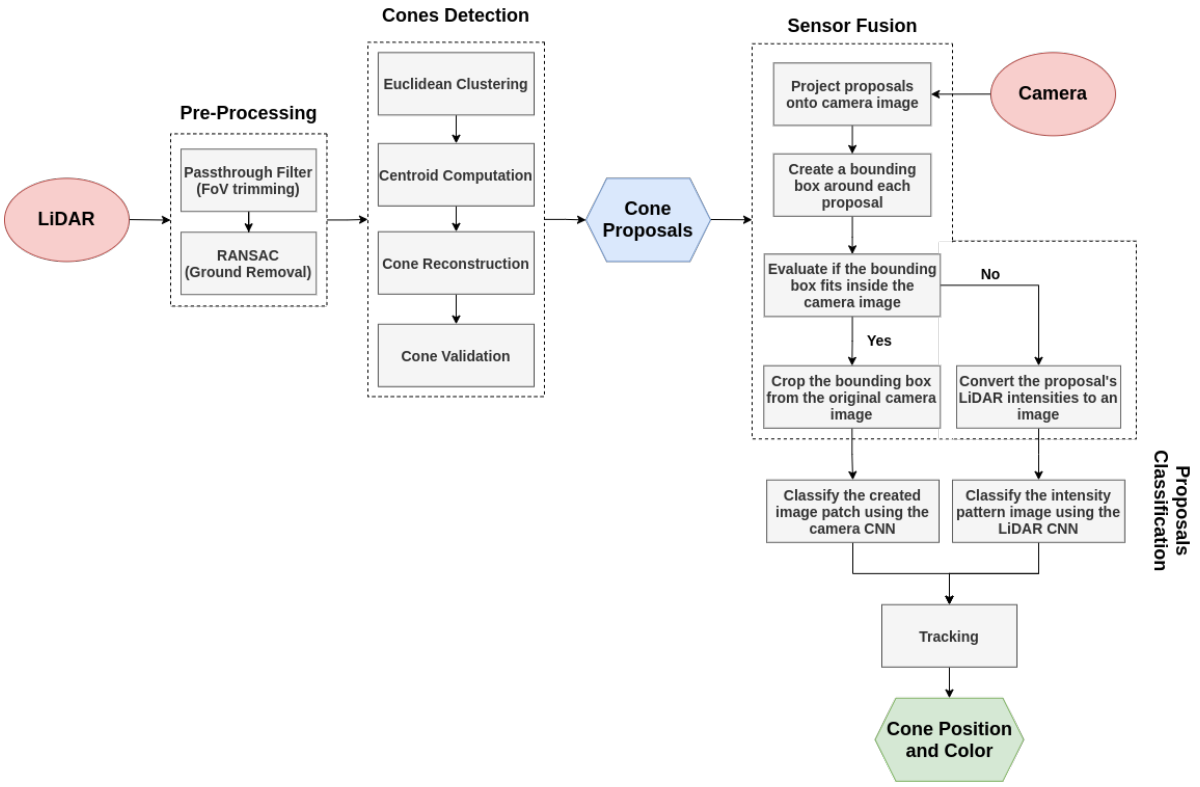
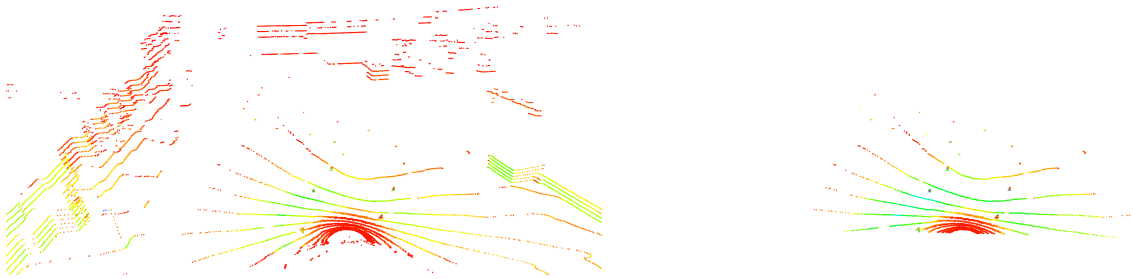


Figure 4.1: Overview of the proposed perception pipeline.

multi-step point cloud processing that includes a pass-through filter to trim the LiDAR Field of View (FoV), a ground removal algorithm to remove points that belong to the ground, a clustering algorithm to extract and group the point cloud into clusters, a cone reconstruction algorithm to retrieve points incorrectly removed during the ground removal process and a final cone validation step to remove any remaining outliers.

4.1.1 Pass-through Filter

As described in Chapter 3, the LiDAR used in FST 10d, the FST autonomous prototype, has a range of around 100 meters and a horizontal FoV of 360 degrees with a 30-degree vertical FoV angle. Due to these LiDAR specifications, most of the points of the generated point cloud lay outside the Region Of Interest (ROI), the region where the cones that delimit the track are placed, which is mostly in front of the car, up to a longitudinal distance of 25 m and up to a lateral distance of 10 m. On top of that, due to the LiDAR being positioned in the front wing of the car, as shown in Fig. 3.2, almost



(a) Raw point cloud (b) Pass-through filtered point cloud

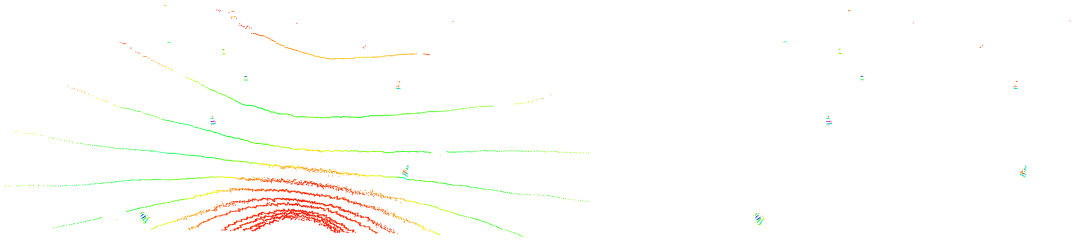
Figure 4.2: Raw point cloud returned by the LiDAR, represented in (a), and the resultant point cloud after the pass-through filtration, represented in (b).

half of the LiDAR FoV is obstructed by the car itself, which causes it to contain a lot of noisy points and making it unusable.

In order to remove these points from the point cloud and avoid using them on future point cloud processing steps, which would increase their total execution time, a pass-through filter, from the Point Cloud Library (PCL) [34], is applied to the raw point cloud generated by the LiDAR. This pass-through filter performs a simple filtering along any specified axis, removing points that lay outside the defined range for each axis. Thus, applying this pass-through filter on all three axes allows not only to remove all the points that are behind the LiDAR, and therefore outside the ROI, but also points that are much higher than the height of a cone used in Formula Student competitions and points that are much farther, widthwise, than the width of a Formula Student track. A comparison between the raw point cloud and the resultant point cloud after the application of the pass-through filter is shown in Fig. 4.2, where it can be seen that most of the points that are outside the ROI, like those generated from the LiDAR beams hitting the walls and those that are behind the LiDAR, are removed after this LiDAR processing step.

4.1.2 Ground Removal

Although most of the points that lay outside the ROI were removed from the point cloud in the previous LiDAR processing step, there are still several points in the ROI that were generated by the reflection of the LiDAR laser beams on the ground, as it can be noted in Figures 4.2(b) and 4.3(a). These points can be considered noisy data as they mostly belong to the ground plane and do not represent cones. Following the same prin-



(a) Point cloud before ground removal

(b) Point cloud after ground removal

Figure 4.3: Point cloud after the pass-through filtration, before the ground removal, represented in (a), and the resultant point cloud after the ground removal step using RANSAC with a plane model, represented in (b).

ciple of removing these points to avoid using them in subsequent LiDAR processing steps, an implementation of the iterative Random Sample Consensus (RANSAC) [12] method, from the sample consensus module of the PCL library, was used. This implementation allows choosing a mathematical model, from fifteen available, to be used as the mathematical model whose parameters will be estimated by the RANSAC method. RANSAC is a resampling technique that generates candidate solutions by using the minimum number of observations (data points) required to estimate the underlying model parameters. An overview of the RANSAC algorithm is given in Alg. 1.

Algorithm 1 RANSAC algorithm (Derpanis *et al.* [8])

1: Select randomly the minimum number of points required to determine the model parameters.
2: Solve for the parameters of the model.
3: Determine how many points from the set of all points fit with a predefined tolerance ϵ.
4: If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold τ , re-estimate the model parameters using all the identified inliers and terminate.
5: Otherwise, repeat steps 1 through 4 (maximum of N times).

In this case, the points to be removed belong mostly to the ground which, in the Formula Student context, can be approximated to a plane. Thus, the iterative RANSAC method was implemented using the PCL sample consensus plane model, which defines a plane model for 3D plane segmentation. This allows identifying the ground plane

and removing it from the point cloud. The comparison between the before and after the ground removal using RANSAC is shown in Fig. 4.3, where it can be seen that all the LiDAR points belonging to the ground plane are removed, resulting in a point cloud where most of the points belong to cones, as shown in Fig. 4.3(b).

4.1.3 Euclidean Clustering

After the previous two LiDAR processing steps, one gets a point cloud similar to the one shown in Fig. 4.3(b), which corresponds to an unorganized point cloud where all of the 3D points are contained in the ROI, the majority of them belonging to cones. However, despite knowing that most of these 3D points belong to cones, one does not know to which cone each of these points belongs, *i.e.*, at this point, there can be either as many cones as the number of 3D points in the point cloud or a single cone to which every 3D point belongs.

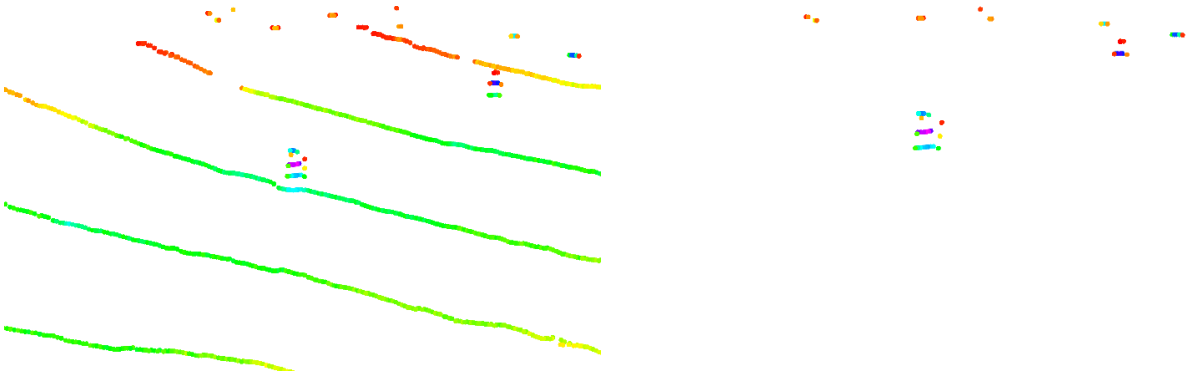
To address this problem, the Euclidean Distance Clustering method [35], from PCL, is used, which performs cluster extraction in a Euclidean sense, *i.e.*, it computes the Euclidean distance between each one of the 3D points of the point cloud and it groups them into clusters based on predefined parameters such as the minimum and the maximum number of points that a cluster can contain in order to be considered valid, the maximum distance between two points so that they can be considered part of the same cluster, and the spatial tolerance for new cluster candidates. These parameters can be tuned taking into consideration the dimensions of a cone used in Formula Student competitions and the expected distance between cones, which is also regulated by the competitions' rules [13].

