Imitation Learning of Hierarchical Driving Model: from Continuous Intention to Continuous Trajectory

Yunkai Wang, Dongkun Zhang, Jingke Wang, Zexi Chen, Yue Wang†, Rong Xiong

Abstract—One of the challenges to reduce the gap between the machine and the human level driving is how to endow the system with the learning capacity to deal with the coupled complexity of environments, intentions, and dynamics. In this paper, we propose a hierarchical driving model with explicit model of continuous intention and continuous dynamics, which decouples the complexity in the observation-to-action reasoning in the human driving data. Specifically, the continuous intention module takes the route planning and perception to generate a potential map encoded with obstacles and goals being expressed as grid based potentials. Then, the potential map is regarded as a condition, together with the current dynamics, to generate the trajectory. The trajectory is modeled by a network based continuous function approximator, which naturally reserves the derivatives for high-order supervision without any additional parameters. Finally, we validate our method on both datasets and simulators, demonstrating superior performance. The method is also deployed on the real vehicle with loop latency, validating its effectiveness. Our code is available at https://github.com/ZJU-Robotics-Lab/RoBoCar

I. INTRODUCTION

Autonomous driving is an appealing research topic in recent years because of its ability to reduce labor costs and traffic accidents. In typical autonomous driving systems, vehicles need to perform observation-to-action reasoning to generate safe and efficient driving in various scenarios. One of the challenges to reduce the gap between the machine and the human level driving is how to endow the system with the learning capacity to deal with the coupled complexity of environments, intentions, and dynamics [1]. Toward this goal, there are several learning paradigms. The most mature paradigm is to decompose the observation-to-action reasoning into separated modules e.g. detection, tracking, and planning, which has good generalization given the carefully human-designed system structure, but it calls for intensive human supervision for each module [2]. To relieve the difficulty, another paradigm is to model the observation-to-action reasoning as a black box, thus massive labeled data can be automatically generated by simply recording the human driving process [3]. However, this paradigm has much lower generalization, partly due to the almost missing of the human system design, losing the human knowledge intrinsically reserved in the system structure [4].

In this paper, we follow the second paradigm to only use the annotation-free data to learn a vehicle driving model. At the same time, motivated by the idea in the first paradigm, we also inject human knowledge via the system structure design, which is to explicitly model a hierarchical structure to decouple the intentions and dynamics from the human driving data, as shown in Fig. 1.

Specifically, to model the trajectory, we regard the network as a continuous function approximator, upon which the high-order dynamics can be naturally derived and constrained without additional parameterization, leveraging the position, velocity, and acceleration in the human driving data. In contrast, the previous works mainly use only the discrete positions as supervision, which loses the continuous motion characteristics of the demonstration data [5]. Some previous works also use additional outputs to capture the velocity, but they fail to guarantee the relation of derivatives between position and velocity [5] [6]. To the best of our knowledge, this is the first work to represent the driving trajectory as a continuous function approximator network. For the high-level intention, we follow our previous work [7] to model the driving intention generation process as a mapping from coarse route planning to a potential map without time parametrization. Therefore, the intention generation can be regarded as a continuous tactical decision maker learned from human data. Since many previous works model the intention as a discrete command [8] e.g. turn left and go straight, their motion model has to learn a process actually with larger variance than ours.

As one can see, beyond imitating the human driving data, we imitate the design of the conventional hierarchical planning system structure that consists of a local path planner and a trajectory planner, to decouple the intention and dynamics. From a network training perspective, the potential map is explicitly supervised as a side task [9], which acts as a semantic regularizer and brings better interpretability.
The hierarchical supervision can also be regarded as self-supervised augmentation [10] by representing the human driving data in both time-dependent and independent forms. These designs are expected to improve the generalization of the proposed driving model. To summarize, the contribution of the papers are

- A function approximator network to approximate the trajectory in a continuous manner, which naturally reserves the derivatives for high-order supervision.
- A hierarchical driving model is explicitly designed with intention and dynamics modules, which decouples the complexity in the observation-to-action reasoning.
- Open-loop and close-loop validation on both datasets and simulations. Besides, practical implementation is proposed to tolerate the network latency, which enables generalization testing in real world.

II. RELATED WORK

In 2016, benefiting from powerful GPUs and deep neural networks, Bojarski et al. [3] developed DAVE-2 and tested their method on a real-world platform to prove that deep learning methods can achieve autonomous driving. Although this method can just achieve lane following in simple scenarios, it was a bold start and inspired the following works including [11] [12].

