Deep Imitation Learning for Autonomous Driving in Generic Urban Scenarios with Enhanced Safety

Jianyu Chen*, Bodi Yuan* and Masayoshi Tomizuka

Abstract— The decision and planning system for autonomous driving in urban environments is hard to design. Most current methods manually design the driving policy, which can be expensive to develop and maintain at scale. Instead, with imitation learning we only need to collect data and the computer will learn and improve the driving policy automatically. However, existing imitation learning methods for autonomous driving are hardly performing well for complex urban scenarios. Moreover, the safety is not guaranteed when we use a deep neural network policy. In this paper, we proposed a framework to learn the driving policy in urban scenarios efficiently given offline connected driving data, with a safety controller incorporated to guarantee safety at test time. The experiments show that our method can achieve high performance in realistic simulations of urban driving scenarios.

I. INTRODUCTION

Decision and planning for autonomous driving in urban scenarios with dense surrounding dynamic objects is particularly challenging. The difficulties come from multiple respects: 1) Complex road conditions including different road topology, geometry and road markings in various scenarios such as intersection and roundabout; 2) Complex multi-agent interactions where their coupled future motions are unknown; 3) Various traffic rules such as traffic lights and speed limit.

The majority of autonomous driving community, including both industry and academy, is focusing on the non-learning model-based approach for decision making and planning. This model-based approach often requires manually designing the driving policy model. For example, a popular pipeline is to first do scenario-based high-level decision, then predict the future trajectory of surrounding objects, and then treat the predicted trajectories as obstacles and apply motion planning techniques to plan a trajectory for the ego vehicle [23], [18], [19], [10].

However, the manually designed policy model is often sub-optimal. There are two main reasons for this: 1) The model-based approach often requires defining some motion heuristics or at least some cost functions to indicate what does a desired decision and planning look like. However, designing an accurate cost function that can make the vehicle do what we *really* want can be extremely difficult [11]; 2) For highly entangled interactions among multiple agents, simple policy models are not adequate. However, complex behavior models such as game theoretic models are not solvable in their general form (e.g, general sum multi-player

games) with the current non-learning methods. Besides its sub-optimality, the model-based approach is also expensive with respect to development and maintenance, as it relies on human engineers to improve its performance.

While it is difficult to design a decision and planning system for autonomous driving, an experienced human driver can solve the driving problem easily, even in extremely challenging urban scenarios. Thus an alternative is to learn a driving policy from human driver experts using imitation learning. Applying imitation learning has several benefits. First, we do not need to manually design the policy model or the cost function which can be sub-optimal. Second, we only need to provide expert driving data which is not difficult to obtain at scale, and the computer will then learn a driving policy automatically.

There are already existing works of imitation learning approaches for driving which typically focus on predicting direct control commands such as steering and braking from raw sensor data such as camera images [2], [9], [6], [21]. However they can only handle simple driving tasks such as lane following. Direct mapping from raw sensor data to control is too complex, which requires a huge amount of training data to cover most situations. Besides, the end-to-end architecture lacks transparency as we cannot explain the decision making process in a neural network, which makes it hard to evaluate and debug.

A more serious problem for the current imitation learning approaches, especially with function approximation such as deep neural network, is safety. Currently no theoretical results can guarantee the safety of a policy composed of a deep neural network. Safety is the most crucial issue for autonomous driving and it must be considered strictly.

In this paper, we propose a framework which efficiently obtains the intelligence of decision making for handling complex urban scenarios using imitation learning, and then provides safety enhancement to the learned deep neural network policy. We design a bird-view representation as the input and define future trajectory as the output of the driving policy, which is similar to Waymo [1] and Uber's recent works [8]. The designed representation significantly reduces the sample complexity for imitation learning. Then a safety controller based on safe set theory is incorporated, which generates control commands to track the planned trajectory while guaranteeing safety. Experiments show that the framework is able to obtain a deep convolutional neural network policy which is intelligent enough to achieve high performance in generic urban driving scenarios, with only 100k training examples and 20 hours training time on a single

II. RELATED WORKS

The first imitation learning algorithm applied to autonomous driving was 30 years ago, when ALVINN system [20] used a 3-layer neural network to perform road following based on front camera images. Helped by recent progress in deep learning, NIVIDIA developed an end-to-end driving system using deep convolutional neural networks [2], [3], which can perform good lane following behaviors even in challenging environments where no lane markings can be recognized. Researchers also trained deep neural networks to predict the control output from camera image and evaluate their open loop performance (e.g, the prediction error). [25] used an FCN-LSTM architecture with a segmentation mask to train a deep driving policy. [24] proposed an object-centric model to predict the vehicle action with higher accuracy. Although both [25] and [24] achieved good prediction performance for complex urban scenarios, they did not provide closed loop evaluation either on real world or simulated environments. [22] used imitation learning to drive a simulated vehicle in closed loop, however it is restricted to limited scenarios such as lane following and lane changing with fixed number of surrounding vehicles.

