RIANet: Road Graph and Image Attention Network for Urban Autonomous Driving

Timothy Ha, Jeongwoo Oh, Hojun Chung, Gunmin Lee, and Songhwai Oh

Abstract— **In this paper, we present a novel autonomous driving framework, called a road graph and image attention network (RIANet), which computes the attention scores of objects in the image using the road graph feature. The process of the proposed method is as follows: First, the feature encoder module encodes the road graph, image, and additional features of the scene. The attention network module then incorporates the encoded features and computes the scene context feature via the attention mechanism. Finally, the low-level controller module drives the ego-vehicle based on the scene context feature. In the experiments, we use an urban scene driving simulator named CARLA to train and test the proposed method. The results show that the proposed method outperforms existing autonomous driving methods.**

I. INTRODUCTION

In recent years, vision-based recognition algorithms have shown a dramatic improvement in the field of autonomous driving. For example, pedestrian detection [1], lane detection [2], and semantic segmentation [3] have been developed and shown remarkable performance. However, applying such vision-based methods to vehicle control is a challenging task. In most cases, it is required to elaborately design a handcraft rule-based controller to use the vision-based features.

There have been studies which applied vision-based features in a learning-based controller. Previous methods have used image-based features [4]–[6], semantic segmentation based features [7], or detection based features [8] to train an imitation learning based autonomous driving controller. However, these works have shown insufficient performance compared to commercialized handcraft controllers.

One of the most critical problems when applying the vision-based approaches to vehicle control is that the controller does not know which part of the image is more important. Each object in the image has different importance according to the scene context. For example, when a driver sees traffic lights at an intersection, not all traffic lights are equally important. The driver should focus more on the traffic lights at the front than the traffic lights at the crossroad. Likewise, drivers usually pay more attention to jaywalking pedestrians than pedestrians walking on the sidewalk.

To compensate for such a problem, several works investigated the importance of image features along with the LiDAR sensor data. Zhao et al. [9] proposed a LiDAR and image fusion based 3D semantic segmentation method which can be used to identify the scene context of objects. In addition, Prakash et al. [10] used the attention mechanism [11] to capture the importance of the image feature according to the LiDAR sensor data. However, LiDAR-based attention is not enough to fully understand the scene context for two reasons. (1) It is difficult to directly extract meaningful information such as the road direction and connection because the LiDAR data is a high-dimensional representation. (2) The LiDAR data lack prior knowledge about the traffic (e.g., right-hand or left-hand traffic), which is important for determining whether an object is moving in a safe direction. As studied in previous works [12]–[14], the vehicle movement is highly influenced by the direction of the road. Therefore, a new type of state representation, which can reflect the road structural context of the scene, is required to overcome the limitation of LiDAR-based attention.

In this paper, we propose an autonomous driving framework named *road graph and image attention network* (RIANet), which considers the importance of object features according to the road structure. To reflect the graph information of the road, we represent the road structure in the form of a graph which is called *road graph*. Unlike other attention-based controllers, the proposed method leverages the road graphical features along with visual features extracted from image and LiDAR sensors. The attention mechanism allows the network to compute the scene context and prioritize the importance of objects according to the road structure.

We use imitation learning (IL) to train the network. The evaluation and data collection are conducted on a 3D urban scene driving simulator named CARLA [4]. The agent in the CARLA simulator is required to deal with dangerous situations such as lane changing, unexpected obstacle avoidance, crossing intersection, and unprotected turn. The experiment shows that the proposed method outperforms the baseline methods in terms of all the suggested metrics. Our contributions are summarized as follows:

- We propose a novel autonomous driving framework which considers the importance of objects according to the road structure.
- We demonstrate that the road graph features can effectively reflect the road structural context of the scene.
- We experimentally show that the driving performance can be improved by fusing the road graph and the other sensor data features.

II. RELATED WORK

State Representation for Autonomous Driving. There are a number of works that use IL for training an autonomous driving agent. Previous methods used various types of state representation such as the vehicle position [15], the LiDAR feature [16], and the egocentric camera view [4]–[6]. However, IL shows a lower performance when using a single raw input features. Instead, numerous studies have proposed to combine pre-processed features as state representation. Sobh et al. [7] incorporated semantic segmentation feature with image and LiDAR features. Behl et al. [8] predicted both semantic segmentation and object detection features from an RGB image and used them to train an agent. However, their work does not focus on the relationship between the road structure and the pre-processed features.

