Lateral Ego-Vehicle Control without Supervision using Point Clouds

Florian Müller*, Qadeer Khan* and Daniel Cremers

Computer Vision and Artificial Intelligence Group
Technical University of Munich, Germany
{f.r.mueller, qadeer.khan, cremers}@tum.de

Abstract— **Existing vision based supervised approaches to lateral vehicle control are capable of directly mapping RGB images to the appropriate steering commands. However, they are prone to suffering from inadequate robustness in real world scenarios due to a lack of failure cases in the training data. In this paper, a framework for training a more robust and scalable model for lateral vehicle control is proposed. The framework only requires an unlabeled sequence of RGB images. The trained model takes a point cloud as input and predicts the lateral offset to a subsequent frame from which the steering angle is inferred. The frame poses are in turn obtained from visual odometry. The point cloud is conceived by projecting dense depth maps into 3D. An arbitrary number of additional trajectories from this point cloud can be generated during training. This is to increase the robustness of the model. Online experiments show that the performance of our method is superior to that of the supervised model.**

I. Introduction

In recent years, deep learning approaches have shown a promising trend in the context of lateral sensorimotor control [1], [2], [3]. The trained network can directly map input data to the steering commands [4], [5]. Labeled training data is usually acquired by recording the raw sensory input and the corresponding steering commands executed by an expert driver traversing a reference trajectory. One of the main challenges of this approach is the lack of failure cases in the training data, caused by the driver's obligation to follow traffic rules and to remain within its own driving lane. Without failure cases in the training data, the model has no way of learning to recover from a divergence from the reference trajectory [6]. This is a common issue with deep learning that models tend to fail at inference time when encountering images that are out-of-distribution from the training set [7]. Previous works have attempted to solve this problem by generating training images with lateral displacement and adjusted steering label [5], [8], [1]. However, due to limitations in the maximum lateral offset they can generate, the learned driving policy tends to be not robust enough [9].

In contrast to supervised approaches, Reinforcement Learning (RL) can be used to learn a driving policy for lateral control [10], [11]. RL does not require explicit data-label pairs for training. Rather, the model learns a suitable policy by randomly exploring the environment in a hit and trial method using a pre-defined reward function [12]. However, in the context of self-driving, random exploration of the driving environment is not a feasible solution as it may involve dangerous traffic violation, potentially causing crashes. This is why [7], have an expert driver that assumes control whenever the car starts to deviates off-course.

As depicted in Figure 1, we propose a scalable framework that neither requires supervised labels nor needs data collected in violation of traffic rules. The model learns to predict the lateral offset to subsequent frames from an unlabeled sequence of RGB images, captured with a single front facing camera. Internally, the input images are converted to 3D point clouds, which enables the generation of an arbitrary number of realistic synthetic trajectories parallel to the reference trajectory. The labels for each frame are generated from reconstructed camera poses using visual odometry. This removes the requirement of labeled training data.
The primary contributions of our work are summarized below:

1) We demonstrate how a model for lateral vehicle control can be trained from only an unlabeled sequence of images.
2) We show how generating additional training data leads to enhanced robustness of the model at inference time.

II. Method

In this section the individual components of our framework depicted in Figure 1 are described in further detail. The framework trains a deep learning model for the task of lateral ego-vehicle control from an unlabeled sequence of RGB images.

A high-level overview of the individual components is summarized below

1) **Point Cloud Generation:** Using a self-supervised training paradigm, depth maps are predicted from the RGB images, which are then used to generate 3D point clouds.
2) **Camera Pose Estimation:** Using a general-purpose visual odometry pipeline, camera poses are estimated for each frame.
3) **Point Cloud Augmentation:** By aligning, shifting and cropping the point clouds, new point clouds are synthesized, which simulate additional trajectories.

This work was supported by the Munich Center for Machine Learning. The final authenticated version is available online at https://doi.org/10.1007/978-3-031-09037-0_39
* These authors contributed equally