The result of the Euclidean clustering is shown in Fig. 4.4, represented by the 3D centroid of each cluster, which is computed after the Euclidean clustering step is done. Notice that the farthest points may not be considered to be clusters as the number of points is less than the minimum number of points that a cluster needs to contain in order to be considered valid.

Figure 4.4: Result of the Euclidean clustering method applied to the filtered point cloud shown in Fig. 4.3(b). Each purple sphere represents the 3D centroid of its corresponding cluster, identifying the different clusters.

4.1.4 Cone Reconstruction

During the ground removal process, described in Sec. 4.1.2, it was noticed that the bottom-most layer of points hitting a cone was often being removed because it was being associated with the ground plane, as shown in Fig. 4.5. Given the small number



(a) Point cloud before ground removal

(b) Point cloud after ground removal

Figure 4.5: Point cloud before the ground removal step, represented in (a), and the resultant point cloud after the ground removal step, represented in (b), which shows the removal of the last layer of points that was hitting the cone.

of LiDAR layers hitting cones resulting from the not so fine LiDAR vertical resolution, which is an hardware drawback, these layers can act as invaluable information, especially in the cone classification step presented in Sec. 4.2.2. Thus, an extra step in the LiDAR processing is done in order to retrieve these so valuable layers of 3D points.

To retrieve these points, a cylindrical area, with similar dimensions to those of a Formula Student cone, is created around each cluster using its corresponding centroid, as shown in Fig. 4.6(a). The cylindrical areas, which are an approximation to the geometrical shape of a cone, are then used to assess if any of the 3D points from the removed ground point cloud is contained inside it. This is done by evaluating if

$$(\vec{q} - \vec{p_1}) \cdot (\vec{p_2} - \vec{p_1}) \geq 0 \text{ and} \tag{4.1}$$

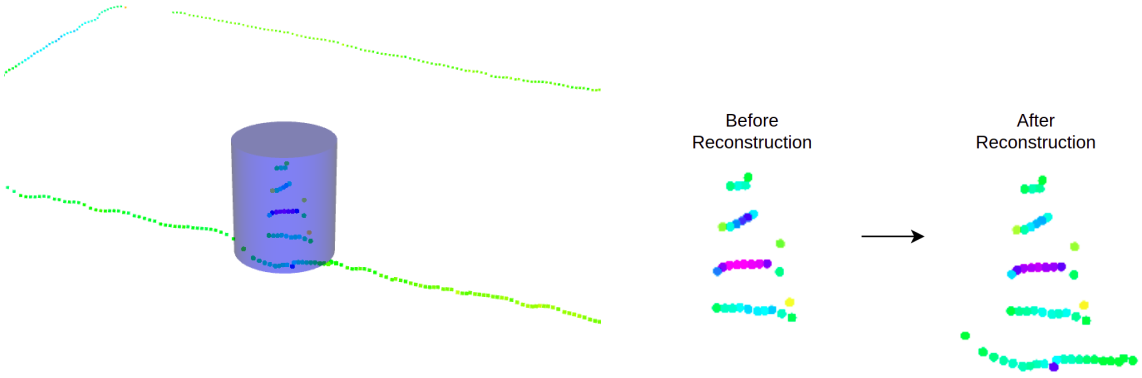
$$(\vec{q} - \vec{p_2}) \cdot (\vec{p_2} - \vec{p_1}) \leq 0, \tag{4.2}$$

which verifies if a 3D point, $\vec{q}$, lies between the planes $\vec{p_1}$ and $\vec{p_2}$ of the two circular facets of the cylinder, and by making sure that

$$\frac{\| (\vec{q} - \vec{p_1}) \times (\vec{p_2} - \vec{p_1}) \|}{\|\vec{p_2} - \vec{p_1}\|} \leq r, \tag{4.3}$$

which confirms that the 3D point $\vec{q}$ lies inside the curved surface of the cylinder with radius r.

The points from the removed ground point cloud that are contained inside the created cylindrical area are then added back to the filtered point cloud and associated with its corresponding cluster. A representation of the cylindrical area used to retrieve the bottom layer of the cone's cluster and a comparison between the before and after the cone reconstruction step can be seen in Fig. 4.6.

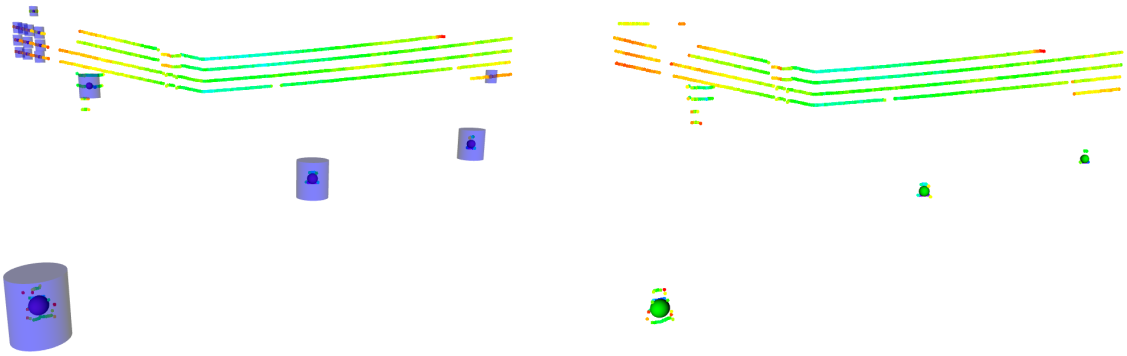


(a) Cylinder created around each cluster's centroid. (b) Cluster's point cloud before and after the cone reconstruction step.

Figure 4.6: In (a) it is represented the cylindrical area, created around each cluster's centroid, used to retrieve points that belong to the cone's cluster but that were removed in the ground removal step. The result of the cone reconstruction process is represented in (b), where it is shown the retrieval of the bottom-most layer of points.

4.1.5 Cone Validation

Even though in Formula Student competitions the tracks are in a substantially controlled environment when compared to urban traffic, there can be obstacles or objects nearby that are not cones and therefore not part of the track. Despite not being part of the track, these obstacles can be present in the LiDAR FoV and in the desired ROI, which may cause them to be considered as clusters while not being cones. To avoid using these outlier clusters in the classification task, one last LiDAR processing step is done. In this step, and similarly to what was done in Sec. 4.1.4, a cylindrical area with similar dimensions to those of a Formula Student cone, is fitted around each of the cone proposal's centroid and, using (4.1), (4.2) and (4.3), it is assessed if all the points of the proposal's cluster are contained inside the respective cylinder. At this point, only the clusters whose all points fall inside the correspondent cylindrical area are considered and used to generate cone proposals to later be classified and assigned a color. The result of this last LiDAR processing step is shown in Fig. 4.7, where it can be seen that several clusters that were being considered to be cones, were removed after the cone validation step, leaving for the classification step only clusters that represent cones.



(a) Identified clusters before cone validation step. (b) Validated clusters after cone validation step.

Figure 4.7: In (a) it's represented the identified clusters before the cone validation step and the cylinder used to validate them, and, in (b), it is represented the validated clusters after this validation process.

4.2 Cones Classification

The clusters validated in the last step of the LiDAR, which may not all be cones, are considered to be cone proposals. These cone proposals need to be classified in order to validate if they are actually cones and with which color. Since these cone proposals, generated through the LiDAR processing described in Sec. 4.1, may be outside the Field of View (FoV) of the camera, which commonly happens in tight corners or when the cone proposals are very close to the car, two different Convolutional Neural Networks (CNN) are used to classify the proposals: one that resorts to the camera image to classify the image patches of the proposals that fall inside the camera's FoV, as it will be described in Sec. 4.2.3, and one that resorts to the point cloud intensity to generate grayscale images of the cone proposals that are outside the camera's FoV, as it will be further explained in Sec. 4.2.2. In Sec. 4.2.1 it is described how the cone proposals generated by the LiDAR processing described in Sec. 4.1, can be fused with the camera image to ease the cone's classification process by choosing which CNN should be used to classify each cone proposal.

4.2.1 Sensor Fusion

The choice of the CNN to be used in the classification task is based on whether the camera is able to see the cone proposal or not, which means that it is necessary to determine which cone proposals are inside the camera's FoV and which are outside. To determine this, one starts by fitting a 3D bounding box, with similar dimensions to those of a cone used in Formula Student competitions, around each 3D cone proposal's centroid, as shown in Fig. 4.8. Considering that on an image it is possible to define a bounding box using only its top-left and bottom-right points, only the two correspondent 3D points, represented by the red spheres in Fig. 4.8, need to be projected onto the camera's image plane.

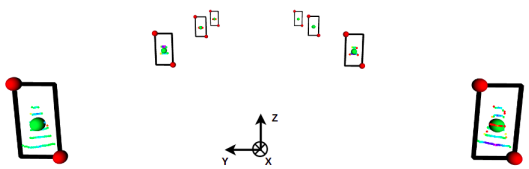


Figure 4.8: Representation of the 3D bounding boxes created around each 3D cone proposal's cluster and the corresponding top left and bottom right 3D points, represented by the red points, that will be projected onto the image plane in order to create a 2D bounding box on the image.

To compute these two points for each 3D centroid $\mathbf{P} = (X, Y, Z) \in \mathbb{R}^3$, two new points, $\mathbf{P}_1$ and $\mathbf{P}_2$, are created. These points are created by keeping the centroid's X coordinate, which represents the centroid's depth of the cone proposal, and by taking into account the cone's dimensions, as follows:

$$\mathbf{P}_1 = (X, Y + w/2, Z + h/2), \tag{4.4}$$

$$\mathbf{P}_2 = (X, Y - w/2, Z - h/2), \tag{4.5}$$

where w and h represent the width and height of a cone, respectively. Here, the first point, $\mathbf{P}_1$, given by (4.4), represents the top-left point of the bounding box while the second point, $\mathbf{P}_2$, given by (4.5), represents the bottom-right point of the bounding box.