Graph Representation for Autonomous Driving. Recent works have been proposed to use a graph representation of a road for numerous purposes. LaneGCN [13] and Vectornet [12] used a graph representation of a road and incorporate it with vehicle trajectory features. Their work is, however, only restricted to a trajectory prediction problem and does not consider camera image features. In Road-GNN [14], vehicle information, such as its position and velocity are encoded into a graph feature and are used for training a controller for an autonomous vehicle. However, Road-GNN does not use

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grants funded by the Korea Government (MSIT) (No. 2019-0-01190, [SW Star Lab] Robot Learning: Efficient, Safe, and Socially Acceptable Machine Learning, 50%, and No. 2019-0-01371, Development of Brain-Inspired AI with Human-Like Intelligence, 50%). *(Corresponding author: Songhwai Oh.)*

The authors are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul 08826, Korea (e-mail: {timothy.ha, jeongwoo.oh, hojun.chung, gunmin.lee}@rllab.snu.ac.kr, songhwai@snu.ac.kr).

image features as inputs and assumes that the agent is given the ground truth obstacle positions. VGM [17] constructed a graph structure by capturing image features which an agent has visited during navigation. However, their work is only restricted to a visual navigation problem, and the network does not consider the dynamic objects in the environment.

III. Problem Setting

We first clarify our problem settings before explaining the proposed framework. We consider an urban scene driving environment. The goal of the agent is to navigate a given route while following traffic rules. The route is composed of multiple goal locations in the road environment. At each time step t, the agent is given a high-dimensional observation o_t which consists of the following components.

Road Graph. The agent is given the topology information of a drivable road in the graph form. A road graph $G_{global} = (V, E)$ consists of nodes V and edges E. A node $n_i \in V$ represents a point on the road and is distributed along the centerline of the road segment. An edge $e_{i\to j} \in E$ connects the node n_i and the node n_j. The direction of $e_{i\to j}$ represents the direction of the road segment. We use the method from [14] to extract a road graph G_{global}. We sample nodes on the road at intervals of 3m. We connect edges between nodes depending on the direction and connection of the roads. In addition, we connect two nodes if it is possible for a driver to change the lane along the edge. A node n_i contains a node feature f_{n_i} and an edge $e_{i\to j}$ contains an edge feature $f_{e_{i\to j}}$, respectively. In simulation and training, the agent only observes a subgraph G_t, which consists of the nearest nodes to the ego-vehicle. The detailed explanations about a road graph and subgraphs are described in Section VI-B.

Camera Image. The agent is given a front ego-view camera image which has a resolution of 400×300 with a $100°$ FOV. To remove radial distortion [10], we crop the image to 256×256 before entering the image into the feature encoder.

Additional Sensor Data. The agent is given LiDAR, GPS, IMU, and speedometer sensor data. The LiDAR point cloud is pre-converted into a 2D BEV grid image by the method in [10], [18]. A 2D BEV grid image contains 256×256 pixels and covers a 32m $\times$ 32m area in front of the ego-vehicle. A 2D BEV grid image has two channels: The first channel represents the points above the ground plane while the second channel represents the points below the ground plane. There are GPS, IMU, and speedometer sensor data as well. These sensor give the position and speed data of the ego-vehicle to the agent. The position and direction of the ego-vehicle are localized from the GPS and IMU sensor inputs. We use the extended Kalman filter for localization.

IV. Road Graph and Image Attention Network

In this section, we explain the road graph and image attention network (RIANet), which leverages the attention score [11] between a road graph and image features. The proposed framework is divided into three modules: (1) a feature encoder, (2) an attention network, and (3) a low-level controller. The feature encoder module encodes a road graph, a camera image, and additional sensor data into features. The attention network module takes these feature embeddings as inputs, applies the attention mechanism to fuse features, and extracts the context feature of the scene. The low-level controller module calculates the target speed from the context feature and controls the ego-vehicle using a PID controller.

A. Feature Encoder

The feature encoder module takes an observation $o_t = \{G_t, I_t, S_t\}$ as an input and encodes the road graph G_t, the camera image I_t, and the additional sensor data S_t into feature embeddings. The encoder for each component is a neural network and we use a different network architecture

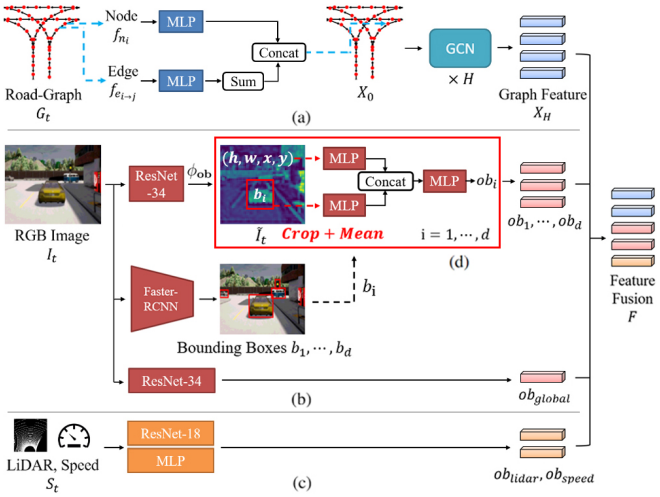

Fig. 1. The structure of the feature encoder module. (a) Road graph encoder. (b) Camera image encoder. (c) Additional feature encoder. (d) The encoding process of an object feature.

for each encoder. The entire network structure of the feature encoder module is shown in Figure 1.

