Bachelor thesis

Detection of Traffic Cones from Lidar Point Clouds

Daniel Štorc

Supervisor: Ing. Jan Čech, Ph.D.

Faculty of Electric Engineering

Department of Cybernetics

May 20, 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Štorc Daniel** Personal ID number: **483427**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Detection of Traffic Cones from Lidar Point Clouds

Bachelor’s thesis title in Czech:

Detekce dopravních kuželů z lidarových dat

Guidelines:

Traffic cones are used to delineate race tracks in competition of the autonomous student formula. The race vehicle is equipped with cameras and a lidar besides other sensors.
1. Propose a detector of traffic cones from lidar point clouds.
2. Implement and evaluate the accuracy and computational time of the detector using real data acquired by the formula lidar.
3. Optionally, for the sake of redundancy of the perception system, attempt to recognize cone colors from lidar only.

Bibliography / sources:

[1] N. Gosala et al. Redundant Perception and State Estimation for Reliable Autonomous Racing. In arXiv:1809.10099v1, 2018
[2] G. Zamanakos, L. Tsochatzidis, A. Amanatiadis, I. Pratikakis. A comprehensive survey of LIDAR-based 3D object detection methods with deep learning for autonomous driving. Computers & Graphics Volume 99, 2021.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Čech, Ph.D. Visual Recognition Group FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **27.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jan Čech, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, ...

Acknowledgment

I would like to thank to my thesis supervisor Ing. Jan Čech PhD.. His guidance and the advice he provided me were crucial in making of this thesis. Without this guidance, the thesis would not be as good as it is. I would also like to thank him for all the work he has done as the eForce Driverless faculty supervisor, in the past three years. His advice was always helpful and clarified the problems we have faced.

I also would like to thank all those who participated in the eForce team. The current and past members have created something that has allowed me to spend time doing something I love. I would like to express my thanks to these people, because each one of them was crucial to my thesis and the team.

- Ing. Ondřej Šereda, for building the DV.01 almost alone and giving me valuable advice on all aspects of the car.
- Michal Horáček, for the support and helping me stay sane during the time I was leading eForce Driverless.
- Ing. Tomáš Roun, for all information he provided about computer vision and software development. His work helped the team get where it is today.
- Bc. Josef Med for the advice and time he spent making my life easier, when he was captain of eForce FEE Prague Formula.
- Vojtěch Michal, for being able to give advice on anything from software development to control theory. Without his help, I would not have had as much time as I had for my work on eForce.

Finally, I would like to thank my parents, brothers and sister for their unconditional support that they provided me during my studies and their understanding for the time I spent away from home.

Annotation

The bachelor thesis focusses on the development of the traffic cone detector from the LiDAR pointcloud for the autonomous formula. In the thesis, the rules of competition that influence the detector are discussed. Next, the principle of the LiDAR is described together with the attributes that influence the design of the detector. The method to find the optimal configuration of the detector parameters is shown, and the impact of the change of individual parameters is discussed.

Keywords: LiDAR, Traffic cones, Detector, Autonomous formula, Formula Student

Anotace

Bakalářksá práce se zaměřuje na návrh detektoru dopravních kuželek z LiDARových dat pro autnomní formuli, která se účastní soutěže *Formula Student*. V práci je rozebírán vliv pravidel soutěže na návrh detektoru. Dále je popsán princip LiDARu a jeho vlastnosti ovlivňující detektor. Následně je popsán postup nalezení správné konfigurace parametrů detektoru a vyhodnocení vlivu změn jednotlivých parametrů.

Klíčové slova: LiDAR, Dopravní kužely, Detektor, Autonomní formule, Formula Student

Contents

Contents I
List of Figures II
List of Algorithms IV
1 Introduction 1
1.1 Formula Student 1
1.2 eForce FEE Prague Formula 2
1.2.1 DV.01 3
1.3 Motivation 5
2 State of the art 7
2.1 LiDAR detectors 7
3 Proposed solution 8
3.1 Known facts - rules 8
3.2 Known facts - LiDAR 9
3.2.1 LiDAR principle 9
3.2.2 Ouster OS1-64 10
3.2.3 Transformations 11
3.2.4 Maximum useful distance 11
3.2.5 Points distribution 13
3.3 Proposed algorithm 15
3.4 Filtering valid points 15
3.5 Detecting ground plane 16
3.6 Selecting candidates 17
3.7 Detection 18
3.8 Time optimisation 20
4 Implementation 23
4.1 Python and C++ 23
4.2 Finding parameters 23
4.2.1 Scoring 23
4.3 Grid search 24
5 Evaluation 26
5.1 Datasets 27
5.1.1 Training dataset 27
5.1.2 Test dataset 28
5.2 Results 28
6 Conclusion 32
A Attachments 35

List of Figures

1 Season of Formula Student teams, taken from [1]. 1
2 DV.01 and FSE.X [2] 3
3 Sensors placement on DV.01 [3] 4
4 Image of the track for the trackdrive event at FSG 21 in Hockenheim. The track is marked with blue cones at the left side of the track and yellow cones at the right side of the track. Image of [3]. 5
5 Measurement error of mono camera against LiDAR. The graph shows that the error of measurement is is growing with distance. Also the error is not stable, it varies detection by detection. Taken from [4]. . 6
6 Example of bad light conditions which are difficult for the camera to process. LiDAR 6
7 Sample of the input grid maps. Left: the height map. Right: the density map. Taken from [5]. 7
8 Example of cones at FSG 2022 [6] 8
9 Internal mechanism of the rotating LiDAR, adapted from [7]. 9
10 LiDAR used on DV.01 - Ouster OS1-64 located on the front wing of the formula. 10
11 View of modeled situation 11
12 Simulation of rays hit on cone in regard to the distance. 13
13 Percentage of valid point at individual pointclouds from the LiDAR. All of this data was obtained at testing with DV.01 on airfield near Milovice in March 2022 14
14 Distribution of points along the Z-axis of LiDAR coordinate systems 14
15 Illustration of removing the points that are in *AIR_EXCLUDE* distance from the air point. 18
16 Points returned by selection candidates, where we can see the lines returned with no difference in their z coordinate 19
17 Time analysis of the 20 runs of the detector over 215 pointclouds averaged. The experiment was done at computer with i5 2.4 GHz, 8 GB and NVME SSD. 21
18 Example situation of points at the border between bins. The green square represents current bin, if the neighbours weren't included no ground point would be added and the chance of detecting would decrease. 22
19 Image of LiDAR data pointcloud 26
20 Hard to label cones, first cones are clearly cones without any other information, but the ones in the back may be hard to say without looking at the lines which forms the track 27
21 **Air Limit** parameter precision curve around working point. Red square represents selected value. 28
22 **Ground Limit** parameter precision curve around working point. Red square represents selected value. 29
23 **Air exclude** parameter precision curve around working point. Red square represents selected value. 29
24 **Ground include** parameter precision curve around working point. Red square represents selected value. 30

25 **Grouping distance** parameter precision curve around working point. Red square represents selected value. 30
26 **Empty distance** parameter precision curve around working point. Red square represents selected value. 31

List of Algorithms

1 High-level overview of the detection algorithm 15
2 Filtering of valid points 15
3 RANSAC algorithm 16
4 Selection of candidates - first part 17
5 Selection of candidates - second part 18
6 Detection - Clustering 19
7 Detection of cones 20

Chapter 1
Introduction

This thesis describes the development of a traffic cone detector from the raw LiDAR pointcloud, with a focus on the precision and speed of detecting the cones. With the usage of this detector on the racing car in Formula Student competition. The detector will be used to find the limits of the track because of its wide field of view and accurate measurements.

Section 1.1
Formula Student

The Formula Student competition was founded in 1980 by SAE (Society of Automotive Engineers), with the purpose of giving students of technical universities practical experience of designing, building cars, and understanding the economy behind manufacturing working racing cars. In 2005 the first Formula Student Germany (FSG) competition was organised, and this brought great popularity in the mainland Europe. FSG has since become the most important competition for European teams, not only because it is the biggest competition in the world, but also because FSG publishes rules, which are the basis for every other competition in Europe.

The season of Formula Student teams is typically divided into three main parts. Design phase, where teams look for new ways to make their car faster. The second phase, manufacturing, consists of building a new car according to the designs made earlier in the season. And finally, there is competition for Europe teams in summer, where teams compete against each other [1].

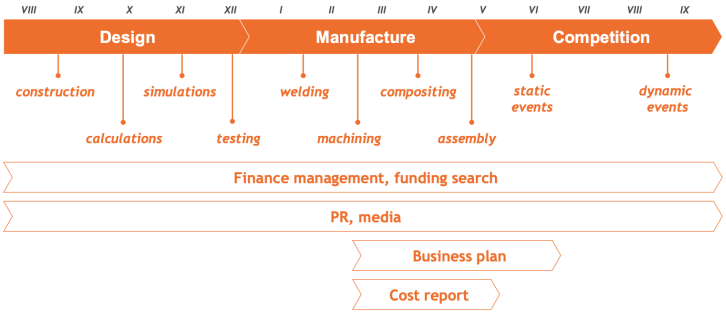


Fig. 1: Season of Formula Student teams, taken from [1].

Formula Student competition is traditionally divided into two major categories, combustion vehicles and electric vehicles. In 2017 the new autonomous category was introduced, which could be a combustion vehicle or an electric vehicle. The event itself is divided into two parts. Dynamic and static disciplines. In static disciplines, students must present their understanding of the development and economy of building a racing car. In dynamic disciplines, cars compete against time in four separate subevents.

The four subevents of the dynamics of the autonomous category are acceleration, 75 m sprint, which test the longitudinal acceleration of the car. The second is the skid pad, an eight-shaped track to show the lateral stability of the formula. The

last two events are autocross and trackdrive, in these disciplines the car must drive through one (autocross) or ten laps (trackdrive) on an unknown track. These events test overall capability of car, and in case of trackdrive even it's abilities to map the track and optimise the path of itself.

In year 2020 Formula Student Germany (FSG) announced that from year 2021 this competition in Europe will have only one class, where every vehicle taking the competition must be able to drive autonomously. This was later delayed to 2022, due to the outbreak of the COVID-19 pandemic in 2020.

Section 1.2

eForce FEE Prague Formula

The eForce FEE Prague Formula Team was founded in 2010 and is the first Czech team that participated in the Formula Student competition in the electric category. This team is managed by the Faculty of Electrical Engineering at the CTU in Prague.

