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 are to manually design the driving policy, which can be sub-optimal and expensive to develop and maintain at scale. Instead, with imitation learning we only need to collect data and then 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 three-dimensional simulations of urban driving scenarios, with only hours of data collection and training on a single consumer GPU.

I. INTRODUCTION

Decision and planning for autonomous driving in urban environments navigation 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, roundabout and ramp merging; 2) Complex multi-agent interactions where their coupled future motion is unknown; 3) Various traffic rules such as traffic lights and speed limit; 4) Fast computation required to react to highly dynamic surrounding objects and potential emergencies.

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 pretty 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 [22], [18], [19], [10].

However, the manually designed policy model is often sub-optimal. There are two main reasons of why: 1) The model-based approach often requires defining some motion heuristics or at least some cost functions to indicates 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 between multiple agents, simple policy models are not adequate. However, complex behavior models such as game theoretic models are not solvable by the current non-learning methods under their general form (e.g., general sum multi-player games). Besides its sub-optimality, the model-based approach is also expensive with respect to infrastructure development and maintenance.

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, the computer will then learn a driving policy automatically.

There are already existing works for imitation learning approaches to driving which typically focus on predicting direct control commands such as steering and braking from raw sensor input such as camera [2], [9], [6], [21]. However they generally do not achieve enough good results to make complex decision making. Direct mapping from raw sensor data to control is too complex, which needs a huge amount of training data, resulting in high sample complexity. 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 networks, is safety. Currently no theoretical results can guarantee the safety of a policy composed of a neural network. However, safety is the most crucial issue for autonomous driving, no one would like to use an autonomous driving system that safety guarantee is impossible.

In this paper, we propose a framework which provides a high sample efficiency imitation learning to obtain the intelligence of decision making for handling complex urban scenarios, and then provides safety enhancement to the learned deep neural network policy. We design a bird-view representation as our input and define future trajectory as our output, 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 collision avoidance safety. Our experiments show that our framework is able to obtain a high capability 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 GTX 1080 Ti.

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 from raw camera sensor data. Helped by recent progress in deep learning, NVIDIA 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 road markings can be recognized. Researchers also trained deep neural networks to predict the control command from camera image and evaluate their open loop performance (e.g. the prediction error). [24] used a FCN-LSTM architecture with a segmentation mask to train a deep driving policy. [23] proposed an object-centric model to predict the vehicle action with higher accuracy. Although both [24] and [23] can achieve good prediction performance for complex urban scenarios, they did not provide closed loop evaluation either on real world or simulated vehicles.

CARLA simulator [9] has been developed and opensourced 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 lean 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 direct camera image as the input representation. 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 agent of interest. Waymo [1] used a similar mid-to-mid representation and learned a deep model that combined with a perception and control model, 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 will 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’s computational 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 [15], [16] algorithm to develop our safety controller, which is guaranteed to be safe, computational efficient and do not require prediction of obstacles.

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 command and perception information from the driving environment, which can be either a simulated or a real 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 routing 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 get ego vehicle pose estimation, 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 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 key idea of imitation learning is to learn a controller that can imitates the behavior of the expert. At the data collection phase, an expert (either a human driver or a controller) receives the observation \( o \) and output action \( a \) at time step \( t \). We then store the observation-action pairs \( \mathcal{D} = \{ (o_i, a_i) \}_{i=1}^N \) as our dataset. Let the policy function be \( f(o; \theta) \), where \( \theta \) is its parameter. \( f \) can be any function approximators, including the deep neural network used in our work. The imitation learning problem is then formulated as a supervised learning problem, where the goal is the optimize the policy function parameter \( \theta \) so that to minimize some loss function \( \mathcal{L} \): \[
\min_{\theta} \sum_{(o_i, a_i) \in \mathcal{D}} \mathcal{L}(f(o_i; \theta), a_i)
\]

In this section, we will introduce how we design the observation \( o \), the action \( a \), the policy function \( f \), the loss function \( \mathcal{L} \), and how we augment the data to obtain a robust policy.
Fig. 1: Framework overview of 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.

A. Observation-action Representation

A straightforward input-output representation is the raw sensor data (e.g., front view camera image) for the observation, and the direct control command (e.g., throttle, steering, braking) for the action. However, learning the complex mapping from raw sensor to direct control end-to-end in one step is too sample 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 make the learned policy generalize well, the dataset needs to cover enough data for each dimension of sensor information such as texture, weather, light conditions and object appearances. The direct control command is also influenced by different vehicle dynamics, thus a new policy should be trained if the vehicle dynamics is changed.