CARLA simulator [9] has been developed and open-sourced recently. It enables training and testing autonomous driving systems in a realistic three-dimensional urban driving simulation environment. Based on CARLA, [6] used conditional imitation learning to learn an end-to-end deep policy that follows high level commands such as go straight and turn left/right. [21] defined several intermediate affordance such as distance to objects, learned a deep neural network to map camera image to the affordance, and then performed model-based control based on the affordance.

The above methods are all using front camera images as the input. However the complexity of direct visual information has limited the performance of such methods. Bird-view representation is a good way to simplify the visual information while maintaining useful information for driving. Uber [8], [7] used a rasterized image which includes information for the map and objects as the input, and learned a convolutional neural network to predict the future trajectory of the vehicle. Waymo [1] used a similar mid-to-mid representation and learned a deep model that combined with perception and control modules, could drive a vehicle through several urban scenarios.

Collision avoidance safety is an important topic in robotics. Potential field method [14] introduces an artificial field around the obstacle and push the vehicle away when the distance is close. The method is efficient, but it cannot guarantee the safety. Reachability analysis [17] calculates the reachable set of the agent's state using game theory, and constrains the agent from reaching unsafe state. However, it is computationally expensive. Planning based methods [4], [5] may achieve safe motion in real time computation, but it requires the prediction of trajectories for each object. In this work we use safe set algorithm [15], [16] to develop our safety controller, which is guaranteed to be safe, computationally efficient and do not require prediction of obstacles' future motion.

III. FRAMEWORK OVERVIEW

Our system acts as an intelligent driving agent in a closed loop environment, as shown in Fig.1. The agent receives routing and perception information from the driving environment. It then outputs the control command such as throttle, steering and braking to be applied to the ego vehicle. The system includes two main building blocks: a deep imitation learning trajectory planner and a safety & tracking controller. The deep imitation learning trajectory planner is responsible to handle almost everything about driving intelligence, such as how to follow the given route in various road conditions, how to react to surrounding objects, and how to handle different traffic light states. This module is learned end-to-end from the perception results to the planned trajectory. The safety & tracking controller is responsible to guarantee safety and handle vehicle dynamics. This module is designed with non-machine-learning methods.

In this work, we assume that we already have a functioning perception module to process the raw sensor data. For example, we can use a localization system to estimate ego vehicle pose, an object detection system to detect the bounding box, position and heading of surrounding objects, and a traffic light detector to tell the states of traffic lights. Furthermore, we have access to the High-Definition map data for the area that the ego vehicle is operating, as well as the routing information which guides the ego vehicle to a specified goal position. This is a reasonable assumption as they are not difficult to obtain with current autonomous driving technology, at least under a good weather. We will not discuss the development and properties of the perception module in this work, but concentrate on the decision and planning part.

IV. DEEP IMITATION LEARNING FOR URBAN AUTONOMOUS DRIVING

The goal of imitation learning is to learn a controller that imitates the behavior of the expert. At data collection phase, an expert (either a human driver or a controller) receives observation $\mathbf{o}_t$ and output action $\mathbf{a}_t$ at time step t. The observation-action pairs $\mathcal{D} = \{(\mathbf{o}_t, \mathbf{a}_t)\}_{t=1}^{N}$ are then stored as the dataset. Let the policy function be $f(\mathbf{o}; \theta)$, where θ is its parameter. f can be any function approximator, including deep neural network as used in this paper. The imitation learning problem is then formulated as a supervised learning problem, where the goal is the optimize the policy function parameter θ to minimize the loss function $\mathcal{L}$:

$$\min_{\theta} \sum_{(\mathbf{o}_i, \mathbf{a}_i) \in \mathcal{D}} \mathcal{L}(f(\mathbf{o}_i; \theta), \mathbf{a}_i) \qquad (1)$$

