Integrating Algorithmic Sampling-Based Motion Planning with Learning in Autonomous Driving

YIFAN ZHANG, JINGHUAI ZHANG, JINDI ZHANG, and JIANPING WANG,
City University of Hong Kong, Hong Kong SAR and City University of Hong KongShenzhen Research Institute, China
KEJIE LU, University of Puerto Rico at Mayagüez, Puerto Rico
JEFF HONG, Fudan University, China

Sampling-based motion planning (SBMP) is a major algorithmic trajectory planning approach in autonomous driving given its high efficiency and outstanding performance in practice. However, driving safety still calls for further refinement of SBMP. In this article we organically integrate algorithmic motion planning with learning models to improve SBMP in highway traffic scenarios from the following two perspectives. First, given the number of points to be sampled, we develop a new model to sample "important" points for SBMP by predicting the intention of surrounding vehicles and learning the distribution of human drivers' trajectory. Second, we empirically study the relationship between the number of sample points and the environment, which is largely ignored in conventional SBMP. Then, we provide a guideline to select the appropriate number of points to be sampled under different scenarios to guarantee efficiency. The simulation experiments are conducted based on the vehicle trajectory dataset NGSIM. The results show that the proposed sampling strategy outperforms existing sampling strategies in terms of the computing time, traveling time, and smoothness of the trajectory.

CCS Concepts: • **Computing methodologies → Robotic planning;**

Additional Key Words and Phrases: Autonomous driving, sampling-based motion planning, vehicle intention prediction, imitation learning

ACM Reference format:
Yifan Zhang, Jinghuai Zhang, Jindi Zhang, Jianping Wang, Kejie Lu, and Jeff Hong. 2022. Integrating Algorithmic Sampling-Based Motion Planning with Learning in Autonomous Driving. *ACM Trans. Intell. Syst. Technol.* 13, 3, Article 39 (January 2022), 27 pages.
https://doi.org/10.1145/3469086

This work was partially supported by Hong Kong Research Grant Council under GRF project 11200220, Science and Technology Innovation Committee Foundation of Shenzhen under Grant No. JCYJ20200109143223052, and NSF under Grant CNS-1730325.

Authors' addresses: Y. Zhang, J. Zhang, J. Zhang, and J. Wang (corresponding author), City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong and City University of Hong Kong Shenzhen Research Institute, 8 Yuexing 1st Road, Nanshan District, Shenzhen, 518057, China; emails: {yif.zhang, jzhang538-c, jd.zhang}@my.cityu.edu.hk, jianwang@cityu.edu.hk; K. Lu, University of Puerto Rico at Mayagüez, 259 Avenida Alfonso Valdés Cobián, Mayagüez, 00682, Puerto Rico; email: kejie.lu@upr.edu; J. Hong, Fudan University, 220 Handan Road, Yangpu District, Shanghai, 200437, China; email: hong_liu@fudan.edu.cn.

2157-6904/2022/01-ART39 $15.00
https://doi.org/10.1145/3469086

1 INTRODUCTION

In the past decade, both academia and industry put a large amount of effort into autonomous driving. There have been autonomous vehicles put into commercial practice, for example, the taxi service provided by Google's driverless car, Waymo [54]. Meanwhile, an increasing amount of autonomous driving accidents have emerged, which should remind us that big challenges still remain — such as seeing surroundings robustly, perceiving objects in real time, and acting safely — to achieve full autonomy.

Motion planning plays a key role in acting safely, which means generating a collision-free trajectory from the current location to the immediate destination. In the literature, there are four main categories of motion planning algorithms: graph search-based planners [4, 26, 43], sampling-based planners [27, 33], interpolating curve planners [17, 34], and numerical optimization approaches [57]. Among them, sampling-based motion planning (SBMP) has recently become a mainstream motion planner [3, 38, 40] given its capability of potentially finding global optimal paths and solving high-dimensional motion planning problems in a shorter time using advanced algorithms [25]. Moreover, SBMP does not need precise obstacle geometry required in many other methods [46].

In order to consistently capture the dynamically changing environment, sensors equipped on the autonomous vehicle usually are synchronized at a fixed updating frequency, for example, 10 Hz [32]. This requires the motion planner to finish its decision in 100 ms in order to be responsive to the surrounding environment changes. However, SBMP runs longer than 100 ms in complex environments according to the authors of [40]. To shorten this gap, on the one hand, SBMP needs to find a collision-free trajectory with fewer sample points. Thus, identifying necessary and sufficient sample points becomes the key to reducing computation time. On the other hand, predicting the future situation around the ego vehicle in a short time window accurately shall gain more time for SBMP to make the decision as the planned trajectory can be valid for a longer time period.

In current works [33, 40], the number of sample points is fixed and predefined for all surrounding environments. However, when the surrounding environment is simple, with few adjacent vehicles, sampling more points than necessary will prolong the computation time. Thus, the number of points to be sampled should be determined by the environment scenario to further reduce the computation time.

In this article, we design a novel learning-based SBMP framework to solve the aforementioned issues in highway traffic scenarios. We first predict the intention of surrounding vehicles to allow motion planning to "see" ahead. Prediction information enables a motion planner to generate the trajectory with (1) a longer valid time period and (2) smooth velocity over a longer time period. Then, we design a new sampling model by integrating prediction and learning from the human drivers' trajectory, which contributes to generating a smooth and collision-free trajectory in a shorter time. Given this well-trained sampling model and environment scenarios, we design a strategy to dynamically determine the number of sample points. Our main contributions are summarized as follows:

- We propose a new automatic labeling strategy for data preprocessing to correctly label and extract useful driving scenarios, and design the first 2-stage prediction model that greatly improves the accuracy of prediction on vehicles' intentions.
- We design a new bias sampling model by integrating the prediction of surrounding vehicles and imitation learning. Specifically, we train a CVAE model to generate collision-free sample points only near the human-driving trajectory through learning from the trajectory dataset. Given the number of points to be sampled, the proposed sampling model will lead to faster

motion planning and help generate a smooth collision-free trajectory that is on par with and even better than that generated by human drivers.
- We propose a new approach to adaptively determine the number of sample points for motion planning. We empirically study the relationship between the number of sample points and the environment scenario. The selected number of points to be sampled can further reduce the computation time while guaranteeing a feasible and safe solution.
- We design an online planning strategy to mitigate the time lag problem caused by the implementation aspects. We estimate the shifted initial state and use its estimation as the initial state in both the sampling model and the planning algorithm to calibrate the time lag issue. Our proposed method can not only perform independently but can also cooperate with the existing mitigation schemes.
- We evaluate our sampling model from the perspectives of prediction accuracy, success rate of finding a collision-free trajectory, computation time of planning, and quality of trajectory by comparing it with other state-of-the-art sampling models. We also validate the performance improvement brought by the prediction and imitation learning modules individually.

Compared with our conference paper [56], the new contributions in this extension include (1) developing an adaptive approach to determine the number of sample points for a given environment scenario so that the proposed solution can be practically adopted in autonomous driving; (2) designing a new online planning strategy to mitigate the time lag problem; and (3) comparing the performance of the proposed model with more baseline models and validating the performance improvement brought by the prediction and imitation learning modules individually.

The rest of the article is organized as follows. In Section 2, we review the general process of the SBMP algorithm and briefly introduce our framework. We present our 2-stage intention prediction model in Section 3 and evaluate its performance. In Section 4, we leverage prediction and imitation learning to design a new sampling model. In Section 5, we introduce our empirical study on determining the number of sample points. In Section 6, we evaluate our sampling strategy. We introduce the related works of motion planning in Section 7. In Section 8, we present our conclusions.

2 THE BACKGROUND AND NEW FRAMEWORK

In this section, we explain the basic modules of SBMP and then present the proposed learning framework for it.

In this article, we consider that a vehicle's state at any given time can be described by $[p_x, p_y, v, \theta]$, where p_x and p_y are the coordinates of the center of the vehicle's rear axle, v is the instantaneous velocity, and θ is the heading angle of the vehicle with respect to the road direction (y-axis). These four features represent the state and dynamic of a vehicle well. Here we note that, although we consider a 4-dimensional state space, our approach can be adopted to other scenarios that define states in higher-dimensional spaces. SBMP aims to find the best trajectory that connects a vehicle's initial state, x_{init}, to any possible goal state, x_{goal}, in a goal region, $\mathcal{X}_{goal}$. To this end, there are many intermediate states being sampled, x_s, which are on feasible trajectories. Each of such intermediate states is defined as a sample point. A state space C consists of x_{init}, $\mathcal{X}_{goal}$, and all sample points. We further define $C_{free} \subseteq C$ as the collision-free state space. SBMP generally consists of a sampling phase and planning phase introduced as follows.

- ***Sampling phase*:** This procedure discretizes the state space to generate the sample points. Although the uniform sampling strategy is commonly used, many bias sampling strategies have been proposed, such as goal biasing [35], bias Gaussian sampling [33], and informed

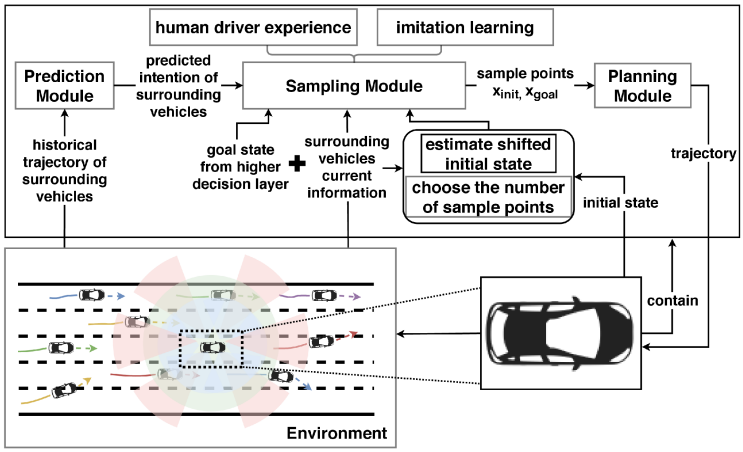

Fig. 1. Framework of proposed motion planning.

sampling [16]. The key advantage of bias sampling is that, by eliminating unnecessary sample points, a bias sampling scheme may reduce the search space and speed up the planning algorithm.

- ***Planning phase*:** Given the sample points, obstacle information, and differential constraints, this procedure generates a collision-free roadmap [28] or tree [25] connecting x_{init} and x_{goal}. To make sure that the generated trajectory between two points satisfies the vehicle kinematic constraints, we apply the most general car-like kinematic model to represent its differential constraints. If a tree is constructed in this phase, the algorithm stops because the final trajectory can be obtained directly. If a roadmap is constructed where there are several feasible trajectories, graph search algorithms — such as Dijkstra [11], A* [19], and D* [51] — can be applied to find the optimal one connecting x_{init} and x_{goal}.

The sampling strategy in the sampling phase is the key to finding a trajectory quickly and effectively. Therefore, in this article, we mainly focus on designing a better sampling strategy that can reduce the computation time and improve the trajectory smoothness in the planning phase. As shown in Figure 1, our proposed SBMP includes three modules: a prediction module, imitation learning–based sampling module, and planning module. We make contributions to the first two modules.

