Road Graphical Neural Networks for Autonomous Roundabout Driving

Timothy Ha, Gunmin Lee, Dohyeong Kim, and Songhwai Oh

***Abstract*— We propose a novel autonomous driving framework that leverages graph-based features of roads, such as road positions and connections. The proposed method is divided into two parts: a low-level controller which follows the trajectory calculated by a graph-based path planner, and a high-level controller which determines the speed of the vehicle to follow the traffic flow. The high-level controller uses a road graphical neural network (Road-GNN), which encodes a road graph into latent features to perceive the surrounding environment. We use a 3D driving simulator to test the performance of Road-GNN, which is implemented based on the satellite image data of 30 roundabout intersections. To show that the proposed method can be generalized to various road environments, the proposed method is tested using roundabouts which are different from the training set. In the experiment, the proposed method successfully trains the agent and drives an ego-vehicle through various roundabout environments. The results show that the graph-based method is effective for autonomous driving.**

I. INTRODUCTION

Along with increasing interests in autonomous driving, the investigation of reinforcement learning (RL) using the driving simulator has been actively carried out by a number of researchers. However, most of the previous works cannot be generalized to other driving environments. One of the most widely used examples is the highway environment [1], [2], but their works are only limited to a single one-way road. There are more complex environments, such as merging roads [3], [4], intersections [5], or city streets [6]. The previous studies, however, use only a simple and fixed road environment for training and testing.

One of the main reasons for the generalization difficulty is that it is difficult to capture the generalized feature of the road environment. For example, [1] and [2] use the relative speed and position of vehicles as inputs features, but they cannot capture information about how vehicles move on a complex road. Another example is the semantic top-view image, which includes road lines [7]. However, as shown in the recent studies [8], [9], it is not efficient to use images for representing the topology of roads.

In this paper, we propose a generalized autonomous driving framework to use graphical representation of roads. The proposed graphical representation includes the information about the structure of roads, and the positional relationship between vehicles and the road. Unlike other works using the graph representation [8], [9], there are four distinctive features of the proposed framework.

- It is the first work which uses graph-based features to control vehicles to the best of our knowledge.
- We do not focus on a simple environment such as highways or merging roads but on various roundabout intersections, where the topology of the road is more complicated and multiple vehicles can enter from multiple directions simultaneously.
- The relationship between vehicles and the road is also considered when calculating graph features
- We use long short-term memory (LSTM) [10] along with graph neural networks (GNN) [11] to capture the historical features of vehicles on the road graph.

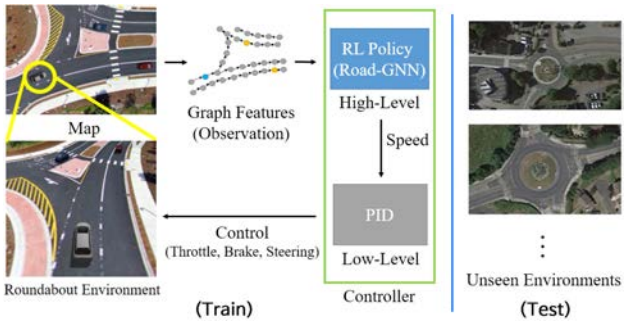

Fig. 1. Overview of training a Road-GNN based controller and autonomous driving using Road-GNN in unseen environments.

The proposed method uses a two-level controller: First, a low-level PID controller follows a trajectory determined by a graph-based path planner. The path planner perceives the road graph and finds a path based on the Dijkstra algorithm. Second, a high-level controller determines the target speed of the vehicle to follow the traffic flow. The high-level controller uses the encoded feature of the road graph, which is extracted by a road graphical neural network (Road-GNN). A Road-GNN compresses the graph features of the road into a latent feature used for the high-level controller. We use the RL framework to train the network, and it successfully controls the vehicle in various and complex unseen road environments. Figure 1 shows an overview of how Road-GNN works in training and test phases.

We trained the network from a set of road environments and tested it on a different set to show the generalizability of Road-GNN. We have collected satellite images of roundabouts for training and testing and applied them to our driving simulator to simulate a realistic road environments and structures. Note that the satellite images are only used to construct road models and get state representations, and we use a 3D simulator and a 3D dynamic model for the experiments. The image data are collected from Google search and NAVER Maps [12], an online mapping service. We have created road environments by pre-processing and reconstructing the collected images. The simulator is implemented within ROS Gazebo [13], and the 3D vehicle model we use is the ROS Prius model [14], which has throttle, brake, and steering control similar to a real vehicle. In the experiment, we show that our agent successfully drives in an unseen environment by learning road graph features. The proposed method outperforms the other methods, which use different features and networks.