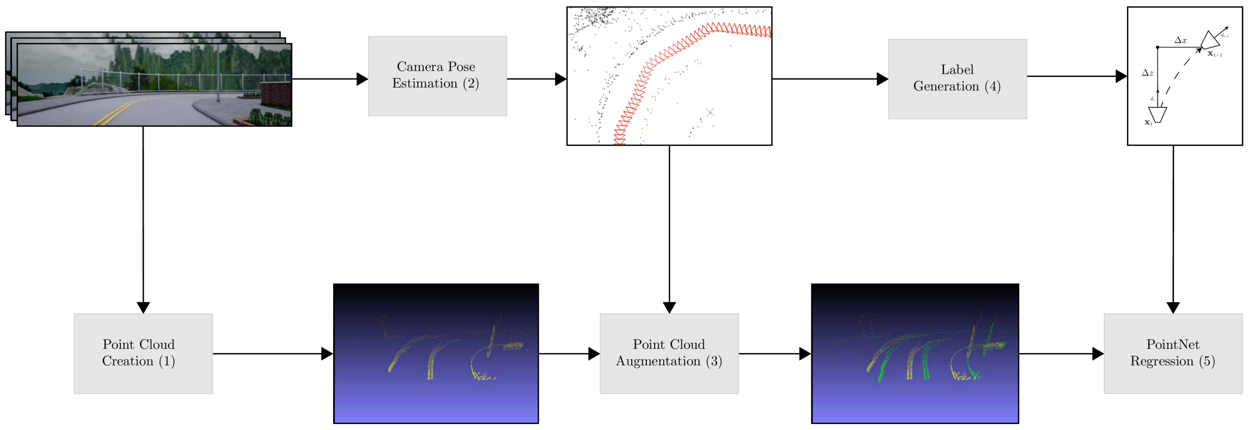


Fig. 1: This figure describes the high level overview of the proposed framework (1) Point clouds are generated from RGB images (Figure 2, Section II-A). (2) The camera pose of each frame is estimated using visual odometry (Section II-B).(3) New camera frames are generated by aligning, shifting and cropping point clouds from existing frames (Section II-C). (4) Labels representing the offset of the next frame in lateral direction are generated from the reconstructed camera poses (Section II-D). (5) The resulting point clouds and labels are used to train a deep learning model to predict the lateral offset of the next frame given a point cloud (Section II-E).

4) **Label Generation:** Using the camera poses, the lateral offset to a subsequent frame is calculated for each frame. This serves as a target label to train the model.
5) **Model Training:** A deep learning model is trained to predict the lateral offset to the next frame given a point cloud as input.

These components are described in detail in the following subsections.

A. *Point Cloud Generation*

Note that the point cloud for a corresponding RGB image is needed for 2 purposes:

1) As an input to the model for predicting the steering angle for lateral vehicle control
2) It is used for synthesizing additional training trajectories. Hence, the data collection along trajectories which would otherwise violate traffic rules can be avoided.

Figure 2 shows an example of a point cloud projected into 3D using the dense depth map of a corresponding RGB image. The depth map is obtained from a depth estimation network which takes an RGB image as input and can be trained in an entirely self-supervised manner [13]. The training only requires a monocular sequence of RGB images. Note that the depth produced is normalized. Hence, calibration needs to be done to find the appropriate factor to scale the normalized depth map to the world scale.

In the point clouds generated from dense depth maps, it may be hard to recognize relevant high-level features like road markings, traffic poles etc. These high level features tend to be important for the model to take the appropriate steering decisions [14]. Moreover, having redundant points in the point cloud which do not yield useful information for the vehicle control model would impose an unnecessary computational burden. To counteract both issues, an edge filter [15] is applied on the RGB image used to predict the depth map. The resulting point cloud is then filtered to only include points on the detected edges. The edges can further be dilated by a small amount to prevent losing finer details. Additionally, points beyond a certain distance from the camera are discarded. This increases the concentration of points in the relevant areas closer to the camera. Points closer to the camera are more important for the control model for immediate decision making.

B. *Camera Pose Estimation:*

The monocular sequence of RGB images contains no explicit information about the camera poses. The poses serve two purposes:

1) To determine the target labels when training the network for lateral ego-vehicle control. The input to this network is the filtered point cloud.
2) For aligning point clouds when additional trajectories are generated.

The camera poses in the global frame of reference can be obtained from the image sequence using a general-purpose visual odometry pipeline such as [16], [17], [18]. They produce the 6DoF camera pose and a sparse 3D representation of the scene as depicted in Figure 3. The camera pose can be expressed as a transformation matrix belonging to the special Euclidean group SE(3) representing a rigid body motion.

The path traversed by a sequence of these camera poses is referred to as the *reference trajectory*. However, note that

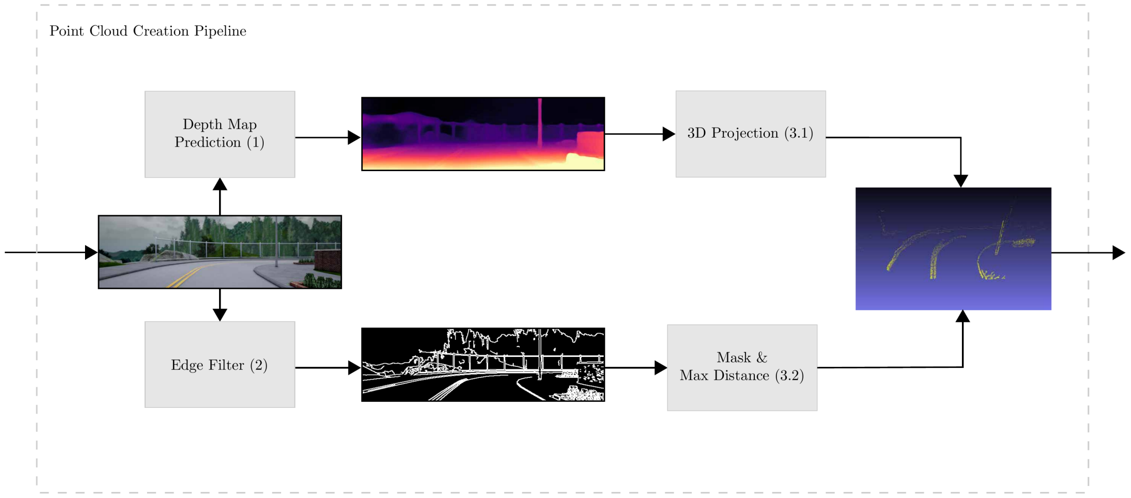


Fig. 2: The figure describes the high level overview of the point cloud creation pipeline. An (inverse) depth map is predicted from an input RGB image, which in turn is projected to a 3D point cloud. Additionally an edge filter is applied to filter the point cloud by removing points beyond a certain threshold distance