2.1 Framework

As mentioned, our proposed framework consists of three modules: a prediction module, sampling module, and planning module. The first two modules are based on various learning approaches and provide more accurate and useful inputs to the third module. We give a brief introduction of these three modules as follows.

- *Prediction module.* In this module, we predict surrounding vehicles' intentions based on their history trajectories with high accuracy by improved the labeling method and training model.
- *Sampling Module.* In this module, given the environment scenario, the number of sample points is determined as an input to the sampling model. The sampling model generates sample points near the human-driving trajectory given the current and predicted environment information, the initial state, and the goal state.

- *Planning Module*. This module takes the set of sampling points, the initial state, and the goal state to generate a collision-free and smooth trajectory. Planning algorithms in [25, 27] can be applied in this module.

3 PREDICTION OF VEHICLE'S INTENTION

In this section, we introduce the prediction module to allow the planning algorithm to see ahead and prolong the valid period of the trajectory. We first briefly present the architecture of the prediction module in Section 3.1. Next, in Section 3.2, we analyze existing automatic labeling schemes and propose a more advanced scheme. In Section 3.3, we present details of our prediction module, involving network structure, training strategy, and the post-processing algorithm to predict accurate intentions of surrounding vehicles. Finally, in Section 3.4, we conduct numerical experiments to validate our model.

3.1 Overview of the Prediction Module

Many learning models have been proposed for intention prediction in the past few years [39, 52]. In general, existing learning models take a vehicle's neighbor information and its past features as inputs, including position, velocity, acceleration, and so on. These models usually output the probabilities of different future intentions, including car-following, lane-changing-left, and lane-changing-right [52]. However, in our framework, we aim to predict the intention of a vehicle merely based on its history trajectory since it is not always available to acquire the neighboring information (such as obstacle positions) of adjacent vehicles in real driving scenarios.

In addition, despite the promising results of utilizing learning-based prediction models, we note that the accuracy of lane-changing is usually below 90% in recent work [52] (and with different traditional models in Table 1), which could compromise motion planning in autonomous driving. This is because unexpected lane-changing behavior may affect the safety of neighboring vehicles and may interfere with the planned path of other vehicles. To improve accuracy, we refine the prediction module as follows:

- We improve the existing automatic labeling scheme to precisely identify the start point as well as the end point of lane-changing, assign the correct class label to each frame, and extract meaningful driving patterns, which will be utilized to develop the intention prediction algorithm.
- To improve the accuracy of intention prediction, we first propose to predict vehicles' intentions with different stage models based on our analysis on trajectory datasets. Specifically, we classify the history trajectory of a vehicle into different stages (e.g., before, during, and after changing lanes) and use one learning model to predict the intention in each stage. To train each stage model, we further separate the labeled data into different sets, each containing different patterns.

In this article, we propose a novel prediction module comprising two stages: (1) regular and (2) after-crossing-line (vehicle crosses the line in previous 2 seconds). As shown in Figure 2, three scenarios will be included in the Stage 1 Model: (1) typical car-following, (2) car-following before lane changing (BLC), and (3) lane changing pattern I (i.e., before crossing the line between two lanes). For the second stage, we consider the frames in a certain period after the vehicle crosses the line. In this stage, the model will consider four possible scenarios: (1) typical car-following, (2) car-following after lane changing (ALC), (3) lane changing pattern II (i.e., immediately after crossing the line), and (4) driving on line. In the next subsection, we will discuss how to automatically assign labels to vehicles with different time epochs in a dataset.

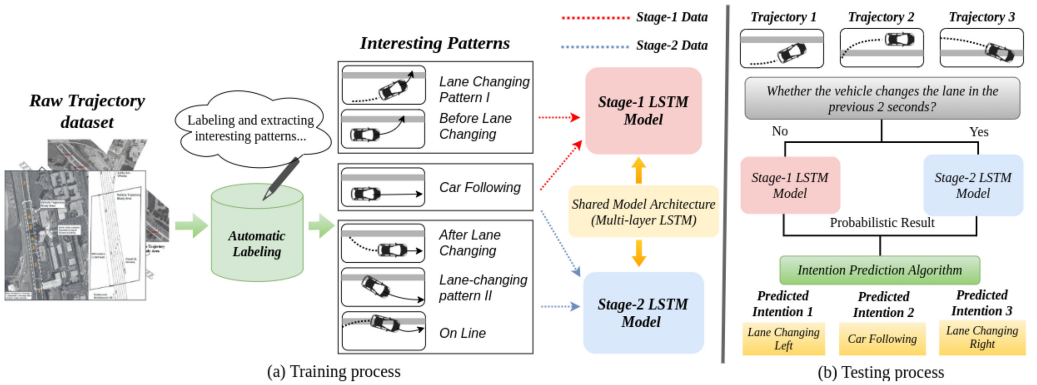

Fig. 2. Overview of (a) the training process and (b) the testing process of our prediction model. The model architecture is shared by both single-stage models, which is illustrated in detail in Figure 8. Each single-stage model is trained individually with a different training set of interesting driving patterns.

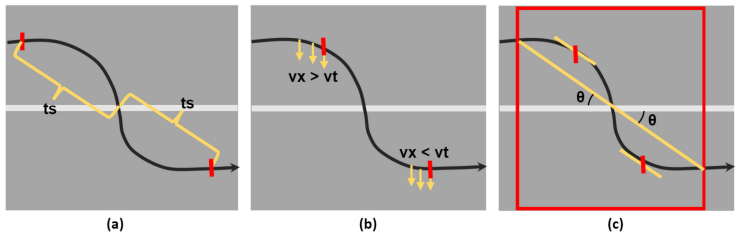

Fig. 3. Illustration of three types of automatic labeling schemes: (a), (b), and (c) illustrate labeling schemes in [10, 44, 49, 52], respectively. Positions of vehicle between two red vertical line segments are labeled as lane-changing.

3.2 Automatic Labeling

Given a large trajectory dataset, which contains detailed trajectory information of each vehicle over a long period, it is crucial to apply an automatic labeling scheme to classify the driving behaviors of vehicles at each time epoch and extract useful patterns for further learning. Currently, there are three major automatic labeling schemes in the field [10, 44, 49, 52], as illustrated in Figure 3. Nevertheless, they are not able to cover all of the cases in which some time points are wrongly labeled. We extract some cases from the NGSIM dataset to illustrate that the aforementioned schemes cannot handle them correctly, as shown in Figure 4. In addition, all existing automatic labeling schemes will skip the cases in which a vehicle is driving along the line during lane changing, as shown in Figure 4(f), which is a common situation.

The first type of scheme [10] identifies the time that a vehicle takes in crossing the line between two lanes and then specifies the behavior of the vehicle as lane-changing if and only if the vehicle is within t seconds before or after the crossing event, where t is a predefined threshold as shown in Figure 3(a). A major problem with this type of scheme is that, with a fixed t, some car-following cases may be wrongly labeled as lane-changing, as shown in Figure 4(a).

The second type of scheme [44, 49] assumes that a vehicle starts lane-changing if its lateral velocity towards an adjacent lane exceeds a threshold v_t in N consecutive timesteps before line crossing, and it ends lane-changing if it crosses the line or its lateral velocity is lower than v_t after line-crossing, as shown in Figure 3(b). Its main issues are that a vehicle may continue lane-

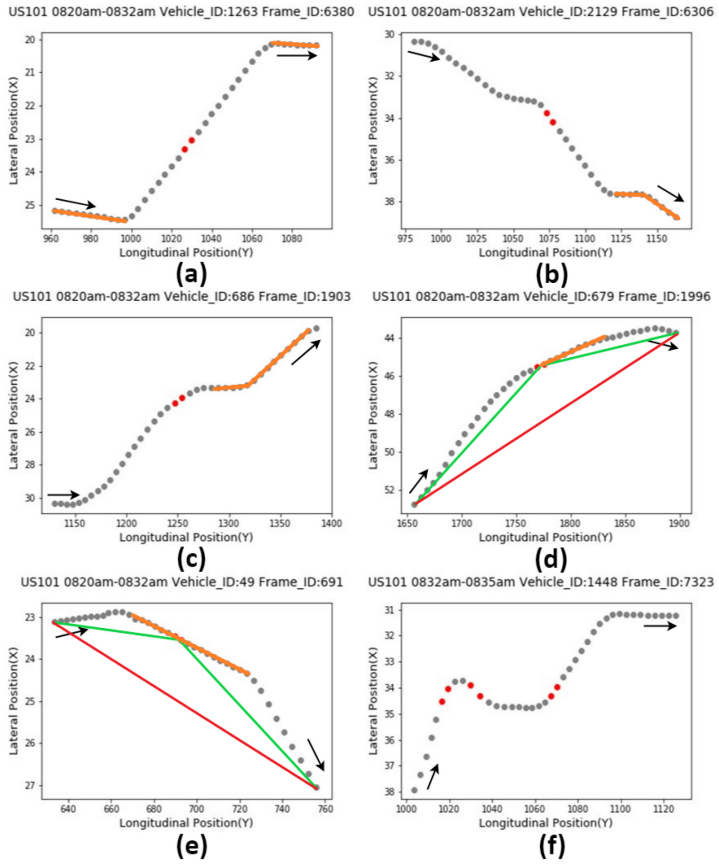

Fig. 4. Some cases in real trajectory data. Each scatter point indicates a position in the trajectory. Each red line indicates the tangent line from −2 s before lane-changing to +2 s after lane-changing, which is utilized to label critical points by previous work. Two green lines in each figure are the tangent lines found by our method. Each yellow line segment indicates positions with possible wrong labels in previous methods. Red dots indicate the positions where the vehicle is closest to the lane line.

changing after the cross-over or change its steering angle several times during lane-changing, as shown in Figure 4(b) and 4(c).

A recent study in [52] proposed a third type of scheme, which first identifies two positions of a vehicle, corresponding to 2 seconds before and after line-crossing, respectively. Next, the two points will be used to create a straight line whose angle with the line is θ, shown as a yellow line in Figure 3(c). Finally, it attempts to find two parallel tangent lines that have the nearest tangent points to the crossing point and will use these two points as the starting and end points for lane-changing. Although this scheme leads to better prediction performance, it also suffers from the inaccurate labeling issue when a vehicle changes its steering angle multiple times during lane-changing, as shown in Figure 4(b) and 4(c). Moreover, this scheme cannot find the desired tangent line in some cases, as shown in Figure 4(d) and 4(e).

To assign a correct label to each case and improve the performance of intention prediction, we propose a novel automatic labeling scheme as follows. Similar to the scheme in [52], we first identify two points A and B corresponding to t seconds before and after line-crossing, respectively.

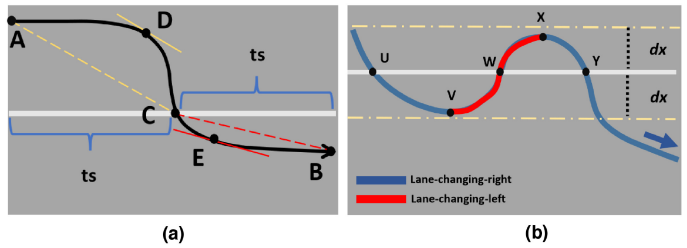

Fig. 5. Illustration of our proposed automatic labeling method on lane-changing cases. (a) Regular lane-changing. (b) Lane-changing with on-line driving.

