Robust Autonomous Vehicle Pursuit without Expert Steering Labels

Jiaxin Pan*[1], Changyao Zhou*[1], Mariia Gladkova[1,4], Qadeer Khan[1,2] and Daniel Cremers[1,2,3,4]

***Abstract*—In this work, we present a learning method for both lateral and longitudinal motion control of an ego-vehicle for the task of vehicle pursuit. The car being controlled does not have a pre-defined route, rather it reactively adapts to follow a target vehicle while maintaining a safety distance. To train our model, we do not rely on steering labels recorded from an expert driver, but effectively leverage a classical controller as an offline label generation tool. In addition, we account for the errors in the predicted control values, which can lead to a loss of tracking and catastrophic crashes of the controlled vehicle. To this end, we propose an effective data augmentation approach, which allows to train a network that is capable of handling different views of the target vehicle. During the pursuit, the target vehicle is firstly localized using a Convolutional Neural Network. The network takes a single RGB image along with cars' velocities and estimates target vehicle's pose with respect to the ego-vehicle. This information is then fed to a Multi-Layer Perceptron, which regresses the control commands for the ego-vehicle, namely throttle and steering angle. We extensively validate our approach using the CARLA simulator on a wide range of terrains. Our method demonstrates real-time performance, robustness to different scenarios including unseen trajectories and high route completion. Project page containing code and multimedia can be publicly accessed here: https://changyaozhou.github.io/Autonomous-Vehicle-Pursuit/.**

***Index Terms*—Deep Learning Methods, Motion Control, Visual Tracking.**

THE latest technological advances in the field of autonomous driving have sparked a growing interest in developing more efficient and robust transportation solutions. In this regard, autonomous multi-vehicle platooning and convoying have appeared as a prominent research direction, which promotes various advantages over single vehicle operation such as improved traffic flow, reduced fuel consumption and improved overall transportation efficiency [1]–[4].

In this work, we focus on following a chosen vehicle and, therefore, refer to it as vehicle pursuit. This task can be defined as an imitation of the driving behavior of a specific target vehicle without overtaking or violating any traffic rules. In this case, the target vehicle guides the ego-vehicle across a wide spectrum of scenarios and, thus, enables autonomous operation of the ego-vehicle without human intervention. This way, vehicle automation allows to eliminate human errors and ensures precise trajectory control. Moreover, credibility of such methods can only be ascertained via deployment in actual scenarios. Nonetheless, this may be too risky to perform using hardware implementations as it involves direct interaction with the driving environment.

Autonomous driving simulators provide a viable alternative for the training and validation of new approaches for urban driving. Recent open-source platforms, such as [5], offer multi-sensor high-fidelity data to safely test and evaluate driving algorithms. In addition, recording various driving situations and rare corner cases is made feasible due to flexible and easy-to-control simulation. This has inspired us to choose this platform for the design and deployment of our method for the task of vehicle pursuit.

Specifically, we offer a deep learning method for estimating both the lateral and longitudinal steering commands (also referred to as control values) of an ego-vehicle, which pursues a desired target vehicle. Our network relies on a stream of images capturing the target vehicle and the velocities of both cars. The target vehicle is controlled by a human, whereas the ego vehicle is autonomous and controlled by our method. We handle a wide range of scenarios, such as sharp turns, roundabouts, as well as sudden velocity changes of the target car without any human expert supervision and labeling. The latter is achieved as we utilize model predictive control (MPC) for the generation of training labels, including throttle and steering angle. In comparison to other classical controllers, MPC has superior performance as shown by [6], [7].

It is important to mention that MPC is only applied offline to extract control labels for training and validation of the neural network and, thus, it is not required online during test time. As we follow a dynamic object, we deploy existing 3D object detectors to localize the target. We make our approach more robust to small inaccuracies in predicted steering commands by incorporating novel views into the training set, which are efficiently generated with the same sensor setup using dense depth maps and an image rendering approach. We demonstrate the full pipeline in Fig. 1.

Our main contributions can be summarised as follows:

- We propose a novel framework that enables autonomous vehicle pursuit without expert driver supervision. Specifically, during inference time our neural network operates in real-time and is capable of simultaneously predicting the commands for both longitudinal and lateral control from a single RGB camera and vehicle velocity streams.

Manuscript received: February, 24, 2023; Revised May, 26, 2023; Accepted August, 2, 2023.

This paper was recommended for publication by Editor Markus Vincze upon evaluation of the Associate Editor and Reviewers' comments.

* These authors contributed equally.

[1]All authors are with the Computer Vision Group, TU Munich, Germany {jiaxin.pan, changyao.zhou, mariia.gladkova, qadeer.khan, cremers}@tum.de

[2] Qadeer Khan and Daniel Cremers are with Munich Center for Machine Learning, Germany

[3] Daniel Cremers is with the University of Oxford, United Kingdom

[4] Mariia Gladkova and Daniel Cremers are with Munich Data Science Institute, Germany

Digital Object Identifier (DOI): see top of this page.