T. Ha, G. Lee, D. Kim, and S. Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul, 08826, Korea (e-mail: {timothy.ha, gunmin.lee, dohyeong.kim}@rllab.snu.ac.kr, songhwai@snu.ac.kr).

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01190, [SW Star Lab] Robot Learning: Efficient, Safe, and Socially-Acceptable Machine Learning).

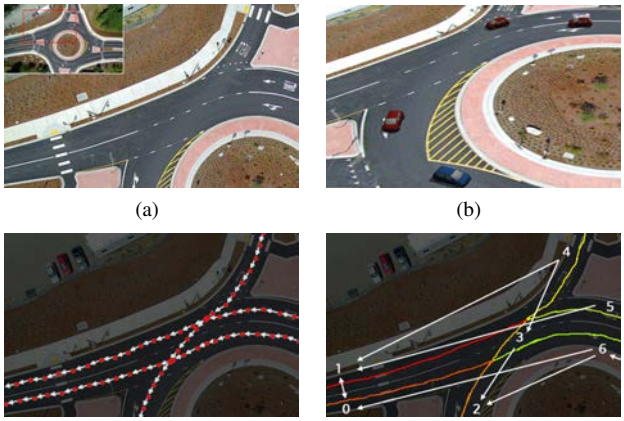

Fig. 2. An example of a roundabout environment. (a) A part of the original roundabout image data. (b) The 3D simulator used in the experiments. The vehicle model we use is the ROS Prius model [14] whose dynamics is similar to a real vehicle. The roads in the simulator are reconstructed based on the satellite image data. (c) The point-level graph G_p, which is organized from the data. The red dots are the nodes in G_p and the white arrows are the edges connecting the nodes. (d) The segment-level graph G_{seg}. Each segment is colored with different colors and tagged with the numbers. The white arrows are the connection between the segments in G_{seg}.

II. Related Work

There are several works that simulate road environments. One of the most popular examples is the highway simulator using the NGSIM dataset [15]. CARLA [6] is another example of road environments, which simulates a city street. In most of the works using these simulators, however, the road structures do not change from training to testing [1], [2]. If the driving agent is tested on a new unseen environment, it is likely to fail to drive because it lacks the knowledge of the new environment [6], [16].

Reinforcement learning is one of the most widely used algorithms to train an autonomous driving agent. Ippei Nishitani et al. [3] used a deep RL network to train a vehicle controller that can merge into a highway while reducing the impact on the overall traffic flow. Dhruv Mauria Saxena et al. [17] trained an RL agent which can drive in a simulator that simulates a dense and crowded road. However, those works are limited to a simple 2D simulator and their road environments are fixed to simple highway models that do not actually exist in the real world. Błażej Osiński et al. [18] implemented a driving simulator based on real-world data and trained an autonomous driving agent through RL. However, it is assumed that no other moving vehicles exist in the simulator, which makes it less realistic.

Graph representation of a road is studied in VectorNet [8]. VectorNet vectorizes components on a road and expresses the relationship between each vector as a graph. LaneGCN [9] proposed a lane graph representation that preserves the structures of roads. Both VectorNet and LaneGCN use GNN to extract graph-based features, but they only focus on the prediction of vehicle movements.

III. Roundabout Environment

We consider a driving environment where multiple vehicles are moving on a road and interacting with each other. In the environment, vehicles other than the ego-vehicle, which is controlled by the agent, are controlled by a pre-defined controller. Also, each simulated vehicle follows a pre-defined path. The goal of the agent is to successfully follow the given path and make the ego-vehicle reaching the goal position safely. To follow the traffic flow, the agent must regulate the speed properly and know when and where to stop. This is only possible if the agent is sufficiently trained to recognize the surrounding circumstance.

We formulate the problem by a Markov decision process (MDP). For every time step t, the policy of the agent π observes the current state s and executes an action a from the discrete action set. Then, the agent obtains a reward r and the next state s' from the environment as a result of the action a. We then optimize the policy π to maximize the expected sum of the rewards with a discount factor γ.

$$\mathbb{E}_\pi[\sum_t \gamma^t r(t)] \tag{1}$$

In the remainder of this section, we describe state, action, and reward setup. After that, we also explain how other vehicles move in the environment.