The aforementioned bounding box points, in the three-dimensional space $\mathbb{R}^3$, can be mapped to points in pixel coordinates on the image plane, in the two-dimensional space $\mathbb{R}^2$, using the 3×4 perspective projection matrix P, as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim K \begin{bmatrix} R|T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{4.6}$$

with $P = K \begin{bmatrix} R|T \end{bmatrix}$, where K represents the camera's intrinsic matrix and $\begin{bmatrix} R|T \end{bmatrix}$ represents the extrinsic matrix. Notice that both the 3D points $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$ and the 2D points $\begin{bmatrix} u & v & 1 \end{bmatrix}^T$ are represented in homogeneous coordinates, hence the additional last component, which represents a scale factor and that, by convention, is set to 1. The camera's intrinsic matrix K transforms 3D camera coordinates into 2D homogeneous image coordinates using a perspective projection modeled by the ideal pinhole camera model [38], represented in Fig. 4.9, and is parameterized by Hartley *et al.* [19] as

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.7}$$

where f_x and f_y represent the camera's focal length, *i.e.*, the distance between the camera's pinhole and the image plane, and c_x and c_y represent the principal point offset, which is the location of the principal point relative to the image plane origin.

The process of finding these four intrinsic parameters, which are specific to the camera model that is being used, is called camera calibration, and it can be done using available camera calibration methods from OpenCV [42], which, by defining real-world coordinates of 3D points using a chessboard pattern of known dimensions, as shown in Fig. 4.10, and find the pixel coordinates (u, v) for each 3D point using multiple images taken from different perspectives.

The extrinsic matrix $\begin{bmatrix} R|T \end{bmatrix}$, given by

$$\begin{bmatrix} R|T \end{bmatrix} = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{array}\right], \tag{4.8}$$

which is a 3×4 matrix that takes the form of a rigid transformation by concatenating a 3×3 rotation matrix R, whose columns represent the directions of the LiDAR axes in

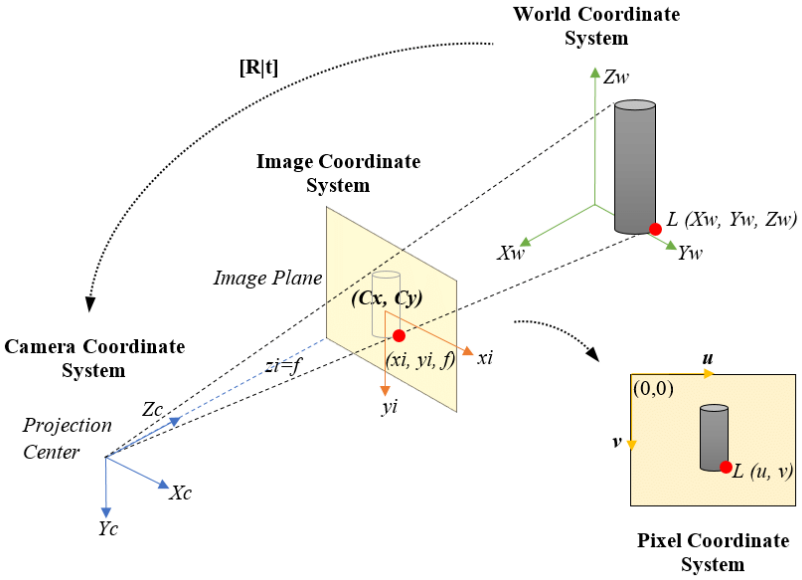


Figure 4.9: Representation of the pinhole camera model: ideal projection of a 3D object on a 2D image. Figure taken from [26].

camera coordinates, and the 3×1 translation column-vector T, which can be interpreted as the position of the LiDAR origin in camera coordinates. The extrinsic matrix describes how the LiDAR is transformed relative to the camera, *i.e.*, it allows to map points from the 3D LiDAR coordinate system to the 3D camera coordinate system.

The process of finding the extrinsic matrix that correctly maps 3D points from the LiDAR coordinate system to the 3D camera coordinate system can be referred to as LiDAR-Camera calibration. Although this calibration can be done using methods that rely on 2D-3D point correspondences, such as those proposed in [17] and [36], the calibration method proposed in [10], which uses 3D-3D point correspondences, has proven to deliver better results and be more consistent. This method involves using augmented-reality tags, namely ArUco markers [16], and the LiDAR point cloud to find the extrinsic calibration parameters by finding two sets of 3D points, one in the camera frame and another in the LiDAR frame, and solving for the $[R|T]$ matrix using the Iterative Closest Point (ICP) [45] algorithm to minimize the error between the 3D-3D correspondences.

Using Eq. (4.6) to project onto the image plane the two points from each of the

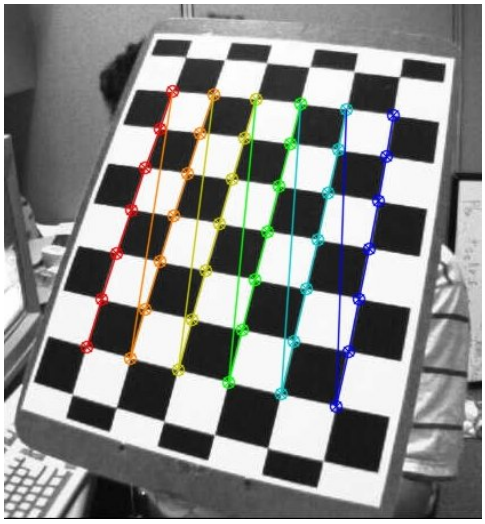
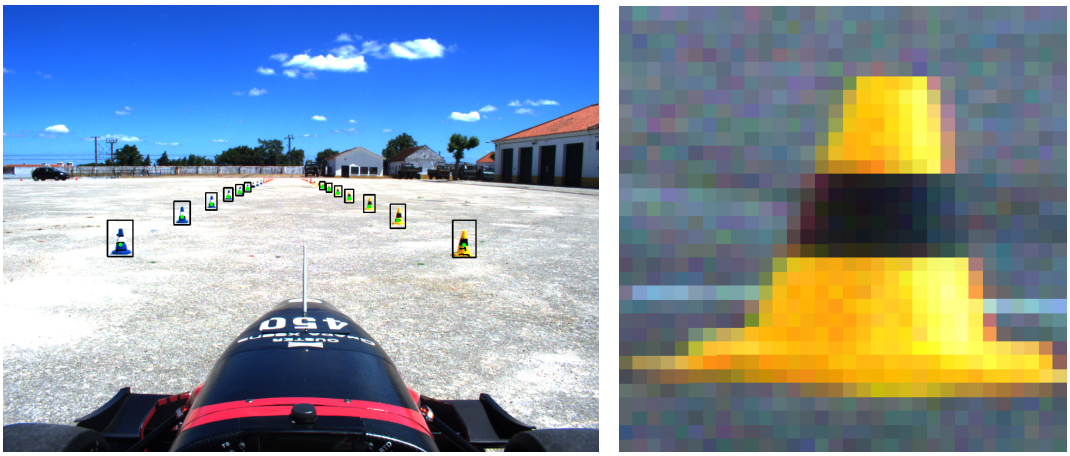


Figure 4.10: Representation of the camera's intrinsic calibration process, where a chessboard is used to detect patterns on the image and, consequently, find the pixel coordinates (u,v) for each 3D point in the different images taken.

3D planes fitted around each cone proposal's 3D centroid, it is possible to create 2D bounding boxes around the cone proposals in the image plane, as shown in Fig. 4.11(a). By simply checking if the entire 2D bounding boxes fit inside the camera image, it can be concluded which cone proposals fall inside the camera's FoV and which don't. For those who do, the corresponding bounding box is cropped from the original camera image, resulting in an image patch similar to the shown in Fig. 4.11(b), which corresponds to the input image of the CNN further described in the camera classification section, Sec. 4.2.3. On the other hand, the cone proposals that are outside the camera's FoV will go through a preparation process that converts their 3D point cloud clusters into grayscale images so that these cone proposals can be classified using only information given by the LiDAR, as further described in Sec. 4.2.2.

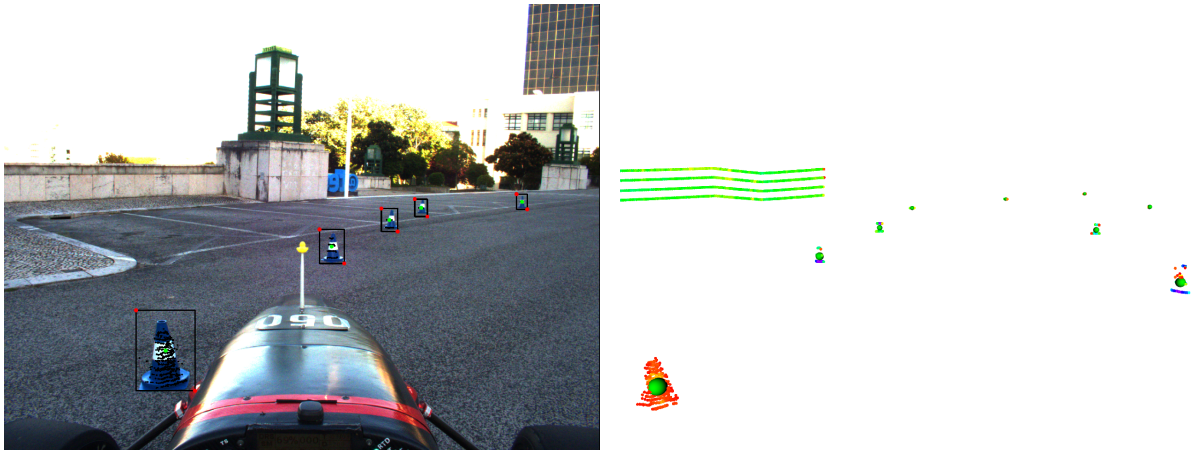


(a) 2D bounding boxes created around cone proposals. (b) Cropped bounding box.

Figure 4.11: In (a) it is represented the 2D bounding boxes created using the 3D-2D projection of the top-left and bottom-right points of the 2D bounding boxes as well as the projection of the cone proposal's centroid and cluster, represented by the green circle and the black points, respectively. In (b) it is represented an example of a cropped bounding box that will later be classified using the camera CNN.

4.2.2 Cone's Color Classification with LiDAR

As mentioned in Sec. 4.2, the cone proposals generated by the LiDAR processing described in Sec. 4.1, may not all fall inside the camera's FoV. Therefore, it is crucial to be able to classify these cone proposals using only the LiDAR information. An example of a situation that shows the importance of detecting and classifying both cones that are inside the image and cones that are outside it, is represented in Fig. 4.12. In such tight corners, only the cones placed on the outer side of the turn fall inside the camera's FoV due to the camera being rigidly mounted to the car and pointing forward, as represented in Fig. 3.2. This means that the cones placed on the inner side of the curve will not be able to be classified using the camera image. In such situations, and due to the nature of the path planner algorithm developed by the FST Lisboa team, the resulting trajectory may not be representative of a good trajectory for the car to follow. The developed path planner uses Support Vector Machine [6] (SVM) – a supervised machine learning model that uses classification algorithms for two-group classification problems – to compute the centerline between the two classes of cones (blue and yellow) that delimit the track sides. It is therefore essential that these two classes of cones are available to the path



(a) 2D bounding boxes created around cone proposals. (b) Point cloud with the validated cone proposals.