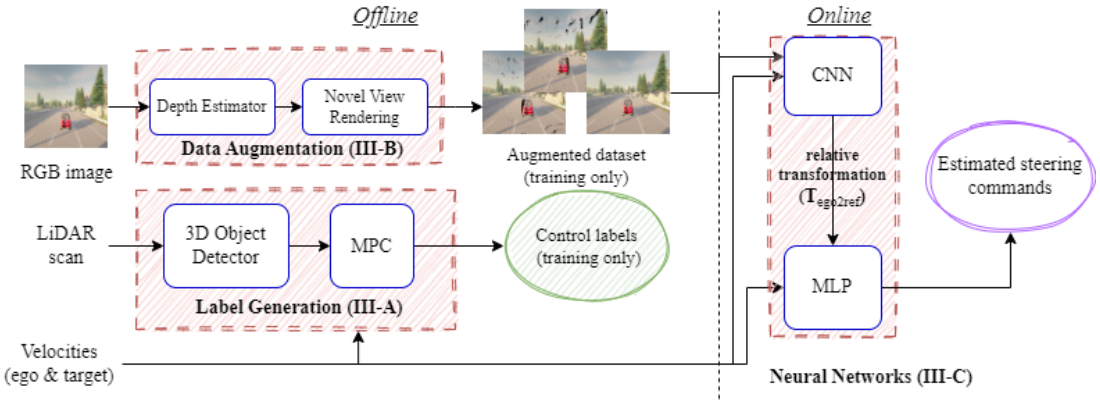

Fig. 1: **Overview of proposed framework:** Our proposed framework consists of preparatory offline steps (left) and online deployment (right), which are respectively done during training and testing of our system. *Left:* Firstly, we perform label generation using Model Predictive Control - MPC (Sec.II-A). MPC takes location and orientation of the target with respect to the ego-vehicle estimated by a LiDAR-based 3D detector. Next, data augmentation is done by utilizing estimated dense depth maps for novel view synthesis (Sec. II-B). *Right:* Our learning method is trained to predict accurate lateral and longitudinal control values by leveraging obtained labels, augmented image dataset and velocities of both ego and target vehicles (Sec. II-C). At test time, our approach only needs RGB image sequence and velocities as input for ego-vehicle control.

- We extensively test our method across a wide range of scenarios such as roundabouts, sharp turns and intersections using an autonomous driving simulator [5]. Our approach demonstrates higher tracking accuracy and robustness to variable conditions than the method inspired by [8].
- Training and evaluation code along with videos showing qualitative results is available here: https://changyaozhou.github.io/Autonomous-Vehicle-Pursuit/.

I. Related Work

There have been multiple proposed solutions for the task of car-following, which can be roughly classified into two categories: those based on traditional techniques using control and optimization theory and those that deploy a neural network or a learning-based algorithm.

Control-based methods. [9] involves a feedback controller to obtain desired velocity for steady platooning. Similarly, in [10] a kinematic model, describing the motion of vehicles, is created. Then a speed controller is designed to adjust the vehicle's longitudinal and lateral speed. Meanwhile, [2] focuses on energy-saving adaptive cruise control for eco-following. In our work, we leverage an MPC controller to collect training labels for our learning control method, which eliminates the necessity of manual annotation and recording of expert motion trajectories. [11] propose a two-layered control structure based on distance measurements and communication between the vehicles. In our case, we do not require a separate sensor for distance measurement. Rather, the distance and orientation are implicitly determined from an RGB image.

Learning-based approaches. [12] propose an end-to-end hierarchical deep neural network to follow a target pedestrian according to input images. In our case, vehicle pursuit poses an additional challenge due to the high and variable velocity of the target car. [13] designs a dual-task CNN, which simultaneously performs detection of the pursued object in 2D and semantic segmentation to determine drivable road regions. Based on the neural network estimates, a PID controller is utilized during inference time. In comparison, our approach neither requires semantic labels nor slow control optimization at inference time. We perform 3D object detection and MPC optimization offline and, thereby, ease the computational load at test time. [8] utilizes a Convolutional Neural Network (CNN) for the prediction of steering control values given a three-camera setup, where each camera is forward-facing and laterally displaced. This work is the closest to our method. Nonetheless, our method does not require a third camera as we effectively render novel camera views using dense depth maps, which also makes our method more robust to noisy disturbances. Several recent works focus on the driving task via imitation learning by following a pre-defined route as in TransFuser [14] or relying on a high-level control planner policy [15] that infers motion of all nearby traffic participants in a viewpoint-invariant manner. The ego-vehicle in our work, in contrast, follows a dynamic trajectory, which is unknown a-priori, and reactively adapts to the target car behavior. Moreover, our control method generalizes well to the viewpoint changes due to effective data augmentation. Furthermore, [14] and its extension [16] deploy an additional LiDAR sensor along with the desired goal position to predict the future waypoints of the vehicle at inference time. In our work, we only use RGB images to determine the position and orientation of the goal (target vehicle) at inference and do not require extra LiDAR sensor to be present in the vehicle setup. [17] proposes a image-based learning method for lateral control. Our work in contrast presents a solution for both lateral and longitudinal control.

II. Our Approach

In the next sections we provide a detailed overview of our proposed method for the car-following task as demonstrated in Fig. 1. Specifically, we describe three main steps of our pipeline: 1) label generation using Model Predictive Controller (MPC) and a 3D object detector (Sec. II-A); 2) data augmentation via image rendering from novel views (Sec. II-B); 3) two-stage neural network training to predict control values for the ego vehicle such as throttle and steering angle to successfully follow the target car (Sec. II-C).

A. Label Generation using MPC

Ground truth data generation is carried out in a two-stage manner. Firstly, we leverage a 3D object detector to localize the target vehicle given sensor data from the ego-vehicle (Sec. II-A1). Secondly, the obtained target pose together with velocity information is forwarded to the designed MPC algorithm in order to obtain optimal control values for the ego vehicle (Sec. II-A2). The label generation for car following can be seen as an alternative to an "expert driver", which also allows richer and more diverse supervision [18].