A. State

In our environment, the state s is represented by using a graph representation. An example of the graph representation is shown in Figure 2. The topology of the road can be expressed by both the point-level graph G_p and the segment-level graph G_{seg}. First, the point-level graph G_p contains each point p_i as a node, and each p_i represents a 2D point on the map M. The edge from the point p_i to the point p_j is represented by $e_{i\to j}$. The edges of G_p are connected along the direction in which vehicles are allowed to move. Second, the segment-level graph G_{seg} contains each segment seg_k as a node, and each segment seg_k represents a set of the points. Depending on the position, each point p_i belongs to one of the segment seg_k in G_{seg}. Similar to G_p, the graph G_{seg} is connected in the direction of the traffic flow.

Each edge $e_{i\to j}$ and node p_i of the graph G_p has its own features that reflect the vehicle and road conditions. For example, the edge feature of $e_{i\to j}$ is calculated from the relative position between p_i and p_j.

$$Feature(e_{i\to j}) := Pos(p_j) - Pos(p_i) \tag{2}$$

Here, $Pos(p_i)$ represents the 2D position vector of p_i on the map M. If the edge $e_{i\to j}$ does not exist in the graph G_p, the edge feature is considered to be a zero vector. The node feature of p_i is calculated depending on the vehicle conditions such as position and velocity. Here, we assume that there are a total of N vehicles including the ego-vehicle, and each vehicle is numbered as $v_0, \cdots, v_{N-1}$. For a vehicle v_k, the node p_i has the features of v_k if the node p_i is the nearest point to the vehicle v_k on the graph G_p. The feature includes the position and velocity of v_k, and an occupancy indicator, which is 1 if p_i is the nearest point to the vehicle v_k and 0 otherwise.

$$Feature(p_i) := [F(v_0), F(v_1), \cdots, F(v_{N-1})], \tag{3}$$

where

$$F(v_k) = \begin{cases} f(v_k), & \text{if } p_i = \underset{p_j \in G_p}{\operatorname{argmin}}\, d(v_k, p_j) \\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{4}$$

$$f(v_k) = \left[Pos_{R_k}(v_k)^T, Vel_{R_k}(v_k)^T, 1 \right]^T \tag{5}$$

Here, $d(v_k, p_j)$ is the L2 distance between the vehicle v_k and the point p_j. $Pos_{R_k}(v_k)$ and $Vel_{R_k}(v_k)$ are the position

and velocity vector of the vehicle v_k, respectively, which are relatively calculated from the nearest edge to the vehicle v_k. When $e_{nearest_k}$ is the nearest edge to the vehicle v_k, $p_{nearest_k}$ is the starting node of $e_{nearest_k}$, and θ_k is the angle of the direction of $e_{nearest_k}$, then

$$Pos_{R_k}(v_k) := R_k^{-1}\left[Pos(v_k) - Pos(p_{nearest_k})\right] \quad (6)$$

$$Vel_{R_k}(v_k) := R_k^{-1} Vel(v_k), \quad (7)$$

where

$$R_k = \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} \quad (8)$$

The matrix R_k is the rotation matrix of θ_k.

The agent observes the state s, where the observation is a subset of graph G_p, which includes the K nearest nodes to the ego-vehicle. The subset graph is given to the agent in the form of the adjacency matrix A, the node feature matrix A_N, and the edge feature matrix A_E. The node and edge feature matrix contain each node and edge features as elements. Since there are K nodes in the subset graph and N vehicles on the environment, A, A_E, A_N matrix have the size of $(K \times K)$, $(K \times N \times 5)$, and $(K \times K \times 2)$, respectively.

B. Action

For every time step, the agent controls the vehicle using a two-level controller. The low-level controller adjusts both the throttle and steering through PID control while the high-level controller determines the target speed of the vehicle. The action is defined as the target speed given to the vehicle. Similarly to other recent works which use an MDP formulation in driving [3], [19], we discretize the actions as the set of target speeds $\{0\,\mathrm{m/s}, 3\,\mathrm{m/s}, 6\,\mathrm{m/s}, 9\,\mathrm{m/s}, 12\,\mathrm{m/s}\}$.

