End-To-End Multi-Modal Sensors Fusion System For Urban Automated Driving

Ibrahim Sobh
Valeo Egypt AI Research
ibrahim.sobh@valeo.com

Loay Amin
Valeo Egypt AI Research
loay.amin@valeo.com

Sherif Abdelkarim
Valeo Egypt AI Research
sherif.abdelkarim@valeo.com

Khaled Elmadawy
Valeo Egypt AI Research
khaled.elmadawy@valeo.com

Mahmoud Saeed
Valeo Egypt AI Research
mahmoud.saeed.ext@valeo.com

Omar Abdeltawab
Valeo Egypt AI Research
omar.abdeltawab.ext@valeo.com

Mostafa Gamal
Valeo Egypt AI Research
mostafa.gamal.ext@valeo.com

Ahmad El-Sallab
Valeo Egypt AI Research
ahmad.el-sallab@valeo.com

Abstract

In this paper, we present a novel framework for urban automated driving based on multi-modal sensors; LiDAR and Camera. Environment perception through sensors fusion is key to successful deployment of automated driving systems, especially in complex urban areas. Our hypothesis is that a well designed deep neural network is able to end-to-end learn a driving policy that fuses LiDAR and Camera sensory input, achieving the best out of both. In order to improve the generalization and robustness of the learned policy, semantic segmentation on camera is applied, in addition to applying our new LiDAR post processing method; Polar Grid Mapping (PGM). The system is evaluated on the recently released urban car simulator, CARLA. The evaluation is measured according to the generalization performance from one environment to another. The experimental results show that the best performance is achieved by fusing the PGM and semantic segmentation.

1 Introduction

Recently, a trend is setting towards end-to-end autonomous driving, where the visual perception task is implicit in the learning process, and the challenge is to map the raw inputs data to control actions. While humans are able to perceive the environment mainly by visual cues, computer vision based on camera perception alone is not as efficient; this is due to the mature and experienced high-level functions of the human perception. For example, the human brain is adequately trained to perceive depth well enough through stereo vision, while this task is difficult to mirror in computers. In addition, a self-driving car must be fully attentive, which requires a high level of perception of the surroundings and mapping of the environment–a task that is beyond the capabilities of the camera sensor alone. Hence, other sensors are usually used to achieve a complete surrounding view perception for covering wide and long range areas. Nowadays, one of the essential sensors in automated driving is LiDAR, which is known for its physical ability to perceive depth and 3D structures, regardless of lighting conditions. However, they lack the ability to capture visual features such as colors and rich textures. The common trend nowadays is to fuse different sensors to get the best features of the combination, and to achieve a better performance than using individual sensors.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

There are two popular approaches to enable self-driving; traditional pipeline methods and end-to-end learning methods. The traditional pipeline methods are based on explicit perception, tracking, mapping and localization, planning and control. In end-to-end approaches, the deep neural networks are trained to directly produce the control outputs from raw sensor inputs. The most popular approach is imitation learning, which is a supervised learning approach from human demonstrations. The main assumption here is that the function approximation ability of deep neural networks will be able to map the raw inputs to sensible actions imitating expert demonstrations. Most of the developed end-to-end systems use front camera as the sole system sensor. In its basic form, imitation learning is incapable of following a global plan, or intended user actions, especially with advanced situations like intersections, which are usually encountered in urban scenarios.

In this paper, we propose a deep neural network fusion architecture that incorporates semantic segmentation and LiDAR with different representations. The multi-stream architecture uses a late-fusion approach that fuses the output of several separate encoder networks. The fused features feed the different actions decoders, which are selected based on the global plan instructions.

We evaluate the effect of feeding the semantic segmentation view instead of the raw RGB frames. Our hypothesis is that in order to improve the network generalization in different environments, we need to provide the most abstracted views. The semantically segmented view along with the LiDAR projected bird view, or the front view with depth information mapped together, support this hypothesis. For semantic segmentation, we have ground truth segmentation provided by the simulator, which we use to train a segmentation network based on U-Net [10] architecture. We conduct our experimental work on the urban driving simulator CARLA [4] where the training scenario is performed on town 1 and the testing is performed on town 2–which is completely unseen in training. For LiDAR sensor, perception is conducted using two views; bird view projection where height information is lost, and front view using Polar Grid Mapping (PGM) method which does not lose depth in the process. We evaluate the performance based on carefully designed metrics that capture the corner case scenarios and evaluate the robustness of the system.