Figure 4.12: Cone proposals seen by the camera, represented in (a) by the bounding boxes on the image, and the respective point cloud with all the validated cone proposals, represented in (b).

planner algorithm so that a valid hyperplane (which is used as the centerline) between both classes can be computed by the SVM algorithm. In Fig. 4.12(b) it is possible to see multiple validated cone proposals, represented by the green spheres in the point cloud, that are not in the camera's FoV and therefore can not be classified using the camera's image. In this particular case, most of those cone proposals correspond to yellow cones that delimit the right side of the track.

According to the rules imposed by the Formula Student competitions [13] and as shown in Fig. 1.2, the different cones used in Formula Student Driverless must have always the same dimensions, shape and color pattern. This means that a yellow cone will always be distinguishable by its yellow-black-yellow pattern and a blue cone by its blue-white-blue pattern. Since the intensity of the LiDAR points correspond to the returned strength of the laser pulse that generated the point, which is directly influenced by the object's reflectivity, each cone can also be distinguished in the point cloud according to its intensity pattern, as shown in Fig. 4.13. Therefore, since black reflects less than yellow and white reflects more than blue, a yellow cone can be described, from top to bottom and in terms of intensity, as having a high-low-high pattern, while a blue cone can be described as having a low-high-low pattern. Notice that the orange cones are not considered here due to their intensity pattern being similar to the intensity pattern

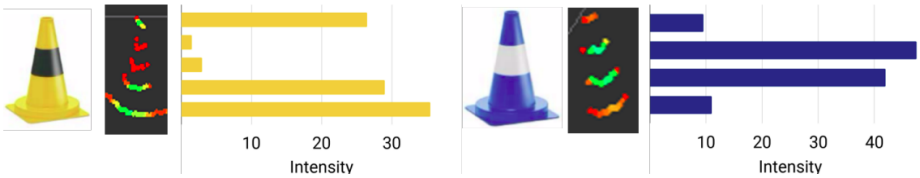


Figure 4.13: Point cloud and intensity gradients of the yellow and blue cones used in the formula student competitions. An high-low-high and low-high-low pattern, for the yellow and blue cones, respectively, can be identified by reading the intensity values along the vertical axis. Image taken from [24].

of the blue cones. The orange cones can be distinguished by their orange-white-orange pattern which, in the point cloud, is also translated to a low-high-low intensity pattern. Furthermore, these cones are traditionally placed on straights and therefore can be seen by the camera.

This intensity gradient pattern is then exploited to estimate the cone proposals' color using a CNN with a similar architecture to the one proposed by Kabzan *et al.* [24]. The input of this CNN corresponds to a 32×32 grayscale image with the intensity points of the cone proposal and the output corresponds to the probability of the proposal being blue, yellow and unknown. As shown in Fig. 4.14, this CNN can be divided into two stages: the feature extraction stage and the classification stage. The backbone of the CNN, which performs the feature extraction, consists of four convolutional layers with a max-pooling layer between them, which not only reduces the number of parameters in the model by downsampling the feature map but it also makes feature detection more robust to object orientation and scale changes. In order to introduce non-linearity into the model, the Rectified Linear Unit (ReLU) activation function is used after each convolutional layer and after each of the five fully connected layers that compose the classifier. The output of the network is normalized to a probability distribution over the three classes using Softmax, which is defined as:

$$\text{Softmax}\left(\vec{x}_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \tag{4.9}$$

where $\vec{x}_i$ represents the input vector (the output of the last fully connected layer), the upper term of the fraction represents the exponentiation part of the function, that converts each element of the input vector into a positive value, and the bottom term represents

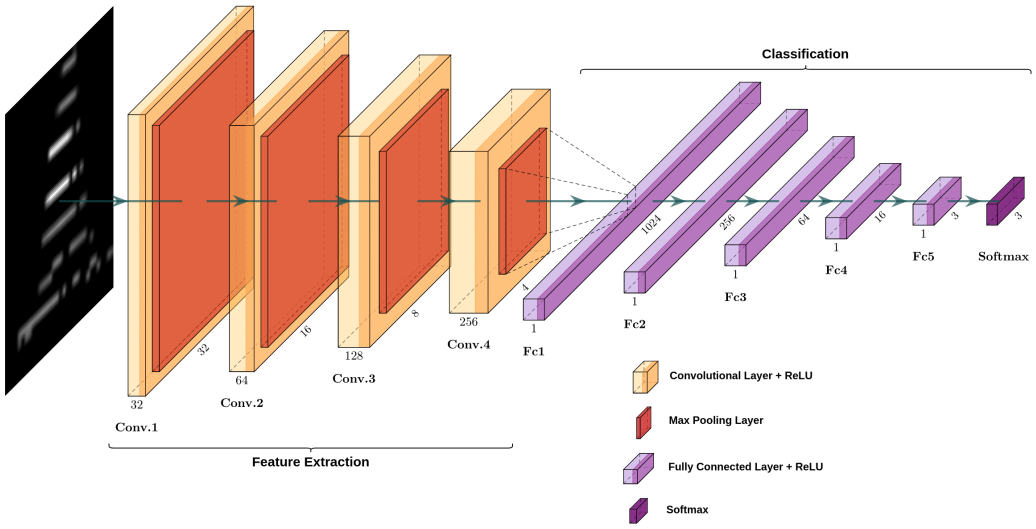


Figure 4.14: Illustration of the CNN architecture used for estimating the cone's color using only the LiDAR information. The CNN is composed of four convolutional layers, on the feature extraction stage, and five fully connected layers, on the classification stage. Its input is a 32×32 grayscale image with intensity points of the cones and it outputs the probability of the cone proposal belonging to each of the three classes (blue, yellow and unknown).

the normalization term, which ensures that all the output values of the function sum to 1 and that each one is in the range $[0, 1]$ for the K number of classes (three in this case).

Furthermore, the aforementioned CNN uses the cross-entropy loss function given by

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}, \tag{4.10}$$

which increases as the predicted probability diverges from the actual label, k, penalizing and adjusting the model weights based on how far the predicted probability is from the actual expected value. It is also less computationally expensive when computing the network's gradients, which, consequently, converges faster to the optimal value. Moreover, while training the model, Dropout [22] - a regularization technique that randomly "deactivates" neurons in the neural network - and batch-normalization [23] - a technique that normalizes the distributions of the hidden layer's inputs - are used to prevent complex co-adaptations between neurons, control overfitting and improve the generalization of the network.

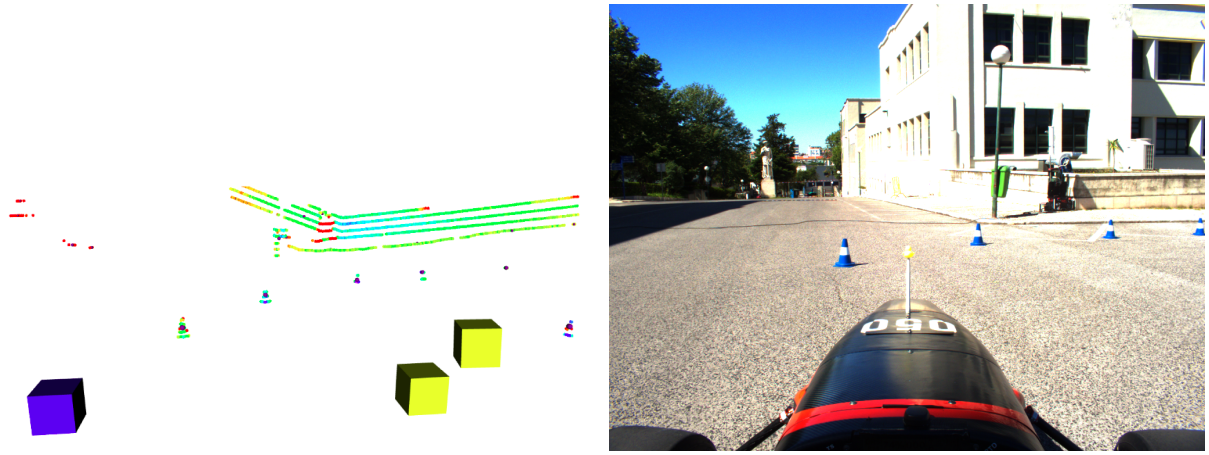
(a) Blue cone. (b) Yellow cone. (c) Unknown.

Figure 4.15: Representation of the input image of the CNN with the point intensities of the cone proposals mapped into a 32×32 image. In (a) it is represented a blue cone with its low-high-low intensity pattern, in (b) a yellow cone with its high-low-high intensity pattern and in (c) an unknown, which corresponds to a noisy cone proposal.

The input image is created by mapping every point of the cone proposal's cluster to a 32×32 image whose pixel values store the intensity of the LiDAR beam in that point. To correctly map these points into a 32×32 matrix, which represents the 32×32 image that the network takes as input, a linear interpolation is performed, leaving a boundary of 3 points around the cone. In order to enhance the difference between the intensity values of the multiple layers, *i.e.*, enhance the intensity pattern of the cones, the pixel values are scaled using a linear scale factor, benefiting high-intensity values and penalizing low ones, which results in images similar to the ones shown in Fig. 4.15.

This network has been trained with a custom dataset of over 19 000 images with the mapped intensity values of blue and yellow driverless Formula Student cones and noisy identified clusters to represent the unknown class. The dataset is composed of images generated using the raw LiDAR data gathered from previous runs using the same LiDAR and it is continuously growing as more raw data is acquired.

An example of the output of the LiDAR CNN that shows the importance of using such CNN is shown in Fig. 4.16. In Fig. 4.16 (a) it is possible to observe three cones, two yellow and one blue, that are not visible in the correspondent camera image, represented in Fig. 4.16 (b), but whose color was estimated using the LiDAR CNN. This example shows the importance of using this CNN, as it allows to classify the color of the cones that delimit the inner side of the curve when the cones are not visible in the camera image.



(a) Point cloud with the LiDAR CNN output.

(b) Correspondent camera image.

Figure 4.16: Example of the output of the LiDAR CNN. In (a) it is represented the point cloud with the generated cone proposals evaluated by the CNN and their correspondent predicted color. In (b) it is shown the correspondent camera image, where the evaluated cone proposals are not visible, highlighting the importance of the LiDAR CNN.

4.2.3 Cone's Color Classification with Camera

The classification of the remaining cone proposals, whose projected 2D bounding boxes fit inside the camera image, is performed using a CNN with an architecture similar to the one described in Sec. 4.2.2. However, this CNN takes as input a 32×32 RGB image that corresponds to the image patch of the projected bounding box that surrounds the cone proposal, as shown in Fig. 4.11(b). Furthermore, and as illustrated in Fig. 4.17, the architecture of the network described in this section has one less convolutional layer in the feature extraction stage, a change that allowed to achieve better computational times while maintaining an acceptable accuracy and reliability, as demonstrated in Chapter 5.

