VTGNet: A Vision-based Trajectory Generation Network for Autonomous Vehicles in Urban Environments

Peide Cai, Yuxiang Sun, Hengli Wang and Ming Liu, Senior Member, IEEE

Abstract—Reliable navigation like expert human drivers in urban environments is a critical capability for autonomous vehicles. Traditional methods for autonomous driving are implemented with many building blocks from perception, planning and control, making them difficult to generalize to varied scenarios due to complex assumptions and interdependencies. In this paper, we develop an end-to-end trajectory generation method based on imitation learning. It can extract spatiotemporal features from the front-view camera images for scene understanding, then generate collision-free trajectories several seconds into the future. The proposed network consists of three sub-networks, which are selectively activated for three common driving tasks: keep straight, turn left and turn right. The experimental results suggest that under various weather and lighting conditions, our network can reliably generate trajectories in different urban environments, such as turning at intersections and slowing down for collision avoidance. Furthermore, by integrating the proposed network into a navigation system, good generalization performance is presented in an unseen simulated world for autonomous driving on different types of vehicles, such as cars and trucks.

Index Terms—End-to-end driving model, imitation learning, autonomous vehicle navigation, path planning.

I. INTRODUCTION

FROM a global perspective, approximately 1.3 million people die yearly due to road traffic [1], and nearly 94% of them are related to human driving errors [2]. It is shown that autonomous vehicles (AVs) may play an essential role in reducing this amount, and can also alleviate traffic congestion, cut down air pollution and reduce energy consumption in transportation by as much as 90% [3].

In order to achieve autonomous driving, vehicles need to perceive and understand the surroundings independently, then use the extracted information to generate collision-free trajectories to the goal position. However, it is still a challenge to enable AVs to mimic the capabilities of human drivers, especially in real-world urban environments where trajectory generation is a crucial task. Within this framework, the solution trajectory can be represented as a time-parametrized function \( \pi(t) : [0, T] \rightarrow \mathcal{X} \), where \( T \) is the planning horizon and \( \mathcal{X} \) is the configuration space of the vehicle [4]. Basically, methods for planning and decision-making for AVs can be divided into three categories: traditional sequential planning, end-to-end control and end-to-end planning, which are illustrated in Fig. 1.

Peide Cai, Yuxiang Sun, Hengli Wang and Ming Liu are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong SAR, China (e-mail: pcaiaa@connect.ust.hk; sun.yuxiang@outlook.com; hwangdf@connect.ust.hk; eelium@ust.hk)
A vision-based trajectory generation network (Fig. 2) for autonomous vehicles based on imitation learning, which requires less prior information than traditional approaches.

- Extensive experiments and the presented reliability of VTGNet after comparing it with different baselines on different driving occasions (e.g., turning at intersections, and slowing down for collision avoidance) under various weather and lighting conditions (e.g., snowy, rainy, and at night).

- Generalizable driving capability on different types of vehicles demonstrated in unseen high-fidelity simulated environments that are different from the training domain.

The rest of this paper is structured as follows. In Section II, we review the related work of autonomous driving on learning-based methods for planning and control. In Section III, we present the details of the proposed network and the composition of our dataset. Experimental setups, results and discussions are shown in Section IV and V. Finally, we conclude this paper in Section VI.

II. RELATED WORK

A. End-to-end Control

The ALVINN proposed by Pomerleau [17] in 1989 is a pioneer attempt to use the neural network for autonomous driving. Due to the composition of a limited number of neural network layers, this method only works in very simple scenarios. In 2016, with the development of convolutional neural networks (CNNs) and computing powers of GPUs, Bojarski et al. [8] developed a more advanced driving model named DAVE-2. It achieves autonomous lane following in relatively simple real-world scenarios, such as flat or barrier-free roads, in which the front-view camera is used to stream the images and transmit them into CNNs to compute steering commands. This method was later refined by Jhung et al. in their proposed DAVE-2SKY network [9]. It consists of supervised pre-training of the driving network and reinforced post-training to refine the performance. Followup works include [13] and [18].