1) 3D Object Detection: In recent years, data-driven approaches have shown great performance for the task of 3D object localization in autonomous driving scenarios. Numerous camera-based [19], [20] and LiDAR-based methods [21], [22] have been proposed and extensively evaluated on various driving benchmarks. In this work, we leverage an off-the-shelf LiDAR-based 3D object detector and obtain 3D position $x \in \mathbb{R}^3$ and yaw angle θ of the target vehicle. In case of false positives, we prune the detections based on a simple, yet effective heuristic based on the distance to the ego-vehicle.

2) Model Predictive Controller: Model Predictive Control (MPC) is a method that is used to control a process while satisfying a set of constraints [23]. It takes a prediction horizon instead of a single time step into account and aims to get an optimal control result by minimizing the cost function within the prediction horizon. In the following, we describe the cost function and dynamic model designed for our vehicle-pursuit scenario.

Cost Function. We consider several factors when designing the cost function for optimization. For brevity, the variables associated with the ego and target (reference) vehicle are denoted with the *ego* and *ref* subscripts respectively.

To follow the target vehicle, the ego-vehicle must maintain a safe distance between the two vehicles to avoid a collision especially when a sudden application of brakes is applied by the target car. The safe distance d_{safe} is formulated in proportion to the velocity of the target vehicle $d_{\text{safe}} = L \cdot (1 + w_1 \cdot v_{ref})$, where L should be at least the length of the vehicle and $w_1 = 0.2$ is a scaling constant that is empirically chosen in our experiments. Then the ideal distance between two vehicles along the x- and y-axis, namely D_x and D_y, are computed respectively as

$$D_x = w_2 \cdot d_{\text{safe}} \cdot (\cos\theta_{ego} + \cos\theta_{ref}) \tag{1}$$

$$D_y = w_2 \cdot d_{\text{safe}} \cdot (\sin\theta_{ego} + \sin\theta_{ref}), \tag{2}$$

where θ_{ego}, θ_{ref} are yaw angles of the corresponding vehicles, $w_2 = 0.5$ is a scaling constant.

The *positional cost term* to be minimized during the optimization is $cost_x = \left|(x_{ref} - D_x) - x_{ego}\right|$ and $cost_y = \left|(y_{ref} - D_y) - y_{ego}\right|$ computed separately for x-and y-axis based on the ideal distances D_x and D_y provided above. Moreover, the yaw angle of both vehicles should be as close as possible. Thus, we introduce an *angle cost term* as $cost_\theta = (\theta_{ego} - \theta_{ref})^2$. Meanwhile, we assure that the estimated yaw angle θ_{ego} is within the range $[-\pi, \pi]$. The last component, *velocity cost term* of the cost function computed as $cost_v = w_3 \cdot \left|v_{ref} - v_{ego}\right|$ is based on the velocities of both vehicles, where a scaling constant $w_3 = 2$ is chosen empirically. The final optimization objective is defined as a sum of the aforementioned terms, namely the positional, angle and velocity cost terms.

In addition, we enforce a reasonable range of the lateral and longitudinal control values during optimization. Specifically, the range of throttle is [0, 1] and the range of steering angle is [-1, 1]. This range corresponds to the steering limits of the CARLA platform.

Vehicle Model. The model is used to update the state of the vehicle within the prediction horizon given the control values estimated by the optimization method.

Given the current state of the vehicle x_t, y_t, v_t and θ_t we therefore obtain its next state at time $t+1$ as

$$\begin{aligned} x_{t+1} &= x_t + v_t \cdot \cos\theta_t \cdot dt \qquad & v_{t+1} &= v_t + a_t \cdot dt - w_3 \cdot v_t \\ y_{t+1} &= y_t + v_t \cdot \sin\theta_t \cdot dt \qquad & \theta_{t+1} &= \theta_t + (v_t \cdot dt \cdot \Phi)/l, \end{aligned} \tag{3}$$

where l is the length of vehicle and $w_3 = 0.05$. a_t and Φ are proportional to throttle and steering angle respectively, which are optimized by the MPC.

With the updated states of the target and ego vehicles, the process repeats with a new iteration of cost function evaluation and optimal control values prediction. Despite the constant target velocity assumption in the optimization horizon, our full pipeline is robust to imperfect ground truth MPC labels and can handle sudden velocity changes of the target vehicle well.

B. Data Augmentation

Small inaccuracies in the estimated lateral and longitudinal control values can cause variation in the perceived sensory data. Accumulation of such errors can even make the ego-vehicle lose track of the target car and crash. Hence, we propose a data augmentation approach, which allows the network to recover from divergences from the route being followed, thus generalizing to unseen scenarios and disturbances. Specifically, we include off-trajectory samples into consideration. While the on-trajectory samples are collected when the ego vehicle follows the target vehicle perfectly, the off-trajectory samples should be generated by sampling positions around the ideal trajectory. The off-trajectory positions of the ego-vehicle are computed by applying uniform offsets within some ranges to the on-trajectory position including longitudinal, lateral, and rotational offsets.

1) Augmentation Steps: To generate images from the view of the ego-vehicle, which is located at an off-trajectory position, image rendering can be used. For this, we only utilize

on-trajectory RGB images and the corresponding dense depth maps. For depth map, we propose to utilize existing stereo-based depth estimation methods [24], [25].