The rest of the paper is organized as follows; first we discuss the related work, followed by an overview of the system and details of the network architecture, training and segmentation network, and finally we conclude with the experimental section including the different experiments conducted on CARLA simulator, and the main conclusions.

2 Related Work

To the authors' knowledge, the first end-to-end control of autonomous cars was demonstrated in 1989 in Autonomous Land Vehicle In a Neural Network (ALVINN) [9]. In the aforementioned paper, a neural network of three fully-connected layers was used for the task of road following. The training of the network was based on simulated road images. The network has two inputs, from a camera and a laser range finder. The output is the direction the vehicle should follow. End-to-end off-road obstacle avoidance system is presented in [7] where a 6-layer convolutional neural network is trained on low resolution images from left and right cameras to predict the steering angles. More recently, NVIDIA reported through the DAVE-2 project that their CNN was successful in controlling their modified cars on public roads without human intervention [1]. How the trained network makes its decisions is explained in [2] by visualizing which parts in the input image most influence the output steering decisions.

End-to-end driving through command-conditional imitation learning is introduced in [3], where low level controls such as the angle at which the car should steer in each instance and the how much throttle is applied are learned from demonstrations. While high level commands such as whether to go straight, right or left at crossroads, are communicated through a planner. Accordingly, the control is separated from planning. An approach to transferring driving policies from simulation to reality through modularity and abstraction is presented in [8], where the driving system has binary semantic segmentation perception module, a command-conditional driving policy based on the perception module, implemented by a branched convolutional network, and finally a low-level PID controller. In this setup, the driving policy is not directly based on raw input.

For planning, our work utilizes the command-conditional imitation learning [3]. Additionally, fusion is conducted between the LiDAR sensor in different representations, and high level abstraction

semantic segmentation. Furthermore, driving performance in urban areas is analyzed and compared for different settings.

3 Multi-Modal Fusion Architecture for End-to-End Conditional Imitation Learning

The basic approach of the proposed fusion system is to extend the conditional imitation learning architecture in [3] to accommodate different modalities. A multi-stream architecture is shown in Figure 1. In order to apply fusion of two inputs, late-fusion architecture is used where each input is encoded in a separate stream, and then both encoded features are concatenated before the next fully connected layer. Late-fusion design gives a suitable capacity for each input so that the CNN learns the hierarchical representation for each input separately. Moreover, after concatenation, the fully connected layers learn how to weigh the encoded feature maps during the training process. The external command input (left, right, straight) selects the output head to be activated. In our work, we adopt the branched architecture as recommended in [3].

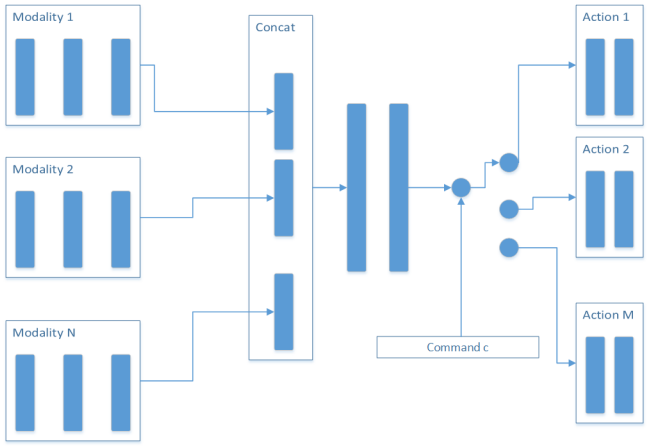

Figure 1: End-to-end multi-modal network architecture

Given a set of modalities $\{m_1, m_2, .., m_k, .., m_N\}$, an external command c_i, and a desired control action a_i, the training dataset becomes $D = \{(o_i, c_i, a_i)\}_{i=1}^{N}$, where o_i is the fused feature map's representation using late-fusion concatenation;

$$o_i = concat(\{F^1(m_1), F^2(m_2), .., F^3(m_k), .., F^N(m_N)\})$$

Each stream is mapped using its corresponding CNN F^i. The overall loss function becomes:

$$minimize_\theta \sum_i l(F((o_i, c_i; \theta), a_i)$$

The whole network is differentiable and hence can be trained end-to-end. The mapping multi-stream networks $F^1, F^2, ..F^N$ are also implicitly trained end-to-end. However, for some cases, pre-processing could be applied in order to enhance the generalization of the network.