However, the works mentioned above only target lane-following tasks. Since they only consider the camera input for decision making, a wrong turn may be taken at intersections for the lack of high-level navigation commands. In order to resolve such ambiguities, driving models proposed by Codevilla et al. [12] and Gao et al. [11] have been designed to receive not only the perceptual inputs but also the driving intentions. In this way, the network is more controllable. Similarly, Hecker et al. [19] trained a driving model to use route information and 360-degree camera images to predict future steering and speed controls.

B. End-to-end Planning

To make the driving model more generic, new methods have been recently proposed for end-to-end planning, which leaves the control component outside the neural network. Bergqvist et al. [20] design and test several path planning networks with various types of input, such as gray-scale images, past egomotions, detections of surrounding objects and lane marking estimations. All models are trained with imitation learning, and the results show that the path generated by long short-term memory (LSTM) or CNN-LSTM is smooth and feasible in many situations. However, this work only considers the lanekeeping task. Differently, we add the driving intentions in our model to make it more applicable.

Bansal et al. [21] propose to use overhead environmental representations to generate trajectories with recurrent neural networks. Then a control module is designed to translate the trajectories into steering and acceleration actions. However, the detailed top-down views used in this work are expensive to create, maintain and transfer, which relies on prebuilt high-resolution maps of the driving areas. Similarly, Rhinehart et al. [22] consider the future trajectories as a distribution conditioned on overhead feature maps, from which a set of possible paths can be sampled for driving tasks. Different
from these works, the highly-engineered feature maps are not needed in our proposed VTGNet, which requires less prior information of the environments and can achieve both scene understanding and trajectory generation in an end-to-end framework. Furthermore, the generalization performance of our method is also evaluated on different types of vehicles, which has seldom been considered in prior related works.

III. METHODOLOGY

In this work, the trajectory generation problem is solved by an end-to-end network using imitation learning. The goal is to avoid the laborious hand engineering in traditional modular pipelines and make the generator generalizable to different environmental conditions. Therefore, we design a novel network architecture using CNNs and LSTMs, and train it on a large-scale urban driving dataset.

A. Network Architecture

The architecture of the proposed VTGNet is shown in Fig. 2. At each time step $t$, the inputs to the network are camera images $I_t = \{I_1, ..., I_{12}\}$ and the movement information of the vehicle $m_t = \{m_1, ..., m_{12}\}$ in the past 12 frames. The output of the network is the predicted trajectory $T_t$ conditioned on the input observations and the high-level driving command $c_t$. To ensure safety, the preview horizon of the generated trajectory is set to 3s (i.e., 22 frames), which is double the average human perception-brake reaction time (RT) of 1.5s to stop a vehicle. Therefore, the generated trajectory can be denoted as $T_t = \{T^1, ..., T^{22}\}$. Here, $m_n \in \mathbb{R}^3$ and $T_n \in \mathbb{R}^3$ both contain the velocity and x,y positions in the current body frame.

For implementation, we construct three sub-networks that can be selected by the given command $c_t$ to conduct different tasks, i.e., turn left, turn right and keep straight, which is similar to some other works [12, 24-26]. Once a specific branch is chosen, the images $I_t$ are first processed separately and then turn an image module $F_t$ implemented with CNNs, which extracts a feature vector of length 512 from each image.

Then, a balance module $F_b$ expands the dimension of each history movement vector $m^k$ from 3 to 128 to balance the influence of the vision and motion feature vectors, where we draw on the experience of [11]. The outputs of these two modules are then concatenated together at every time step into joint vectors of length 640, represented by

$$ J^n = < F_i(I^n), F_b(m^n) >, 1 \leq n \leq 12, \quad (1) $$

where $< \cdot , \cdot >$ represents the concatenation operation, $F_i(I^n)$ is the output of the image module for the n-th image, $F_i(m^n)$ is the output of the balance module for the n-th motion state vector, and $J^n$ is the n-th joint vector. Then the set of $J = \{J^1, ..., J^{12}\}$ is processed by an LSTM network to generate the spatiotemporal feature $\in \mathbb{R}^{256}$ of the surroundings. It is further compressed by 2 fully connected layers to get the final vector of size $22 \times 3$, which represents the trajectory $T_t$ generated at time step $t$ in the future 22 frames.

