Real-time Motion Planning Framework for Autonomous Vehicles with Learned Committed Trajectory Distribution

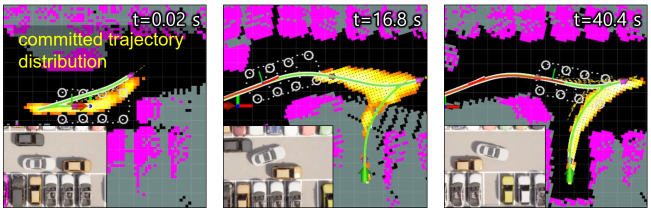
Minsoo Kim[1], Seho Shin[2], Joonwoo Ahn[1], and Jaeheung Park[1,3]

***Abstract*— This study proposes a real-time motion planning framework that leverages the prediction of a portion of the optimal trajectory for sampling-based anytime planning algorithms. Existing algorithms predict the entire optimal path and bias random samples toward it for fast path planning. However, these algorithms may not be suitable for real-time frameworks because the bias-sampling strategy should consider the sequential nature of real-time execution. Therefore, the proposed algorithm predicts a portion of the optimal path, known as the *committed trajectory*, step by step as a probability distribution using a neural network. This distribution is then used in a sampling-based anytime planning algorithm as a non-stationary way of biasing random samples. The proposed algorithm can sequentially plan the near-optimal motion, allowing the vehicle to reach the desired goal pose in a timely and accurate manner. In various test parking scenarios, the proposed algorithm reduces the parking time by approximately 38% compared with conventional motion planning algorithms and by 10% compared with another real-time framework that biases samples toward the entire optimal trajectory.**

I. Introduction

Motion planning has been actively investigated to realize fully autonomous vehicles. The objective of motion planning is to find a feasible path (trajectory) to reach the desired goal pose while considering environmental constraints and the nonholonomic constraint of vehicles. Despite substantial advances, motion planning remains a challenging task. As the focus of the environment shifts from structured to unstructured environments, such as alleyways or parking lots, the presence of variously shaped obstacles increases, thereby requiring a high computational effort for planning. Furthermore, a planned path should be near-optimal to ensure that the vehicle reaches the desired goal pose in a timely and accurate manner.

To solve the motion planning problems, a hierarchical approach, known as ***plan-and-execution***, has been widely studied [1], [2], which involves planning and control in a hierarchical structure. The vehicle plans the optimal path from its initial pose to the desired goal pose and then executes the control to follow it. However, this approach may be inefficient if the planning step takes longer than the control loop (e.g., approximately 20 ms), causing the vehicle to stop and wait. Even allocating more time than the control loop, such as approximately 1 s, may not be sufficient to ensure real-time planning (e.g., under 1 s) because the planning time can vary significantly depending on the situation. Ultimately, the hierarchical approach presents difficulties in achieving real-time planning.



(a) (b) (c)

Fig. 1. Illustrations of the proposed algorithm. In the initial planning phase, (a) the learned anytime *predictor* generates random samples close to the initial near-optimal committed trajectory. (b), (c) As the vehicle tracks the committed trajectory, the next near-optimal committed trajectory is sequentially predicted to quickly optimize the remaining trajectory.

To achieve real-time motion planning for practical use, various sampling-based algorithms [3]–[7] have been studied in an online manner, the so-called anytime framework [4]. In this framework, the vehicle tracks a *portion* (denoted as ***committed trajectory***) of the current best trajectory in a tree within the control loop as an initial path, and continuously builds and optimizes the tree to improve the remaining trajectory, i.e., ***plan-with-execution***. This framework has the advantage of not requiring the vehicle to stop until the whole near-optimal path is found. Despite this advantage, implementing it in cluttered environments remains a challenge. Due to environmental constraints, the initial path computed during the control loop may be far from the near-optimal path. Accordingly, the vehicle inevitably tracks a local-minima path that is far from the portion of the optimal path. As a result, the vehicle takes a long time to reach the desired goal pose, which can be inefficient and may negate the advantage of the anytime framework.

To handle such environments, sampling-based algorithms can be incorporated with bias sampling. A representative approach involves predicting the whole optimal path and biasing random samples toward it [8]–[10]. However, this approach can be inefficient when applied to the anytime framework. As the framework sequentially plans the committed trajectory that the vehicle is about to track, a subsequent part of the whole path may have lower priority for sampling. Consequently, despite the use of bias sampling, the anytime framework may fail to find an initial portion (*committed trajectory*) of the optimal path. Even if the remaining path becomes close to optimal, it

*This work was supported by Samsung Electronics Co., Ltd. under Grant 0418-20210090. (*Corresponding author: Jaeheung Park)*

[1]M. Kim, J. Ahn, and J. Park are with the Department of Intelligence and Information, Graduate School of Convergence Science and Technology, Seoul National University, Seoul, South Korea ({msk930512, joonwooahn, park73}@snu.ac.kr).

[2]S. Shin is with Samsung Advanced Institute Of Technology, Samsung Electronics, Suwon, South Korea.

[3]J. Park is also with ASRI, RICS, Seoul National University, Seoul, South Korea, and the Advanced Institutes of Convergence Technology (AICT) Suwon, South Korea (park73@snu.ac.kr)

can still take a considerable amount of time to reach the goal in such environments.