Because the above algorithms cannot turn at intersections to follow routes, they are unable to complete their navigation tasks. Some following algorithms try to fusion driving intentions with deep neural networks, thus they can navigate on a route set by humans or route planning algorithms. Driving intentions can be represented in many forms, such as network choosing commands [8] [9] [13] [14], one-hot vector [15] and routed map [16]. [8] is the first proposed method to select networks by input commands, [13] uses imitation learning to accelerate the initial exploration process of reinforcement learning algorithms rather than learning driving from scratch. [9] proposes to learn basic knowledge such as depth map and segmentation to guide the learning of the driving module. [14] exploring the limitations of behavior cloning for autonomous driving, and proposes that the speed prediction branch can improve driving mission success rates for its regularization effect. Similarly, we not only implicitly increase the prediction of velocity, but also the prediction of acceleration.

The above algorithms can finish closed-loop navigation tasks both in simulation and real-world, however, there are still many challenges, such as environmental understanding, high-dimensional action space exploration, and multi-modal distribution of training data, these lead to the creation of many recent algorithms. In the viewpoint of intention modeling, [17] models discrete intentions as hierarchical behaviors to explicitly improve tactical driving by reinforcement learning (RL). [7] uses Generative Adversarial Networks (GAN) to model continuous driving intentions and increases understanding of the environment. We follow this method and extend it. [18] [19] use Graph Neural Network (GNN) to model implicit intentions in relation to surroundings such as road-agents and road components. Multi-modal distribution of intentions is also an important research direction, [15] use GAN to generate multiple plausible paths, similar to [20]. [16] uses Gaussian Mixture Model (GMM) to model multi-modal distribution of behaviors, similar to [21] and [22]. In the viewpoint of dynamics representation, most works represent dynamics as a series of waypoints and velocities [15] [5]. [23] models dynamics as the distribution of velocities and yaw angles. [22] models dynamics as distribution of the coefficients of second-order polynomial trajectories. Different from the above works, we model dynamics as a continuous function approximator network with arbitrary higher-order derivative.

III. METHODOLOGY

The overall system architecture of our proposed method is shown in Fig. [2] which is divided into a driving intention module and a trajectory generation module. The driving intention module maps the image $V$ and routing planning $R$ into continuous intention $I$, model it as goal guided potential, and further inject it with obstacle potentials, generating a potential map $C$. Then the trajectory generation module maps a stack of past potential maps $C_{t_0-K:t_0}$, and the current velocity $v_{t_0}$ into a continuous trajectory $T$ that has high-order smoothness.

A. Driving Intention Module

The driving intention module is an extension of our previous work [7], which uses an undifferentiable motion planner to generate control, thus blocking the end-to-end learning. We first briefly introduce the generation of continuous intention and the potential map.

Intention Modeling: The module takes as input front-view RGB image $V$ combined with routing planning $R$. The routing planning $R$ is generated by usual navigation software, like GoogleMap, and represented as a local crop of the map centering on the current vehicle position from GPS. Thus $R$ is a robot-centric representation, which changes its coordinates when the robot moves. We model the continuous driving intention $I$ as a mask of $V$ to indicate the local path to the goal in the current image coordinates, which is generated by a U-Net based conditional generator $G$ with a concatenation of $V$ and $R$ being the input:

$$I = G(V, R)$$

To build the ground truth $I_{gt}$ of $I$, we refer to the future path executed by the robot in the real world and transform it into the current image coordinates. Therefore, we have a pixel level supervision for $I$ formulated as a loss term:

$$L_{int}(G) = \mathbb{E}_{V,R,I_{gt}}[||I_{gt} - G(V, R)||_1]$$

Adversarial Training: As the robot only execute in one direction in an intersection in the real world, we only have one-way ground truth for image taken at intersections. For other directions in the intersection, we cannot build $I_{gt}$ To improve the diversity, we change the routing planning $R$, and utilize an adversarial training for supervision. Specifically,
we build a discriminator $D$ to discriminate whether the driving intention map is generated from the distribution of ground truth data $I_{gt}$, so that exact one-to-one supervision is not required, leading to a conditional adversarial loss term:

$$
\mathcal{L}_{\text{intadv}}(G, D) = \mathbb{E}_{V,R,I_{gt}} [\log D(V, R, I_{gt})] + \mathbb{E}_{R,V} [\log (1 - D(V, R, G(V, R)))]
$$

(3)

**Potential Map Building:** Given the driving intention conditional by the global routing, we actually have goal guidance in robot-centric coordinate. As this mask must lie on the local ground plane, we can transform $I$ into a bird-eye view representation. By smoothing the binary mask, we have a goal guided potential map $C_{goal}$. Furthermore, we represent the perception result as an obstacle potential map $C_{obstacle}$ in the same bird-eye view coordinates. In this paper, we use LiDAR for obstacle detection, but any other obstacle perception is usable. Combining the two potential maps, we have a local potential map $C$ for motion planning, which can be regarded as a local artificial potential field.

