Formula Student car perception with PointNet

Author
Pau Matas Albiol
pau.matas@estudiantat.upc.edu

Advisor
Javier Ruiz-Hidalgo
j.ruiz@upc.edu

Advisor
Josep R. Casas
josep.ramon.casas@upc.edu

Abstract

In this work, we present a study of the use of a 3d point cloud computer vision method as perceiver in a Formula Student (FS) car. PointNet is one of the first published neural network architectures able to work with point clouds. Here we study the performance of this algorithm introduced in the perception pipeline of a FS team. We evaluate the accuracy detecting cones in an experimental environment that reruns the inputs captured in real car runs to decide whether to include this algorithm on the car or not.

1. Introduction

In Formula Student competitions there is a modality of driverless cars. Driverless competitions require the car to identify the track limits delimited by cones and autonomously control itself to move along the track as fastest as possible.

We will focus on one task of the perception[1] pipeline. The job of this task is, given some proposal regions, to decide whether there is a cone in it or not.

In this paper, we study a computer vision method proposed by Qi *et al.*[4] that can be used to classify point clouds. This is not a trivial approach as point clouds are not ordered data and this difficults the job of neural networks (NN).

PointNet is selected because, although there are more recent proposals about the implementation of NNs over point clouds that open new lines of investigation, PointNet was the first to work directly over the dimensional space and stands as the one with the simplest architecture. For that reason we will focus on applying PointNet for the selected task of the pipeline.

In the original work, the method is built to be able to classify at point level and at cloud level, thus the model always classifies either points separately or all together as a cloud. Moreover, they implement the classification as multi-class as they work with multiclass datasets, which is a more general method. For our case, we implement a binary classifier given the binary nature of the problem to solve.

All the code developed by Qi was published in a single repository [3] using *tensorflow*; moreover, a version of the implementation in *pytorch* can be found [6]. The code we have developed to execute the experiments was heavily inspired by the second repository. The base of the perception pipeline of the FS team BCN eMotorsport, which sets the context where this task has sense; was publicly developed by A. Huguet [1].

[1]Perception is the name of the team's section responsible for identifying the track

2. Use case

It has been said that the goal of the part of the pipeline we are focusing on is to distinguish cones from proposals, which is a simplification of the process. Giving more context, the car has an integrated Light Detection and Ranging (LiDAR) sensor. This sensor emits pulses of light and measures the time it takes for them to return. These pulses are used to create point clouds that describe the environment, which the car then collects. An example oh how a point cloud would look like is the figure 1. Each point cloud is simply a set of points, each of which contains its x, y, and z coordinates, as well as the intensity of the signal.[2].

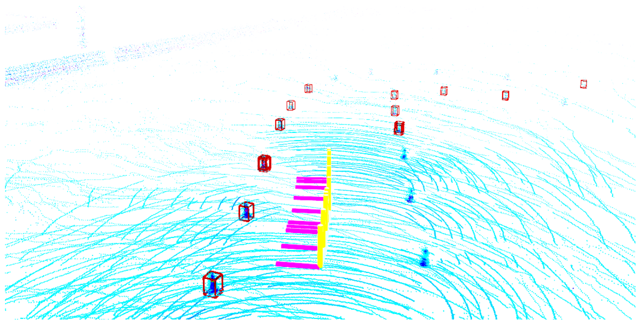


Figure 1: Example of a point cloud. It has also a representation of the car as a succesion of 3D axes and a representation of the results of the previous processes as red boxes.

[2]This data collection and structuring is done by LIMO-Velo[1].

Following the pipeline, the entire environment point cloud is processed with a clustering algorithm that generates region proposals (also called bounding boxes). These proposals are regions of the point cloud with high probability to be a real cone. Additionally, they are our input and can be seen as red boxes in the figure 1.

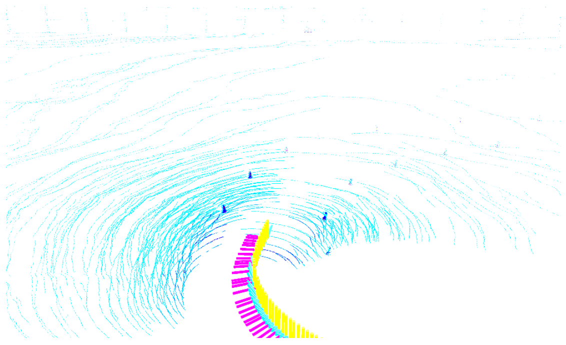
To date, the filtering of these proposals, with the aim of identifying only those that are definitely cones, has been performed using a set of convolutions and heuristics. This will be our baseline. The algorithm has been validated on the car, and its ability to detect cones is sufficient for a functional perception of the track at the end of the pipeline. The algorithm also experiences difficulties when working with runs in which the car is moving faster, resulting in fewer points per proposal. The main issue with using heuristics is the need to manually adjust and fine-tune numerous parameters to suit each specific track. In addition, making changes for one track often means that the heuristics stop working for other tracks. This lack of generalization led us to consider the use of machine learning for this part of the pipeline.