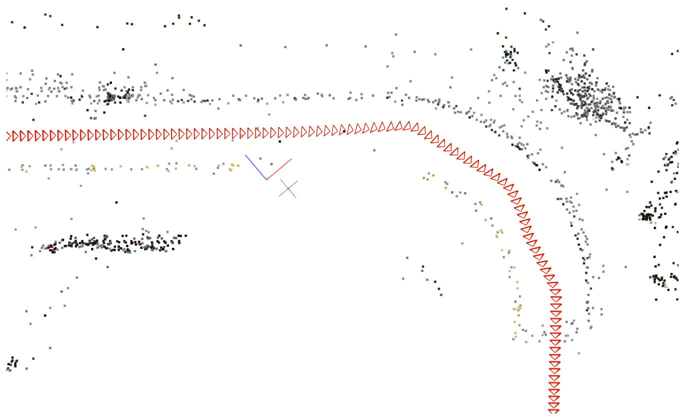


Fig. 3: Using visual odometry, the camera poses and a sparse 3D representation is reconstructed simultaneously from the sequence of RGB images. The figure shows a birds eye view on the reconstructed trajectory for a small section of the road. Each red triangle represents the reconstructed pose of one camera frame in the image sequence. The dots represent a sparse reconstruction of the 3D scene and are obtained by matching features extracted from the images in multiple frames.

the camera poses obtained from a monocular sequence of images does not necessarily reflect the actual scale. As was the case in Section II-A, calibration is done to determine the appropriate scaling factor. Note that the 3D reconstruction of the scene obtained by running visual odometry is too sparse to be used as an input to the vehicle control model. This is why we have used the approach described in Section II-A that is capable of generating dense depth maps, while being trained in a completely self-supervised manner.

C. Point Cloud Augmentation

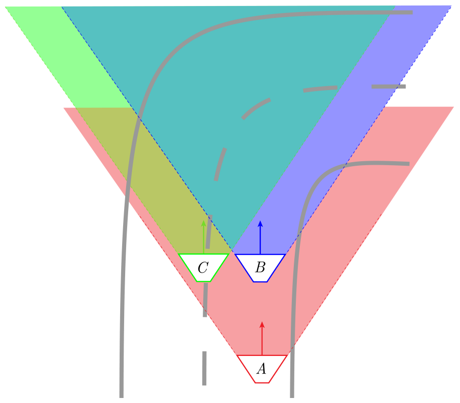
After obtaining the camera poses and point clouds for each camera frame, we finally have all prerequisites to generate additional trajectories needed to train the vehicle control model. Additional trajectories can be generated by laterally translating the point clouds from the base frame in the reference trajectory to a new frame position. Note that the reference trajectory's point cloud does not necessarily contain all the points encompassed by the field of view (FOV) of the new frame position. Those missing points are included from frames in the reference trajectory preceding the base frame. Conversely, points that do not fall within the field of view of the new camera frame position are discarded. The process is illustrated in Figure 4, wherein Camera B is the base frame in the reference trajectory, while Camera A is a preceding frame also in the reference trajectory. We show how a point cloud at the new position represented by Camera C can be synthesized. The process can be split into the following steps:

1) Align the point cloud of a preceding frame in the reference trajectory with that of the base frame.
2) Shift the aligned point cloud from the base frame of reference to the new frame position
3) Remove all points outside the field of view of the new frame position.

Next, we discuss these steps in detail.

Aligning Point Clouds

Consider two camera frames A and B. The *global transforms* of the frames relative to the world frame are represented by T_A and T_B. We have two local point clouds P_0 and



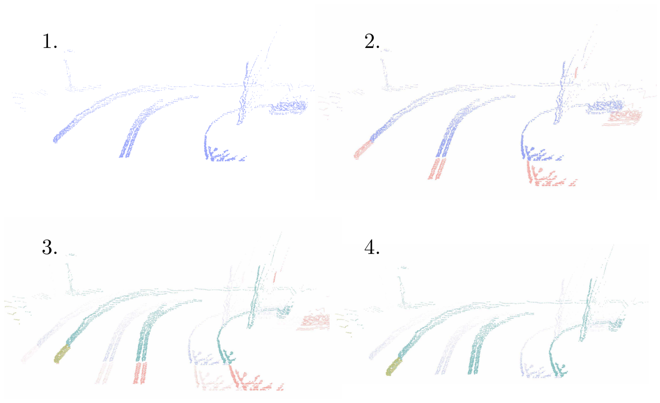


Fig. 4: **Generation of a new camera frame. Top:** Visualization of the fields of view of different camera frames. Camera C's field of view contains points not included in camera B's field of view. These points are added by aligning a previous camera frame A. **Bottom: 1.)** Point cloud of camera B (blue). **2.)** Aligning A's point cloud (red) to add missing points. We choose A to be far enough in the back to cover the field of view of C. **3.)** Changing the points' reference frame from B to C. (The faded points represent the points with reference frame B.) **4.)** Finalizing the point cloud creation by removing all points outside the field of view of C.