The low-level controller is designed to follow a pre-determined path. For each episode, the start and goal segments are chosen randomly and the path is set to connect segments. The path is calculated from the entire graph representation and the Dijkstra algorithm. The detailed path finding process is as follows: First, the shortest path in the segment-level graph G_{seg} is calculated using the Dijkstra algorithm. Here, the weights for each edge is considered equal to ones. From the segment-level path, the point-level path is constructed from the points belonging to each segment. For example, when the segment-level path is $\{seg_0, seg_1, \cdots, seg_n\}$, then the point-level path is defined as follows.

$$path = seg_0 \cup \cdots \cup seg_n \quad (9)$$

Note that the segment seg_k is an ordered set of the points p. When $seg_i = \left\{p_0^i, \cdots, p_{l_i-1}^i\right\}$, $seg_j = \left\{p_0^j, \cdots, p_{l_j-1}^j\right\}$, and the two segments are connected in the direction of $seg_i \rightarrow seg_j$, then the two sets are combined as follows:

$$path = \left\{\cdots, p_0^i, \cdots, p_{l_i-1}^i, p_0^j, \cdots, p_{l_j-1}^j, \cdots\right\} \quad (10)$$

Here, l_i is the number of points in the segment seg_i. There can be cases where two segments seg_i and seg_j have edges in both directions $seg_i \rightarrow seg_j$ and $seg_j \rightarrow seg_i$. The physical meaning of it is that each segment represents one of the two lanes of a multi-lane road, and a vehicle can change the lanes between them. In these cases, the two segments are combined at a random point when calculating the path. For example, if a point p_{rand}^i is randomly selected from seg_i and the nearest point to p_{rand}^i in seg_j is p_{near}^j, then the two segments are combined as follows:

$$path = \left\{\cdots, p_0^i, \cdots, p_{rand}^i, p_{near}^j, \cdots, p_{l_j-1}^j, \cdots\right\} \quad (11)$$

The low-level PID controller follows the point-level path by controlling both the throttle and steering of the vehicle. First, the steering is controlled to minimize the deviation of the vehicle from the path. The deviation is calculated by the nearest edge e, which connects the two points in the point-level path. Here, we define x_{pos} as the length of the line perpendicular from the vehicle to the nearest edge e. x_{direct} is the difference of the angle between the direction of the vehicle v_k and the nearest edge e. The steering PID controller minimizes the sum of two values: $x_{pos} + x_{direct}$. Second, the throttle is controlled to set the speed of the ego-vehicle to a given target speed. Using PID control, the throttle inputs minimize the difference between the ego-vehicle speed and the target speed. Since the throttle cannot take a negative input, the brake input is taken instead when the speed is higher than the target speed.

For safety, there is also a collision-avoidance system in the controller. First, the future position of each vehicle is estimated using a unicycle vehicle model. The vehicle steps on the brake when a collision is expected. After the vehicle has stopped completely, the system re-estimates the future positions assuming that the vehicle can move at a low speed. If a collision is not expected, then the vehicle moves again based on the agent and PID controller.

C. Reward

The goal of the agent is to make the vehicle moves from the start area to the goal area. We set a positive reward of $+1$ when the agent successfully reaches the goal area. The start and goal areas are pre-determined and randomly changed before an episode starts. To prevent the ego-vehicle from being stationary in one place, we also give a small negative reward of -0.01 for each time step before the agent reaches the goal area or exceeds the time limit.

D. Other Vehicle Movements

There are several other vehicles that move in the simulator. We use the same path planner and PID controller used in the ego-vehicle to simulate the other vehicle movements. Each vehicle is set up to follow different paths, which vary from episode to episode. Each controller for the simulated vehicle and the ego-vehicle differs in two ways: First, the simulated vehicles only move at the fixed target speed $9\,\mathrm{m/s}$ while the ego-vehicle changes the target speed according to input actions. Note that although the target speed is constant, the speeds of the vehicles are not maintained due to the collision-avoidance system. Second, the simulated vehicle is re-spawned in a new random starting position when it reaches the goal area. This re-spawning repeatedly occurs until the ego-vehicle finally reaches its goal area.