To alleviate this limitation, this study proposes an anytime *predictor* that estimates the optimal committed trajectory for the anytime framework. The learned *predictor* infers the optimal trajectory within the framework step by step, resulting in finding the initial near-optimal committed trajectory within the control loop and sequentially predicting and optimizing the next committed trajectory in advance (see Fig. 1). The main contribution of this work is summarized as follows.

- The proposed anytime *predictor* learns to infer the optimal committed trajectory using a data-driven approach. To the best of our knowledge, this is the first work to directly learn the *committed trajectory* for the anytime framework.
- For computational efficiency in the framework, a state-of-the-art model is modified as a lightweight architecture and used as a *predictor*. Furthermore, data augmentation is proposed to efficiently collect various committed trajectory data.
- Experimental results demonstrate that bias sampling to the committed trajectory is more effective in the anytime framework than bias sampling to the whole trajectory.

The remainder of the paper is structured as follows. Related studies are categorized and introduced in Section II. Section III explains the anytime framework and presents the problem formulation. Section IV describes the learning-based anytime framework and the proposed anytime *predictor* in detail. Section V provides the experimental results and discussions. Finally, the conclusion is presented in Section VI.

II. Related Works

To realize real-time motion planning, various algorithms have been explored including model-free planning methods and anytime planning methods, in an online manner.

The dynamic window approach [11] is a classical model-free planning method that directly searches for control velocities under a cost function and executes them in every control loop. However, it suffers from local-minima and exhaustive parameter tuning. Recently, the Monte Carlo Tree Search algorithm has emerged as a commonly used model-free planning method [12]–[14]. This algorithm simulates future action sequences (e.g., steering, acceleration, and deceleration) from the current state and selects the best action to reach the desired goal pose. This process is repeated in every control loop. To execute the optimal action, these algorithms [12]–[14] incorporate neural networks, which guide the simulation and estimate the quality of simulated actions. However, to achieve high performance, multiple neural network computations are required within a single control loop (e.g., 20 ms), which necessitates significant computational resources and can limit practical use. In [15], a neural network directly plans a path to the goal in each control loop, and the vehicle follows it using a model predictive controller. However, such a path may not be kinematically feasible, leading the vehicle to easily deviate from the path.

The online sampling-based anytime planning algorithm is another approach to real-time motion planning, known as the *anytime framework* [3]. This framework operates as a *plan-with-execution* system, where the vehicle tracks a portion (*committed trajectory*) of the current best path step by step, while the remaining path is continuously searched and optimized in the background [4]–[7]. However, to account for not only simple but also cluttered environments, the algorithm requires bias sampling to obtain the near-optimal path in a short time. The CL-RRT [5], [6] algorithm is a representative example of an anytime algorithm that utilizes bias sampling toward the center of the lane or narrow regions. However, depending on the specific scenarios, user-defined sampling rules may detrimentally affect performance.

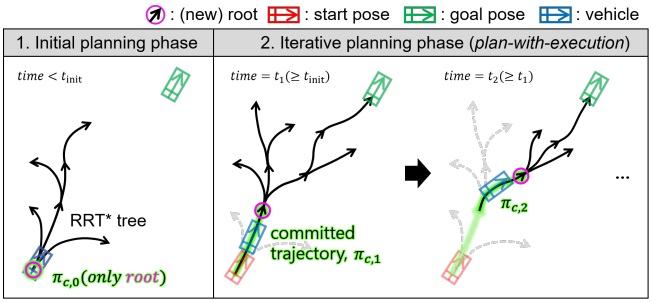


Fig. 2. Description of the anytime framework.

Recent studies [7]–[10] have used data-driven approaches instead of user-defined rules. In [7], informed regions were predicted using neural networks in the anytime framework, and samples were biased toward these regions to accelerate convergence toward the optimal path. However, this sampling strategy can only be applied after obtaining a whole initial path, and this work does not account for nonholonomic constraints. In [9], a neural network estimated the optimal path from the current pose of the vehicle to the goal pose (*whole optimal path*), which was then used to bias samples in a *plan-and-execution* system. This work has demonstrated the fastest initial path planning among other state-of-the-art algorithms. Nevertheless, it may not be suitable for the anytime framework because bias sampling should consider the sequential nature of real-time execution. Thus, in this study, our key idea is to bias samples near the optimal committed trajectory step by step instead of near the whole optimal trajectory.

III. Preliminaries

A. Anytime Framework

The anytime algorithm is defined as a planning algorithm with the property of asymptotic optimality [3]. It quickly plans a feasible but not necessarily optimal path, and gradually improves it toward the optimal path over time, such as the optimal variant RRT, known as RRT*. A real-time implementation of this anytime planning algorithm is called the anytime framework [4]. The anytime framework [3] consists of initial and iterative planning phases (see Fig. 2).