Road Graph Encoder. The road graph encoder extracts a feature embedding of the road graph G_t using graph convolutional network (GCN) [19]. Similar to the graph encoding method in [14], we first regularize each node feature f_{n_i} and edge feature $f_{e_{i\to j}}$ using fully-connected networks ψ_{node} and ψ_{edge}. We then sum up the edge feature along the starting node index and concatenate it with the node feature. If the number of nodes in G_t is N, the encoding process is formulated as follows:

$$\tilde{f}_{n_i} = \psi_{node}(f_{n_i}), \text{ for } 1 \le i \le N \tag{1}$$

$$\tilde{f}_{e_{i\to j}} = \psi_{edge}(f_{e_{i\to j}}), \text{ for } 1 \le i, j \le N \tag{2}$$

$$\tilde{f}_i = concat(\tilde{f}_{n_i}, \sum_k \tilde{f}_{e_{k\to i}}), \text{ for } 1 \le i \le N \tag{3}$$

We get the embedded feature matrix $X_0 \in \mathbb{R}^{N\times Z}$ by applying the encoding process to each element in G_t. The i-th row of the X_0 is the feature $\tilde{f}_i \in \mathbb{R}^Z$, where Z is the feature size. We update the feature matrix X_0 using a H-layers GCN network [19]. The update process of the k-th layer of GCN is formulated as follows:

$$X_k = \sigma(\tilde{A} X_{k-1} W_k + B_k), \tag{4}$$

where $\tilde{A} \in \mathbb{R}^{N\times N}$ is a normalized adjacency matrix of G_t with self-loops, σ is a leaky-ReLU [20], and W_k and B_k are the weight and bias of the k-th GCN layer. We iteratively update X_0 and obtains the final graph feature embedding X_H.

Camera Image Encoder. We expect the camera image encoder to capture some important visual information such as traffic lights, pedestrians, and obstacles. To capture and encode the object features, we use a ResNet [21] based feature map and a Faster R-CNN [22]. The encoding process of an object feature is shown in Figure 1(d). We first extract the feature map of the image I_t using the ResNet based network. The ResNet based feature map $\phi_{ob}(I_t) = \tilde{I}_t$ has the channel size of C and the same height and width as the input image I_t. We then detect objects in the image I_t using Faster-RCNN and obtain a bounding box set $\{b_i\}_{i=1}^d$, where b_i is a bounding box and d is the number of the detected bounding boxes. d is not a fixed value and can be changed according to the image I_t. Bounding box b_i has the corresponding classification feature c_i and positional feature

p_i. The classification feature c_i is a classification result of the i-th object and encoded in a one-hot vector. The positional feature $p_i = (h, w, x, y)$ represents the positional information of a bonding box b_i, where h, w, x, and y represent the height, the width, the x and y position of the center of the bounding box b_i in the 2D pixel space. For each bounding box b_i, we crop the feature map $\tilde{I}_t$ according to the size and location of the bounding box b_i. We get the spatial mean of the cropped feature map and incorporate it with c_i and p_i using MLP networks. For a cropped feature map with the size of $C \times h \times w$, the spatial mean has the size of C. The feature embedding for the i-th object is denoted as ob_i. The object feature encoding process is formulated as follows:

$$\tilde{I}_t = \phi_{ob}(I_t) \tag{5}$$

$$f_{img} = \psi_{img}(mean(crop(\tilde{I}_t; b_i))) \tag{6}$$

$$f_{lin} = \psi_{lin}(concat(c_i, p_i)) \tag{7}$$

$$ob_i = \psi_{ob}(concat(f_{img}, f_{lin})), \tag{8}$$

where ψ_{img}, ψ_{lin}, and ψ_{img} are MLP networks, $crop$ is a crop operator, $mean$ is a spatial mean operator, and $concat$ is a concatenation operator.

In addition to each object feature embedding ob_i, the camera image encoder also encodes the global image feature embedding ob_{global} to capture the global feature of the image I_t. The image I_t is encoded into ob_{global} by the ResNet based network ϕ_{global} as follows:

$$ob_{global} = \phi_{global}(I_t), \tag{9}$$

where ob_i and ob_{global} are one-dimensional vectors and have the same feature size of Z.

Additional Feature Encoder. The additional feature encoder take the additional sensor data S_t as inputs. The S_t contains LiDAR data f_{lidar} and speed data f_{speed}. We encode each f_{lidar} and f_{speed} into feature embedding. First, LiDAR data is encoded by a ResNet model ϕ_{lidar}. As explained in Section III, the LiDAR data f_{lidar} is a pre-processed 2D BEV grid image. The LiDAR data f_{lidar} is encoded into a feature ob_{lidar} by ϕ_{lidar}. The speed data f_{speed} is also encoded into a feature ob_{speed} by the MLP network ψ_{speed}. The encoding process of the additional feature encoder is formulated as follows:

$$ob_{lidar} = \phi_{lidar}(f_{lidar}) \tag{10}$$

$$ob_{speed} = \psi_{speed}(f_{speed}) \tag{11}$$

B. Attention Network

The attention network module outputs the scene context by incorporating all the encoded features from the feature encoder. To leverage the attention between each feature, we use the transformer model [11] and compute the attention scores between the features. The network architecture of the attention network module is shown in Figure 2(a). We first integrate all the feature embeddings in Section IV-A into the feature fusion F as follows:

$$F_{ob} = [ob_1, \cdots, ob_d, ob_{global}, ob_{lidar}, ob_{speed}] \tag{12}$$

$$F = concat(X_H, F_{ob}) \tag{13}$$

The feature F_{ob} contains the object feature embeddings $ob_1, \cdots, ob_d \in \mathbb{R}^Z$, the global image feature embedding $ob_{global} \in \mathbb{R}^Z$, the LiDAR feature embedding $f_{lidar} \in \mathbb{R}^Z$, and the speed embedding $f_{speed} \in \mathbb{R}^Z$. Each feature has the same size of Z and these features are stacked into the feature F_{ob}. The feature $F_{ob} \in \mathbb{R}^{(d+3)\times Z}$ and the road graph feature $X_H \in \mathbb{R}^{N\times Z}$ are concatenated into the feature fusion $F \in \mathbb{R}^{(N+d+3)\times Z}$.

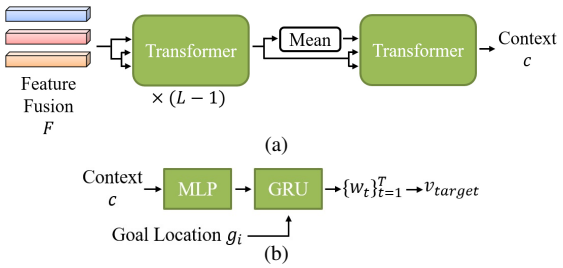

Fig. 2. The structure of the networks. (a) The attention network module. (b) Waypoint prediction network.

We use the attention mechanism [11] to obtain the attention scores between each feature in F. The attention function $Attn(\cdot, \cdot)$ is formulated as follows:

$$Attn(M_1, M_2) = concat(head_1, \cdots, head_n)W^O, \tag{14}$$

$$\text{where } head_i(M_1, M_2) = softmax(\frac{Q_i {K_i}^T}{\sqrt{Z}})V_i \tag{15}$$

$$Q_i, K_i, V_i = M_1 W_i^Q, M_2 W_i^K, M_2 W_i^V, \tag{16}$$

where M_1 and M_2 are arbitrary inputs to the attention function, $softmax$ is the softmax function, n is the number of heads, $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{Z\times Z}$, and $W_O \in \mathbb{R}^{(nZ)\times Z}$ are weight matrices. The transformer consists of an attention function, a MLP network ψ_{tran}, and a layer normalization LN. The transformer is formulated as follows:

$$Transformer(M_1, M_2) = LN(\psi_{tran}(sub) + sub), \tag{17}$$

$$\text{where } sub = LN(Attn(M_1, M_2) + M_1) \tag{18}$$

The transformers are stacked L times and compose a total of L transformer layers. Except for the last layer, the process of each layer is formulated as follows:

$$F_k = Transformer_k(F_{k-1}, F_{k-1}), \text{ for } 1 \leq k < L \tag{19}$$

Starting from $F_0 = F$, the fusion transformer module iteratively computes F_k. The last transformer layer computes the scene context feature c from $F_{L-1} \in \mathbb{R}^{(N+d+3)\times Z}$ and $\bar{F}_{L-1} \in \mathbb{R}^{1\times Z}$ as follows:

$$c = Transformer_L(\bar{F}_{L-1}, F_{L-1}), \tag{20}$$

where $\bar{F}_{L-1}$ is the feature mean of F_{L-1}. The size of the scene context feature $c \in \mathbb{R}^{1\times Z}$ is invariant to the number of nodes N and the number of object features d.

C. Low-Level Controller

The low-level controller takes the scene context c as an input and computes the steering control value and the target speed for the ego-vehicle.

Steering. We compute the steering value which makes the ego-vehicle follow the pre-defined path ζ. The path ζ is calculated through the following process. Similar to [10], we assume that the goal is to navigate a given route $\gamma = \{g_i\}_{i=1}^l$, where g_i is a goal location and l is the number of the goal locations. We first find a node $n_{g_i}^{near} \in G_{global}$ which is the nearest node to g_i. We then use the Dijkstra algorithm and calculate the shortest path in G_{global}, which visits all the nodes $\{n_{g_i}^{near}\}_{i=1}^l$. The path ζ is constructed after applying a line smoothing method to the Dijkstra shortest path. The steer PID controller computes the deviation between the current ego-vehicle position and the path ζ and finds the steering value which makes the ego-vehicle follow the path ζ.