In this section, we will introduce how we design observation $\mathbf{o}$, action $\mathbf{a}$, policy function f, loss function $\mathcal{L}$, and how we augment the data to obtain a robust policy.
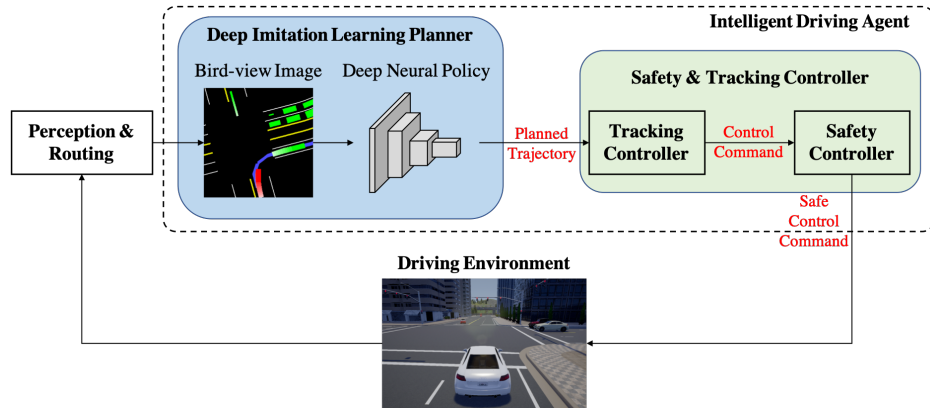
Fig. 1: *Framework overview of the our system. The agent takes information from the perception and routing modules, generates a bird-view image and outputs the planned trajectory using a deep neural policy. The safety & tracking controller then calculates the safe control command to be applied to the ego vehicle in the driving environment.*
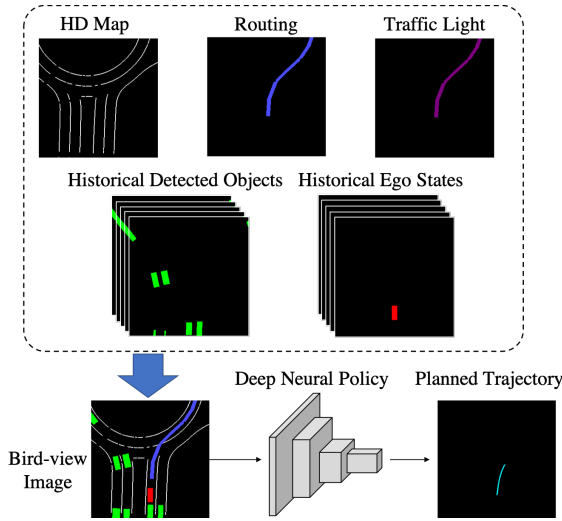

Fig. 2: *Observation-action Representation of our deep imitation learning planner. The bird-view observation combines information of HD map, routing, traffic light, historical detected objects and historical ego states. The output action is a planned trajectory represented by a vector.*

A. Observation-action Representation

A straight forward input-output representation is raw sensor data (e.g, front view camera image) for observation, and direct control command (e.g, throttle, steering, braking) for action. However, directly learning the complex mapping from raw sensor data to control output is too inefficient and hard to generalize. The raw sensor data contains extremely high dimensional information which can be influenced by different textures and appearances of roads and objects, different weather conditions, and different daytime. To allow generalization of the learned policy, the dataset needs to cover enough data for each aspect of sensor information such as texture, weather, light condition and object appearance. The direct control output is also influenced by different vehicle dynamics, thus a new policy needs to be trained if the vehicle dynamics is changed.

To alleviate this problem, we used a bird-view representation for the observation $\mathbf{o}_t$. This bird-view representation

is a concise description of only the useful information for decision making and planning, discarding irrelevant information such as texture. As shown in Fig.2, our bird-view input representation is composed of the following five parts:

1) **High-definition (HD) Map:** The HD map contains information of road conditions. Here we render all the lane markings represented by yellow or white polylines on the 2D RGB image.

2) **Routing:** The routing information is provided by a route planner after we define the start and end point. It is represented by a sequence of waypoints for the ego vehicle to follow. We render it as a thick blue polyline in the image.

3) **Traffic Light State:** When the state of the traffic light becomes red, we set the color of the route to be purple, otherwise it will maintain blue.

4) **Past Detected Objects:** The past detected surrounding objects (e.g, vehicles, bicycles, motorcycles) in a past time period are rendered as green boxes, with reduced level of brightness meaning earlier time-steps.

5) **Past Ego States:** Similar to the detected objects, the past ego states are represented as boxes with reduced brightness. The color of the boxes are set red.