**B. Trajectory Generation Module**

The trajectory generation module acts as the motion planner in a conventional pipeline, but it is differentiable, leveraging the end-to-end imitation learning from the human demonstration driving data. The main difficulty for this module is to keep the continuous and smooth nature of the trajectory. Thus the down-stream controller can track the trajectory in a high frequency asynchronously.

**Continuous Function Modeling:** For a network with linear decoder, we have a linear mapping from the last hidden layer $f(z)$ to the output $y$ as

$$
y = \sum_i w_i f(z_i) + b
$$

(4)

where $w_i$ and $b$ are the network parameters, $f$ is the nonlinear activation function. Furthermore, regarding the input to the network as condition $c$ and a function variable $x$, we have an interpretation of (4) as

$$
y(x) = \sum_i w_i f(z_i, c(x)) + b
$$

(5)

In this form, the nonlinear activation functions of the last hidden layer form a set of basis functions $f$ parameterized by $z_i,c$, deriving a linear combination of basis functions to represent $y(x)$. Compared with the conventional basis functions, the basis functions of (5) have adaptive forms inferred by the input condition $c$ and $x$. Consequently, this form provides a continuous function approximator, of which the high-order derivatives are naturally derived during the network back-propagation process:

$$
\frac{\partial y}{\partial x} = \sum_i w_i \frac{\partial f}{\partial z_i,c} \frac{\partial z_i,c}{\partial x}
$$

$$
\frac{\partial^2 y}{\partial x^2} = \sum_i w_i \left( \frac{\partial^2 f}{\partial z_i,c} \frac{\partial z_i,c}{\partial x} + \frac{\partial f^2}{\partial z_i,c} \frac{\partial z_i,c}{\partial x} \right)
$$

(6)

The 2nd order derivative of $y(x)$ is the Hessian matrix. When the dimension of $x$ is high, the computation is intractable. However, if the continuous function is parameterized by low dimensional variable $x$, evaluation of the high-order derivative is feasible.

**Trajectory Representation:** At the perspective of continuous function approximator (5), we therefore use it for trajectory learning. Note that the variable of trajectory $T$ is the time $t$, we have the high-order derivatives for trajectory i.e. the velocity $v(t)$ and acceleration $a(t)$:

$$
v(t) = \frac{\partial T(t)}{\partial t}
$$

$$
a(t) = \frac{\partial^2 T(t)}{\partial t^2}
$$

(7)

By simply regarding $x$ as $t$, we represent a continuous trajectory $y(t)$ as a neural network with high-order derivatives, and...
is differentiable for learning. Thanks to the 1-dimensional variable \( t \), the evaluation of (6) is very efficient. Moreover, inspired by the conventional Fourier analysis, which is very effective for modeling dynamic system input and output, we also use sinusoidal function \([24]\) as the form of nonlinear activation function \( z \), instead of the usual ReLU functions.

Finally, we have the neural trajectory as

\[
y(t) = \sum_i w_i \cos(z_{i,c}(t)) + b \tag{8}
\]

This function can be supervised by the any order of derivatives to learn the parameters \([24]\). In this paper, at time \( t_0 \), we build loss terms based on the observed position \( \hat{y}(k) \), velocity \( \dot{v}(k) \) and the acceleration \( \ddot{a}(k) \) at several sampling time \( k \):

\[
\mathcal{L}_T(y) = E_{t_0}[ \sum_{k \in \{t_0, t_0+T\}} ||\hat{y}(k) - y(k)||^2_2 \\
+ \lambda_1 ||\dot{v}(k) - \frac{\partial y(t)}{\partial t}|_{t=k}||^2_2 \\
+ \lambda_2 ||\ddot{a}(k) - \frac{\partial^2 y(t)}{\partial t^2}|_{t=k}||^2_2] \tag{9}
\]

where \( T \) is the time horizon for the trajectory.

We briefly summarize the advantage of modeling continuous trajectory as the smoothness for execution, which is guaranteed by the infinitely order differentiable sinusoidal basis functions, and the minimal parameterization for compatible position, velocity, and acceleration generation, which is guaranteed by the continuous function approximator when regarding the last layer of the network as a linear combination to the output.