Target Speed. We also compute the target speed of the ego-vehicle from the future waypoints which is predicted from a waypoint prediction network [6], [10]. The network architecture of the waypoint prediction network is shown in

Figure 2(b). As demonstrated in [6], [10], we also empirically found that it is better to use the waypoint prediction network than to compute the target speed directly. The waypoint prediction network takes the scene context c as an input and predicts the waypoints in the ego-vehicle coordinate frame. The scene context c is first passed to the MLP network. The output of the MLP network is entered to the GRU [23] as a hidden state. Starting from the initial input of $w_0 = (0, 0)$, the GRU iteratively outputs the differential waypoints δw_t. During the iteration, the GRU incorporates the current goal location g_i in its inputs. The waypoints w_t are constructed as $w_t = \sum_{\tau=1}^{t} \delta w_\tau$ for the future time steps $t = 1, \cdots, T$ and we use $T = 4$. After the waypoints are predicted, we compute the target speed $v_{target} = \|w_1 - w_0\|_2 / \delta t$, where δt is the time interval between the waypoints. We use a PID controller which has the same configuration as [6], [10] and provides the throttle and brake values to the ego-vehicle, which make the ego-vehicle have the target speed v_{target}.

V. Training

We train our network using IL [6], [10]. The output of the waypoint prediction network is the waypoints $\{w_t\}_{t=1}^{T}$. We use L_1 loss between the predicted waypoints w_t and the expert waypoints w_{gt}. Given the expert waypoints $\{w_t^{gt}\}_{t=1}^{T}$, the loss function is formulated as follows:

$$loss = \sum_{t=1}^{T} \|w_t - w_t^{gt}\|_1 \tag{21}$$

VI. Experiments

In experiment, we use a widely-used driving simulator named CARLA [4]. We train the proposed network with an expert dataset and compare the evaluation results with the baseline methods. All data collection and experiments are conducted in the CARLA environment.

A. Experimental Settings

We use the CARLA 0.9.10 for the experiments. There are eight types of maps in CARLA, and in each environment, the agent needs to complete the given route while handling challenging situations such as lane changing, unexpected obstacle avoidance, and crossing intersection. We follow the evaluation settings in [10] and use Town05 for evaluation. We use two different types of scenarios: Town05 Short and Town05 Long. The scenarios in Town05 Short setting have 10 routes of 100-500m in length and the scenarios in Town05 Long setting have 10 routes of 1000-2000m in length.

B. Dataset

The training data is collected by a handcraft expert policy which uses privileged information about the environment. The dataset is provided by [10] and we especially use the ClearNoon dataset. In our setting, we do not use the data from Town10 for training because CARLA 0.9.10 does not provide the HD-map of Town10, which is required for constructing a road graph. We instead use Town01, Town02, Town03, Town04, Town06, and Town07.

The road graph data is not contained in the dataset of [10]. Therefore, we generate the road graph feature G_t according to the vehicle position and rotation in the dataset. We first construct the global graph G_{global} using HD-map data of each town. HD-map data in CARLA uses the OpenDRIVE format [24] and provides border information for each lane of roads. We obtain the center trajectory of each lane through borderline information and sample nodes from the center trajectory at intervals of 3m, including the endpoint of each center trajectory. HD-map also provides linkage information of lanes. We connect edges so that nodes in the same lane can be sequentially linked. In addition, edges are connected between end nodes of linked lanes. Inspired by [14], we define node and edge features as follows:

$$f_{n_i} = (x_{n_i}, y_{n_i}, \mathbb{1}_{ts}, \mathbb{1}_{tr}, \mathbb{1}_{oc}, \mathbb{1}_{oc} \cdot v) \tag{22}$$

$$f_{e_{i\to j}} = (x_{e_{i\to j}}, y_{e_{i\to j}}), \tag{23}$$

where (x_{n_i}, y_{n_i}) is the 2D position of the node n_i, $(x_{e_{i\to j}}, y_{e_{i\to j}})$ is the 2D direction of the edge $e_{i\to j}$, v is the speed of the ego-vehicle, and $\mathbb{1}_{ts}$, $\mathbb{1}_{tr}$, $\mathbb{1}_{oc}$ are the indicators. $\mathbb{1}_{ts}$ is the traffic signal indicator which is 1 if the node is on an intersection and 0 otherwise. $\mathbb{1}_{tr}$ is the path indicator which is 1 if the node is included in the pre-defined path ζ. $\mathbb{1}_{oc}$ is the occupancy indicator which is 1 if the node is the nearest node to the current ego-vehicle position. The node position (x_{n_i}, y_{n_i}) and edge direction $(x_{e_{i\to j}}, y_{e_{i\to j}})$ are computed in the ego-vehicle coordinate frame, where x-axis is parallel to the ego-vehicle direction. The position of the ego-vehicle can be inversely calculated from the position of each node and the occupancy indicator.

After constructing G_{global}, we extract the graph presentation G_t. We first select the K nearest nodes to the ego-vehicle position. We then select only the nodes in front of the ego-vehicle with a small margin μ. We use $K = 96$ and $\mu = 10$m in our setting. The position and direction of the ego-vehicle are estimated by the extended Kalman filter and used to compute the ego-vehicle coordinate frame. The road graph is computed in real-time when evaluating the agent.

