UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

LiDAR cone detection as part of a perception system in a Formula Student Car

A Degree Thesis

Submitted to the School

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Arnau Roche López

In partial fulfilment

of the requirements for the degree in Telecommunications Technologies and Services Engineering

Advisor: Josep Ramon Casas Pla

Barcelona, June 2019

Abstract

This project seeks to design and implement algorithms to detect the 3D position and color of cones in a track with a LiDAR sensor, within the framework of the construction of the first electric and autonomous car for Formula Student in Spain. The project includes the sensor distribution in the vehicle, the computational and communications system used, as well as the integration with ROS (Robot Operating System) to work in real time. An introduction to the LiDAR sensor that is used (Velodyne VLP-32C) is realized and a robust detection system is proposed, with different alternatives that can work in parallel: a system with only LiDAR and another with fusion of LiDAR and cameras.

Resum

Aquest projecte cerca dissenyar i implementar algoritmes per detectar la posició 3D i el color de cons en una pista amb un sensor LiDAR, en el marc de la construcció del primer cotxe elèctric i autònom per a Formula Student que es fa a Espanya. El projecte inclou la disposició dels sensors al vehicle, el sistema de computació i comunicació utilitzat, així com la integració amb ROS (Robot Operating System) perquè funcioni en temps real. Es realitza una introducció al sensor LiDAR que s'utilitza (Velodyne VLP-32C) i es proposa un sistema robust de detecció, amb diverses alternatives que poden funcionar en paral·lel: un sistema amb només LiDAR i un altre amb fusió de LiDAR i càmeres.

Resumen

Este proyecto busca diseñar e implementar algoritmos para detectar la posición 3D y el color de conos en una pista con un sensor LiDAR, en el marco de la construcción del primer coche eléctrico y autónomo para Formula Student que se hace en España. El proyecto incluye la disposición de los sensores en el vehículo, el sistema de computación y comunicación usado, así como la integración con ROS (Robot Operating System) para que funcione en tiempo real. Se realiza una introducción al sensor LiDAR usado (Velodyne VLP-32C) y se propone un sistema robusto de detección, con varias alternativas que pueden funcionar en paralelo: un sistema con solo LiDAR y otro con fusión de LiDAR y cámaras.

Dedication

To my family, for all this intensive year dedicated to the project.

To the Driverless UPC 2018-2019 team members, for trusting me to be in charge of an important part of the project and work in an unbeatable environment. I am glad to consider a lot of them as my new friends.

To Perception members, Roger Aylagas, Eloi Bové, Queralt Gallardo and Víctor García, for all the discussions and new approaches raised always in a good atmosphere that has contributed to our goal achievement.

Acknowledgements

I would like to thank my tutor advisor Josep Ramon Casas for his enormous dedication to the project and for the invaluable advices that he has given us all this entire year, as well as for his constant attention.

I would also like to thank you Albert Aguasca for his illusion for the project and his desire to collaborate with all his available resources and help.

This project wouldn't have been the same without all the teammates from Driverless UPC, a group of students from different universities that has been evolving to be a team that will surely achieve their goals.

Revision history and approval record

Revision	Date	Purpose
0	11/05/2019	Document creation
1	22/05/2019	Document revision
2	06/06/2019	Structure revision
3	22/06/2019	Fist part revision
4	25/06/2019	Second part revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Arnau Roche López	arnau.roche@gmail.com
Josep Ramon Casas Pla	josep.ramon.casas@upc.edu

Written by:		Reviewed and approved by:	
Date	25/06/2019	Date	25/06/2019
Name	Arnau Roche López	Name	Josep Ramon Casas Pla
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

The aim of this project is the design, implementation and improvement of cone detection algorithms with LiDAR and cameras as part of a perception system in a Formula Student car. The system has to determine in real time the 3D coordinates of the obstacles present in the environment, in order to use that information for the car localization and mapping (SLAM) as well as the computation of the optimal trajectory. The project is part of the Formula Student Driverless design competition, whose goal is to design and build an autonomous formula 1 style race car [Figure 1].

Figure 1: CAT 12D (Xaloc), the first autonomous car for Formula Student in Spain

The global project is carried out at Driverless UPC Barcelona, the recently created Formula Student Driverless team with 22 students from Industrial, Telecommunications, Informatics and Mathematical Engineering, as a synergy of ETSEIB and ETSETB engineering schools. The team purpose is to build the first Formula Student Driverless car in Spain, including the autonomous system to control it, joining different points of view and knowledge.

1.1. Statement of purpose and objectives

The main goal of Driverless UPC is to build a reliable car and compete in Formula Student Germany (https://www.formulastudent.de/), Formula Student Czech Republic (https://www.fsczech.cz/) and Formula Student Spain (http://formulastudent.es/). This project is focused on the perception section, in charge of the cone recognition, so its concrete objectives are:

1. Work with a LiDAR sensor to develop and integrate a preliminary cone detection algorithm with ROS (Robot Operating System) to work in real time.
2. Improve a basic implemented algorithm with new available options and techniques to increase its robustness and speed.

3. Develop a global architecture to fuse the sensor data from a LiDAR and cameras in order to implement an alternative detection algorithm based on the merged data, using the camera image and the LiDAR depth information.

1.2. Requirements and specifications

1.2.1. Requirements

- The vehicle has to perceive information about its environment with the installed sensors (cameras, LiDAR, IMU...) to be processed in real time.
- The system has to identify obstacles in the track, with the ability to distinguish blue, yellow or orange cones with the maximum robustness.
- The processing or fusion of different sensors should take reasonable time for a system that has to work in real time.

1.2.2. Specifications

- The vehicle has to incorporate a LiDAR sensor able to measure the distances to multiple targets around the car with pulsed laser light, using its Time of Flight (TOF). It has to include also two camera devices to perform stereo matching for cone distance detection.
- The LiDAR has to have, at least, approximately 180º of horizontal Field of View (FOV) and 25º of vertical FOV to capture the obstacles located in front of the car.
- The LiDAR has to be located at the front of the vehicle in order to have a clear visibility of the track, and the cameras have to be located in an elevated position to capture a wide Field of View (FOV).
- The vehicle has to contain a computational system with CPU and GPU capabilities to execute the algorithms in less than 80ms/frame (12.5 fps).
- The cone detection algorithms must be able to detect at least 4 cones of the track: two on each side.
- The communication between sensors and the computational system is based on Ethernet (LiDAR) and USB 3.0 (cameras).
- The LiDAR, cameras and computational equipment must be powered directly or with the required voltage adaptations with the vehicle low voltage battery (24V).

1.3. Methods and procedures

This project is part of the global project Driverless UPC Barcelona, which started on September 2018 with a group of 22 students. The dimensions of the project involve a one year period dedication and a division of tasks between different students. In my case, the subject PAE (Projecte Bàsic d'Enginyeria) reflected the first semester occupation, and this Thesis Degree culminates the second semester work with the finished prototype. In PAE, a first preliminary cone detection algorithm with PCL was developed, without ROS integration or color estimation. This Thesis Degree expands the algorithm improving its structure, steps and speed, integrating it with ROS and adding a color estimation method. Moreover, the LiDAR Velodyne VLP-32C was received on March, so from that date there has been the possibility to work with real data. Moreover, a fusion algorithm (LiDAR and cameras) is developed to complement the previous one.

1.4. Work plan

The Perception Team GANTT [Figure 2] is presented with only the detailed tasks directly related to this Degree Thesis (blue) and the milestones [Table 1]:

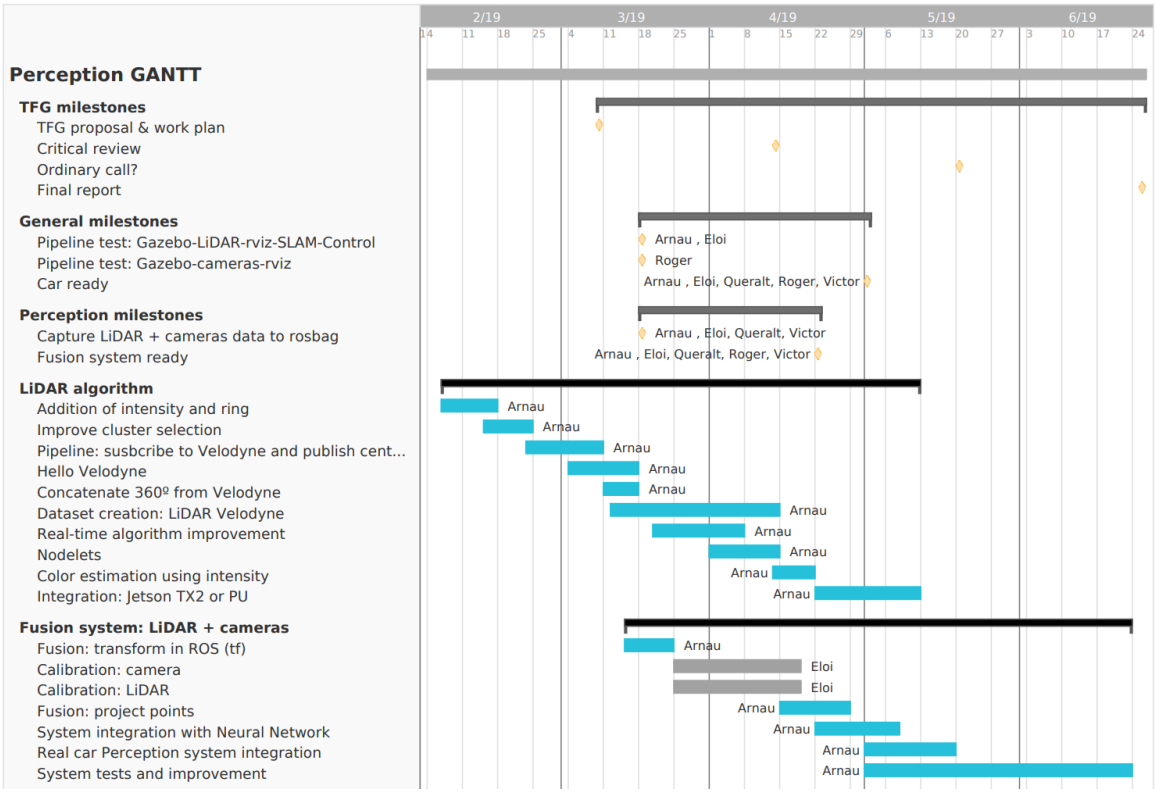

Figure 2: GANTT diagram

Short title	Milestone / deliverable	Date (week)
Pipeline test: Gazebo-LiDAR-rviz-SLAM-Control	System working from detection to control	13/03
Capture LiDAR+cameras to rosbag	LiDAR+cameras dataset	13/03
Fusion system ready	Working system	15/04
Car ready (hardware)	Start testing with the integrated system	01/05

Table 1: Milestones

1.5. Deviations and incidences

A big project implies high dependency between the multiple parts. In this case, some of the pipeline tests have been delayed because of the integration with Estimation & Control team. Moreover, dataset creation with LiDAR and cameras has also been extended in time, because a manual car to capture both sensors in a synchronized way has been “constructed”, and an independent PC has been installed to save the data while moving the car. The camera drivers to capture images to ROS has been an important deviation because of libraries and software incompatibility, as opposed to the LiDAR, which includes a compatible driver with ROS. The general integration with the real car has also

been slightly delayed, because of the Hardware section unforeseen events that have hampered the movement of the electric car in a safe way (we are dealing with a high voltage battery of 600V).

2. State of the art of the technology used or applied in this thesis

2.1. Formula Student

Formula Student is an international design competition among universities from all around the world where the students create, build and develop a formula 1 type race car. In 1981, the first Formula SAE competition was held in Texas (United States), with 6 teams and 40 students. The competition has been growing every year and, since 1998, the British edition of Formula Student is celebrated on Warwickshire, with around 100 international teams.

Nowadays, different competitions are organized in different countries: Australia, Austria, Brazil, Canada, Czech Republic, Germany, Hungary, India, Italy, Japan, Spain, United Kingdom, United States Lincoln, United States Michigan... [Figure 3] [1]

Figure 3: Some of the Formula Student competitions

In Europe, the main and principal competition is Formula Student Germany [2]. In order to be accepted at the event, the teams must comply with several rules established by the competition [3], which include administrative regulations (general requirements, documentation, deadlines…), general technical requirements (design requirements related to chassis, brake system, powertrain, electronics…), technical inspections performed at the competition (electrical inspection, tilt test, rain test, noise test…) and scored events (static and dynamic).

The competition is split into three classes:

- Internal Combustion Engine Vehicles
- Electric Vehicles
- Driverless Vehicles (electric or combustion)

The last category was introduced three years ago and it has less participants than the others, but it will become a crucial one in the following editions, because some events will have to be completed mandatorily in driverless mode in the future for all the categories (also electric and combustion).

At present, teams in FSD participate with their own design and autonomous system, and use a large variety of sensors, mainly cameras and LiDAR's. At Formula Student Germany 2018, 17 of 18 teams had a LiDAR, 19 of 18 teams had cameras and 1 of 18 teams had a radar in their car [4]. Following these previous experiences and the state of the art, our team determined that for the first year of competition it would be interesting to fit the car with a LiDAR and a pair of stereo cameras, and we achieved our goal thanks to our sponsors.

At Formula Student Germany (and also the main European competitions) the teams compete in 3 static and 5 dynamic disciplines [5]:

Static events:

The economic and communication abilities of the students are evaluated [see scores in Figure 4] in the three static events, that include the areas of design, layout, construction, marketing and pricing of a product.

- Business Plan: the team has to justify their car design and how it will be marketed fulfilling the demands of a target group (creation of a company…)
- Cost Report: the team has to elaborate a BOM (Bill of Material) with all the car components and detail some of the manufacturing steps and their costs.
- Engineering Design: the team has to provide a short technical description of the car and its autonomous system and should be able to argue all the design decisions made with the judges.

Dynamic events:

The driverless car performance on the racetrack is also evaluated [Figure 4]. The car should complete all these events in autonomous mode (without driver).

- Track Drive: a track made of cones of unknown trajectory with straights, curves and chicanes has to be completed in one run consisting of ten laps, with the less time possible (the endurance and map planning of the car is evaluated).
- Autocross: a track made of cones of unknown trajectory with straights, curves and chicanes has to be completed in one lap with the less time possible (the speed and autonomous real time "track discovery" is evaluated).
- Efficiency: energy consumption relative to speed in the track drive event is recorded to give the team a score.
- Acceleration: the time for an acceleration in a 75m straight racetrack is measured.
- Skid Pad: an 8-shaped circuit with 2 laps in each circle has to be performed in the less time possible.

In FSD, the tracks are delimited with yellow cones on the right and blue cones on the left, so the recognition of track boundaries is easier for the teams. There are also orange cones delimiting special areas, such as the starting points or the braking zones.