This CNN consists of three convolutional layers that perform feature extraction and, similarly to the CNN described in Sec. 4.2.2, after each convolutional layer a max-pooling layer is used to downsample the feature maps after using Rectified Linear Unit (ReLU) as the activation function to introduce non-linearity to the model. As for the classifier, it consists of five fully connected layers, where to the output of each one is also applied the ReLU activation function. The output of the network is normalized to a probability distribution over the five classes using Softmax (defined in (4.9)). Additionally, while training the model, dropout and batch normalization are also used to randomly "deac-

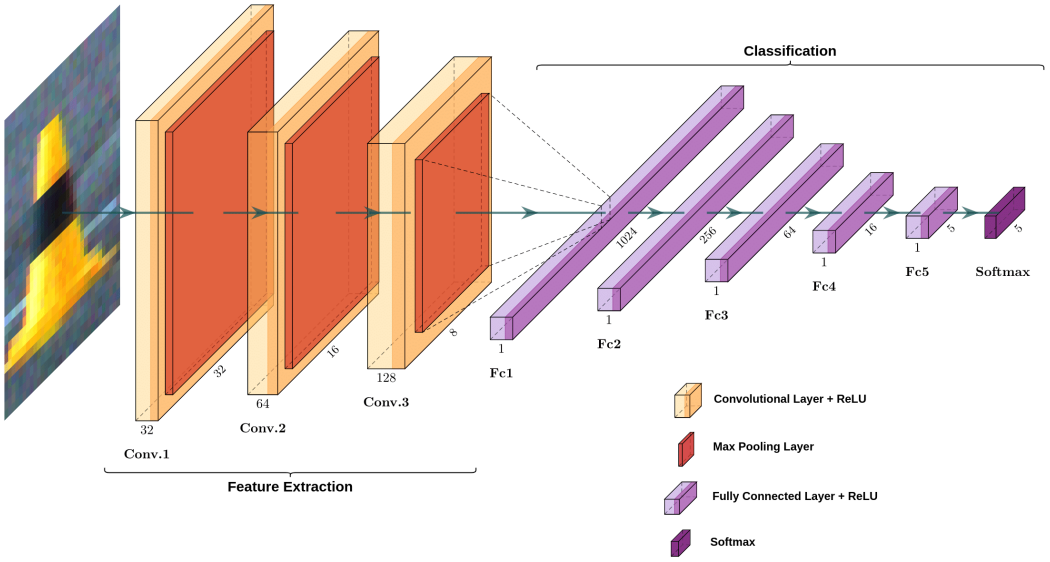
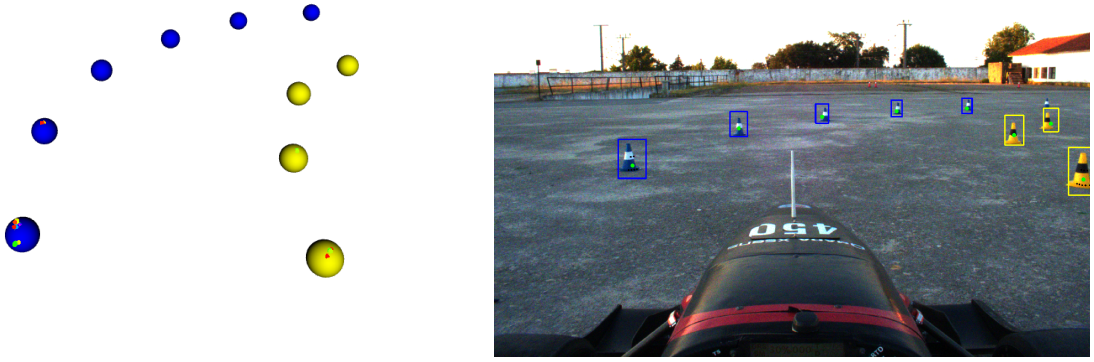


Figure 4.17: Illustration of the CNN architecture used for estimating the color of the cones using the camera image. The CNN is composed of three convolutional layers, on the feature extraction stage, and five fully connected layers, on the classification stage. It receives, as input, a 32×32 RGB image with the cropped bounding box containing the cone proposal and it outputs the probability of that cone proposal belonging to each of the five classes (blue, yellow, orange, big orange and unknown).

tivate" some neurons in the neural network and to normalize the distributions of the hidden layer's inputs, preventing complex co-adaptations between neurons, controlling overfitting and improving the generalization of the network.

This network has been trained with a custom dataset of over 100 000 images of the four classes of Formula Student Driverless cones and background images to represent the unknown class. The dataset was originally only composed of images from FSOCO [11] – a collaboration between Formula Student teams that aims to accelerate the development of camera-based solutions in the context of Formula Student Driverless – but has since been enlarged with more images gathered from runs on different testing sites and with different weather/light conditions.

An example of the output of this CNN is shown in Fig. 4.18. In Fig. 4.18 (a) it is possible to observe the point cloud with the clusters of the generated cone proposals and the correspondent color classified by the CNN and, in Fig. 4.18 (b), it is shown



(a) Point cloud with the Camera CNN output.

(b) Correspondent camera image with the camera CNN output.

Figure 4.18: Example of the output of the camera CNN represented in: (a) the point cloud with the generated cone proposals and their correspondent predicted color and in (b) the camera image with the generated bounding boxes around the cone proposals and their correspondent predicted color.

the correspondent camera image with the generated bounding boxes, also with the correspondent predicted color, used to create the image patches that are classified by the camera CNN.

4.3 Tracking

Given the simplicity of the CNNs described in Sec. 4.2.2 and in Sec. 4.2.3, which allow to classify and associate a color to the cone proposals generated through the LiDAR processing described in Sec. 4.1, a Nearest Neighbor [5] based tracking algorithm was developed in order to mitigate possible misclassifications returned by either CNNs. By taking into account previous observations, it is possible to keep track of the colors associated with each cone proposal and thus decide the cone's color based on the number of times each color was associated with the proposal. The general overview of this tracking algorithm is shown in Fig. 4.19.

This tracking algorithm starts by fusing the information returned by both neural networks, the cone detections with an associated position and color. Once all the cone

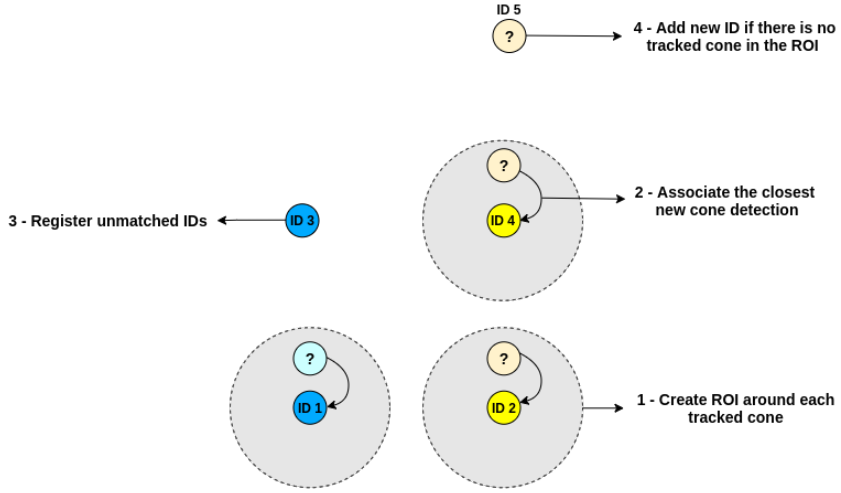


Figure 4.19: General overview of the implemented Nearest Neighbor based tracking algorithm. New cone detections are associated with previously tracked cones if they are contained in the search radius. Unmatched IDs are registered and new IDs are created for the non-associated cone detections.

detections are fused, a different ID is assigned to each of them and, simultaneously, the cone's color is registered. Having assigned the first IDs, the Nearest Neighbor based tracking algorithm is used to associate the next set of cone detections to the already existing and tracked cone detections. So, for each previously tracked cone detection, it is searched for the closest cone detection, within the new set of cone detections, and the tracked cone's position is updated and the color is registered, both using the information of the closest cone detection. It should be noticed that, to avoid miss associating new cone detections to a tracked cone that, although is the closest one, it is too far away, a searching radius is used taking into account the minimum distance between cones defined by the Formula Student competitions rules. The cone detections that were not associated with any previously tracked cone are considered to be new cones and an ID is assigned to them, allowing them to be used as tracked cones in the next iteration. On the other hand, the previously tracked cones that were not associated with any new cone detection are registered as being missed and are not considered as cone detections in the current iteration, although remaining to the next iteration as tracked cones. By keeping note of the number of times a tracked cone was missed it is possible to remove not only outliers that in the meantime stopped being identified but also cones that have already left the LiDAR FoV, *e.g.*, cones that have already been passed by the car and

are now behind it. An overview of the implemented tracking algorithm is detailed in Alg. 2.

Algorithm 2 Tracking algorithm

```
Join the classified cone detections of both Convolutional Neural Networks.
if tracked cones array is empty then
    for each new cone detection do
        Assign it an ID.
        Add it to the tracked cones array.
        Register the associated color.
    end for
else
    for each cone in tracked cones array do
        Create a Region Of Interest (ROI) around the previously tracked cone.
        Search for the closest new cone detection.
        Evaluate if the closest new cone detection is inside the ROI.
        if closest new cone detection inside ROI then
            Update the tracked cone position using the position of the closest new cone
            detection.
            Register the color associated with the closest new cone detection.
            Decide the color of tracked cone by searching for the most seen color.
            Remove the new cone detection from the new cone detections array.
        else
            Register that the tracked cone was missed.
            if cone missed too many times then
                Remove it from the tracked cones array.
            end if
        end if
    end for
    for new cone detection in the remaining cone detections array do
        Assign it a new ID.
        Add it to the tracked cones array.
        Register the associated color.
    end for
end if
```

Furthermore, the cone's color is decided by searching for the color that was most associated with that cone ID. Taking the example shown in Fig. 4.20, the cone tracked with the ID 56, was classified eight times as being blue, but the last color classification came up as yellow cone. However, the cone still went through as being a blue cone

because it has already been previously classified as being blue many more times. The same logic applies to when the cone was classified as unknown. This example shows the importance of this tracking algorithm. If it was not for this tracking algorithm, the cone detections would be instantaneous, *i.e.*, the most recent cone color, which was yellow, was going to effectively be the chosen color, propagating this misclassification to the subsequent path planning algorithms.

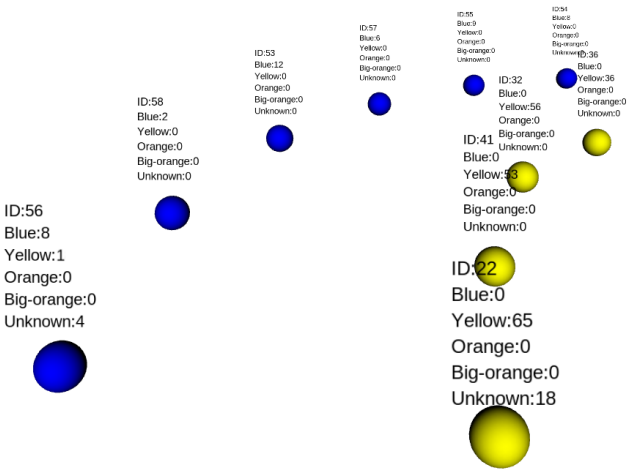


Figure 4.20: Example that shows the importance of the tracking module. Even though the blue cone tracked with ID 56 was classified as being yellow, it still went through as being blue because it had already been previously classified eight times as being blue.