In general, we can write the function $F$ of the trained trajectory generation network VTGNet as follows.

$$ T_t = F(I_t, m_t, c_t) = F(\{I^1, ..., I^{12}\}, \{m^1, ..., m^{12}\}, c_t). \quad (2) $$

B. Network Details

1) The Feature Extractor: We use MobileNet V2 [14] as the image module $F_i$ to extract visual features. It consists of 17 bottleneck convolutional blocks and each block uses the depthwise separable convolutions rather than a fully convolutional operator. As shown in Fig. 3, this convolution operator consists of three separate layers. The first layer uses the pointwise convolution with $1 \times 1$ kernels to expand the input features to a higher dimension. The second layer is a depthwise convolution, which applies a single convolutional filter per input channel. The third layer is another pointwise convolution to build new output features. For each convolution block, if the number of channels of the input and output features are equal, and the stride of the depthwise convolution is 1, the residual connect will be used, as shown in Fig. 3(a).

2) The Trajectory Generator: We build an LSTM module that consists of three recurrent layers, which means that three LSTM layers are stacked together and each LSTM receives the output of the upper one. The final results are generated by the third layer. In addition, the number of features in the hidden state of the LSTMs is set to 256.

3) Loss Function: As the name suggests, the basic idea of imitation learning is to train a network that mimics human behaviors, which can be solved with supervised learning. Let $T'_t$ denote the expert-provided trajectory in the training dataset at sample time $t$, and $\theta$ denote the learnable parameters of the network $F$. Then the optimal parameters $\theta^*$ can be trained by minimizing the average prediction loss:

$$ \theta^* = \arg \min_{\theta} \frac{1}{T_t} \sum_{t} \mathcal{L}(F(I_t, m_t, c_t; \theta), T'_t), \quad (3) $$
TABLE I
THE ENVIRONMENTAL DISTRIBUTIONS OF OUR DATASET

| Environments | # Turn Left | # Turn Right | # Keep Straight |
|--------------|-------------|--------------|-----------------|
| Sun          | 3705        | 5355         | 7921            |
| Rain         | 3369        | 4401         | 16548           |
| Snow         | 1280        | 3975         | 6089            |
| Dusk         | 1059        | 1194         | 4836            |
| Night        | 2339        | 2705         | 10930           |
| Overcast     | 2733        | 3103         | 7016            |

where $\mathcal{L}$ is the per-sample loss at sample $t$. It is defined as the L2-norm between the prediction and the ground truth:

$$\mathcal{L} = \|T_t - T'_t\|^2.$$ (4)

C. Dataset Generation and Analysis

We build our dataset from the RobotCar dataset, which was recorded in dynamic urban areas from May 6, 2014 to November 13, 2015. This dataset captures scenarios with various weather and lighting conditions, along with some long-term changes such as construction and roadworks. The front-view images are recorded by a Point Grey Bumblebee XB3 camera on top of the vehicle. In addition, the ground-truth position and velocity information can be acquired from the fused GPS/inertial solution, which are recorded at a frequency of 50 Hz.

1) Data Generation: From the RobotCar dataset, we aim to extract both camera images and movement information to train and test VTGNet. Therefore, the data without GPS information is first filtered out and a total of 29 driving routes are collected. Another issue that needs to be considered is data balance. Specifically, we need to balance the portion for different driving cases (e.g., car following, slowing down for collision avoidance, etc.) and the distribution of driving samples in varied environmental conditions. The final dataset distribution is shown in Fig. 4 and Tab. I. Then this data is reconstructed for training and testing in this work, where we need to equip every image with the trajectory of the ego-vehicle in the past 1.5 seconds and the future 3 seconds. For implementation, we first interpolate the ground-truth universal transverse mercator (UTM) position and velocity series to the image timestamps, which are recorded at 15 Hz, and then convert their coordinates to the vehicle body frame. After that, we label every image with a corresponding intention command indicating the driving direction of the vehicle based on the ground-truth trajectories. Finally, we crop the raw images to the shape of (1247, 384) by removing the sky and hood on the top/bottom of the images because these areas are less informative for scene understanding. We believe that the learning process could be more efficient by doing so.