The final bird-view image is rendered with pixel size 192×192, and is always aligned to the ego vehicle's local coordinate. The actual size of the field of view is $(40m, 40m)$, where the ego vehicle is positioned at $(20m, 8m)$.

The action of the policy is also altered from direct control output to trajectory $\mathbf{a}_t = [x_{t+1}, y_{t+1}, \cdots, x_{t+H}, y_{t+H}]$, where x_i and y_i are x and y position in the local coordinate of ego vehicle at time step i, H is the preview horizon and t is the current time step. While direct control output can be significantly influenced by vehicle dynamics, the future trajectory would have little difference if the vehicle dynamics does not change too much. The output trajectory can be tracked by a vehicle specific tracking controller, which is easy to design as will be illustrated in the next section.

B. Network architecture

Taking the bird-view image as input, we use a CNN model to predict ego vehicle's future trajectory. Our CNN model

has the same conv-layers as VGGNet16 [12], followed by a fully connected layer with 1000 hidden units, which is then fully connected with the final output layer with $2H$ units. The output layer represents the H predicted trajectory points $(\hat{x}_{t+i}, \hat{y}_{t+i})$ in the ego vehicle local coordinate.

We want to optimize the displacement error d_{t+i} between the expert's actual trajectory point position (x_{t+i}, y_{t+i}) and the predicted point position $(\hat{x}_{t+i}, \hat{y}_{t+i})$:

$$d_{t+i} = ((x_{t+i} - \hat{x}_{t+i})^2 + (y_{t+i} - \hat{y}_{t+i})^2)^{\frac{1}{2}}.$$

The overall loss function is defined as

$$\mathcal{L}_t = \frac{1}{H} \sum_{i=1}^{H} d_{t+i}^2.$$

C. *Data augmentation*

If we directly solve the supervised learning problem (1), the resulting policy may be unstable and the vehicle will easily run out of the road. This is due to the co-variant shift of the vanilla imitation learning algorithm, which only learns from normal driving data. In the test phase, small prediction error can be accumulated and the vehicle may reach some unseen states so that it is unable to recover.

To solve this problem, we periodically introduce noise to the expert controller during the data collection phase, and let the expert recover from the perturbation. The control noise is added every 8 seconds, and will last for 1 second. The vehicle's pose might be pushed away from the waypoints. The expert then provides demonstrations of recovering from perturbations. The states during the noise phase are removed in order not to contaminate the dataset. This data augmentation trick significantly improves the performance of the learned policy, as shown in our experiments.

V. SAFETY ENHANCEMENT & TRAJECTORY TRACKING CONTROL

Since we use deep neural network, the safety and feasibility of the planned trajectory cannot be guaranteed. In this section, the design of the safety enhancement controller and the trajectory tracking controller will be introduced.

A. *Trajectory Tracking Controller*

Given the planned future trajectory $[\hat{x}_{t+1}, \hat{y}_{t+1}, \cdots, \hat{x}_{t+H}, \hat{y}_{t+H}]$, a tracking controller is implemented to calculate the desired acceleration a_t and steering angle δ_t to drive the vehicle that follows the trajectory. A target waypoint $(\hat{x}_{t+m}, \hat{y}_{t+m})$ is selected where $1 \leq m \leq H - 1$ ($m = 5$ in this paper). The controller is then decoupled to longitudinal and lateral control:

1) Longitudinal Controller: The target speed is set to be

$$v_d = \frac{1}{dt} \|(\hat{x}_{t+m+1}, \hat{y}_{t+m+1}) - (\hat{x}_{t+m}, \hat{y}_{t+m})\|_2$$

where dt is the time interval between two consecutive time steps. The desired acceleration a_t is then obtained using PID control to eliminate the speed tracking error $e_v(t) = v_d - v(t)$, where $v(t)$ is the current speed of ego vehicle.
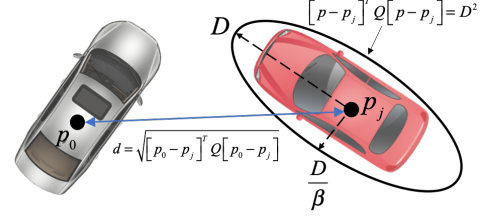

Fig. 3: *Illustration of the safety index. Gray is the ego vehicle, red is a surrounding vehicle. The safety constraint is similar to the ellipse around the red vehicle, while also considering the relative speed of the two vehicles.*