The implemented tracking has proven to be a powerful filter, as it helps to smooth out cone color estimation and make the pipeline more robust to misclassifications, something that significantly helps the path planning algorithms in a previously unknown track.

Chapter 5

Experimental Results

In this chapter, the overall results of the proposed perception pipeline will be evaluated in terms of performance and computation cost. First, in Sec. 5.1, the metrics used to evaluate the performance of the classifiers used to classify the cone proposals' color will be detailed. The overall results of the Convolutional Neural Network (CNN) that solely relies on the information from LiDAR are presented in Sec. 5.2. Similarly, in Sec. 5.3, it is presented the overall results of the CNN that relies on the created image patches with the cone proposals. Then, in Sec. 5.4, an ablation study on the LiDAR processing methods is conducted in order to justify the design choices made on the LiDAR pipeline. Finally, in Sec. 5.5, it is performed a comparison of the overall perception pipeline results with and without the tracking module.

5.1 Metrics

In order to evaluate the performance of the trained models, several performance metrics should be used, as a model can provide satisfying results when using a certain metric but give poor results when using others. Most of the metrics used to evaluate the performance of a model can be derived from the confusion matrix of a model, which is a square matrix whose dimensions depend on the number of classes and that describes the complete performance of a model. As shown in Fig. 5.1, which represents a confusion matrix for a binary classification problem, each quadrant of the matrix represents a category. True Positives (TP) represent the number of samples from the positive class that were correctly predicted as positive. Similarly, True Negatives (TN) represent the

number of samples from the negative class that were correctly predicted as negative. False Positives (FP) represent the number of samples from the negative class that were incorrectly predicted as being positive. False Negatives (FN) represent the number of samples from the positive class that were incorrectly predicted as being negative. It should be noticed that, for a multi-class problem, the same logic can be applied for each class individually. From these four categories multiple metrics can be defined.

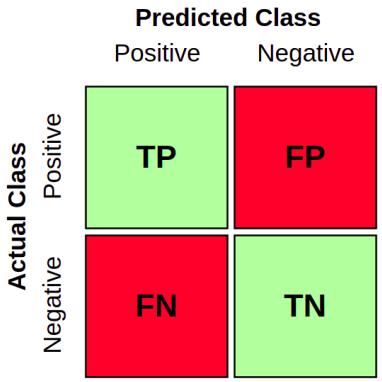


Figure 5.1: Example of a confusion matrix for a binary classification problem.

The first metric, and the most commonly used to evaluate the model's performance, is the accuracy metric. The accuracy metric is given by

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5.1}$$

and it gives the overall accuracy of the model by performing the ratio between the number of correct predictions and the total number of samples, *i.e.*, it tells how much of the samples were correctly classified. Although the accuracy is the most intuitive performance measure, it only works well if each class is equally represented.

The precision metric, given by

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{5.2}$$

represents the ratio of correctly predicted positive samples to the total predicted positive samples. High precision can be obtained when there is a low false positive rate. Therefore, precision is a good performance metric to use when there is high costs for

false positives. On the other hand, if there is high costs associated with false negatives, the recall metric should be used. Recall is given by

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{5.3}$$

and it gives a measure of how accurately the model is able to identify the relevant data by telling the fraction of all positive samples that were correctly predicted as positive. For problems where both precision and recall are equally important, the F1 Score metric should be used. F1 Score metric, given by

$$\text{F1\_Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{5.4}$$

tells how precise and robust the model is by combining, through an harmonic mean, the precision and recall metrics into a single metric.

5.2 LiDAR Classification

The CNN used to classify the generated cone proposals that do not fall inside the camera's FoV, whose architecture is represented in Fig. 4.14, was trained using a custom dataset of over 19 0000 images equally distributed for the three classes, as shown in Table 5.1. This dataset was split into training dataset and test dataset using the 80/20 "rule", *i.e.*, 80% of the dataset is used for training and the remaining 20% is used for testing. The model was trained on an NVIDIA GeForce GTX 1070 for 100 epochs with a batch size of 128 and using the Adam optimizer [25] with a learning rate starting on 0.0001. Resorting to a learning rate scheduler, the learning rate is dynamically reduced by a factor of 0.1 whenever the validation loss stops decreasing for more than 10 epochs, which indicates that the learning has stagnated.

Table 5.1: Class distribution of the training dataset used for training the LiDAR CNN.

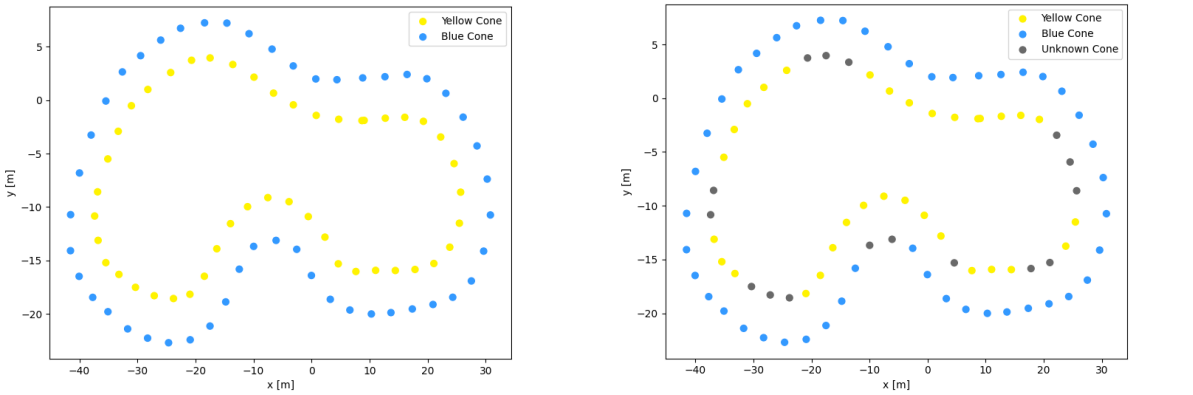
Class	**Class Distribution (%)**	**Number of Images**
Blue Cone	33.59	5 145
Yellow Cone	34.76	5 323
Unknown	31.65	4 847
Total	100	15 315

To validate the importance of the using the LiDAR CNN, a study was conducted on the number of cones that it helps classifying. For this, the full perception pipeline was run over raw data gathered from a lap performed around a Formula Student Driverless track with and without using the LiDAR CNN. The results of this study, represented in Table 5.2, show that using the LiDAR CNN allows identifying 16 more cones in a track with a total of 93 cones, which represents an increase of 17.2% in the number of identified cones. Notice that the number of identified yellow cones is greater than the number of blue cones because, as shown in Fig. 5.2, the track is mostly composed of right-hand turns, which means that the yellow cones will be the less seen by the camera. From Fig. 5.2 it is possible to conclude that the LiDAR CNN is essential to identify the cones placed in the inner side of the curves, where mostly cones from the opposite class can be seen by the camera.

Table 5.2: Comparison between the number of cones identified using and without using the LiDAR CNN.

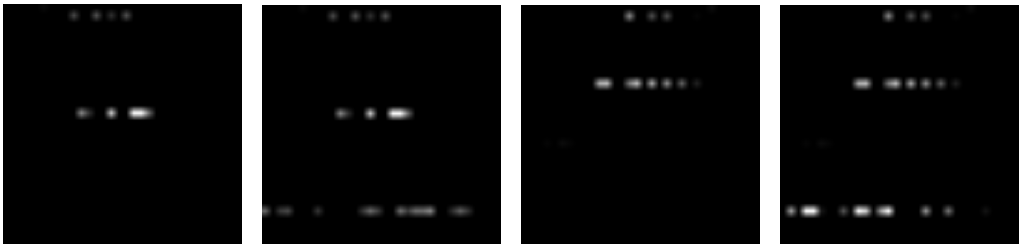
		Total Number of Cones		
Method	**Class**	**On Track**	**Identified**	**Identified by LiDAR CNN**
With LiDAR CNN	Blue Cone	48	48	2
	Yellow Cone	45	45	14
	Total	**93**	**93**	**16**
Without LiDAR CNN	Blue Cone	48	46	-
	Yellow Cone	45	31	-
	Total	**93**	**77**	**-**

In order to validate the importance of the cone reconstruction step detailed in Sec. 4.1.4, which retrieves cone points that were incorrectly removed in the ground removal step, detailed in Sec. 4.1.2, a comparison between the LiDAR CNN results with and without performing this reconstruction step was conducted. To perform this comparison, a new test dataset was created with the same cone images used in the splitted test dataset but without performing cone reconstruction. Fig. 5.3 shows two examples of cone images used in the created test dataset and the two correspondent cone images present in the splitted test dataset. The results obtained from the trained model on both test datasets are detailed in Table 5.3 and in the confusion matrices represented in Fig. 5.4.



(a) Cones detected using the LiDAR CNN. (b) Cones detected without using the LiDAR CNN.

Figure 5.2: Comparison between the number of cones detected using and without using the LiDAR CNN.



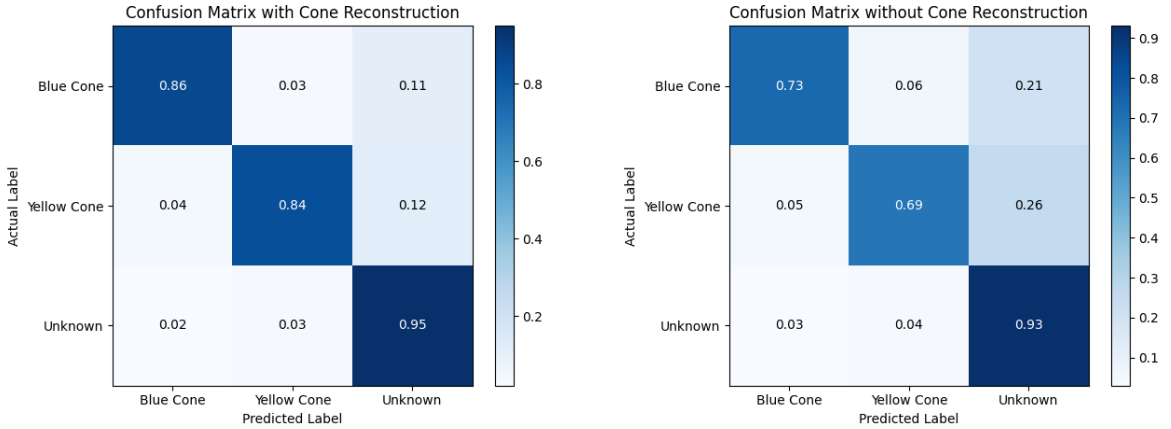
(a) (b) (c) (d)

Figure 5.3: Example of cone images used in the test datasets used to compare the LiDAR CNN results with and without performing cone reconstruction. Figures (a) and (b) show the same blue cone before and after the cone reconstruction step, respectively, while Figures (c) and (d) show a yellow cone before and after cone reconstruction.

