Florian Müller
Hands on Deep Learning for Computer Vision

Extending PointNet

Subtask Control

1. Introduction

1.1. Goal of this project

The goal of this project is to control a car in the CARLA driving simulator using a deep learning model trained on point clouds. Given a point cloud of a driving situation, the model should predict the correct steering angle. The steering angle is represented by a relative number between -1 and 1, where -1 corresponds to the leftmost and 1 to the rightmost possible steering angle. The project is based on the PointNet model [1], which in its original form is used for classification of objects and segmentation of scenes represented by point clouds.

1.2. Point clouds

The data structures used to represent steering situations in this project are point clouds. A point cloud represents thee dimensional objects as an unordered set of points. Each point is represented by a vector of its coordinates. While this representation is more memory efficient than ordered data structures like 3D voxel grids, the fact that the points are unordered poses a challenge. A point cloud containing n points can have $n!$ different permutations representing the same object. Converting the point clouds to 3D voxel grid or similar data structures, results in unnecessarily voluminous data, since most of the points in the voxel grid will be empty. This project is therefore based on an implementation of the PointNet paper [1] to work with raw point clouds.

Furthermore the focus of this project lays in dense point clouds, generated from depth maps. In comparison to LiDAR point clouds, they have the advantage of low costs of generation, since they don't require expensive hardware, but can make use of simple cameras. The point clouds used in this projects are extracted from depth maps provided by the CARLA simulator, an open-source simulator for autonomous driving research [2]. A comparison between LiDAR and dense point clouds can be found in section 7.1 *LiDAR vs. dense point cloud*.

1.3. PointNet

PointNet is the name of a neural network architecture, designed by researchers from Stanford University and published in a paper released in April 2017. In its original form it consumes point clouds as input and outputs k scores of the k candidate classes for a classification task or $n \times m$ scores for each of the n points and m semantic subcategories for a segmentation task. For this project I will focus on the classification model and modify it in order to reach the goal of predicting steering values. The PointNet architecture is designed to make the model invariant to permutations as well as rotations and transformations of the input. Invariance to permutations is achieved by using a max-pooling layer in the network architecture as a symmetric function, which returns a global feature vector. Invariance to rotations and transformations is achieved by a joint alignment network, called T-net (see. figure 1). More detailed explanations of the PointNet architecture can be found in the PointNet paper [1].

1.4. Project environment

This project is based on a PyTorch implementation PointNet [3]. Training data is generated using the CARLA driving simulator [2]. The training graphs are created using Visdom [4].

1.5 Dataset

The dataset provided for the project is generated using the CARLA driving simulator. It initially contained two episodes with 1000 frames per episode. New episodes were generated later. For each frame there is a dense local point cloud, a first-person-view RGB image, a semantic segmentation of the image and driving data such as the steering angle. To train the model, the point cloud is used as

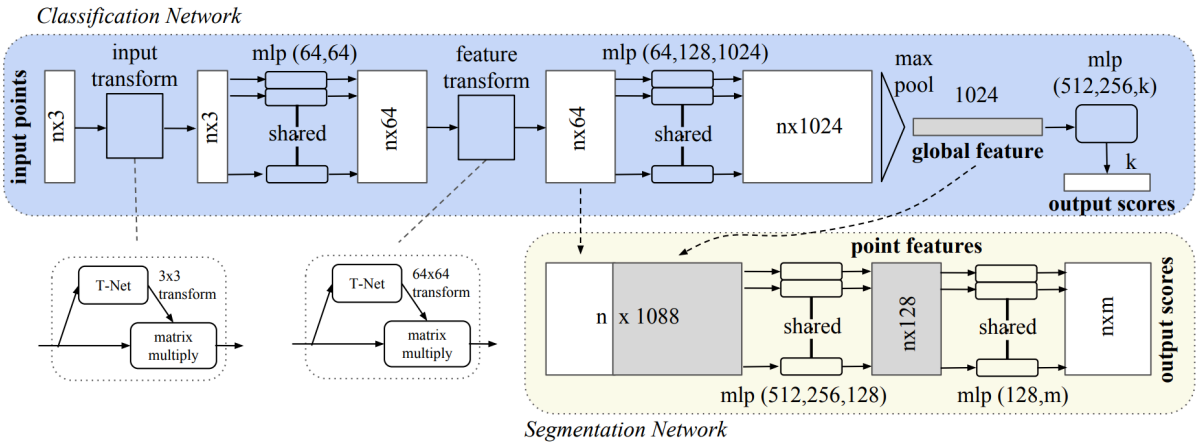

Figure 1. Point Net architecture

input, while the steering value is used as label for the regression task.

All point clouds used in this dataset are "local", which means they are all centred around the same position. The dataset contains different road course situations, which are briefly outlined in the following section.

1.6. Important terms

Some terms will be used repeatedly in the course of this document. Do avoid misunderstandings, I will briefly outline the most important ones in this section.

T-junction describes road junctions with two possible paths "left" and "right". An example can be found in figure 2.

Side intersections are road courses with two possible paths "straight" and either "left" or "right". An example can be found in figure 3.

Natural turn describes road turns with only one possible path. An example can be found in figure 4.

Other possible road courses are *straight roads* (figure 5) and 3-way.-junctions (not included in the CARLA maps used in this project).

Offline testing refers to evaluating the model's performance using pre-generated point clouds and ground truth steering data from the dataset.

Figure 2. T-junction

Figure 3. Side intersection

Figure 4. Natural turn

Figure 5. Straight road

Online testing refers to testing the model in the CARLA simulator to evaluate its practical performance.

CARLA dataset generally refers to a dataset generated using the CARLA driving simulator as described in the previous section.

2. Modifying PointNet for control task

The original PointNet implementation is designed for a classification task. Given a point

cloud, a vector of size k is returned, representing the probabilities of the point cloud belonging to one of the k classes. To make the PointNet architecture suitable for the regression task of predicting a steering angle, I modified the network architecture to output a single value and appended a hyperbolic tangent function to scale the output to a value between -1 and 1.