During the initial planning phase, the anytime planning algorithm runs for a user-defined time, known as the initial planning time (t_{init}). It depends on the field of application and is typically set to the control loop [5], [6] as a closed-loop system, or determined by analyzing the planning time in various situations [3]. Once the initial planning time has elapsed, the algorithm returns a portion of the current best

trajectory in the tree, represented as an edge within the tree. This portion (edge) is defined as a *committed trajectory*, π_c, and the end pose of π_c becomes a new root of the current tree. The cost-to-go heuristic [5], [6] of the node or the sum of the path cost from the root to the node and the cost-to-go heuristic (i.e., solution heuristic) [3] is used to select the current best trajectory. The vehicle tracks the committed trajectory while the remaining tree is continuously optimized from the new root. When the vehicle reaches the new root, the next portion (edge) of the current best trajectory is selected as the committed trajectory, and this process is repeated until the vehicle reaches the goal.

B. Problem Definition

The problem addressed in this study is to find the near-optimal committed trajectories in the anytime framework, thereby reducing the time it takes to reach the goal. To clarify this study, the problem is formulated as follows.

In the anytime framework, the resulting trajectory, π, can be defined as,

$$\pi(q_s, q_g) = \bigcup_{q_k = q_s}^{q_g} \pi_{c,k}(par(q_k), q_k). \tag{1}$$

q_s and q_g denote the start and goal poses, respectively. q_k denotes the root of the tree and $par(q_k)$ is its parent node. Thus, $\pi_{c,k}(par(q_k), q_k)$ represents a committed trajectory. Following Eq. (1), the resulting trajectory is a sequence of the committed trajectories. Hence, the problem in the anytime framework is formulated as minimizing the cost function,

$$cost(\pi(q_s, q_g)) = \sum_{q_s}^{q_g} cost(\pi_{c,k}(par(q_k), q_k)). \tag{2}$$

The cost function can be any user-defined function with the following property: $cost(\pi(q_0, q_1)) = cost(\pi(q_0, q_k)) + cost(\pi(q_k, q_1))$ given any node q_k on the trajectory, $\pi(q_0, q_1)$. According to this property, the objective of the anytime framework is to sequentially find a lower-cost committed trajectory.

Finally, the goal of this study is to find a sequence of optimal committed trajectories,

$$\{\pi^*_{c,k}\} = \text{argmin}_{\{\pi_{c,k}\}} \sum_{q_s}^{q_g} cost(\pi_{c,k}). \tag{3}$$

Solving Eq. (3) is difficult as the previous committed trajectory, $\pi_{c,k-1}$, affects the next committed trajectory, $\pi_{c,k}$, because the committed trajectory is sequentially visited. In other words, when the committed trajectory, $\pi_{c,k}$, is far from near-optimal, the resulting path cannot be near-optimal, even if the subsequent committed trajectory, $\pi_{c,k+1}$, is optimized. Thus, this study proposes an anytime *predictor* that infers the optimal committed trajectory, $\tilde{\pi}^*_c$ using a data-driven approach. The *predictor* is used in the anytime framework to sequentially bias samples toward the optimal committed trajectory.

IV. Learning to Generate Committed Trajectory Distribution

A. Overview of Learning-based Anytime Framework

The proposed architecture of the anytime framework is detailed in Fig. 3. The primary contribution of this study is the learned anytime *predictor* specifically designed for predicting the optimal committed trajectory. This is continuously applied to the anytime framework and generates a sampling distribution in a non-stationary way according to the committed trajectory that is sequentially tracked. Accordingly, this learned anytime *predictor* plays a key role in sequential near-optimal motion planning in the anytime framework. The description of the proposed learning-based framework is depicted in Fig. 4.

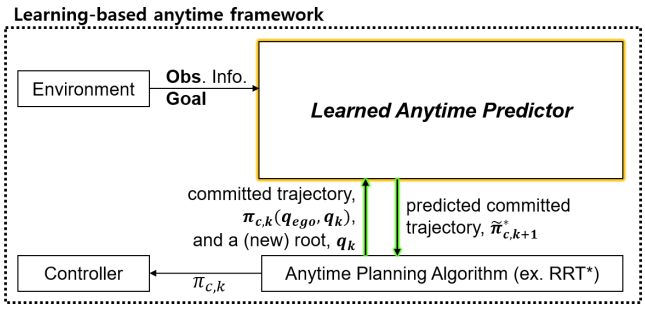


Fig. 3. Architecture of the learning-based anytime framework.

As the initial planning phase begins, the *predictor* infers a probability distribution ($\mu_{c,1}$) of the initial optimal committed trajectory ($\pi^*_{c,1}$) based on inputs: the tree root (q_k), the goal pose (q_g), obstacle information, and a null committed trajectory ($\pi_{c,0}$) that only contains the initial vehicle pose. N_{bias} samples are drawn from this distribution, and N_{uni} uniform samples are generated simultaneously to ensure completeness and asymptotic optimality [3]. With these N samples, the anytime planning algorithm builds the tree. Both N_{bias} and N_{uni} are set to the same value to balance exploration and exploitation. Once the initial planning phase is completed, the current best trajectory is selected based on the solution heuristic [3], and the vehicle tracks the committed trajectory of this best trajectory. In the iterative planning phase, the committed trajectory, $\pi_{c,1}(q_{ego}, q_1)$, currently being tracked and the new root become the next inputs, along with obstacle information and the goal. Then, the distribution for the next committed trajectory, $\pi_{c,2}(q_1, q_2)$, is inferred. This is because the vehicle will track the next committed trajectory; thus, it should be predicted and intensively optimized in advance. Once more, N_{bias} samples and N_{uni} uniform samples are used. This process is repeated in every control loop until the vehicle reaches the goal.