Next, we link the two points to the line-crossing point C, as shown in Figure 5(a). Starting from the point A, we can find the first point D on the trajectory, whose tangent line is parallel to the line AC. For this point D, we regard it as the starting point of this lane-changing behavior. Similarly, from the other side, we can find another tangent point E closest to B, whose tangent line is parallel to the line BC. We regard this point as the end point of lane-changing behavior.

Based on the procedures presented earlier, our automatic labeling scheme provides two levels of labels. The first-level label specifies the category of future behavior for each vehicle at each position. In the second level, we provide an additional label to each case for training purposes. Specifically, in a regular lane-changing case, such as Figure 5(a), we give a car-following label to positions before D and after E, and a lane-changing-right label to positions between D and E as their first-level labels, which represent their future intentions in the short term. In the second level, we label positions between A and D as BLC, positions between D and C as lane-changing pattern I, between C and E as lane-changing pattern II, and between E and B as ALC. These second-level labels will be utilized when we assign the extracted driving patterns to train the corresponding stage models. For the on-line driving case, we adopt a slightly different strategy to derive the two levels of labels. We first specify the second-level label as on-line driving if (1) the vehicle crossed a line multiple times in a short period t_x and (2) the maximum distance to the line is smaller than a threshold d_x. For example, in Figure 5(b), positions between U and Y are labeled as on-line driving. Next, we identify some turning points and determine the first-level labels. For example, in Figure 5(b), positions before V and after X are labeled as lane-changing-right, and the positions between V and X are labeled as lane-changing-left.

3.3 2-Stage Model, Training Strategy, and Intention Prediction Algorithm

Based on the data labeled by the new labeling scheme, we design the prediction module, which consists of two learning models as shown in Figure 2. We choose Multi-layer Long Short-Term Memory (LSTM) architecture for both, each following the network design of baseline model II in Figure 8 with hidden dimension 128. The prediction module will take the lateral and longitude positions, velocity, acceleration, heading angle, and lane_ID of the past 10 frames as input for each of the adjacent vehicles and output their intentions.

To train the two models, we use 6 sequences of trajectory data in two datasets: NGSIM US-101 [14] and I-80 [13]. NGSIM (Next Generation Simulation) is a publicly available trajectory dataset containing real traffic data from four different places. Both NGSIM US-101 and I-80 contain 45 minutes of trajectory data (three sequences) from a highway with five lanes. Each sequence comprises trajectories of multiple vehicles from 15 minutes. We use data from the first 12 minutes for training and the last 3 minutes for testing, with all data labeled by our automatic labeling scheme. As shown in Figure 2, we train the Stage-1 model with three cases, and the Stage-2 model

ALGORITHM 1: Intention Prediction Algorithm

```
data ← EXTRACTATTRIBUTES(Local_X, Local_Y, Vehicle_Velocity, Vehicle_Acceleration, Lane_ID, Theta)
if vehicle changes lane in previous 2 s then
    tag ←ASSIGNTAG("Model_A")
else
    tag ← ASSIGNTAG("Model_B")
end if
pred_label, dx ←PREDICT(data)
if tag == "Model_B" then
    decision ← pred
else
    if dx > 0 then
        if pred == "Lane-changing-right" then
            decision ← pred
        else if dx < threshold then
            decision ← "On line"
        else
            decision ← pred
        end if
    else
        if pred == "Lane-changing-left" then
            decision ← pred
        else if dx < threshold then
            decision ← "On line"
        else
            decision ← pred
        end if
    end if
end if
return decision
```

with four cases. While extracting these training cases from datasets, the frequencies of these cases are not the same and the car-following case is the dominating one in a common traffic scenario. To deal with the data-imbalance problem and improve the performance of the neural network, we increase the proportion of BLC and lane-changing pattern I cases by randomly sampling the same number of pieces from each data pool (lane-changing-left, lane-changing-right, and car-following, respectively). In this way, the Stage-1 model can learn more about lane-changing. The same strategy is also applied to train the Stage-2 model. To train both models, we set the learning rate to be 0.00125 in the first 60 epochs, 0.000625 in the second 60 epochs, and we use softmax cross-entropy as the loss function. Intermediate cells of the single-stage model can be either Bidirectional LSTM (Bi-LSTM) or LSTM Cell, as illustrated in Figure 8. While the former produces slightly higher accuracy, the latter is simpler and more efficient for training and testing.

We will now briefly explain how to use the well-trained model to predict the intention of vehicles in real time. The algorithm is presented in Algorithm 1. After checking whether the vehicle changes lanes in the previous 2 seconds, we assign the testing data to the corresponding stage model for prediction. For the before-lane-changing model, the intention of the vehicle simply equals the category with the highest probability. As for the after-lane-changing model, however, the situation is much more complex. For a given state, it is necessary to compare its current dx to a threshold derived from statistics analysis (Figure 6). dx is a variable that describes the lateral moving distance of a vehicle after lane-changing, as depicted in Figure 6. $dx > 0$ indicates a lane-changing-right while $dx < 0$ implies a lane-changing-left. If $|dx|$ is in the range of the threshold and the predicted label does not match the orientation of dx, it is highly likely that the vehicle is fluctuating along the line. In this case, we tend to assign it with the car-following label. Otherwise. the predicted behavior of the vehicle belongs to the category with the highest probability.

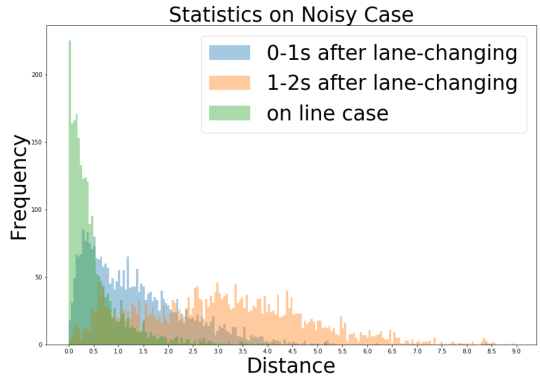

Fig. 6. Statistics of moving distance dx between a lane-changing case and a driving on-line case.

Table 1. Comparison Results on Test Set with Each Testing Scenario Randomly Selected in all Testing Scenarios

Method	Real Label	Predict Label		
		Following	Left	Right
Baseline Model I	Following	95.07%	1.83%	3.10%
	Left	13.91%	85.17%	0.92%
	Right	11.06%	0.99%	87.95%
Baseline Model II (LSTM)	Following	93.96%	2.43%	3.61%
	Left	11.72%	87.00%	1.28%
	Right	8.99%	1.33%	89.68%
Baseline Model II (Bi-LSTM)	Following	96.85%	1.31%	1.84%
	Left	9.04%	89.24%	1.72%
	Right	10.68%	0.84%	88.49%
CS-LSTM [10]	Following	96.93%	1.36%	1.71%
	Left	9.84%	88.40%	1.76%
	Right	11.47%	1.51%	87.02%
2-Stage Model (LSTM)	Following	96.67%	1.33%	2.00%
	Left	7.77%	**91.78%**	0.45%
	Right	7.56%	0.50%	**91.94%**
2-Stage Model (Bi-LSTM)	Following	**97.09%**	0.94%	1.97%
	Left	8.53%	91.04%	0.43%
	Right	7.45%	0.61%	**91.94%**

3.4 Numerical Results

To evaluate the performance of our 2-stage model, we conduct extensive experiments. For a fair comparison, we compare the performance of our 2-stage models with different baseline models using the same labeled set for training and testing. In the first experiment, we utilize the test set with all vehicles randomly selected, each at a random time epoch. The results are summarized in Table 1, in which we can see that all models achieve relatively high accuracy in these three classification tasks. Baseline Model I is similar to the LSTM model utilized in [52] but without neighbor features as input, which achieves the lowest performance in lane-changing prediction. In contrast to Baseline Model I, Baseline Model II is designed with multiple LSTM layers, as shown in Figure 8. The

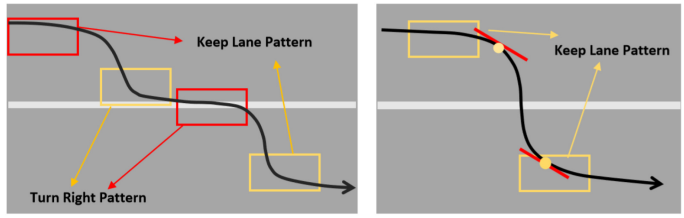

Fig. 7. (a) Red and yellow boxes show that there are the same driving patterns assigned to different behavior labels. (b) Yellow box shows that there are different driving patterns assigned to the same behavior label.

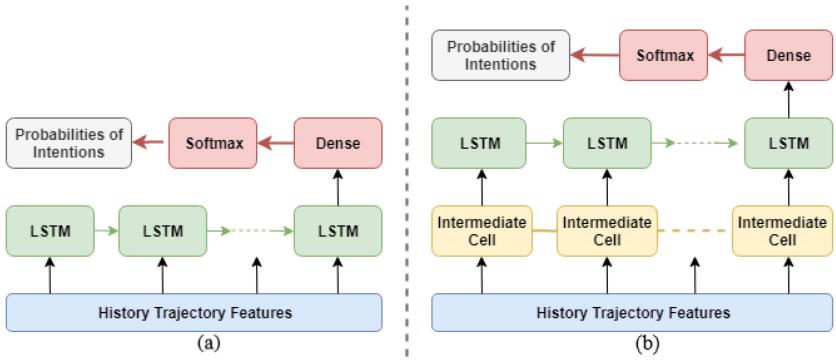

Fig. 8. Visual illustration of different baseline models, also used as single-stage model. (a) Baseline model I: single LSTM layer (same as LSTM model in [52] without neighbor features as input). (b) Baseline model II: Multiple LSTM layers. Intermediate cell can be either Bi-LSTM or LSTM. If Bi-LSTM is selected, the message is communicated in both directions.

accuracy of Baseline Model II is improved while applying a more complicated network structure Bi-LSTM (i.e., Baseline Model II (Bi-LSTM)). However, the accuracy of lane-changing is still below 90%. We also investigate the effect of neighboring information by utilizing social pooling [10]. The proposed CS-LSTM [10] achieves high accuracy (96.93%) on predicting lane-following behavior, but it sacrifices the accuracy of lane-changing cases. Compared with the CS-LSTM model and Baseline Model II (Bi-LSTM), our 2-stage models, including 2-stage model (LSTM) and 2-stage model (Bi-LSTM), achieve about 3% higher accuracy in predicting lane-changing cases since our model can handle the differences in driving patterns of different stages well, as illustrated in Figure 7. Such differences may confuse the single-stage models discussed earlier. Moreover, our 2-stage model (LSTM) is as efficient as the single-stage Baseline Model II (LSTM) in terms of computation time. Due to the complicated network structures of Baseline Model II (Bi-LSTM) and CS-LSTM, these two models are more time-consuming than our 2-stage model (LSTM).