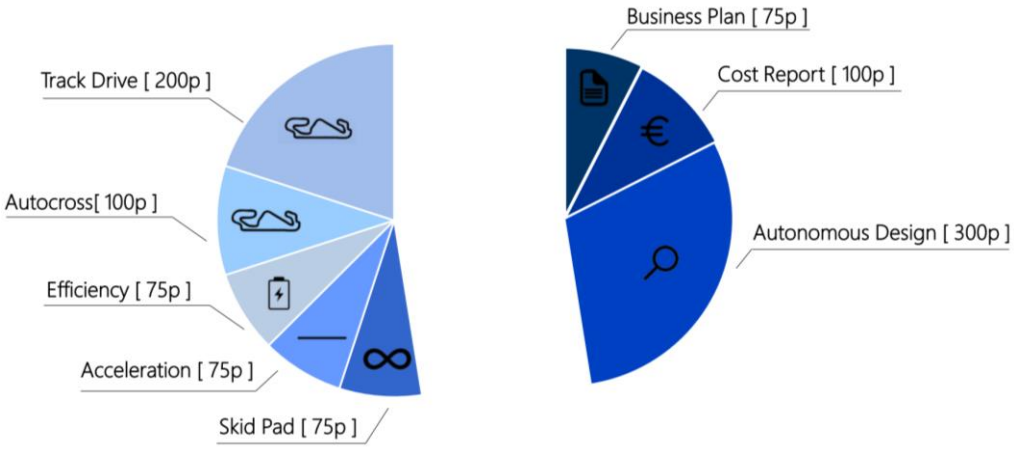

Figure 4: Maximum points awarded in Formula Student Driverless competition

2.2. Formula Student team

The development of an autonomous race car for Formula Student is a multidisciplinary task because it involves a lot of technical fields: the monocoque design and construction, the mechanical, dynamical and aerodynamic characterization, electronics, powertrain, autonomous design… In our case, we use a monocoque and the electronics inherited from the electric team (CAT-10e and CAT-11e), and we are focused on the autonomous adaptation of a vehicle. Then, the team is composed of the following sections [Figure 5]:

- Hardware: dedicated to all the mechanical components and their specifications (batteries, motors, electrical supply…) to convert the car into an autonomous one.
- Perception: intended to user all the car sensors (cameras and LiDAR) and their data to process it and identify the position and color of cones in the track.
- Estimation & Control: covers the use of the data processed by the perception team to estimate and compute a trajectory for the car, using a SLAM system and controlling the car actuators (brake, steering wheel…)
- Marketing and Management: includes the administration part of the team (marketing, social networks, car renders…)

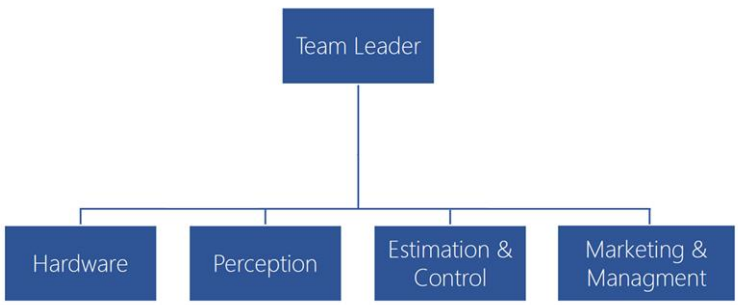

Figure 5: Team sections structure

This structure is based on some of the better Formula Student Driverless teams, such as AMZ (Akademischer Motorsportverein Zürich) from ETH Zürich [6].

2.3. Autonomous system: perception in a FS car

The autonomous car has to use all the data available to identify the cones present in the track and determine the optimal trajectory. As mentioned above, blue and yellow cones identify the right and left track limits, and orange cones are destined to specific points (starting point, brake zones…) [see Figure 6].

Figure 6: Orange, blue and yellow cones distribution in a FSD track

Regarding the autonomous system [Figure 7], the environment is perceived with the LiDAR and stereo cameras in terms of a point cloud and two images, and this data is processed to obtain the cone coordinates and color. This information is fused with the IMU+GPS data to locate the car in the track and create a map of the cones, that will be used to compute a trajectory. Then, the output of "Estimation & Control" is the torque and heading angle needed to control the motor, the brake and the steering wheel actuator, respectively. Finally, the motor and the brake and steering actuators are controlled following the previous results.

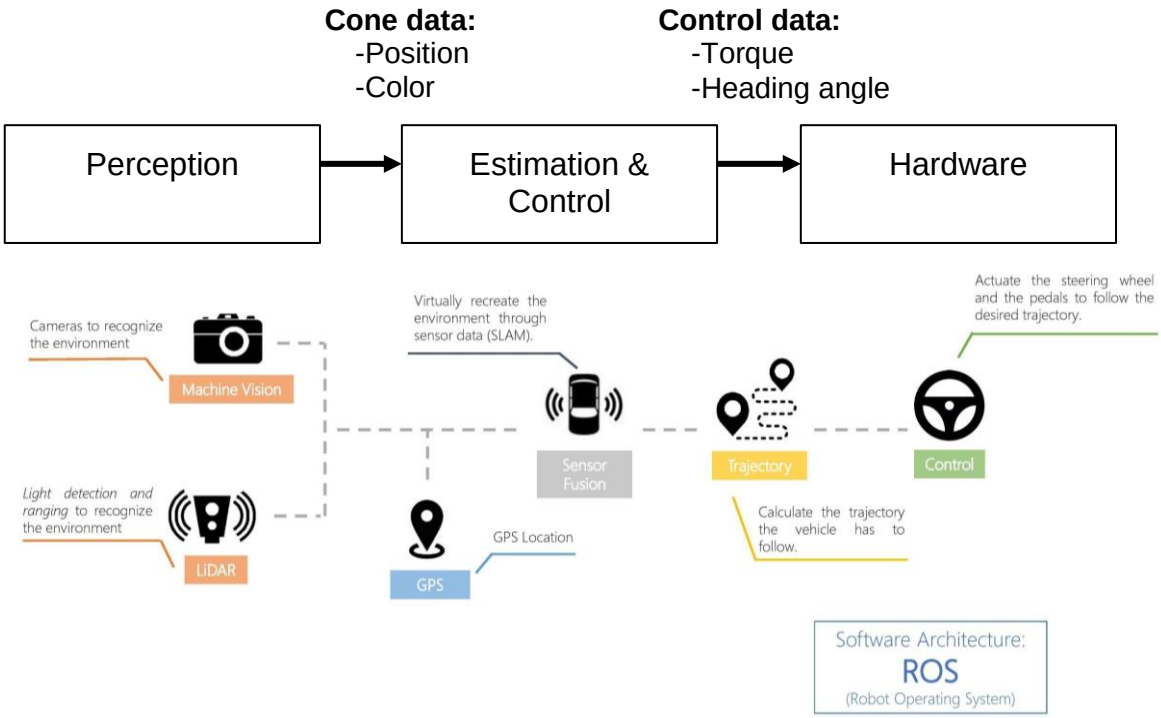

Figure 7: Autonomous system

This degree thesis is related to the perception section. As it has been explained, its goal is to obtain the 3D coordinates and color of the cones present in the track using the raw data provided by the available sensors (LiDAR and cameras). Then, the section is conceived as a “black box”:

Figure 8: Black box analogy of the perception section

2.4. Image and point cloud processing

The perception section has to deal with image and point cloud processing.

For image processing, OpenCV [7] library and Tensorflow [8] are the main tools used. In the first case, high number of available algorithms implemented are a valuable reason to choose this library; in the second one, this option facilitates the implementation of a Neural Network in GPU for image cone detection, to increase the computational speed.

For point cloud processing, the best option is Point Cloud Library (PCL). It is a project presented in 2011 [9], which has been growing in this recent years, but it is still less developed than OpenCV.

When working with 3D and 2D sensors together, sensor fusion appears as an option to implement. Nowadays, calibration between LiDAR sensors and cameras is a cutting-edge challenge, but in this project we deal it exploring the performance of an Automatic Calibration algorithm released in 2017 [10].

Sensor data processing can be faced in many different ways, and the numerous FSD teams and their multiple solutions are examples of that. Munich Motorsport used only a mono camera to detect the 3D positions of the cones, using basic triangulation and the known cone dimensions established by the competition [11]; AMZ Driverless has developed a complete and complex system with mono camera, stereo camera, LiDAR, wheel speeds, IMU and other sensors, which has led them to be the best driverless team in FSGermany 2017, FSItaly 2018 and FSGermany 2018 [12].

2.5. Autonomous driving

Most likely, the automotive sector will evolve towards electric and autonomous driving in the coming years. This project is focused on converting an electric car into an autonomous one, taking advantage of the latest sensor technologies and the ROS operating system, which is widely used in robotics and in real-time applications.

3. Methodology / project development:

3.1. System architecture

The perception system is composed of 2 main sensors: a LiDAR (Laser Imaging Detection and Ranging) and a pair of stereo cameras.

3.1.1. LiDAR

LiDAR is an optical remote sensing technology to measure the distance from a sensor point to different possible obstacles using the time of flight (ToF) between a laser emission and reflection point. This way, the 3D coordinates of the points where the signal is reflected are located in the scene according to their radial distance and the angle of the optical path, generating a point cloud with all the data captured.

A point cloud is a set of 3D points (space coordinates [*x*,*y*,*z*] and other optional associated photometry information) representing the elements captured by an specific device. In general terms, there are two types of point clouds [Figure 9]:

- Organized point clouds: structured in a 2D array of points with the possibility to be converted to an image plus depth information. From the sensor origin point of view, the points are aligned on a 2D grid filled up with elements in all possible positions. This is the type of data provided by range sensors such as Kinect, Xtion, RealSense, iSense, Orbec, Swissranger, Occipital, PMD, VicoVR, Zed...
- Unorganized point clouds: structured in a 1D array of points that represent the spatial elements where the reflections are captured. There is no discretized 2D grid, because there is density variation (zones without points and other zones with a high number of elements). This is the type of data that provide the laser sensors, such as the LiDARs.

The advantage of organized point clouds appears in terms of the algorithms used to process the geometrical information. The organization in a 2D matrix implies that the nearest neighbor operations can be done more efficiently, because the adjacent points in the matrix (for example, pixels) are known [13]. However, in an unorganized point cloud, the distances to determine neighborhoods need to be calculated for every point.

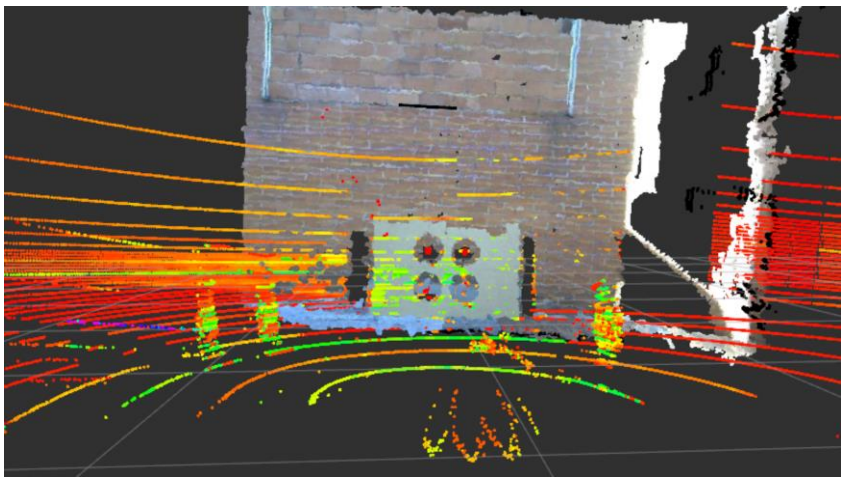

Figure 9: Images from our stereo cameras (organized point cloud) and our LiDAR (unorganized point cloud)

A complete RGB+D image is appreciated from the camera point of view; a point cloud based on layers (with unknown regions where there is no information) is produced by the LiDAR.

The LiDAR used is a Velodyne VLP-32C, offered as an sponsorship by Velodyne LiDAR, an advanced sensor specially designed for automotive applications. Considering our use, some of the specifications [Table 2] to take into account are:

- Field of View (FOV): the observable world seen by the LiDAR (it determines the obstacles of the environment that the sensor can perceive).
- Resolution: the angular possible positions of the laser beams is a key factor that determines the point cloud resolution, because it affects the space discretization (the space points that will receive a laser to find out its depth). Then, the vertical resolution is marked by the number of layers (horizontal scan lines), and the horizontal resolution by the scan frequency.
    - Layers: number of vertical laser beams (as it is a device with horizontal rotation, the number of layers is fixed to 32).
    - Frequency: based on the speed of the internal rotation mechanism that establishes the rate at which the laser scanner completes an entire revolution.
- Working range: maximum range of the device to detect a reflection.

FOV H	FOV V	Resolution	Layers	Frequency	Working Range	Communication
360º	40º (-15º to 25º)	0.1ºH - 0.4ºH 0.33ºV	32	5-20Hz	200m	Ethernet (UDP)

Table 2: Velodyne VLP-32C main specifications [14]

Related to the previous specifications, there's a trade-off between horizontal resolution and frequency [Table 3]: if the internal motor rotates at slower speed (less frequency), the resolution increases as the laser is able to sample more beams in a revolution [Figure 11 right]; otherwise, a high speed will produce less resolution but better "frame rate" (more revolutions/second are provided by the device) [Figure 11 left]. The speed can be set up according to the application: in our case, the LiDAR works at 10Hz to ensure good resolution as well as enough frame refresh while the car is moving forward. At 10Hz, there is a frame every 0.1s, so at a "high" speed of 30km/h (8.33m/s) the car will have advanced 0.833m for every LiDAR revolution. Since the cones are located at approximately 3m, the LiDAR frequency was established to 10Hz to not lose cones (high frame rate) and maintain good resolution.

Figure 10: Velodyne VLP-32C

RPM	Resolution
300	0.1ºH
600	0.2ºH
900	0.3ºH
1200	0.4ºH

Table 3: Velodyne VLP-32C rotation speed – resolution [15]

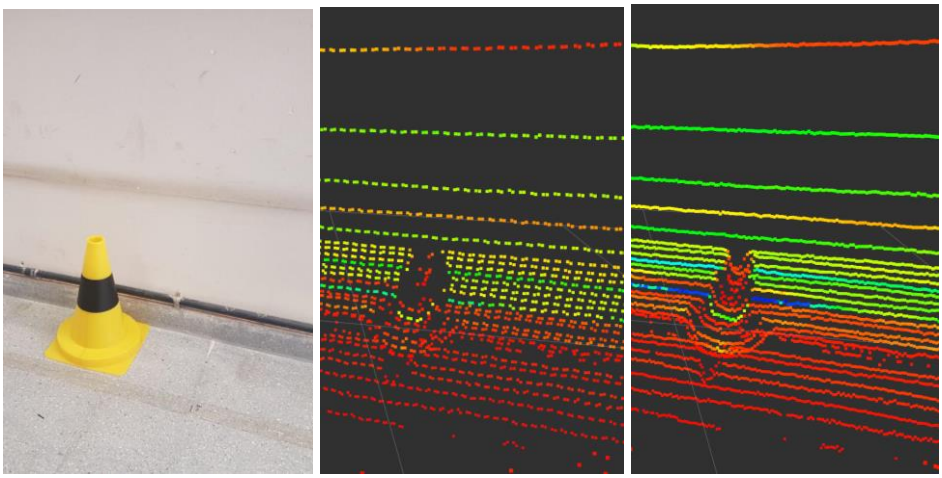

Figure 11: Velodyne VLP-32C resolution comparison: low resolution (300rpm) | high resolution (1200rpm)

Then, Velodyne VLP-32C LiDAR produces an unorganized point cloud with the following data:

x	y	z	intensity	ring
x	y	z	intensity	ring
...	...	...	...	...

Table 4: Velodyne point cloud data structure (.pcd): list of points with coordinates, intensity and ring

The listed points incorporate information of intensity and ring. Intensity is a reflectivity measurement of the point (high intensity is commonly related to metallic objects or clear colors, cases in which the returned laser is less attenuated, for example) and ring indicates the layer number (from 1 to 32). An example is shown in Figure 12.