**Conditional Generation:** To embed the proposed differentiable representation into the trajectory generation module, we need to specify the condition \( c \). In driving scenarios, the factors we consider for trajectory planning include the ego-vehicle dynamics, the surrounding obstacles dynamics, as well as the stationery environmental obstacles. For the first factor, we include the current velocity \( v_{t_0} \), as one of the condition. For the second and third factors, since the potential map \( C \) only reflects a cut of the surrounding obstacles, we encode a window of multiple previous potential maps \( C_{t_0-K:t_0} \) via a CNN and a RNN, denoted as \( r \). Resultantly, we have a condition at \( t_0 \) as

\[
c_{t_0} = [ v_{t_0} \ r(C_{t_0-K:t_0}) ]^T \tag{10}
\]

which becomes a mapping from \( t \) to the frequency and phase of the sinusoidal basis function. Then we can derive the trajectory from \( t_0 \) to \( t_0+T \) by sweeping \( t \) from \( t_0 \) to \( t_0+T \).

**C. Loss Function Design and Training**

Based on the loss terms design introduced above, we can simply set the loss function for training the network as a sum of (2), (3) and (8):

\[
\mathcal{L} = \mathcal{L}_{intadv}(\mathcal{G}, \mathcal{D}) + \mathcal{L}_{int}(\mathcal{G}) + \mathcal{L}_T(y) \tag{11}
\]

**Data Relabeling:** When predicting for the future trajectory, there can be non-causal labeling for the current data. For example, a vehicle stops in a queue, waits for the traffic light turning green. Say, there are 3 s left for the light turning.

Then the retrospective trajectory label with a time horizon 5 s for the current time is: stop for 0–3 s and accelerate for 3 s–5 s. Therefore, the supervision becomes a trajectory crossing the preceding vehicle. However, the driver at the current time actually cannot know that the light turns in 3 s, hence such supervision is reasonable unless the driver was informed with an external aid. To relieve this labeling problem, we re-label the supervision for such crossing cases with causality by modifying the trajectory with deceleration at a constant acceleration before the collision happens. We consider it is reasonable behavior for human driving without referring to the future.

**Training Details:** We use a 4-layer convolutional neural network with batch normalization and leaky ReLU activation function in the CNN. A 3-layer Gated recurrent units (GRU) with 0.2 dropout in the RNN and 5-layer fully connected deep network for our continuous trajectory model. We use a batch size of 32 and Adam optimizer with an initial learning rate of 0.0003. The networks are implemented in PyTorch and trained on AMD 3900X CPU and Nvidia RTX 2060 Super GPU until the model converges.

**D. Closed-loop System Implementation**

Different from the works using only a point position for tracking, which may not perform well due to the limited onboard computing frequency, we implement an asynchronous driving architecture by separating the long-horizon trajectory planning and trajectory tracking in two threads, achieving a receding horizon controller [25].

**Controller design:** Specifically, the planning thread generates continuous trajectories with time horizon \( T = 3 s \) in a fixed frequency or triggered, and annotates the trajectory with vehicle’s pose and the starting time \( t_{start} \). The control thread is running in another fixed higher frequency. It gets vehicle’s current pose at time \( t_{now} \) and queries the trajectory with \( t = t_{now} - t_{start} \) as the reference point. Then we use a feedback controller for throttle and brake control, and a nonlinear rear wheel feedback controller [26] for steering control:

\[
\begin{align*}
\alpha_c(t) &= k_{qe}d(t) + k_v \varepsilon_c(t) + a_c(t) \\
\delta_c(t) &= \tan^{-1} \left( \frac{\omega_c(t)L}{v(t)} \right)
\end{align*}
\tag{12}
\]
where \( L \) denotes the wheelbase, and at time \( t \), \( a_c \) represents throttle or brake (for \( a_c > 0 \) throttle equals to \( a_c \) and brake is 0, for \( a_c < 0 \) brake equals to \(-a_c \) and throttle is 0), \( \delta_t \) is the steering angle, \( \omega_c \) is the angular velocity, \( e_v \) is the tracking error between the current velocity \( v \) and the reference velocity \( v_r \), \( e_d \) is the longitudinal error as shown in Fig.3. \( k \) are controller gains for proper error feedbacks. Note that quantities with subscript \( r \) are obtained from trajectory with time \( t \). The angular velocity is calculated by:

\[
\omega_c(t) = \frac{v_r(t)\kappa_r(t)\cos\theta_c(t)}{1 - \kappa_r(t)e(t)} - (k_0|v_r(t)|)\theta_c(t) - (k_c v_r(t) \frac{\sin\theta_c(t)}{\theta_c(t)})e(t)
\]

where \( e \) and \( \theta_e \) are the lateral error and heading error as shown in Fig. 2. \( \kappa_r \) is the curvature of the reference trajectory. \( k \) are the controller gains for proper error feedbacks.