Since the prediction of before and after lane-changing is more important to our framework, we conduct a second experiment in which we choose test cases randomly but only at the time epoch when the vehicle is 4 seconds before or after the line-crossing. As shown in Table 2, in such a more challenging scenario, the accuracy of the single-stage model (Baseline Models I and II) as well as CS-LSTM decreases considerably, especially for the accuracy of the car-following case. By comparison, our 2-stage models can still achieve high accuracy for all three cases, which demonstrates the advantages and potentials of the proposed method. Also, compared with the first two baseline models, the baseline model with Bi-LSTM as intermediate cells significantly

Table 2. Comparison Results on Test Set with Each Testing Scenario Randomly Selected in ±4 s Interval of Cross-over

Method	Real Label	Predict Label		
		Following	Left	Right
Baseline Model I	Following	75.91%	14.62%	9.47%
	Left	14.46%	84.16%	1.38%
	Right	12.29%	1.34%	86.36%
Baseline Model II (LSTM)	Following	76.67%	15.62%	7.71%
	Left	10.64%	87.69%	1.67%
	Right	12.54%	2.86%	84.60%
Baseline Model II (Bi-LSTM)	Following	80.74%	11.47%	7.79%
	Left	9.26%	88.21%	2.53%
	Right	11.22%	1.87%	86.91%
CS-LSTM [10]	Following	79.61%	12.84%	7.55%
	Left	10.01%	87.49%	2.50%
	Right	11.51%	2.42%	86.07%
2-Stage Model (LSTM)	Following	83.03%	10.36%	6.61%
	Left	7.73%	**90.97%**	1.30%
	Right	8.32%	1.60%	90.08%
2-Stage Model (Bi-LSTM)	Following	**84.01%**	9.20%	6.79%
	Left	8.51%	90.29%	1.20%
	Right	7.80%	1.87%	**90.33%**

boosts overall performance, particularly in the accuracy of the car-following cases. However, this distinction is eliminated for the 2-stage models in that the 2-stage model (LSTM) achieves nearly the same performance as the 2-stage model (Bi-LSTM). As a result, we utilize the 2-stage model (LSTM) given its higher efficiency and high accuracy.

4 SAMPLING WITH IMITATION LEARNING AND PREDICTION

In this section, we design a new sampling model by integrating the prediction result from Section 3 and imitation learning. Through leveraging prediction results, some potential collision spaces can be eliminated from the state space and the generated trajectory can be valid for a longer time. Imitation learning [31, 37] can further help reduce the useless sample points while keeping the sample points near the human driving trajectory. Given these "important" sample points, a safer and human-like trajectory with smoother velocity and longer valid time period can be generated by the planning module. To achieve this goal, we need to design and train a unified model to generate sample points that are collision-free in the given scenario with any predicted environment, initial state, and goal state as input. We apply Conditional Variational Autoencoder (CVAE), which is good at learning from the given data and generating new data, to learn this sampling distribution.

In the rest of this section, we first introduce the basics of CVAE. Then, we describe how to apply the CVAE model to our problem and its training process. Finally, we present how to integrate the prediction module and the well-trained CVAE model to generate sample points.

4.1 Basics of CVAE

Variational Autoencoder (VAE) [29] is a mainstream generative model. It can generate new samples that follow the same distribution as the training data. In this article, we choose CVAE [50], an

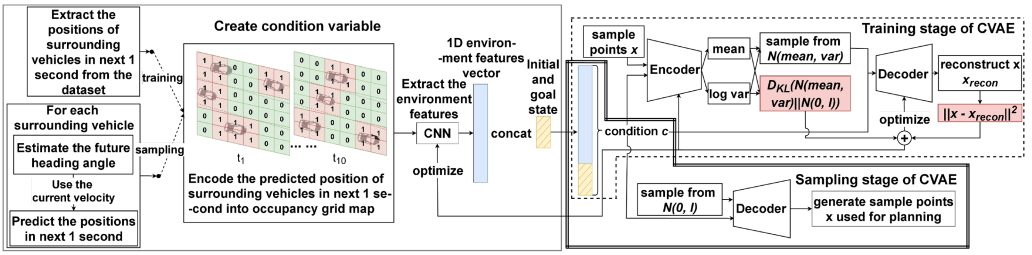

Fig. 9. Architecture of our model applying CVAE.

extension of VAE, as the generative model. With sampled data x, we denote the latent variable of x as z. Conditioned on c, we train CVAE to maximize the objective function [50] for a given sample x:

$$E_{q_\phi(z|x,c)}[\log p_\psi(x|z,c)] - D_{KL}[q_\phi(z|x,c)||p_\psi(z|c)], \tag{1}$$

where ϕ and ψ are parameters of encoder and decoder functions, respectively. Given sample x, we first utilize an encoder to capture the distribution of its latent variable z conditioned on c and approximate the distribution to a function $p_\psi(z|c)$. After decoding the latent variable z conditioned on c, we hope to maximize the expectation of log-likelihood $\log p_\psi(x|z,c)$ in order to regenerate x.

Following [50], we train the CVAE model with two objectives: (1) given the input data x and condition c, the output of the encoder (i.e., the latent variable z) follows a probability distribution $N(\mu, \sigma^2)$ that is close to $N(0, I)$, and (2) the output of the decoder (i.e., the reconstructed $\bar{x}$) must be close to the input x. To achieve the first objective, we define that the encoder can produce the mean and variance of a Gaussian distribution, that is, μ and σ^2. Therefore, we can use the KL divergence between $N(\mu, \sigma^2)$ and $N(0, I)$ as the first part of the loss function to optimize CVAE. To achieve the second objective, we sample random variable z from normal distribution $N(\mu, \sigma^2)$ and then use c and z as the inputs of the decoder to reconstruct $\bar{x}$. Consequently, the second part of the loss function is defined as $||x - \bar{x}||^2$. Once trained, given c, we can sample from $N(0, I)$ to generate $\bar{x}$.

4.2 Our Training Model

The network architecture used for training and sampling of CVAE is shown in Figure 9. In addition to the encoder and decoder networks, we need another convolutional neural network (CNN) to preprocess our 3D environment information, which involves current and future environment information. The surrounding environment contains only the neighboring vehicles in a fixed range of distance. In the training phase, we use the ground truth to describe the future environment. Now, we introduce c, x, loss function, and the training process.

Condition c. In our model, the condition c includes the environment information, the initial state x_{init}, and the goal state x_{goal}. We define the environment information in every 100 ms as a frame. For each frame, we encode it into an occupancy grid map as follows. Let the width of the lane be w, let the lateral distance of the ego vehicle to the lane center be d, let the ego vehicle position be $[p_x, p_y]$, and let the neighboring range be r. Considering the camera and Lidar sensing ability, we set $r = 250$ feet. If the ego vehicle is in the right of the center lane, d is negative and vice versa. We draw a map with a length of $2 * r$ and width $3 * w$. The center of this map is $[p_x - d, p_y]$, which represents the relative position of the ego vehicle. This map is divided evenly into several small grids. As the vehicle size is usually around 9 feet * 16 feet, the size of a grid is set as 4 feet * 6 feet, which is smaller than the vehicle size, to make it sensible to the vehicle movement even in a short time period. Therefore, the whole occupancy grid map is divided into $\frac{3*w}{4} \times 50$ grids. Let (pos_i, pos_j), $i < \frac{3*w}{4}$, $j < 50$ be the center of each grid. Given the 1 s (10 frames)

predicted trajectory of surrounding vehicles, the raw information is encoded in a 3D occupancy matrix $M_{env} \in \mathbb{R}^{m \times n \times 10}$ as follows:

$$M_{env}[i][j][t] = \begin{cases} 1, & if\ (pos_i, pos_j)\ is\ occupied\ at\ time\ t \\ 0, & if\ (pos_i, pos_j)\ is\ free\ at\ time\ t, \end{cases}$$

where $m = \frac{3*w}{4},\ n = 50$.

This 3D occupancy grid matrix is input to the CNN model for contextual feature extraction. The output is a k-dimension feature vector $v_{env} \in \mathbb{R}^k$. Therefore, the condition variable can be represented as a $(k + 8)$-dimension vector which contains $v_{env}, x_{init}, x_{goal}$ as shown in the left part of Figure 9.

Variable x. Variable x contains the ego vehicle's states, which are extracted from the trajectory that starts with x_{init} and ends with x_{goal}. As the environment information in c has the temporal dimension, x also needs to have time-sequential sample points corresponding to the frames in the 3D environment matrix. The variable x is a vector $x \in \mathbb{R}^{10*4+1*4}$, which contains 10 states corresponding to the 10 future frames and 1 random state out of the predicted range. Therefore, x can be represented as $x = [x_1, x_2, \ldots, x_{10}, x_s]$, where x_i is the state of the ego vehicle in the i^{th} predicted frame and x_s is a state out of the predicted range but before the goal state. This additional state x_s helps our model generate sample points from x_{10} to x_{goal} rather than only in the next 1 s since the planning trajectory is usually longer than 1 s. Thus, the objective of finding a feasible trajectory from x_{init} to $\mathcal{X}_{goal}$ can be guaranteed.

We take the trajectory with a length of 40 frames (4 s) and its corresponding environment as a training case, which means that the human driver finishes the trajectory in 4 s. Lots of training cases of different lane-changing scenarios are retrieved evenly. If we use these training cases to train our model directly, the model can be applied only to the cases in which the travel time must be around 4 s. Obviously, this is unpractical. To handle this problem and augment the training data, one lane-changing trajectory is used to generate multiple training cases with different initial states but the same goal state. In other words, the time intervals between the initial state and goal state of the augmented training cases are different.

Loss function. We use the KL divergence, an important term of the CVAE objective function, as one part of the loss function. Another part is the reconstruction error between the input x and the output of the decoder $\bar{x}$. The loss function is:

$$||\bar{x} - x||^2 + \sum_i D_{KL}[q_\phi(z_i|x, c)||p_\psi(z_i|c)]. \tag{2}$$

We model the prior distribution of the latent variable at each future time epoch $p_\psi(z_i|c)$ as an isotropic Gaussian with unit-variance. The loss function penalizes the large divergence between the distribution of each latent variable $N(\mu_i, \sigma_i^2)$ and $N(0, I)$ for approximating $N(\mu_i, \sigma_i^2)$ to $N(0, I)$, where μ_i and σ_i^2 are output by encoder for the i^{th} latent variable. In addition, it encourages $\bar{x}$ to be similar with x as much as possible, which indicates that the time-sequential sample points should be sampled near human driving trajectories.