Observing Table 5.3 and Fig. 5.4, one can conclude that the overall results obtained when cone reconstruction is not performed is around 10% worse across all performance metrics. As shown in Fig. 5.3, the input images of the LiDAR CNN when no reconstruction is performed, lack the bottom-most layer of points. This is especially noticeable for further away cones which, due to the not so fine vertical resolution of the LiDAR, lack on the number of beams that hit them. For these cones, the missing layer of points breaks the intensity pattern used as the base for this CNN, turning the intensity pattern of the cones into low-high and high-low intensity patterns, respectively for a blue cone and a

Table 5.3: Results of the trained LiDAR CNN model on the test dataset.

Method	**Class**	**Precision (%) ↑**	**Recall (%) ↑**	**F1 (%) ↑**	**Accuracy (%) ↑**
With Cone Reconstruction	Blue Cone	93.34	85.98	89.51	87.77
	Yellow Cone	94.38	84.02	88.90	
	Unknown	76.51	94.93	84.73	
Without Cone Reconstruction	Blue Cone	90.08	73.03	80.66	77.17
	Yellow Cone	89.14	69.01	77.79	
	Unknown	60.87	92.99	73.58	



(a) Confusion Matrix with Cone Reconstruction (b) Confusion Matrix without Cone Reconstruction

Figure 5.4: Resultant confusion matrices from the LiDAR CNN on the test dataset, with and without the cone reconstruction step from the LiDAR processing.

yellow cone. These intensity patterns are much less distinguishable, which justifies the increase of false positives for the unknown class, evidenced by its 15% lower precision. This means that if the cone reconstruction step is not performed, the will be more blue and yellow cones incorrectly classified as belonging to the unknown class. Notice that for this classification task, a false positive in the blue and yellow classes is far worse than a false positive in the unknown class because classifying a blue cone as a yellow, and *vice-versa*, can drastically influence the path planner module that follows. However, the increase in the false positives for the unknown class means that blue and yellow cones will be identified much fewer times by being associated to the unknown class, which in the situations that this CNN is required, may be critical.

5.3 Camera Classification

The CNN used to classify the remaining generated cone proposals, those that fall inside the camera's FoV, and whose architecture is represented in Fig. 4.17, was trained using a custom dataset of over 100 0000 images. This dataset was originally only composed of images from FSOCO [11] – a collaboration between formula student teams that aims to accelerate the development of camera-based solutions in the context of Formula Student Driverless - but has since been enlarged with more images gathered from runs on different testing sites, with different weather/light conditions and with different cameras. This dataset was also split into training dataset and test dataset using the 80/20 "rule", which means that 80% of the dataset is used for training and the remaining 20% is used for testing. The model was trained on an NVIDIA GeForce GTX 1070 for 250 epochs with a batch size of 64 and using the Stochastic Gradient Descent (SGD) optimizer [33] with a learning rate starting on 0.00001. Similarly to the train performed on the LiDAR CNN model, a learning rate scheduler was used to dynamically reduce the learning rate by a factor of 0.1 whenever the validation loss stops decreasing for more than 10 epochs. The class distribution of the training dataset is detailed in Table 5.4, where it is possible to observe that the orange and big orange cone classes are around 10% less represented than the remaining classes. This is due to the dataset having been created using data from Formula Student Driverless tracks, which, in general, have far fewer orange and big orange cones, as they are only used on stopping zones and on the starting line.

Table 5.4: Class distribution of the training dataset used for training the camera CNN.

Class	**Class Distribution (%)**	**Number of Images**
Blue Cone	24.65	25 452
Yellow Cone	25.76	26 598
Orange Cone	15.21	15 694
Big Orange Cone	13.89	14 341
Unknown	20.49	21 154
Total	**100**	**103 239**

As mentioned in Sec. 4.2.3, the architecture of the feature extractor of this CNN was reduced from four convolutional layers to only three. This design choice was based on a study conducted on the camera CNN performance in terms of classification and

computation costs. The results obtained from this study are represented in Table 5.5, Table 5.6 and Fig. 5.5.

Table 5.5: Average inference time of the camera CNN on GPU and CPU for both architectures.

	Average Inference Time (ms) ↓	
Architecture	**GPU**	**CPU**
4 Convolutional Layers	1.9	4.5
3 Convolutional Layers	1.6	2.3

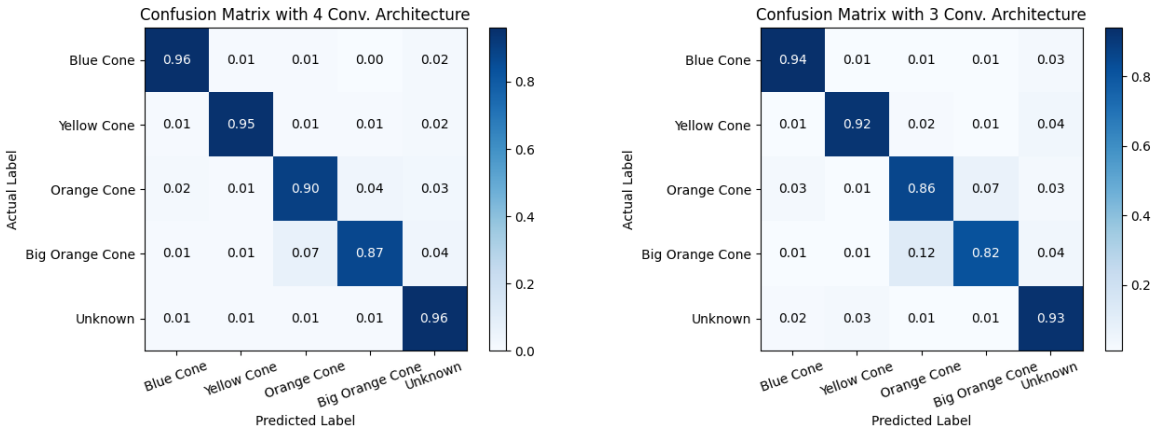
Table 5.6: Results of the trained camera CNN model on the test dataset.

Architecture	**Class**	**Precision (%) ↑**	**Recall (%) ↑**	**F1 Score (%) ↑**	**Accuracy (%) ↑**
4 Convolutional Layers	Blue Cone	96.32	96.00	96.16	93.58
	Yellow Cone	97.25	94.99	96.11	
	Orange Cone	89.48	90.01	89.74	
	Big Orange Cone	91.00	87.00	88.96	
	Unknown	90.69	95.99	93.26	
3 Convolutional Layers	Blue Cone	94.84	94.00	94.42	90.40
	Yellow Cone	95.37	91.99	93.65	
	Orange Cone	83.23	86.00	84.59	
	Big Orange Cone	86.53	82.00	84.20	
	Unknown	87.26	93.84	90.43	

Regarding the inference times of the CNN, represented in Table 5.5, it is possible to conclude that the architecture that resorts only to 3 convolutional layers is faster on both GPU and CPU. Although the decrease in inference time is less noticeable when the model is run on GPU, decreasing only 0.3 ms, on CPU the inference time decreases by almost 50%, from 4.5 ms to 2.3 ms.

From Table 5.6 and from the confusion matrices represented in Fig. 5.5, it can be observed that both models achieve over 90% on almost all performance metrics for almost all classes. It is also possible to observe that the expected loss of performance between the two models is not very accentuated, decreasing only around 3% on all performance metrics, except for the Orange and Big Orange Cone classes, when the model with three convolutional layers is used. The bigger performance decrease in these two classes can be justified by the fact that these two cones are only distinguishable by the extra white stripe on the big orange cones and, since a deeper CNN is able to extract deeper features, the model with four convolutional layers can extract deeper features and thus correctly classify these classes more often. Furthermore, the lower performance on the orange and

big orange cones in both models can also be explained by the class distribution of the training dataset, as these cones are around 10% less represented. This small decrease in the classification performance of the camera CNN allied to the possibility of being able to run the CNN on CPU, justifies the choice of decreasing the feature extractor architecture of the camera CNN.



(a) Confusion Matrix with a 4 Convolutional Layers Architecture.

(b) Confusion Matrix with a 3 Convolutional Layers Architecture.

Figure 5.5: Resultant confusion matrices from the camera CNN on the test dataset. In (a) it is represented the confusion matrix for the model with a 4 convolutional layers architecture and, in (b), the confusion matrix for the model with a 3 convolutional layers architecture.

5.4 LiDAR Processing Ablation Study

To justify the need of all the LiDAR processing steps described in Sec. 4.1, an ablation study was conducted on the implemented methods presented in the LiDAR pipeline. The results from this ablation study are depicted in Tables 5.7 and 5.8.

Initially, the pass-through filter used to trim the LiDAR FoV was removed, significantly increasing the number of cone proposals identified and, consequently, the number of cone proposals sent to both classification CNNs. Removing the pass-through filter, which, per iteration, only takes an average of 1.65 ms to trim the LiDAR FoV, causes an increase in the euclidean clustering average execution time, increasing it by over 6

ms. Since the euclidean clustering method is responsible for grouping the point cloud into clusters, the greater is the LiDAR FoV, the greater will be the number of points in the point cloud, which explains this increase in the execution time of the algorithm and in the number of cone proposals identified. Furthermore, these increases in the number of points in the point cloud and in the number of cone proposals identified, cause the subsequent LiDAR processing and classification steps to also significantly increase their execution times per iteration, as there are more information to be processed and more cone proposals to be classified. Consequently, the average execution time per iteration of the full perception pipeline increases by over 2300%, from which can be concluded that the pass-through filter is essential in the proposed LiDAR pipeline.

Table 5.7: Number of cone proposals and correspondent distribution per iteration for each CNN after removing certain LiDAR processing steps.

Method	**Cones on Track**	**Number of Cone Proposals**	**Avg. Number of Cone Proposals per Iteration**	
			LiDAR CNN	**Camera CNN**
Full LiDAR Pipeline	147	168	2.25	5.24
Without Pass-through Filter	147	4 937	44.05	176.19
Without Ground Removal and Cone Reconstruction	147	109	1.46	3.39
Without Cone Validation	147	214	2.86	6.68

The second method removed from the LiDAR processing was the ground removal and, consequently, the cone reconstruction, as it is no longer necessary. Removing the ground removal step from the LiDAR pipeline decreases the number of identified cone proposals and, consequently, the number of classifications performed by both classification CNNs. Although these decreases cause a reduction on the execution time per iteration of the classification task, which is explained by the reduced number of cone proposals to be evaluated, this reduction in the number of cones means that less cones are being identified. Since the ground is not being removed, the euclidean clustering algorithm tends to associate the ground points to cones that are closer to them, which can cause the group of points to exceed the defined maximum number of points that a cone can have and therefore not being considered as clusters. Furthermore, by removing the ground removal step, the number of points in the point cloud significantly increases, even more than when the pass-through filter is removed. As previously explained, this causes a significant increase in the average execution time per iteration of the euclidean

Table 5.8: Analysis of the average execution time per iteration after removing certain LiDAR processing steps.