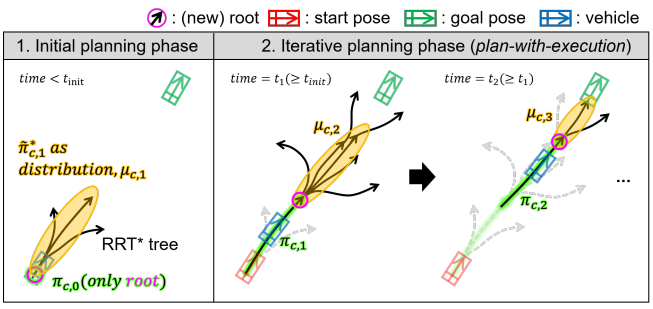


Fig. 4. Description of the learning-based anytime framework.

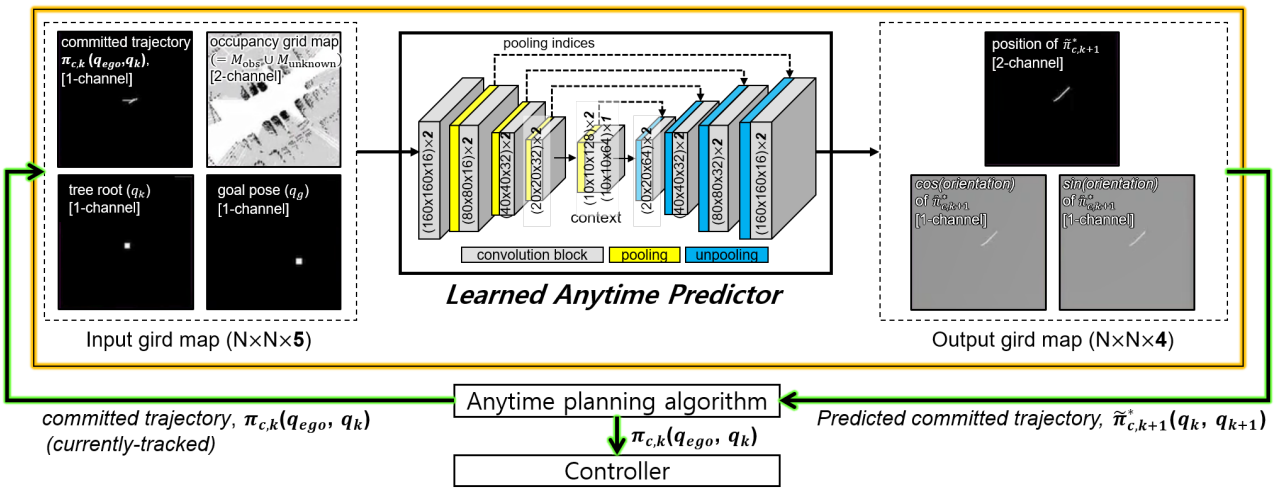


Fig. 5. Description of the network architecture of the anytime *predictor*. Input and output grid maps are examples from the path in Fig. 6(b).

B. Learned Anytime Predictor

The anytime *predictor* is learned by supervised learning. The network model and training strategy are based on a previous study in [9], which exhibits significant path prediction performance. Here, in contrast to the model in [9], this study modifies the model into a lightweight architecture for improved computational efficiency within the anytime framework. In addition, data augmentation is performed to improve the accuracy of committed trajectory prediction by collecting various input-output pairs. The architecture is depicted in Fig. 5.

The input of the network is a 5-channel grid map (M_{in}). Each channel consists of an occupied grid map, an unknown grid map, the root (q_k) of the tree, the goal (q_g), and the committed trajectory ($\pi_{c,k}$) that the vehicle currently tracks. The committed trajectory is represented by a set of relative positions from the pose of the vehicle. The goal and the tree root are relative poses from the vehicle, depicted as a $n \times n$ square containing position and orientation information.

The network output is a 4-channel grid map (M_{out}). The first two channels represent the classification results of whether each grid belongs to the subsequent committed trajectory, $\pi_{c,k+1}$. Thus, applying the softmax function to each grid, the probability that each grid belongs to $\pi_{c,k+1}$ is obtained (the trajectory distribution, $\mu_{c,k+1}$). The other two channels are the regression results. A grid in each channel represents the cosine and sine values of the orientation of the vehicle on the trajectory, respectively. In the sampling step, grid samples are extracted from $\mu_{c,k+1}$ by low variance sampling [16]. Then, each sample's orientation is assigned by calculating $\textbf{atan2}(sin(\theta), cos(\theta))$, corresponding to that grid on the last two channels. Thus, each sample is represented as a pose, $p^{[i]}_{\text{pred}}(x_i, y_i, \theta_i)$, on the predicted committed trajectory.

The network model is described as follows. A convolution block consists of convolution layers with 3 × 3 kernels and 1 stride. A 2 × 2 max pooling layer or 2 × 2 unpooling layer follows each block. All convolution blocks include a batch normalization layer and a ReLU activation function except for the last layer. The letters in the convolution block indicate the shape of feature maps, (WxHxC) × $\boldsymbol{N}$, by $\boldsymbol{N}$ convolution layers. The number of feature map channels in the block is reduced to one-quarter of that of the model in [9]. The total time for predicting the trajectory and extracting 100 samples from its distribution becomes less than 20 ms.