P_1, which were captured from camera A and B, respectively. We call P_0 and P_1 local point clouds, because the coordinates of their points are given relative to their respective reference frames A and B.

Aligning point cloud P_0 with point cloud P_1 is synonymous to representing the coordinates of P_0 relative to frame B. The relative transform representing the rigid body motion from camera B to camera A is the same transform that changes the reference frame of the coordinates of point cloud P_0 from A to B.

Applying the transform T_A to P_0 changes the reference frame of the coordinates of P_0 from frame A to the world frame. It moves P_0 to its global position. Let us call this global point cloud P_{0_W}. Applying the inverse of T_B to P_{0_W} changes the reference frame for the coordinates of P_{0_W} from the world frame to frame B. This new point cloud P_{0_B} is aligned with P_1, as the coordinates of both point clouds are given relative to frame B.

In summary, the transform we need to apply to P_0 to align it to P_1 is given by:

$$T_{BA} = T_B^{-1} T_A \tag{1}$$

Applying T_{BA} to P_0 and concatenating the result with with P_1 gives us a new point cloud $P_{0,1}$, containing all the points of P_0 and P_1 with coodinates given relative to frame B:

$$P_{0,1} = \begin{bmatrix} T_{BA}P_0 & P_1 \end{bmatrix} \tag{2}$$

If we choose A to be a camera frame earlier in the sequence than B, then the scene represented by the combined point cloud $P_{0,1}$ includes points from a wider range than P_1. This allows us to generate a new camera frame by shifting and cropping the point cloud in the next step.

Shifting and Cropping Point Clouds

To generate a new trajectory, we have to generate point clouds from the point of view of the camera frames in the new trajectory. This paper focuses on trajectories parallel to the base trajectory. Therefore, new camera frames are always shifted in lateral direction from the base camera frames.

The field of view of a camera C shifted in lateral direction from camera B contains points not included in the field of view of camera B. In the previous step, we therefore aligned previous point clouds to the point cloud of camera B to obtain a combined point cloud $P_{0,1}$ including all points in the field of view of camera C. The coordinates of $P_{0,1}$ are given relative to frame B and its global transform is given by T_B.

We want frame C to be shifted by some value x in lateral direction from frame B. We will refer to frame B as the *base frame*. The transform representing a lateral shift by value x is given by T_x:

$$T_x = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, x \in \mathbb{R} \tag{3}$$

The relative transform of frame C to frame B is therefore given by T_x. In general, T_x could be any rigid body motion but this paper focuses on trajectories parallel to the base trajectory and therefore on lateral shifts. To obtain the global transform T_C of frame C relative to the world frame, we have to change the reference frame of T_x from B to the world frame. We do so by applying T_B to T_x.

$$T_C = T_B T_x \tag{4}$$

$$T_x = T_B^{-1} T_C \tag{5}$$

Now we want to change the reference frame of $P_{0,1}$ from frame B to frame C. We can do so in two steps. First we change the reference frame of $P_{0,1}$ from frame B to the world frame by applying T_B. Then we can change the reference frame from the world frame to frame C by applying T_C^{-1}. This two step transform is equal to applying T_x^{-1} directly to $P_{0,1}$.

$$\begin{aligned} P_{0,1_C} &= T_C^{-1} T_B P_{0,1} \\ &= (T_B^{-1} T_C)^{-1} P_{0,1} \\ &\overset{(5)}{=} T_x^{-1} P_{0,1} \end{aligned} \tag{6}$$

$P_{0,1_C}$ is now at the position a local point cloud perspective of camera C. But it also contains points outside the field of view of camera C. To get a realistic point cloud, we need to discard the points whose projections lay outside the image plane of camera C.

To obtain the 2D coordinates of the points' projections, we multiply the intrinsic camera matrix (A) with the point cloud and divide each point by its depth to obtain its coordinates on the 2D image plane.

$$\begin{bmatrix} x_{im} \\ y_{im} \\ 1 \end{bmatrix} = \frac{1}{Z} A \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{7}$$

Finally we drop each point from the point cloud for whose projection do not fulfill the following criteria:

$$0 <= x_{im} < width \land 0 <= y_{im} < height$$

where $width$ and $height$ denote the dimensions of the image plane.

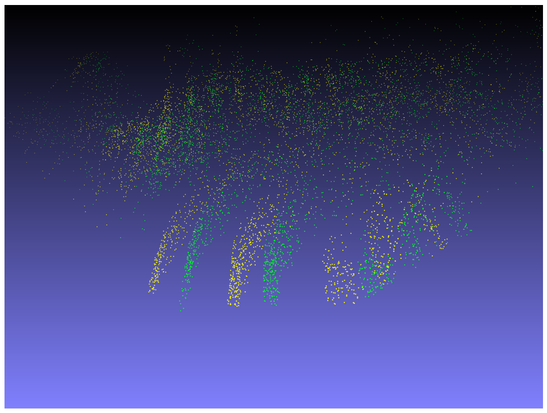
Counteracting Imperfect Camera Poses

The larger the distance of the new frame C to its base frame B, the more points in the field of view of camera C are not laying in the field of view of camera B. Therefore the larger the distance, the more point clouds from camera frames earlier in the sequence need to be aligned to include their points in the generated point cloud of the new camera frame C.