Our data augmentation approach comprises the following steps. Firstly, we generate a local pointcloud by backprojecting every image pixel to the 3D camera coordinate frame given a dense depth map. Then we select an offset and transform the pointcloud from the on-trajectory ego-vehicle coordinate frame to the off-trajectory coordinate frame by applying a rigid body transformation $T_{ego}^{ego'} \in SE(3)$. Thirdly, the points are projected to the image plane by applying the inverse operation to backprojection. This way, we warp image pixels from one view onto another.

Although we introduce off-trajectory samples, we do not need to perform 3D object detection from Sec. II-A1 for every novel view. Instead, we can directly compute the relative transformation between off-trajectory ego vehicle coordinate and target vehicle coordinate according to Eq. (4) and use it as a training label for our neural network, where *ego* and *ego'* represent on- and off-trajectory ego vehicle respectively, $T_{ref}^{ego'} \in SE(3)$ represents a rigid transformation between the coordinate frame of the off-trajectory ego-vehicle and the target vehicle.

$$T_{ref}^{ego'} = T_{ego}^{ego'} \cdot T_{ref}^{ego} \tag{4}$$

T_{target}^{ego} can be extracted from the 3D object detector (Sec. II-A1), while $T_{ego}^{ego'}$ is computed according to the longitudinal, lateral, and rotational offsets to the on-trajectory ego-vehicle pose.

2) Sampling distances: In this section we explain the sampling distances at which additional off-trajectory camera views are generated, which makes our model robust to noisy control values. We synthesize novel views at positions sampled from a uniform distribution every 0.2 meters in the lateral x-direction for up-to 3 meters on either side of the on-trajectory data. In addition, we add small random noise to sampled points. Specifically, sampling noisy offsets x_{off} can be formulated as $x_{off} = k \cdot 0.2 - \frac{N}{10} + \varepsilon$, where $k \in \{0, 1 \dots N\}$ and $\varepsilon \sim \mathscr{U}(-0.05, 0.05)$. Meanwhile, for each of these lateral offset positions we sample longitudinal ($y_{off} \sim \mathscr{N}(0, 0.66)$) and rotational ($\theta_{off} \sim \mathscr{N}(0, 0.05)$) offsets. This way, 30 additional off-trajectory samples are collected at each frame. Given the offset values, a transformation matrix $T_{ego}^{ego'}$ between the off-trajectory vehicle and on-trajectory vehicle is computed.

Then the relative transformation between the coordinate frame of the off-trajectory ego-vehicle and the target vehicle $T_{ref}^{ego'}$ can be computed, as defined in Eq. (4).

As we render novel views with lateral displacement given a single depth map, void regions are inevitable to obtain due to limited field of view, especially at image borders. We postprocess our rendered images by applying central crop and resizing before feeding them to the network. Nonetheless, our method for car following is robust against the remaining void regions caused by occlusion or discontinuities in the predicted depth map. [26] showed that the sensorimotor control model focuses on high-level features such as lane markings, and pavement for immediate decision making. Thus, our processed rendered images still contain relevant information for the model to predict correct control commands. Visual examples can be seen on our Project Page: https://changyaozhou.github.io/Autonomous-Vehicle-Pursuit/.

It is worth noting that attempting to collect such novel views in real life by physically driving the car off-course may be too dangerous as the vehicle may invade lanes, sidewalks etc. thereby risking the safety of other road participants.

C. Neural Networks

After the data augmentation step discussed above, we can now train the models. As shown in Fig. 1 (right), the whole pipeline consists of two parts, that are trained separately:

1) A Convolutional Neural Network (CNN) model takes an RGB image from the ego and velocities from both vehicles as input and estimates the relative transformation between the two vehicles. Here we use the AlexNet [27] model pre-trained on ImageNet [28] as backbone. After flattening, we add linear layers to do regression for relative transformation.

2) A Multi-layer Perceptron (MLP) model takes the predicted relative transformation as well as velocities as input and predicts the longitudinal and lateral control values (throttle, steering angle) for the ego-vehicle. The MLP model consists of five successive units, each containing a linear layer with ReLU as the activation function, a batch normalization layer and a final extra linear layer.

We found that additionally providing velocity information of the two vehicles to the CNN model improves its performance. We believe this information facilitates providing scale information to the model for determining the metric transformation while given a single-camera RGB stream.

1) Training CNN: The inputs of the CNN model are an RGB image sequence captured by a monocular camera mounted on the ego vehicle and velocities for both vehicles detected by speed sensors. The training labels are relative transformations between two vehicles, which have been obtained from a 3D object detector (Sec. II-A1). A 6-layer MLP model is designed for training, combining ReLU activation and Batch normalization.

The CNN model is trained with both on-trajectory data from 9 trajectories, and off-trajectory data generated through image rendering. The collected dataset contains around 210k images, with 180k allocated for training and 30k for validation. The model is successively trained with three different learning rates for 100 epochs in total (1e-3 for 20 epochs, 1e-4 for 60 epochs, and 1e-6 for 20 epochs).

2) Training MLP: As shown in Fig. 1 relative transformations predicted by the CNN model along with the velocities of both vehicles are fed into an MLP model, which is used to predict the control values, namely the throttle and steering angle. The corresponding control values generated by the MPC controller (Sec. II-A2) in each frame are utilized as ground truth labels for training the MLP model.