C. Data Gathering with Augmentation

The dataset was collected using the CARLA simulator [17] (see Fig. 6(a)). To address typical motion planning challenges in cluttered environments, this study focuses on the parking problem. First, the target parking spot is randomly determined, and then, the ego vehicle is randomly spawned on the road within a certain distance from this spot. The occupancy of other spots is determined with a probability of 50 %. To include complex situations, two irregularly positioned vehicles are randomly spawned on the road with a probability of 25 %.

Second, a path is planned using the RRT* algorithm, assuming fully-known obstacle information (see Fig. 6(b)). For the RRT* algorithm, after finding an initial path, the path is optimized for 15 s to obtain the near-optimal path, π^*. During this optimization time, rejection sampling [18] is applied, accelerating convergence by rejecting samples that do not improve the cost of the current path. Third, the path, π^*, is partitioned into committed trajectories, denoted as $\pi^*_{c,k}$, using tree nodes as dividing points. For example, in Fig. 6(b), the optimal path is composed of three edges (committed trajectories) of the tree.

Finally, the ego vehicle sequentially tracks these trajectories, and the data sample is collected every 0.5 m. The occupancy grid map (*OGM*) is constructed using a 3D-LiDAR sensor as a 160px × 160px grid map with a resolution of 0.2 m/px. The root and goal are represented as 7px × 7px squares. In this collection step, data augmentation is employed. For instance, in Fig. 6(c), when the vehicle tracks the first trajectory, $\pi^*_{c,1}$, it gathers two pieces of data. Data#1 include the committed trajectory ($\pi^*_{c,1}$), root (q_1), goal (q_g), and the *OGM* as input data, and the label data are the second committed trajectory ($\pi^*_{c,2}$). Data#2 are the augmented data, where the first and second trajectories are combined as the first committed trajectory ($\pi^{*(aug)}_{c,1}$), and the label data are the next committed trajectory

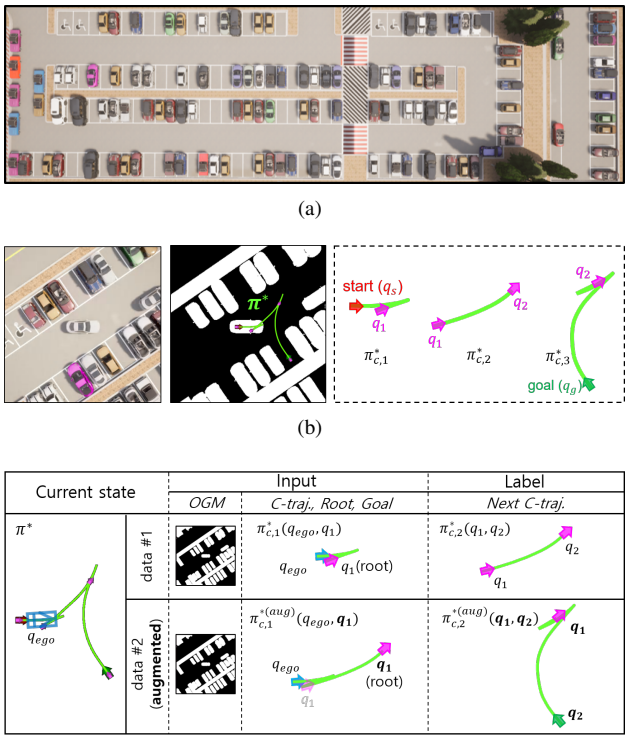


Fig. 6. Illustrations for data gathering. (a) Parking environment in the simulator. (b) Optimal path and its committed trajectories. (c) Example of data gathering with augmentation.

$(\pi^{*(aug)}_{c,2})$, which is the same as the third committed trajectory $(\pi^{*}_{c,3})$. This augmentation improves inference accuracy by providing diverse input-output pairs sharing the same *OGM* and goal but with various roots and committed trajectories.

Data were collected by planning parking paths in 1069 situations, and the proposed augmentation method increased the total number of data points from 25815 to 45551. All data were collected in the form of a 2D grid map, as depicted in Fig. 5. The *predictor* was trained using 85 % of the collected paths as training data, and 15 % of the paths were for test data.

D. Loss Function and Hyperparameters for Training

The network model is trained by optimizing the following loss function,

$$Loss = \sum_{g \in G} (f_{ce}(g)L_{ce}(g) + f_{mse}(g)L_{mse}(g)). \quad (4)$$

The first term is the grid-wise cross-entropy loss for classifying the position of the trajectory on the output grid map (the first two channels). The second term is the grid-wise mean squared error loss for regressing the orientation of the trajectory on the output grid map (the other two channels). $f_\bullet(g)$ denotes a weighting function that assigns weight $w_\bullet$ to grids that are on the committed trajectory and 1 otherwise. This is because most grids exclude trajectory information (see Fig. 5), which can slow down the convergence speed of the neural network. By weighting the loss function for grids corresponding to the committed trajectory, the training process is accelerated [9].

The network parameters are optimized using the Adam optimizer. The learning rate is set to 0.0004. The weight decay is applied to regularize the network, and its scaling factor, λ, is set to 0.01. The network is trained using the Pytorch library with a batch size of 512 for 200 epochs. Both w_{ce}, and w_{mse} are set to 20, respectively, which yields the lowest validation loss.