The camera poses predicted by visual odometry may deviate slightly from the true poses and the further we go back in the sequence, the more significant this deviation becomes. If we just align the previous point clouds and add all their points to the generated point cloud, the result looks quite fuzzy (see Figure 5a).

To counteract this issue, we can remove all points from previous, aligned point clouds that lay within the field of view of our base frame. Consider camera frame A, which is earlier in the camera sequence than camera frame B. Let P_0 be the point cloud captured from camera A and P_1 the point cloud captured from camera B. After aligning P_0 to frame B, we obtain the 2D coordinates of the projections of its points onto the image plane of camera B as described in Equation 7. Then we drop each point from P_0 for whose projection the following criteria **does** hold:

$$0 <= x_{im} < width \land 0 <= y_{im} < height$$


(a)

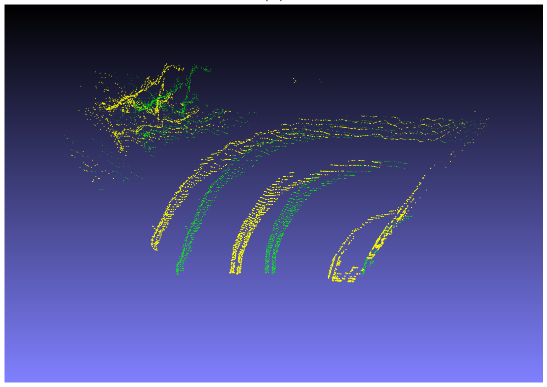

(b)

Fig. 5: **Counteracting Imperfect Camera Poses. a)** Multiple edge filtered point clouds are aligned to generate a new camera frame. Due to imperfect predicted camera poses, the resulting point cloud looks very fuzzy. **b)** By only adding points outside the field of view of the original camera and limiting the maximum distance, we can increases sharpness of the point clouds. Moreover, the maximum distance is limited to a threshold, as the depth prediction becomes less accurate with greater distance and closer points are more relevant for the prediction of the lateral offset of the next frame.

where $width$ and $height$ denote the size of the image plane of B. We only keep the points whose projection lies outside the image plane of camera B. This ensures that missing points are being added, but the existing point cloud is not blurred (see Figure 5b).

D. Label Generation

We have already discussed the process of preparing the training data and generating additional trajectories. The next step is to generate the target labels for this data in order to be able to train a model for lateral vehicle control. Note that the camera is rigidly attached to the car, therefore the camera pose for any frame can also be used to determine the pose of the car at the corresponding timestep. For each car pose, we would like to find the appropriate steering angle to be executed such that a subsequent car pose is attained. We model the dynamics of front wheeled driven car using the bicycle model [19]. We assume the no-slip condition between the front and rear wheels [20]. This holds true when the car is moving straight or making turns at low/moderate speeds. Then the steering angle(δ) of the car is described by the equation [21]:

$$\delta = \tan^{-1}(\Delta x \cdot \alpha) \tag{8}$$

Where Δx is the lateral distance between 2 frames. This can be determined from the camera poses. The 2 frames are chosen such that they are approximately a fixed longitudinal distance apart. This is to cater for the car moving at variable speed or in case of dropped camera frames during data collection. α is a constant that can be calibrated at inference time depending on the car.

E. Model Architecture

Using the generated point clouds and labels, a deep learning model is trained to predict the lateral deviation (Δx) between the current frame and a subsequent frame. The model can be considered as comprising of 2 main components. The first takes raw point clouds of the current frame as input and produces a global feature vector to furnish a latent representation of the scene as seen by the ego-vehicle. We adapt the PointNet architecture [22] prior to the classification head for this. The next component consists of multi-layer perceptions which map this global features vector to furnish the output. A hyperbolic tangent function forms the last layer. It is scaled by an arbitrary fixed value *a* to allow predictions in the range of -*a* and *a*. The loss function chosen to train the model is *mean squared error* of the lateral deviation (Δx) between the 2 frames and that predicted by the model. Figure 6 shows a visualization of the model architecture.

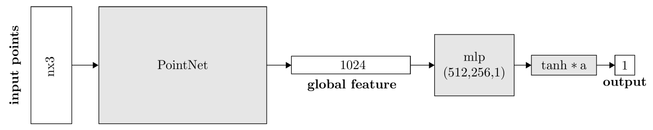


Fig. 6: Shows the architecture of the model which takes a point cloud as input and predicts the lateral deviation (Δx) to a subsequent frame. [22] is used as the base architecture to produce a feature vector with length 1024 from the point cloud with n points. Meanwhile, "mlp" represents a multi-layer-perceptron, with the numbers in brackets representing layer sizes. "tanh" is a hyperbolic tangent function. It is scaled by a fixed value *a* to allow predictions in the range of -*a* to *a*.