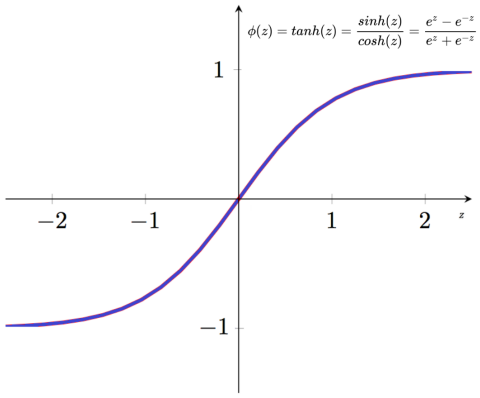

Figure 6. Hyperbolic tangent

Additionally I modified the loss function from negative log likelihood (NLL) to mean squared error (MSE), which is more suitable for the regression task.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE quantifies the prediction error by calculating the average squared difference of the predictions to the actual values. It puts an over-proportionately higher penalty on large differences than on small differences, which is desired for this project.

3. Initial challenges

In this section I will outline the challenges I faced during the initial experiments. Results of the experiments are described in the next section.

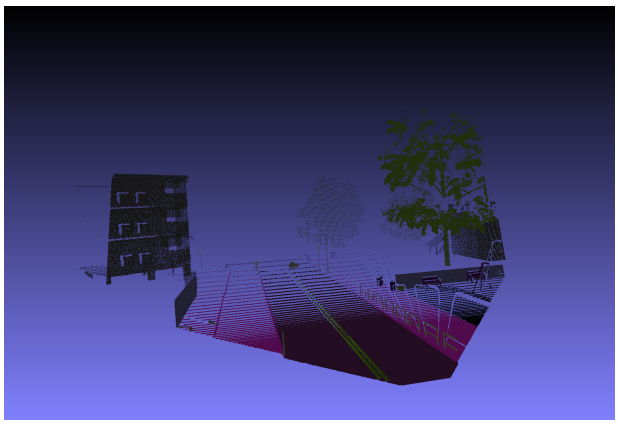

Figure 7. Full dense point cloud augmented with semantic segmentation.

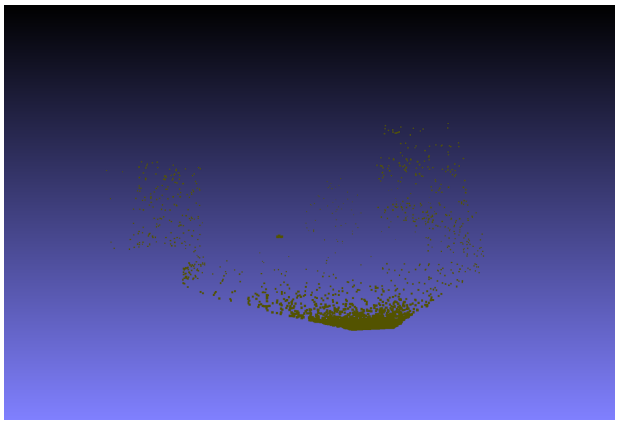

Figure 8. Point cloud downsampled to 2500 points

3.1. Large point clouds

The first challenge was the point clouds' size. Initially I tried to use the full dense point clouds for the training, but this quickly resulted in GPU memory issues, even with batch sizes as low as four. Each dense point cloud contains around 190.000 points, with most of the points concentrated very close to the camera. The points very close to the camera don't provide much information about the correct steering angle though, since they don't really influence the steering decision and look very similar in most of the situations.

In order to increase the batch size, I tried to randomly sample 2.500 points from the point clouds. This solved the memory issues, but as the experiments in the next section show, the model's performance in predicting steering values was rather underwhelming. A major downside of randomly downsampling the point

clouds is that many relevant structures are lost. While it is easy for a human to detect the road in the dense point cloud, it is hard to do so in a point cloud containing 2.500 randomly selected points (see figure 8).

In order to reduce the number of points while keeping relevant information, I manually extracted relevant points from the point clouds using the semantic segmentation included in the CARLA dataset. By projecting the segmentation image onto the point cloud, each point received a segmentation class. This class could be used in the next step to separately extract points belonging to the road, the sidewalk and the road lines. The new point clouds which only contained road points still had 140.000 points, but could be downsampled to 5.000 points without loosing much structure.

To further improve the performance, I processed the segmentation images using a simple sobel operator. From the resulting image, I extracted the road edges and road lines and used them to extract the corresponding points from the point clouds (see figure 9). The resulting point clouds now had around 8.000 points, so it wasn't even necessary to sample them down anymore. I will reference to the point clouds resulting from this kind of processing as *edge-processed point clouds* for the curse of this document.

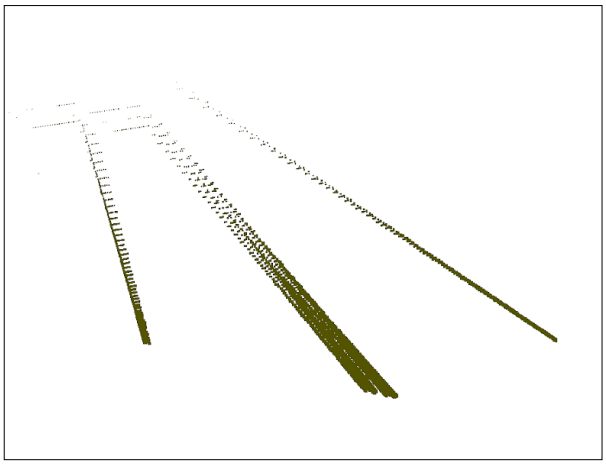

Figure 9. Edge-processed point clouds: road edges and road lines extracted from a dense point cloud

3.2. No smooth steering

Since the training data is generated using CARLA's built-in autopilot, the steering is rather abrupt and sharp than smooth (see figure 10). As a result the ground truth steering values of two frames following each other might differ significantly, which makes it harder for the model to predict the correct value. In order to overcome this issue, I experimented with smoothing the ground truth steering values using a sliding window of different widths and settled on a sliding window of 13 frames. As it turned out later, this approach didn't lead to a good performance during testing online in the simulator. More on this in section 6, *Testing in the CARLA simulator.*

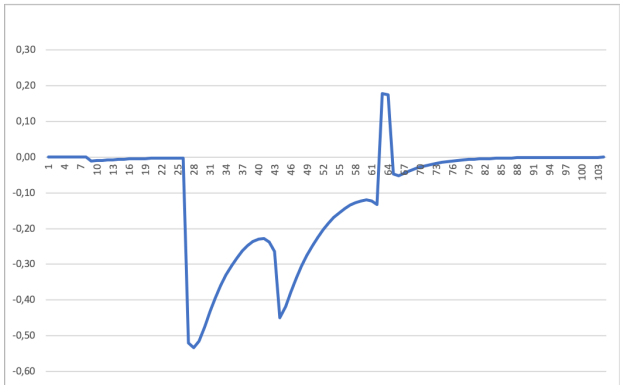

Figure 10. Steering values of one steering situation. Instead of being a smooth steering process, the autopilot constantly oversteers and compensates.