V. Experiments and Results

A. Experiment Settings For Evaluation

Experiments were conducted in 10 parking scenarios, distributed evenly according to the parking time required using a hierarchical approach to cover various test scenarios (see Fig. 7). As the environment becomes more complex from *S1* to *S10*, more parking time may be required. The proposed algorithm was compared with (**i**) a hierarchical approach and (**ii**) *Anytime-RRT** [3]. The hierarchical approach first plans a near-optimal path using the RRT* algorithm, and the vehicle tracks the path. In the RRT* algorithm, after finding an initial path, 5 s are allocated to converge to the low-cost path. *Anytime-RRT** is the baseline anytime framework (refer to Section. III).

Furthermore, the proposed algorithm, *Learned**C**.Anytime-RRT**, was compared with the previous work [9], using their neural network model instead of the proposed anytime *predictor*. The network model predicts a whole-optimal trajectory in the learning-based anytime framework (refer to Section. IV). This framework is denoted as *Learned**W**.Anytime-RRT**. It is for analyzing the difference between directly minimizing Eq. (2) by biasing samples to the whole trajectory, $\tilde{\pi}^*(q_{ego}, q_g)$ and sequentially minimizing Eq. (2) by biasing samples to the committed trajectory, $\tilde{\pi}^*_{c,k}$.

All algorithms were executed 10 times in each scenario, and the means of the parking time and the trajectory length (L_p) of the vehicle were measured. The parking time was defined as the time from when the algorithm started planning to when parking was completed. Furthermore, the path deviation for the anytime frameworks was calculated using

$$D^{\mathbb{1}}_c(\pi^*_{c,1}, \pi_{\text{pred}}) = \frac{\sum_{i=1}^{N} \min_{p_t \in \pi^*_{c,1}} d(p_t, p^{[i]}_{\text{pred}})}{N}. \quad (5)$$

Eq. (5) is the average difference between the initial optimal committed trajectory, $\pi^*_{c,1}$, and N samples ($p^{[i]}_{\text{pred}} \in \pi_{\text{pred}}$) from the anytime *predictor* in the initial planning phase. $\pi^*_{c,1}$ was the first portion (edge) of the optimal path (refer to Fig. 6(b)). $d(p_t, p^{[i]}_{\text{pred}})$ was defined as $w_{\text{pos}}\|\mathbf{x}_t - \mathbf{x}^{[i]}_{\text{pred}}\|_2 + w_{\text{ori}}|\theta_t - \theta^{[i]}_{\text{pred}}|$. w_{pos} and w_{ori} were set to 0.35 and 0.65, respectively, to consider different units of the two terms.

The RRT* algorithm was used as the anytime planning algorithm, and its steer function was the hybrid curvature [19]. The maximum curvature and its rate were set to 1/6.0 m^{-1} and 0.15 m^{-2}. The collision checking area was represented as a set of white circles, covering a full-size vehicle (5.255 m $\times$ 1.899 m) (see Fig. 8). Collision checking was performed using the obstacle grids (pink-colored grids). The steering controller was a Kanayama controller [20]. The reference velocity was set to 3.6 km/h and adjusted based on the curvature difference between the vehicle and the path, cusps, and goal [2]. The control loop was set to 20 ms. The initial planning time (t_{init}) for the anytime frameworks was set to the control loop.

Fig. 7. Test parking scenarios in the CARLA simulator for the experiments.

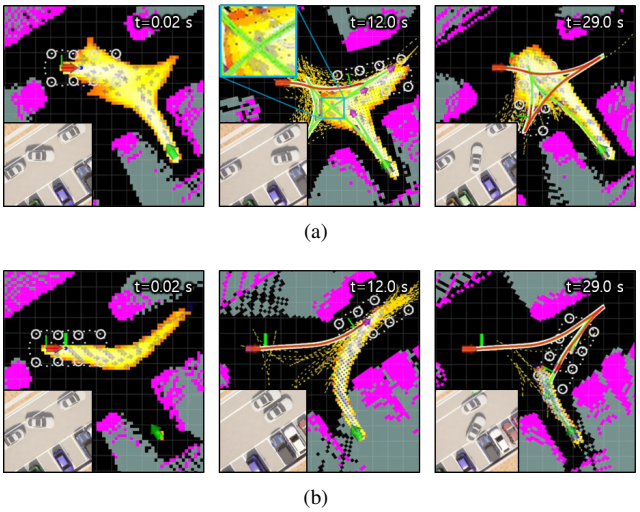


Fig. 8. Illustrations of the experiment results in ***S6***. (a) *Learned**W**.Anytime-RRT**. (b) *Learned**C**.Anytime-RRT**. The arrows on distributions indicate random samples, and the yellow lines indicate the tree. The green and red trajectories are the parking path and the trajectory of the vehicle, respectively.

B. Results and Discussions

The experimental results are shown in Tables I and II. As shown in the tables, the proposed algorithm yielded the minimum-length trajectory, and the vehicle reached the desired goal pose in the shortest time in most scenarios.