Training process. The training process is illustrated in Figure 9. The 3D environment matrix is transformed to a 1D vector by the CNN. This vector is concatenated with the initial and the goal states to constitute the condition c. Then, the sample point x is concatenated by the condition c and mapped to the latent space by the encoder. The encoder outputs two values for each latent variable at each future time point: one is mean value μ_i and the other is logarithmic variance $\log \sigma_i^2$. Then, we can sample each latent variable z_i from $N(\mu_i, \sigma_i^2)$ and concatenate them to form the latent variable z. After that, we concatenate z with the same condition c. The decoder projects this vector

ALGORITHM 2: Position Prediction of Surrounding Vehicles

$P_{right}, P_{follow}, P_{left} \leftarrow$ PredictModule
$P_m \leftarrow \text{MAX}(P_{right}, P_{follow}, P_{left})$
$P_s \leftarrow \text{SECONDMAX}(P_{right}, P_{follow}, P_{left})$
if $P_m \geq 80\%$ & $\theta \in [\theta_{m-}, \theta_{m+}]$ **then**
    $\theta_{new} \leftarrow \theta$
else if $P_m \geq 80\%$ & $\theta \notin [\theta_{m-}, \theta_{m+}]$ **then**
    $\theta_{new} \leftarrow \frac{\theta+\theta_m}{2}$
else if $P_m < 80\%$ & $\theta \notin [\theta_{m-}, \theta_{m+}]$ **then**
    $\theta_{new} \leftarrow \frac{\theta+\theta_m}{2} + \eta * \theta_s$
else
    $\theta_{new} \leftarrow \theta + \eta * \theta_s$
end if
for $v \in SurroundingVehicles$ **do**
    for $t \in next\ 1\ second$ **do**
        $pos_t^v \leftarrow [x_0^v + vel^v \cdot t \cdot \sin\theta_{new}, y_0^v + vel^v \cdot t \cdot \cos\theta_{new}]$
    end for
end for
return pos

from the latent space to get $\bar{x}$. The loss value is calculated using Equation (2) to optimize the CNN, encoder network, and decoder network.

4.3 Our Sampling Model

After training, the decoder is capable of using $N(0, I)$ and c to generate $\bar{x}$ as shown in the sampling stage in Figure 9, where c can be obtained by the same way introduced in the training process for given environment information, initial state, and goal state. The initial and goal states can be obtained from the localization system and higher decision layer, respectively. Therefore, to obtain c, an input of the decoder in the sampling stage, we need to predict only the position of surrounding vehicles in the next 1 s and construct the 3D environment matrix.

For each future frame, we need every surrounding vehicle's position and size to complete the corresponding occupancy grid map. The size of each surrounding vehicle can be obtained by the perception system in the autonomous vehicle. The position of a surrounding vehicle in each future frame is calculated using the velocity and heading angle. We can use the current instantaneous velocity to estimate the velocity in the next 1 s. The future heading angle may change a lot in the next 1 s, especially, when this vehicle changes lane. We leverage the output of our prediction module to estimate the future heading angle. Since our prediction module can provide probabilities of lane-changing-left P_{left}, lane-changing-right P_{right}, and car-following P_{follow} for each surrounding vehicle, we sort these three probabilities and define the largest one as P_m, where the subscript m represents the major intention. The second largest one we define as P_s, where the subscript s represents the secondary intention. We define the normal range of heading angle for lane-changing-left, lane-changing-right, and car-following as $[\theta_{follow-}, \theta_{follow+}]$, $[\theta_{left-}, \theta_{left+}]$, and $[\theta_{right-}, \theta_{right+}]$ respectively, according to the statistics of NGSIM real data [13, 14]. For further estimation, let $\theta_{follow} = \frac{\theta_{follow-}+\theta_{follow+}}{2}$, $\theta_{left} = \frac{\theta_{left-}+\theta_{left+}}{2}$, and $\theta_{right} = \frac{\theta_{right-}+\theta_{right+}}{2}$. Denote the current heading angle as θ, estimated future heading angle as θ_{new}, the current velocity of vehicle i as vel^i, and the position of vehicle i at time t as $pos_0^i = [x_t^i, y_t^i]$. The algorithm used to estimate the position of surrounding vehicles in the next 1 s is presented in Algorithm 2.

We encode their future positions into occupancy grid maps and then input them to the CNN to extract the environment features. Then, the vector of environment features is concatenated with the initial and goal state to construct the condition variable c as shown in the left part of Figure 9.

Finally, the well-trained decoder can generate sample points using the aforementioned condition variable c and the points sampled from $N(0, I)$.

5 SELECTION OF THE NUMBER OF SAMPLE POINTS

In the conventional SBMP, the number of sample points is a given input without regard to the environment, which may lead to unnecessarily prolonged sampling time for some simple environment scenarios or insufficient number of sample points to find a feasible trajectory in some complex environment scenarios. In this section, we aim to design a strategy to determine the number of sample points sufficient to find a feasible collision-free trajectory for any given environment scenario so that SBMP does not sample more than necessary or an insufficient number of sample points. To achieve this goal, we need to explore and quantify the relationship among the environment scenario, the number of sample points, the computation time, and the success rate. With such a built relationship, we can find the right number of sample points for each environment scenario that can achieve the lowest computation time while ensuring a similar performance using less sample points.

We focus on the highway traffic scenarios in which the complexity of the environment scenario is mainly characterized by two factors: the number of surrounding vehicles, N_{SV}, and the Euclidean distance between the initial state and goal state, d_{ig}. The former can capture the complexity of the surrounding environment and the latter indicates the size of the state space, which is represented as follows.

$$\begin{aligned} d_{ig} &= ||x_{init} - x_{goal}||_2 \\ &= ((p_{x_{init}} - p_{x_{goal}})^2 + (p_{y_{init}} - p_{y_{goal}})^2 + (v_{init} * \sin\theta_{init} - v_{goal} * \sin\theta_{goal})^2 \\ &\quad + (v_{init} * \cos\theta_{init} - v_{goal} * \cos\theta_{goal})^2)^{\frac{1}{2}}. \end{aligned}$$

In the rest of this section, we first present the general approach to determine the number of sample points empirically. We then discuss how to choose the initial value and step value while applying our approach.

5.1 General Approach

We design an online approach to explore and quantify the relationship between the number of sample points and the environment scenario while running SBMP. The number of sample points is denoted as n and the environment scenario is described using a tuple (N_{SV}, d_{ig}). It is noted that the d_{ig} here is discretized to several intervals; otherwise, all cases will have different d_{ig}. Their relationships, (N_{SV}, d_{ig}, n), are stored in a table, *AdaptiveNumber*, where (N_{SV}, d_{ig}) can be regarded as the key and n here is the value of key (N_{SV}, d_{ig}), which represents the recommended number of sample points in this environment scenario. As the objective is to achieve lower computation time while ensuring a similar success rate, one more table is required to store the average computation time for different n under the same (N_{SV}, d_{ig}), which helps update the value n in table *AdaptiveNumber* according to the lowest computation time. This table is denoted as T, where the key is (N_{SV}, d_{ig}, n) and the value, t, is the average computation time used for planning with n sample points in the environment scenario (N_{SV}, d_{ig}). We briefly explain how to create and maintain these two tables.

- **Initialization.** We initialize table *AdaptiveNumber* and table T as empty tables.
- **Query.** Given the environment scenario (N_{SV}, d_{ig}) as key, retrieve the corresponding value from table *AdaptiveNumber*. If the retrieved n is not empty, run SBMP to find a feasible trajectory. Otherwise, run the insert process.

- **Insert.** For a new environment scenario (N_{SV}, d_{ig}) that has not been encountered before, n is initialized as a large number to guarantee that a feasible trajectory can be found. We run SBMP and record the computation time τ. Insert (N_{SV}, d_{ig}, n) into table *AdaptiveNumber*, then insert $(N_{SV}, d_{ig}, n, \tau)$ into table T.
- **Update** T**.** When finishing SBMP and recording the computation time τ, update the average computation time $(N_{SV}, d_{ig}, n, \frac{(k-1)*t+\tau}{k})$ in table T, where t is the current average computation time and k is the counter for calculating the average computation time.
- **Update** *AdaptiveNumber***.** Table *AdaptiveNumber* is updated in the following cases: (1) If this environment scenario has been encountered several times and the success rate of finding a feasible trajectory is 100%, decrease the value n of key (N_{SV}, d_{ig}) by a predefined step value. (2) In case it fails to find the feasible trajectory using the selected n, let the vehicle follow the current trajectory and increase the value n of key (N_{SV}, d_{ig}) by a predefined step value. (3) Let $T(N_{SV}, d_{ig}, n)$ be the retrieved value t of key (N_{SV}, d_{ig}, n) in table T. If $T(N_{SV}, d_{ig}, n') < T(N_{SV}, d_{ig}, n)$, update the value n to n' for key (N_{SV}, d_{ig}) in table *AdaptiveNumber*, that is, $n = \arg\min_n T(N_{SV}, d_{ig}, n)$. In the case of (1) and (2), after updating, we randomly recalculate a number of cases under the same (N_{SV}, d_{ig}) using the current adaptive number n to test offline the feasibility of it. If there is no failure, we keep the current number n. Otherwise, we further increase it and retest it until it succeeds.

We note that the step value for updating table *AdaptiveNumber* and the initial value of n can be set empirically according to different road, weather, light, and other conditions. In the following subsection, we will show how to set the step value empirically.

5.2 Empirical Study on Initial Value and Step Value

As motion planning is usually used for lane-changing to avoid collisions, we extract all of the lane-changing cases from NGSIM US-101 [14] and I-80 [13] datasets for analysis, where the US-101 dataset is used for empirical study and the I-80 dataset is used for verification of the results (dataset introduction is available in Section 3.3). Specifically, we leverage the empirical result obtained from US-101 and apply it on I-80 to show that the initial value and step value could be the same in the same type of environment. Each extracted case is a lane-changing case of a 4-s human-driving trajectory, including a 2-s trajectory before lane-changing and a 2-s trajectory after lane-changing. In addition to the trajectory of the lane-changing vehicle, surrounding vehicles' trajectories are included in each lane-changing case. The lane-changing vehicle is the subject vehicle in the rest of this section.

For empirical study, we first cluster these lane-changing cases into different clusters according to the numbers of surrounding vehicles. Define C_i as the cluster in which there are i surrounding vehicles for all cases. Define n as the number of sample points and set n as to be 1,000, 500, and 100. For each case in C_i, we apply the planning algorithm FMT* [25] with our sampling model presented in Section 4 to complete the motion planning task for the subject vehicle given the initial state and goal state from the ground truth.

After running all US-101 lane-change cases, we can obtain the average computation time table T. We denote $\bar{t}$ as the average computation time, calculating the average distance, $\bar{d}_{ig}$, and the number of cases that fail to find a feasible solution in each cluster as shown in Table 3. In general, we can observe that only 100 sample points are needed to achieve 100% success rate in the clusters where $N_{SV} < 9$ or $\bar{d}_{ig} < 116.08$ since the surrounding environment is simple or the state space is small. We note that a feasible trajectory cannot be found in some clusters where both N_{SV} and $\bar{d}_{ig}$ are moderate, that is, $N_{SV} > 8$ and $\bar{d}_{ig} >= 137.46$ due to insufficient sample points, that is, 100 in our experiment. Such cases are referred to as failed cases.

Table 3. Computation Time Using Different Numbers of Sample Points in Different Environment Scenarios

N_{SV}	3	4	5	6	7	8	9	10
$\bar{d}_{ig}$	221.74	211.88	198.18	185.78	176.31	174.85	164.03	151.85
$\bar{t}$ (ms) (n = 1,000)	83.58	111.54	115.90	118.80	136.97	131.52	134.90	145.10
$\bar{t}$ (ms) (n = 500)	42.83	53.66	58.30	65.50	79.55	78.74	82.44	87.65
$\bar{t}$ (ms) (n = 100)	12.44	13.74	15.38	17.28	20.38	22.81	23.53[1]	25.15[2]
N_{SV}	11	12	13	14	15	16	17	18
$\bar{d}_{ig}$	137.46	141.80	116.08	114.24	103.24	102.83	90.56	81.99
$\bar{t}$ (ms) (n = 1,000)	147.52	159.43	165.66	161.70	178.35	185.58	184.81	185.64
$\bar{t}$ (ms) (n = 500)	90.80	96.05	98.85	98.53	105.83	106.40	105.10	100.61
$\bar{t}$ (ms) (n = 100)	26.51	28.71[3]	30.10	30.10	30.93	33.51	34.36	33.23

[1]3 cases fail to find feasible trajectory.
[2]1 case fails to find feasible trajectory.
[3]1 case fails to find feasible trajectory.