IV. ROAD-GNN

We now explain our road graphical neural network (Road-GNN), which is used to train the agent. Road-GNN processes a graph observation of the state and calculates the encoding used for the policy and value networks of the agent. Through this networks, we train the RL agent and control the ego-vehicle. Road-GNN takes three steps to encode the road

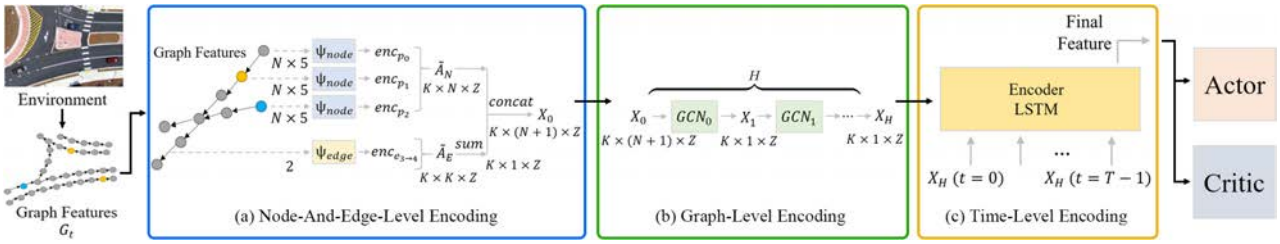

Fig. 3. The overall process of Road-GNN. (a) Node-and-Edge-Level encoding. (b) Graph-Level encoding. (c) Time-Level encoding.

graph: (1) node-and-edge-level encoding, (2) graph-level encoding, and (3) time-level encoding. The overall process of Road-GNN is illustrated in Figure 3.

A. Node-and-Edge-Level Encoding

Since each node and edge feature have different units, we first regularize them using the node and edge encoder networks, ψ_{node} and ψ_{edge}. If the graph observation at time t is G_t, and the number of the nodes in G_t is K, then the networks encoding for each node and edge are as follows:

$$enc_{p_i} = \psi_{node}(Feature(p_i)), \text{ for } 0 \le i < K \quad (12)$$

$$enc_{e_{i\to j}} = \psi_{edge}(Feature(e_{i\to j})), \text{ for } 0 \le i, j < K \quad (13)$$

In the case of node features, encoding is processed individually for each vehicle features.

$$enc_{p_i} = \psi_{node}(Feature(p_i)) \quad (14)$$

$$= [\psi_v(F(v_0)), \psi_v(F(v_1)), \cdots, \psi_v(F(v_{N-1}))] \quad (15)$$

For the vehicle feature encoding network ψ_v, we use a simple fully-connected network. Here, the network ψ_v is not shared among all the vehicles v_k. To distinguish the features of the ego-vehicle from the other vehicles, we use ψ_{ego} for the ego vehicle v_{ego}, and ψ_{other} for all the other vehicles. The network ψ_{other} is shared among each vehicle v_k except for v_{ego}. By encoding the elements in the feature matrix A_N and A_E, we can get the encoded feature matrix $\tilde{A}_N$ and $\tilde{A}_E$, which have the size of $(K \times N \times Z)$ and $(K \times K \times Z)$ respectively, where Z is the size of the encoded vector.

B. Graph-Level Encoding

After node-and-edge-level encoding, we use a GCN [11] based model for graph-level encoding. First, $\tilde{A}_N$ and $\tilde{A}_E$ are incorporated into a single matrix. The edge encoding matrix $\tilde{A}_E$ is summed up along starting node index, and then concatenated with the node encoding matrix $\tilde{A}_N$.

$$Sum(\tilde{A}_N) = \left[\sum_i enc_{e_{i\to 0}}, \cdots, \sum_i enc_{e_{i\to K}}\right] \quad (16)$$

$$X_0 = Concat(\tilde{A}_E, Sum(\tilde{A}_N)) \quad (17)$$

Starting from X_0, a GCN network ψ_{graph} iteratively calculates the next layer output X_{k+1} from X_k. The layers are stacked H times, and the final output of GCN is X_H.

$$X_{k+1} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X_k W_k + B_k), \quad (18)$$

where

$$\tilde{A} = A + I \quad (19)$$

$$\tilde{D}_{i,j} = \begin{cases} \sum_j \tilde{A}_{i,j}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \quad (20)$$

Here, A is the adjacency matrix, σ is a non-linear function, W is a weight, and B is a bias. Only for the first layer GCN_0, the results of each vehicle channel in X_0 is added into one as like a 3D convolution layer.