Compared with the hierarchical approach, the anytime frameworks eliminate the need for the vehicle to stop for the planning process (see the parking time (*plan*) in the Tables). Even if the planning time is excluded from the comparison, the *execution* time of *Learned**C**.Anytime-RRT** is similar to or lower than that of the hierarchical approach. In other words, *Learned**C**.Anytime-RRT** performed near-optimal motion in real-time, allowing the vehicle to track the shortest trajectory in most scenarios. The proposed *predictor* achieved near-optimal motion planning with a higher probability than *Anytime-RRT**. The parking time was reduced by an amount ranging between 10 s and 80 s. These results demonstrate that an appropriate predicted trajectory significantly improves the performance of the anytime framework.

In the relatively complex scenarios (*S6-S10*), it has been shown that predicting the committed trajectory (*Learned**C**.Anytime-RRT**) is more suitable for the anytime framework than predicting the whole trajectory (*Learned**W**.Anytime-RRT**). This is due to the sequential nature of the anytime framework. Even if the whole trajectory is accurately predicted, most of the regions in a distribution may be useless. It may be more efficient to concentrate on sequentially predicting and optimizing the committed trajectory that the vehicle is about to track (see Fig. 8(b)). For instance, a sample near the parking spot may be of lower priority in the initial planning phase (see the leftmost figure of Fig. 8(a)). This can be seen from the fact that *Learned**C**.Anytime-RRT** has a lower $D_c^{\mathbb{1}}$ than *Learned**W**.Anytime-RRT**. This means that more useful samples for the initial planning phase are extracted. Therefore, the proposed algorithm allows the vehicle to track a shorter trajectory (L_p) than other anytime frameworks, even if they optimize the remaining trajectory equally close to optimal.

Furthermore, in *Learned**W**.Anytime-RRT**, the prediction performance degraded when frequent forward/backward direction switches were required. As illustrated in the (cyan-colored) middle figure of Fig. 8(a), when a path includes an intersected region, there can be different desired orientations on the same grid, making it difficult to accurately predict the entire path in a *2D* space. In contrast, this problem rarely occurs when predicting committed trajectories because the path is divided and sequentially predicted. This advantage was particularly observed in ***S6***, ***S9***, and ***S10***, where the parking time improved by a range of at least 10 s to a maximum of 30 s.

VI. Conclusions

This study introduced a real-time motion planning framework for autonomous vehicles. The primary contribution is the proposed learned anytime *predictor*, which directly infers the optimal committed trajectory for the anytime framework. Furthermore, a lightweight network is used as a *predictor* to reduce the computation time, and a novel data augmentation method is applied to efficiently collect various committed trajectory data. The experiments in the parking problem showed that this idea is found to be more suitable for the anytime framework by biasing samples to a near-optimal committed trajectory than to the

TABLE I

EXPERIMENTAL RESULTS IN SCENARIOS, ***S1*** TO ***S5***.

S	Planning Algorithms	parking time [s] (*plan* + *execution*)	L_p[m]	$D_c^{\mathbb{1}}$[-]
1	Hierarchical approach	50.3 (*8.3* + *42.0*)	15.5	-
	*Anytime-RRT**	60.2 (***0.0*** + *60.2*)	24.1	4.38
	*Learned**W**.Anytime-RRT**	42.2 (***0.0*** + *42.2*)	15.9	0.66
	LearnedC.Anytime-RRT*	**40.7** (***0.0*** + ***40.7***)	**15.1**	**0.62**
2	Hierarchical approach	65.3 (*21.6* + *43.7*)	16.4	-
	*Anytime-RRT**	60.9 (***0.0*** + *60.9*)	25.7	3.96
	*Learned**W**.Anytime-RRT**	**41.1** (***0.0*** + ***41.1***)	**14.9**	0.61
	LearnedC.Anytime-RRT*	44.4 (***0.0*** + *44.4*)	16.4	**0.47**
3	Hierarchical approach	67.0 (*15.9* + *51.1*)	17.5	-
	*Anytime-RRT**	64.6 (***0.0*** + *64.6*)	22.3	4.32
	*Learned**W**.Anytime-RRT**	51.0 (***0.0*** + *51.0*)	**16.6**	1.02
	LearnedC.Anytime-RRT*	**50.6** (***0.0*** + ***50.6***)	16.8	**0.44**
4	Hierarchical approach	86.1 (*32.9* + *53.2*)	18.3	-
	*Anytime-RRT**	77.4 (***0.0*** + *77.4*)	31.2	4.50
	*Learned**W**.Anytime-RRT**	52.5 (***0.0*** + *52.5*)	17.7	1.07
	LearnedC.Anytime-RRT*	**52.1** (***0.0*** + ***52.1***)	**17.2**	**1.02**
5	Hierarchical approach	94.4 (*29.3* + *65.1*)	22.4	-
	*Anytime-RRT**	74.3 (***0.0*** + *74.3*)	30.1	4.20
	*Learned**W**.Anytime-RRT**	56.0 (***0.0*** + *56.0*)	21.6	0.95
	LearnedC.Anytime-RRT*	**54.7** (***0.0*** + ***54.7***)	**21.2**	**0.24**

**Note*. Video Link: `https://youtu.be/SjsawjhbQvQ`

TABLE II

EXPERIMENTAL RESULTS IN SCENARIOS, ***S6*** TO ***S10***.