2) Data Analysis: Our final dataset contains 88,558 images of driving sequences in Oxford for 61.52 km, which covers 6 environmental conditions: overcast, sun, rain, snow, dusk and night. Each environment has a special style of visual appearance. For example, in the rain scenario, the roads are often covered by fallen leaves and sometimes raindrops on the camera lens cause blurry areas in the image. In the night scenario, the shutter speed is much slower than it is in the daytime, which can lead to motion artifacts around objects when the vehicle moves. In the snow scenario, the undrivable areas are often covered by white snow, contrasting sharply with the drivable areas. Sample visual appearances can be seen in Fig. 9. In addition, to visualize how the dataset distributes geologically, we plot their locations on an aerial map, as shown in Fig. 5.

To further analyze the distribution of the visual features and the difference between the training and testing sets, we embed a subset of our dataset into a two-dimensional space using t-distributed stochastic neighbor embedding (t-SNE) [27]. Fig. 7 (a) shows that the visual features basically distribute according to the environmental conditions. We can also observe in Fig. 7 (b) that the training and testing sets share similar feature distributions.
Fig. 6. Different baselines in this work. The feature extractor for these networks is the same as the one in our proposed VTGNet shown in Fig. 2. The size of the output features is shown above the layers.

Fig. 7. The distribution of the visual features from our dataset, calculated by the t-SNE method. (a) shows the distribution in the training set. (b) shows the difference between the training and test sets.

IV. EXPERIMENTAL SETUP

A. Training Details

We implement the proposed VTGNet with Pytorch under the environment of CUDA 10.0, and train it on our large-scale driving dataset introduced in Section III D with the NVIDIA 1080Ti graphics card. The split ratio of training, validation and test is set to 7:1:2. During training, we use the adaptive moment estimation (Adam) optimizer [28] with the initial learning rate of 0.0001 and batch size of 15. The network is trained to converge when no further decrease in the validation loss can be observed. For comparison, we also train and fine-tune four other baselines on the same training set. They are introduced as follows and are shown in Fig. 6.

CNN-FC. This network takes as input the image sequences. The CNNs first extract the visual features in the past 12 frames (1.5 s). Then these features are concatenated together into a vector $\mathbf{f}_I \in \mathbb{R}^{1024}$ to be compressed by 4 FC layers. This method follows the idea of TCNN introduced in [13].

CNN-LSTM. This network takes as input the image sequences. The extracted features in the past 12 frames are processed by a three-layer LSTM block into a vector $\mathbf{f}_L \in \mathbb{R}^{512}$. The vector is then compressed by 3 FC layers. This method follows the idea of CNN-LSTM introduced in [13].

CNNState-FC. This network takes as input both image and movement sequences. The extracted features in the past 12 frames are concatenated together into a vector $\mathbf{f}_I \in \mathbb{R}^{1024}$. In addition, the movement information is projected into a higher dimensional vector $\mathbf{f}_m \in \mathbb{R}^{128}$. Then $\mathbf{f}_I$ and $\mathbf{f}_m$ are compressed together by 4 FC layers. This method follows the idea of Merging Model introduced in [20].

DuoLSTM. This is a variant of our VTGNet. In this network, the image and movement sequences are first processed separately by two LSTM modules. The generated features are then compressed together by 4 FC layers. This variant is used to verify whether processing the visual and state features separately by LSTMs would benefit the performance.

B. Evaluation Metrics

We adopt 9 metrics to evaluate the performance of different networks. All of them are the average values computed over the entire test dataset.

- $T$ denotes the average time cost for a network to infer a trajectory.
- Accel and Jerk measure the smoothness of the generated trajectory. Accel is the average acceleration of the preview time, and Jerk is the average time derivative of the acceleration. The lower their values, the smoother and more comfortable the corresponding trajectories.
- IoU measures the accuracy of the network predictions. Based on the vehicle width, we expand the generated and ground-truth trajectories to two driving areas ($D, D'$). Then, this metric can be calculated as follows:

$$IoU = \frac{D \cap D'}{D \cup D'}$$

- $E_{ad}$ represents the average displacement error [29]. It is the L2 distance between the generated and the ground-truth trajectory.
- $E_{fd}$ is the final displacement error [29], which means the L2 distance between the final waypoints of the generated and ground-truth trajectories.
- $E_s$ and $E_v$ represents the lateral and longitudinal error, respectively, of a trajectory.
- $E_b$ measures the mean velocity error in the preview horizon of a trajectory.