IV. EXPERIMENTS

A. Open-loop Experiments on Datasets

We first validate the effectiveness of our proposed method on two publicly available and widely recognized datasets: KITTI Raw Data [27] and Oxford Radar RobotCar [28].

Datasets: KITTI Raw Data (KITTI) contains 6 hours of traffic scenarios at 10 Hz using a variety of sensor modalities including high-resolution stereo cameras, a Velodyne 3D laser scanner, and a high-precision GPS/IMU inertial navigation system (INS). Oxford Radar RobotCar (RobotCar) is a radar extension to The Oxford RobotCar Dataset, providing data from Dual Velodyne HDL-32E LIDARs and one Bumblebee XB3 trinocular stereo camera with optimized ground truth visual odometry for 280 km of driving around Oxford University.

Implementation Details: Since there is no routing planning \( R \), intention image \( I \) and potential map \( C \) in these datasets directly, we first build them up before training. For KITTI, we extract a time-independent 30m path from INS which is then projected into image coordinate by perspective mapping to generate intention map \( I \). Potential map \( C \) is acquired by fusing point cloud and reprojected generated intention image \( I \) by driving intention module using inverse perspective mapping. Note that the point cloud data is first transformed into INS coordinate, ahead of fusion, with calibration information between Velodyne and INS. As for Oxford, the data preparation process is similar to KITTI, with the exception that we use visual odometry (VO) data as ground truth path due to poor GPS signals Oxford provides. Additionally, we undistort image data collected by the stereo camera according to [28]. For both datasets, we split them as 70% are used for training with 10% and 20% for evaluation and testing.

Evaluation Metrics: We use 5 evaluation metrics to assess the effectiveness of various models, all of which are computed over the testing part of datasets.

- Average Displacement Error (ADE, \( E_{ad} \)) [5]: Average L2 distance between the ground truth and generated paths over all positions. Note that the corresponding positions are found w.r.t time rather than the nearest neighbour.
- Final Displacement Error (FDE, \( E_{fd} \)) [5]: L2 distance between the last ground truth and generated positions.
- Average Longitudinal Error \( E_x \), Average Lateral Error \( E_y \), and Average Velocity Error \( E_v \) [5] represent mean longitudinal, lateral displacement and velocity scalar error between the ground truth and the corresponding quantities estimated from the generated trajectory respectively. Among them, \( E_x \) can measure the model’s ability to model system dynamics.

Ablation Study: We exhibit the principle of utilizing our method by defining five ablated models from the original model as below:

- w/o intention: we skip the intention module and use raw RGB images as input instead of potential maps;
- w/o v0: we remove the initial velocity input to Neural Trajectory;
- w/o cos: replace sinusoidal function to ReLU;
- w/o HOS: the model has no high-order supervision;
- big HOS: big weights of high-order supervision in the loss function.

We refer to Table I for checking the influence of different models on error metrics. When some stimuli are missing, performance drops significantly, which validates the importance of intention. The performance degenerates most is the model w/o v0, because the initial velocity \( v_0 \) reflects the current dynamics, without which the trajectory does not have boundary value condition. Results on w/o HOD and big HOD indicate that the trade-off between different order supervisions is indispensable. On the one hand, high-order supervision rectifies dynamic outputs of Neural Trajectory. w/o cos verifies a better approximation accuracy when basis function is selected more appropriate for the dynamic system.

Comparative Study: For comparison, we investigate three recent baseline methods, and these models are introduced as follows:

- NI+DT and NI+DT+RNN: No intention input and discrete trajectory output. We follow the method in [11]. In NI+DT, the CNN module intends to extract features from sequential RGB images inputs, and the concatenated features are fed into fully connected layers to generate a discrete trajectory. In NI+DT+RNN, the network runs analogical to NI+DT except that the extracted features are fused by Long Short-Term Memory (LSTM) instead of simply concatenating them.
- DI+DT: Discrete intention input and discrete trajectory output. We follow the idea of [8] to give three discrete
Experiments Setup: Since we need to obtain highly interpretable and smooth trajectories as expert-provided data to imitate, we use human driving data as our training data rather than built-in AI driving data. We use Logitech G29 driving commands as discrete intentions to switch three different networks. Instead of generating control commands, we change it to output a series of discrete trajectory points and velocities.