$$X_1 = \sigma(\sum_{j=0}^{N} \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X_0^j W_0^j + B_0^j) \quad (21)$$

C. Time-Level Encoding

Each graph observation G_t at time t is encoded by the graph encoding network. In Road-GNN, the network uses the historical memory of G_t with a time length of T. To deal with a historical data, we use a long short-term memory (LSTM) [10] model. First, a readout layer ψ_{read} converts the graph-level encoding of each $G_{t-T+1}, \cdots, G_t$ into a series of vectors. Each vector is entered into a LSTM encoding network in the order of time. The final output of the LSTM network becomes the final feature which is used for the policy and value network of the RL agent.

D. Learning Algorithm

We train the agent using proximal policy optimization (PPO) [20], which is a widely used reinforcement algorithm for many recent studies [7], [17]. In our PPO implementation, the agent has the actor and critic networks, and each network shares the common input features, as shown in Figure 3.

V. Experiments

In the experiment, we examined the generalizability of the proposed method. We have tested the proposed method in various realistic scenarios and environment we implemented.

A. Environment and Network Details

To implement the roundabout environment, we collected 30 satellite images of roundabouts. As illustrated in Figure 4, the collected images have various road structures. Each image is paired with a graph representation constructed from the map structure. We generated the graph representation of a map in the following steps: First, we manually drew the possible trajectories and posed the start and goal area on the map. The nodes of the graph are regularly sampled from the trajectories and grouped into the segments. The grouping process is also performed manually, and we divided the segments at the intersection points. The edges between the nodes and the segments are connected depending on the map structure and the road direction. When the graph is entered into Road-GNN, some additional edges are added to make GNN features flow through the edges. The edge is added if one of the following conditions is satisfied.

- The two nodes are close ($<2\,\mathrm{m}$).
- It is possible to change the lane between the two nodes.
- The two nodes are occupied with vehicles

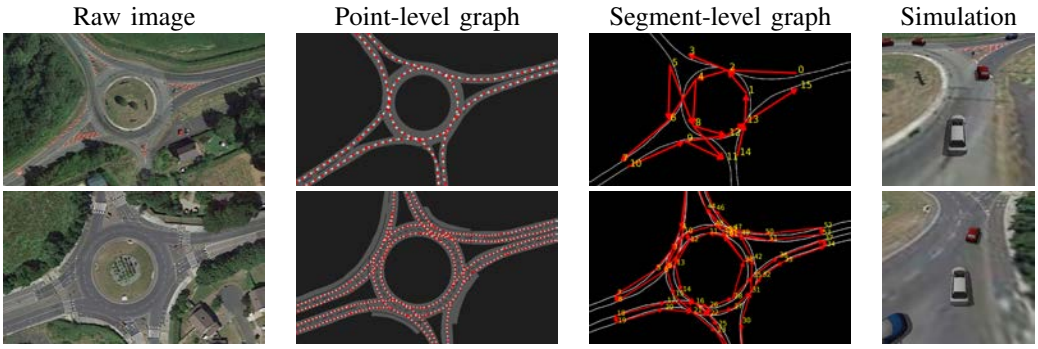

Fig. 4. Examples of collected map images. The first column shows raw images of roundabouts. The second column shows point-level graphs extracted from raw images. Red dots represent nodes, and white arrows represent edges. The third column shows segment-level graphs. Numbers in yellow represent the index of each segment and red arrows represent the connection between segments. The last column shows how each roundabout appears in a simulator.

TABLE I
NETWORK ARCHITECTURE PARAMETERS

Network	Value
ψ_{node}, ψ_{edge}	FC 5×8, FC 2×8
GCN K, N, Z, H	64, 8, 8, 3
History Length T	10
ψ_{read}	FC 512×64
Actor	FC 64×8 - FC 8×8 - FC 8×5
Critic	FC 64×8 - FC 8×8 - FC 8×1
Non-Linear function	Leaky-ReLU [22]

The number of nodes, K, is selected considering the trade-off between the computational cost and the road representability. Empirically, we found that large K requires more computational time, while small K does not fully capture road information. In the environment, the number of vehicles is kept to eight, including the ego-vehicle. The PID controller of each vehicle is executed at 30 Hz, while the action and its target speed are updated at 6 Hz. For RL training, we used the implementation of [21] but converted it into a discrete version (PPO2) [20]. We implemented the encoder and readout networks, i.e., ψ_{node}, ψ_{edge}, and ψ_{read}, using fully-connected networks. The LSTM encoding network has two stacked hidden states with the size of 64×64. The details of the network architecture is described in Table I.