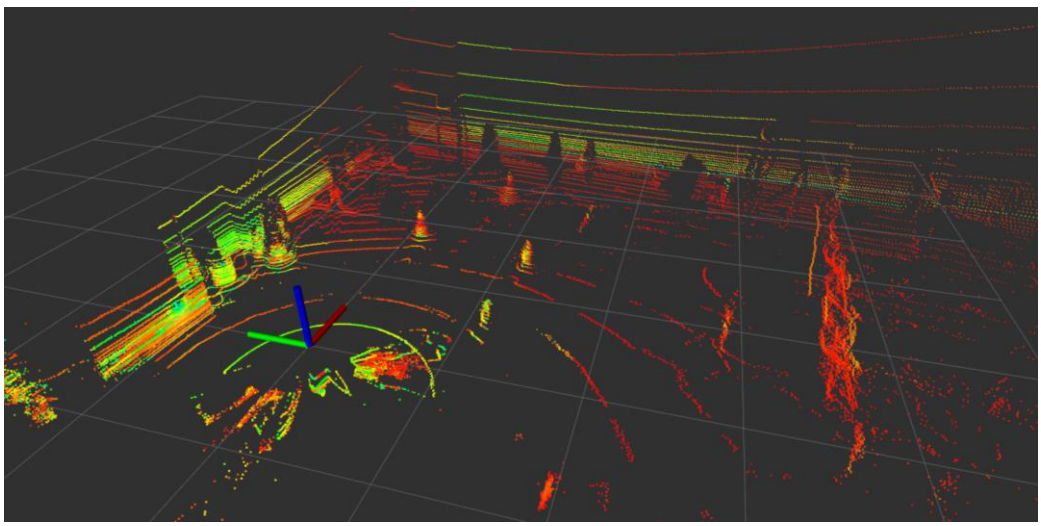

Figure 12: Room corner with cones on the floor: the rings are the LiDAR layers and the colors show the intensity values

3.1.2. Stereo cameras

Another system capable of providing 3D positions is introduced into the car: a pair of stereo cameras DFK33UX252 from The Imaging Source [Figure 13].

Figure 13: Stereo cameras DFK33UX25

Dynamic range (bit)	8 to 12 bits
Resolution (px)	2048x1536
Frame rate (fps)	120
Sensor type	CMOS Pregius
Interface	USB 3.0 and global shutter

Table 5: DFK33UX25 specifications

3.1.3. Computational and communications system

For computer vision algorithms (image and point cloud processing) in real time it is necessary to have high computational resources available. The car has a central computer, Processing Unit (PU), intended to compute all the Control & Estimation algorithms as well as the point cloud Perception algorithms. It is a specific device resistant to vibrations with an Intel i7 processor and 16GB of RAM (Cincoze DX-1000) [Figure 14]. For some image processing such as neural networks, GPU processing provides better results in terms of speed: the car also incorporates an Nvidia Jetson TX2 [Figure 15] for the image related operations.

Figure 14: Cincoze DX-1000

Figure 15: Nvidia Jetson TX2

Both units have Robot Operating System (ROS) (see 3.2.1) installed, and the communication between them is constant. Considering that ROS is based on IP communications, the systems are connected using a switch. Moreover, that switch accepts the possibility to use a router, which adds wireless accessibility to the whole system (it is an important option when testing the car: more than one person can be connected at the same time). The computational load is divided between the two machines: the LiDAR (Ethernet) is received and processed in the PU, and the cameras (USB 3.0) in the Nvidia. What's more, the PU also outputs the control output variables in a low-level signal protocol (CAN), used as a communication with DSpace, that receives all the low-level signals and controls the required electrical and mechanical parts of the

car (communication with PCB's that control actuators, for example). Figure 16 shows a diagram of the complete system.

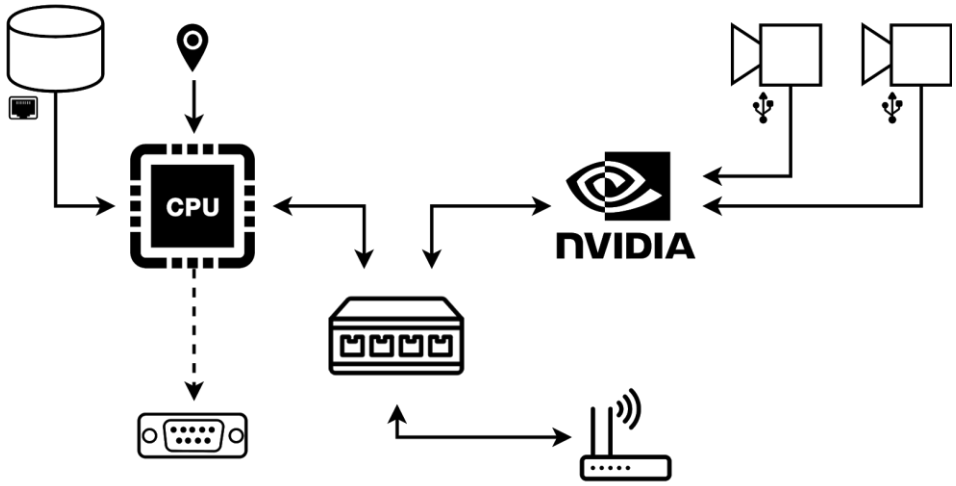

Figure 16: Diagram of the computational and communications system

3.1.4. Car distribution

The LiDAR is located at the front wing. The computational system box is located at the back of the driver's seat. On top of it, the pair of stereo cameras emerge above the driver's seat. Finally, the inertial measurement unit (IMU) and GPS, are below the rear wing [Figure 17]. Dedicated supports have been designed to attach the LiDAR in a way to reduce the vibrations and protect the cameras from water (cameras are not waterproof, so a white cover has been constructed to protect them).

Figure 17: Car sensors (LiDAR at the front and cameras attached to the main hoop, see Figure 64 in Appendix 8.1 for a real car picture)

The distribution of sensors has been designed to comply with the competition rules (there are parts where sensor mountings are not allowed, for example) and maximize the FOV (fields of view). Stereo FOV is approximate 90º horizontal, and LiDAR FOV is 200º (even though the device has 360º of horizontal FOV, the car front wing hides the direct view for the rear angles) [Figure 19].

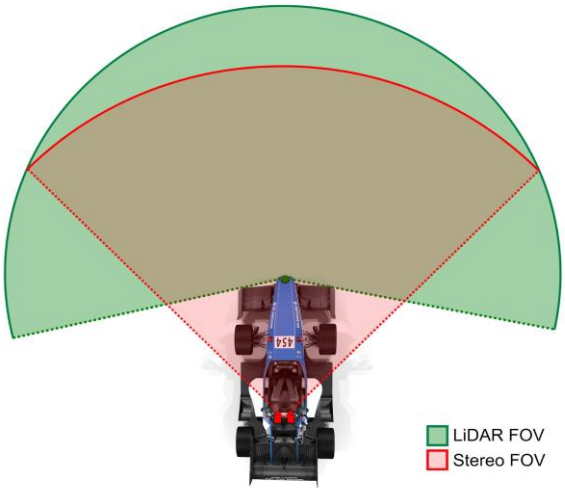

Figure 19: Horizontal Field of View (FOV) comparison between the two sensors

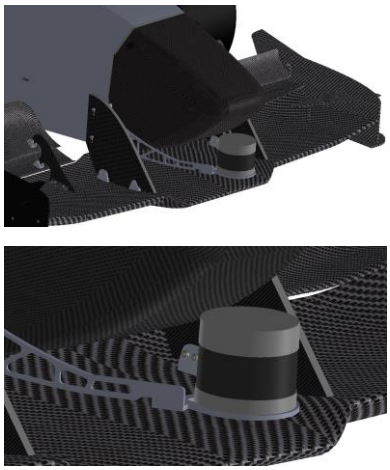

Figure 18: LiDAR support attached to the front wing

3.2. ROS architecture

One of the biggest challenges of this project is the real time computation. The perception algorithms developed are the basis for the car control computations, since they have to work at high speed and in a synchronized pipeline. For that reason, the autonomous system is integrated in a ROS environment.

3.2.1. What is ROS?

Robot Operating System (ROS) is an open-source communication framework that seeks to be a standardized protocol in the robot world [16], and the autonomous car is nothing more than a wheeled robotic system. In this field, many frameworks have been designed for particular purposes, but ROS tries to offer a multi-lingual programming framework with high compatibility for all uses, in a scaled system based on blocks with inputs and outputs, called nodes.

Nodes are the processes that perform computation according to their available inputs and outputs, which are messages published in specific topics. A message is simply the data structure (LiDAR points, camera images, GPS coordinates, positions…) passed between nodes and sent to a given topic, which is a name for a type of messages. Therefore, a node usually subscribes into a topic (with a specific messages) and publishes into another topic the modified data, as shown in the example of Figure 20.

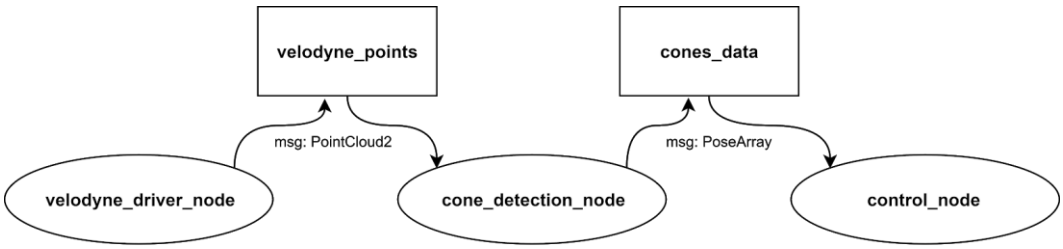

Figure 20: Simplified example of ROS communication

velodyne_driver_node publishes into the topic *velodyne_points* (raw data) and *cone_detection_node* is subscribed to the raw data *velodyne_points* to process it and produce *cones_data*, which will be useful for control.

The communication between nodes is based on peer-to-peer connectivity. Instead of a central server in which all the computations are made, the nodes can be on different host machines and all the messages are transferred via TCP/IP protocol. That topology requires a *master* lookup mechanism to allow processes to find each other at runtime and to provide naming and registration. This provides a great flexibility for integration, as the multiple nodes of the system can be distributed across different computation units.

3.2.2. Why use ROS?

ROS is an ideal system for the project:

- It is free and open source.
- It allows multi-lingual programming (C++ and Python).
- It has some available useful tools, for example:
    - Visualization of the node graph structure in a GUI [*rqt_graph*].
    - Visualization and monitoring of the messages data in real time [*rviz,* Figure 21]
    - Transformation library, to keep track of multiple coordinate frames over time [*tf* and *tf2*, with a graphical example in Figure 22].
    - Dynamic reconfiguration of parameters in real time [*rqt_reconfigure*].
    - Logging and playback of topics, to save the input and output data of all the nodes, and to be able to simulate the same situation with the same sensors data in another moment [*rosbag*].
- It has compatibility with OpenCV and PCL libraries, essential in the perception algorithms.
- It is a modular system that perfectly meets the requirements of a collaborative project (the nodes can be developed by different people following the “black box” analogy).

Working with ROS nodes facilitates the final integration in a large system, as well as the development. Comparisons and improvements between different computer vision algorithms are easier with the possibility to replay the captured messages in a synchronized time. For example, when working with two cameras, a LiDAR and an IMU, a rosbag is recorded to save all the data in the same way it was received, so modifications in the code can be tested in exactly the same conditions.

The visualization tool (RViz) is also a powerful engine to show, in real time, the detected cones computed by the nodes compared to the raw data from the LiDAR or the cameras. That is to say, displaying the 3D bounding boxes where the algorithms signal that there is a cone detected. Furthermore, the control algorithms can also exploit the RViz tool showing the planned trajectory and the car’s position.

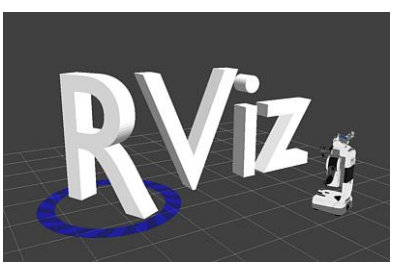

Figure 21: RVIZ logo

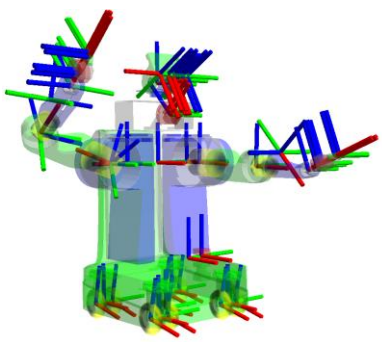

Figure 22: Multiple TF's in an example robot

3.2.3. ROS code structure

When programing in ROS, the user has to define the node structure, which it is based on publishers and subscribers. In our case, the algorithms are organized as follows:

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "cone_detection_node");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("velodyne_points", 1000, pc_callback);
  ros::Publisher cone_pub =
            n.advertise<geometry_msgs::PoseArray>("cones_data", 1000);
  ros::spin();

  return 0;
}

void pc_callback(const sensor_msgs::PointCloud2ConstPtr& cloud_msg)
{
  //Use cloud_msg to detect cones and fill centroids: computer vision algorithm