- CI+DT and CI+DT+RNN: Continuous intention input and discrete trajectory output. The same structure to NI+DT and NI+DT+RNN, but we replace sequential RGB images inputs to sequential potential maps as continuous intention.
- CI+CT (Ours): Continuous intention input and continuous trajectory output method that we proposed.

From the results shown in Table III we can clearly see that our network performs the best on all metrics, which means the trajectories our method generates are more smooth and accurate in accordance with displacement-level metrics such as $E_{ad}$ and high-order metrics like $E_{ord}$.

Table III shows that DI+DT outperforms NI+DT and NI+DT+RNN, which indicates that the superiority of introducing intention. The intention helps to guide the direction of trajectory generation, resulting in the improvement of zero-order metrics. Additionally, model performance is enhanced by introducing continuous intention rather than discrete commands. Most importantly, our model significantly decreases the high-order metrics on both KITTI and RobotCar. Thanks to the continuous representation, we manage to precisely model the smoothness and the intrinsic constraints between derivatives.

Fig 4 displays typical qualitative results on these models. For various kinds of scenarios, our model generates the most human-like trajectory. Note figures in Fig 4(c) show no ground truth as the vehicle stops actually because of the front obstacle, and the performance of models is enhanced contributed from the use of semantic potential maps.

B. Closed-loop Experiments in Simulation

As a driving task, only open-loop validation is not sufficient, since the error is accumulated. To further validate the efficacy of our proposed method, we evaluate our method with closed-loop experiments both in the CARLA simulation and a real-world vehicle platform.  

Experiments Setup: Since we need to obtain highly interpretable and smooth trajectories as expert-provided data to imitate, we use human driving data as our training data rather than built-in AI driving data. We use Logitech G29 driving force-racing wheel as our data acquisition equipment, and collect about 2 hours of human driving data in CARLA simulation with a speed limit of 30 km/h. We collect data in Town01 with 4 different weather as training condition and test our method in Town02 with other 2 different weather as testing condition.

We compare our method performance with several previously proposed approaches [6], [8], [9], [13], [14], [29]–[31] on the original CARLA benchmark [32] and the NoCrash benchmark [14]. The original CARLA benchmark allows us to compare algorithms on sets of strictly defined goal-directed navigation tasks. It provides 4 simple navigation tasks, the last of which has a few dynamic obstacles. The NoCrash benchmark is much more challenging. The vehicle drives in 3 different traffic conditions: empty town where no dynamic objects exist, regular traffic which has moderate number of cars and pedestrians, and dense traffic with large number of pedestrians and heavy traffic condition. Almost all current methods have a low success rate under the most difficult dense condition under testing condition.

Comparative Study: The comparison results on CARLA benchmark are shown in Table III We achieve a 100% success rate on all tasks under training conditions and 3 tasks under testing conditions, outperforming all other methods. On the last task under testing condition, our success rate is slightly lower than that of LaTeS. Overall, our method achieves competitive performance with state-of-the-art methods.

We test our method 3 times on NoCrash benchmark and the results are shown in Table IV. In empty condition task, our method achieves a 100% success rate on all tasks under training conditions and 3 tasks under testing conditions, outperforming all other methods. In regular traffic task, our method is relatively 35% higher than the second place in testing condition and equals to LaTeS in training condition. In dense traffic task, our method is relatively 35% and 91% higher than the second place in training and testing. Several
### TABLE III
RESULTS ON CARLA BENCHMARK

| Task          | Condition                  | MT [9] | CIL [8] | CIRL [13] | CAL [29] | CILRS [14] | LSD [30] | LaTeS [6] | Ours |
|---------------|----------------------------|--------|---------|-----------|----------|------------|----------|----------|------|
| Straight      | Train                      | 96     | 98      | 98        | 100      | 96         | -        | -        | 100  |
| One turn      | New Town & New Weather     | 87     | 89      | 97        | 97       | 92         | -        | -        | 100  |
| Navigation    |                            | 81     | 86      | 93        | 92       | 95         | -        | 100      | 100  |
| Nav. dynamic  |                            | 81     | 83      | 82        | 83       | 92         | -        | 100      | 100  |