During its first years, the eForce team has managed to achieve several great results. Most notable are successful competitions in 2015 in North America, where FSE.04x has managed to win all competitions it had entered, the first overall electric winner at FS Czech with FSE.07 (2018).

With the new trends in the automotive industry towards autonomous mobility and the introduction of a new category in the FSG competition, two senior members of eForce decided to start a new team *eForce Driverless* This happened at the beginning of 2019, shortly after that, with the help of *Ing. Jan Čech Ph.D.* who has joined as the faculty advisor of the newly founded team, the development of DV.01 started. DV.01 was based on the chassis of *FSE.07*, with new electronics and new actuators that made autonomous driving possible.

The first season of eForce Driverless was heavily affected by the COVID-19 pandemic,due to the situation, all normal competitions that year were cancelled. Instead, a new competition *FS Online* replacement was announced for *FS East* and in this competition, eForce Driverless managed to beat well-established teams such as *TU Delft* with the support of *MIT*.

In the second season, the team qualified for three competitions, FS Czech, FS Spain, and FS Germany. In each of these competitions, the team did well, and at FS Czech ended at third place. FS Spain was sixth, and at FS Germany was also sixth among the best in the world.

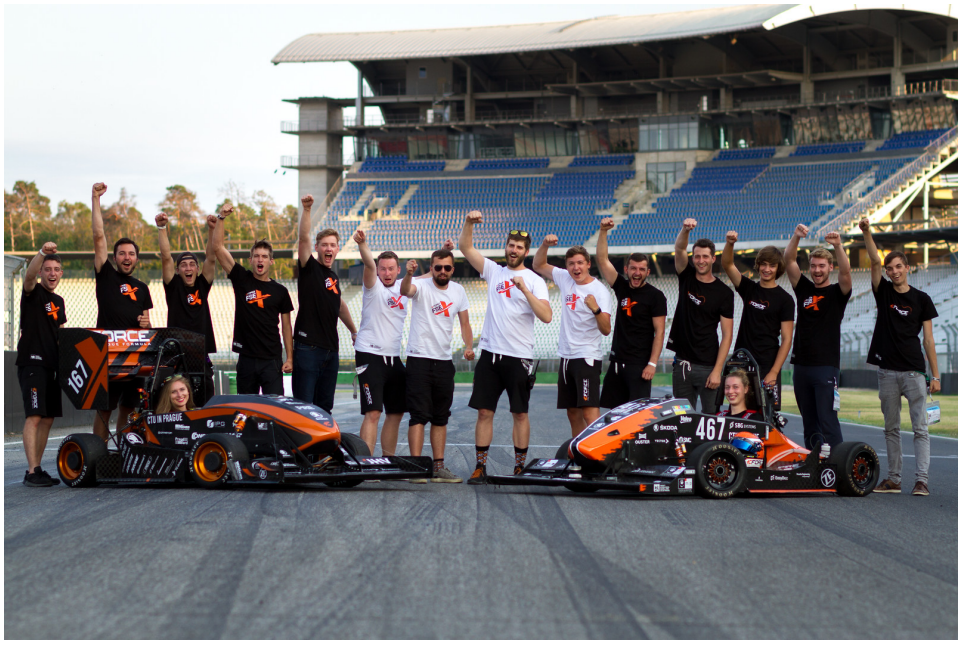


Fig. 2: DV.01 and FSE.X [2]

Subsection 1.2.1

DV.01

Formula DV.01 is the first autonomous formula built by eForce Driverless. DV.01 has a rear wheel drive with a power of up to 70 kW. The power source of the car is a 400 V accumulator of our own design.

The main processing unit responsible for the autonomous pipeline in DV.O1 is Zotac MBOX (Intel i7, Nvidia GeForce 2070 Super). Zotac receives the information from the Stereolabs ZED camera via USB, LiDAR data from Ouster OS1-64 from the ethernet connection (UDP protocol) [8]. And lastly information about car, including position, velocity, and acceleration from dual antenna INS from SBG Systems over CAN bus.

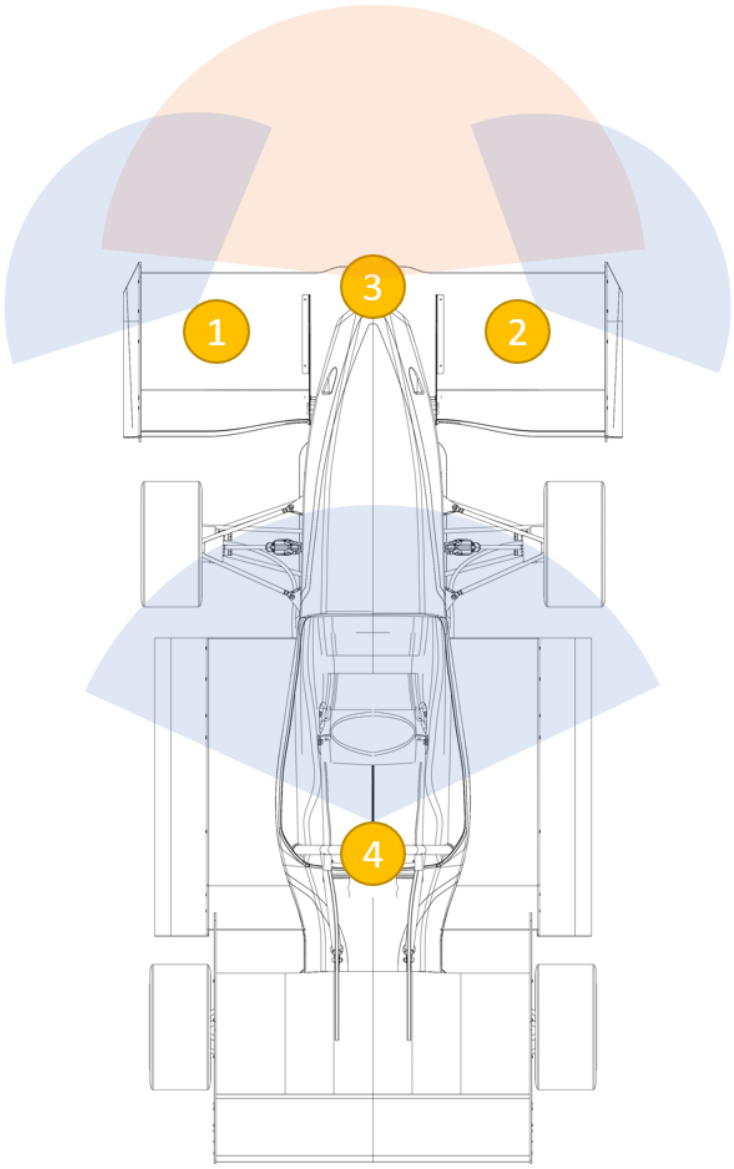


Fig. 3: Sensors placement on DV.01 [3]

The autonomous pipeline is implemented using the ROS2 framework, with most of the building blocks (nodes) written in Python. ROS framework was used for modularity and the possibilities to exchange and reuse function blocks if needed without problems, with the condition to have the same output and input topics.

Currently, the main information source for the car is its stero camera. The images from this camera go to the neural network based on the YOLO-V3 architecture. [9] The goal of this modified YOLOV3 is to find cones that delimit the track. The cones detected in the image are then transformed to the car coordinates relative to the point at which the image was taken.

Section 1.3

Motivation

The track of every dynamic discipline for the driverless formula in the competition is marked by traffic cones, see 4. It is important to know quickly and precisely where such cones are for the formula to be able to map its surroundings and plan its movement.

Fig. 4: Image of the track for the trackdrive event at FSG 21 in Hockenheim. The track is marked with blue cones at the left side of the track and yellow cones at the right side of the track. Image of [3].

Of many sensors equipped at DV.01 it is possible to detect cones with two of them, the camera and LiDAR. LiDAR has many advantages compared to other sources of surrounding data (mainly stereo cameras). The most important thing is its accurate measurement. The error of range measurement on LidAR OS1-64 it is from 0.7 cm to 5 cm [10]. This amount of precision is more than ten times better than that of the cameras at 10 m distance [4].

This increased precision is most noticeable when mapping the surrounding, when the accuracy of the input data for the mapping algorithm is one of the most important parameters. [11] With improved processing, these algorithms can trust more input data and converge faster to the complete result, in this case map of the surrounding and location of the car inside it.

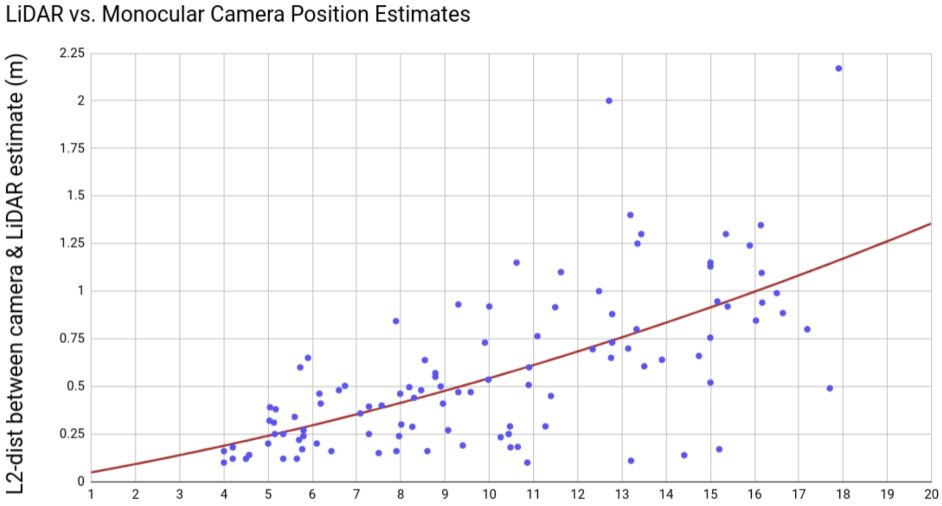


Fig. 5: Measurement error of mono camera against LiDAR. The graph shows that the error of measurement is is growing with distance. Also the error is not stable, it varies detection by detection. Taken from [4].

The benefits of the wide field of view of the LiDAR also help to navigate in small radius turns. In this type of turns, the camera has a problem seeing the cones on the inside of the track. The problem is with how the geometry of the car works in turn, see .