To alleviate this problem, we use a bird-view representation as the observation $o_t$. This bird-view representation is a concise description of only the useful information for decision making and planning, discarding irrelevant information such as texture, light conditions and object appearances. As shown in Fig. 2, our bird-view input representation is composed of the following five parts:

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

2) **Routing**: The routing information is calculated by a route planner after we define the start and end point. It is represented by a sequence of way points 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 influencing the ego vehicle becomes red, we set the color of the routing to be meta, otherwise it will maintain blue.

4) **Historical Detected Objects**: The historical bounding boxes of detected surrounding objects (e.g., vehicles, bicycles, pedestrians) in a past sliding window are rendered as green boxes, with reduced level of brightness meaning earlier time-steps.

5) **Historical Ego States**: Similar to the detected objects, the historical 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 $192 \times 192$, and is always aligned with the ego vehicle view. 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 command to future trajectory $a_t = [x_{t+1}, y_{t+1}, \cdots, x_{t+H}, y_{t+H}]$, 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. Although direct control command could 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 input the bird-view image as input, we use CNN model to future trajectory. Our CNN model has the same conv-layers as VGGNet16 [12], followed by a fully connected layer with 1000 hidden units, and then fully connected with the final output layer of 2H units. The output layer corresponds to \( H \) predicted trajectory points \((\hat{x}_{t+i}, \hat{y}_{t+i})\) in the ego vehicle local coordinate.

We hope to optimize the displacement error \( d_{t+i} \) between the expert’s motion trajectory point position \((\hat{x}_{t+i}, \hat{y}_{t+i})\) 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

\[
L_t = \frac{1}{H} \sum_{i=1}^{H} d_{t+i}^2.
\]

C. Data augmentation

If we directly solve the supervised learning problem \( \mathcal{L} \), the resulting policy will 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 might reach some unseen states so that it is unable to recover. For example, in our experiment, the expert controller will always keep close to the given way points, then once the vehicle get a little away from the way points, it cannot come back because it haven’t been taught what to do in this situation.

To solve this problem, we introduce control noise to the expert controller during the data collection phase, and let the expert recover from the perturbation. The control noise is added periodically every 8 seconds, and will last for 1 second. The vehicle’s pose might be pushed away from the way points. 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

As a well-known pitfall of deep learning techniques, no guarantee can be stated about the properties of the output of the learned deep neural network. In our case, we use deep neural network to plan the trajectory, thus its safety and dynamics feasibility 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 \([\hat{x}_{t+1}, \hat{y}_{t+1} \ldots \hat{x}_{t+H}, \hat{y}_{t+H}]\), a tracking controller is implemented to calculate the desired acceleration \(a_t\) and steering angle \(\delta_t\) to drive the vehicle that follows the trajectory. A way point \((\hat{x}_{t+m}, \hat{y}_{t+m})\) is selected where \(1 \leq m \leq H - 1\) (we pick \(m = 5\) here). 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 to subsequent steps. The desired acceleration \(a_t\) is then obtained using PID control to minimize the speed tracking error \(e_v(t) = v_d - v(t)\), where \(v(t)\) is the current speed of ego vehicle.

2) Lateral Controller: The normal 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 normal vector of the ego vehicle heading is \(\mathbf{n}_{\text{ego}}(t) = (\cos \theta_t, \sin \theta_t)\), where \(\theta_t\) is the yaw angle of the ego vehicle. Then the desired steering angle is obtained using PID control to minimize the heading error:

\[
e_{\text{yaw}}(t) = \cos^{-1}(\mathbf{n}_{\text{ego}}(t) \cdot \mathbf{n}_{\text{ego}}(t))
\]

B. Safety Enhancement Controller

The acceleration and steering command \(a_t\) and \(\delta_t\) calculated by the tracking controller does not guarantee safety, e.g. no collisions to other agents. Thus here we incorporate a safety controller that will modify \(a_t\) and \(\delta_t\) 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 \(t\), it calculates a control safe set \(U_S(t)\) of control command \(u(t) = [a_t \delta_t]^T\). The control safe set has the property that if \(u(t) \in U_S(t)\), the ego vehicle would 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 \(x\) includes states (e.g. position, velocity, heading) of both the ego vehicle and a surrounding object. In this work, the safety index is defined as:

\[
\phi(x) = D - d^2(x) - \alpha d(x)
\]

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 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 \(\beta\) is the aspect ratio.

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.
The simulation environment we use. Left is the map layout, right is a sample view at an intersection.

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 is like to inject an ellipse constraint as shown in Fig. 3. It also considers the relative speed between the ego and surrounding vehicle, that if their relative speed is high, it is more likely to be unsafe.