According to the experiment result, these failed cases occur due to insufficient sample points. While setting up the initial value of n and the step value for updating table *AdaptiveNumber*, we need to avoid such failed cases as much as possible.

- We first discuss the initial value of n. From Table 3, we analyze the value of d_{ig} and N_{SV} in the failed cases and find that d_{ig} is greater than 140 in all of these failed cases, and N_{SV} in these failed cases is greater than 8. Combining these two thresholds, when $N_{SV} > 8 \ \& \ d_{ig} > 140$, the initial value of n should be 500 to ensure the success rate. Otherwise, the initial value of n could be 100.
- As for the step value, it is set to be 50 for those cases with 500 as the initial value. Similarly, the step value is set to be 10 for other cases in which the initial value of n is 100.

Given the aforementioned initial value and step value, we apply our proposed approach on the I-80 dataset and there is no failed case using the predefined initial value.

6 IMPLEMENTATION AND EXPERIMENTS

Given the aforementioned sampling strategy, including the imitation learning-based sampling model and the strategy for selecting the number of sample points, we first present the online planning strategy from the implementation aspects. Then, we evaluate our sampling strategy through conducting extensive data-driven simulations. The code can be found at https://github.com/Yifanny/Integrate-Algorithmic-Sampling-based-Motion-Planning-with-Learning.

6.1 Online Planning Strategy

When the motion planner is put into practice, the motion planner is called at a fixed frequency, for example, 10 Hz [32], to update the planned trajectory. While replanning the trajectory, the ego vehicle is not static. It moves along the current trajectory while replanning the new one, which

leads to the disparity between the initial state used for the replanning process, denoted as x_{init}, and the initial state of executing the replanned trajectory, denoted as x'. This problem is referred to as a time lag problem, which may make the replanned trajectory suboptimal or even infeasible in practice. Some methods have been proposed in [5, 6] to mitigate this problem by estimating the initial state. In such methods, some leading time is allocated to represent the reserved computation time of the motion planner and then predict the initial state using the speed command of the ego vehicle. Although these existing schemes can be applied in some scenarios, their assumption that the computation time is fixed is not valid for our framework. As shown in Table 3, it takes different amounts of time for a motion planner to calculate a collision-free path depending on the surrounding environment, distance between the initial and goal position, and the number of sample points. To tackle this issue, we propose a new scheme to estimate the initial state, which aims to mitigate the time lag problem and can cooperate with existing mitigation schemes.

We estimate the shifted initial state $\tilde{x}'$ and use it as the initial state in both the sampling model and the planning algorithm. Leveraging table T in Section 5, the average computation time $\bar{t}^n_{env}$ is used to estimate the computation time t given (N_{SV}, d_{ig}) and n. As the ego vehicle follows the current reference trajectory until the new one is obtained, given the estimated computation time t, we can calculate $\tilde{x}'$ based on the current trajectory. Denote $\pi(\tau)$ as the current trajectory and t_0 as the current time epoch. x' can be estimated as $\tilde{x}' = \pi(t_0 + \bar{t}^n_{env})$. $\tilde{x}'$ is used as the initial state in both the sampling model and the planning algorithm in the following evaluation experiments.

6.2 Experiments and Numerical Results

To evaluate our sampling strategy, we adopt FMT* [25] as the planning algorithm and conduct data-driven simulations in which we apply the cost function and trajectory-generation method suggested in [55]. Specifically, to evaluate and compare our sampling strategy with other sampling strategies, we focus on the success rate of finding a collision-free trajectory, computation time of the planning algorithm, the travel time of driving over the trajectory, and the acceleration variation. The inference of neural networks that are applied in the sampling model is very time-efficient, that is, less than 10 ms using our personal computer equipped with one GeForce RTX 2080Ti. Moreover, with advances in technology, future computing units in autonomous vehicles can be more powerful [45]. Thus, we compare the computation time of the planning algorithm. Here, the acceleration variation represents the smoothness of the trajectory [57], and a smaller value implies that passengers feel more comfortable. Due to limited space, we compare our scheme with three strategies: uniform sampling, bias Gaussian sampling [33], and critical sampling [24].

Besides comparing with the three models mentioned, we conduct two more experiments to illustrate and validate the improvement obtained by prediction and imitation learning modules individually. In the first experiment, we do not apply the imitation learning module. In the second experiment, we do not apply the prediction module. We conduct these two experiments as follows.

- In the first experiment, we remove the imitation learning module and keep the prediction module. The prediction module helps to identify the potential collision-free space and, thus, reduces the state space. The predicted positions of surrounding vehicles in the next 1 s are regarded as potential collision-free space.
- In the second experiment, we design the imitation learning model without considering the prediction. For a fair comparison, we also apply CVAE to learn the distribution of the human trajectory. In our final complete model, the input of the model includes ten future frames to represent the environment variable. For comparison, in this model, we take only the current frame as the environment variable. We also use a CNN to extract the environment feature from the environment variable. As is done for the final model, we concatenate the initial

Table 4. Success Rate Versus Replanning Time Interval

Replanning time interval	200 ms	400 ms	600 ms	800 ms	900 ms
Uniform	0%	0%	0%	0%	0%
Bias Gaussian	70%	60%	30%	0%	0%
Critical sampling	100%	100%	100%	80%	50%
Partial model without imitation learning	100%	100%	100%	90%	70%
Partial model without prediction	100%	100%	100%	70%	30%
Complete model with imitation learning and prediction	100%	100%	100%	90%	70%

state, the goal state, and the extracted environment feature to constitute the condition variable. Since there is only the current frame representing the environment variable, we define variable x as a state between the initial state and the goal state. The other network structure is designed the same as the final complete model. The loss function also contains the KL divergence and the reconstruction error. After training, similar to the complete model, it can generate sample points given the points sampled from $N(0, I)$ and the condition variable. The condition variable contains the extracted environment feature, the initial state, and the goal state.

In our complete sampling model, the CNN is designed using 16 kernels with size 5. The encoder and decoder have the same architecture, which contains two fully connected layers, where the first one has 512 hidden units and the second one has 128 hidden units. They share the CNN with the same weights to extract environment features. As the NGSIM US-101 dataset [14] includes 3 sub-datasets, we use two for training and one for testing. We utilize an Adam optimizer with a learning rate of 0.0001 and a batch size of 256 to train the model for 500 epochs. Since both prediction module and imitation learning module are trained with the NGSIM dataset, we select lane-changing cases from the intersection of test datasets to evaluate the performance. While applying the critical sampling strategy [24] for comparison, we generate the critical samples by training a three-layer fully connected neural network using the same training dataset. The criticality of each sample point in the training dataset is computed via the betweenness centrality as outlined in [24].

The data-driven simulation experiments are conducted as follows. We randomly extract several lane-changing scenarios from the testing dataset, where each lane-changing scenario includes a lane-changing vehicle (i.e., the ego vehicle) and several surrounding vehicles. Thus, we can have the whole lane-change trajectory of the human driver, in which the state of the vehicle 2 s before it crosses the line between two lanes is set as the initial state of the ego vehicle, and its goal state is the state 2 s after it crosses the line. We let our motion planner control the ego vehicle to change lane, and other surrounding vehicles keep traveling according to the ground truth. Once the motion planning is triggered, we generate a set of sample points using our sampling model, given 1 s predicted information. Then, we use the FMT* [25] algorithm to plan the path and, finally, generate the trajectory for the ego vehicle using the method developed in [55]. We apply a different replanning time interval as shown in Table 4. Since [55] utilizes a fixed-final-state-free-final-time controller, we skipped further discussions on the impacts of different controllers and assume that the ego vehicle can travel on the generated trajectory. Figure 10(a) shows an example of generated sample points and the ground-truth trajectory in the next 1 s using these points. To illustrate that our model is not overfitting, we test on not only the US-101 dataset but also the I-80 dataset. The randomly selected test cases from the I-80 dataset in Figure 10(b) show that our model also performs well on the I-80 dataset. Note that the surrounding environments in these images are drawn according to the occupancy grid matrix. Thus, the same vehicle in different frames may occupy a different number of grids.

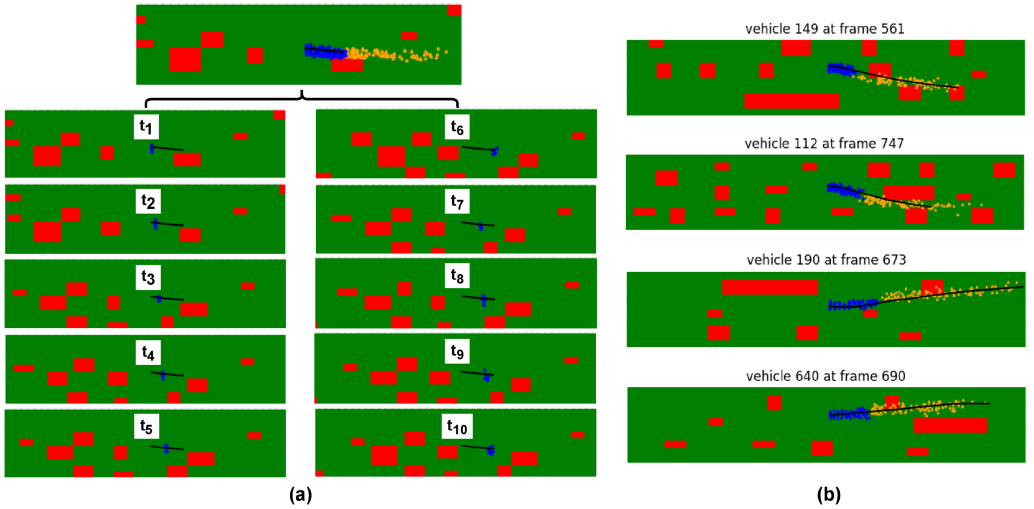

Fig. 10. Several test cases of US-101 and I-80 applying our sampling model. The red grids represent surrounding vehicles and the green zone represents the collision-free space. The top image shows the final generated sample points. The blue points are the sample points in the next 1 s and the orange ones are the sample points in the next 1-4 s. (a). An example of US-101 using our sampling strategy. The black line is the ground-truth trajectory in the next 1 s. Bottom images show the separated next 1-s sample points according to different time epochs, which illustrates that the generated sample points are collision-free at the corresponding time epoch. (b). Randomly selected several lane-change cases from I-80 dataset. The black line is the ground-truth trajectory in the next 4 s.