With bad lighting conditions the camera starts to lose its performance, this is more obvious with the neural networks that are trained to detect the colour of the cone, during sunset or sunrise this problem is that the colour of the light also changes the perceived colour of the cones, see Figure 6. The LiDAR does not have this kind of problem, since it does not depend on the ambient light condition.

Fig. 6: Example of bad light conditions which are difficult for the camera to process. LiDAR

Chapter 2
State of the art

Section 2.1
LiDAR detectors

Most of the current LiDAR detectors use neural networks. as their backbone. The first step of these detectors is to flatten the input user data to one or more images. [12][13]. Images can be generated by multiple methods. The most common is the bird view of the pointcloud. Individual pixels in the image are calculated by several methods. The first step of every method is to set the resolution to the area that a pixel represents. This influences the amount of information we lose and the input dimension that a neural network must have. The image is typically greyscaled, but can also be coloured or multichannel. If the image is grayscaled, then the information is usually the maximum height of the points in the given pixel. When there are more channels, we also add intensity (average or maximum) and density. With more channels, we can add other information, for example, divide the point cloud to height maps. Examples of such projection is at Figure 7.

The common thing about all of detectors that use this principle is the fact that they use neural networks and thus require a GPU (Graphics Processing Unit) in the machine to efficiently process input data. With the usage of neural networks comes also an increase in computational time that ranges from 0.1 s to 3.3 s with dependency on image resolution and network architecture [5][14].

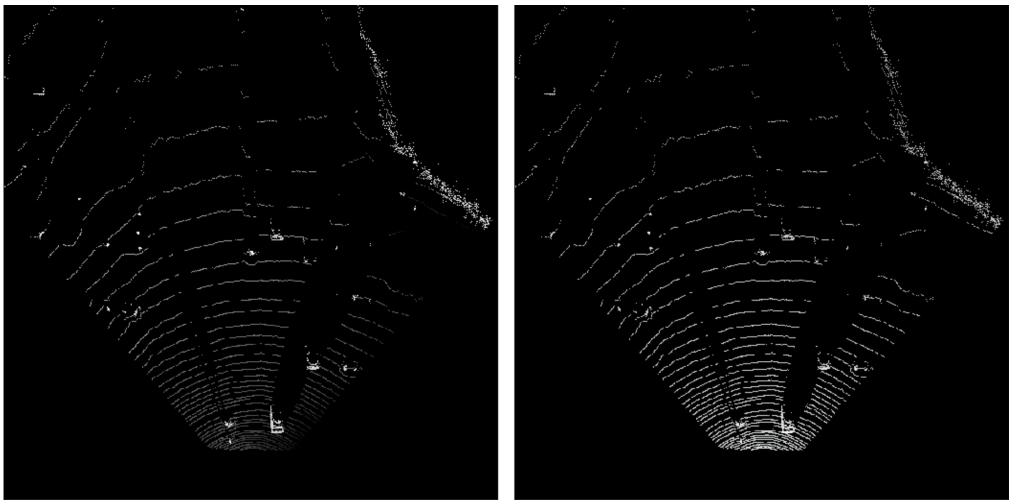


Fig. 7: Sample of the input grid maps. Left: the height map. Right: the density map. Taken from [5].

Chapter 3
Proposed solution

This section is divided into two main blocks. In the first part, we will present some known facts that directly or indirectly influence the design of the detector. The second block presents the proposed detector of traffic cones and explains what individual steps do.

Section 3.1
Known facts - rules

Formula Student competition is held on a closed track and in a very specific environment. The exact type of cones and the rules on how they are placed is known. The basis is the FSG Handbook [6], from which we can get some basic information about the placement of cones.

1. The maximum distance between the cones is 5 m. In corners it might be less
2. Left side of the track is marked with blue cones
3. Right side of the track is marked with yellow cones
4. Start of the track / lap is indicated by big orange cones
5. Entry or braking zone for acceleration event is marked by small orange cones

For us, the most important cones to detect are the cones that indicated the way of the track, not the special indicators (large orange cones). The cones that are important to detect have a base 22.8 cm x 22.8 cm with a height of 32.5 cm, as can be seen in Figure 8.

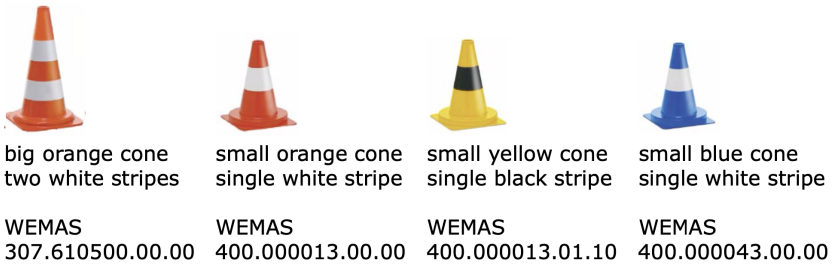


Fig. 8: Example of cones at FSG 2022 [6]

Formula Student competitions are usually held on racetrack (FSG - Hockenheimring, FS Czech - Autodrom Most, FS Spain - Circuit de Barcelona-Catalunya), from this we can say that the surface directly below us and in close proximity will be plane, or it will be surface which is close to the plane. We can also safely assume that above any of the cones at the track there will be no object present. This is also given by the location of the competitions and the nature of the tracks that we race on. There must be a safety proximity near the race in case of incidents. This is also much more important for the autonomous category, as there is no person behind the wheel who could save the situation.

This takes us to what we can expect on the side of the race track. In most of the locations, there is empty space. Gravel in the case of FS Spain or grass in the case of FS Czech. However, in FS Germany there is a wall made of tyres. The tyre wall is present it is similar in height to the cones (approximately 50 cm), but the wall is continuous without any space between.

Section 3.2

Known facts - LiDAR

Subsection 3.2.1

LiDAR principle

LiDAR works on the principle of measuring time of flight of laser pulse. From the time of flight of the pulse, we can measure the range by (1).

$$r = \frac{1}{2n} c\Delta t \tag{1}$$

Where c is speed of the light, Δt is time of the flight and n is the index of refraction of the propagation medium (for air $n \approx 1$) [7]. The LiDAR also returns the power of returned pulse, the power can be expressed by (2)

$$P_r = E_p \frac{c\eta A_r}{2r^2} \beta T_r \tag{2}$$

where E_p is the total energy of a transmitted laser pulse, c is the speed of light, A_r is the receiving sensor area. r is the detection range, β is the reflectance of the target surface, and T_r is the loss of energy through the air in our case [7].

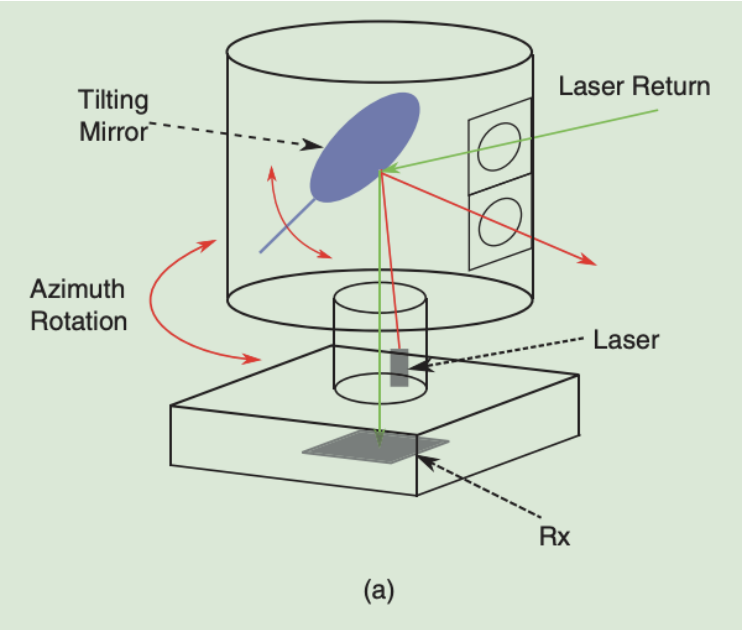


Fig. 9: Internal mechanism of the rotating LiDAR, adapted from [7].

The Ouster OS1-64 is an LiDAR with wide field of view, this is done by rotating the LiDAR around the base (see Figure 9). The Ouster OS1-64 has 64 lasers to

remove the need for tilting mirror and increase resolution.

For the moving LiDAR we need to compensate the speed at which the LiDAR moves. If we do not compensate for the speed the car, the final pointcloud would became distorted. The compensation is done by the LiDAR firmware, which takes the input from the IMU located in the LiDAR itself.

Subsection 3.2.2

Ouster OS1-64

Fig. 10: LiDAR used on DV.01 - Ouster OS1-64 located on the front wing of the formula.

The LiDAR that we have installed, Ouster OS1-64, has a vertical resolution of 33.2° centered around the horizontal axis, so 16.6° to each side. With up to 1024 or 2048 channels per rotation [10]. The placment of the LiDAR on the car can bee seen on 10.

The data output, as mentioned in Subsection 1.2.1, comes in the form of UDP packets. From every packet we need this information (the complete description can be found in [8]).