The MLP model is only trained with on-track data from 6 trajectories. The size of the dataset is around 30k samples, from which 24k samples are prescribed for training and 6k for validation. We train the model for 350 epochs with the learning rate 1e-4 and 1e-6 successively.

After training, given the relative transformation predicted by the former CNN model and the velocities of both vehicles, the MLP model can predict reasonable control values, leading the ego vehicle to follow the target vehicle stably.

Note that both the CNN and MLP are trained using the Mean Squared Error (MSE) as the loss function.

III. Experiments

A. Experimental Setup

Algorithms for sensorimotor control involve online interaction with the environment. Unfortunately, common autonomous driving benchmarks do not provide an option to conduct online interaction. Therefore, we use the CARLA driving simulator (version 0.9.11) [5] for our experiments. CARLA has been widely adapted for online evaluation in control algorithms such as [13], [14], [16] due to a wide range of sensors that can be attached to the ego-vehicle for both data collection and inference, as well as having maps with different terrains and routes. It also furnishes precise ground truth labels for various tasks such as semantic segmentation, optical flow, dense depth, etc. Real-time vehicle data, such as global position, orientation, velocity etc. can also be extracted from the simulator. The ability of simulators to test vehicle pursuit on various scenarios is very pertinent for our experiments. Nonetheless, it is worth mentioning that driving simulators cannot fully reproduce the full range of interactions in the real world. Specifically, other vehicles and pedestrians operate in a controlled manner as they have been modeled to follow a certain behavior. Moreover, simulators cannot fully replicate the physical hardware as they rely on mathematical models and approximations. For instance, a simulator may not be able to model sensor degradation under different circumstances such as heavy rain in the case of LiDAR. Despite these disadvantages, we believe that simulators remain a valuable tool for our work as a cost-effective and flexible alternative to the hardware implementations.

Data Collection. Data were collected from 9 trajectories across Towns 03 and 04. The target vehicle is run on autopilot by the Traffic Manager. Meanwhile, the ego vehicle is controlled by an MPC algorithm to follow the target while maintaining an appropriate safety distance. LiDAR and RGB sensors are placed on the ego-vehicle. We use the approach from [22] to determine the 3D position and orientation of the target vehicle using LiDAR data. This information is in turn used by the MPC to determine the appropriate throttle and steering angle values as labels for training the neural network. At data collection, the RGB images are of size 1200 x 352 which are, firstly, center-cropped to 600 x 350 to remove the void regions and then resized to 128 x 128 before being fed to the CNN. Note that at inference time, only the RGB camera is used. The only information that we assume is given is the speed of the target and ego-vehicle. This can easily be read from a car's speedometer measurements.

The overall performance is evaluated on 5 trajectories from Towns 03 and 04 that are to a large extent different from the routes collected during the training phase. Since we noticed their partial overlap with the training routes, we further test the generalization capability of our method by adding 5 trajectories from two completely unseen maps, i.e. Town 01 and 05. While we describe quantitative results in Sec. III-C, we also provide videos that show qualitative performance of our model on different scenarios: https://changyaozhou.github.io/Autonomous-Vehicle-Pursuit/.

B. Metrics

To quantitatively evaluate the performance of vehicle pursuit, we adapt the Infraction Count (IC) and Route Completion (RC) metrics from [14]. Higher IC and RC values indicate better system performance. Specifically for our application of vehicle pursuit, we introduce two additional metrics, namely Mean Translation Error (MTE) and Control Difference (CD). MTE captures how accurately the ego-vehicle can replicate trajectories of the target vehicle. CD represents the accuracy of the model in estimating the control values at each matching point. In order to compute Mean Translational Error values, we first find corresponding matching points in the ego and target trajectories using bipartite matching with the Hungarian algorithm [29]. Formally, we define MTE as

$$MTE = \frac{1}{N}\sum_{i=1}^{N} ||T_i||_2^2, \tag{5}$$

where N is the number of matching points and $||T_i||_2$ is the Euclidean norm of relative translational error at point i. Control Difference can be computed with

$$CD = \frac{1}{N}\sum_{i=1}^{N} \sqrt{(a_{ego_i} - a_{ref_i})^2 + (\delta_{ego_i} - \delta_{ref_i})^2}, \tag{6}$$

where a_{ego_i} and a_{ref_i} are predicted and target vehicle throttle, δ_{ego_i} and δ_{ref_i} are predicted and target vehicle steering angle at point i respectively.

C. Quantitative Results

TABLE I: Performance of different methods.

Models	RC ↑	IC ↑	MTE ↓	CD ↓
Baseline	36.87	0.01	2.90	0.29
Three-camera [8]	40.09	0.11	2.15	0.48
Our approach (SS Depth)[a]	90.72	**0.49**	**0.61**	**0.21**
Our approach (Stereo Depth)	**90.88**	0.41	0.80	0.24
GT Depth+ GT Transformation (Oracle)[b]	92.90	0.42	0.57	0.17

[a] SS = Self-Supervised; [b] GT = Ground Truth;

Using metrics introduced in Sec. III-B, we evaluate our approach against different approaches, described in detail in the following subsections. As can be seen in Table I, our final approach outperforms all other methods and achieves performance comparable to that of the oracle. This is despite the oracle being trained with ground truth data. Table I also demonstrates that our pipeline is not dependent on one particular approach for depth prediction. It works equally well for both the self-supervised neural depth estimation network [24] and the stereo based classical approach [25]. Moreover, in case of the latter approach, our network is capable of handling inaccurate or noisy depth estimates. We now discuss details of the various model configurations described in Table I.