III. Experiments

We use the CARLA (CAR Learning to Act) [23] simulator (stable version 0.8.2) for our experiments. It allows for an online evaluation to assess the true driving quality. This is as opposed to offline evaluation where 2 models with the same offline metrics can have drastically different driving performance [24].

The unit of measurement we use for online evaluation is the *ratio on lane* metric adapted from [14]. It gives the ratio of frames the ego-vehicle remains within its own driving lane to the total number of frames. The ego-vehicle is considered driving within its own lane if no part of the bounding box of the car is on the other lane or off the road and the car is not stopped due to a collision with other traffic.

Data Collection:
Image data is collected at a fixed frame rate by traversing the ego-vehicle in auto-pilot mode in Town 01 of the CARLA simulator. Images of size 640 x 192 (90°FOV) are used to train the depth estimation network using [13]. Meanwhile, [17] determines the camera poses. The depth is used to generate a 3D point cloud which in turn produce additional shifted point clouds at off course trajectories. We generate ten additional point cloud trajectories at uniform lateral distances in the range of [-2,2] meters from the single reference trajectory. The missing points at shifted point clouds can be compensated for by aligning preceding point clouds using the camera poses. Only the missing points from the preceding frames are added. This is because imperfect camera poses may result in duplication of objects, thereby confusing the model. The camera poses additionally allow to determine the target labels to train the network. The number of points input to the network are fixed to 4096. This is done by filtering the point cloud to retain only the high level edge features, while points beyond 20m are discarded.

Quantitative Evaluation:
The evaluation at inference time is done at different starting positions in the town that are unseen during training. Each episode is executed for 135 frames at fixed throttle and the mean ratio on lane metric is reported across all unseen episodes. Results of our method are reported in Figure 7.

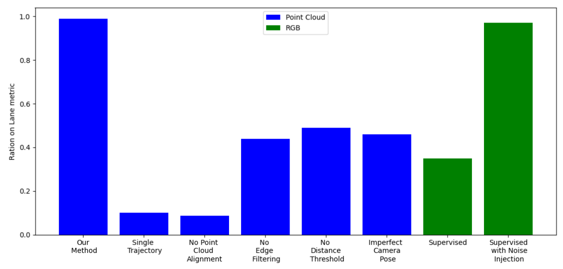


Fig. 7: Average ratio of frames the ego-vehicle remains within its driving lane for different model configurations across different starting positions unseen during training. Higher value is better. Note that our method is better than the supervised and at par with the supervised approach with noise injection [9]. However, in contrast to this noise injection strategy, our method does not run the risk of traffic violations during the data collection phase. This is further explained in Section IV

Moreover, comparison with different point cloud configurations is also done to see the impact of the various components of our framework on the overall driving performance. Configurations explored are: training with a single trajectory rather than with multiple generated trajectories, aligning preceding point clouds to generate a new trajectory versus only shifting, limiting the maximum depth distance, using an edge filter to filter the point cloud and dealing with imperfect camera poses. We additionally compare with supervised RGB model baselines. Therein, we demonstrate that the performance of our framework is superior to the supervised model and at par with the supervised model trained with noise injection [9] which we explain in Section IV.

IV. Discussion

In this section, an explanation of the various configurations given in Figure 7 are described. The consequence of these configurations on the online driving performance are also discussed.

Single Trajectory v. Multiple Generated Trajectories:
The core of our proposed method revolves around the capability of generating additional trajectories from a single reference trajectory, as described in Section II-C. We therefore evaluate the impact of this by comparing with the single trajectory model trained only on the reference trajectory. It can be seen that the ratio on lane metric for the single trajectory is far inferior in comparison to our multi trajectory model. One plausible explanation is that when the model deviates off course, it cannot make the appropriate correction, since the reference trajectory data does not capture such scenarios during training. Figure 8 shows a birds eye view (BEV) visualization of the models' paths at one of the starting positions in the training set. While our model is able to follow the road, single trajectory model crashed straight into the barrier.

Aligning Point Clouds when Generating new Trajectories:
Section II-C described the process of generating point clouds in new trajectories by aligning point clouds from a preceding frame and then shifting. To examine the necessity of aligning previous point clouds, we train another model whose point clouds were generated without aligning previous point clouds but instead only shifting and cropping the reference trajectory. Shifting the point cloud will yield empty regions that cannot be captured within the FOV of the source camera in the reference trajectory. This is particularly true of trajectories that are generated at farther distances away from the reference. It is further exacerbated for source images captured when the car is executing turns. As can be observed, the performance of such a model trained with partially observable point clouds drops dramatically.