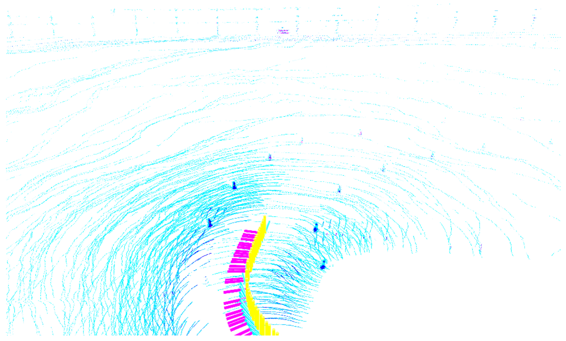
Moreover, one of the most impressive features of Huguet's LIMO-Velo is its ability to store recent LiDAR frames and, despite the LiDAR's displacement (and the car's power unit (PU) limitations), to process and work with the accumulated point clouds online. This allows us not only to propose an approach for each frame, but also to do so for a certain time interval accumulation. The resulting accumulation intervals are none other than an overlap of all the point clouds of the LiDAR frames received during the time interval and projected to the 3D space. This is beneficial for our research because the wider the time intervals are, the more resolution and points we have in our inputs, and therefore, the better our model will potentially perform. On the other hand, given the PU memory capacity, we can work with intervals of around $5s$ (which supposes ~ 400 LiDAR frames[3]) or less. This would be crucial because the environment can change significantly for different time intervals.

We can observe those differences clearly in figure 2: accumulating $5s$ of data (2c) we can see part of the track where we come from (bottom-right). However, and more importantly, it gives us more intuition of upcoming cones than the rest of images. While with $1s$ (2a) we can barely observe three cones in front of the car[4] and with an interval of $2s$ 2b, almost five of them can be intuited; with $5s$ we can observe a few more but with more points. If we had to classify as humans, we would probably choose the bigger interval but we need to explore if a neural network needs as much resolution.

[3]LIMO-Velo sends point clouds with a frequency of $40Hz$

[4]Successive car positions are represented by the 3 axes


(a) $1s$ accumulated


(b) $2s$ accumulated

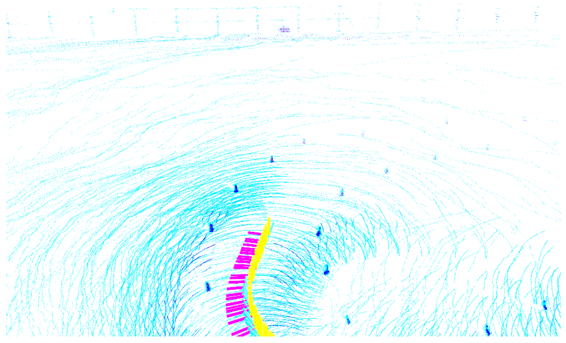

(c) $5s$ accumulated

Figure 2: Screenshots of the complete point cloud perceived with different time interval accumulations. The data comes from the same moment and zone of the track of a car real-life run which sensors measurements were saved.

3. Architecture

In this section, the network architecture (represented in figure 3) will be explained through the challenges about working with point cloud data it solves[5].

At first, point clouds are unordered, are just sets of points, therefore the model should not be affected by permutations on this set. Given this problem the solution proposed by Qi *et al.* is to use a max pooling layer. This way we can aggregate symmetrically the information of the set.

Secondly, point clouds have no orientation, hence trans-

[5]For more detailed explanation refer to the original paper [4].