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

A. Quantitative Analysis on Different Models

We first show the evaluation results on the test set from different model architectures in Tab. II. In the following, the analyses are given from three perspectives: time cost, trajectory smoothness and error.
which leads to better performance on this imitation learning state, such as environmental changes among the frames, We conjecture that the LSTM module can infer the hidden information directly from the consecutive visual features. On the other hand, the superiority of CNNState–FC indicates that past movement is beneficial for trajectory generation, which functions as the state memory, similar to the memory of humans. Although the position and orientation information is contained in the raw images, using a CNN alone is not sufficient to extract this information and generate future trajectories. Second, the DuoLSTM model presents a worse performance than VTGNet, not only for the error metrics but also for the trajectory smoothness. Therefore, we conclude that using a single LSTM module to process the multi-modal information together would help to generate more reasonable outputs. By contrast, separating this step into two parts by assigning two columns, it can be seen that the 5 models run at close inference speeds, in which the DuoLSTM is fastest. And the proposed VTGNet ranks second place with a slightly longer inference time than DuoLSTM of about 0.9 ms.

Trajectory smoothness: Tab. II reveals that the CNN–FC model generates the smoothest trajectories with the lowest values of Accel and Jerk (0.18/0.35). And VTGNet ranks second place with slightly higher values of these metrics (0.27/0.52). By contrast, the other models present poor performance in terms of the Jerk metric. The Jerk value from CNN–LSTM is 6.02, which is about 11 times higher than our VTGNet, indicating coarse trajectories from this model. The case is similar for DuoLSTM with the Jerk value of 5.4, which is much higher than that of VTGNet.

Error analyses: From the fourth to the last column of Tab. II we can see that VTGNet generates more accurate trajectories than the other models in terms of the IoU and error-related metrics such as $E_v$ and $E_{ad}$. This superior performance indicates that our model performs more like humans than the others. In addition, we can infer the reasonability of our network design by comparing the results from different models, which is discussed as follows.

First, Tab. II shows that both CNN–LSTM and CNNState–FC perform better than CNN–FC, with lower errors and higher IoU. The superiority of CNN–LSTM indicates that the recurrent architecture, such as LSTM, is much more efficient than the FC to process spatiotemporal information. We conjecture that the LSTM module can infer the hidden state, such as environmental changes among the frames, which leads to better performance on this imitation learning task. However, it is difficult for the FC to extract valuable information directly from the consecutive visual features. On the other hand, the superiority of CNNState–FC indicates that past movement is beneficial for trajectory generation, which functions as the state memory, similar to the memory of humans. Although the position and orientation information is contained in the raw images, using a CNN alone is not sufficient to extract this information and generate future trajectories.

Second, the DuoLSTM model presents a worse performance than VTGNet, not only for the error metrics but also for the trajectory smoothness. Therefore, we conclude that using a single LSTM module to process the multi-modal information together would help to generate more reasonable outputs. By contrast, separating this step into two parts by assigning two...
Fig. 9. Sample qualitative results for our model VTGNet in various environments. Columns (a) and (b) show the result for turn left and columns (e) and (f) for turn right. Column (c) shows the results for car following and column (d) for lane keeping. Note that the extra rainy night scenes are used to test the generalization capability, thus they are not included in the training set.

Fig. 10. Special cases for qualitative evaluation. Markers are interpreted the same as in Fig. 9. In column (a), our model can generate trajectories to bypass the obstacles ahead. In column (b), trajectories with slow velocities are generated to ensure safety when the ego-vehicle drives close to other road agents. In column (c), we present some failure cases in the daytime and nighttime, where the generated trajectories may crash into obstacles or road edges in some tough scenes with dim/bright lights.