3.1 Image stream representation

Feeding raw RGB images could be subject to variable lighting conditions from train to test environment. One solution for this issue is data augmentation, which will be discussed later in the experiments section. Another proposed solution is to provide an abstractly derived view from the RGB input; such as the semantic segmentation. We follow the U-Net architecture [10] in order to segment the input raw RGB image. Following this idea, the generalization performance improves significantly over using raw RGB frames.

3.2 LiDAR stream: Bird view representation

The concept of using an abstract view is also supported by using the LiDAR input, which provides an abstract view of the 3D structure of the environment and provides better generalization. Projection of LiDAR point cloud to a 2D projection is performed. This representation is shown to be significantly less computational expensive than raw point clouds, and it provides a visual representation of the surrounding objects in addition to their distance. The maximum depth shown in bird view is 50 meters in front of the car and 25 meters on each side of the car.

3.3 LiDAR stream: Polar Grid Mapping (PGM) representation

LiDAR point clouds provide the valuable depth information. However, they are computationally expensive to process. Our novel Polar Grid Mapping (PGM) method represents the LiDAR point cloud as a 2D image where each row represents a layer from the LiDAR sensor and each column represents an angle, and the value of the pixel itself is a floating point value ranging from 0.0 to 1.0 representing the distance of the point from the sensor. While front-view and top-view projections represent the LiDAR point cloud as a 2D image as well, the PGM representation offers an advantage over both representations in that, given enough horizontal angular resolution, it does not lose any points from the point cloud. For example if a LiDAR sensor has n layers, and the angles of beam rotations are discretized to be m values to cover the 360 degrees, we would end up with an image of size $n \times m$ that represents the whole point cloud.

4 Experiments

4.1 CARLA Simulator

CARLA [4] is an open source simulator recently released for autonomous driving in urban areas. The CARLA simulator provides a highly realistic environment including roads, buildings, vehicles and pedestrians, making it suitable for training and testing autonomous driving systems such as end-to-end artificial neural networks. In addition to a high fidelity environment, a configurable sensors suite is available to the users, including RGB cameras and basic LiDAR sensors. Moreover, abstraction is made available through the semantic segmentation ground truth provided by the simulator. CARLA provides two rich urban towns with different layouts. Other simulators such as Udacity simulator and TORCS [12] are not designed for urban area driving where there is a lack of intersections, lane rules and other complications that distinguish urban driving from track racing. The two towns are shown in Figure 2.

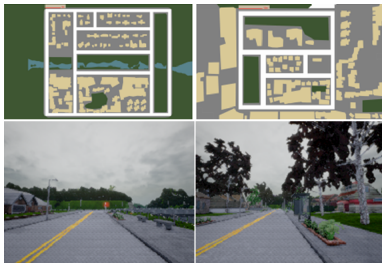

Figure 2: The two CARLA towns. **Left**: Map and sample view of Town 1. **Right**: Map and sample view of Town 2.

4.2 Data Collection

Town 1 is used only for model training, while Town 2 is used for evaluation as an unseen environment. CARLA comes with an autopilot for urban area driving. Training data for imitation learning is collected through the autopilot functionality by conducting a random walk-through of the urban area of Town 1. More precisely, at crossroads, the autopilot makes a random choice whether to go straight, right, or left. Two important modifications are applied to this basic data collection procedure. First, a

basic random walk typically produces unbalanced data where most of the collected samples are from straight driving scenarios. Accordingly, the collected data is balanced between the three different commands: straight, right and left, by sampling down the dominant straight samples. Moreover, in this urban setup, right turns are found to be more difficult to be be performed smoothly compared to the wider left turns. Consequently, relatively more data for right turns are collected with respect to left turns.

For the training process discussed in section 4.4, the selected command is recorded for each training sample. Then, in order to enable the learning to recover from unexpected drifting during testing time, a random noise is sampled from a normal distribution and added to the steering angle produced by the autopilot. The next actions performed by the autopilot are actually corrective actions to recover from drifting caused by the injected random noise. The collected raw data which is synchronized at 10 FPS, includes the following:

1. **Front camera sensor** produces RGB images where the camera is fixed at the car front, at a height of 1.5 meters.
2. **LiDAR sensor** where the Field of View (FOV) is 360, fixed at the center of the car, at a height of 2.5 meters with an upper FOV of 10 and a lower FOV of -30 degrees vertically. It produces a 3D point cloud with a rotating frequency of 10 rotations per second as well as 200,000 points per point cloud. The maximum range of the LiDAR sensor is up to 50 meters of diameter.
3. **Semantic segmentation view** is provided by the simulator and provides 13 class labels placed in the same physical location as the front camera sensor.