In the rest of the section, we randomly select one lane-changing case for illustration. The initial state is before lane-changing and the goal state is after lane-changing. The travel time for the human driver is 4 s. There are 11 surrounding vehicles at the initial state. As it is difficult to find a feasible solution using a small number of sample points generated by uniform sampling, we set the number of sample points as 1,000 by default for a fair comparison. Here, we repeatedly run the replanning process until the vehicle arrives at the goal region or collision occurs.

We first compare the success rate using different sampling strategies under different replanning time intervals. In this experiment, if the ego vehicle arrives at the goal region without any collision, it is regarded as a success. The number of sample points is set as 1,000. We run the FMT* algorithm under each sampling strategy with 10 randomly selected lane-changing cases from the test dataset. From Table 4, we can see that our sampling strategy performs much better than the others. The critical sampling strategy is designed for the static environment without considering the movement of surrounding obstacles. Thus, the success rate decreases with the increase of the replanning time interval. In the model without the imitation learning module, the success rate is not decreased since the potential collision-free space is removed by the prediction module. The results in Table 4 show that the prediction module highly affects the success rate. With the prediction module, our sampling strategy has a higher chance to provide sample points which lead to a collision-free trajectory.

Then, we consider the travel time and computation time when FMT* succeeds under bias Gaussian, critical sampling, and our sampling strategies. We first compare the performance among the partial model without the imitation learning module, the partial model without the prediction module, and the complete model to validate the impact of each individual module on the overall performance. We set the replanning time interval as 300 ms and select the trajectory with the

Table 5. Computation Time in Different Replanning Steps

Time step (ms)	Bias Gaussian (ms)	Critical sampling (ms)	Partial model without imitation learning (ms)	Partial model without prediction (ms)	Complete model (ms)
0	158	61	472	280	19
300	199	232	65	271	254
600	217	120	243	129	236
900	945	215	599	568	235
1200	279	258	743	351	180
1500	203	175	76	102	167
1800	76	197	596	92	199
2100	55	52	31	2	34
2400	8	19	54	4	32
2700	11	1	77	1	2
3000	5	arrived	arrived	2	arrived
3300	arrived	-	-	arrived	-
Average time	196	133	296	164	136

shortest travel time for each sampling strategy. As illustrated in Table 5, the computation efficiency is low in the partial models, especially the one without the imitation learning module. The main reason behind this is the existence of some useless sample points in the partial models. In the rest of this section, we mainly compare our complete model with other baseline models. The results in Table 5 show that both the critical sampling strategy and our strategy take less time (3 s) than a human driver (4 s) and bias Gaussian (3.3 s) to reach the goal state. Table 5 shows that the computation time for all of them first increases and then decreases as the ego vehicle approaches the goal state, which is mainly caused by the varying of search space over time. The results show that the average computation time of replanning with the bias Gaussian sampling strategy is much longer than the two other strategies. Our sampling strategy achieves almost the same performance as critical sampling in terms of computation efficiency. Moreover, our computation time is always less than the replanning time interval of 300 ms. This experiment demonstrates that our method is capable of producing a safe trajectory with high efficiency.

Our sampling strategy is capable of generating a smoother trajectory, which is measured by the acceleration variation as shown in Figure 11. To compare the smoothness of the generated trajectory under each given replanning time interval, we first choose the test cases in which the ego vehicle arrives at the goal region without any collision under all three sampling strategies then choose the test case of the trajectory with the shortest travel time for comparison. We calculate the acceleration for each trajectory to represent the smoothness of the trajectory. As shown in Figure 11, the acceleration variation of the trajectory generated by our sampling strategy is always smaller than that of the bias Gaussian sampling strategy and critical sampling strategy for all replanning time intervals, which demonstrates that our sampling strategy helps generate a smoother trajectory than the others. Moreover, we find that a longer replanning time interval leads to a smaller variance for the bias Gaussian and our sampling strategies, which implies that a longer replanning time interval alleviates the jerky velocity problem and brings more comfort to passengers.

7 RELATED WORKS

Trajectory and intention prediction has become an increasingly significant topic in the field of autonomous driving. Understanding the future movements of adjacent vehicles will contribute to better motion planning for the ego vehicle. Trajectory forecasting [10, 21, 36, 42] and intention prediction algorithms [39, 52] are introduced as follows. The former directly generates future trajectories of nearby vehicles, while the latter focuses on the intention (e.g., lane-changing

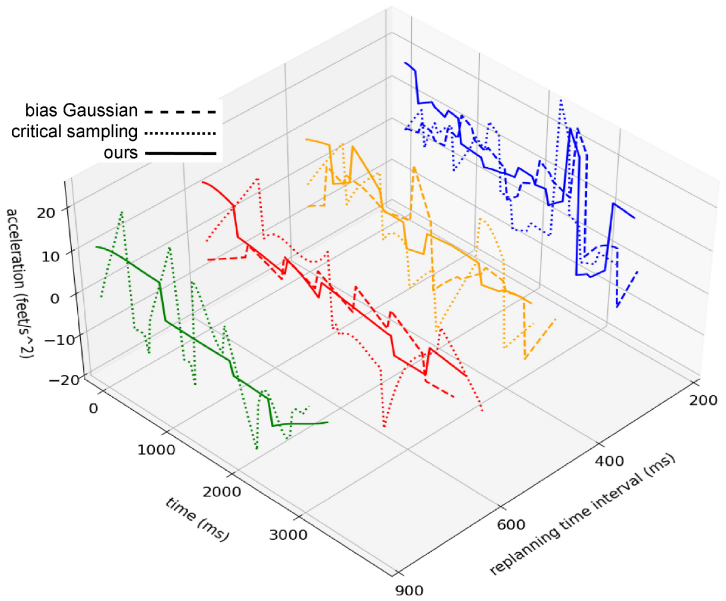

Fig. 11. Acceleration variation with different replanning time intervals.

behavior) of each neighbor. Among trajectory forecasting algorithms, some studies [21, 36] try to model on-road uncertainty with advanced techniques (e.g., Bayesian neural network), while others [10] utilize well-designed modules (e.g., social pooling, graph neural network) to model social interaction. Although these algorithms may help to produce more accurate trajectories and give multimodal results, these models are relatively more complex, requiring more information that may not always be available, and the computational time is much longer when compared with intention prediction methods. Intention prediction algorithms [39, 52] produce very accurate prediction results regarding driver's intention (around 90%). In the highway scenario, where the behavior is predictable given the intention, intention prediction is an efficient and reasonable way to forecast future situations.

Motion planning has been studied in the robotic domain for decades. There are four main types of motion planning algorithms: graph search–based methods [15, 43], numerical optimization approaches [30, 57], interpolating curve methods [17, 34], and sampling-based methods [27, 33].

Graph search–based methods: Algorithms in this category will first discretize the configuration space into lattices [15], handcrafted lane graphs, or cell-grids [41], and then apply the searching algorithm to find the shortest path, such as Dijkstra, A*. Many teams apply this type of algorithm for autonomous driving motion planning [15, 43] because of its capability of searching the optimal path in a short time. It can fulfill the real-time requirement for the driverless vehicle well. The main issue with this type of method is that it cannot guarantee the continuity of the resulting path.

Numerical optimization: The motion planning problem can be formulated as a numerical optimization problem that aims to minimize or maximize the predefined objective function subject to some constraints. Usually, the objective function encourages a shorter travel time, a more smooth path, and penalizes the jerky velocity. The kinematic constraints, collision-free requirements, and the vehicle's physical limits (e.g., velocity, acceleration, and steering angle) are considered to be the constraints. Given such an optimization problem, many algorithms can be applied to solve it,

for example, sequential quadratic programming (SQP) [57]. However, it is time-consuming and cannot guarantee finding a feasible solution.

Interpolating curves: Given a set of waypoints and a parametric interpolating curve, it can construct a smooth trajectory that passes these waypoints and avoids obstacles. There are several commonly used interpolating curves, including lines and circles [48], clothoid curves [34], polynomial curves [17], Bézier curves [18], and spline curves [12]. This kind of method is simple for implementation and does not require high computational power. The problem is that the shape of the curve is completely controlled by parameters. In addition, the generated trajectory totally depends on the given waypoints, which is not robust enough.

Sampling-based methods: We have introduced this kind of method in Section 2. It is capable of finding the solution in the higher-dimensional configuration space while maintaining the optimality of the solution. RRT and PRM are two of the most famous sampling-based algorithms. RRT samples one point at a time; thus, it needs multiple rounds of sampling and planning. PRM samples n points at the same time and plans once. Many extensions of RRT and PRM have been developed for practical usage [33, 40]. Some work has been proposed to accelerate SBMP by integrating machine learning [2, 9, 23, 24]. However, some studies [2, 9] are designed specifically for RRT$^\sharp$ [1] and Augmented CL-RRT [8]. The authors of [23, 24] do not consider the dynamic environment. The work proposed in this article can be applied to any sampling-based algorithm that samples all points at the same time. Moreover, our framework is adaptive in the dynamic environment with the prediction of the environment.

Artificial potential fields (APF) methods: The APF method is also used to solve this problem [7, 22, 47]. It assumes that the vehicle moves in a field that consists of an attractive field, that is, the goal state, and several repulsive fields, that is, surrounding obstacles and road boundaries. The potential gradient descent algorithm is used to determine the feasible path. The main limitation of this method is that the solution may be trapped in local minima. Also, it is very hard to accurately model the potential field in a complex environment.

Many deep learning–based methods, called *imitation learning*, are proposed to solve this traditional problem. Reinforcement learning [20, 37], behavioral cloning [53], and generative adversarial networks [31] are the main techniques used in this domain. This kind of method aims to learn from human drivers and teach the driverless vehicle how to drive. Similar to most of the deep learning techniques, it requires a huge amount of data and powerful computational hardware to support it. Furthermore, the training data are usually collected by normal driving cases so that the final learned driving policy may not be able to handle some abnormal emergency cases. Instead of learning driving policy, our proposed framework leverages imitation learning to learn the sampling distribution and applies the algorithmic method to find a feasible trajectory, which can guarantee a collision-free trajectory with shorter computation time.

8 CONCLUSION AND FUTURE WORK

In this article, we integrate imitation learning and algorithmic motion planning to develop a new SBMP framework. It inherits the strength of algorithmic motion planning in finding a collision-free trajectory. Meanwhile, it adopts different learning models to supply more accurate input to algorithmic motion planning so that a smooth human-like trajectory can be generated. In particular, by predicting surrounding vehicles' intention, accurate future environment information can be supplied to motion planning. Meanwhile, through imitation learning, points near the human-driving trajectory will be sampled, which helps to find a collision-free and smooth trajectory in shorter time. To make the proposed framework more practical, we also conducted an empirical study on adaptively choosing the number of points to be sampled and designed an online

planning strategy. Data-driven experiments show that our sampling strategy not only accelerates the computation but also alleviates the jerky velocity. Moreover, our sampling strategy can generate a trajectory that leads to less driving time than that by human drivers. The proposed framework can be widely applied on top of any existing sampling-based motion planning algorithms that sample all points at the same time.

Our framework can be further explored in more general traffic scenarios, such as urban road areas where the behavior of traffic participants is typically disordered and hard to predict. To this end, we count on not only the prediction algorithm but also the sampling model to prolong the valid prediction period and handle a higher degree of uncertainty. We believe that studies on making better use of road information and modeling social interaction will jointly improve the performance of our sampling model. Moreover, in addition to the lane-changing behavior of vehicles, the intention of vulnerable road users and the intention of vehicles in junction scenarios, that is, turning behavior, are equally important and worth of further investigation.