2) Lateral Controller: The normalized vector from the ego vehicle position to the target way point is $\mathbf{n}_{\text{target}} = \frac{(\hat{x}_{t+m}, \hat{y}_{t+m})}{\|(\hat{x}_{t+m}, \hat{y}_{t+m})\|_2}$. The normalized vector of the ego vehicle heading is $\mathbf{n}_{ego}(t) = (\cos\theta_t, \sin\theta_t)$, where θ_t is the yaw angle of the ego vehicle. Then the desired steering angle is obtained using PID control to eliminate the heading error:

$$e_{yaw}(t) = \cos^{-1}(\mathbf{n}_{ego}(t) \cdot \mathbf{n}_{target}(t))$$

B. *Safety Enhancement Controller*

The acceleration and steering command a_t and δ_t calculated by the tracking controller does not guarantee safety. We incorporate a safety controller that will modify a_t and δ_t to enhance safety, if their original values are not safe.

Our method is developed based on the safe set algorithm [15], [16]. The key idea is that for each time step t, we will calculate a control safe set $U_S(t)$ of the control command $u(t) = \begin{bmatrix} a_t & \delta_t \end{bmatrix}^T$. The control safe set has the property that if $u(t) \in U_S(t)$, the ego vehicle will stay safe.

To obtain $U_S(t)$, a definition of safety needs to be stated. Here we define a safety index $\phi(x)$, which is a function of the state x, where x represents states (e.g, position, velocity, heading) of both the ego vehicle and a surrounding object. In this paper, the safety index is defined as:

$$\phi(x) = D - d^2(x) - \alpha \dot{d}(x)$$

where $d(x)$ is a shaped distance between the ego vehicle and the surrounding vehicle:

$$d(x) = \sqrt{[p_0 - p_j]^T Q [p_0 - p_j]}$$

where p_0 indicates the position of the ego vehicle and p_j indicates the position of the surrounding vehicle. Q is a 2-by-2 matrix such that $[p - p_j]^T Q [p - p_j] = 1$ represents an ellipse around the surrounding vehicle with long axis equal to 1 and short axis equal to $\frac{1}{\beta}$, where β is the aspect ratio of the ellipse. Let the state safe set X_S be the level set of the safety index $X_S = \{x : \phi(x) \leq 0\}$. Then intuitively, the state safe set introduces an ellipse constraint as shown in Fig.3. It also considers the relative speed between the ego and surrounding vehicle. If their relative speed is high, it is more likely to be unsafe.

We can choose the control safe set to be $U_S(t) = \left\{ u(t) : \dot{\phi} \leq -\eta \text{ if } \phi \geq 0 \right\}$ where $\eta > 0$ is some margin. It
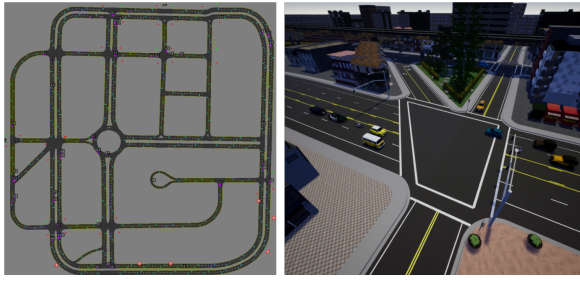
Fig. 4: *The simulation environment we use. Left is the map layout, right is a sample view at an intersection.*

can be proved that if $x(0) \in X_S$ and $u(t) \in U_S$ for $t \geq 0$, then $x(t) \in X_S$. Now if we approximate the ego vehicle dynamics to a control affine function $\dot{x} = f(x) + Bu$, the control safe set can be written as:

$$U_S(t) = \{u(t) : L(t)u(t) \leq S(t) \text{ if } \phi \geq 0\}$$

where $L(t) = \frac{\partial x_0}{\partial x_j} B$ and $S(t) = -\eta - \frac{\partial \phi}{\partial x_j} \dot{x}_j - \frac{\partial \phi}{\partial x_0} f$, x_0 and x_j are the states of the ego and surrounding vehicle, respectively.

If there are multiple surrounding objects, we can calculate the intersection of the control safe set for each object, which is a convex polytope. Letting $u(t) = \begin{bmatrix} a_t & \delta_t \end{bmatrix}^T$ denotes the control command output from the trajectory tracking controller, the safety controller maps it into the control safe set U_S by solving the following quadratic programming problem:

$$u^*(t) = \arg\min_{u \in U_S} \frac{1}{2}(u - u(t))^T W(u - u(t))$$

where W is a 2-by-2 weight matrix. We can thus obtain the modified safe control command $u^*(t) = \begin{bmatrix} a_t^* & \delta_t^* \end{bmatrix}^T$. Then the low-level controller will track the given acceleration a_t^* and steering angle δ_t^*.