B. Quantitative Analysis on Different Environments

In this section, we test and evaluate our VTGNet separately on different environments with various lighting and weather conditions, with the aim to see whether this architecture could generalize to different visual appearances. The results are shown in Tab. III. It can be seen that VTGNet achieves robust results under different conditions and performs best in the snow environment. In this scene the generated trajectories have the lowest acceleration (0.204 m/s²) and error results (Ev, Ead, etc.). This is probably because the drivable areas are more distinct in this scenario with the undrivable areas covered by white snow. The comparison of different environments is shown in Fig. 9.

C. Qualitative Analysis

Fig. 9 displays sample qualitative results of VTGNet in 4 typical environments, in which a challenging scene rainynight is presented in the last row. Because of the raindrops on the camera lens, the halo effects in this scene are sometimes conspicuous. It is worth noting that this scene is not included in the training set, and is only used to qualitatively test the generalization performance. In general, we can see that our VTGNet is able to generate collision-free trajectories under various lighting and weather conditions. The performance is also reliable in some tough scenarios. For example, in night-(f), the bus in the left area presents a severe motion artifact, and VTGNet generates a safe trajectory to turn right. Moreover, the generalization performance in the unseen challenging scene rainynight is also acceptable with a little lower IoU than other scenes. Furthermore, we show some collision-avoidance cases in Fig. 10(a,b). For example, in the first row of column recurrent blocks to process different inputs brings no benefit but degrades the performance. The reason may be that some hidden relations between the visual and movement information can only be well extracted by the LSTM using combined sequences.

Fig. 8 shows the displacement error over the 3.0 s preview horizon from different models. It can be seen that their errors and related error bounds all increase as the preview time increases, and our VTGNet achieves the best results with the smallest error and variance.
(b), the future velocity of the ego-vehicle slows down from 16.2 km/h to 7.2 km/h to follow a slow vehicle ahead.

We have discussed the performance on single frames above. In the following, we present demonstrations on whole sequences. We test VTGNet and two other baselines on 3 consecutive driving sequences covering varied environmental conditions, and the results are shown in Fig. 11. It can be seen that VTGNet generates trajectories closer to the ground truth with smaller and more stable displacement errors.

D. Further Experiments in an Unseen World

We further evaluate qualitatively the generalization ability of our model without any fine tuning in previously unseen environments from CARLA [30]. This is an open-source simulator providing a high-fidelity dynamic world and different vehicles with realistic physics. In this simulation, we set routes on three maps and test the online performance of VTGNet on two types of vehicles: a car (Audi TT) and a truck (Tesla CyberTruck). Note that the latter vehicle is much larger and harder to control than the former vehicle. Therefore, we design two PID controllers adapted to these vehicle properties to translate the generated trajectories to three kinds of actions: steering angle, throttle and brake. The test environments and vehicles are shown in Fig. 12. Finally, we show the test results in Fig. 13. It can be seen that the generated trajectories (red dots) are reasonable for lane following and turning at intersections in different environments such as the scenes of (b), (f) and (g). In addition, the vehicle is able to slow down for collision avoidance by reducing the throttle and applying more brake, such as the scenes of (h) and (i). More related results are shown in the supplemental videos.

Different from the generalization results in [12] and [31], the feasibility of our model is tested on different vehicles of real sizes with varied physical properties. We accredit this performance to our novel end-to-end architecture, which allows the control module to flexibly adapt to different vehicle platforms.

E. Reasoning and Limitations

Compared with the hand-crafted feature extractors, one advantage of CNNs is that they can learn to extract useful features automatically from raw RGB images. We confirm this idea by visualizing the averaged feature maps from a series of convolution layers of our model. The results are shown in Fig. 14. We can see that the obvious extracted features are the road boundaries, which are helpful when generating the trajectories.

Despite the success of the proposed network, there still exist some limitations in our model, which may mainly originate from the camera used. In the night environments, to capture sufficient light for acceptable visual appearance, the exposure time of the camera is automatically adjusted to be longer than it is in the daytime. However, such operation may lead to motion artifacts as the vehicle moves. In addition, vision-based methods rely on image quality, which is prone to be affected by illuminations, so our network might not perform well under too dim or too bright scenarios. As a consequence, in such scenarios, the trajectory generated by our VTGNet may not be reliable for the lower-level controllers to compute reasonable actions. Sample failure cases are shown in Fig. 10(c).