Edge Filtered v. Full Point Cloud:
The authors of [14] alluded to high level features such as lane markings, sidewalk/road intersections, barriers etc. being important for the vehicle control model to hold its driving lane. However, in uncolored point clouds those features are clearly less visible than in RGB images. This is why it is important to make them more prominent to the model

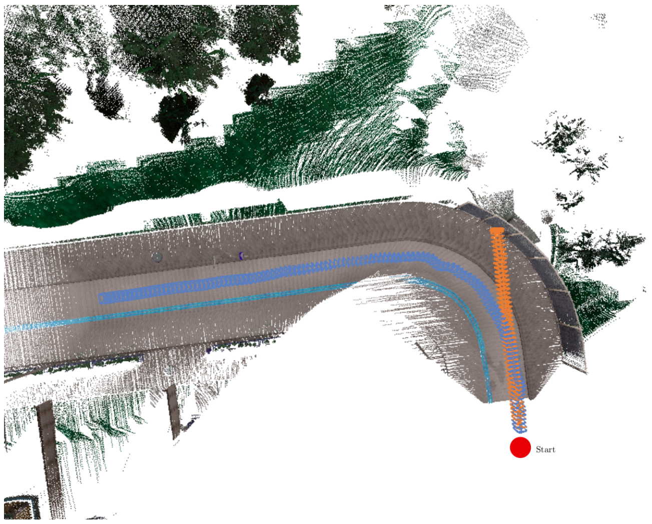


Fig. 8: **Path visualization**. The red dot marks the starting position. *Out method* (blue) was trained using multiple generated trajectories and is able to follow the curve. Meanwhile, the *Single Trajectory* (orange) model was trained using only the single reference trajectory and drives straight into the barrier.

and increase the density of relevant information in the point cloud. This is done by filtering seemingly irrelevant points from the point cloud by applying an edge filter as described in Section II-A. To evaluate the benefit of edge filtering, another model was trained on the full, unfiltered point clouds. The ratio on lane performance of this model drops, thereby advocating in favor of our edge filtering approach.

Limiting the Distance:
Section II-A also mentioned limiting the distance of points to the camera as part of an approach to increase concentration of points in relevant areas closer to the camera. If a model is trained without a maximum point distance, its performance drops. One tenable explanation for this is that at larger distances, the depth prediction is less certain. This imperfect depth could result in object duplication at farther distances when aligning point clouds. This can be observed in the top right region of the 2nd point cloud in Figure 4. Hence, when such anomalies at farther depths are removed from the point cloud by limiting the maximum distance, the performance of the model is enhanced. Moreover, points in the immediate vicinity of the ego-vehicle are more important for executing the appropriate steering command rather than the points farther away that have imprecise depths.

Dealing with Imperfect Camera Poses When aligning previous point clouds as part of the process of generating a point cloud for a new trajectory, imperfect camera pose estimations can cause blurriness if objects in the 3D space do not align. Subsection II-C described an approach of counteracting this blurriness by only adding missing points outside the base camera frame's field of view instead of all points. To evaluate the effectiveness of this approach, we trained another model wherein all points are used for training. As can be seen the performance of such a model drops.

Supervised RGB model:

We additionally train a supervised convolutional model to compare with our framework using point clouds. This new model takes in an RGB image as input and directly predicts the appropriate steering command for vehicle control. In contrast to our method this RGB model is supervised and uses the ground truth steering commands as labels during training. The performance of this model is much lower than our method. This is despite the fact that our model was not trained with any ground truth steering labels. Therefore, the superior performance of our method can be attributed to the ability to generate additional off trajectory training data from a single traversal of the ego-vehilce causing it to be more robust.

Affect of Perturbations: To further compare the robustness of our method with the supervised RGB model, we add perturbation of varying degree into the steering command predicted by the model at each time step. The amount of perturbation is sampled from a uniform random distribution. Figure 9 shows that as magnitude of perturbation is enhanced, our model maintains a fairly consistent performance. In contrast, the supervised RGB model has a steep performance drop. This is because when perturbations are introduced the ego-vehilce may drive off-course for which the supervised model is not capable of counteracting such situations. On the other hand, our model is accustomed to handling such situations as it was fed additional off trajectory point clouds during training.

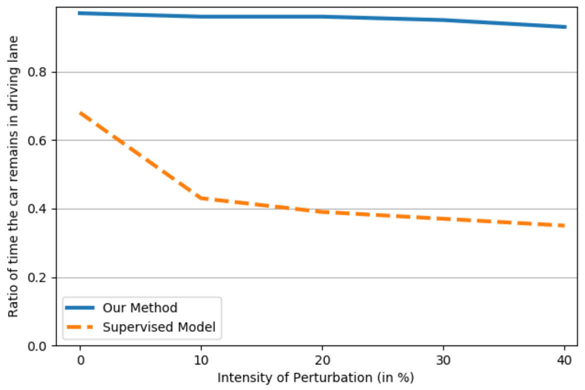


Fig. 9: **Avg. ratio on lane for different levels of perturbation on the training trajectories.** The performance of both the supervised RGB model (orange) and Our method (blue) decreases with higher intensity of perturbations, but the descent is more dramatic for the supervised model.

Figure 10 shows a BEV visualization of the paths of Our (blue) and the supervised RGB (orange) models when noise is added in one of the training trajectories. Despite the perturbations, our model is able to keep the car on its lane. Meanwhile, the supervised RGB model looses track shortly after the start, is not able to correct its path and eventually crashes.