3.3. Straight steering bias

The initially provided dataset consisted of two episodes, each of which contained three steering situations. The remaining frames were straight roads. In total only 8% of the training examples had a steering magnitude greater than 0.05. For the first experiments I used a random sampler. This resulted in training the model on around 92% straight road situations. By predicting straight steering all the time, the model would have already scored an accuracy of 92% on this dataset.

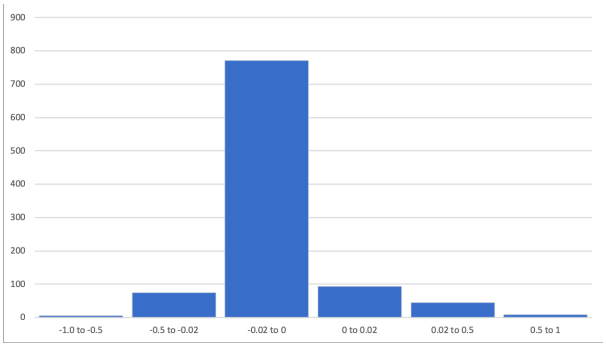

Figure 11. Distribution of steering values in the initial dataset. Vertical axis: number of frames, horizontal axis: steering value. The dataset contained 1000 frames.

To overcome this issue, I implemented a balanced sampler. The sampler sorts the frames in four different bins depending on their steering value - left (-1 to -0.05), slight left (-0.05 to 0), slight right (0 to 0.05) and right (0.05 to 1). For each bin, the same number of examples per bin is returned. But this restricts the batch size to be a multiple of four, as well as providing little flexibility in the number of bins or the weight per bin.

Later I implemented a slightly different version of the balanced sampler - a weighted sampler. This new sampler first calculates the sizes of the different bins. It then sets a weight per training example, which corresponds to the desired percentage of the bin divided by the bin's size.

Each element's weight is then used as the probability of choosing this element in the sampling process. This new implementation of a balanced sampler has more flexibility regarding number of bins and batch sizes as well as allowing to set custom weights for each bin.

4. Initial experimental results

4.1. Measuring performance

Initially I measured performance as percentage of predictions which differed less than 5% from the ground truth steering values. However, this measurement turned out to be of little practical use, as it didn't reflect the actual testing performance in the simulator. A model which is always 4% off on straight roads will soon steer the car into the crash barrier despite a great theoretical accuracy, while a different model can be more than 5% off when taking a different path in a curve, yet staying on the road and showing a great practical performance. I therefore decided to stop measuring performance using the 5% rule and instead focussed on practical experiments in the simulator to estimate the models' performance.

4.2. Training with downsampled point clouds

Figures 12 and 13 show the predictions after training the model on point clouds downsampled from 190.000 to 2.500 and 5.000 points respectively, while using a random sampler and the original non-smoothed steering values.

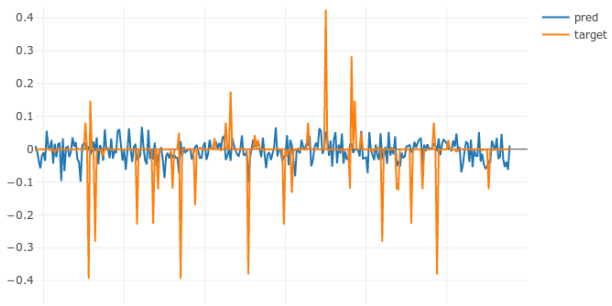

Figure 12. Offline test result of a model trained with 2.500 points per cloud for 50 epochs. Horizontal axis: number of frame, vertical axis: steering value. Most predictions lay in the range from -0.1 to 0.1, frames with higher steering angles are not predicted properly yet.

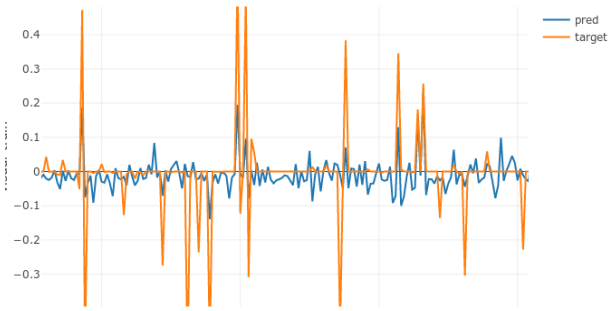

Figure 13. Offline test result of a model trained with 5000 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Frames with high steering value are predicted better than previously, but still not a great performance.

The orange line shows to ground truth steering values while the blue line shows the predictions.

4.3. Training with extracted points

The following graphs (figure 14 to 19) show the results of the second round of experiments, using extracted points and the first balanced sampler as described in section 3.3 *Straight steering bias.*

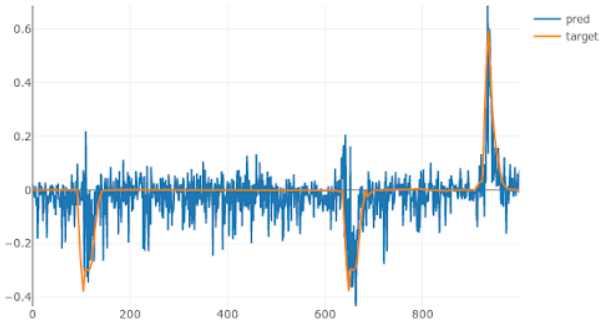

Figure 14. Offline test result of model trained on dataset with extracted road points using 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. The blue prediction line line roughly follows the orange target line in situation of curves, but fails to do so in situations with straight roads.