VI. CONCLUSION AND FUTURE WORK

In this paper, to avoid the disadvantages of error propagation and over-complexity in the traditional pipelines of autonomous navigation, we proposed a vision-based trajectory generation network for autonomous driving in urban environments, named VTGNet. For acceptable training and testing performance,
we first created a large-scale driving dataset from the original Robotcar dataset. Then, we implemented the proposed network with a twofold pipeline, where the first part is a feature extractor composed of bottleneck CNN layers based on MobileNet V2, and the second part is a trajectory generator, which consists of a 3-layer LSTM module to process the spatiotemporal features and 2 FC layers. This framework was designed to take as input the front-view image and movement sequences in the past 1.5 seconds with the intention command to generate feasible trajectories 3 seconds in the future. The whole network is differentiable and was then trained end-to-end using imitation learning. Thirdly, we validated the robustness and reliability of our proposed network by comparing it with 4 baselines under different weather and lighting conditions. Finally, we integrated our VTGNet with separated PID controllers into two types of driving platforms (i.e., car/truck), and conducted several online driving tests in an unseen simulated world. The generalization performance from the results validate the feasibility of the proposed framework. In short, we separated the control module from the end-to-end control framework and adjusted different controller parameters to the varied vehicle properties. The lesson learned from this is that the end-to-end modular network is a possible way for developing generalizable learning agents.

Since the camera may not be reliable under conditions that are too dim or too bright, in the future, we aim to develop a framework based on our current work to use multi-modal information from complementary sensors, such as lidar, radar and a thermal camera. We believe by doing so, the system could be more robust in challenging environments.

Fig. 13. Online driving tests in unseen simulated environments with different vehicles. The generated trajectories are marked as red dots. Additionally, noticeable obstacles are bounded by green boxes.

Fig. 14. (a) The input RGB image, and (b-d) the averaged feature maps of some bottleneck convolutional layers in the feature extractor of our VTGNet.
[13] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 2174–2182.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[15] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] P. Cai, Y. Sun, Y. Chen, and M. Liu, “Vision-based trajectory planning via imitation learning for autonomous vehicles,” in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Oct 2019, pp. 2736–2742.

[17] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in Advances in neural information processing systems, 1989, pp. 305–313.

[18] T. Wu, A. Luo, R. Huang, H. Cheng, and Y. Zhao, “End-to-end driving model for steering control of autonomous vehicles with future spatiotemporal features,” in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2019, pp. 950–955.

[19] S. Hecker, D. Dai, and L. Van Gool, “End-to-end learning of driving models with surround-view cameras and route planners,” in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 435–453.

[20] M. Bergqvist and O. Rödholm, “Deep Path Planning Using Images and Object Data,” Master’s thesis, Chalmers University of Technology, Gothenburg, Sweden, 2018.

[21] F. Codevilla, A. Krizhevsky, and A. Ogale, “Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst,” arXiv preprint arXiv:1812.03079, 2018.

[22] N. Rhinehart, K. M. Kitani, and P. Vernaza, “R2P2: A reparameterized pushforward policy for diverse, precise generative path forecasting,” in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 772–788.

[23] M. Green, “‘how long does it take to stop?’ methodological analysis of driver perception-brake times,” Transportation human factors, vol. 2, no. 3, pp. 195–216, 2000.

[24] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, “Exploring the limitations of behavior cloning for autonomous driving,” in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 9329–9338.

[25] X. Liang, T. Wang, L. Yang, and E. P. Xing, “CIRL: Controllable imitative reinforcement learning for vision-based self-driving,” in ECCV, 2018.

[26] F. Codevilla, A. M. Lopez, 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.

[27] L. v. d. Maaten and G. Hinton, “Visualizing data using t-SNE,” Journal of machine learning research, vol. 9, no. Nov, pp. 2579–2605, 2008.

[28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.

[29] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, “You’ll never walk alone: Modeling social behavior for multi-target tracking,” in Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009, pp. 261–268.

[30] 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.

[31] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, “Driving policy transfer via modularity and abstraction,” in CoRL, 2018.