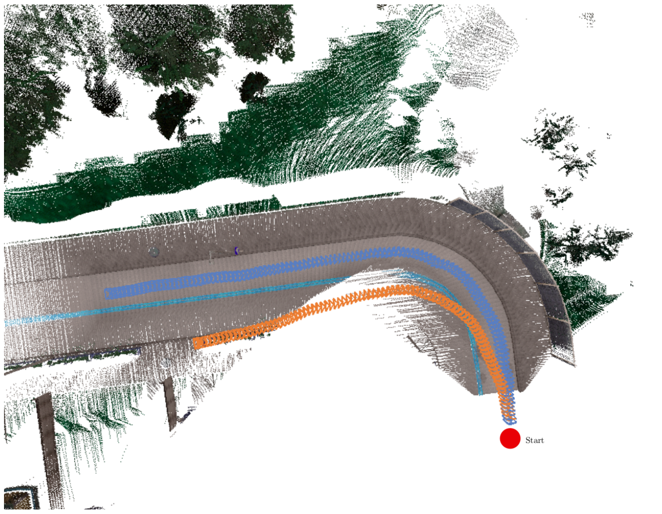


Fig. 10: **Path visualization on a trajectory in the training set with noise level of 10%.** The blue path belongs to the model trained with *Our Method*, while the orange path represents the *Supervised RGB* model. The red dot marks the starting position. *Our Method* is able to keep the lane despite added noise, while *Supervised RGB* leaves the lane shortly after the start and is not able to recover.

Supervised model with noise injection:

We additionally compare our approach with the supervised RGB model proposed by [9]. It is similar to the supervised model described above except that during data collection noise is injected into the steering command. This causes the vehicle to diverge from its normal path. The corrective steering maneuver taken by the expert driver to limit this divergence and bring the vehicle back on course is recorded. Performance of such a model is comparable to our approach. This is because the data used in training such a model contains off-trajectory images and corresponding labels. However, executing such a strategy for data collection with traffic participants may be extremely dangerous and may even involve violation of traffic rules. Moreover, it involves having an expert driver capable of handling dramatic maneuvers resulting from this noise injection. Most importantly, since our approach does not require any noise injection, it does not run the risk of violating traffic rules during data collection.

V. Conclusion

In this paper, we put forth a framework to train a point cloud based deep learning model on the task of lateral control of an autonomous vehicle. The model is capable of learning a robust driving policy from merely an unlabeled sequence of RGB images, captured with a single front facing camera on a single reference trajectory. The efficacy of our approach comes from the capability of generating additional data from the same sequence. The additional data appears as if it emerges from an off course trajectory. This counteracts the limitations of imitation learning which suffer from insufficient robustness in the real world due to a lack of failure cases in the training data. Online experiments on the driving simulator showed that its performance is superior to a supervised baseline CNN trained on the same initial data set. Given no labels are required for the training data, the approach is scalable to large quantities of data. This makes it a more robust alternative to current supervised end-to-end lateral control methods.

Acknowledgement:
This work was supported by the Munich Center for Machine Learning.

References

[1] M. Toromanoff, E. Wirbel, F. Wilhelm, C. Vejarano, X. Perrotton, and F. Moutarde, "End to end vehicle lateral control using a single fisheye camera," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3613–3619.

[2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[3] D. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *NIPS*, 1988.

[4] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1856–1860.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *arXiv e-prints*, p. arXiv:1604.07316, Apr. 2016.

[6] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[7] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8248–8254.

[8] C. Hubschneider, A. Bauer, M. Weber, and J. M. Zöllner, "Adding navigation to the equation: Turning decisions for end-to-end vehicle control," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–8.

[9] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 4693–4700.

[10] Q. Zhang, R. Luo, D. Zhao, C. Luo, and D. Qian, "Model-free reinforcement learning based lateral control for lane keeping," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–7.

[11] D. Li, D. Zhao, Q. Zhang, and Y. Chen, "Reinforcement learning and deep learning based lateral control for autonomous driving [application notes]," *IEEE Computational Intelligence Magazine*, vol. 14, no. 2, pp. 83–98, 2019.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] C. Godard, O. M. Aodha, M. Firman, and G. Brostow, "Digging into self-supervised monocular depth estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 3827–3837.

[14] Q. Khan, P. Wenzel, D. Cremers, and L. Leal-Taixé, "Towards generalizing sensorimotor control across weather conditions," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4497–4503.

[15] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[16] J. M. M. Mur-Artal, Raúl Montiel and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[17] J. L. Schönberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise View Selection for Unstructured Multi-View Stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[19] D. Wang and F. Q, "Trajectory planning for a four-wheel-steering vehicle," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[20] R. Rajamani, "Vehicle dynamics and control," in *Second Edition, Publisher: Springer*, 2012.

[21] Q. Khan, P. Wenzel, and D. Cremers, "Self-supervised steering angle prediction for vehicle control using visual odometry," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Banerjee and K. Fukumizu, Eds., vol. 130. PMLR, 13–15 Apr 2021, pp. 3781–3789. [Online]. Available: http://proceedings.mlr.press/v130/khan21a.html

[22] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[23] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[24] F. Codevilla, A. M. López, V. Koltun, and A. Dosovitskiy, "On offline evaluation of vision-based driving models," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 236–251.