- Packet header - the most necessary information is the **measurement id** value between 0 and 1024 in our configuration
- 64 channel blocks - block for every vertical channel
  - Range - rounded to the nearest millimeter
  - Intensity - Signal intensity photons

Subsection 3.2.3

Transformations

To transform the range values, the channel block id and the measurement id to the coordinates $(x,\ y,\ z)$, we need to apply the Eq. (3) to Eq. (9) from [8]. Where n is the sensor offset from the origin of the LiDAR coordinate system, $range$ is the measurement of the ith channel of the selected block. The $beam\_azimuth\_angle \in (-3, 3)$ and $beam\_altitude\_angles \in (-16.88, 16.42)$ for the ith channel can be obtained from the LiDAR.

$$r = range + n \tag{3}$$

$$\theta_{encoder} = 2\pi \cdot \left(1 - \frac{\text{measurement id}}{1024}\right) \tag{4}$$

$$\theta_{azimuth} = -2\pi \frac{beam\_azimuth\_angle}{360} \tag{5}$$

$$\Phi = 2\pi \frac{beam\_altitude\_angles}{360} \tag{6}$$

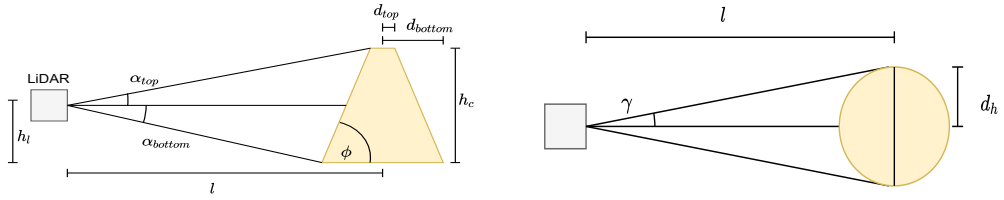
$$x = (r - n)cos\left(\theta_{encoder} + \theta_{azimuth}\right) cos(\Phi) + ncos(\theta_{encoder}) \tag{7}$$

$$y = (r - n)sin\left(\theta_{encoder} + \theta_{azimuth}\right) cos(\Phi) + nsin(\theta_{encoder}) \tag{8}$$

$$z = (r - n)sin(\theta_{encoder}) \tag{9}$$

Subsection 3.2.4

Maximum useful distance



(a) Side view (b) Top view

Fig. 11: View of modeled situation

To find the maximum useful range of the LiDAR, we can model the best sitation for the detecting the cone and calculate the number of points per cone as a function of distance. In the first step, we need to find out at what view angle of the LiDAR the cone has at the given distance. However, before that, it is necessary to calculate some LiDAR parameters that are not listed directly on the datasheet. All other parameters, which are not calculated, are taken from the LiDAR data sheet [10]. At (10) and (11) we calculate the angular resolution of LiDAR. The # *horizontal channels*

and # *verticals channels* are used the same as the configuration used in the car itself.

$$\text{vertical resolution} = \frac{vertical\ FOW}{\#\ vertical\ channels} = \frac{33.2^\circ}{64} \doteq 0.519^\circ \tag{10}$$

$$\text{horizontal resolution} = \frac{horizontal\ FOW}{\#\ horizontal\ channels} = \frac{360^\circ}{1024} \doteq 0.352^\circ \tag{11}$$

Where l is the distance from the centre of the cone to the LiDAR, h_l is the height of the LiDAR above the ground, h_c is the height of the cone, d_{top} and d_{bottom} are the diameters of the top and bottom of the cone, see Figure **??**.

$$\alpha_{top} = -atan2(h_l, l - d_{bottom}) \tag{12}$$

$$\alpha_{bottom} = atan2(h_c - h_l, l - d_{top}) \tag{13}$$

Equations (12) and (13) give us the view angles of the cone, but to find the number of points, we need to obtain the angle of the vertical ray that is capable of hitting the cone.

$$\alpha_{corrected_{\{top,bottom\}}} = floor\left(\frac{\alpha_{\{top,bottom\}}}{\text{vertical resolution}}\right) \cdot \text{vertical resolution} \tag{14}$$

For later calculations, it is useful to convert this angle to the ray index by (15).

$$r_{\{l,h\}} = \frac{\alpha_{corrected_{\{top,bottom\}}}}{\text{vertical resolution}} \tag{15}$$

The last thing we do before we can get the result is to get the width of the cone at the height at which the ray hits it. Where ϕ in (17) is the angel of the side of the cone, see Figure 8.

$$h = h_l + d \cdot \sin\alpha \tag{16}$$

$$d_h = \tan\phi \cdot (h_c - h) + d_{top} \tag{17}$$

Knowing the width and distance of the target line at which we should project our rays, we can calculate how many of the rays on the horizontal axis will hit. We first calculate the view angle in the horizontal direction at which we see the cone and what rays from the LiDAR correspond to it. We do it only for one side of the cone, since the cone is symmetrical. The steps are almost the same as for the vertical rays.

$$\gamma = atan2\,(l, d_h) \tag{18}$$

$$\gamma_{corrected} = floor\left(\frac{\gamma}{horizontal\ resolution}\right) * horizontal\ resolution \tag{19}$$

The final number of points for a given distance is by

$$\#\ \text{of points}\,(l) = \sum_{r_i=r_l}^{r_h} 2 \cdot \frac{\gamma_{corrected}(r_i)}{horizontal\ resolution} \tag{20}$$

The result if we iterate over several distances can be seen in Figure 12. From the figure, we can see that the possible number of points at the cone dramatically decreases.

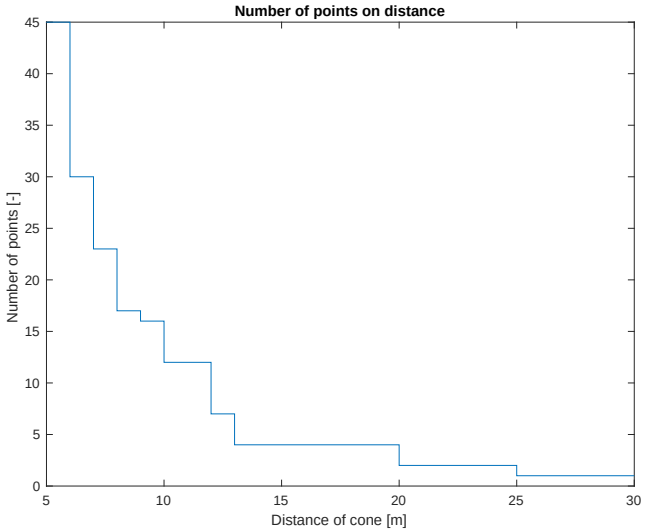


Fig. 12: Simulation of rays hit on cone in regard to the distance.

This also limits our effective maximum detection distance to this value. However, Figure 12 can be too optimistic. This figure shows the ideal situation where the centre of the cone is aligned with the LiDAR.

Subsection 3.2.5

Points distribution

The second thing we need to keep in mind is how the points are returned to us. LiDAR in the configuration used in DV.01 returns 65536 points (64 rows of points and 1024 columns). However, only $\sim$ 12% is actually reflected from the ground or objects, see Figure 13. All other points are lost, mostly because such points are not reflected back at the sensor.

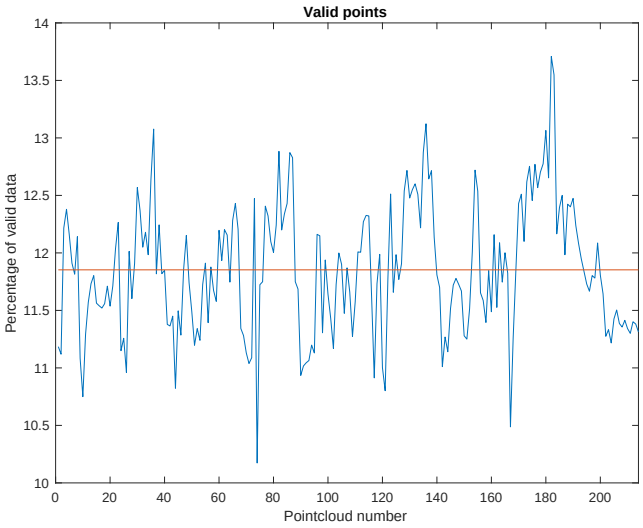


Fig. 13: Percentage of valid point at individual pointclouds from the LiDAR. All of this data was obtained at testing with DV.01 on airfield near Milovice in March 2022

The mounting point of LiDAR on the car (see Figure 10) heavily influences the look of the output data. In the case of DV.01, where LiDAR is placed only slightly above the ground (10 cm) the majority of points are located near each other and are at ground level. The LiDAR was placed on the front wing for two reasons. At the front wing, we can place the LiDAR at the level of the cones in order to get the best chance to scan the cones with LiDAR. The second reason is the unobstructed view. Only in this position are we able to detect cones that are near us at wide angles.

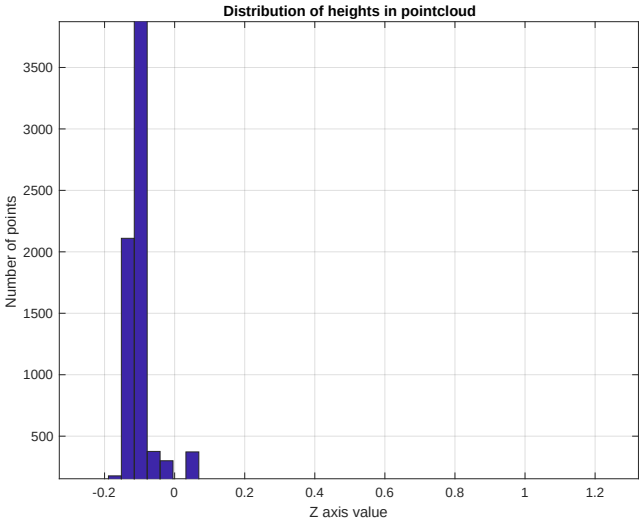


Fig. 14: Distribution of points along the Z-axis of LiDAR coordinate systems

Section 3.3

Proposed algorithm

The detector of the traffic cones has at the input pointcloud, an unsorted array of points. The output of the detector is and n-coordinates, each representing the detected cone. The selected approach was from the beginning mainly influenced by the requirement of computational speed of detection. To be able to detect cones with focus on speed, we need to utilize the facts that we discussed in previous sections of this thesis. First, we know that most of the points are near the ground, see subsubsection 3.2.5. Therefore, we should be able to find the ground surface below the car. This surface can be approximate with the plane, and this simplification can be done due to the short range of possible defections (see subsubsection 3.2.5). The base algorithm can look as follows in algorithm 1.

Algorithm 1 High-level overview of the detection algorithm

```
1: function CONE_DETECT(pointcloud)
2:     pointcloud_valid ← filter_valid_points(pointcloud)   ▷ Remove invalid points
3:     plane ← find_plane(pointcloud_valid)                 ▷ Find the ground plane
4:     candidates ← filter_points(pointcloud_valid, plane)  ▷ Select candidates for
   the cones
5:     detections ← detect(candidates)                      ▷ Final detection of cones
6:     return detections
7: end function
```

Section 3.4

Filtering valid points

This section describes the line 2 of the algorithm 1. The manual for the LiDAR Ouster OS1-64 says that all rays that do not have reflection back, have their range values set to 0 m [8]. This we get as the point $(0,\ 0,\ 0)$, in the XYZ frame of the LiDAR, thanks to the coordinate calculation which we established in 3 to 9. Therefore, the algorithm for removing invalid points is as follows.