S	Planning Algorithms	parking time [s] (*plan* + *execution*)	L_p[m]	$D_c^{\mathbb{1}}$[-]
6	Hierarchical approach	108.2 (*11.5* + *96.7*)	33.3	-
	*Anytime-RRT**	99.2 (***0.0*** + *99.2*)	36.4	4.08
	*Learned**W**.Anytime-RRT**	90.8 (***0.0*** + *90.8*)	32.2	1.23
	LearnedC.Anytime-RRT*	**80.1** (***0.0*** + ***80.1***)	**28.2**	**0.19**
7	Hierarchical approach	108.6 (*9.8* + *98.8*)	33.6	-
	*Anytime-RRT**	104.6 (***0.0*** + *104.6*)	45.6	4.09
	*Learned**W**.Anytime-RRT**	81.0 (***0.0*** + *81.0*)	34.6	1.55
	LearnedC.Anytime-RRT*	**75.3** (***0.0*** + ***75.3***)	**28.9**	**0.40**
8	Hierarchical approach	113.6 (*26.6* + *87.0*)	33.2	-
	*Anytime-RRT**	101.1 (***0.0*** + *101.1*)	43.9	4.66
	*Learned**W**.Anytime-RRT**	**56.3** (***0.0*** + ***56.3***)	24.5	1.69
	LearnedC.Anytime-RRT*	59.7 (***0.0*** + *59.7*)	**24.4**	**0.91**
9	Hierarchical approach	165.0 (*33.0* + *132.0*)	44.7	-
	*Anytime-RRT**	129.4 (***0.0*** + *129.4*)	53.6	4.36
	*Learned**W**.Anytime-RRT**	121.3 (***0.0*** + *121.3*)	39.9	1.71
	LearnedC.Anytime-RRT*	**93.0** (***0.0*** + ***93.0***)	**33.7**	**0.79**
10	Hierarchical approach	166.8 (*51.8* + *115.0*)	36.5	-
	*Anytime-RRT**	166.9 (***0.0*** + *166.9*)	58.3	4.23
	*Learned**W**.Anytime-RRT**	106.4 (***0.0*** + *106.4*)	36.6	1.90
	LearnedC.Anytime-RRT*	**80.6** (***0.0*** + ***80.6***)	**26.8**	**0.32**

**Note*. Video Link: `https://youtu.be/Rg8AK8eUNkc`

entire trajectory. The proposed algorithm planned near-optimal motion more reliably in the initial planning phase and reduced the time required to reach the desired goal pose by sequentially optimizing the next committed trajectory in advance.

As an improved version of the anytime framework, the proposed algorithm can be extended to other mobile-based robotic systems. As the anytime framework has the advantage of quickly replanning infeasible paths by reusing the tree structure, the proposed algorithm will be tested in dynamic environments in future work. Furthermore, we plan to test the proposed algorithm in other cluttered environments with arbitrarily shaped obstacles and conduct real-vehicle tests.

REFERENCES

[1] S. Shin, J. Ahn, and J. Park, "Desired orientation rrt (do-rrt) for autonomous vehicle in narrow cluttered spaces," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4736–4741.

[2] M. Kim, J. Ahn, and J. Park, "Targettree-rrt*: Continuous-curvature path planning algorithm for autonomous parking in complex environments," *IEEE Transactions on Automation Science and Engineering*, 2022.

[3] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.

[4] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki, "Anytime solution optimization for sampling-based motion planning," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 5068–5074.

[5] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on control systems technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[6] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4991–4996.

[7] Y. Chen, Z. He, and S. Li, "Horizon-based lazy optimal rrt for fast, efficient replanning in dynamic environment," *Autonomous Robots*, vol. 43, no. 8, pp. 2271–2292, 2019.

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

[9] H. Banzhaf, P. Sanzenbacher, U. Baumann, and J. M. Zöllner, "Learning to predict ego-vehicle poses for sampling-based nonholonomic motion planning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1053–1060, 2019.

[10] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural rrt*: Learning-based optimal path planning," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 1748–1758, 2020.

[11] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[12] L. Lei, R. Luo, R. Zheng, J. Wang, J. Zhang, C. Qiu, L. Ma, L. Jin, P. Zhang, and J. Chen, "Kb-tree: Learnable and continuous monte-carlo tree search for autonomous driving planning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 4493–4500.

[13] S. Song, H. Chen, H. Sun, M. Liu, and T. Xia, "Time-optimized online planning for parallel parking with nonlinear optimization and improved monte carlo tree search," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2226–2233, 2022.

[14] P. Cai and D. Hsu, "Closing the planning–learning loop with application to autonomous driving," *IEEE Transactions on Robotics*, 2022.

[15] J. J. Johnson, L. Li, F. Liu, A. H. Qureshi, and M. C. Yip, "Dynamically constrained motion planning networks for non-holonomic robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6937–6943.

[16] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[18] C. P. Robert, G. Casella, and G. Casella, *Monte Carlo statistical methods*. Springer, 1999, vol. 2.

[19] E. Heiden, L. Palmieri, L. Bruns, K. O. Arras, G. S. Sukhatme, and S. Koenig, "Bench-mr: A motion planning benchmark for wheeled mobile robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4536–4543, 2021.

[20] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 384–389.