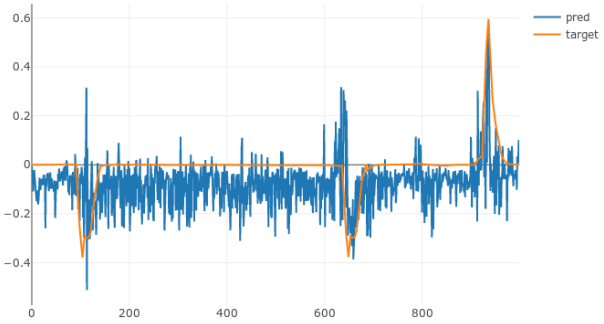

Figure 15. Offline test result of model trained on dataset with extracted road points using 20.000 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Increasing the number of points doesn't increase the performance of predictions.

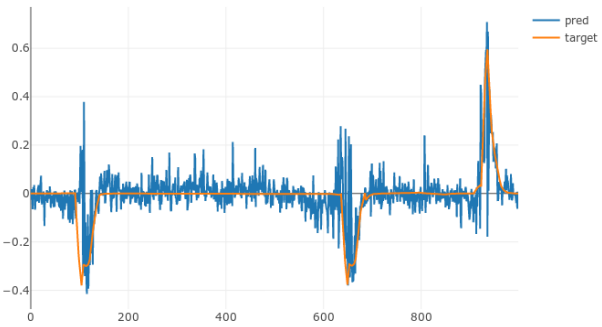

Figure 16. Offline test result of model trained on dataset with extracted sidewalk points using 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Using only sidewalk points slightly improves the performance in comparison to using only road points.

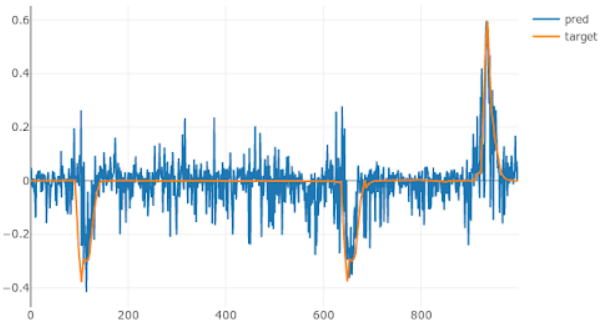

Figure 17. Offline test result of model trained on dataset with extracted road, sidewalk and road line points using 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. It performs worse than the previous model which only used sidewalk points.

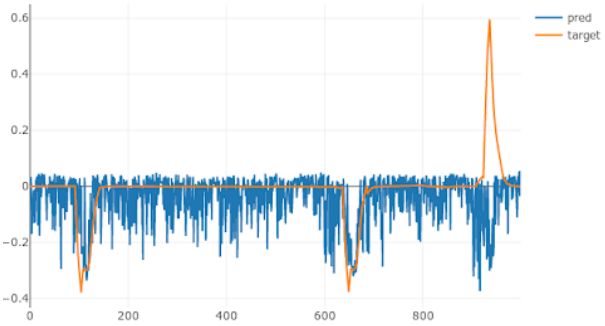

Figure 18. Offline test result of model trained on dataset with extracted road points and only left turning situations using 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Straight roads are predicted worse than before, right turn is predicted as left turn.

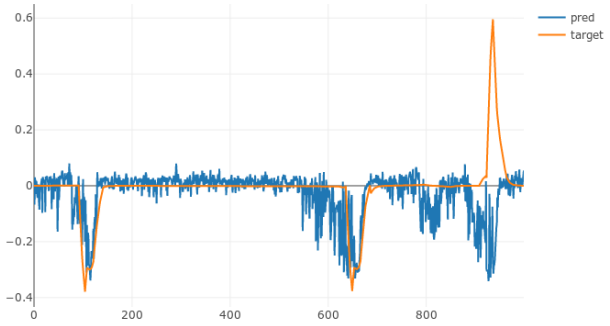

Figure 19. Offline test result of model trained on dataset with extracted points of road edges and road lines and only left turning situations using 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Performance on straight roads is much better than before, turns are clearly distinguishable from straight roads.

4.4. Training with random sampler

The experiments shown in the previous section showed good performance in the prediction of curves, but bad performance in the prediction of straight roads. Originally the problem had been that curves were not predicted properly, which was blamed on the straight road bias. But now that straight roads were predicted worse than curves, I experimented with the default random sampler again, which doesn't put weight on any specific kind of situation but returns examples in the same distribution in which they occur in the dataset. Results are shown in figure 20 and figure 21.

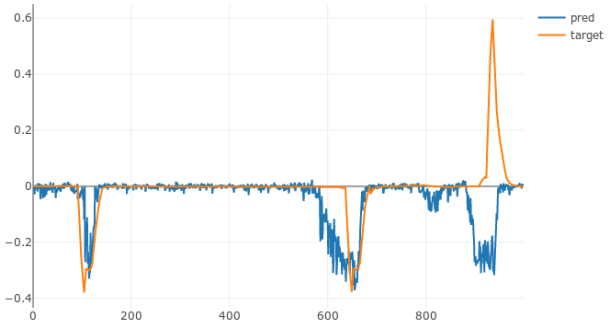

Figure 20. Offline test result of model trained using random sampler on dataset with extracted points of road edges and road lines and only left turning situations using 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. The prediction line now follows the target line much more closely than before.

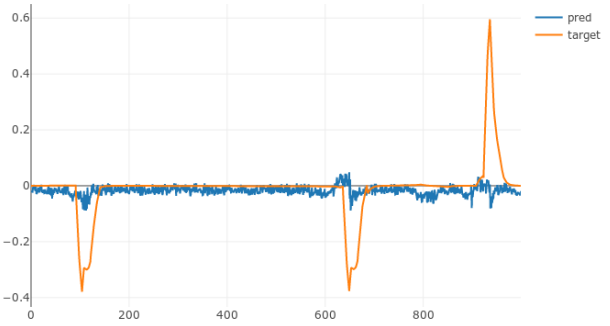

Figure 21. Offline test result of model trained with random sampler on dataset with extracted points of road edges and road lines using left and right turning situations and 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. When reintroducing right turns to the dataset, the straight steering bias is back and turns are not properly predicted anymore.

4.5. Training with weighted sampler

To overcome the straight steering bias while keeping the good performance on straight roads, I implemented a different version of the balanced sampler, as described in section 3.3, straight steering bias. Figure 22 shows the experimental result. The model was trained on left and right turns, but failed to predict the right turn of the test set.