Algorithm 2 Filtering of valid points

```
1: function FILTER_VALID_POINTS(pointcloud)
2:     pointcloud_valid = []                                ▷ Empty return array
3:     for point ∈ pointcloud do
4:         if point.x = 0 and point.y = 0 and point.z = 0 then   ▷ Checking for
   (0.0, 0) points
5:             continue;
6:         else
7:             pointcloud_valid.append(point)
8:         end if
9:     end for
10:    return pointcloud_valid
11: end function
```

Section 3.5

Detecting ground plane

In this section we discuss the line 3 of algorithm 1. A detection of the ground plane can be a challenging task with some uncertainty in our input data, it is hard to say what ground plane is. We cannot use the least squares method to find a plane because such a plane would be heavily affected by cones and objects around the track. But with the knowledge that most of the points that are returned and are valid are from the ground plane. We can use the robust algorithm the RANSAC (RANdom SAmple Concensus) algorithm [13]. In our case, the input data for the RANSAC is not the whole pointcloud, but only a small subsample of 300 points from the overall 65 536 points. With this, we can decrease the time it takes for the RANSAC to calculate which points are in our inliners and which not.

We can safely say that most of the points are at ground level see Figure 14. From this we can assume that, by picking N random points, we would also most likely pick points from the ground. This leads to a modified RANSAC algorithm. The

Algorithm 3 RANSAC algorithm

```
1:  function RANSAC(pointcloud)
2:      max_iterations ← ITERATION LIMIT
3:      inliners_stop ← INLINERS LIMIT
4:      inliners_distance ← INLINERS MAXIMUM DISTANCE
5:      test_sample = pick_random(pointcloud, 300)      ▷ Pick 300 random points
6:      iteration = 0
7:      best_plane = None
8:      best_plane_inliners = 0
9:      while iteration ≤ max_iterations do
10:         plane = pick_random(test_sample, 3)          ▷ Pick 3 random points
11:         inliners = calculate_inliners(plane, inliners_distance)
12:         if count(inliners)> inliners_stop then
13:             return plane, inliners
14:         else
15:             if count(inliners)>count(best_plane_inliners) then
16:                 best_plane = plane
17:                 best_plane_inliners = inliners
18:             end if
19:         end if
20:     end while
21:     return best_plane, best_plane_inliners
22: end function
```

RANSAC algorithm 3 in a maximum of *ITERATION LIMIT* iterations tries to find the best plane for the randomly selected points, see line 10. The plane is constructed from two random points from the selected points. It is done by cross product of these two point, which is the normal vector of the plane. Then we calculate at line 11 the number of inliners. This is done by checking if the distance to the plane is less than INLINERS *MAXIMUM DISTANCE*. If the number of inliners is more than threshold *INLINERS LIMIT* we end the algorithm and return the plane with its inliners. If we do not find the plane in the maximum nuber of iterations we return

the plane which had most inliners so far.

It is possible, to improve upon the result we get from RANSAC by fitting best plane in the sense of least squares on inliners, which we get from the RANSCAC algorithm. This is due to the fact that RANSAC only returns a plane that fits the best number of points with the error which we allow. By running optimisation in the sense of least squares, we minimise this error.

Section 3.6

Selecting candidates

Process behind selecting possible candidate for the cone is two-part. In the first part, we divide the points accordingly by their height above the plane detected in the previous step. The categories to which we are dividing in the algorithm 4, are as follows.

1. **Air points**, points that are very high above the plane. These kinds of point do not have any chance of becoming candidates, but are still useful for filtering out invalid points later on.
2. **Ground points**, Most of the points are there. These points are very close to or near the ground plane.
3. **Candidate points**, this is the category that is most useful. The number of points is not enough to use only them for detection later on.

Algorithm 4 Selection of candidates - first part

```
1:  function SELECT_DIVIDE_POINTS(pointcloud, plane)
2:      ground_points = []
3:      air_points = []
4:      candidate_points = []
5:      for point ∈ pointcloud do
6:          if height_above_plane(point, plane) < GROUND_LIMIT then
7:              ground_points.append(point)
8:          else
9:              if height_above_plane(point, plane) < AIR_LIMIT then
10:                 candidates_points.append(point)
11:             else
12:                 air_points.append(point)
13:             end if
14:         end if
15:     end for
16:     return air_points, ground_points, candidate_points
17: end function
```

In the next step, we do find for every point in air points any point that is in *AIR_EXCLUDE* from the air points, for illustration, see Figure 15. We then remove these points from later usage. We do this because we know that the races are run on an open-space track and nothing can be above the cones. This is done on the line 5 of algorithm 5.

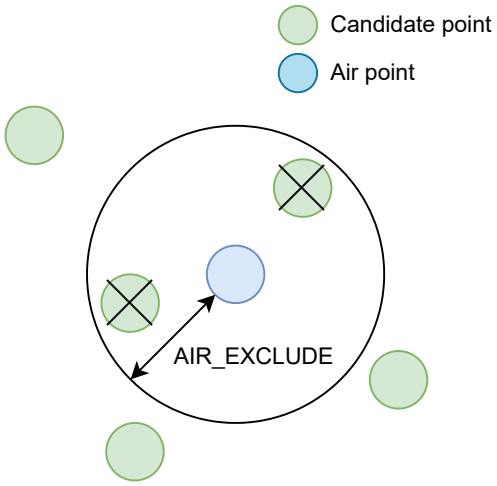


Fig. 15: Illustration of removing the points that are in *AIR_EXCLUDE* distance from the air point.

In the second part of this step (line 13 of algorithm 5), we will expand our possible candidates for the cones by including near-ground points to the candidate points. By doing so, we can greatly expand the number of points on which we can try to find cones.

Algorithm 5 Selection of candidates - second part

```
1:  function SELECT_POST_FILTER(ground_points, candidate_points, air_points)
2:      for point ∈ air_points do                ▷ Filter out points below air objects
3:          for candidate ∈ candidate_points do
4:              if candidatenearpoint then
5:                  candidate_points.remove(candidate)
6:                  break
7:              end if
8:          end for
9:      end for
10:     for candidate ∈ candidate_points do                ▷ Include ground points
11:         for point ∈ ground_points do
12:             if candidatenearpoint then
13:                 candidate_points.add(point)
14:             end if
15:         end for
16:     end for
17: end function
```

Section 3.7

Detection

With the results of the previous step, we get a filtered unsorted list of points that may or may not be our cones. The most common false positives that we can detect are grass and walls. Both of these problems can be addressed.

The first step of the detection is to cluster points into individual cones, or what we first think the cones are. This is done only by their relative distance algorithm for such clustering, according to the algorithm 6. The value of the parameter *CLUSTER_DISTANCE* used in the line 7 must be larger than the width of the cone (22.8 cm), to ensure that all points of the cone can fit within this distance.

Algorithm 6 Detection - Clustering

```
1:  function CLUSTERING(points)
2:      clusters = ∅
3:      while points ≠ ∅ do
4:          point = points.select_random();
5:          cluster = ∅
6:          for p ∈ points do
7:              if distance(point, p) < CLUSTER_DISTANCE then
8:                  cluster.add(p)
9:                  points.remove(p)
10:             end if
11:         end for
12:         clusters.add(cluster)
13:     end while
14:     return clusters
15: end function
```

The false candidate points, e.g. straws of grass, are in most cases present in the form of the isolated points. If we know this, we can add a filter to look for the minimum density of points at the location. If the number of points is greater than a minimum value, then we can keep the points. It can remove some of the points that are farther away, but since the focus of this detector is to minimise false positives, we can accept this loss in improving the precision.

With filtered grass and other sparsely located points, we can focus on detecting the cones. Often, from the principle of LiDAR technology, we only get one horizontal line of the points. This type of return is most often seen with the ground at longer distances (range over 20 m). From the points in one line, we cannot for sure say if the result is cone or not. Example of points in one line can be seen on Figure 16. To eliminate this issue, one needs to look for any difference in the z coordinate of the points in the group of the clustered points.

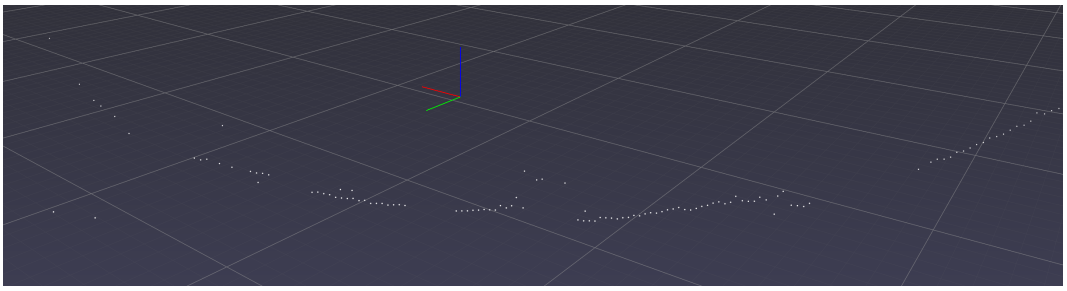


Fig. 16: Points returned by selection candidates, where we can see the lines returned with no difference in their z coordinate

The last thing that must be resolved is the detection of walls. Again, we can

use knowledge of the rules. The rules and experience say that no cones can be right next to each other. This is soled by introducing a filter at the end that looks at the original group of points and the location of the possible cone. If there is no point near our cone, then we can say that this is the cone, otherwise this points are excluded from the detected cones.

For the output of the detector and later stages, it is not very convenient to transform the points that form the detected cone into a single coordinate. The most basic method to obtain this single coordinate is to take the average of all points.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} \frac{p_i.x}{n} \\ \sum_{i=1}^{n} \frac{p_i.y}{n} \\ \sum_{i=1}^{n} \frac{p_i.z}{n} \end{pmatrix}$$

However, this does not represent the real location of the cone. With the LiDAR data, we do get only one side of the cone, and when we average it we only get positions that are inside the cone but not exactly in the centre of the cone. We still can accept this kind of error in our detector. The maximum possible error in the cone would be 8 cm (diameter of the base of the cone). This error is negligible in the scope of the autonomous car. The error in path tracking and safety limits (the distance allowed from the cone) will be much larger than the error introduced by this approach.

The better method to get the real position of the cone is to find the centre of the circle, which is formed by the points projected onto the cone surface in one height. If we select the best circle, then it is possible to find this centre, for example, by the method of least squares.