Note that our safety controller is not restricted to be applied together with the specific imitation learning planner in this paper. It can be applied as a module with any upper level planner to modify their control output to enhance safety.

VI. EXPERIMENTS

A. Simulation Environment and Data Collection

We collect data and evaluate our proposed method on CARLA simulator [9]. CARLA is an open-source high-resolution simulation platform for development and validation of autonomous driving systems. It simulates not only the raw sensor data such as camera image and Lidar point cloud, but also detailed vehicle dynamics. A system evaluated on CARLA is likely to have similar performance if applied to a real driving environment. Furthermore, in our system we use the processed bird-view image as input, which has no domain difference with that of the real world and thus the policy can be easily transferred from simulation to real world.

Fig.4 shows the map layout and a sample view of the simulation environment we use for training. It includes various urban scenarios such as intersection and roundabout. The map has a range of $400m \times 400m$, containing about $6km$ total length of roads. We put 100 vehicles running

TABLE I: *Average prediction displacement error (in meters)*

	Training Condition	New Town
M_0	0.16	0.44
M_1	0.18	0.29

autonomously in the simulator to simulate a multi-agent environment. The vehicles will randomly choose a direction at the intersection, follow the route, slow down for front vehicles and stop when the traffic light is red.

At data collection phase, we use a model-based controller to act as the expert. The controller is the same as other agents. When ego vehicle is running, we record the rendered bird-view image and the corresponding ego vehicle state (global positions and yaw angle) every 0.1 second. The future ego vehicle trajectory is calculated by transforming its future global positions to the ego vehicle's current local coordinate.

B. Bird-view Image Generation

To render the bird-view input image, we build a buffer to store the historical states (position, velocity, heading, size) of all vehicles. The states are then transformed to the ego vehicle's current local coordinate. The HD map contains information of lane markings, which is extracted from the OpenDrive data provided by CARLA. Routing information is a sequence of waypoints provided by the global planner of CARLA, and is rendered as a thick blue line.

C. Training

We run the simulation for about 5 hours and generated 120k frames. 100k frames are used for training and 20k for evaluation. The model is trained from scratch using Adam optimizer [13], with initial learning rate of 10^{-4} for 30 epochs. It is then fined-tuned with learning rate of 10^{-5} for another 10 epochs. Batch size is set to 50. The model converges in about 20 hours on a single GTX 1080 Ti.

D. Models

Besides our final model with data augmentation and safety controller, we also train and test the models without data augmentation and/or without safety controller for comparison. We thus have three models: 1) M_0 - the model without data augmentation and safety controller; 2) M_1 - the model with data augmentation but without safety controller; 3) M_2 - the model with both data augmentation and safety controller.

E. Open Loop Evaluation

For open loop evaluation, we calculate the average displacement error in both Town03 (Training Condition) and Town01 (New Town), as shown in Table I.

We also did an ablation study on how data augmentation improves the performance. We notice that model M_0 performs well under most cases. However, once the vehicle runs into abnormal states, M_0 can hardly predict a good trajectory to help the vehicle recover to normal states. On the contrary, model M_1 has much better ability to help the vehicle recover. Fig.5 gives one example.

Moreover, we give several examples of model M_1's outputs. Fig.6 (a) and (b) show that it can output reasonable
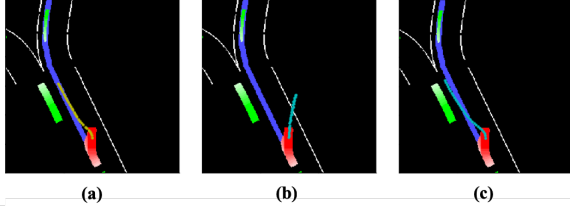
Fig. 5: *Comparison between models trained with data augmentation or without data augmentation (a) Groundtruth trajectory (b) Predicted trajectory from M_0 (c) Predicted*
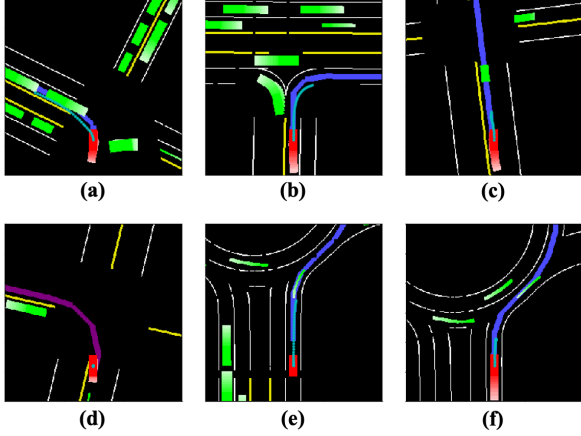