B. Experimental Results

In each experiment, we choose six different maps for training and three different maps for testing. The agent cannot see maps used for testing during training. We have trained the agent with three different random seeds and tested it with 100 episodes for each random seed. We use five baseline methods for comparison. Method A and B are not learning-based methods. Method A is a random controller that selects the target speed randomly. Method B is the controller that has the knowledge of the target speed of nearby and controls the ego-vehicle with the same target speed as other vehicles. Since the main contribution of the paper is the representative power of Road-GNN, we fix an RL algorithm and compare Road-GNN against other widely used representations and network models. Both MLP and LSTM methods use the history of all the vehicle features such as the velocity and positions, but no traffic information is used. CNN method rasterizes images of the map as described in [7]. The rasterized image channels include the road lines of the maps and the history of the locations of the ego and other vehicles. The network model in the CNN method perceives an area of $32\,\mathrm{m} \times 24\,\mathrm{m}$ around the ego-vehicle with resolution of 320×240. Figure 5 shows learning curves of different algorithms we tested.

In the performance comparison experiment, we have first measured the mean speed and cumulative reward of the ego-vehicle to test if the agent can select a proper speed. As given in Table II, the proposed method shows the highest performance in terms of both the speed and cumulative reward. Meanwhile, the other RL-based method, such as MLP, LSTM, and CNN methods, cannot reach the performance of Road-GNN and failed to drive properly. They were even much slower than not only Road-GNN but also Method B which uses the same controller as other simulated vehicles. The proposed method is the only method that has outperformed Method B in terms of both the speed and the cumulative reward. Therefore, it can be said that the graph representation and Road-GNN is useful for training an RL agent and for driving complicated road environments.

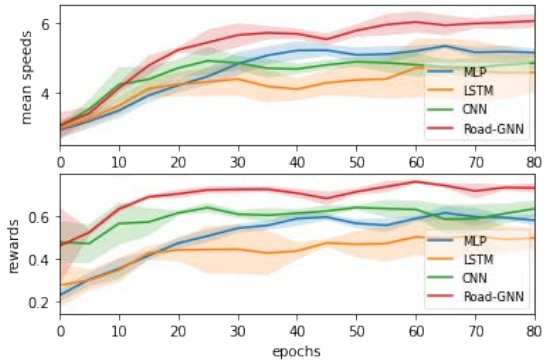

Fig. 5. Learning curve of RL algorithms. The shaded regions represent the standard deviations of the results.

TABLE II
PERFORMANCE COMPARISON

Method	Mean Speeds [m/s]	Rewards	#Param	Flops
Method A	3.035	0.407	-	-
Method B	5.525	0.727	-	-
MLP	5.009	0.582	0.104M	**0.208M**
LSTM	4.602	0.557	**0.070M**	1.396M
CNN [7]	4.747	0.641	11.528M	163.380M
Road-GNN	**5.838**	**0.743**	0.101M	3.631M
Road-GNN w/o LSTM	5.518	0.710	0.080M	2.369M

We also checked the number of parameters and flops of each network to test the efficiency of Road-GNN. While the proposed method uses about the same number of parameters as MLP and LSTM, it shows better performance than all compared methods. However, it should be taken into account that MLP and LSTM methods do not consider road structural information. Compared with CNN [7], which also can consider road structures, Road-GNN shows a better performance than CNN even though it uses fewer parameters and flops.

As an ablation study, we also trained Road-GNN without

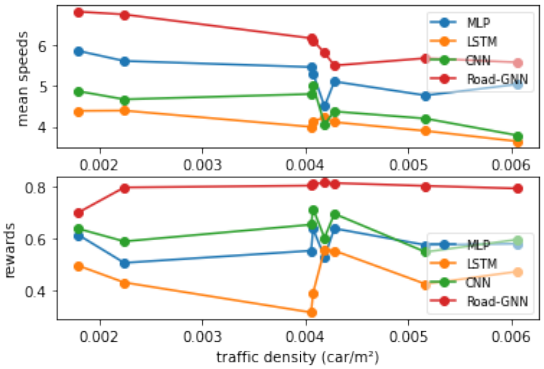

Fig. 6. Experiment with various difficulty levels. Each dot represents a test result in one map. Each method is tested 20 times per map.