Method	Average Execution Time per Iteration (ms)							Total Execution Time (ms)	
	Method	Euclidean Clustering	Full LiDAR Pipeline	LiDAR CNN		Camera CNN			
				GPU	CPU	GPU	CPU	GPU	CPU
Full LiDAR Pipeline	4.13	0.33	4.13	3.6	5.18	8.38	12.05	16.11	21.36
Without Pass-through Filter	1.65	6.55	37.23	70.48	101.32	281.9	405.24	389.61	543.79
Without Ground Removal and Cone Reconstruction	2.16	35.71	37.30	2.34	3.36	5.42	7.80	45.06	48.46
Without Cone Validation	0.02	0.33	4.14	4.58	6.58	10.69	15.36	19.41	26.08

clustering algorithm (over 35 ms) and, consequently in total execution time of the full perception pipeline. Therefore, since removing this step decreases the number of cones identified while increasing the total execution time, it is essential to maintain it on the LiDAR processing.

Finally, it was assessed if it was worth performing the cone validation step that allows to remove most of the outliers from the cone proposals identified. Removing this method, slightly increases the number of cone proposals identified and, consequently, the number of classifications performed by the classification CNNs. As this method is performed over the already clustered cone proposals, the average execution time per iteration of the euclidean clustering remains unchanged. This extra step only takes an average of 0.02 ms and it is the only extra delay added to the LiDAR pipeline, *i.e.*, it does not cause any further delay on the LiDAR pipeline. However, with this small delay comes a decrease of over 3 ms on the total execution time of the perception pipeline, as the number of classifications performed by both the LiDAR CNN and the camera CNN are reduced.

5.5 Tracking

In order to test the influence of the developed tracking algorithm, the proposed perception pipeline was run over raw and unseen data from an acceleration track in a controlled environment, where it is known that, for the first 75 m, cones at the left side of

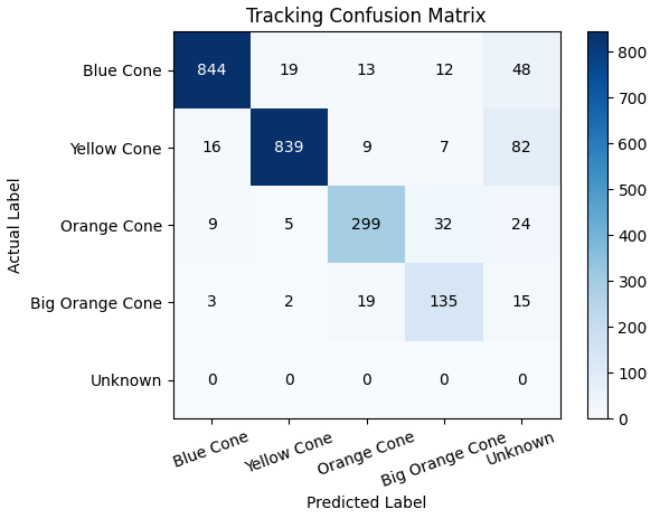


Figure 5.6: Confusion matrix from raw data of an acceleration track.

the car correspond to blue cones and cones at the right side correspond to yellow cones. It is also known that at the beginning there are four big orange cones, that marks the starting line, and at end there are several orange cones that mark the stopping zone. The confusion matrix represented in Fig. 5.6 shows the number of correctly and incorrectly classified coned for each class. Notice that, since this test was done under a controlled environment in a large open space, there is no samples from the Unknown class. The number of color corrections made by the tracking algorithm is depicted in Table 5.9, from where it is possible to conclude that 92% of the cones whose color was incorrectly classified, got its color corrected by the tracking algorithm. Most of the remaining misclassifications belong to cases where the cone was associated with the Unknown class several times at the beginning, which can happen for further cones if the camera-LiDAR calibration is not perfect. For these cones, only when the respective color surpasses the unknown classifications, the color is corrected.

Table 5.9: Number of misclassifications per class and number of color corrections.

Class	**Number of Misclassifications**	**Number of Color Corrections**
Blue	92	84
Yellow	114	106
Orange	70	65
Big Orange	39	36
Total	**315**	**291**

Chapter 6

Conclusion

The main goal of this thesis was to develop a full perception pipeline to be implemented in the Formula Student Técnico (FST) autonomous prototype. This pipeline must perform in real-time given not only the constraints that define the Formula Student tracks but also the low availability of perception sensors. Since the layout of the tracks is unknown, the perception pipeline must be able to accurately detect all the different colored cones in order to safely navigate through the track at the fastest pace possible.

The perception pipeline proposed in this thesis exploits the best features from the two available perception sensors (a camera and a LiDAR) by combining the cone's position estimated by the LiDAR pipeline, which processes the raw LiDAR data to generate cone proposals, with the cone's color likelihood estimated by either the camera CNN, a custom CNN that classifies the image patches of the correspondent generated cone proposals, or the LiDAR CNN, a custom CNN that classifies the color of the cone proposals' cluster resorting to the intensity pattern that distinguishes the different cones. Furthermore, given the simplicity of the custom CNNs used, a Nearest Neighbor based tracking algorithm was developed to correct the estimated cone's color taking into account the estimated color of previous observations.

The results show that the proposed perception pipeline is able to accurately detect and classify the cones even when running the pipeline on CPU, something that can bring benefits on the Efficiency event of the Formula Student Competitions. Furthermore, they also show the importance of the developed LiDAR CNN in order classify the detected cones that do not appear in the camera image, which most of times correspond to the cones places in the inner side of the curves. Moreover the developed tracking algorithm

has shown to help correcting the color of misclassificated cones, avoiding the propagation of those misclassification to the path planning algorithms that follow.

Further work can extend the taken approach to detect and classify other objects in other environments by relaxing the assumptions made for the Formula Student context.

Bibliography

[1] P. Adarsh, P. Rathi, and M. Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 687–694, 2020. 9, 13

[2] A. Agarwal, K. Brandes, R. Chang, K. Doherty, M. Kabir, K. Strobel, N. Stathas, C. Trap, A. Wang, and L. Kulik. Robust, High Performance Software Design for the DUT18D Autonomous Racecar, 2018. 11, 12

[3] A. Barjatya. Block matching algorithms for motion estimation. *IEEE Transactions Evolution Computation*, 8:225–239, 01 2004. 10

[4] T. Chen, Z. Li, Y. He, Z. Xu, Z. Yan, and H. Li. From perception to control: an autonomous driving system for a formula student driverless car, 2019. 11, 12

[5] K. Choeychuen, P. Kumhom, and K. Chamnongthai. An Efficient Implementation of the Nearest Neighbor Based Visual Objects Tracking. In *2006 International Symposium on Intelligent Signal Processing and Communications*, pages 574–577, 2006. 39

[6] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, 20:273–297, 1995. 13, 32

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893, 2005. 13

[8] K. Derpanis. Overview of the RANSAC Algorithm, 2005. 12, 22

[9] A. Dhall. Real-time 3D Pose Estimation with a Monocular Camera Using Deep Learning and Object Priors On an Autonomous Racecar, 2018. 10

[10] A. Dhall, K. Chelani, V. Radhakrishnan, and K. Krishna. LiDAR-Camera Calibration using 3D-3D Point correspondences, 2017. 30

[11] D. Dodel, M. Schötz, and N. Vödisch. FSOCO: The Formula Student Objects in Context Dataset, 2020. 38, 49

[12] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981. 10, 22

[13] Formula Student Germany. FS Rules 2020, 2020. 2, 5, 23, 33

[14] Formula Student Germany. FSG Competition Handbook 2021, 2021. 4, 5

[15] Formula Student Técnico Lisboa. Autonomous Design Report - FST10d, 2020. 6, 12, 13

[16] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. 30

[17] A. Geiger, F. Moosmann, O. Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. *2012 IEEE International Conference on Robotics and Automation*, pages 3936–3943, 2012. 30

[18] N. Gosala, A. Buhler, M. Prajapat, C. Ehmke, M. Gupta, R. Sivanesan, A. Gawel, M. Pfeiffer, M. Burki, I. Sa, and et al. Redundant Perception and State Estimation for Reliable Autonomous Racing. *2019 International Conference on Robotics and Automation (ICRA)*, May 2019. 9, 11, 12

[19] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. 29

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition, 2015. 10

[21] M. Himmelsbach, F. Hundelshausen, and H. Wuensche. Fast segmentation of 3D point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565, 2010. 11

[22] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. 35

[23] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015. 35

[24] J. Kabzan, M. de la Iglesia Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart. AMZ Driverless: The Full Autonomous Racing System, 2019. 9, 10, 34

[25] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2017. 45

[26] L.Fernandez, V. Avila, and L. Gonçalves. A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors, 2017. 30

[27] N. Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. 10

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019. 19

[29] J. Pedoeem and R. Huang. YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers, 2018. 9

[30] M. Quigley. ROS: an open-source Robot Operating System. In *IEEE Intelligent Conf. Robotics and Automation (ICRA) 2009*, 2009. 19

[31] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger, 2016. 9

[32] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018. 9

[33] S. Ruder. An overview of gradient descent optimization algorithms, 2017. 49

[34] B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, 2011. 21

[35] R. Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *KI - Künstliche Intelligenz*, 24:345–348, 2010. 23

[36] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic self calibration of a camera and a 3D laser range finder from natural scenes. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4164–4169, 2007. 30

[37] K. Strobel, S. Zhu, R. Chang, and S. Koppula. Accurate, Low-Latency Visual Perception for Autonomous Racing:Challenges, Mechanisms, and Practical Solutions, 2020. 9, 10

[38] P. Sturm. *Pinhole Camera Model*, pages 610–613. Springer US, Boston, MA, 2014. 29

[39] The European Commission. Mobility and transport: Intelligent transport systems, 2020. 1

[40] The European Commission. Road safety: 4,000 fewer people lost their lives on eu roads in 2020 as death rate falls to all-time low, 2020. 1

[41] H. Tian, J. Ni, and J. Hu. Autonomous Driving System Design for Formula Student Driverless Racecar. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6, 2018. 12

[42] M. Wang, Y. Li, and B. Zheng. A camera calibration technique based on OpenCV. In *The 3rd International Conference on Information Sciences and Interaction Sciences*, pages 403–406, 2010. 29

[43] H. Yong and X. Jianru. Real-time traffic cone detection for autonomous vehicle. In *2015 34th Chinese Control Conference (CCC)*, pages 3718–3722, 2015. 10

[44] M. Zeilinger, R. Hauk, M. Bader, and A. Hofmann. Design of an Autonomous Race Car for the Formula Student Driverless (FSD), 2017. 10

[45] Z. Zhang. *Iterative Closest Point (ICP)*, pages 433–434. Springer US, 2014. 30