1) Baseline: This is the simplest approach with the model being trained on the on-trajectory images only. As shown in in Table I, this method has the lowest performance. This is because, at inference time, the model may cause the ego-vehicle to diverge from its driving lane, resulting in the vehicle being exposed to laterally displaced images. Since the model is only trained on on-trajectory images, it would not be able to correct off-trajectory motion and bring itself back on track. Accumulated displacements will cause the ego-vehicle to diverge and eventually lose track or crash.

2) Three-camera Method: The performance of the baseline model can be improved by collecting off-trajectory images by mounting multiple cameras at different positions on the ego vehicle. In this model configuration, we adapt the strategy of [8] by mounting three cameras on the ego vehicle, where each camera is placed with a 0.5-meter lateral displacement. The middle camera would capture on-trajectory images and the other two would capture off-trajectory images. Thus, this model is trained with three times more samples than the baseline model.

Although the performance of this model in terms of RC and IC is better than the baseline model, the improvement is only marginal. The reason is that the images from the additional cameras are only at an offset of 0.5 m from the on-trajectory position. Therefore, if the ego-vehicle diverges beyond this limit at inference time, it will have difficulty recovering. A solution for further improvement could be to place the additional cameras at further distances. But this is difficult, as the distance between the cameras is constrained by the width/dimensions of the car. Moreover, adding more cameras could lead to higher costs and synchronisation challenges.

3) Our approach (Self-Supervised / Stereo Depth): We demonstrate the performance of our method described in Sec. II with different depth estimation methods, which affect the quality of rendered off-trajectory images. In particular, we consider depth maps from a self-supervised network [24] and from a classical Semi-Global Block Matching (SGBM) method [25]. To determine relative transformation of the target vehicle with respect to the ego-vehicle we use the pre-trained model of [22].

As shown in Table I, the performance of our model is far superior to both the baseline and three-camera models. Meanwhile, its performance is comparable to the oracle. Due to image rendering, we are able to provide our network with significantly higher number of off-trajectory views than the baseline or three-camera methods. At the same time, our solution does not require additional sensors or the car to physically move off-trajectory. This can be viewed as the strength of our method and attributed to high performance against other approaches.

4) Ground Truth Depth + Ground Truth Transformation (Oracle): Here, the ground truth depth and ground truth relative transformation of the target vehicle collected from the CARLA simulator are used to train the model. Since the method fully relies on the ground truth measurements, it is defined as the Oracle. Nonetheless, based on Table I, the performance of this model is marginally better than our method, despite the fact that our approach is trained without ground truth labels.

D. Additional Experiments

We conduct an ablation study and further experiments to elaborate and assess different aspects of our method.

1) Impact of Involving Ground Truth (GT) Data: As mentioned in Sec. III-C4, we view the model trained with GT depth map and GT transformation as the Oracle. However, the GT data (collected in the CARLA simulator) might not always be accessible. Therefore, we perform additional experiments and investigate the extent to which the performance of our model is affected if the GT measurements are not available during training. Two variations for "GT Depth" and "GT transformation" models in Table I are adopted.

Specifically, "GT Depth + 3D Object Detector" utilizes ground truth dense depth maps from the CARLA simulator for image rendering. Meanwhile "SS Depth + GT Transformation" uses ground truth position and orientation of the target vehicle from the CARLA simulator. This information is then directly used for the control label generation using MPC as described in Sec. II-A2.

The first 4 rows of Table II show the evaluation results of 4 models trained with and without ground truth depth map and ground truth transformation, where the performance is compared to the Oracle. "GT Depth + 3D Object Detector" model demonstrates that using 3D object detections instead of the ground truth transformation labels for training the CNN module does not negatively influence the performance. Moreover, the performance of the "SS Depth + GT Transformation" model also indicates that the depth map predicted by the self-supervised model is sufficiently accurate and compares well with the GT depth map. Thus, the images rendered using self-supervised depth maps would look similar to those rendered using GT depth. In the last two rows of Table II, we investigate the performance of the transformation (CNN) and control (MLP) modules independently. To test for the MLP, we use the actual transformation labels obtained directly from the simulator as input, instead of the transformation labels predicted by the CNN. Likewise, to test for the CNN, we feed the predicted transformation output to an MPC controller, instead of the MLP. The results show that the MLP behaves very well when provided with precise transformation labels. It reaches 100 RC score, implying that the gap in RC performance likely comes from the limitations of the CNN

TABLE II: Evaluation results of methods with or without ground truth data.

Models	RC ↑	IC ↑	MTE ↓	CD ↓
GT Depth+GT transformation[a]	**92.90**	0.42	0.57	**0.17**
GT Depth+3D Detector	91.13	0.46	0.62	0.19
SS Depth+GT transformation	90.42	**0.51**	**0.54**	0.19
Our approach (SS Depth+3D Detector)[b]	90.72	0.49	0.61	0.21
Our approach (with T labels)[c]	100	0.19	0.85	0.10
Our approach (with MPC controller)	91.30	0.13	0.78	0.13

[a] GT = Ground Truth; [b] SS = Self-Supervised; [c] T = Transformation;

architecture which can possibly be reduced in the future by using a more powerful architecture. However, note that the IC and MTE metrics are worse off. Similarly, replacing the MLP with MPC does not show any significant performance enhancement. This demonstrates that the MLP is already achieving performance on par with the MPC, given the noisy labels from the transformation module.