  geometry_msgs::PoseArray centroids;
  cone_pub.publish(centroids);
}
```

In the code snippet above and in Figure 23, the structure for the node *cone_detection_node* is shown. Inside the *main*, a subscriber of LiDAR points is declared (topic *velodyne_points*), specifying the queue size and the callback function, as well as a publisher (topic *cones_data*). When a new message is published at *velodyne_points*, ROS automatically runs the callback function *pc_callback*. The queue size establishes the number of messages that will be kept waiting if the callback is slower than the arrival frequency of new data: if the queue is full, old messages will be dropped. Inside the callback, the message is available to be processed, so the computer vision algorithm goes here. At the end of the callback, the output information is published (in this case, the cone information about the position and color).

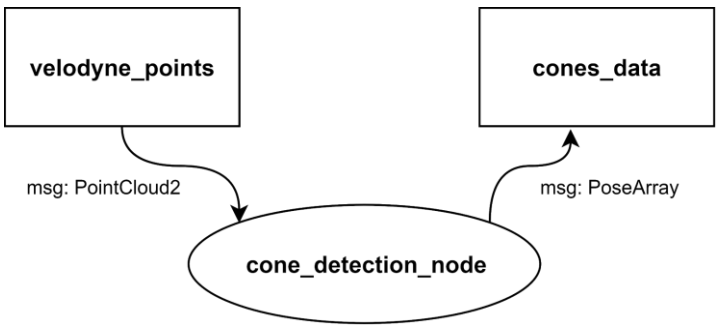

Figure 23: Graphical example of the cone snipped above

This is a simplified structure followed by all the algorithms developed. There are two types of messages used:

- Input messages: ROS messages that wrap the raw data from the sensors (PointCloud2, Image, CompressedImage, CameraInfo…).
- Output messages: the cones detected are exported in PoseArray messages.

The input messages are determined by the sensor and its driver, and the output messages are established by the team as a convention for all the algorithms. We use *PoseArray* with the following structure [Table 6]:

Header	Includes the sequence_ID, timestamp and frame_ID

Pose[]	It is an array of cones with the following information:
	• Position information:
	  ○ X coordinate
	  ○ Y coordinate
	  ○ Z coordinate
	• Probability information
	  ○ Yellow probability
	  ○ Blue probability
	  ○ Orange small probability
	  ○ Orange big probability

Table 6: PoseArray messages structure

This structure is used in all algorithms to allow a final fusion of results

Header in these messages is extremely useful for time synchronization: all the cones detected in a LiDAR frame or camera frame have their timestamp and frame_ID, which allows the entire system to know when the cones were detected and the location of the car in that moment (timestamp) as well as the center of coordinates used to give the distances (frame_ID).

When the results are fused and are provided to Estimation & Control, an extensive message with more information (such as the IMU+GPS data of the detection moment) is used.

3.2.4. Nodelets

ROS modular system is a great architecture to develop a project organized in parts with different functions, which communicate with each other over TCP/IP. In Figure 24 it can be appreciated that the communication is achieved using the network (blue line).

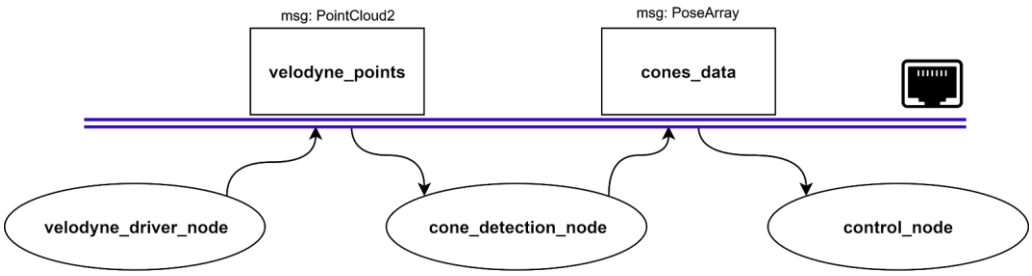

Figure 24: ROS node communication over TCP/IP

The problem appears when the messages are point clouds or images, which imply large amounts of uncompressed data. The node structure sends the messages through the network infrastructure, which works well with small sized position messages, GPS data, transformations… but it can be saturated if it has to deal with huge size messages.

The TCP/IP communication offers modular versatility to the system (the user can choose at which machine the node is executed because only the network communication has to be guaranteed), but if the nodes are in the same host, it is unnecessary to package,

send, receive and unpackage the message every time (this implies a large load in the network buses and card). Therefore, when two ROS nodes are executed in the same physical machine, it is better to just send a memory pointer to the message that will be used directly, taking advantage of zero-copy transference between nodes. To do this, ROS proposes the use of nodelets [17] instead of nodes. Figure 25 shows that the communication is achieved using the CPU and memory pointers (red line), in the case when nodelets run in the same machine.

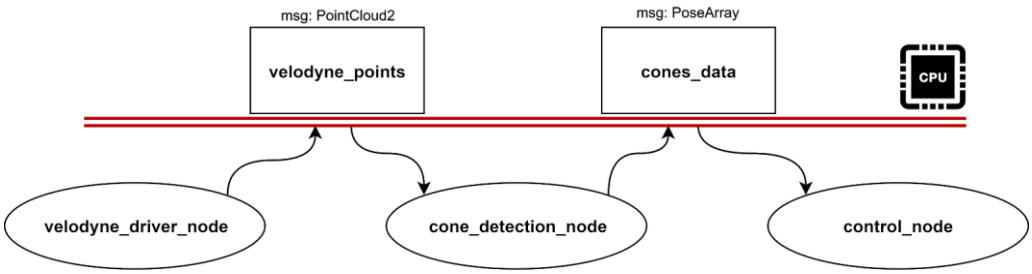

Figure 25: ROS nodelet communication (zero-copy pointers)

In order to do it, there is a special node named *Nodelet manager*, which is able to handle nodelets as threads: if a topic is the output of a nodelet and the input of another, if they are handled by the same nodelet manager, the data will be passed as a pointer; if not, the communication will be done as if they were normal nodes.

It is important to point out that the implementation in nodelets does not improves the computation speed inside the node, but it is a quicker way to transfer information from one algorithm to another. ROS nodelets are only available in C++.

Figure 26: ROS logo

3.3. Cone detection algorithms

Formula Student is a demanding competition where a lot of factors have to work properly to achieve the team goal. Focusing the attention in perception, sensors and algorithms dependency is a key factor for all the autonomous system, since the perception algorithms “give eyes to the car”. For that reason, the design has been made with a clear objective of robustness: multiple algorithms should be able to work in real time to contemplate all the possible failure scenarios [Figure 27].

There is a point cloud processing algorithm that performs 3D cone detection and color estimation using only data from the LiDAR. Moreover, there is a stereo matching block that uses a neural network image detection to propagate the bounding boxes and output also the 3D cone position and color. Finally, there is a sensor fusion algorithm that uses both sensors to produce the same results (cones position and color).

The three algorithms perform cone detection and produce comparable results. A final block deals with the three inputs and chooses which system is better for a final output, according to the meteorological conditions, the computational speed, the sensors availability (e.g. if the cameras are disconnected, the LiDAR algorithm is still working).

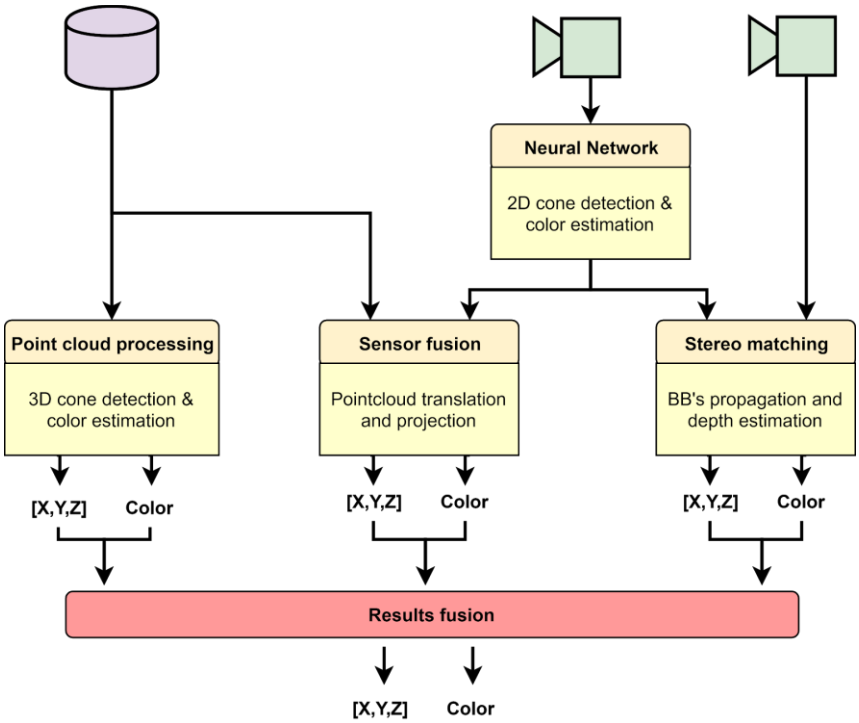

Figure 27: Perception cone detection algorithm system

The algorithms are designed considering a controlled environment. The autonomous car has to be able to compete in a Formula Student race track, delimited by yellow and blue cones, in an open zone with only the ground and the cones.

In this thesis degree, the cone detection methods that imply LiDAR (point cloud processing and sensor fusion) are discussed. The neural network detection has been developed by Roger Aylagas and the stereo matching algorithm by Eloi Bové.

3.3.1. Point cloud processing (LiDAR only)

Point Cloud Library (PCL) is an open source project for 2D and 3D image and point cloud processing, which contains a high number of state-of-the art algorithms [9]. For this project, PCL is used to process the point cloud obtained with the LiDAR due to its numerous filtering, segmentation, and feature estimation available functions.

Figure 28: Point Cloud Library logo

PCL is compatible with ROS, so the implementation of ROS nodes/nodelets with PCL is the chosen option to process the point clouds. In order to obtain the point clouds in ROS, Velodyne LiDAR offers a driver that publishes the raw data of the device in a topic */velodyne_points* at which an algorithm can subscribe and get the data, all implemented

in nodelets for a better performance. Moreover, PCL has some default point cloud types of data, but not compatible to handle all the VLP-32C data fields. For that reason, a new type of data is created with not only the coordinates of the point, but also the intensity and ring (*PointCloudXYZIR*).

The first exposed algorithm is based on four main steps [Figure 29]:

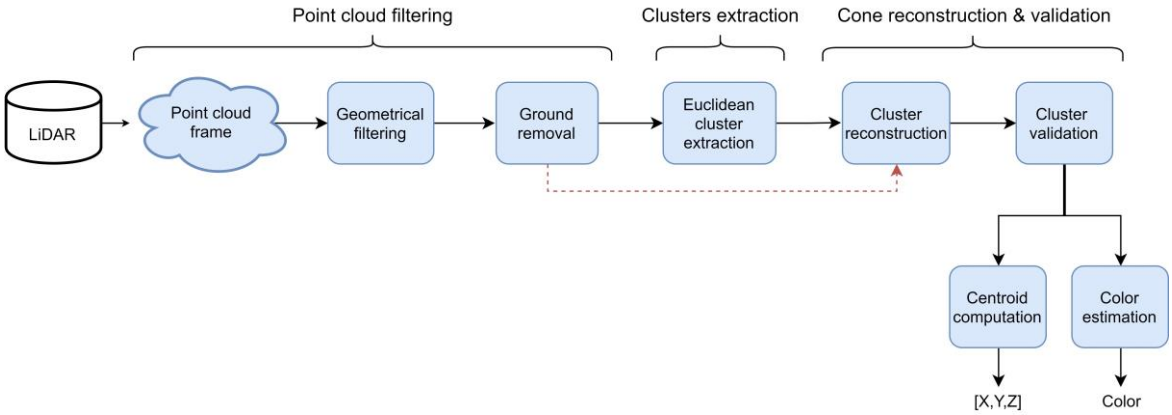

Figure 29: LiDAR only - Algorithm structure

Point cloud filtering

Velodyne VLP-32C produces up to 600.000 points/second, which is a high amount of data to process in real time. With the available FOV in CAT12D (Xaloc), the LiDAR provides around 30.000 points for each frame (an entire revolution), with a range of approximately 200m. Only some of that set of points represent cones, and analyze all the point cloud looking for cones implies high computational load and excessive time. Then, there is a first step of filtering based on two parts:

1. **Geometrical filtering**: two geometrical filters are applied to discard the points located too far away and too close to the LiDAR (passthrough filters) [Figure 30].
    - The points located at more than 20m at front and 5m at sides are irrelevant for the algorithm. At 20m the low LiDAR resolution would prevent a cone to be detected, and cones located more than 5m away of lateral distance from the car are not interesting for the control computation.
    - There are parts of the car that are always reached by the LiDAR laser beams (front wing, monocoque parts…) and are always in the same position. There is no need to look for cones in a square of 1x1m from the LiDAR center.

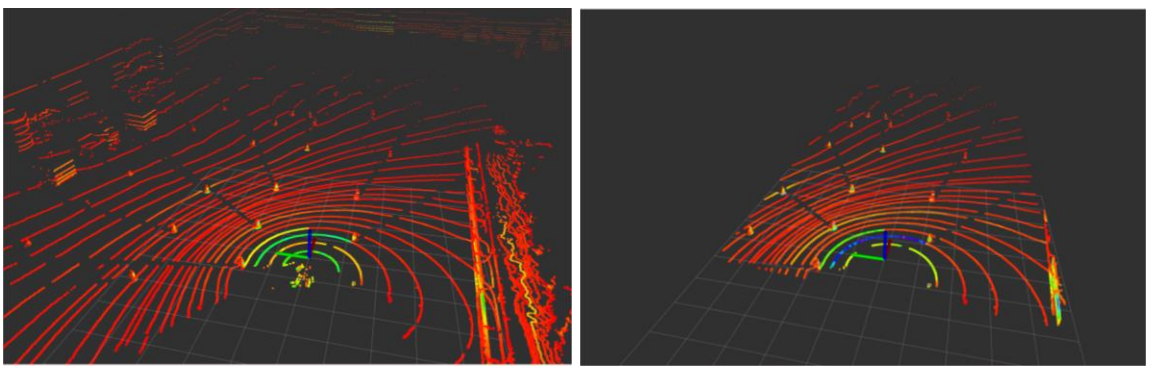