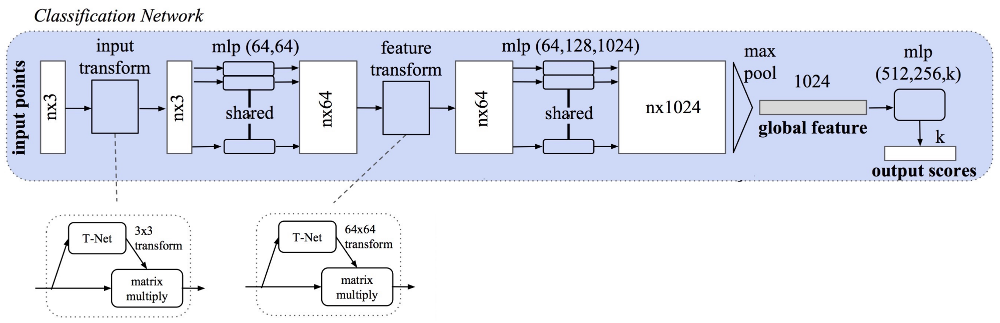


Figure 3: Block diagram of PointNet classification network.

lations or rotations to the complete set should not modify the result of its classification. To solve this challenge Qi *et al.* use a spatial transformer network (T-Net). This kind of networks transform the data into a canonical representation of it. In PointNet it is used before the processing of the input and before the processing of point features.

Then, these components are connected with multi-layer perceptrons (MLP) to finally converge in a soft-max activation function. As we are working with a binary classification, this last layer and activation function can be changed to a unique neuron and a sigmoid activation.

4. Experiments

4.1. Synthetic data

Before starting experiments with real data, we realised that the dataset ModelNet40 [5] had a *cone* class. ModelNet40 is one of the datasets used by Qi *et al.* in the original PointNet paper and has 40 different classes of different objects. The classes are distributed as shown in the table 1 and partitioned $\sim 80\%$ of the records for training and $\sim 20\%$ for testing. The records of the dataset are 3-dimensional meshes of the surface of the objects. Given a mesh and a number of desired points, it is easy to obtain a set of points from all around the surface.

Class Name	Total	Class Name	Total	Class Name	Total	Class Name	Total
airplane	726	cup	99	laptop	169	sofa	780
bathtub	156	curtain	158	mantel	384	stairs	144
bed	615	desk	286	monitor	565	stool	110
bench	193	door	129	night_stand	286	table	492
bookshelf	672	dresser	286	person	108	tent	183
bottle	435	Plower_pot	169	piano	331	toilet	444
bowl	84	Glass_box	271	plant	340	tv_stand	367
car	297	guitar	255	radio	124	vase	575
chair	989	keyboard	165	range_hood	215	wardrobe	107
cone	187	lamp	144	sink	148	xbox	123

Table 1: Table of ModelNet40 classes distribution.

With all the generated sets of points we performed data augmentation in order to simulate partial views of the objects. The data augmentation was also useful in order to create an adapted ”*cone* vs the world” dataset. Specificaly, it was used to have more samples of the *cone* class until balancing both classes.

With the adapted dataset, we trained the network and analized the results. These seemed unrealistic as the model reached $> 95\%$ training accuracy. However, when we validated the model, we found that it failed to identify validation samples of ModelNet40 cones as *cones*. It is likely that the model overfitted to the cone shapes it had seen, labeling all other shapes as *no-cone*. It should be noted that during training, these other shapes were objects such as tables, bathtubs, guitars and so on.

Overfitting always sounds bad in a machine learning context, but in this case, we are working in a simple environment. It is likely that there will be no strange objects on the track and the cones will be similar to the officially regulated ones, which could be taken as an advantage.

After the experiment, the only benefit that we could get from synthetic data in our task was to pretrain the models with synthetic data before being trained with real data. We tested models from scratch versus models with a previous pretrain with synthetic data and there was no significant difference between them. Since the cones from ModelNet40 were not the official competition cones and the *no-cone* samples were not realistic, these results had sense. For this reason, the idea of pretraining the models was abandoned.

4.2. Real data

While keeping in mind the idea of a variable sample size and time interval for the inputs of the model, we built an iterative dataset that does not save bounding boxes, but rather generates them during the process. To do this, we needed a database with all the points detected, the time at which they were detected, all the positions and orientations of the car at each time, the hand-marked locations of all the cones, and, for the *no-cones*, the locations proposed by the previous step of the pipeline[6] that did not match the hand-marked cones. This set of data would make up a *run*.

[6]In practice, the centroid of every red box of the figure 1

The selected runs have the following characteristics:

- 1st run: It was recorded in location A and the car completed two laps of the track autonomously.
- 2nd run: It was also recorded in location A, but this time the car was stopped for a minute in front of a long straight. This could be useful, as in competitions there is often some time when the car is stopped but activated detecting the track before starting to run.
- 3rd run: It is the last of the runs recorded in location A and it was run on a different track (from the one in the first run). The car was driven by a person rather than running autonomously, which makes it go considerably faster and therefore accumulate fewer points per cone and see more cones per interval. It should be noted that this run is also useful because, although the main goal of the perception pipeline is autonomous driving, it is also used for data related to non-autonomous driving.
- 4th and 5th runs: These were recorded in location B on two tracks that are very similar to each other and also to a competition track. The car was running in autonomous mode in both runs.
- Validation run: This was recorded in location C on a track that is different from the others and the car was also running in autonomous mode.