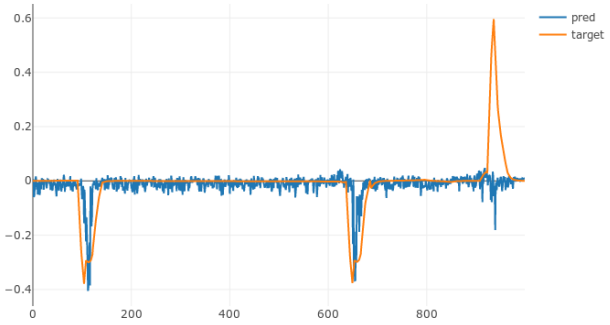

Figure 22. Offline test result of model trained with new weighted sampler on dataset with extracted points of road edges and road lines using left and right turning situations and 2.500 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Left turns and straight roads are predicted quite well, but the model fails to predict the right turn of the test set, even though right turns were included in the training set.

4.6. Training with natural turns

To evaluate the how much influence uncertain steering situations like T-junctions have on the prediction performance, I retrained a model on a new dataset, containing 10 episodes of natural turns, with 100 frames per episode. As before, I preprocessed the point clouds by extracting the road edges and lines and used the weighted sampler during the training process. Figure 23 shows the testing result.

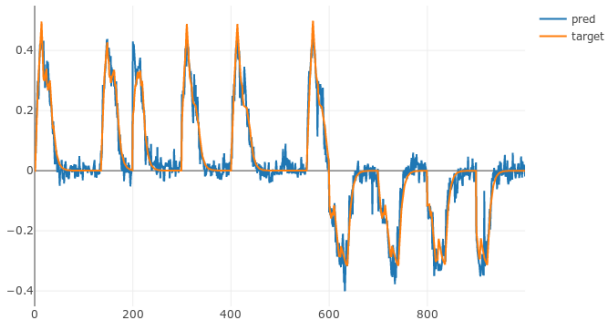

Figure 23. Offline test result of model trained on natural turns using 1.000 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Without situations with two possible paths (T-junctions), the model is able to predict left and right turns quite well.

5. The steering indicator

5.1. Concept

The experiments showed that the model's prediction performance is much lower, when there are two possible paths, like at T-junctions. To overcome this problem, I introduced a steering indicator to the PointNet architecture. The steering indicator consists of two bits and is passed to the model as additional input beside the point cloud. It is directly appended to the global feature vector, which therefore grows to a size of 1026. When the model is supposed to go left at the next junction, it is set to "10". When the model is supposed to go right at the next junction, it is set to "01".

To make the model ready for three-way-junctions (not included in the CARLA maps used in this project, but included in other maps), I later added "00" as a third option to indicate the intent to drive straight.

In frames in which it is not possible to turn left or right, the model is supposed to follow the road, regardless of the steering indicators value. This is important to note. The steering indicator doesn't tell the model when or how much to steer, it is just an indication in which direction to steer when there is more than one possible path.

In favour of readability, I will refer to the steering indicator "10" as *left*, to "01" as *right* and to "00" as *straight* for the course of this document.

To set the steering indicator during training, I used three different approaches. For datasets with a single turn situation per episode, I added up all the steering values of the respective episode. If the sum was negative, it was a left turn and the steering indicator for the whole episode was set to *left*, otherwise to *right*. For episodes with multiple turn situations, the approach was slightly different. I looped through the episode from behind and whenever the steering value fell below the threshold of -0.05, I set the steering indicator to *left* for all frames until the steering value surpassed a threshold of 0.05. Then I toggled the steering indicator to *right* until it fell below -0.05 again and so on. The resulting steering indicator pattern can be seen in figure 24.

Later, when I started training on situations with side-intersections, the approach of summing up the steering values was not ideal anymore, since the sum still might be slightly positive or negative when driving straight. Therefore, I started saving the steering indicator when generating data and used the saved indicator during training.

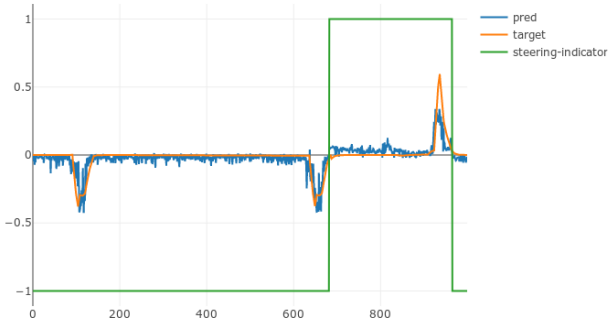

Figure 24. Offline test result of model trained with steering indicator (green line) on dataset containing left and right turns including T-junctions. Other settings: weighted sampler, 1000 points per cloud, 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Im comparison to figure 22, there is an improvement in the prediction of the right turn.

5.2. Experimental results

Introducing the steering indicator resulted in an improved performance in previously uncertain situations, like T-junctions. At the same time the performance for straight roads stayed equally well. Testing graphs are shown in figure 24 and figure 25.

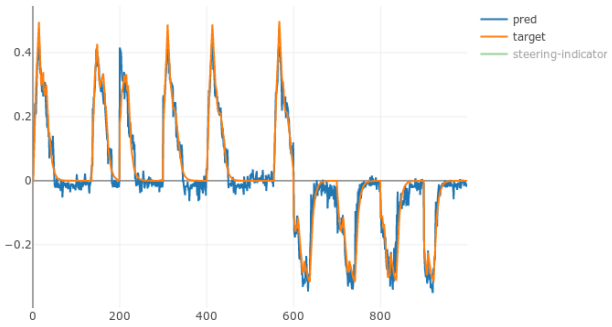

Figure 25. Offline test result of model trained with steering indicator on dataset containing left and right natural turns. Other settings: weighted sampler, 1000 points per cloud, 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. Im comparison to figure 23, the performance increased slightly, as there are less outliers.

6. Testing in the CARLA simulator

6.1. Setting up

After introducing the steering indicator, the offline testing results looked good enough to be tested in the more realistic environment of the CARLA simulator. The CARLA simulator provides a python api, which enables developers to programatically control the car. Data like a depth map, RGB images and semantic segmentation is provided with each frame. Dense point clouds can be generated using the depth map. Using the semantic segmentation, road edges and lines can be extracted from the point cloud on-the-fly. The processed point cloud is then passed to the trained model, which returns the predicted steering angle. The steering angle is passed back to the simulator, which steers the car accordingly and proceeds to the next frame. Generating and processing the point cloud as well as predicting the steering angle takes less 5 milliseconds, which makes it possible to do it on-the-fly.