Figure 30: LiDAR only - geometrical filtering

2. **Ground removal**: approximately a third of the LiDAR reflections come from the ground. This part of the algorithm removes all the points representing the ground, using a RANSAC (RANdom Sample Consensus) technique. RANSAC is an iterative method to estimate the mathematical parameters that characterize a geometrical model (plane, cylinder, cone, …) [18].

   Assuming a dataset with a lot of points, some of them representing the ground (*inliers*) and some of them not (*outliers*), the method tries to fit the mathematical model established (a plane) to several random samplings of the data. It consists of two steps: random hypothesis and verification.

   First of all, a minimal random subset of data is selected to compute the chosen model (for a plane, 3 points are used). Then, the entire dataset is analyzed to verify which points are consistent with the model previously established. In order to determine if a point is an *inlier* or an *outlier*, a distance threshold *d* is used to consider the noise and error deviations and decide if the element is a part of the plane or not. The group of inliers selected are the *Consensus Set*.

   The RANSAC algorithm computes different *Consensus Sets* in an iterative way, with different number of *inliers*. The set with more *inliers* is selected as the better representation of the mathematical model in all the dataset [Figure 31]. Finally, all of the *inliers* are removed to extract the ground.

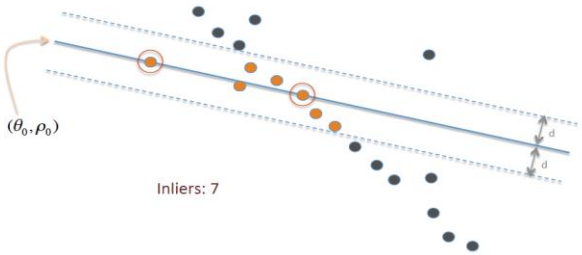

Figure 31: Inliers (orange) and outliers (blue) when fitting a plane [Wikipedia]

Other RANSAC variants can improve its robustness. For example, MSAC and MLESAC use weighted distances and maximum likelihood estimator instead of fixed thresholds to classify *inliers* and *outliers*, respectively; another alternative is to use the consistency between normal vectors jointly with the distance to the model to decide, but assuming the computational cost to obtain the normal vectors [19].

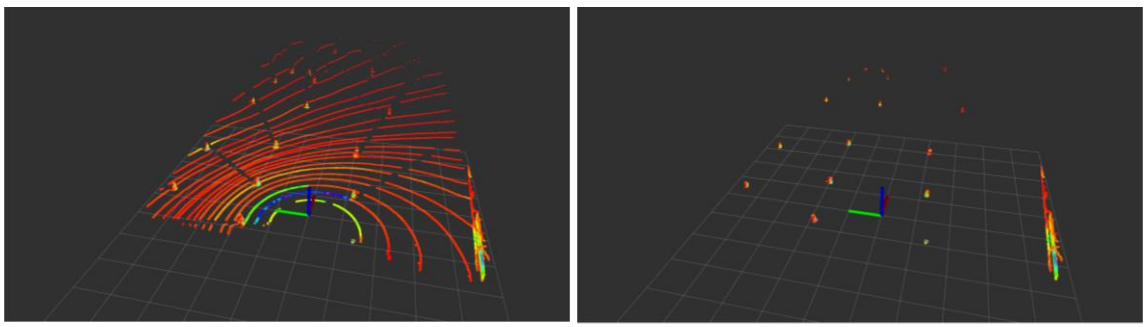

Figure 32: LiDAR only - ground removal

Clusters extraction

With the filtered point cloud, a Euclidean Cluster Extraction is effectuated to determine the possible positions of cones. The idea of the algorithm is to detect possible cone candidates just analyzing the density distribution of points in the space, using the Euclidean distance as a criteria. That strategy uses the assumption that the point cloud has been successfully filtered in the ground removal process.

This step starts with the creation of a k-dimensional tree (in this case, k-3D tree), a data structure based on a binary tree used to organize the point cloud data. Starting from the top of the tree, each level splits all the children elements in two leafs (it “divides the space” into halfs), according to a hyperplane determined by an specific dimension that changes (every level down uses the next dimension to split [X, Y, Z], in a cyclical way) [20].

This data structure is very effective when there is the need to find the nearest neighbors, because the search performed in the tree allows the algorithms to focus the search in specific areas improving the speed.

Then, the algorithm sets up an empty list of clusters that is filled with the point cloud points according to different radius searches. A *tolerance* value is defined to specify the radius of the 3D Euclidean distance: for every point, if a neighbor point is inside the search sphere and it has not been processed yet, it is added to a list that will form the cluster [21]. The cluster *tolerance*, then, is a critical threshold in this case: a small value could produce a cone to be seen as multiple clusters, and a big value could cause two cones to be seen as only one cluster.

Other critical values that are adjusted in the algorithm to restrict the cases are the minimal and maximal number of points that a cluster can have. In order to detect the far cones, the minimum is established in 5 points and the maximum around 150/200 (the near cones produce a high number of points in the LiDAR).

Cone reconstruction and validation

The clusters are identified and stored in a *PointIndices* variable, that indicates the indices which contain the points that belong to the same clusters, in relation to the whole point cloud frame. But all the clusters are not necessarily cones, so a verification process needs to be done.

Some mathematical and geometrical operations could be done to make a decision, such as shape analysis using normal vectors in surfaces, but it would imply a lot of computational time. Since we are working in a controlled environment, we don’t expect different type of obstacles in the track, but only cones: it is more important to detect cones positions quickly than to have the possibility of distinguishing a cone from a tire on the floor, for example.

According to that reason, different simple conditions are applied [Figure 33]:

- The cluster must have its minimum Z coordinate close to the plane. All the cones must be on the ground; if the distance is too high, the cluster is discarded.
- The cluster must have approximately the known cone dimensions (indicated in the FSG Competition Handbook 2019 [22]). If is bigger, it is discarded as candidate.
- The cluster must have more points in its higher part than in its lower part. If a cluster has more points up than down can’t be a cone, so it is also discarded.

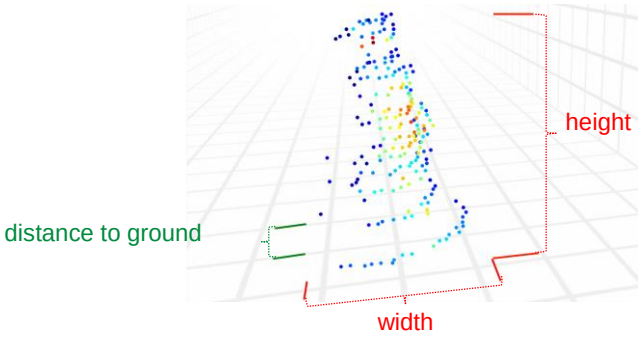

Figure 33: Simple cluster conditions to distinguish a cone

To be able to apply that simple conditions and keep the correct candidates, the cone has to be extracted with all its points. However, the ground removal step deletes the first lower rings of the cones (see first plot in Figure 34), because they are so close to the ground and the lower rings are inliers in the RANSAC plane fitting.

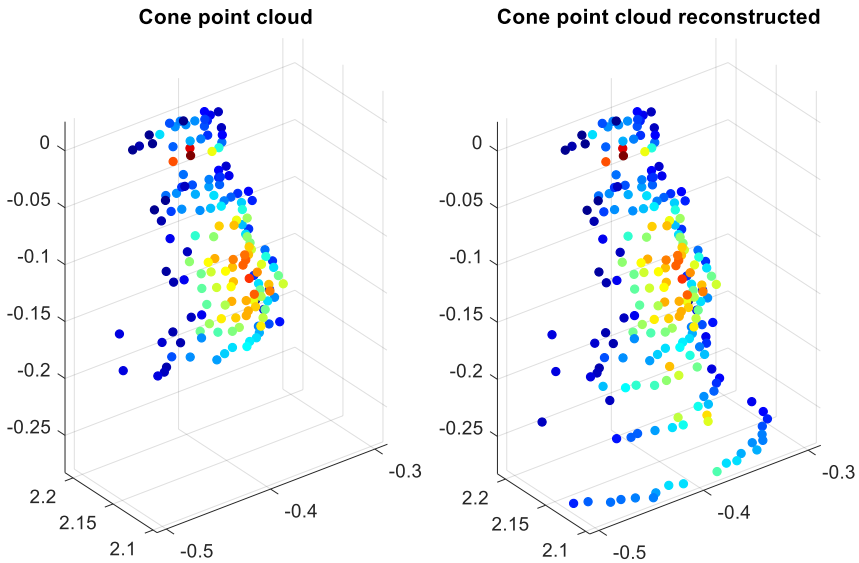

Figure 34: Cone point cloud reconstruction process

The first cone point cloud is obtained from the clustering step, which uses the ground removed point cloud (the cone has lost the lower rings). The second cone point cloud is reconstructed after searching the nearest neighbor's points.

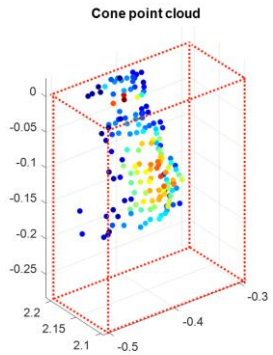

Figure 35: Cone reconstruction

As a solution, a cone recuperation system is proposed: starting from the centroid point of the cluster, a box search of near points that not belong to the cluster is performed to add the points of the plane that were removed. The search, then, is done only in the plane fitted with RANSAC, because there is where the deleted points are. Figure 35 shows the search area when reconstructing the cone, which is defined according to the cone dimensions. Figure 36 shows the cone reconstruction: the initial cluster points are represented using an intensity scale, but the added points from the plane are drawn in white.

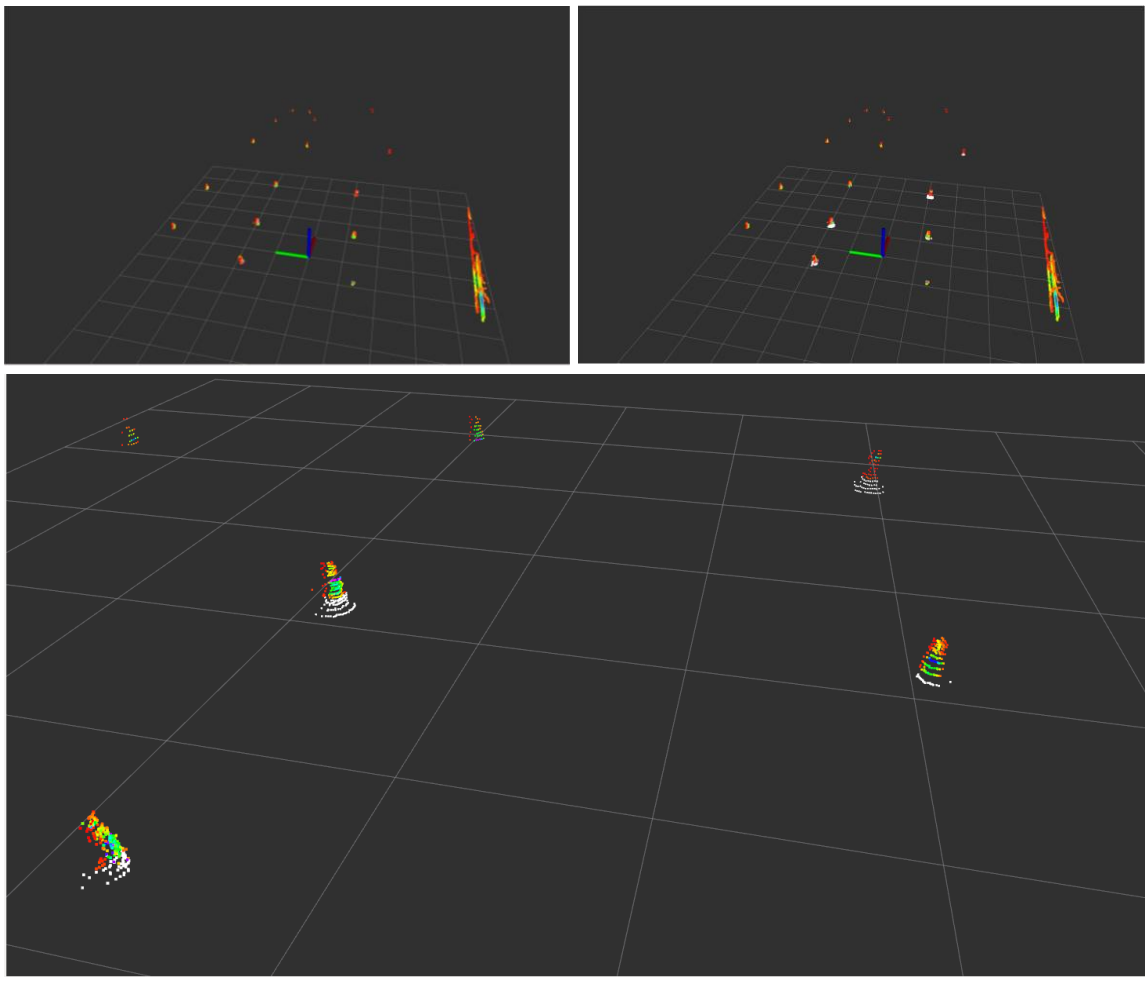

Figure 36: LiDAR only - Cone reconstruction and validation
White points are points that come from the plane point cloud and are near to each cone: they are incorporated to the cluster because they were discarded when removing the ground

Color estimation using intensity

At this point, the algorithm has detected the 3D clusters of the cones and, consequently, their positions from the LiDAR origin. However, the Estimation & Control team needs to determine the track limits, that is, the track borders that affect the possible paths of the car (the system has to know the areas in which the vehicle can be directed and the areas in which not) [Figure 37]. For that requisite, the color of the cones becomes essential, because the yellow and blue cones always delimit the track on the right and left, respectively.

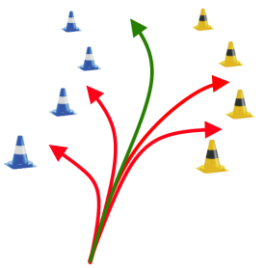

Figure 37: Color importance in path planning

In image processing, color detection would not be a problem since it is information provided directly from the sensor (camera), but with point cloud processing and a LiDAR, there is no color information. Instead, Velodyne VLP-32C offers an intensity field representing the reflectivity of each point. An interesting characteristic of the competition cones is their black and white centered strips, a property that is proposed to be exploited to estimate the cone color.

Figure 38 shows the intensity analysis for a blue cone. The point cloud density is affected by the cone situation (✳) in relation to the car (+) and the reconstruction process (a reconstructed cone has incorporated the lower rings and, then, it has more points). For each ring, the intensity values of all the points are saved onto an array, and different metrics are computed: average intensity in the ring, maximum intensity in the ring and median of the ring intensities. As it can be appreciated, the white surface of the cone produces a major value of intensity in the points than the rest of the cone parts.