This set of runs allow to have different types of data of interest and try to generalize the cone detection. It should be noted that the different locations of the records are important because, in previous pipelines, the change in environment, asphalt conditions or floor inclination, had caused the system to fail.

Combining all the different runs we obtained 561 bounding boxes labeled as *cone* and 43 as *no-cone* that captured from all the positions were the car had been on the track became an wide dataset. The unbalance of both classes is a problem which was solved with the repetition of *no-cones* until parity. This decision was taken because generalizing *cones* is more important for the task than generalizing *no-cones*, and overfitting *no-cones* is a price we are willing to pay.

Once we had built the dataset, we faced a problem related to real data. This problem is: to decide the size of the bounding boxes. The size is the fixed number of points in every proposal point cloud (n). The value n needs to be fixed because PointNet, as many other NN, for architectural reasons can not receive inputs with a variable shapes. In reality there is no pair of bounding boxes with the same number of points, so it was needed to establish a method to obtain equally sized samples. A popular option is to up-sample or down-sample the inputs with size $\neq n$ in order to obtain equally sized inputs. The sampling method chosen for up-sampling was to repeat random points until reaching n points. This allowed to avoid creating non-existing patterns or shapes by averaging or interpolating. To down-sample, we simply selected n random points from the set.

In order to decide the exact n's to train with, we constructed the plot in figure 4. We can observe (in the heatmap) that the majority of cones, with at least one point in their corresponding bounding box, are located approximately $[15, 30]$ meters from the car along the runs. Ideally, we would like to use the average number of points in the bounding boxes ("Mean of size" in the bar plot) in this distance interval as the sample size. However, we observe that for all the time intervals, the bins corresponding to these distances are far from the highest bins. For the training, we selected a set of n values per time interval. In these sets it had to be presence of large n values according to the plot higher bins, in order to observe the behaviour with cones with a large amount of points. Moreover, there had to be a representation of popular values according to the mean of points for cones in the most popular distances. We have included some intermediate values, too.

5. Results

Below there is the table of results (table 2) with the heuristics algorithm as a baseline. It should be noted that the statistics for the models are based on an unseen run, but the heuristics were manually fine-tuned based on the results of all the available runs.

delta[7] (s)	n[8]	accuracy	F1-score	precision	recall
(baseline)		0.665	0.834	0.723	0.988
1	**10**	**0.734**	**0.846**	**0.734**	**1**
	50	0.598	0.748	0.599	0.997
	150	0.520	0.684	0.566	0.866
2	10	0.584	0.737	0.651	0.851
	25	**0.721**	**0.838**	**0.721**	**1**
	100	0.673	0.804	0.706	0.934
	250	0.526	0.689	0.584	0.840
5	10	0.582	0.736	0.635	0.876
	25	**0.708**	**0.829**	**0.751**	**0.925**
	100	0.634	0.776	0.634	1
	300	0.641	0.781	0.641	1

Table 2: Table of results

If we focus on the baseline before starting the analysis of the results. It is shown that the baseline performs quite well, correctly identifying almost all the real cones but generating an excessive number of false positives (FP). We can

[7]Temporal interval
[8]Sample size

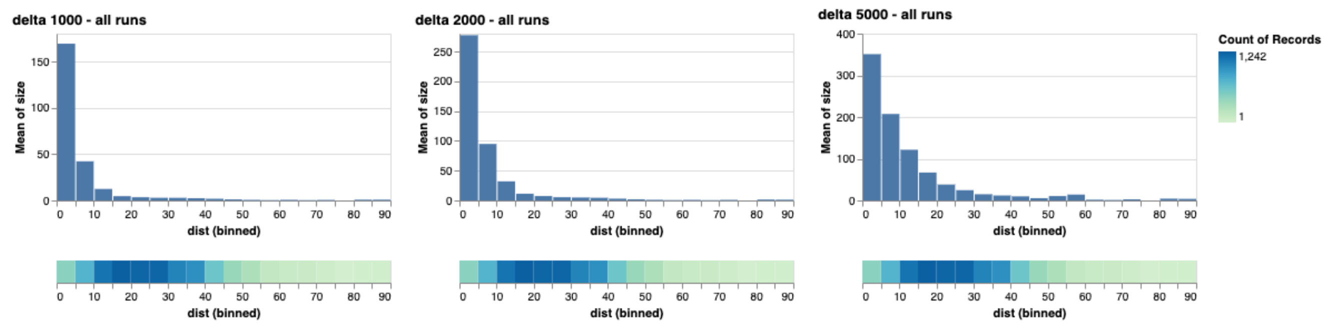


Figure 4: Plots made to decide the sample sizes.