6.2. Results of online testing on natural turns

Initial results from testing the model on natural turns revealed, that the model started steering way too early and way too long. This was caused by the approach of smoothing the ground truth steering data in order to compensate the sharp steering caused by data generation in the simulator. But even after retraining the model on the original steering data without smoothing, the model's performance in the simulation was still underwhelming, despite good performance during offline testing. After further investigation I realised that the point clouds were augmented by random noise and rotations during the training process. In the simulation this wasn't the case. The point clouds therefore had a different structure, which caused the bad performance. By disabling data augmentation during training, the model was finally able to properly predict the steering angle in natural turns in the simulator.

Another problem revealed from testing in the simulator was that the model constantly steered slightly to the left on straight roads when the left steering indicator was set and slightly to the right when the right steering indicator was set.

Additionally, when the car got off the perfect path, it suddenly failed to predict the correct steering entirely. In order to overcome those issues, I generated data off the perfect path. This was accomplished by letting the built-in CARLA autopilot drive through the simulation while adding a random number between -0.3 and 0.3 to the steering value intended by the autopilot. This caused the car to drive in wavy lines and the autopilot to try to compensate and to keep the car on the road. For each frame I captured the edge-processed point cloud as well as the original steering value intended by the autopilot before adding the random number.

By training on this data, the model got more confident in situations off the perfect path and was able to bring the car back to the centre of the road when steering too far after a turn. It also resolved the issue of the model constantly steering slightly left/right on straight roads. Whenever the car got too far off the centre of the lane, the model was able to correct its path.

6.3. Results of online testing on T-junctions

While the model showed good performance on natural turns after retraining with data including wavy lines, the performance on T-junctions was still very poor. Even though offline testing plots suggested excellent steering predictions (see figure 24), the model was indecisive in which direction to steer during online testing and crashed into the barrier after driving in a few wavy lines.

The problem was partially solved by generating more training data including wavy lines with multiple levels of randomness. Using the built-in CARLA autopilot and the previously described sampling technique, I generated training data of multiple T-junction situations, each with both possible directions (left/right) and 6 different levels of randomness, ranging from 0.1 to 0.6. After training the model for 500 epochs on this data, it was able to confidently steer the car around the T-junction turns in the direction indicated by the steering indicator.

6.4. Feature transform

PointNet is designed to be invariant with regard to rotations of the input. This makes perfect sense for classification and segmentation tasks, since object remain members of the same class after being rotated. For the regression task of predicting the steering angle from a given point cloud, this invariance with regard to rotation could be problematic. When rotating the point cloud of a right turn by 180 degrees, it becomes very similar to a left turn (ct. figure 26).

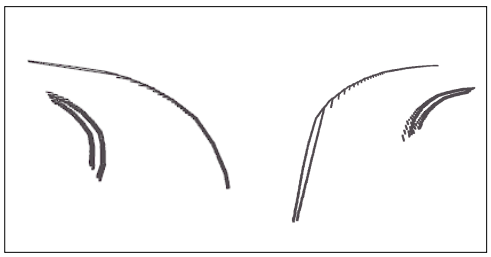

Figure 26. Left: point cloud of left curve. Right: same point cloud, rotated by 180 degrees. After the rotation, the left curve looks a lot like a right curve.

To evaluate the influence of the rotation invariance on the model's performance, I trained multiple models with the same settings as previously, but deactivated the part of PointNet responsible for rotation invariance. (the second T-net in the PointNet architecture, figure 1). The experiments showed no improvement in performance for most of the models and even lower performance for a few of them. Therefore I discarded my assumption and decided to continue with activated rotation invariance.

6.5. Combining the models

So far I was training separate models for natural turns and t-junctions. In order to obtain a more general model, capable of steering the car in all situations of the simulator, I generated data from all possible situations of

CARLA's Town01 map (T-junction, side-intersection, natural turn, straight road), each with all possible steering indicators and multiple levels of randomness.

The model trained on this huge dataset for 500 epochs performed equally well as the previous separate models on natural turns and T-junctions but almost always failed to predict the steering values of side-intersections.

While investigating the issue, I found a bug in my implementation - the car's camera angle used for testing was slightly different from the camera angle used to sample training data. After fixing this bug by changing the testing car's camera angle, the model was able to predict the steering values for side intersections and also improved its performance in the other situations, noticeable by much smoother turns.

6.6. Live setting of the steering indicator

As a little gimmick, I added the possibility to set the steering indicator using the keyboard while testing the model's steering abilities. This enables the user to dynamically change the desired route while testing the model.

7. Towards an unsupervised model

7.1. LiDAR vs. dense point cloud

The final model achieved good performance on all situations available in CARLA Town01 and Town02. But it required the point clouds to be preprocessed by extracting road edges and lines, using segmentation data not available in real world scenarios. To avoid having to preprocess the point clouds, I experimented with a different kind of point cloud with a lower density by design - the point clouds generated by a LiDAR sensor.

In reality LiDAR sensors have the disadvantage of being more expensive than the hardware needed to generate dense point clouds using simple cameras, although the price of LiDAR sensors has decreased in the recent past. Advantages include more evenly distributed points and all-round vision. Using the CARLA simulator, it's possible to attach a LiDAR sensor to the car. The LiDAR sensor can be configured with regards to numbers of channels and sampling rate.

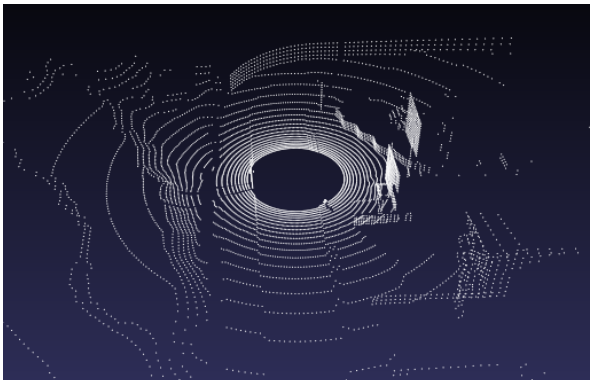

Figure 27. Point cloud generated by CARLA LiDAR sensor with 32 channels and 100.000 points per

To compare the performance of a model trained on LiDAR point clouds with the model trained on preprocessed dense point clouds, I generated a new dataset containing examples of all situations available in Town01 (T-junction, side intersection, natural turn, straight road) using the CARLA's built-in LiDAR sensor with 32 channels and a sampling rate of 100.000 points per second. The resulting point clouds contained around 8000 points, since the simulator captures 12 frames per second.