Also in Figure 38, the example shows a yellow cone located at approximately 3m: the point cloud has less points, but the intensity values in each ring still reflects the intensity differences. In this case, the black strip of the cone produces less intensity points than the rest of the parts.

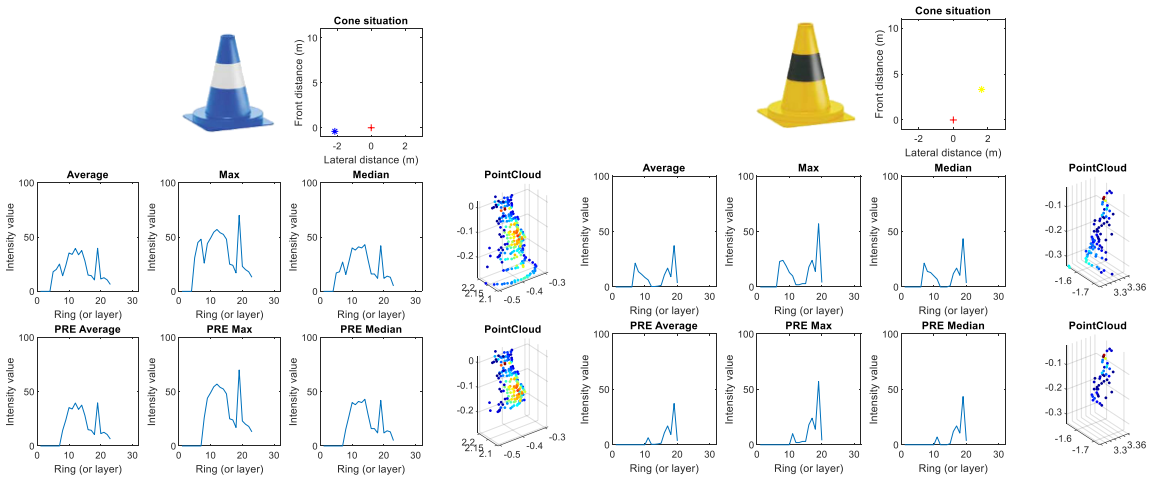

Figure 38: Cone point cloud intensity analysis

The strategy proposed to estimate the color using the intensity relies on some feature differences to distinguish the two cases. The most useful one is the average: if the central stripe produces higher intensity than the other parts, the average has a “∩ shape”, so the cone is blue; if the central stripe produces lower intensity than the other parts, the average produces a “U shape”, so the cone is yellow. That approach is acceptable when working with reconstructed cones, because with the filtered ones there less points and, then a loss of intensity data (see second row plots in Figure 38). A representation of the clusters detected is shown in RVIZ with rectangular markers, with yellow or blue colors according to the decision [Figure 39].

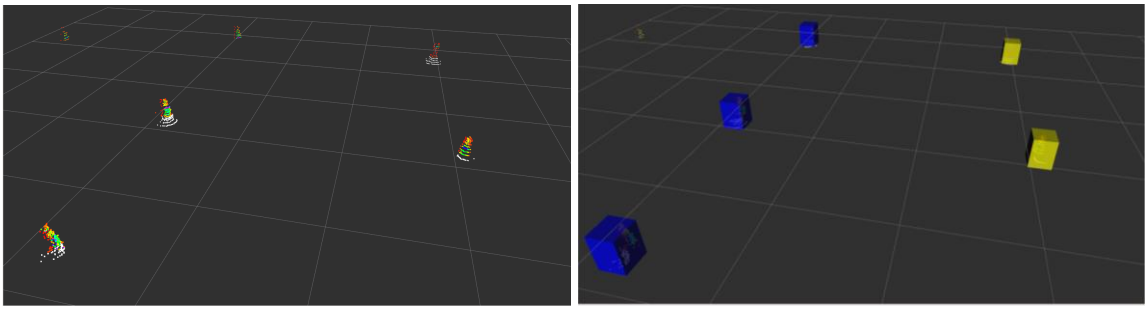

Figure 39: LiDAR only - Color estimation using intensity

In terms of implementation, the decision is taken using a second degree polynomial fitting function [Figure 40]. The average intensity values in function of each ring is the data used to obtain the slope of the curve fitted (positive slope (U) or negative slope (Ո)), and with that information the color is estimated. Despite the cone position detection works very well, the reliability of this color estimation method using only LiDAR is not comparable to the camera one. To improve that problem, the fusion algorithms are explored.

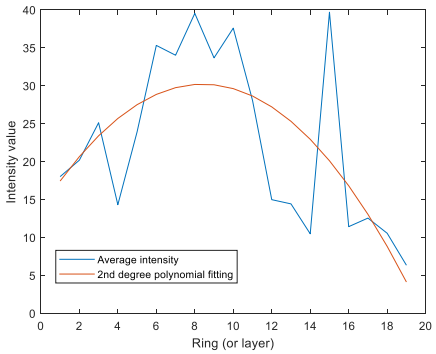

Figure 40: Second degree polynomial fitting for intensity values

Finally, when the cone position (its centroid) and color is decided for all the cones in a frame, the information is published in a ROS topic (*LidarNodelet/cones_color* in Figure 41), and will be used in Estimation & Control.

The ROS graphical representation of the nodelets and the interconnections is shown in Figure 41, which reflects the *pcl_manager* as the nodelet manager of all the steps. The zero-copy transfer process between nodelets characterizes the modular design of the architecture: each step of the algorithm (implemented in a nodelet) is subscribed to a topic and publishes its result, in a “chain structure” way.

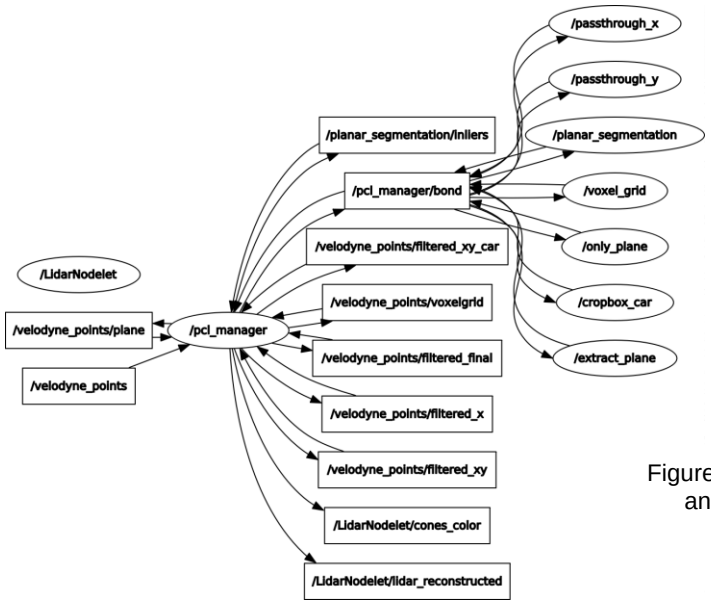

Figure 41: ROS topics (rectangles) and nodes (ovals) graphical structure [rqt_graph]

3.3.2. Sensor fusion (BB2L & L2BB)

The good detection of position but poor color estimation of the LiDAR only algorithm leads to explore the use of the two sensors available in a combined algorithm to improve the final result.

The idea is to merge the data of a 2D sensor (camera) and a 3D sensor (LiDAR) to exploit the good features of the different devices, to detect the cones 3D positions and color. In order to detect the cone color, the implicit information that an RGB camera can give is clearly more reliable than any algorithm that uses intensity. A similar argument justifies that the 3D position of an object is more reliable from a depth laser sensor (LiDAR) than from a camera, even in the case of a stereo camera. From this point (the color information should be extracted from the camera and the depth from the LiDAR), two fusion algorithm can be implemented, according to the detection source:

- Point cloud cone detection (L2BB): the source of the cone detection is the Euclidean Clustering Extraction algorithm explained in 3.3.1, and the color is extracted from the camera image.
- Image cone detection (BB2L): the source of the cone detection is an image detection algorithm (in this case, implemented with Neural Networks) which also gives the color, and the 3D position of the cone (distance) is obtained from the LiDAR point cloud.

Working with two sensors is a complex task, because it implies synchronism mechanisms. ROS is a system dedicated to real-time applications, so synchronism is one of the bases in the messages. In each message, a header has a timestamp information and there are tools which help the developer in the implementation of time coincidence between different sensors. One of this tools, widely used in this project, is *Time Sincronyzer* [23]: it can match messages even if they have different timestamps, with an approximate time policy that links the messages located close in time (in this case, it will link camera images and LiDAR frame point clouds of the same moment). Figure 42 could be a graphical example of the synchronization between LiDAR (first row) and camera (second row) messages, supposing that the first one provides data with less frequency.

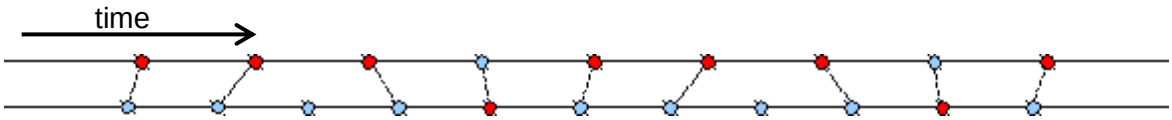

Figure 42: Example of message link between two topics with unequal frequency (ROS documentation)

As it has been explained, when working with fusion is important work with data captured in the same period of time, and ROS timestamps are crucial to synchronize it. According to 3.1.3, cameras and LiDAR are received in different machines (Nvidia and PU), so the system clock must be the same in both systems.

The fusion of 3D and 2D data is proposed to be done by converting the data from the 3D world (LiDAR) to a 2D plane (camera). That process is divided in two main steps [24]: transformation and projection.

Point cloud transformation

The points from the LiDAR are given from its own coordinates origin, that is different from the camera one. As 3.1.4 states, the sensors are located in different parts of the car, so

the data provided is captured from different points of view and positions. To sum up: their reference frame is different.

A first step to be done is to convert the point cloud coordinates from the LiDAR reference frame to the left camera frame (in this case, it is used as the principal). In ROS, that is performed by creating a TF [25], a tool that lets track different coordinate axis over time.

To know where the LiDAR frame is in respect to the left camera frame, in this project we use *velo2cam* calibration package from ROS [10]. That algorithm, used by Eloi Bové in the team, employs a wooden pattern with 4 holes seen by the LiDAR and the camera to produce the 6 necessary parameters for a TF: translation [x,y,z] and orientation [roll, pitch, yaw].

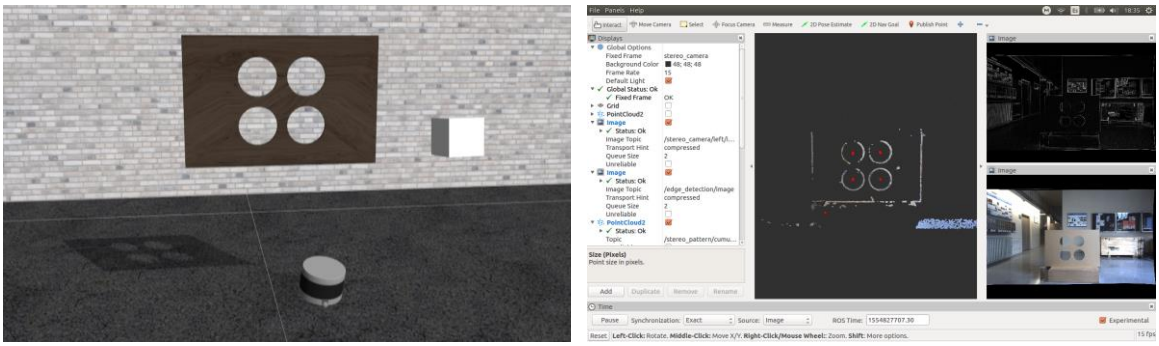

Figure 43: Wooden pattern to calibrate LiDAR-cameras (*velo2cam*) and algorithm working

With the 6 parameters a TF is created and the transformation of the point cloud from *velodyne* to *camera* is performed, so the sensors are registered. It is important to point out that only the filtered point cloud (BB2L) or the cone centroids (L2BB) is the data transformed, to improve the computational cost (there is no need to project all the raw point cloud).

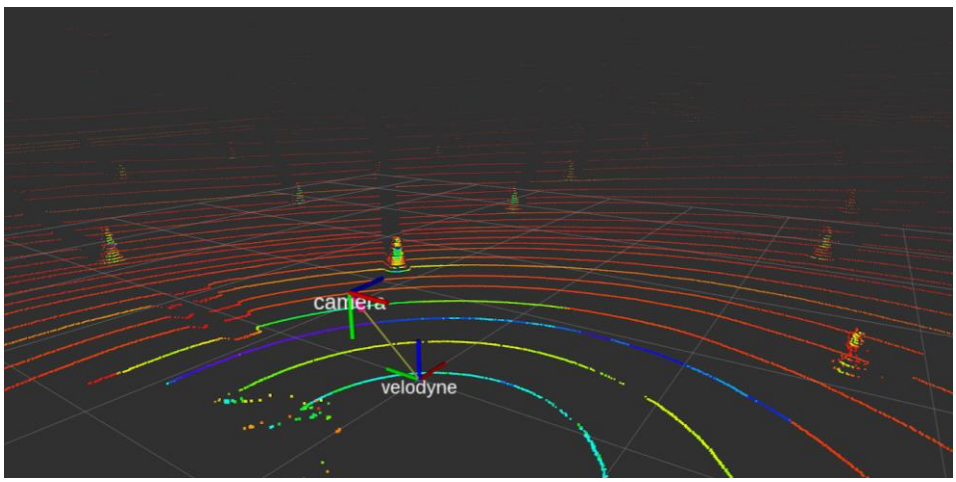

Figure 44: LiDAR point cloud at *velodyne* frame ready to be transformed to *camera* frame

The *Velodyne* axis has Z pointing up, but *camera* axis has Z pointing forward

Point cloud projection

The next step is to project the points, now referenced from the camera frame, to the camera plane. It is done assuming a Pinhole Camera Model with the axis oriented according to *camera*, and with the parameters included in a CameraInfo message from

ROS [26]. The parameters include the projection matrix, the distortion coefficients and the intrinsic camera matrix [27]. To obtain those parameters, we use an extended camera calibration algorithm that works with a chess pattern (Eloi Bové).

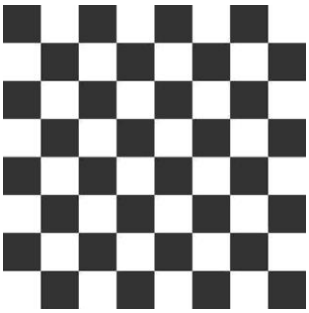

Figure 45: Chess pattern to calibrate cameras

Now, the points from the 3D world are projected into the camera plane and the information between both sensors can be connected. With the sensors information fused, there are two detection sources to use that are classified in two algorithms.

3.3.2.1. L2BB: LiDAR to Bounding Box

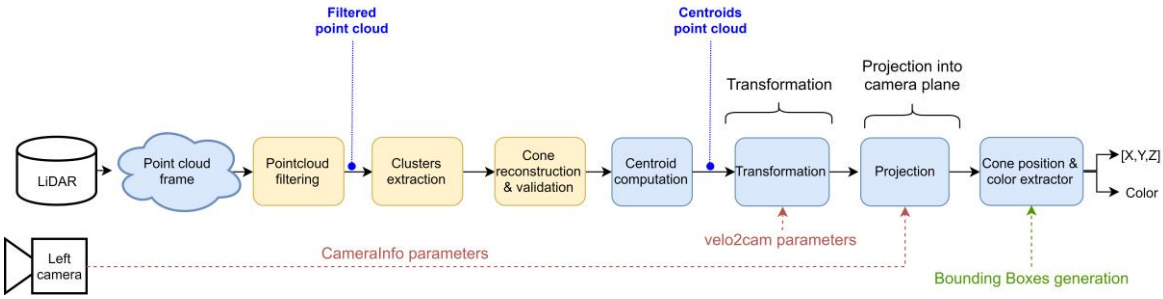

Figure 46: L2BB - Algorithm structure

This first algorithm takes the LiDAR only algorithm as the source for cone detection [Figure 46]. The cones centroids are computed as explained in 3.3.1 section, but the color is not estimated using intensities (only the cones position is computed, see green prisms in Figure 47). The centroids of the cones are transformed and projected into the camera plane, in order to obtain the color from the image. For each centroid drawn on the image, a bounding box is generated according to the distance, and the color of the enclosed region is computed (see points as centroids and rectangles as BB in the top image of Figure 47). Then, the position of the cone is precise because it is obtained with the LiDAR, and also the color because it comes from the image.

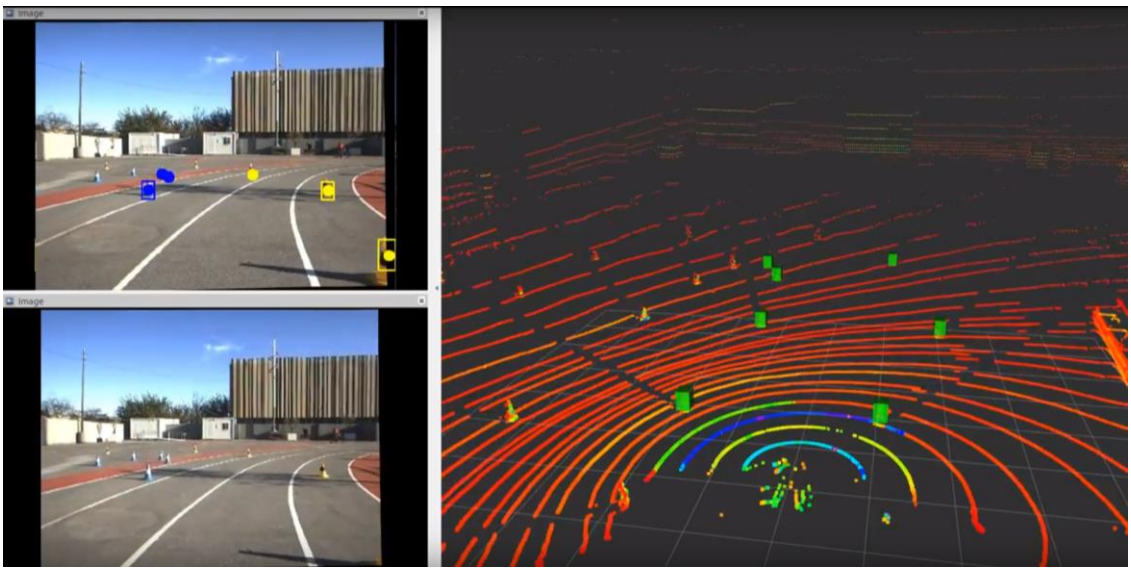

Figure 47: L2BB - Working algorithm with projected centroids (dots) and generated bounding boxes (squares)

3.3.2.2. BB2L: Bounding Box to LiDAR