We can choose the control safe set to be \( U_S(t) = \{ u(t) : \dot{\phi} \leq \eta \text{ if } \phi \geq 0 \} \) where \( \eta > 0 \). It is easy to prove that if \( x(0) \in X_S \) and \( u(t) \in U_S(t) \) 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 \( L(t) = \frac{\partial x}{\partial x_j} B \) and \( S(t) = -\eta - \frac{\partial \phi}{\partial x_j} x_j - \frac{\partial \phi}{\partial x_k} J, \) \( x_0 \) 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 that we just denote as \( U_S \) here. Let \( u(t) = [a_i, \delta_i]^T \) 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 thus obtain the modified safe control command \( u^*(t) = [a_i^*, \delta_i^*]^T \). Then the low-level controller will track the given acceleration \( a_i^* \) and steering angle \( \delta_i^* \).

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 inputs such as camera and Lidar, but also different weather conditions (Sunshine, raining, dark...), road conditions (urban, highway,...) and detailed vehicle dynamics. A system evaluated on CARLA simulator is likely to have similar performance even directly applied 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 can be easily transferred.

Fig. 4 shows the map layout and sample views of urban driving simulation environment we use for training. It includes various urban scenarios including intersection, round-about and merging. The map has the range of 400m \( \times \) 400m, containing about 6km total length of roads.

A system evaluated on CARLA simulator is likely to have different weather conditions (Sunshine, raining, dark...), road resolution simulation platform for development and validation. Furthermore, in our system we use the processed CARLA simulator [9].

TABLE I: Average prediction displacement error (in meters)

|                  | Training Condition | New Town |
|------------------|--------------------|----------|
| \( M_0 \)       | 0.16               | 0.44     |
| \( M_1 \)       | 0.18               | 0.29     |

We run simulation for about 5 hours and generated 120k frames. 100k frames are used for training and 20k for validation. 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.

B. Training

We run simulation for about 5 hours and generated 120k frames. 100k frames are used for training and 20k for validation. 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.

C. 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 control 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.

D. 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 1.

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.

Moreover, we give several examples of our model \( M_1 \)'s output. Fig. 6 (a) and (b) show that our model can output
reasonable trajectories in busy intersections. Fig. 5(c) demonstrate 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. 5(d) gives one example that our vehicle stops at the red light. Fig. 5(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).

E. Closed Loop Evaluation

We 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 to calculate a predicted trajectory. The trajectory is sent to the tracking controller and then the safety controller to output 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 the 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 go straight and turn as stated in [9], because our model can succeed perfectly. Instead, we perform experiments on two complex cases including an 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 collisions or out of lanes. This definition is a little different with the infraction metric in [9], where they separate collision with respect to different objects such as vehicles, bicycles and pedestrians. Our collision means collision to any objects. Since we do not have pedestrians in our environment, in order to compare with other methods stated in [9], we use the smaller infraction value of their collision-vehicle and collision-bicycle. This is reasonable because their total collision rate must be bigger that the collision rate of any single collision type. Our definition for out of lanes contains both cases of running to the opposite lane or to the sidewalk, as stated in [9]. Similarly, we choose the smaller infraction value to compare. For the infraction metric, 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) described 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 as our input representation is the same for all weather conditions.

2) Evaluation Results: Table II shows the success rate for both 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%     |

Table II: Success rate for the intersection and roundabout scenarios evaluated on our three models \(M_0, M_1\) and \(M_2\).
under training condition. But for the new town, our model significantly outperforms all other methods.

Note that our training condition is much more complex 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.

F. Failure Cases

Here we introduce 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 failed to follow the given routing, it shows that 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, 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 the 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 could outperform the existing learning-based methods on the CARLA benchmark.

In this work we directly get the perception information 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 upon our imitation model to improve the performance.

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 | M₀ | M₁ | M₂ | M₀ | M₁ | M₂ |
| 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 | 59 |
| Collision       | 10.0 | 3.26 | 0.42 | 2.5 | 0.81 | 3.95 | 8.88 | 0.44 | 0.40 | 0.23 | 0.36 | 0.34 | 4.43 | 11.7 |

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

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] C. 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 et al. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. arXiv preprint arXiv:1809.10722, 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 et 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. Proceedings. 1991 IEEE International Conference on Robotics and Automation, pages 1398–1404. IEEE, 1991.
[19] B. Paden, M. Cáp, 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:1711.03938, 2017.
[22] D. Wang et al. Deep object centric policies for autonomous driving. arXiv preprint arXiv:1811.05432, 2018.
[23] 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.