C. Implementation details

In the road graph encoder, the MLP networks ψ_{node} and ψ_{edge} are constructed by a fully-connected layer with the size of 6×32 and 2×32, respectively. We use $H = 3$ for the number of GCN [19] layers. The weight of each GCN layer has the size of 64×64 except for the last layer and the weight of the last GCN layer has the size of 64×128. We use a pre-trained ResNet-34 model [21] for the network ϕ_{global} to encode the global image feature ob_{global}. The network ϕ_{global} is pre-trained on ImageNet [26] data and the last layer of ϕ_{global} is changed to a 512×128 size fully-connected layer. We use a ResNet-18 model for the network ϕ_{lidar} to encode the LiDAR feature ob_{lidar}. In contrast to ϕ_{global}, ϕ_{lidar} is not pre-trained. The last layer of ϕ_{lidar} is also changed to a 512×128 size fully-connected layer. We use 1×64 and 64×128 size fully-connected layers for the network ψ_{speed} to encode the speed data feature embedding ob_{speed}. All the feature embeddings ob_{global}, ob_{lidar}, and ob_{speed} have the same feature size of $Z = 128$. We use a Faster-RCNN model [22] which is reimplemented and pre-trained on the CARLA dataset [25] . We use two separate networks for object detection: a traffic light detector and an obstacle detector. The traffic light detector detects the traffic light signal which can be labeled as Red, Yellow, Green, or Off. The obstacle detector detects the obstacles which can be labeled as Pedestrian, Car, Bicycle, or Motorcycle. The classification result of each detector is encoded as a one-hot vector. The bounding boxes of the detected objects of two detectors are combined into the bounding box set $\{b_i\}_{i=1}^{d}$. The feature map $\tilde{I}_t$ is computed from the ResNet-34 model [21] and we use the feature map from the first convolution groups named conv1. The ResNet-34 model ϕ_{ob} shares the parameters with the global image feature encoding network ϕ_{global}. Before applying the crop operation, the feature map from ϕ_{ob} is upsampled to the size of the original image I_t with the bi-linear interpolation. The MLP networks ψ_{img} and ψ_{lin} are constructed by a fully-connected layer with the size of 64×128 and 8×128, respectively. The MLP network ψ_{ob} is constructed by two fully-connected layers which have the size of 256×128 and 128×128. We use two separate networks for ψ_{img}, ψ_{lin}, and ψ_{ob} depending on whether the detected object belongs to a traffic light or

TABLE I
PERFORMANCE COMPARISON

Method	Town05 Short			Town05 Long		
	DS ↑	RC ↑	IpKm ↓	DS ↑	RC ↑	IpKm ↓
CIRLS [5]	21.693 ± 1.751	25.762 ± 0.807	98.141 ± 14.164	7.889 ± 1.047	10.751 ± 0.358	17.020 ± 1.182
RBC [25]	49.947 ± 4.966	84.016 ± 3.246	33.527 ± 4.783	14.692 ± 4.019	82.866 ± 7.282	6.669 ± 0.490
AIM [10]	69.917 ± 15.104	73.106 ± 19.000	10.315 ± 2.489	39.558 ± 5.056	83.018 ± 11.477	3.424 ± 0.248
AIM+R [10]	79.969 ± 18.601	82.286 ± 19.714	8.592 ± 7.382	39.873 ± 5.005	80.251 ± 17.594	2.753 ± 0.325
Transfuser [10]	64.495 ± 7.840	70.915 ± 7.200	12.059 ± 2.835	36.438 ± 8.412	96.650 ± 5.420	3.473 ± 0.452
Transfuser+R [10]	67.031 ± 8.800	73.254 ± 9.837	10.881 ± 2.481	37.283 ± 7.049	92.815 ± 4.607	3.209 ± 0.613
RIANet (Ours)	**87.469** ± 2.363	**93.881** ± 3.827	**6.319** ± 1.419	**44.719** ± 2.513	**96.934** ± 2.927	**2.624** ± 0.163

an obstacle. The speed data encoding network ψ_{speed} is also an MLP network and constructed by two fully-connected layers which have the size of 1×64 and 64×128. All the MLP network uses the leaky-ReLU [20] for the non-linear function. We stack $L = 5$ transformer layers in the attention network module. Each transformer in a layer has four heads. All the linear projection weights of each transformer have the size of 128×128. The MLP network ψ_{tran} is constructed by two fully-connected layers which have the size of 128×512 and 512×128. We use the same network architecture as [6], [10] for the waypoint prediction network.

D. Baselines

We compare the proposed method with several baselines.