### TABLE IV
RESULTS ON NOCRASH BENCHMARK

| Method        | Training Condition | New Town & New Weather |
|---------------|--------------------|------------------------|
|              | Empty              | Regular                | Dense                  |
| CIL [8]      | 79 ± 1             | 60 ± 1                 | 21 ± 2                 |
| CAL [29]     | 81 ± 1             | 73 ± 2                 | 42 ± 3                 |
| MT [9]       | 84 ± 1             | 54 ± 3                 | 13 ± 4                 |
| CILRS [14]   | 97 ± 2             | 83 ± 0                 | 42 ± 2                 |
| LSD+ [30]    | 90 ± 2             | 56 ± 2                 | 24 ± 8                 |
| LaTeS [6]    | 100 ± 0            | 94 ± 2                 | 54 ± 3                 |
| DA-RB [31]   | 100 ± 0            | 94 ± 2                 | 89 ± 3                 |
| Ours         | 100 ± 0            | 94 ± 2                 | 89 ± 3                 |

---

common and representative failure cases in all methods are shown in Fig. 5. In our test, there are very few cases of (a), and (b), which indicates the superiority of introducing continuous intention and long-term planning, and there are also few cases of (c), which indicates the superiority of introducing continuous trajectory with high-order derivatives. Almost all failure cases are collisions with side-on vehicles and pedestrians at intersections in our test, as shown in Fig. 5 (d-f). All methods encounter this problem. One reason is that agents in CARLA simulation are not strong enough to avoid obstacles at intersections. And we suspect that there is less data for this scenario in our training data and thus our model is less concerned with dynamic obstacles that are not in the driving intention area.

**Generalization to other vehicle:** Similar to [5], we also transfer our method from cars to motorcycles to validate the generalization ability for different vehicles of our method, because motorcycles have smaller turning radius, greater acceleration, and its camera view tilts and shifts when turning and braking. We test in the regular task and training condition on NoCrash benchmark, and we even do not change anything including controller parameters. The success rate is 91.7%, which is only 3% below the original 94% and validates the strong generalization ability of our method.

**Robustness against computation latency:** We also evaluate

---

| Success Rate / % | Asynchronous | Synchronous |
|------------------|--------------|-------------|
| Delay Time / ms  |              |             |
| 0                | 100          | 100         |
| 200              | 80           | 90          |
| 400              | 60           | 90          |

**Fig. 5.** Representative failure cases among all methods. (a) Hit a static obstacle; (b) Crash into wrong side of road; (c) Hit a car from rear side; (d-f) Hit moving agents at side. Most of our failure cases are (d-f).

**Fig. 6.** Success rate under different time delays.

**Fig. 7.** Real-world vehicle platform and the actual travel route in the map. To further validate the feasibility and the superiority of the modular design of our proposed method, we only fine-tune the driving intention module on real-world data and directly transfer the trajectory generation module from the RobotCar dataset to a real-world vehicle with Ackermann steering on our campus, shown in Fig. 7. The vehicle is equipped with a Velodyne VLP-16 LiDAR, a MYNT EYE D-1000-120 camera, an Xsens Mti-300 IMU, a Qianxun D300-GNSS GPS module, and an industrial PC with Intel i7-7700 CPU which collects all sensors’ data, runs our method, and sends control signals to the vehicle by CAN bus. We build a route map of campus and plan a global route to guide the vehicle. Most cases have good results shown in Fig. 8 and we test our method 3 times on the same route in a campus environment which is about 1 km long, and 5 human
interruptions occurs, which further validates the feasibility and the superiority of our method. For more details, please refer to our video.

V. CONCLUSION

In this paper, we propose a hierarchical driving model explicitly designed with intention and dynamics modules, which decouples the complexity in the observation-to-action reasoning from the human driving data. We propose a continuous trajectory function approximator which naturally reserves the derivatives for high-order supervision. Furthermore, we validate our method on both datasets and simulators. Finally, practical implementation is proposed to tolerate the network latency, which enables generalization testing in the real world.

REFERENCES

[1] S. Kuuti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, “A survey of deep learning applications to autonomous vehicle control,” IEEE Transactions on Intelligent Transportation Systems, 2020.

[2] W. Ding, L. Zhang, J. Chen, and S. Shen, “Safe trajectory generation for complex urban environments using spatio-temporal semantic corridors,” IEEE Robotics and Automation Letters, vol. 4, no. 3, pp. 2997–3004, 2019.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.

[4] B. Zhou, P. Krähenbühl, and V. Koltun, “Does computer vision matter for action?,” arXiv preprint arXiv:1905.12887, 2019.

[5] P. Cai, Y. Sun, H. Wang, and M. Liu, “Vignet: A vision-based trajectory generation network for autonomous vehicles in urban environments,” arXiv preprint arXiv:2004.12591, 2020.