The final algorithm for detection is summarised in algorithm 7.

Algorithm 7 Detection of cones

```
1:  detections = ∅
2:  function DETECT(candidate_points)
3:      clusters = clustering(candidate_points)
4:      for c ∈ clusters do
5:          if is_dense(c) then
6:              continue;
7:          end if
8:          if wall_near(c) then
9:              continue;
10:         end if
11:         detections.add(cluster)
12:     end for
13: end function
```

Section 3.8

Time optimisation

The detector presented in the previous sections can be made faster, with minimal modifications. The main bottleneck of the proposed algorithm is the step of selecting the candidate points shown in the algorithm 5. In this step, we need to compare every candidate point with every ground point. We group the raw points into smaller

bins, a data structure that holds points in one location, by their position, then we can only compare the candidate points with the ground points near them. The process of dividing the point does take some non-trivial time.

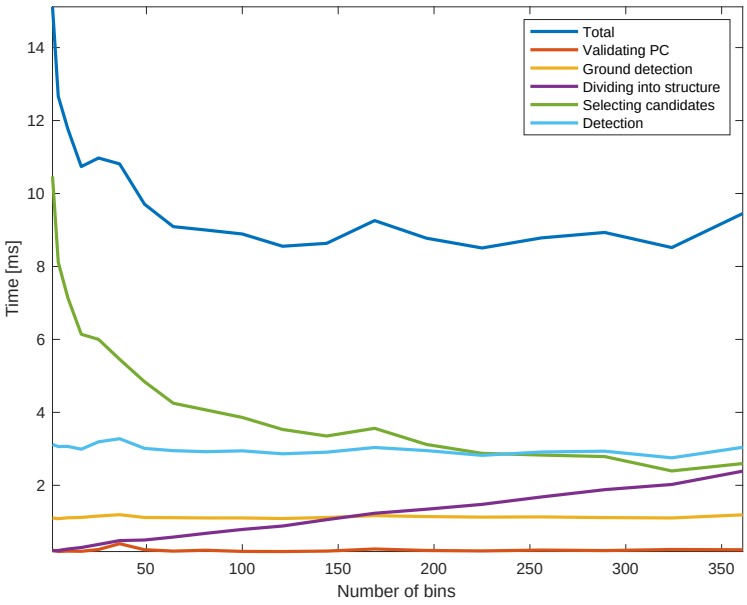


Fig. 17: Time analysis of the 20 runs of the detector over 215 pointclouds averaged. The experiment was done at computer with i5 2.4 GHz, 8 GB and NVME SSD.

As seen in Figure 17, the optimal number of bins is around 100. The times of validation of the point cloud, finding the ground plane, and detecting the final cones remain constant, no matter how many bins we use. But selecting candidates is non-linear with a curve that can approximately look like a function of $\frac{1}{x}$, while the time dividing the points between the bins is linearly increasing.

When modifying the algorithm 5, we need to keep in mind that splitting the space raises another problem, and that is what we should do if our cone lies on the border between one or more bins. The solution is to simply include for each of the bins in this step its immediate neighbours. This situation can be seen in Figure 18. The solution for the problem is to add for each of the bins that we process its neighbours.

Let us analyse the effect of using the bins by introducing a rough estimate that can be calculated when we look at the numbers of points. The complexity of the big O notation can be expressed as (21), where c is the number of candidate points, a is the number of air points, and g is the number of ground points.

$$\mathcal{O}(c \cdot (a + g)) \tag{21}$$

We can safely say that $a \ll g$ so the complexity is $\mathcal{O}(c \cdot g)$. We cannot calculate the theoretical complexity of the modified algorithm because it depends on the distribution of points in the bins. Still, it is possible to say that the complexity for two edge cases is that all points are located in a single bin and the points are evenly distributed across all bins. In the first situation, the complexity remains the same. In the second situation, we can start from the original complexity. The number of points in the bin became $c' = \frac{c}{n}$ and $g' = \frac{g}{n}$ where n is the number of bins. Then the

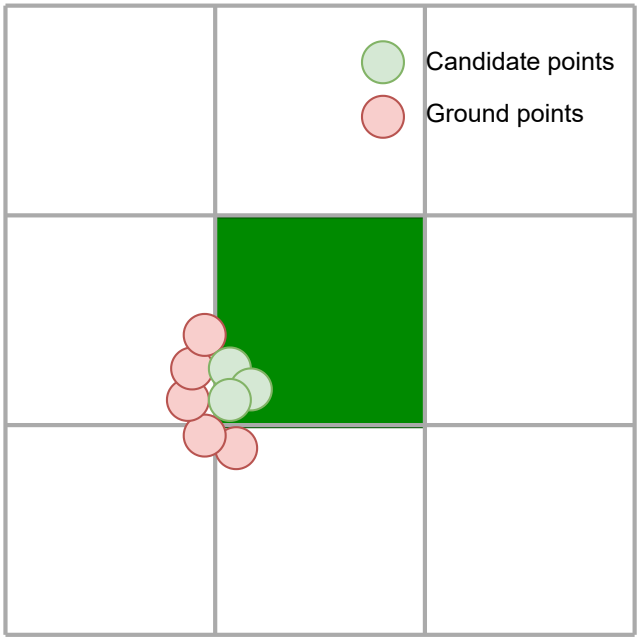


Fig. 18: Example situation of points at the border between bins. The green square represents current bin, if the neighbours weren't included no ground point would be added and the chance of detecting would decrease.

complexity is $mathcalO\,(9n \cdot c'g')$, 9 is the number of bins that we need to proceed for each bin (bin itself plus the eight neighbouring bins).

if we express it with the original values g and c,

$$\mathcal{O}\left(\frac{9 \cdot cg}{n}\right) \tag{22}$$

This leads us to the conclusion that the complexity of the modified algorithm in the optimal situation is (22) and in the worst case is (21). The real value cannot be expressed because the improvement is dependant on the structure of input pointcloud.

Chapter 4

Implementation

Section 4.1

Python and C++

Implementations of the detector described in the previous chapters were made in *Python 3.9* and *C++* (standard C20). The logic behind both of these implementations is identical. Python implementations are heavily dependent on the *numpy* library. [15] Especially to it's operation over arrays (*numpy.all*, *numpy.any*) and to solve least squares problem. The C++ version uses only the Eigen library to solve the least squares problem. [16]

The performance of the C++ versions is without doubt faster. The main reason is the difference in speed between the interpreted languages. Python program will also have worse performance than the C++ program.[17] The second problem with Python is its memory management capabilities. In C++, memory management is completely in the hands of the user and is as transparent as possible. In Python and especially with a library as complex as Numpy, memory becomes impossible to do. The downfall of the Python implementation is memory management. Specifically copying the array to the complex operations inside numpy. The time difference between the Python and C++ implementation can be seen in Table 1. We can see that the C++ version is much faster.

	Best time	Worst time	Average time
Python	12.0 ms	244.5 ms	57.2 ms
C++	3.2 ms	23.8 ms	8.3 ms

Tab. 1: Run times of detector at computer with Intel i5

Section 4.2

Finding parameters

Subsection 4.2.1

Scoring

To find the best configuration, we need to be able to say which configuration is best by a number, based on the output of comparing the detected cones with ground truth data. Possible results are

- True Positive (TP) - correctly detected cone
- False Positive (FP) - incorrectly detected cone
- False Negative (FN) - not detected cone

True positive cones are cones that have a labelled cone at a maximum distance of 10 cm. The false positive cone is a cone that does not have a labelled cone near

it. False negatives are labelled cones that remain after evaluating all detected cones and are not assigned to any detected cone.

The outcome of the evaluation is then weighted on the basis of the distance of the TP/FP/FN. We do this because the closer cones are more important to car navigation than the ones that are far away. The weights used are 10 for close cones (the distance of the cone from the LiDAR is less than 10 m). The weight of 5 for the cones at a medium distance (the distance from LiDAR is between 10 m and 20 m). The other cones have a weight of 1.

We can use the *ROC* curve since the true negatives do not make sense for our detector. Instead, we can use the precision calculated by (23) and the recall calculated by (24) [18].

$$P = \frac{TP}{TP + FP} \tag{23}$$

$$R = \frac{TP}{TP + FN} \tag{24}$$

In the F_1 score is calculated as (25), where P is precision and R is recall.

$$F_1 = 2\frac{P \cdot R}{P + R} \tag{25}$$

Section 4.3

Grid search

The final algorithm of seven parameters that influences the entire detection process. The first four of these seven parameters are used around the selection of candidate points.

1. Ground Limit - Height limit of the points above the ground plane to be considered a ground point.
2. Air Limit - Start of height above ground plane for points to be marked as air points.
3. Air Exclude - Distance of the point to any air point to be removed from further usage.
4. Ground Include - Maximum distance of ground points to the candidate point to be included into candidate points in later steps.

The latter three parameters are used in the detector itself.

5. Grouping Distance - Distance at which points are clustered together.
6. Number of Points - Minimal number of points for the cluster to be processed further.
7. Empty Distance - Empty space size required around any cone.
8. Height Difference - Minimal height difference of the points in the cone

The method used to find the best configuration of the parameters is a grid search. We have generated every possible configuration of parameters with a certain discrete step and then tested every configuration on the training dataset. We have gone through 180000 different combinations. They were generated from the settings in Table 2.

Parameter	Start value	End value	Steps	Selected value
Ground limit	0.05 m	0.2 m	5	0.15 m
Air Limit	0.5 m	3.0 m	5	1 m
Ground Include	0.0 m	0.15 m	5	0.0325 m
Air Exclude	0.0 m	0.1 m	5	0.05 m
Grouping Distance	0.5 m	1.5 m	5	1.25 m
Number of Points	1 points	3 points	3	0.0325 m
Empty Distance	0.1 m	2.0 m	5	0.3 m
Height Difference	0.0 m	0.1 m	4	0.0 m

Tab. 2: Parameters of grid search for each of the parameters with the final value.