Fig. 6: *Example results: (a) Left turn at an intersection. (b) Right turn at an intersection. (c) Blocked by a front vehicle. (d) Stop at red light. (e) (f) Enter a roundabout.*

trajectories in busy intersections. Fig.6 (c) demonstrates that the model learns how to slow down and stop if there is a slow or stopped car in front of the ego vehicle. Fig.6 (d) gives one example that our vehicle stops at the red light. Fig.6 (e) and (f) are examples of entering a roundabout. We can see that the model learns to yield to other vehicles when entering the roundabout in (f).

F. Closed Loop Evaluation

We also implemented our system in CARLA simulator for closed-loop evaluation. For every 0.1 second, we receive the environment information and render the corresponding bird-view image. The deep neural network policy then performs forward inference and output a predicted trajectory. The trajectory is sent to the tracking controller and then the safety controller to output a control command. The control command is then applied on the ego vehicle in the simulator. This process is repeated until it reaches some terminal criterion. We then evaluate the performance of our models under several urban driving cases and different towns.

1) Evaluation Metrics: Similar to the metrics designed in [9], we have two metrics for the closed-loop evaluation. The first is success rate, this metric is applied to some specific urban driving cases such as intersection and roundabout. To calculate the success rate, a start and end point are defined for each case, such as start at several meters before entering an intersection/roundabout, and end at several meters after passing through it. Note here we do not report results on simple cases such as lane following as stated

TABLE II: *Success rate for the intersection and roundabout scenarios evaluated on our three models M_0, M_1 and M_2. The value represent percentage of success trials*

Task	M_0	M_1	M_2
Intersection	16%	96%	**100%**
Roundabout	12%	84%	**96%**

in [9], because the model M_2 can succeed 100%. Instead, we perform experiments on two complex cases including a signalized intersection and a roundabout with multiple surrounding dynamic objects. We compare our three models M_0, M_1 and M_2 under the success rate metric.

The second metric is infraction analysis, which we define as the average distance the ego vehicle can run between two collision or out-of-lane events. This definition is a little different with the infraction metric in [9], where they classify collision events with respect to different kinds of objects such as vehicles, bicycles and pedestrians. In this paper, a collision event means collision to any objects. Since we do not have pedestrians in our environment, in order to compare with methods stated in [9], we use the smaller infraction value of their collision-vehicle and collision-bicycle events. This is reasonable because their total collision rate must be higher than that of any single collision type. Our definition for out-of-lane events contains both cases of running to the opposite lane and to the sidewalk, as stated. Similarly, we choose the smaller infraction value to compare. We compare 7 models, including our three models, as well as Modular Pipeline (MP), Conditional Imitation Learning (CIL), Reinforcement Learning (RL) and Conditional Affordance Learning (CAL) shown in [9], [6], [21]. We also evaluate our performance at a new town (Town01) to see its generalization. Note that we do not evaluate new weather conditions, because our method will not be influenced by different weather because the bird-view representation is not influenced by weather conditions.

2) Evaluation Results: Table II shows the success rate for both the intersection and roundabout scenarios of our three models, where we performed 50 trials for each scenarios and each model. We can see that without data augmentation and safety controller, the vehicle can hardly pass these complex urban scenarios as they cannot even make successful turns at a relatively sharp curve road. When trained with augmented data, the success rate improves significantly. Then when safety controller is added, our final model M_2 can almost perfectly solve the given scenarios.

Table III shows the infraction of our methods and the existing methods on both our training town (Town03) and a new town (Town01), where we performed 50 trials for each model and each town, with 5 minutes for each trial. We then divide the total distance by the total number of infractions to get the infraction value. We can see that our final model M_2 outperforms all learning-based methods on both the out-of-lane and collision metrics. The performance of our model M_2 is similar to the performance of the modular pipeline under training condition. But for the new town, our model significantly outperforms all other methods.