2) Choice of Point Cloud Source for 3D Object Detection: As described in Sec. II-A1 our method utilises point clouds from the LiDAR sensor as the input to the object detection algorithm for determining the state of the target vehicle. MPC then utilizes this state to generate the training labels for the CNN network. In this experiment, we investigate the accuracy of 3D object detection using point cloud data from different sources other than LiDAR such as stereo cameras. For this, we utilize the same pre-trained 3D object detector [22] and test it on both LiDAR scans (Ours) and point clouds generated from stereo images. In particular, we consider two stereo approaches, namely SGBM [25] and CDN [24]. As it can be observed in Table III, LiDAR scans are more accurate for the 3D object detection task when compared to the image-based depth prediction methods. Therefore, it validates our choice of LiDAR point cloud as inaccuracies in 3D detection would significantly deteriorate the control labels from the MPC for our training. It is however important to note that due to the sparsity of LiDAR point clouds, they cannot be used to synthesize images. Hence, image-based depth estimation methods are needed for image generation as elaborated in Section II-B

TABLE III: Mean Squared Error of Object Detection with Different Point Cloud Sources

Point Cloud Source	MSE_x[a]↓	MSE_y[b]↓	MSE_θ[c]↓
SGBM [25]	1.50	3.00	0.24
CDN [24]	4.89	3.97	0.77
LiDAR (Ours)	**0.18**	**0.05**	**0.04**

[a] lateral MSE [b] longitudinal MSE [c] rotational MSE

3) Performance on Different Levels of Perturbation: We also tested the stability and robustness of our model against the baseline by applying external force with different intensity levels in random directions to the ego vehicle as perturbations. The external force can be according to $F_{perturb} = 2 \cdot L \cdot m_{ego}$, where L represents the level of perturbation and m_{ego} is the mass of the ego vehicle.

The external forces are added to the ego vehicle for three continuous frames every 40 frames. Applying such perturbation would cause the vehicle to diverge from its normal path. The subsequent 37 frames allow the model to recover from this divergence thereby leading the vehicle back to follow the target vehicle again. As shown in Fig. 2, the route completion of our method remains almost unchanged despite the increasing level of perturbation. However, the performance of the baseline model decreases dramatically. This experiment demonstrates that our model trained with rendered off-trajectory images is capable to deal with unexpected perturbations.

4) Performance in Different Target Velocity and Acceleration ranges: In our method, the target vehicle velocity and acceleration are crucial variables for successful vehicle pursuit. The ego-vehicle being controlled should closely match the target vehicle's velocity and acceleration. This means that the velocity of the ego-vehicle should rise (or drop) with the increase (or decrease) in the target vehicle's velocity. Otherwise, the ego-vehicle would either not be able to keep up with the target or collide into it.

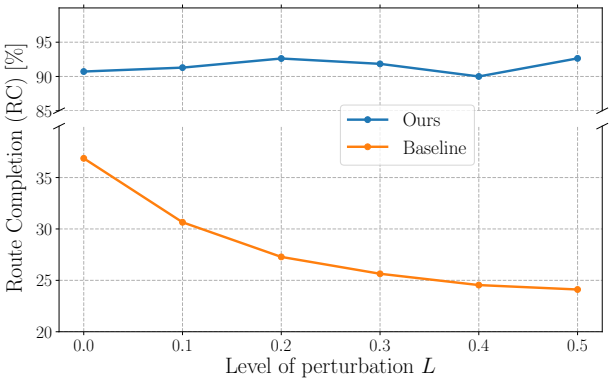

Fig. 2: Average Route Completion (RC) under different perturbation levels. Unlike the baseline, our model maintains high accuracy and consistency with increasing perturbations.

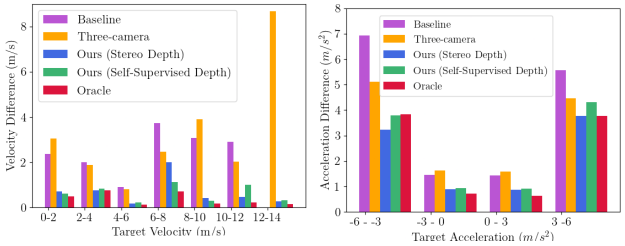

Fig. 3: The mean velocity (left) and acceleration (right) difference between target and ego vehicle for different ranges.

To assess the adaptability of all models to different target velocity and acceleration ranges, we calculate the mean velocity/acceleration difference between the target and ego vehicles within different ranges. Hence, the locations of target and ego vehicles over each trajectory are first matched with each other by the closest distance, then the velocity/acceleration differences are computed. As seen in Fig. 3, both our models with Self-Supervised and Stereo Depth show a lower velocity/acceleration difference in comparison to the Baseline and Three camera model. In fact, they demonstrate performance comparable to that of the Oracle.

5) Computational Analysis: We further test the runtime performance of our model during inference (online performance). When tested on a low-end GPU GeForce MX250 with 384 cores, our method achieves an inference speed of 42 frames per seconds (fps), which can be considered for a real-time application system. In contrast, running the object detection method and MPC optimization at inference time would lead to a slower runtime. In fact, the 3D object detector alone yields only 1.5 fps.