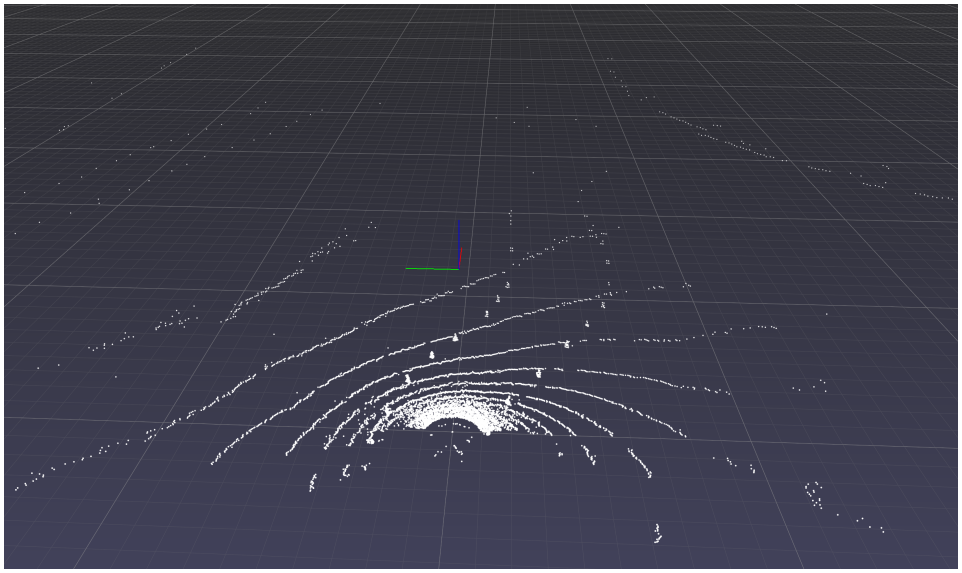


Fig. 19: Image of LiDAR data pointcloud

Chapter 5
Evaluation

All evaluation of the proposed detector was done on a data set manually labelled by volunteers participating in the eForce Driverless project. Unfortunately, with this comes human errors in labelling the dataset. These errors are even made clearer by the fact that the LiDAR data are not as clear as the images that are labelled for the neural networks. When displaying 3D points at 2D displays, we lose perspective and the precise placement of the points can be misinterpreted. Another thing that makes labelling LiDAR data a challenging task is the fact that humans are not used to interpreting these kinds of data, the example pointcloud is at Figure 19. And finally, there is a requirement that the people who label the data do not try to find patterns in the data.

The labelling itself is done by selecting points in 3D view of pointcloud by drawing a rectangle over the point. Doing the labelling this way makes the labelling easier, but the final cones also include ground points.

The input data were always taken from some ride of the formula through the track, and all the pointclouds were just a small translation and rotation from the previous one. However, this does not respond to what the detector has available. All pointclouds must be looked at individually without any memory of the past. This leads to people marking fewer points as cones, which is not entirely bad. The detector should be more precise than detecting every cone that exists. For the autonomous formula, it would be much worse to get false information about the cones than to get an incomplete one. The hard to label cones are at Figure 20.

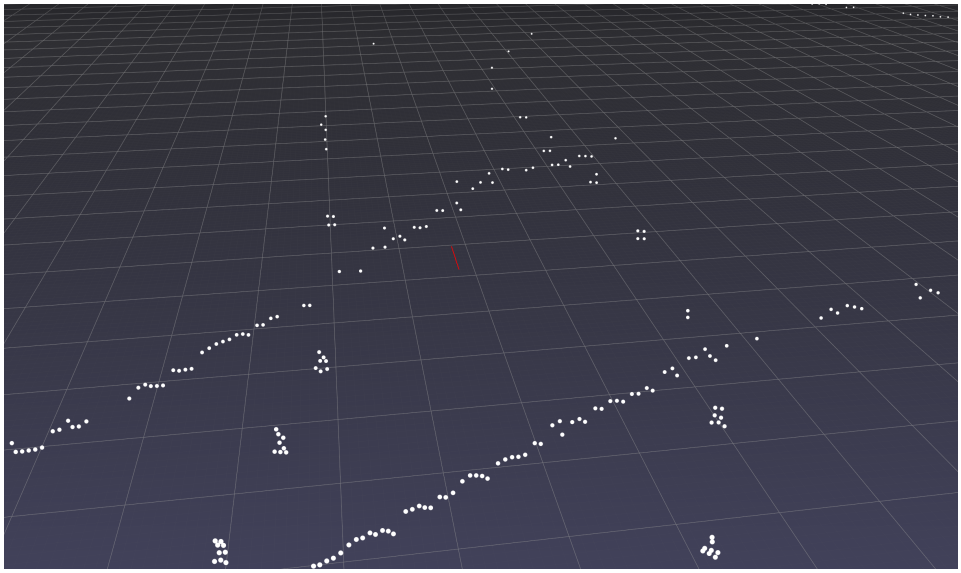


Fig. 20: Hard to label cones, first cones are clearly cones without any other information, but the ones in the back may be hard to say without looking at the lines which forms the track

Section 5.1

Datasets

For the training and evaluation of the detector, two datasets were presented. The first dataset was used to train the detector, i.e. to find the best configuration. This dataset contains almost 300 different point clouds. These pointclouds were taken in three different situations.

Subsection 5.1.1

Training dataset

The first situation is a low-speed track driven at the Milovice airport. This situation is characteristic of a stable position of the car, small to no-pitch rotations, and very small roll rotations of the car. The cones in this situation are relatively far from each other and the surrounding areas contained low grass at the edges of the track. This is one of the best situations for the detector, as there will be a minimum of misleading groups of points.

The second situation in the training dataset is the trackdrive driven at higher speed, near the handling limits of the car. This situation is the worst-case scenario for ground plane detection, since there is a lot of roll and pitch movement caused by quick changes in speed and directions of car movements.

0.49

Subsection 5.1.2

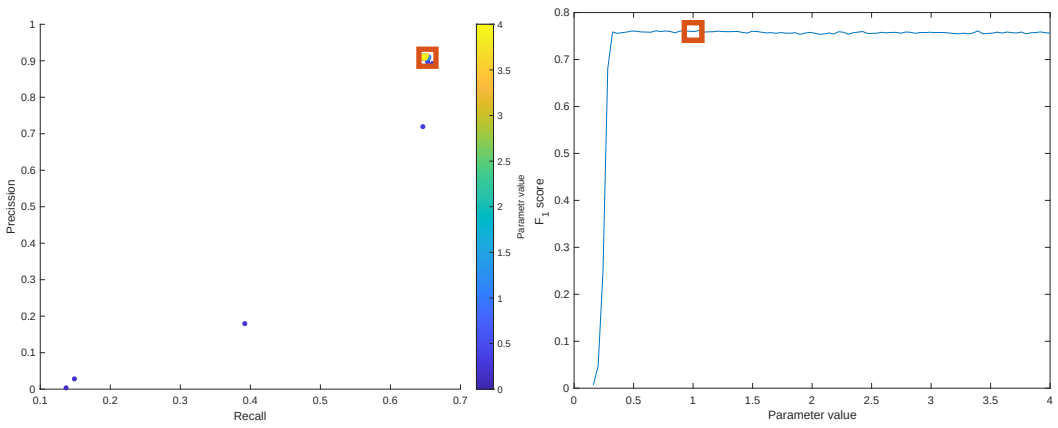
Test dataset

The validation data set contains about 200 pointclouds from several different situations. This situation includes driving in a narrow space with walls and driving in the fog with the reflections from fog visible in the pointcloud. Another situation that is present is the track with puddles of water, the track with high grass in its borers.

Section 5.2

Results

With the parameters found in **??**, we can look at our individual parameters with the help of precision recall curves and the F_1 metric. These two graphs can provide us with information on whether we could find a better solution. All graphs were generated in the test data set.

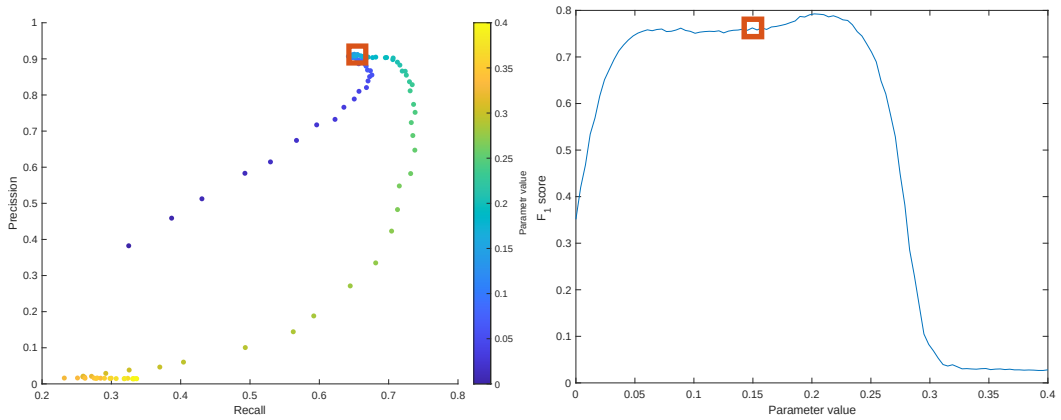


(a) Precision recall curve

(b) F_1 score graph

Fig. 21: **Air Limit** parameter precision curve around working point. Red square represents selected value.

The first parameter is **air limit** as shown in Figure 21. This parameter very quickly converges towards one place on the precision-recall curve. The same happens naturally with the value F_1. The critical value is around 0.4 m, which is similar to the height of the cones.

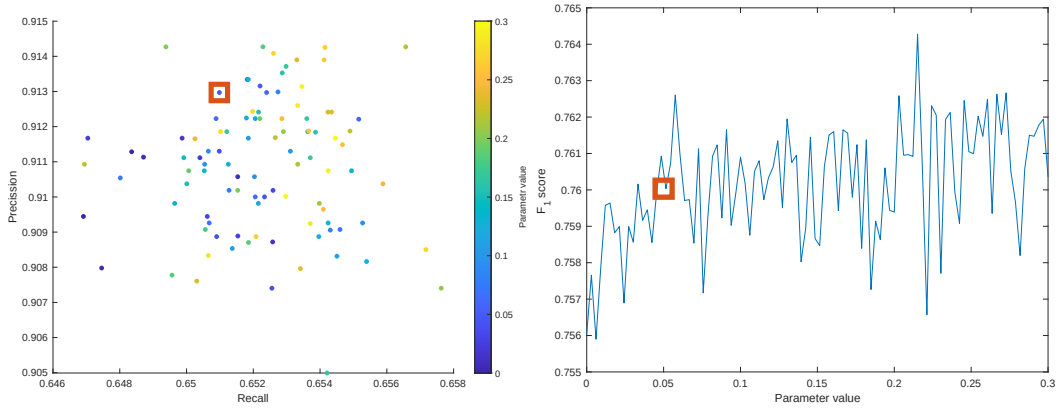


(a) Precision recall curve (b) F_1 score graph

Fig. 22: **Ground Limit** parameter precision curve around working point. Red square represents selected value.