Additionally, the controls collected consist of the steering angle values between -1 to 1, which maps to steering turn angle between -70 to 70 degrees; and throttle of values between 0 and 1. The size of the collected data is 136K samples.

4.2.1 Data Pre-processing

Camera: For a more focused view, the front RGB camera and semantic segmentation view are cropped to a size of 88×200. **Semantic segmentation**: The 13 classes of the semantic segmentation are reduced into 3 classes: road, lane and others. These classes are sufficient for basic driving, making turns and following lanes. Finally, front RGB camera and segmentation inputs are normalized to lie in a range of continuous values from 0 to 1. **LiDAR bird view**: the LiDAR point cloud is projected to a binary image of the bird view for the area in front of the car. **LiDAR PGM view**: For the PGM representation the number of rows depends on the number of layers in the LiDAR sensor. In our case the sensor has 64 layers. We choose the horizontal resolution of the sensor angles to be the 1 degree. We crop the point cloud to include only the point in front of the vehicle, keeping only 180 degrees out of 360 degrees. This leaves us with an image of size 64×180 that represent the front half of the point cloud.

4.2.2 Data Augmentation

For better generalization, data augmentation is applied for front RGB camera frames. The augmentation is applied online during the training process where 1% of the batches are altered by applying random brightness effect, contrast, Gaussian blur and salt-and-pepper noise. One or more of these changes are applied at random permutations.

4.3 Network Architecture

A separate neural network is trained for each experiment. Table 1 shows the basic single input network architecture which consists of 4 convolutional layers followed by 3 heads, each head is responsible for one command to enhance the specialization of the network. Each head consists of two fully connected layers and an output layer for producing steering angle and throttle. The convolutional layers are followed by batch normalization [5], and fully connected layers are followed by dropout [11] with a ratio of 50%. The used configurations are as follow: A 2×2 stride convolution, a 5×5 kernel size for first layer and 3×3 for the rest of the layers. For fusion network architecture, there are two inputs instead of one. Each input is encoded separately through 4 convolutional layers similar to the

single input network architecture. The output feature maps are concatenated, and followed by the fully connected layers.

Table 1: Network Architecture

LAYER	CHANNELS (KERNEL) STRIDE
CONV1	32 (5×5) 2
CONV2	32 (3×3) 2
CONV3	64 (3×3) 2
CONV4	64 (3×3) 2
FC1	64 (PER HEAD)
FC2	32 (PER HEAD)
OUTPUT	2 PER HEAD: STEERING & THROTTLE

As shown in Table 2, the input is different according to the sensor or the abstraction view. The front camera input is an RGB image of shape $88 \times 200 \times 3$. The input shape for the semantic segmentation is $88 \times 200 \times 1$. Bird view LiDAR input shape is $200 \times 176 \times 1$. While the PGM LiDAR input shape is $64 \times 180 \times 1$.

Table 2: Input per network

NETWORK	INPUT
RGB	$88 \times 200 \times 3$
SEG	$88 \times 200 \times 1$
LIDAR BIRD VIEW	$200 \times 176 \times 1$
LIDAR PGM	$64 \times 180 \times 1$

4.4 Training

The networks are trained to minimize the mean-squared error of the steering angle and throttle values given the autopilot ground truth values. We follow the multitasking training approach where we have three tasks and three corresponding network heads for straight, right and left commands. According to the selected command, one of the heads is activated, and the loss is minimized for the current active head. The network is trained for 200 epochs and batch size of 16. For optimization, ADAM optimizer [6] is used where $\beta_1 = 0.9$ and $\beta_2 = 0.99$ and learning rate $\alpha = 0.0001$

4.5 Semantic Segmentation

For the semantic segmentation part of the pipeline, the U-Net [10] architecture is used. The model is trained for 40 epochs with a batch size of 4. ADAM optimizer was used with $\beta_1 = 0.9$ and $\beta_2 = 0.99$ and a learning rate $\alpha = 0.0001$. The input is an RGB image with the shape of $88 \times 200 \times 3$. Initially the model was trained without any data augmentation, but the resultant model was highly overfitted to the training data. To tackle this problem, 50% of the input images are augmented in the same way as illustrated in section 4.2.2. The mean IOU (Intersection Over Union) performance of the resultant model is shown in Table 3.