TABLE III
EXPERIMENT WITH AGGRESSIVE DRIVERS

Method	Collision Rates [%]	Mean Speeds [m/s]	Rewards
Method A	48.3	2.749	0.031
Method B	19.3	5.175	0.488
MLP	18.0	4.838	0.395
LSTM	33.0	4.212	0.225
CNN [7]	22.3	4.547	0.436
Road-GNN	**13.3**	**5.716**	**0.602**

an encoder LSTM [10] model which is described in Section IV-C. We instead encoded time-series features by simple fully-connected layers. In the ablation study, Road-GNN was more effective when it uses an LSTM model. The result shows that LSTM is helpful for Road-GNN to encode time-series features although it requires more parameters.

To show that the proposed method can be generally deployed in various driving environments, we conducted an experiment on eight different maps that have different driving difficulty levels. We tested each method 20 times per map. As a metric of the difficulty, we measured the traffic density of each map, which is defined by the number of vehicles divided by the area of the road. Figure 6 shows the results of the difficulty test. The speed of the vehicles tended to decrease as the traffic density increases. As shown in Figure 6, the proposed method shows the highest performance evenly regardless of the driving difficulty of the environment.

We conducted an additional experiment to test the stability of the proposed method. We assume a more hazardous scenario where some drivers ignore the safety rules. We changed the controllers of some simulated vehicles to move more aggressively on the simulator. The target speed of two simulated vehicles is changed to $12\,\mathrm{m/s}$, which is faster than the other vehicles. We also made them ignore the collision-avoidance system. The result is shown in Table III. The proposed method shows the lowest collision rate and the highest performance, even though it has been trained in a safe environment. In conclusion, Road-GNN is stable and can drive more safely in hazardous environments.

VI. CONCLUSIONS

In this work, we have proposed a new autonomous driving framework, which can be generalized to various road environments, using the GNN based structure named Road-GNN and an RL based controller. The proposed network, Road-GNN, is designed to perceive a graph representation of a road, which includes the road connection and vehicle information. In the experiments, the proposed method has outperformed the other methods which do not use the graph representation. The results have shown that the proposed method can generalize different road structures well and the knowledge learned from roads in the training set can be easily transferred to unseen road structures. For the future works, we plan to incorporate road traffic information such as traffic lights and road signs.

REFERENCES

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

[2] K. Cho, T. Ha, G. Lee, and S. Oh, "Deep predictive autonomous driving using multi-agent joint trajectory prediction and traffic rules," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2076–2081.

[3] I. Nishitani, H. Yang, R. Guo, S. Keshavamurthy, and K. Oguchi, "Deep merging: Vehicle merging controller based on deep reinforcement learning with embedding network," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 216–221.

[4] Y. Lyu, C. Dong, and J. M. Dolan, "Fg-gmm-based interactive behavior estimation for autonomous driving vehicles in ramp merging control *," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1250–1255.

[5] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2034–2039.

[6] 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 (CoRL)*, 2017, pp. 1–16.

[7] M. Henaff, A. Canziani, and Y. LeCun, "Model-predictive policy learning with uncertainty regularization for driving in dense traffic," in *International Conference on Learning Representations (ICLR)*, 2019.

[8] 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 *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 522–11 530.

[9] 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.

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

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

[12] NAVER. (2020) Naver maps. [Online]. Available: https://map.naver.com/

[13] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[14] O. S. R. Foundation, "Demo of prius in ros/gazebo," https://github.com/osrf/car_demo, 2019.

[15] J. Halkias and J. Colyar, "Ngsim interstate 80 freeway dataset," US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA, 2006.

[16] X. Liang, T. Wang, L. Yang, and E. Xing, "Cirl: Controllable imitative reinforcement learning for vision-based self-driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 584–599.

[17] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, "Driving in dense traffic with model-free reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5385–5392.

[18] B. Osiński, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6411–6418.

[19] K. Nishi and M. Shimosaka, "Fine-grained driving behavior prediction via context-aware multi-task inverse reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2281–2287.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[22] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in Proceedings of International Conference on Machine Learning (ICML)*, vol. 30, no. 1, 2013, p. 3.