At first I trained a model on the whole dataset, using the full point clouds containing 8000 points per cloud. During testing in the simulator, I used the same LiDAR sensor settings as during data generation - 100.000 points per seconds with 32 channels. The performance in Town01 was very good from the start. Without any preprocessing, the model was able to predict correct steering angles for natural turns, T-junctions, side intersections and straight roads. In Town02, the model was able to correctly predict steering angels for most situations (natural turns, straight roads and T-junctions), but it struggled with side intersections. When the car was approaching a side intersection, the model steered slightly

towards the turn but then aborted the manoeuvre and counter steered to either continue straight or crash into the barrier.

I then trained a second model using the same dataset as before, but this time only sampling 1000 points per point cloud during training. For testing, I first used the same sensor settings as before (sampling rate of 100.000 points per second with 32 channels). Like in the previous experiment, the model was able to predict correct steering values for all situations of Town01. Then I lowered the sampling rate during testing to 4.000 points per second. Now each point cloud contained only around 300 points, but the model was still able to predict the correct steering values for all situations in Town01. Only after lowering the sampling rate again to as low as 1000 points per second, each point cloud now only containing 80 points, the model wasn't able to predict the correct steering values anymore.

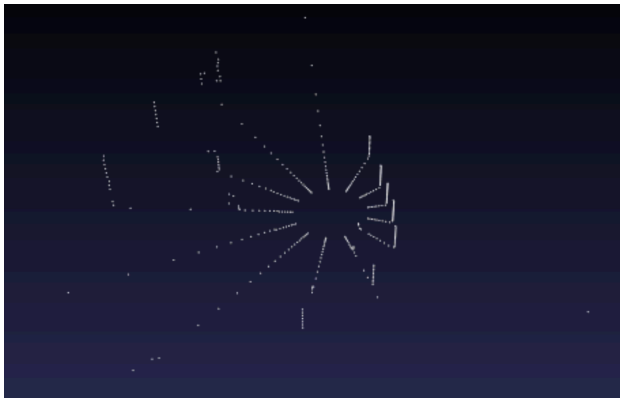

Figure 28. Point cloud generated by CARLA LiDAR sensor with 32 channels and 4.000 points per second.

In Town02 the second LiDAR model's performance was similar to the first one's. It was able to handle natural turns, straight roads and T-junctions, but wasn't able to handle side intersections.

I also experimented with decreasing the field of view of the LiDAR sensor to 180 degrees. To do so, I filtered the point clouds to only keep points with a positive value on axis 1. Using the previous settings (1000 points per cloud, batch size of 64), I trained a model and then tested it in the CARLA simulator to compare its performance to the models trained on 360 degree field of view point clouds. The performance in Town01 was quite similar, the model was able to correctly predict steering values of natural turns, straight roads, T-junctions and side intersections. In the unknown environment of Town02 however, the performance was worse than with the 360 degree model. While the model was able to predict correct steering values for straight roads and T-junctions, it had major issues with some natural turns and all side intersections.

Examining the dataset revealed, that side intersections in Town02 look differently from side intersections from Town01 included in the dataset - while every side intersection in the dataset had a fence on the opposite side of the road, making it easy for the model to recognise the turn, side intersections in Town02 only had a sidewalk and a few pillars (see figure 29).

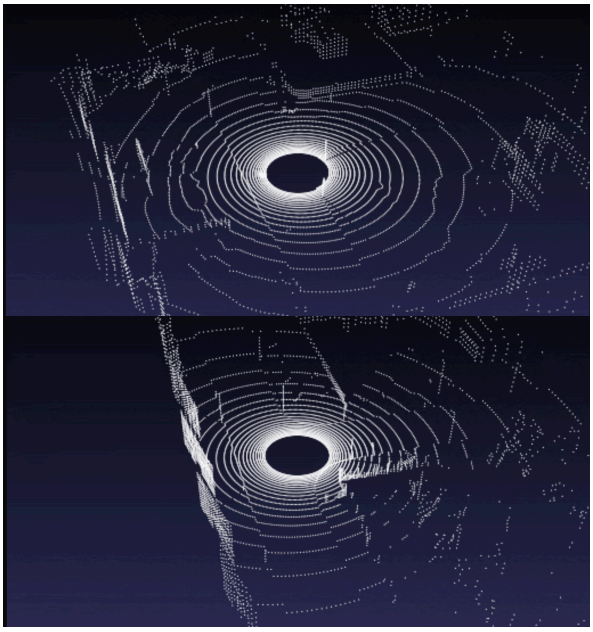

Figure 29. Top: side intersection in Town01, Bottom: side intersection in Town02.

After training a model on situations from Town02, it was able to handle most of the side intersections of Town02, but still struggled with some of them and was less reliable than the model trained on preprocessed dense point clouds.

7.2. Unsupervised segmentation

The LiDAR models showed a good performance without the need to preprocess the data by extracting the road edges or lines. However, unlike for the models trained on preprocessed dense point clouds, it was necessary to include examples of Town02 in the training data in order for the model to be able to handle them in the simulator. On the other hand the dense point cloud models were dependent on ground truth segmentation data, which is not available in a real world scenario. In consideration of the higher costs for LiDAR sensors, it is desirable to find a way to use dense point clouds without being dependent on ground truth segmentation data.

One such way might be to use an unsupervised clustering model to segment the RGB images and to use this segmentation to filter the dense point clouds. In order to assess the potential performance of such an approach, I started researching for state-of-the-art unsupervised clustering networks and found a paper called "Invariant information clustering (IIC)" by Ji et al. 2019 [5]. After setting up the provided implementation, I trained the IIC model on two different datasets to evaluate its performance. One dataset contained semantic segmentation images from one episode of the CALRA dataset with added gaussian noise, the other dataset contained raw RGB images from one episode of the CARLA dataset. Predictions of those two models are shown in figure 30 and figure 31.