Table 3: Mean IOU for each class on different towns

DATASET	OTHER	LANES	ROADS
TOWN 1 TRAINING	99.52%	99.43%	99.19%
TOWN 2 TESTING	99.40%	97.25%	99.17%

The model is trained for three classes: Road, Lanes and Other, on images collected from one town, and tested on images collected from a different town. Testing on a different town is meant to test whether the model has overfitted to the training town or not. Table 3 shows the performance on both the training and the testing towns. It is apparent from the results that augmentation used in training helped the model generalize to a different town.

The explanation for this high performance on the test town is that the augmentation forced the model to ignore the irrelevant details of the training town, such as colors and rendering textures, and instead focus on the higher level feature that affects the decision of the model such as location of the road, and lane lines. These higher level features are shared between both training and testing towns, which made the model easily generalize from training town to testing town.

Another explanation is the large difference in size between the training and the testing data. During training, 120k samples were used, while only 70 samples were used in testing. The reason for such a high difference is that we only used the test set to gauge whether the model has overfitted to the training town, and not as an accurate indication of the model's performance in new environments.

4.6 Evaluation

In order to measure and compare the performance of the different networks, three evaluation metrics are defined. Off-road time: measures the time in seconds spent by the car where its bounding box intersects with curbs. Off-lane time: measures the time in seconds spent by the car where its bounding box intersects with road lanes. Number of crashes: counts the number of conducted crashes. Any crash resets the episode to start elsewhere in the same town. For evaluation purposes, each network drives the car in the simulated environment for 20 minutes in a random walk scenario similar to the one performed in the data collection procedure, section 4.2. The mentioned metrics are calculated and reported during the testing.

Table 4: Results for 20 minutes of random walk driving, the Off-road and Off-lane times; and Number of crashes.

NETWORK	OFF-ROAD	OFF-LANE	CRASHES
TRAINING: TOWN 1			
RGB	47	10	2
SEG_{GT}	8.5	1.8	0
SEG_{UNET}	9	0	0
$LiDAR_{birdview}$	48	11	1
$SEG_{GT} + LiDAR_{birdview}$	8.3	0	0
$SEG_{UNET} + LiDAR_{birdview}$	17	0	0
$LiDAR_{PGM}$	46.5	8.6	2
$SEG_{GT} + LiDAR_{PGM}$	0.9	0	0
$SEG_{UNET} + LiDAR_{PGM}$	1.2	3.4	0
TESTING: TOWN 2			
RGB	370	288	19
SEG_{GT}	18.4	7.8	0
SEG_{UNET}	20	35	0
$LiDAR_{birdview}$	99	68	2
$SEG_{GT} + LiDAR_{birdview}$	11.5	0	0
$SEG_{UNET} + LiDAR_{birdview}$	19	5	0
$LiDAR_{PGM}$	98.3	63.4	3
$SEG_{GT} + LiDAR_{PGM}$	5.6	0	0
$SEG_{UNET} + LiDAR_{PGM}$	**7.6**	**4.7**	**0**

Comparisons are shown in Table 4, where the performance of the driving policies on the training Town 1 and the testing Town 2 is evaluated for different architectures. Single input architectures include RGB camera sensor (RGB), ground truth semantic segmentation provided by the simulator (SEG_{GT}), semantic segmentation provided by the trained UNET (SEG_{UNET}), and LiDAR input ($LiDAR_{birdview}$) and ($LiDAR_{PGM}$). Fusion architectures, that have two inputs, include fusion of ($SEG_{GT} + LiDAR_{birdview}$), fusion of ($SEG_{UNET} + LiDAR_{birdview}$), fusion of ($SEG_{GT} + LiDAR_{PGM}$) and fusion of ($SEG_{UNET} + LiDAR_{PGM}$).

When using front camera only, the model trained on Town 1, as expected, failed to generalize the driving policy on Town 2, due to layout and texture differences between the two towns. This phenomenon, which might be encountered in real situations, makes generalization hard to achieve when depending only on camera inputs. LiDAR sensors, on the other hand, are robust to texture and lighting variations and, hence, generalize much better compared to RGB camera sensors. Semantic segmentation, when available, provides relatively better individual performance due to its simple

abstraction of the input scene into the fundamental three classes needed for making suitable driving decisions. Across all experiments, driving policy based on the abstracted ground truth semantic segmentation, provided directly by the simulator, is better than the one based on the trained UNET. However, the trained UNET gives more practical results for real-life situations. The PGM only view is marginally better than LiDAR bird view and when fusing the LiDAR bird view with the semantic segmentation view, a better performance is observed. Finally, to further improve the driving performance, LiDAR PGM and semantic segmentation are observed to be the best modalities to fuse. In other words, the fusion of semantic segmentation and LiDAR PGM benefits from the advantages of both.