IV. Limitations

We assume that we are only following one target vehicle which is seen at all times. If the target vehicle briefly moves beyond the field of view of the camera or is completely occluded by another vehicle then the model would struggle to follow the target even if it appears back later. An option to address this in future work is to integrate tracking based Vehicle Re-Identification techniques from works such as [30]. For this, our architecture would need to be slightly modified to take in the template of the target vehicle too. Moreover, with this solution, a single target vehicle to be followed can be selected among the multiple vehicles in the scene. Another limitation is that we assume no uncertainty or drop in the communication of velocity value between two vehicles. A solution to the noisy or missing target vehicle velocity reading is to apply approaches such as the Kalman filter [31].

V. Conclusions

In this paper, we proposed a car-following framework for training a network without the need for supervised steering labels. The steering labels are implicitly determined from model predictive control. This is done in conjunction with applying a 3D object detector to extract the relative location and orientation of the target vehicle with respect to the ego-vehicle. Given a single RGB image and velocities of both vehicles, we train a two-stage network comprising of a CNN and an MLP to estimate optimal lateral and longitudinal control values for the ego-vehicle while it performs the task of following the target. Meanwhile, additional off-trajectory images are rendered and included in the training data to enhance the robustness of the model to inaccuracies in estimated control commands. We extensively test our method on the CARLA simulator and show the effectiveness of our pipeline. Additional experiments are performed to quantitatively verify robustness and computational efficiency of our method.

Although the emphasis of the work was on the car-following task only, we believe the proposed approach can be extended to a multi-vehicle platooning application. We believe our work can serve as an intermediate step from vehicle pursuit to a fully autonomous driving and inspire future work in the same direction.

References

[1] Q. Li, Z. Chen, and X. Li, "A review of connected and automated vehicle platoon merging and splitting operations," *T-ITS*, 2022.

[2] V. Turri, Y. Kim, J. Guanetti, K. H. Johansson, and F. Borrelli, "A model predictive controller for non-cooperative eco-platooning," in *2017 American Control Conference (ACC)*. IEEE, 2017.

[3] S. E. Li, Y. Zheng, K. Li, L.-Y. Wang, and H. Zhang, "Platoon control of connected vehicles from a networked control perspective: Literature review, component modeling, and controller synthesis," *IEEE Transactions on Vehicular Technology*, 2017.

[4] W. Zhuang, L. Xu, and G. Yin, "Robust cooperative control of multiple autonomous vehicles for platoon formation considering parameter uncertainties," *Automotive Innovation*, vol. 3, pp. 88–100, 2020.

[5] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[6] B. Varma, N. Swamy, and S. Mukherjee, "Trajectory tracking of autonomous vehicles using different control techniques(pid vs lqr vs mpc)," in *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, 2020, pp. 84–89.

[7] M. Samuel, M. Mohamad, M. Hussein, and S. M. Saad, "Lane keeping maneuvers using proportional integral derivative (pid) and model predictive control (mpc)," *Journal of Robotics and Control (JRC)*, vol. 2, no. 2, pp. 78–82, 2021.

[8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint:1604.07316*, 2016.

[9] Z. Xu, J. Jiang, and Y. Liu, "Experimental research of vehicle-platoon coordination control based on torcs platform," in *2016 35th Chinese Control Conference (CCC)*, 2016.

[10] Y. Shi, T. Li, and Q. Han, "An adaptive car-following strategy for vehicle platooning control," in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2021, pp. 1129–1134.

[11] O. Gehring and H. Fritz, "Practical results of a longitudinal control concept for truck platooning with vehicle to vehicle communication," in *ITSC*, 1997, pp. 117–122.

[12] J. E. Solomon and F. Charette, "A follow-the-leader strategy using hierarchical deep neural networks with grouped convolutions," *SN Computer Science*, vol. 2, no. 3, pp. 1–9, 2021.

[13] P. Jahoda, J. Cech, and J. Matas, "Autonomous car chasing," in *European Conference on Computer Vision*. Springer, 2020.

[14] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[15] D. Chen and P. Krähenbühl, "Learning from all vehicles," in *CVPR*, 2022, pp. 17 222–17 231.

[16] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with transformer-based sensor fusion for autonomous driving," *PAMI*, 2022.

[17] Q. Khan, I. Sülö, M. Öcal, and D. Cremers, "Learning vision based autonomous lateral vehicle control without supervision," *Springer Nature Applied Intelligence*, 2023.

[18] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning*. PMLR, 2020.

[19] P. Li, X. Chen, and S. Shen, "Stereo r-cnn based 3d object detection for autonomous driving," in *CVPR*, 2019, pp. 7644–7652.

[20] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *CVPR*, 2018.

[21] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *CVPR*, 2019, pp. 12 697–12 705.

[22] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *CVPR*, June 2019.

[23] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[24] D. Garg, Y. Wang, B. Hariharan, M. Campbell, K. Weinberger, and W.-L. Chao, "Wasserstein distances for stereo disparity estimation," in *NeurIPS*, 2020.

[25] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *PAMI*, 2007.

[26] Q. Khan, P. Wenzel, D. Cremers, and L. Leal-Taixé, "Towards generalizing sensorimotor control across weather conditions," in *IROS*, 2019, pp. 4497–4503.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[29] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[30] S. D. Khan and H. Ullah, "A survey of advances in vision-based vehicle re-identification," *Computer Vision and Image Understanding*, vol. 182, pp. 50–63, 2019.

[31] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, pp. 35–45, 1960.