[6] A. Zhao, T. He, Y. Liang, H. Huang, G. V. d. Broeck, and S. Soatto, “LateS: Latent space distillation for teacher-student driving policy learning,” arXiv preprint arXiv:1912.02973, 2019.

[7] H. Ma, Y. Wang, R. Xiong, S. Kodagoda, and L. Tang, “Deepgoal: Learning to drive with driving intention from human control demonstration,” Robotics and Autonomous Systems, p. 103477, 2020.

[8] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, “End-to-end driving via conditional imitation learning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 1–9, IEEE, 2018.

[9] Z. Li, T. Motoyooshi, K. Sasaki, T. Ogata, and S. Sugano, “Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability,” arXiv preprint arXiv:1809.11100, 2018.

[10] C. Doersch and A. Zisserman, “Multi-task self-supervised visual learning,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 2051–2060, 2017.

[11] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2174–2182, 2017.

[12] T. Wu, A. Luo, R. Huang, H. Cheng, and Y. Zhao, “End-to-end driving model for steering control of autonomous vehicles with future spatiotemporal features,” in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 950–955, IEEE, 2019.

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

[14] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, “Exploring the limitations of behavior cloning for autonomous driving,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 9329–9338, 2019.

[15] D. Choi, S.-j. Han, K. Min, and J. Choi, “Pathgan: Local path planning with generative adversarial networks,” arXiv preprint arXiv:2007.03877, 2020.

[16] A. Amini, G. Rosman, S. Karaman, and D. Rus, “Variational end-to-end navigation and localization,” in 2019 International Conference on Robotics and Automation (ICRA), pp. 8958–8964, IEEE, 2019.

[17] J. Wang, Y. Wang, D. Zhang, Y. Yang, and R. Xiong, “Learning hierarchical behavior and motion planning for autonomous driving,” arXiv preprint arXiv:2005.03867, 2020.

[18] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, “Trafficpredict: Trajectory prediction for heterogeneous traffic-agents,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 6120–6127, 2019.

[19] R. Chandra, T. Guan, S. Panuganti, T. Mittal, U. Bhattacharya, A. Bera, and D. Manocha, “Forecasting trajectory and behavior of road-agents using spectral clustering in graph-lstms,” IEEE Robotics and Automation Letters, vol. 5, no. 3, pp. 4882–4890, 2020.

[20] X. Huang, S. G. McGill, J. A. DeCastro, B. C. Williams, L. Fletcher, J. J. Leonard, and G. Rosman, “Diversity-aware vehicle motion prediction via latent semantic sampling,” arXiv preprint arXiv:1911.12736, 2019.

[21] N. Deo and M. M. Trivedi, “Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms,” in 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 1179–1184, IEEE, 2018.

[22] X. Huang, S. G. McGill, B. C. Williams, L. Fletcher, and G. Rosman, “Uncertainty-aware driver trajectory prediction at urban intersections,” in 2019 International Conference on Robotics and Automation (ICRA), pp. 9718–9724, IEEE, 2019.

[23] P. Cai, S. Wang, Y. Sun, and M. Liu, “Probabilistic end-to-end vehicle navigation in complex dynamic environments with multimodal sensor fusion,” IEEE Robotics and Automation Letters, vol. 5, no. 3, pp. 4218–4224, 2020.

[24] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, “Implicit neural representations with periodic activation functions,” arXiv preprint arXiv:2006.09661, 2020.

[25] A. C. S. Patterson and V. Kumar, “Safe receding horizon control for aggressive mav flight with limited range sensing,” in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3235–3240, IEEE, 2015.

[26] 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, vol. 1, no. 1, pp. 33–55, 2016.

[27] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” The International Journal of Robotics Research, vol. 32, no. 11, pp. 1231–1237, 2013.

[28] D. Barnes, M. Gadd, P. Murcutt, P. Newman, and I. Posner, “The oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset,” in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3235–3240, IEEE, 2015.

[29] A. Sauer, N. Savinov, and A. Geiger, “Conditional affordance learning for driving in urban environments,” arXiv preprint arXiv:1806.06498, 2018.

[30] E. Ohn-Bar, A. Prakash, A. Behl, K. Chitta, and A. Geiger, “Learning situational driving,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11296–11305, 2020.

[31] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger, “Exploring data aggregation in policy learning for vision-based urban autonomous driving,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11763–11773, 2020.

[32] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” arXiv preprint arXiv:1711.03938, 2017.