For better understanding on how the trained neural networks make their decisions based on the input, attention maps are used to indicate the important parts of the inputs that influence the network output. For visualization, we follow the algorithm described in [2]. According to the samples shown in Figure 3, it is clear that the trained neural networks pay more attention on lanes, roads and mid-range LiDAR bird view or PGM, without being explicitly taught to do so.

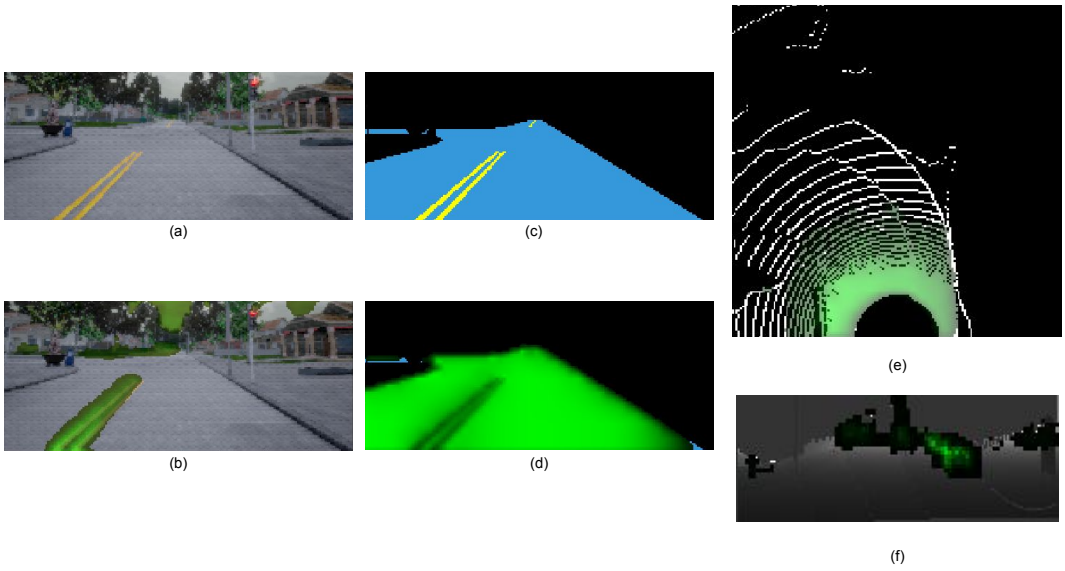

Figure 3: **a**: Original RGB camera image; **b**: Attention map overlayed on the Original RGB image; **c**: Ground truth semantic segmentation; **d**: Attention map overlayed on the semantically segmented image; **e**: Attention map overlayed on LiDAR bird view image; **f**: Attention map overlayed on LiDAR with PGM processing

5 Conclusion

In this work we propose an end-to-end architecture for multi-modal fusion. Late-fusion design is incorporated in order to handle different modalities in separate multi-streams, fusing the resulting feature maps. Our hypothesis is that, using abstraction of the input views, would help the model generalize over unseen environments. Hence, we semantically segment the input RGB pixels into three fundamental classes, and fuse this with the LiDAR abstracted views, bird view and PGM. The evaluation results on CARLA urban simulator suggest that the incorporated abstraction performs significantly better than using the raw RGB alone. Additionally, fusing the best modalities, LiDAR bird view + semantic segmentation, yields performance that surpasses individual streams' performance, where off-road performance is improved by the LiDAR PGM, and off-lane performance is improved by the segmentation.

References

[1] 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 arXiv:1604.07316*, 2016.

[2] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[3] F. Codevilla, M. Miiller, 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)*, pages 1–9. IEEE, 2018.

[4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on Robot Learning*, pages 1–16, 2017.

[5] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 448–456. JMLR. org, 2015.

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

[7] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp. Off-road obstacle avoidance through end-to-end learning. In *2005 Annual Conference on Neural Information Processing Systems, NIPS 2005*, 2005.

[8] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[9] D. A. Pomerleau. Alvinn: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pages 305–313. MIT Press, 1988.

[10] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[12] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4, 2000.