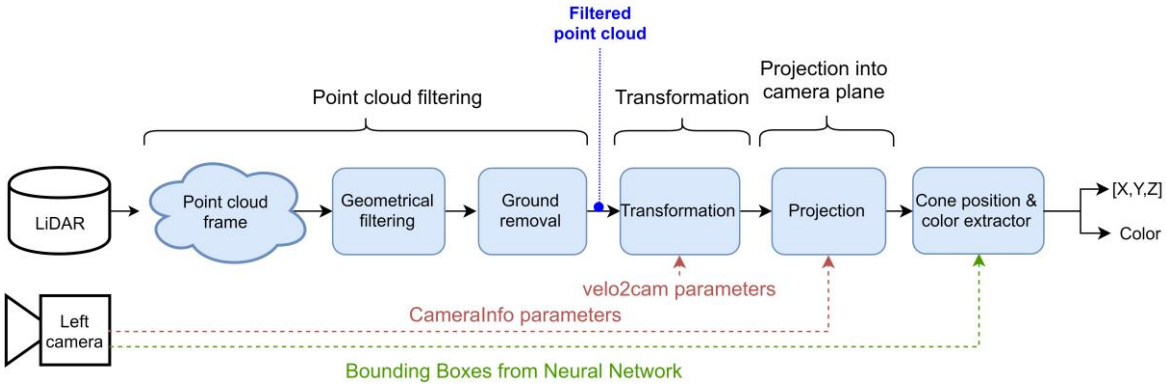

Figure 48: BB2L - Algorithm structure

The second algorithm takes the Neural Network image cone and color detection as the source [Figure 48].

On the one hand, all the filtered point cloud is transformed and projected into the camera plane. A new type of point cloud has been created to deal with that situation: *PointXYZIRnm*. A point from that point cloud stores the usual data (X,Y,Z,I,R) and the projection values at the camera plane (n,m). In that way, when projecting the point cloud into the camera plane, the 3D information of all the points projected is still stored, and is easy to recuperate. On the other hand, the bounding boxes coming from the NN are also located into the camera plane.

With all this data, two 2D point clouds are available to be compared. The points from the filtered data that are enclosed inside a bounding box (this is done using the n and m variables) are considered as part of the cone, and all their 3D data (X,Y,Z) is used to compute a representative centroid (the average).

With the first algorithm, there was the need to have the image to detect the color. In this case, the NN gives the position and color of the cones in the image, so the projection can be done in a white image (there is only need to see which points fall inside the BB's to determine where they are in 3D space).

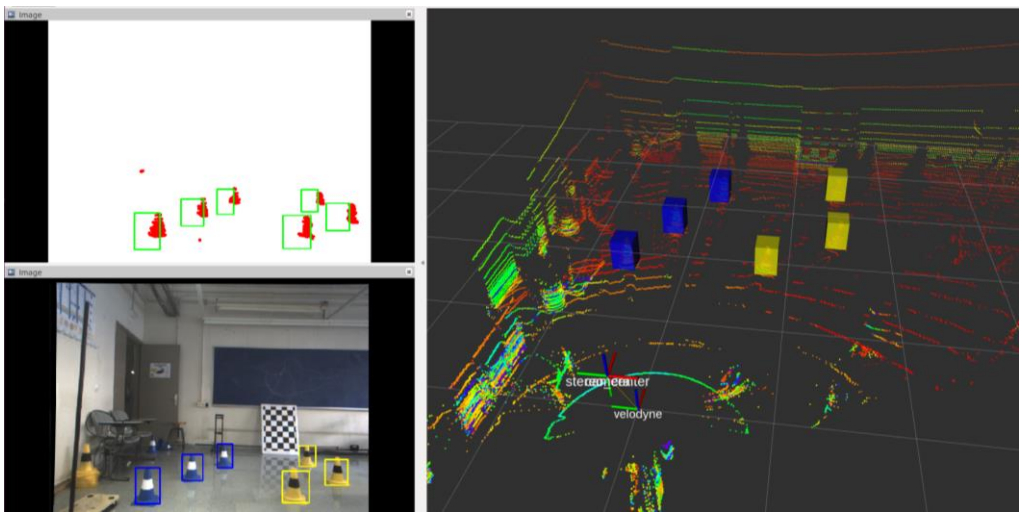

Figure 49: BB2L - Working algorithm with projected filtered point cloud (red) and NN bounding boxes (green)

[in this case, LiDAR and cameras are slightly descalibrated]

3.3.3. Results fusion

When all the algorithms are working, their results might vary significantly. Since the initial cone detection could come from the NN or the Euclidean Clustering Extraction, the final output will be different depending on the situation.

All the output data is standardized in a type of message, so a *Time Sincronyzer* is subscribed to all the results topics and publishes only one decision (which will be used in Estimation & Control). Moreover, the messages are completed with the IMU information of the detection moment, so then the following nodes will have the car position information related to each cone. Finally, the different algorithms produce the cone coordinates from their sensor frame (eg. LiDAR only and fusion from *velodyne* and stereo matching from *camera*), so this node is in charge of transforming all the data into a *base_link* frame (using TF's), where the car IMU is located.

4. Results

The algorithm performance is checked with real sensor data and known cones positions (ground truth), to analyze the variability of the detections and the robustness of the system. In order to do it, a manual car has been "constructed" to locate all the sensors in fixed relative positions (important for fusion between LiDAR and cameras), and it has been equipped with a communications and energy system to be able to move it freely without the need of an electrical power cable (router, switch, 12V battery…), as it can be appreciated in Figure 50 and Figure 51.

Figure 50: Manual car capturing data

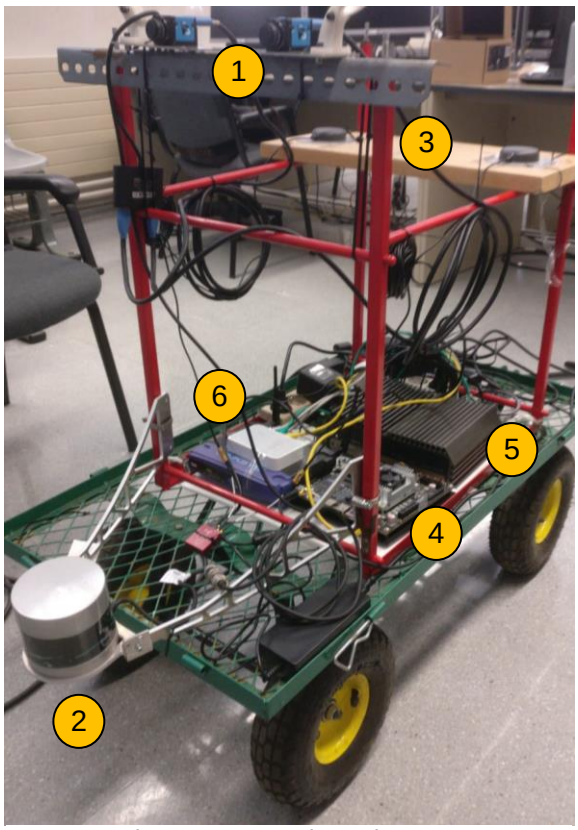

Figure 51: Manual car elements

It has the stereo cameras (1) and the LiDAR (2) in similar positions in relation to the real car, and it also incorporates the IMU+GPS sensor with two antennas (3). Nvidia (4) and Processing Unit (5) are in charge of data capturing or processing in real time, and a router and switch (6) guarantee the communication.

When the sensors arrived, that manual car allowed us to capture the first data with LiDAR and cameras, and test the algorithms in acceptable conditions (with the same sensors that has the real car, located in similar positions and with the possibility to move them all together following a track of cones).

To compare the algorithms, a ground truth set of cones is established at certain positions (a blue and yellow pair of cones at each meter, see Figure 52), and LiDAR only, stereo matching and fusion BB2L algorithms are executed. Regarding the systems related with LiDAR, Figure 53 shows the variability of the detections with BB2L and LiDAR only: the closer cones are successfully detected with both algorithms, but as we move away from the origin, the position variability increases, especially in the BB2L case. The advantage of both algorithms is that the position is extracted from the LiDAR sensor, so the precision is ensured due to the high reliability of the laser distance calculations. However, it is important to point out that the fusion system depends on the Neural Network results and the calibration parameters, so if a bounding box has less projected points inside (because of a BB shift or a calibration mismatch), the centroid computation will be based on less data and the error could increase.

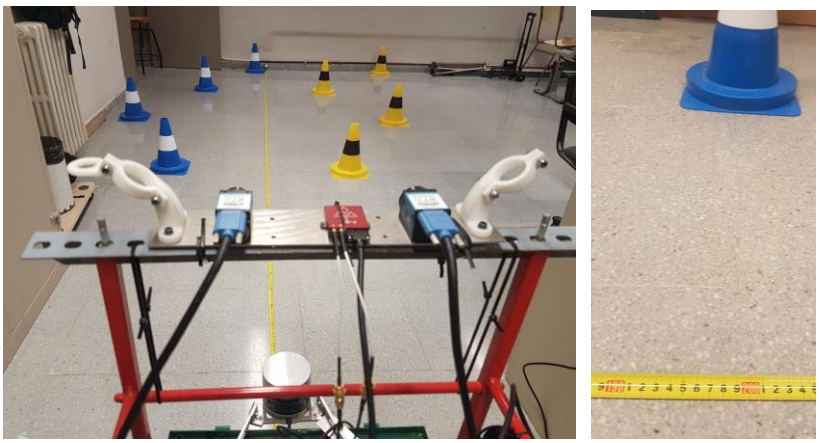

Figure 52: Ground truth of yellow and blue cones, distributed in equispaced rows (1m)

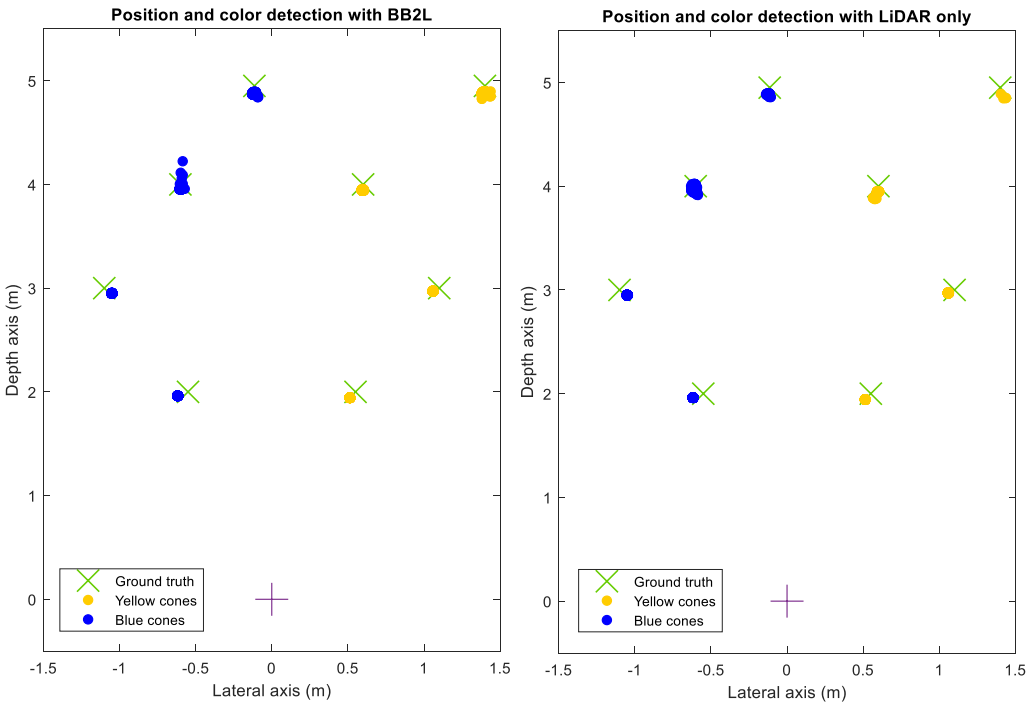

Figure 53: Position and color detection with BB2L and LiDAR only with ground truth data

In Figure 54, the detections with BB2L (left) and LiDAR only (right) are shown in RVIZ, and an interesting result can be highlighted. In the frame captured, the LiDAR algorithm does not detect the farther yellow cone because it has such a low number of points that is not considered a candidate cluster, but it did appeared in Figure 53. The algorithm detects the yellow cone intermittently, but when it is detected, the position is correct. Figure 55 shows the three algorithms working at the same time.

Finally, Figure 56 shows the integration between team sections. Perception is detecting cones and Estimation & Control is computing the track boundaries as well as a trajectory to follow, which will be sent via CAN to the system that moves the car.

Some pictures of the final car result are attached in appendix 8.1.

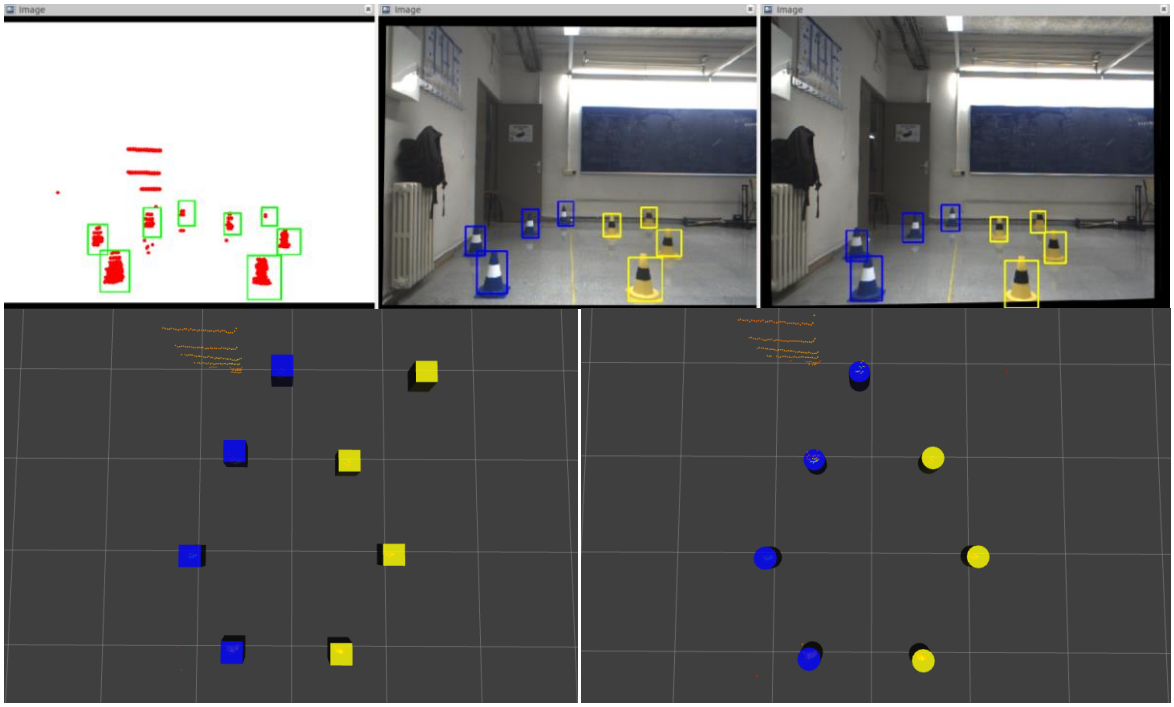

Figure 54: Position and color detection with BB2L and LiDAR only with RVIZ

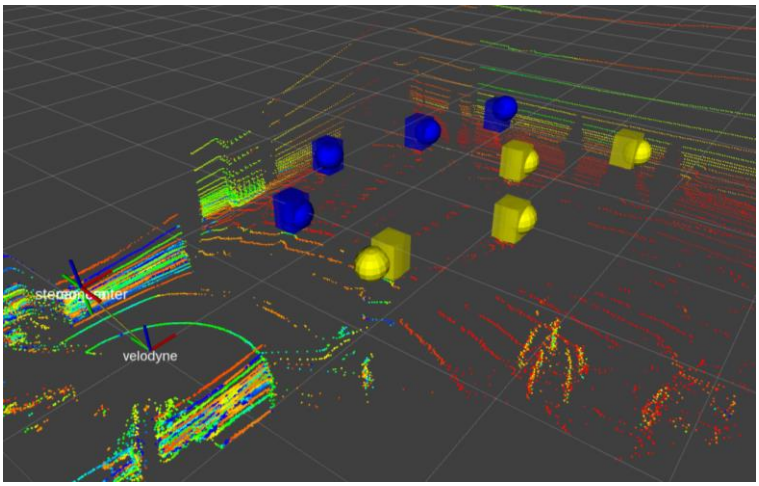

Figure 55: Position and color detection with BB2L, LiDAR only and stereo with RVIZ and the raw point cloud

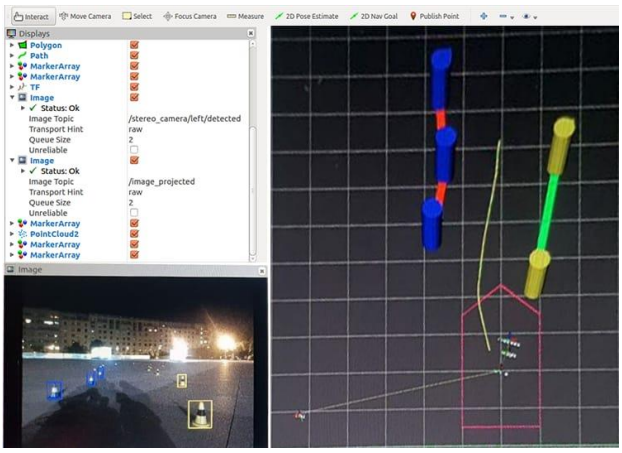

Figure 56: Perception and control integration

5. Budget

In the project, the main software used is free (ROS, PCL, OpenCV…) despite Matlab Calibration Toolbox for the cameras calibration.

The engineering students have dedicated 25h during 22 weeks, which is a total of 525 hours. The degree thesis tutor has collaborated with 2h meetings during 22 weeks, which is a total of 44 hours. The wages are incorporated as 10€/h and 30€/h, respectively.

The budget is related to all the necessary software and hardware for the Perception section.

Concept	Q.	Cost	Total cost	Sponsor rate	Final Cost
Velodyne VLP-32C	1	13.000,00 €	13.000,00 €	100%	0,00 €
The Imaging Source DFK DFK33UX252 cameras + lens + USB3.0 cable	2	1.200,00 €	2.400,00 €	100%	0,00 €
Cincoze DX-1000	1	5.000,00 €	5.000,00 €	33%	3.350,00 €
Nvidia Jetson TX-2	1	350,00 €	350,00 €	33%	234,50 €
D-Link Ethernet switch	1	46,31 €	46,31 €	100%	0,00 €
LiDAR support (aluminum, laser cutting, 3D printing...)	1	140,00 €	140,00 €	100%	0,00 €
Cameras support (3D printing, attachments...)	1	194,00 €	194,00 €	100%	0,00 €
Matlab student license	1	250,00 €	250,00 €	100%	0,00 €
Junior engineer (550h x 10€/h)	550	10,00 €	5.500,00 €		5.500,00 €
Degree Thesis tutor (44h x 30€/h)	44	30,00 €	1.320,00 €		1.320,00 €
TOTAL			28.200,31 €		10.404,50 €

Table 7: Perception section budget

6. Conclusions and future development:

6.1. Conclusions

In this Thesis Degree, the Perception approach to the FSD 2019 competition of Driverless UPC team is presented, detailing the explanations of the LiDAR cone detection algorithms. The design, construction and implementation of a Formula Student car is a project with a huge amount of work, and where a lot of people is involved to fulfill the final goal: compete in Formula Student. Several hard competition deadlines mark the season timings and the whole team has to collaborate when the problems appear and the car is not ready.

For that reason, we wanted to design a robust system for Perception, able to continuously send cones positions and colors regardless the possible situations and problems of the car (vibrations, movements or weather conditions). It can be concluded that this objective is completely fulfilled, because there are three working systems (LiDAR only, fusion and stereo matching) integrated in the correspondent car hardware (PU and Nvidia) that work in real time with ROS. What's more, the results from the different sources cover the minimum of four cones detected stated in the initial specifications.