- **CILRS** [5]. CILRS is a conditional imitation learning method which uses a single front camera image and a speedometer sensor data. The network architecture of CILRS contains a conditional module which takes a navigational command as the condition.
- **RBC** [25]. RBC is a rule-based control which identifies the location of the object through a Faster-RCNN [22] and controls the ego-vehicle based on the hard-coded rules. We used a similar method with [25] but reimplemented some rules to apply it to our setting. RBC detects traffic lights and obstacles as the same as the proposed method. However, unlike the proposed method, RBC estimates the 3D position of a detected object by finding the 3D coordinate of the LiDAR point cloud which is projected closest to the center points of the bounding box. RBC finds the lane in which the obstacle is currently located. If the obstacle is on the current lane and the distance between the ego-vehicle and the obstacle is less than λ, RBC stops the ego-vehicle. We have fine-tuned the parameter λ and use the best value for evaluation. The distance is computed according to the Frenet coordinate frame of the lane. RBC also stops the vehicle if the traffic light closest to the center of the image I_t is detected as red or yellow.
- **AIM** [10]. AIM uses only the camera image as an input. The image input is encoded through ResNet [21] based networks. The encoded features are entered into the waypoint prediction network. We test two type of low-level controller to check the effect of changing low-level controller. Vanilla AIM follows the waypoints predicted by the waypoint prediction network. AIM+R follows the pre-defined path as the same as the proposed method. In AIM+R, the waypoint prediction network only decides the target speed of the ego-vehicle.
- **Transfuser** [10]. Transfuser uses a camera image and a LiDAR feature as inputs. The image and LiDAR input are encoded through ResNet [21] based networks and fused together through the attention mechanism. The fused feature is then entered into the waypoint prediction network. Similar to AIM, we test Transfuser which is the vanilla version and Transfuser-R which uses a path-following low-level controller.

E. Comparison Results

We use three metrics in the evaluation: route completion, driving score, and infractions per km. Route completion (RC) is the percentage of the completed route length relative to the total route length. Driving score (DS) is the product between the route completion and the penalty weight. The penalty weight starts from 1.0 and decreases when the agent commits an infraction such as collisions or running a red light. Infractions per km (IpKm) is the total number of infractions divided by the total distance in kilometers traveled. Higher is better in DS and RC, and lower is better in IpKm. The detailed explanation of each metric can be found from the CARLA official website [27].

We compare the proposed method with the other baselines. Each method is trained and evaluated with five different random seeds. Table I shows the mean and standard deviation of the result. We observe that the proposed method shows the highest performances in all three metrics. Especially, the proposed method shows higher performance than Transfuser which does not use the road graph for the attention mechanism. The result demonstrates that the use of the road graph feature can improve the performance of the autonomous driving controller. To show the effect of changing the low-level controller, we tested AIM-R and Transfuser-R which use the same low-level controller as RIANet. However, the proposed method shows better performance than AIM-R and Transfuser-R even they use the same low-level controller. In addition, the proposed method shows better performance than RBC which uses the same object detector model. The result shows that the proposed method leverages the detected object features more effectively compared to RBC.

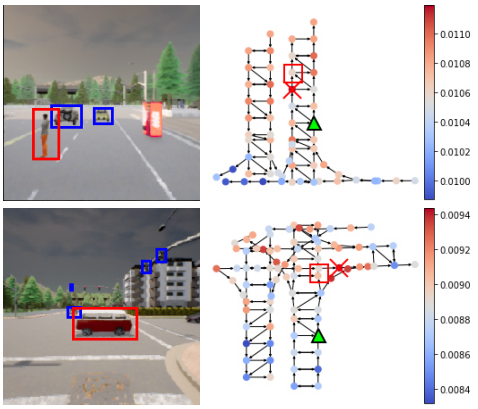

Fig. 3. Visualization of attention scores. (Left) Detected objects on camera images. The object with the highest attention score is marked as red and the other objects are marked as blue. (Right) Road graph of scenes in 2D BEV. Green arrows indicate the position and direction of the ego-vehicle. Red boxes represent the estimated 2D position of the object with the highest attention scores and red x marks represent the 2D position of the node with the highest attention scores.

F. Attention Score Visualization

We visualized the attention score of the detected objects and the nodes of the road graph. The attention score is measured from the first layer of the attention network by taking the mean of the attention weights along the query features. Figure 3 shows two examples on Town05. On the camera images, we marked the object with the highest attention score among the objects in red and the other objects in blue. The results show that the attention network tends to pay more attention to important objects (e.g., a jaywalking pedestrian and a vehicle that crosses the intersection) than other objects. We also visualized the relationship between the

TABLE II
ABLATION STUDY

Configure	Town05 Short			Town05 Long		
	DS ↑	RC ↑	IpKm ↓	DS ↑	RC ↑	IpKm ↓
no-road-graph	84.198 ± 1.971	**95.477** ± 3.575	7.374 ± 2.358	39.660 ± 3.092	**100.000** ± 0.000	3.035 ± 0.203
no-detection	81.729 ± 6.503	90.729 ± 6.503	8.453 ± 1.972	39.091 ± 2.055	98.592 ± 1.991	2.997 ± 0.245
no-global-image	60.681 ± 2.513	93.532 ± 6.832	23.750 ± 1.343	25.778 ± 5.552	86.312 ± 4.694	4.563 ± 0.370
no-LiDAR	82.754 ± 2.694	93.166 ± 3.669	9.657 ± 1.121	39.139 ± 4.177	97.069 ± 4.146	2.976 ± 0.265
Default	**87.469** ± 2.363	93.881 ± 3.827	**6.319** ± 1.419	**44.719** ± 2.513	96.934 ± 2.927	**2.624** ± 0.163

image and node features. We estimated the 3D position of the object with the highest attention score using the estimation method of RBC [25] and plotted the position on the 2D coordinates. We also plotted the road graph on the same figure. Interestingly, the object with the highest attention score (red boxes) and the node with the highest attention score (red x marks) are distributed close to each other on the 2D coordinates. From the result, we can infer that the network considers the relationship between the road graph and the object when calculating the attention scores.