The parameter **ground limit** is not as clear as **air limit**. This is not surprising since the ground level is much less clear than the height of the cone, which is a crucial value for **air limit**.

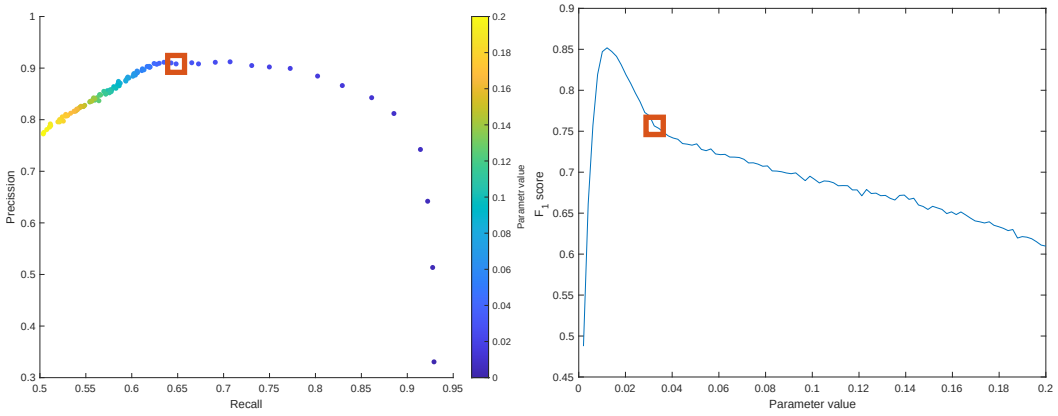
The value found is 0.15 m. Again, it corresponds well with the known facts about the cones used and their height. In Figure 22 we can see that with increasing value the recall is beginning to rise, but after some time it begins to decline quite rapidly. This is the result of no usable points in the second step of detecting the cones (division of points into three groups).



(a) Precision recall curve (b) F_1 score graph

Fig. 23: **Air exclude** parameter precision curve around working point. Red square represents selected value.

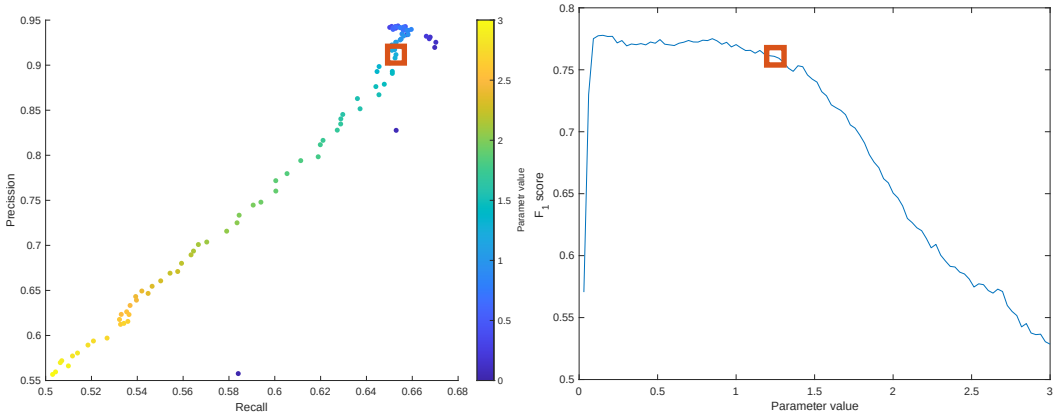
The parameter **air exclude** in the test dataset is irrelevant. This is best found in Figure 23, where all 100 tested points ended in the same area and no trend appears to emerge in the presented data.



(a) Precision recall curve (b) F_1 score graph

Fig. 24: **Ground include** parameter precision curve around working point. Red square represents selected value.

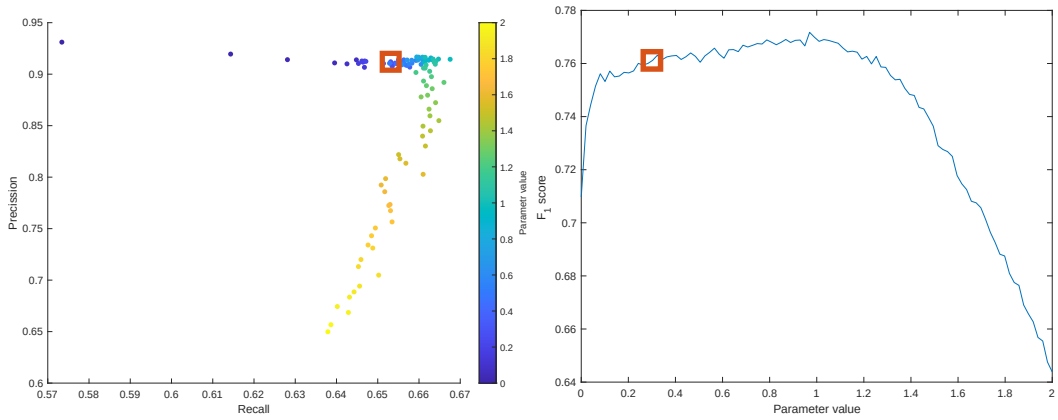
Ground include on the other hand is one of the most relevant parameters. This parameter is mainly indicative of how many points in the end we are detecting the final cone. If it is set too high, the potential for false positives (decreased precision) is evident.



(a) Precision recall curve (b) F_1 score graph

Fig. 25: **Grouping distance** parameter precision curve around working point. Red square represents selected value.

The grouping distance also found from the look at Figure 25 ended at the right spot. It is at the end of the semi-constant F_1 score for this parameter.



(a) Precision recall curve (b) F_1 score graph

Fig. 26: **Empty distance** parameter precision curve around working point. Red square represents selected value.

The empty distance graphs Figure 26 show that after brief bad values at the beginning, the recall and precision stabilise to values of about 1.2 m. At this value, both precision and recall starts do decline very steeply.

Some of these graphs suggest that we could have found a better point as an optimal value for our parameters in the detector. It can be best seen at Figure 24. This is hard to confirm with the fact that these parameters are influencing each other. Taking, for example, *ground limit* and *ground include*, if we had lowered the value of *ground limit*, then we would get much fewer points included in the step where the *ground include* parameter is used. This is also a great advantage of this detector. The uncertainty of what changes one parameter will have on the overall result. Mostly, we can make educated assumptions, and in borderline configurations we can say which values are viable and which are not. But if we are operating around some valid configuration, it is very difficult to say exactly the effect of changing any of the parameters.

Chapter 6
Conclusion

In this thesis, we discuss the design of the traffic cone detector from LiDAR data with a focus on speed. The detector is designed for use in autonomic racing cars that participate in the Formula Student competition. The detector uses facts about the specific environment in which it is used to maximise its chance of detecting traffic cones in the shortest time possible. In the theoretical section of the thesis, the conditions under which the detector is used are described. The specifics of the LiDAR sensor are also described and the advantages and limitations of these specifics are discussed there. The principles of the detector and all steps are then described and explained the reason behind each filter, which has the basis in the facts about track conditions or the LiDAR.

For the detector, we then find the optimal configuration of its parameters using the grid search method. Later, we discuss the effect of each of the parameters on the precision and recall curve.

The proposed detector has grate compatibility with the current cone detection method by neural networks that are run on GPU, and the detector from LiDAR data runs only on CPU. This balances the usage of the resource on the autonomous formula. The detector has shown promissing result and the implementation with the SLAM implemented in [11], the SLAM algorithm would be able to have greater confidence in detected cones.

References

[1] Szeles Marek. Založení studentského výzkumného týmu na vývoj autonomní elektroformule. Master's thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2020.

[2] Formula Student Germany Schulz, Elena. eforce fse.x and dv.01 at fsg 2022, Aug 2021. [Online; Accessed 7th May 2022] `https://media.formulastudent.de/2021/Hockenheim/20210820-Friday/i-rwHRf88/A`.

[3] eForce FEE Prague Formula. `https://eforce.cvut.cz`. Accessed: 2019-04-20.

[4] Huang Yong and Xue Jianru. Real-time traffic cone detection for autonomous vehicle. In *2015 34th Chinese Control Conference (CCC)*, pages 3718–3722. IEEE, 2015.

[5] Alejandro Barrera, Carlos Guindel, Jorge Beltrán, and Fernando García. Birdnet+: End-to-end 3d object detection in lidar bird's eye view. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2020.

[6] FSG: fsg.one [online]. *FSG: Handbook 2022*. Formula Student Germany, 2022. `https://www.formulastudent.de/fileadmin/user_upload/all/2022/rules/FSG22_Competition_Handbook_v1.1.pdf`.

[7] You Li and Javier Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37(4):50–61, 2020.

[8] Ouster. *OS1 Gen 1 - firmware user manual*, 12 2021. Firmvare version 2.3.x [Online; Accessed 12th May 2022] `https://data.ouster.io/downloads/software-user-manual/firmware-user-manual-v2.3.0.pdf`.

[9] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[10] Ouster. *OS1 Gen 1 (serial numbers starting with "os1-") Mid-Range High-Resolution Imaging Lidar*, 12 2021. Firmvare version 2.3.x [Online; Accessed 12th May 2022]`https://data.ouster.io/downloads/datasheets/datasheet-gen1-v2p2-os1.pdf`.

[11] Roun Tomáš. Navigační systém pro autonomní studentskou formuli. Master's thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021.

[12] Waleed Ali, Sherif Abdelkarim, Mahmoud Zidan, Mohamed Zahran, and Ahmad El Sallab. Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[13] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[14] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3266–3273. IEEE, 2018.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[16] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[17] Farzeen Zehra, Maha Javed, Darakhshan Khan, and Maria Pasha. Comparative analysis of c++ and python in terms of memory and time. 2020.

[18] Zachary Chase Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. Thresholding classifiers to maximize f1 score. *arXiv preprint arXiv:1402.1892*, 2014.

Chapter A
Attachments