Real time computation has been a big challenge in this Thesis Degree when designing the algorithms. For sure, a better classification method could be designed with LiDAR only information, probably using 3D features to search and verify cones, but it would certainly not work in real time (is has been experienced). One of the bigger conclusions extracted from this project is that the need of immediacy is an important restriction when developing: if a solution detects perfectly the cones but it does not compute them in real time, it is not a valid solution (the cones would be used in the system too late, when the car will have already crashed). In order to improve the real time computation, the use of ROS nodelets instead of nodes has been a revelation for the image and point cloud transfer rates, improving the speeds considerably.

In terms of the sensors used, there is another observed feature that is important to be mentioned. LiDAR sensors are the most precise option, in this case, to detect the distance to objects because is a built-in field offered by this devices; camera sensors are the better and easier solution to detect objects and its color, using advanced image processing techniques that are more developed than the 3D world, still in a growing period. For that reason, fusion of sensors with BB2L is the option that better results has provided to the team.

Regarding the algorithms results, it can be concluded that both produce useful detections. LiDAR only provides very precise positions, but it is affected by two main problems: the color estimation is not stable and the cones located too far from the device are not detected (they have a low number of points). That last problem is solved with BB2L, because only a point contained inside the BB detection can be used to determine the cone position. However, despite the NN detections and colors are quite good, this proposal suffers from position variability, since the method to estimate the distance at which the cone is located works with the average centroid of all the points that remain in the BB. For example, if the LiDAR perspective produces points from different cones projected into the same BB in the camera plane (that can happen since the LiDAR is located close to the ground with respect to the cameras), the centroid determined of that

BB cone will be affected, because the points coming from other cones will also count in the average computation.

As a final personal conclusion, I enjoyed this project much more than I would have ever expected, and I learned a lot in fields such as 3D point cloud processing and ROS. What's more, this project has allowed me to work with an expensive and special sensor (Velodyne VLP-32C), as well as with a group of fantastic teammates.

6.2. Future development

There are always future development tasks that could be done. Considering the high precision of position in the LiDAR only algorithm, the color estimation in that solution could be improved. A possible way could be to implement an image classifier (a Neural Network, for example) for cones detected with LiDAR, using an intensity projected gray scale image to determine the cones colors. Figure 57 is an example of intensity projection images of cones in a preliminary implementation not finished due to time restrictions.

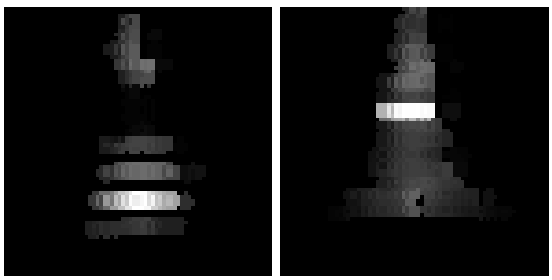

Figure 57: Cone intensity projection image (yellow and blue cone)

The projection is done generating a different plane for each cone (the plane uses the centroid vector of the cone from the LiDAR origin as the normal vector, to ensure that it is parallel to the LiDAR laser beam and all the intensity information is used)

Finally, for BB2L there could be a lot of future development work. The improvement of the calibration results or the distance computation could be some examples. In this last case, a better analysis to discard the points projected from different cones into the same BB's would be a useful enhancement.

7. Bibliography

[1] Formula Student Germany Competition. Formula Student international competitions. [Online]. https://www.formulastudent.de/world/competitions/

[2] Formula Student Germany Competition. FSG: Concept. [Online]. https://www.formulastudent.de/about/concept/

[3] Formula Student Germany Competition. (2018, October) Formula Student Rules 2019. [Online]. https://www.formulastudent.de/fileadmin/user_upload/all/2019/rules/FS-Rules_2019_V1.1.pdf

[4] Formula Student Germany Competition. (2018) FSG Magazine 2018. [Online]. https://www.formulastudent.de/fileadmin/user_upload/all/2018/PR_Media/FSG2018_magazine_v20180725_LQ.pdf

[5] Formula Student Germany Competition. FSG: Disciplines. [Online]. https://www.formulastudent.de/about/disciplines/

[6] Akademischer Motorsportverein Zürich (AMZ) students team. [Online]. http://driverless.amzracing.ch/en/team/2019

[7] OpenCV website. [Online]. https://opencv.org/

[8] Tensorflow website. [Online]. https://www.tensorflow.org/

[9] Steve Cousins and Radu Bogdan Rusu, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 05/2011, pp. 1-4.

[10] C., Beltrán, J., Martín, D. and García, F Guindel, "Automatic Extrinsic Calibration for Lidar-Stereo Vehicle Sensor Setups," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 674–679.

[11] Munich Motorsport. Mono Camera in FSD - FSG Academy 2018.

[12] Juraj, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler Kabzan, "AMZ Driverless: The Full Autonomous Racing System," in *Journal of Field Robotics*, 2019.

[13] Point Cloud Library (PCL). Documentation - Basic structures. [Online]. http://pointclouds.org/documentation/tutorials/basic_structures.php

[14] Velodyne LiDAR. VLP-32C Datasheet. [Online]. https://hypertech.co.il/wp-

content/uploads/2016/05/ULTRA-Puck_VLP-32C_Datasheet.pdf

[15] Velodyne LiDAR. VLP-32C User Manual. [Online]. https://icave2.cse.buffalo.edu/resources/sensor-modeling/VLP32CManual.pdf

[16] Morgan, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler and Andrew Ng. Quigley, "ROS: An Open-Source Robot Operating System,".

[17] Robot Operating System (ROS). Nodelet. [Online]. http://wiki.ros.org/nodelet

[18] Wikipedia. Random Sample Consensus. [Online]. https://en.wikipedia.org/wiki/Random_sample_consensus

[19] Point Cloud Librari (PCL). Segmentation. [Online]. https://pdfs.semanticscholar.org/48e3/a94bf37d571a7e4d314e03ed6eaffd16509f.pdf

[20] Point Cloud Library (PCL). How to use a KdTree to search. [Online]. http://pointclouds.org/documentation/tutorials/kdtree_search.php

[21] Point Cloud Library (PCL). Euclidean Cluster Extraction. [Online]. http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php

[22] Formula Student Germany Competition. (2019) FSG Competition Handbook 2019. [Online]. https://www.formulastudent.de/fileadmin/user_upload/all/2019/rules/FSG19_Competition_Handbook_v1.1.pdf

[23] Robot Operating System (ROS). Message_filters (time synchronizer). [Online]. http://wiki.ros.org/message_filters

[24] Helio Perroni Filho. Projectin Points with ROS Pinhole Camera Model. [Online]. http://xperroni.me/projecting-points-with-ros-pinhole-camera-model.html

[25] Robot Operating System (ROS). TF (Transformations in ROS). [Online]. http://wiki.ros.org/tf

[26] Robot Operating System (ROS). CameraInfo: coordinate system, diagram, projection and rectification. [Online]. http://wiki.ros.org/image_pipeline/CameraInfo

[27] Robot Operating System (ROS). Documentation - sensor_msgs/CameraInfo Message. [Online]. http://docs.ros.org/melodic/api/sensor_msgs/html/msg/CameraInfo.html

8. Appendices

8.1. Final car result

The final car with all the sensors mounted and the computational system installed, as well as other project interesting moments, are shown in the following pictures:

Figure 58: Calibration LiDAR-cameras in different setups

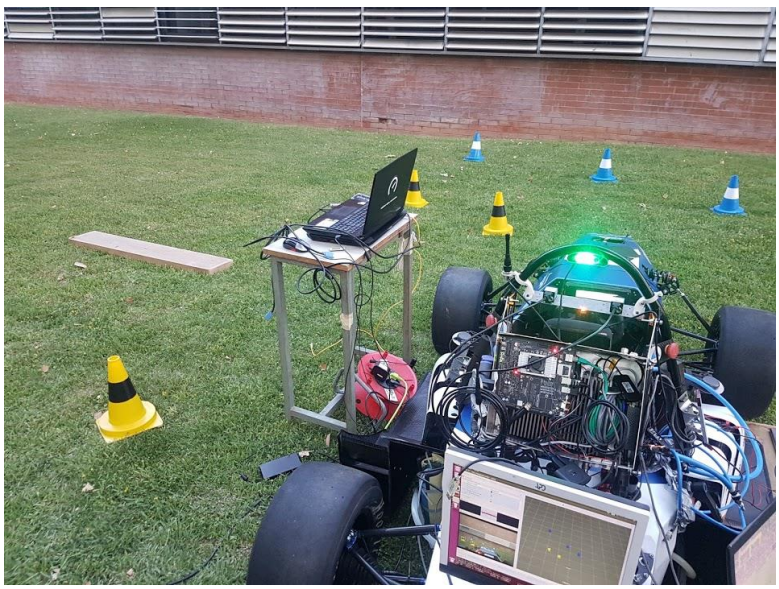

Figure 59: BB2L fusion algorithm running in real time after calibration

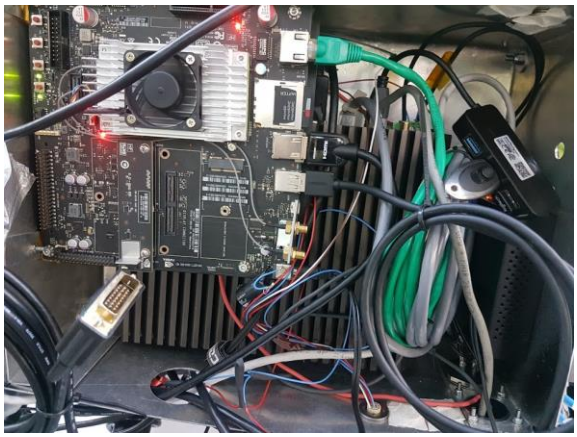

Figure 61: Processing Unit and Nvidia box

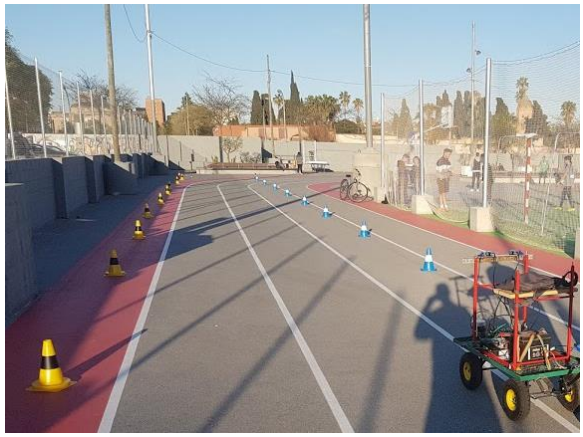

Figure 60: Capturing data with manual car

Figure 62: Velodyne VLP-32C LiDAR

Figure 63: Car monocoque and external elements

Figure 64: Final car design in official Roll-Out 2019

Glossary

BB	Bounding Box
BB2L	Bounding Box to LiDAR (fusion algorithm)
CAN	Controller Area Network
CMOS	Complementary Metall Oxide Semiconductor
CPU	Central processing unit
ETSEIB	Escola Tècnica Superior d'Enginyeria Industrial de Barcelona
ETSETB	Escola Tècnica Superior d'Enginyeria de Telecomunicacions de Barcelona
FOV	Field of View
FSD	Formula Student Driverless
FSG	Formula Student Germany
GPS	Global Positioning System
GPU	Graphic Processing Unit
IMU	Inertial measurement unit
L2BB	LiDAR to Bounding Box (fusion algorithm)
LiDAR	Laser Imaging Detection and Ranging
MLESAC	Maximum Likelihood Estimation SAmple Consensus
MSAC	M-estimator SAmple and Consensus
PAE	Projecte Bàsic d'Enginyeria
PCB	Printed Circuit Board
PCL	Point Cloud Library
PU	Processing Unit (referred to the central computer in the vehicle)
RANSAC	RANdom SAmple Consensus
ROS	Robot Operating System
RPM	Revolutions per minute
SLAM	Simultaneous Localization And Mapping
TCP/IP	Transmission Control Protocol / Internet Protocol
TOF	Time Of Flight
UPC	Universitat Politècnica de Catalunya