Note that our training condition is much more complex

TABLE III: *Infraction analysis for driving in the training condition (Town03) and new town (Town01) using our three models and other existing methods. The value represents average kilometers traveled between two infractions*

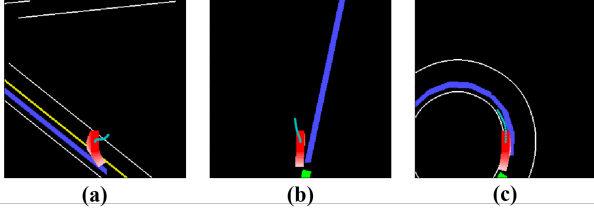| Infraction Type	Training Condition							New Town						
	MP	CIL	RL	CAL	M_0	M_1	M_2	MP	CIL	RL	CAL	M_0	M_1	M_2
Out of lane	10.2	12.9	0.18	6.1	0.30	6.92	**17.7**	0.45	0.76	0.23	0.88	0.29	3.77	**5.9**
Collision	**10.0**	3.26	0.42	2.5	0.81	3.95	8.88	0.44	0.40	0.23	0.36	0.44	4.53	**11.7**

(a) **(b)** **(c)**

Fig. 7: *Failure Cases* **(a)** *driving on lane with yellow lane marking on the right* **(b)** *driving on lane with no lane markings* **(c)** *driving on roundabout with fence*

than the one in [9], where they train it in Town01 and we train in Town03. Town01 contains only single lane roads with almost no curve roads, and there is no roundabout.

G. Failure Cases

Here we analyze three interesting failure cases we found during our evaluation. Fig.7(a) shows a case where the ego vehicle is initialized on a lane where the yellow lane marking is on its right. This makes it look like driving on the opposite lane. As a result, the planned trajectory tries to steer back to the "correct" direction. Although in this case the vehicle fails to follow the given route, the policy has learned something about the structure of the road. Fig.7(b) shows a case where there are no lane markings but only the routing information. The vehicle then goes out of lane when there's a fast vehicle behind. Providing more information such as road boundary should help with this situation. Fig.7(c) shows a case where the vehicle hits on the fence at a small roundabout. This is because there is no concept of collision in our current model. Reinforcement learning can be incorporated to solve this problem by adding penalties of hitting obstacles.

VII. CONCLUSION

In this paper, we proposed and implemented a system to learn a driving policy in generic urban scenarios given offline collected expert driving data, and enhanced the collision avoidance safety. We evaluated our methods on CARLA simulator and found our performance outperform the existing learning-based methods.

In this work we directly get the ground truth information about objects and roads from the simulator, which is impossible in real world. Thus a perception module needs to be developed and the influence of its performance to our system needs to be studied. Furthermore, reinforcement learning methods can be incorporated with our imitation learning model to improve the performance.

REFERENCES

[1] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[2] M. Bojarski et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] M. Bojarski et al. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[4] J. Chen, C. Liu, and M. Tomizuka. Foad: Fast optimization-based autonomous driving motion planner. In *2018 Annual American Control Conference (ACC)*, pages 4725–4732. IEEE, 2018.

[5] J. Chen, W. Zhan, and M. Tomizuka. Constrained iterative lqr for on-road autonomous driving motion planning. In *2017 IEEE 20th International Conference on intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2017.

[6] F. Codevilla et al. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[7] H. Cui at al. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. *arXiv preprint arXiv:1809.10732*, 2018.

[8] N. Djuric et al. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *arXiv preprint arXiv:1808.05819*, 2018.

[9] A. Dosovitskiy et al. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[10] D. González, J. Pérez, V. Milanés, and F. Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1135–1145, 2016.

[11] D. Hadfield-Menell at al. Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774, 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[14] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1398–1404. IEEE, 1991.

[15] C. Liu, J. Chen, T.-D. Nguyen, and M. Tomizuka. The robustly-safe automated driving system for enhanced active safety. Technical report, SAE Technical Paper, 2017.

[16] C. Liu and M. Tomizuka. Enabling safe freeway driving for automated vehicles. In *2016 American Control Conference (ACC)*, pages 3461–3467. IEEE, 2016.

[17] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, 50(7):947–957, 2005.

[18] M. Montemerlo at al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[19] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[20] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[21] A. Sauer et al. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018.

[22] L. Sun, C. Peng, W. Zhan, and M. Tomizuka. A fast integrated planning and control framework for autonomous driving via imitation learning. In *ASME 2018 Dynamic Systems and Control Conference*, pages V003T37A012–V003T37A012. American Society of Mechanical Engineers, 2018.

[23] S. Thrun et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[24] D. Wang et al. Deep object centric policies for autonomous driving. *arXiv preprint arXiv:1811.05432*, 2018.

[25] H. Xu el at. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.