While the segmentations predicted by the model were not very similar to the ground truth segmentation, especially the segmentation provided by the model trained on raw RGB images looked very promising. The light pink class visible in figure 31 corresponds to road lines and sidewalks, albeit being very coarse. In section *Outlook* I provide strategies on how the segmentation performance could be improved.

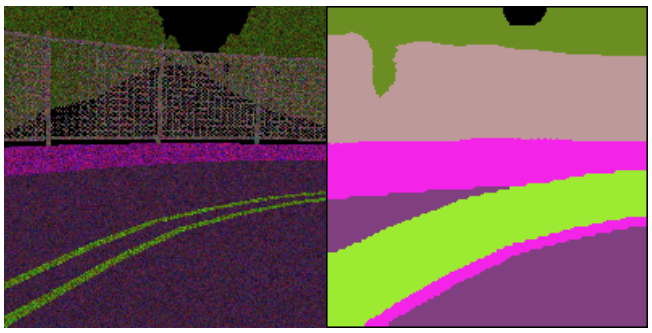

Figure 30. Left: Ground truth segmentation with added Gaussian noise used as input for the segmentation model. Right: Segmentation predicted by the trained IIC model.

Figure 31. Left: RGB image used as input for the segmentation model. Right: Segmentation predicted by the trained IIC model.

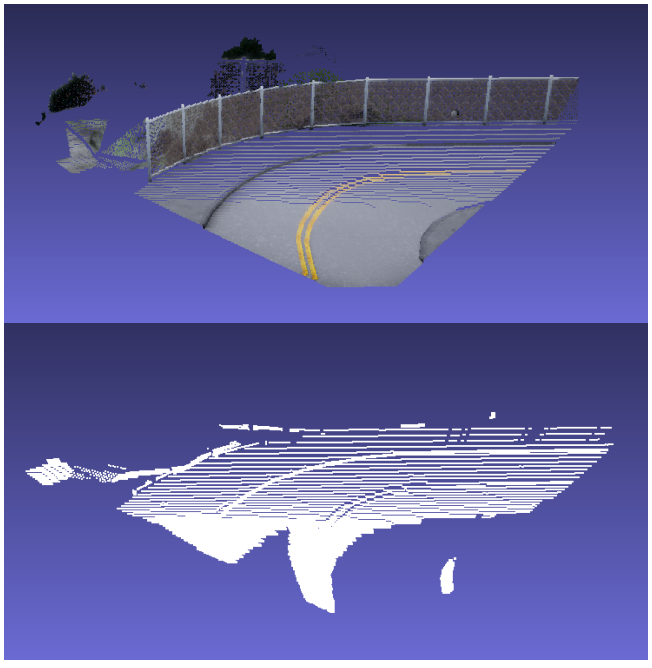

Figure 32. Top: full point cloud (coloured for visualisation purposes). Bottom: points extracted by filtering with predicted segmentation

Next, I projected the predicted segmentations onto the point clouds and extracted the points belonging to the class roughly corresponding to road lines and sidewalk. Then I trained the PointNet model on a dataset consisting of these processed point clouds. After filtering, the point clouds contained around 25.000 points, so it was possible to train on them by using a batch size of 8.

Figure 33 shows the offline prediction performance of this model on one natural turn episode.

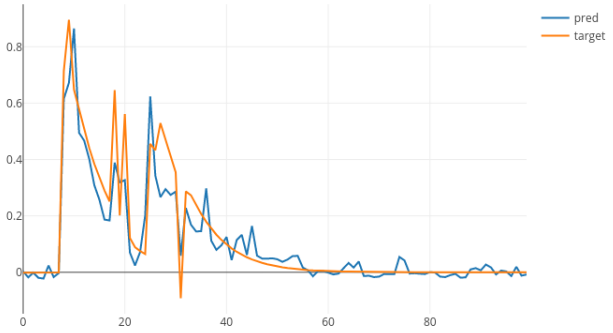

Figure 33. Offline test result of model trained on dataset containing one natural turn, preprocessed using the trained IIC model, using 25.000 points per cloud for 100 epochs. Horizontal axis: number of frame, vertical axis: steering value. The model's performance looks promising, but must be tested in in the simulator before any real conclusions can be drawn from it.

7.3. Future improvements

Improving the unsupervised segmentation would allow to filter the dense point cloud more precisely and thereby improve the overall performance of the steering prediction. Based on visualisations of segmentations predicted by the current IIC model, one could try decreasing the network's receptive field. This would help with capturing finer details in the segmentation, since as of now the segmentation is very coarse. This could be done by either directly modifying the network's receptive field by modifying the architecture or by modifying the effective receptive field by scaling up the input images. A smaller receptive field might also help with the border issue visible in figure 31. According to the authors of the IIC paper, the network's receptive field in pixels should be roughly tied to the size of the smallest detail to be captured in the input images.

8. Conclusion

The goal of this project was to modify or extend the PointNet architecture in order to control a car in the CARLA simulator by predicting the steering value from a given point cloud. The original plan was to use dense point clouds, since they are cheaper to produce. Two approaches using dense point clouds and one approach using sparse LiDAR point clouds were explored.

The first approach depended on preprocessing the point clouds using ground truth segmentation data, but showed great performance during tests in the simulator.

The second approach utilised sparse point clouds generated by a LiDAR sensor without preprocessing to train the PointNet model and showed a great initial performance in Town01, but depended on explicit examples of situations of Town02 in order to perform well there.

The third approach used an unsupervised clustering model to preprocess dense point clouds and was not dependent on ground truth segmentation data. However it showed a less than optimal performance in simulator tests. By improving the performance of the unsupervised clustering model with methods briefly described in the last section, it might be possible to improve the overall performance of predicting steering values.

A general result of this project is that the PointNet model is well suited for the regression task of predicting the steering value from a given point cloud.

9. References

[1] PointNet by Qi et al. 2017 (https://arxiv.org/abs/1612.00593)
[2] CARLA simulator for autonomous driving research: http://carla.org/
[3] PointNet PyTorch implementation: https://github.com/fxia22/pointnet.pytorch
[4] Visdom: https://github.com/facebookresearch/visdom
[5] Invariant information clustering by Ji et al. 2019: https://arxiv.org/abs/1807.06653