extract these conclusions with the precision and the recall. Having a high recall, as 0.988, the baseline detects almost all the real positives. Otherwise, the precision remains at 0.723, a less remarkable proportion for the real cones detected over the cones predicted. These results characterize an algorithm than prioritizes less false negatives (FN) than less false positives in the precision-recall trade-off.

Also before starting the analysis, it has to be said that we will focus on the F1-score for different reasons. The first reason is the different importance of both classes of our problem. Positives (*cones*) are more important to be detected than negatives (*no-cones*). Thus, if we focus in the accuracy we are taking into account true positives and true negatives with the same importance, which is not the desired approach. On the other hand, it has already been mentioned that the precision, the recall and their trade-off are relevant for the problem, because they allow us to know the proportion of TPs over predicted positives, the proportion TPs over real positives and their relation, respectively. Furthermore, with the F1-score we are analyzing the harmonic mean of the precision and the recall, a combination that we will take as our reference metric.

First of all, we can observe that models with a lower sample size (n) tend to perform better, based on the F1-score. The runs with $n = 10$ and delta $= 1s$, $n = 25$ and delta $= 2s$ and $n = 25$ and delta $= 2s$ are clear examples of low n with the best results within the others with the same delta value. This is not surprising, as the bins in the figure 4 corresponding to the most populated distance ranges had a lower mean size for the bounding boxes.

However, the results are not significantly better than the baseline. Also, the same pattern of minimum FNs and more FPs of the baseline is observed from the precision-recall trade-off in all the experiments. Nevertheless, these results encourage us to continue investigating, especially because the baseline results are hand-tuned. This could be a sign of better generalization by the model. In addition, the baseline was able to detect almost all the cones but generated many false positives, while our model, although it does not always detect all the cones, has fewer false positives. This, combined with an improvement in the clustering algorithm, could improve the results.

It should be noted that the main challenge in training the PointNet neural network is to achieve a good T-Net that can create canonical versions of each input. Without this, it is difficult to achieve the desired results. This may have happened to us because working with the mentioned iterative dataset, the construction of data-label pairs is computationally and temporally expensive. Each epoch of training takes about 4 hours to run on our environment[9].

As mentioned, we discovered that the sample size is very important; the closer it is to the average size of the majority of bounding boxes, the better the model performs. This, combined with the nature of our problem (as shown in figure 4), which has a similar average number of points per bounding box in the distance ranges where more cones are visible, makes the width of the temporal interval less relevant than expected.

The code developed to obtain these results can be found in a GitHub public repository [2].

6. Conclussion

The goal of this paper was to determine whether machine learning could be a viable approach for the problem we are facing, and after reviewing the results, we believe it has the potential to improve upon the current system. While there is no guarantee that it will work better than the current system, the possibility of better generalization compared to the baseline is a significant improvement for the team. This would eliminate the need for a person with extensive knowledge of the system and the implications of modifying the many parameters of the heuristics (baseline). Such is a significant advantage for the team and the first step to make the pipeline machine learning based.

[9]The environment has: a Intel Xeon 2.1GHz (x86_64) CPU with 32 cores and 256GB of RAM; and eight GeForce RTX 2080 Ti GPUs with 4352 cores and 11GB of RAM.

In the future, we plan to adapt this synthetic laboratory environment to the pipeline and create software ready to be used by the car in real-time. This would allow to test the car on track (a fresh new track) and determine which of both algorithms performs better.

Also in the future, it would be interesting to try adding intensity as an input, as suggested by Qi *et al.* and other subsequent papers, as it has the potential to improve performance.

References

[1] Andreu Huguet. Limo-velo. `https://github.com/Huguet57/LIMO-Velo`, 2022.

[2] Pau Matas. Pointnet-formulastudent. `https://github.com/PauMatas/PointNet-FormulaStudent-I2R`, 2023.

[3] Charles R. Qi. Pointnet. `https://github.com/charlesq34/pointnet`, 2016.

[4] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[5] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2014.

[6] Fei Xia. pointnet.pytorch. `https://github.com/fxia22/pointnet.pytorch/`, 2017.