REFERENCES

[1] O. Arslan and P. Tsiotras. 2013. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany. IEEE, 2421–2428.

[2] O. Arslan and P. Tsiotras. 2015. Machine learning guided exploration for sampling-based motion planning algorithms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'15)*, Hamburg, Germany. IEEE, 2646–2652.

[3] H. Banzhaf, L. Palmieri, D. Nienhuser, T. Schamm, S. Knoop, and J. M. Zollner. 2017. Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments. In *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC'17)*, Yokohama, Japan. IEEE, 1–8. https://doi.org/10.1109/ITSC.2017.8317757

[4] K. Berntorp, R. Bai, K. F. Erliksson, C. Danielson, A. Weiss, and S. D. Cairano. 2020. Positive invariant sets for safe integrated vehicle motion planning and control. *IEEE Transactions on Intelligent Vehicles* 5, 1 (2020), 112–126.

[5] K. Berntorp, T. Hoang, and S. Di Cairano. 2019. Motion planning of autonomous road vehicles by particle filtering. *IEEE Transactions on Intelligent Vehicles* 4, 2 (2019), 197–210.

[6] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano. 2018. Control architecture design for autonomous vehicles. In *IEEE Conference on Control Technology and Applications (CCTA'18)*, Copenhagen, Denmark. IEEE, 404–411.

[7] F. Bounini, D. Gingras, H. Pollart, and D. Gruyer. 2017. Modified artificial potential field method for online path planning applications. In *IEEE Intelligent Vehicles Symposium (IV'17)*, Los Angeles, CA, USA. IEEE, 180–185.

[8] Amit Chaulwar, Michael Botsch, Torsten Krüger, and Thomas Miehling. 2016. Planning of safe trajectories in dynamic multi-object traffic-scenarios. *Journal of Traffic and Logistics Engineering* 4, 1 (01 2016), 135–140. https://doi.org/10.18178/jtle.4.2.135-140

[9] A. Chaulwar, M. Botsch, and W. Utschick. 2016. A hybrid machine learning approach for planning safe trajectories in complex traffic-scenarios. In *15th IEEE International Conference on Machine Learning and Applications (ICMLA'16)*. 540–546.

[10] Nachiket Deo and Mohan M. Trivedi. 2018. Multi-modal trajectory prediction of surrounding vehicles with maneuver based LSTMs. In *IEEE Intelligent Vehicles Symposium (IV'18)*, Changshu, China. IEEE, 1179–1184.

[11] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 1 (1959), 269–271.

[12] Rida T. Farouki. 2008. *Pythagorean-Hodograph Curves*. Springer.

[13] Federal Highway Administration. 2006. US highway 80 dataset. https://www.fhwa.dot.gov/publications/research/operations/06137/.

[14] Federal Highway Administration. 2007. US highway 101 dataset. https://www.fhwa.dot.gov/publications/research/operations/07030/.

[15] Dave Ferguson, Thomas M. Howard, and Maxim Likhachev. 2008. Motion planning in urban environments. *Journal of Field Robotics* 25, 11–12 (2008), 939–960.

[16] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. 2014. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'14)*, Chicago, IL, USA. IEEE, 2997–3004. https://doi.org/10.1109/IROS.2014.6942976

[17] Sébastien Glaser, Benoit Vanholme, Saïd Mammar, Dominique Gruyer, and Lydie Nouveliere. 2010. Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction. *IEEE Transactions on Intelligent Transportation Systems* 11, 3 (2010), 589–606.

[18] David González, Joshue Pérez, Ray Lattarulo, Vicente Milanés, and Fawzi Nashashibi. 2014. Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC'14)*, Qingdao, China. IEEE, 1430–1435.

[19] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[20] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, Barcelona, Spain. Curran Associates, Inc. 4565–4573.

[21] X. Huang, S. G. McGill, B. C. Williams, L. Fletcher, and G. Rosman. 2019. Uncertainty-aware driver trajectory prediction at urban intersections. In *International Conference on Robotics and Automation (ICRA'19)*, Montreal, QC, Canada. IEEE, 9718–9724.

[22] Y. Huang, H. Ding, Y. Zhang, H. Wang, D. Cao, N. Xu, and C. Hu. 2020. A motion planning and tracking framework for autonomous vehicles based on artificial potential field elaborated resistance network approach. *IEEE Transactions on Industrial Electronics* 67, 2 (2020), 1376–1386.

[23] B. Ichter, J. Harrison, and M. Pavone. 2018. Learning sampling distributions for robot motion planning. In *IEEE International Conference on Robotics and Automation (ICRA'18)*. 7087–7094.

[24] B. Ichter, E. Schmerling, T. W. E. Lee, and A. Faust. 2020. Learned critical probabilistic roadmaps for robotic motion planning. In *IEEE International Conference on Robotics and Automation (ICRA'20)*, Paris, France. IEEE, 9535–9541. https://doi.org/10.1109/ICRA40945.2020.9197106

[25] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. 2015. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research* 34, 7 (2015), 883–921.

[26] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schröder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, et al. 2008. Team AnnieWAY's autonomous system for the 2007 DARPA urban challenge. *Journal of Field Robotics* 25, 9 (2008), 615–639.

[27] Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30, 7 (2011), 846–894.

[28] Lydia E. Kavraki, Mihail N. Kolountzakis, and J.-C. Latombe. 1996. Analysis of probabilistic roadmaps for path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 4, Minneapolis, MN, USA. IEEE, 3020–3025.

[29] Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv:1312.6114*.

[30] Dmitriy Kogan and R. Murray. 2006. Optimization-based navigation for the DARPA grand challenge. In *Conference on Decision and Control (CDC'06)*, San Diego, CA, USA. IEEE, 1–6.

[31] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer. 2017. Imitating driver behavior with generative adversarial networks. In *IEEE Intelligent Vehicles Symposium (IV'17)*, Los Angeles, CA, USA. IEEE, 204–211.

[32] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How. 2008. Motion planning for urban driving using RRT. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'08)*, Nice, France. IEEE, 1681–1686.

[33] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. 2009. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology* 17, 5 (Sep. 2009), 1105–1118.

[34] Larissa Labakhua, Urbano Nunes, Rui Rodrigues, and F. S. Leite. 2008. Smooth trajectory planning for fully automated passengers vehicles: Spline and clothoid based methods and its simulation. In *Informatics in Control Automation and Robotics*. Springer, 169–182.

[35] Steven M. LaValle. 2006. *Planning Algorithms*. Cambridge University Press.

[36] Jiachen Li, Hengbo Ma, Wei Zhan, and Masayoshi Tomizuka. 2019. Coordination and trajectory prediction for vehicle interactions via Bayesian generative modeling. In *IEEE Intelligent Vehicles Symposium (IV'19)*, Paris, France. IEEE, 2496–2503.

[37] Yunzhu Li, Jiaming Song, and Stefano Ermon. 2017. InfoGAIL: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems 30*, Long Beach, CA, USA, Curran Associates, Inc., 3812–3822.

[38] W. Lim, S. Lee, M. Sunwoo, and K. Jo. 2018. Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method. *IEEE Transactions on Intelligent Transportation Systems* 19, 2 (Feb 2018), 613–626.

[39] Shiwen Liu, Kan Zheng, Long Zhao, and Pingzhi Fan. 2019. A driving intention prediction method based on hidden Markov model for autonomous driving. *arXiv:1902.09068*.

[40] L. Ma, J. Xue, K. Kawabata, J. Zhu, C. Ma, and N. Zheng. 2015. Efficient sampling-based motion planning for on-road autonomous driving. *IEEE Transactions on Intelligent Transportation Systems* 16, 4 (Aug 2015), 1961–1976. https://doi.org/10.1109/TITS.2015.2389215

[41] Fabio M. Marchese. 2006. Multiple mobile robots path-planning with MCA. In *International Conference on Autonomic and Autonomous Systems (ICAS'06)*, Silicon Valley, CA, USA, IEEE, 56–56.
[42] K. Messaoud, I. Yahiaoui, A. Verroust-Blondet, and F. Nashashibi. 2019. Relational recurrent neural networks for vehicle trajectory prediction. In *IEEE Intelligent Transportation Systems Conference (ITSC'19)*, Auckland, New Zealand. IEEE, 1813–1818.
[43] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. 2008. Junior: The Stanford entry in the urban challenge. *Journal of Field Robotics* 25, 9 (2008), 569–597.
[44] Jianqiang Nie, Jian Zhang, Xia Wan, Wanting Ding, and Bin Ran. 2016. Modeling of decision-making behavior for discretionary lane-changing execution. In *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC'16)*, Rio de Janeiro, Brazil. IEEE, 707–712.
[45] NVIDIA. [n.d.]. NVIDIA Drive AGX. Retrieved December 18, 2021 from https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/.
[46] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1, 1 (2016), 33–55.
[47] Y. Rasekhipour, A. Khajepour, S. Chen, and B. Litkouhi. 2017. A potential field-based model predictive path-planning controller for autonomous road vehicles. *IEEE Transactions on Intelligent Transportation Systems* 18, 5 (2017), 1255–1267.
[48] James Reeds and Lawrence Shepp. 1990. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics* 145, 2 (1990), 367–393.
[49] Oliver Scheel, Loren Schwarz, Nassir Navab, and Federico Tombari. 2018. Situation assessment for planning lane changes: Combining recurrent models and prediction. In *IEEE International Conference on Robotics and Automation (ICRA'18)*, Brisbane, QLD, Australia. IEEE, 2082–2088.
[50] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems 28*, Montréal, Canada. Curran Associates, Inc., 3483–3491.
[51] Anthony Stentz. 1997. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*. Springer, 203–220.
[52] Shuang Su, Katharina Muelling, John M. Dolan, Praveen Palanisamy, and Priyantha Mudalige. 2018. Learning vehicle surrounding-aware lane-changing behavior from observed trajectories. In *IEEE Intelligent Vehicles Symposium (IV'18)*, Changshu, China. IEEE, 1412–1417.
[53] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral cloning from observation. *arXiv:1805.01954*.
[54] Waymo. [n.d.]. Waymo One. Retrieved December 18, 2021 from https://waymo.com/waymo-one/.
[55] D. J. Webb and J. van den Berg. 2013. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *IEEE International Conference on Robotics and Automation (ICRA'13)*, Karlsruhe, Germany. IEEE, 5054–5061.
[56] Yifan Zhang, Jinghuai Zhang, Jindi Zhang, Jianping Wang, Kejie Lu, and Jeff Hong. 2020. A novel learning framework for sampling-based motion planning in autonomous driving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, New York, NY, USA. AAAI, 1202–1209.
[57] J. Ziegler, P. Bender, T. Dang, and C. Stiller. 2014. Trajectory planning for Bertha—A local, continuous method. In *IEEE Intelligent Vehicles Symposium Proceedings*, Dearborn, MI, USA. IEEE, 450–457. https://doi.org/10.1109/IVS.2014.6856581

Received December 2020; revised April 2021; accepted May 2021