G. Ablation Study

As an ablation study, we tested how each input feature affects the performance of the agent. In the default configuration, we used all the road graph, detected bounding boxes, global image, LiDAR, and speed features. However, in each case of no-graph, no-detection, no-global-image, and no-LiDAR setting, we ignored the corresponding feature embedding X_H, $\{ob_i\}_{i=1}^{d}$, ob_{global}, and ob_{lidar}, respectively. The feature fusion F is computed without the ignored feature. The network is trained once with each configuration and evaluated with three different random seeds. As shown in Table II, The default configuration shows the highest DS and IpKm in both Town05 Short and Town05 Long settings. The default configuration shows a lower performance (1.671% and 3.066%) in RC but shows a higher performance in DS (3.885% and 12.757%) and IpKm (14.312% and 13.543%) compared to no-road-graph. From this comparison, it can be inferred that the agent drives more cautiously when using the road graph. In addition, the average performance of no-road-graph, no-detection, and no-LiDAR were higher than that of Transfuser in Table I, because Transfuser does not use both road-graph and detection features. As a result, the ablation study shows that all the proposed feature inputs are essential for driving successfully.

VII. CONCLUSION

In this work, we have presented a new driving framework that leverages the attention score between the road graph and image feature. We haved proposed RIANet to capture relationship between sensed observations against the road structure via a road graph. The proposed network computes the scene context by incorporating a road graph, image, and additional features through the attention mechanism. In the experiments, we have shown that the proposed method outperforms the baseline methods in terms of all the metrics. The results show that the use of the attention score improves the performance in urban autonomous driving. Since our framework demands road graph information, it is important to obtain an accurate road graph of the scene. In the future work, we plan to study how to detect and correct road graph errors, such as road deformation caused by construction.

REFERENCES

[1] I. Hasan, S. Liao, J. Li, S. U. Akram, and L. Shao, "Generalizable pedestrian detection: The elephant in the room," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 11 328–11 337.

[2] L. Liu, X. Chen, S. Zhu, and P. Tan, "Condlanenet: a top-to-down lane detection framework based on conditional convolution," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 3773–3782.

[3] Y. Yuan, X. Chen, and J. Wang, "Object-contextual representations for semantic segmentation," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 173–190.

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

[5] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 9329–9338.

[6] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning (CoRL)*, 2019.

[7] I. Sobh, L. Amin, S. Abdelkarim, K. Elmadawy, M. Saeed, O. Abdeltawab, M. Gamal, and A. El Sallab, "End-to-end multi-modal sensors fusion system for urban automated driving," in *Advances in Neural Information Processing Systems (NeurIPS) Workshops*, 2018.

[8] A. Behl, K. Chitta, A. Prakash, E. Ohn-Bar, and A. Geiger, "Label efficient visual abstractions for autonomous driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2338–2345.

[9] L. Zhao, H. Zhou, X. Zhu, X. Song, H. Li, and W. Tao, "Lif-seg: Lidar and camera image fusion for 3d lidar semantic segmentation," *arXiv preprint arXiv:2108.07511*, 2021.

[10] 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 (CVPR)*, 2021, pp. 7077–7087.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[12] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 522–11 530.

[13] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 541–556.

[14] T. Ha, G. Lee, D. Kim, and S. Oh, "Road graphical neural networks for autonomous roundabout driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 162–167.

[15] S. Choi, K. Lee, S. Lim, and S. Oh, "Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6915–6922.

[16] G. Lee, D. Kim, W. Oh, K. Lee, and S. Oh, "Mixgail: Autonomous driving using demonstrations with mixed qualities," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5425–5430.

[17] O. Kwon, N. Kim, Y. Choi, H. Yoo, J. Park, and S. Oh, "Visual graph memory with unsupervised representation for visual navigation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 15 890–15 899.

[18] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "Precog: Prediction conditioned on goals in visual multi-agent settings," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 2821–2830.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations (ICLR)*, 2017.

[20] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of International Conference on Machine Learning (ICML) Workshops*, 2013.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, pp. 91–99, 2015.

[23] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[24] Association for Standardization of Automation and Measuring Systems, "Opendrive." [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[25] ERDOS, "Pylot." [Online]. Available: https://github.com/erdos-project/pylot

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[27] CARLA, "Carla autonomous driving leaderboard." [Online]. Available: https://leaderboard.carla.org/