IEEE Transactions on Automation Science and Engineering (T-ASE) paper, presented at ICRA 2026, Vienna, Austria.
Cite as T-ASE paper.

Robust Real-Time Sampling-Based Motion Planner for Autonomous Vehicles in Narrow Environments

Minsoo Kim, Arthur Esquerre-Pourtère, and Jaeheung Park, *Senior Member, IEEE*

***Abstract*—Real-time sampling-based planners increasingly use learned sampling distributions for faster planning in autonomous vehicles. These planners employ a neural network to predict the optimal path and bias some samples toward the path. However, inherent prediction inaccuracies of the network often lead to suboptimal paths, especially in narrow spaces. Learned samples should be used carefully based on accuracy, as inaccurate samples can degrade planning performance. To address this problem, this paper proposes *Learned Adaptive Anytime TargetTree-RRT* (LA3T*)* algorithm. The proposed planner introduces the adaptive biasing ratio. The approach learns to assess the reliability of the learned distribution using the network's confidence. This confidence approximates a proper ratio of learned samples used, thereby adaptively maximizing planning performance while considering a level of prediction accuracy. Furthermore, the LA3T* algorithm incorporates the target tree algorithm. The goal pose is replaced with a set (target tree) of pre-defined optimal path segments, reducing computational efforts in narrow regions. Experiments in various driving tasks explore the benefits of each component through ablation studies. The proposed algorithm significantly increases the success rate and reduces the path length in simulated and real-world scenarios compared to other sampling-based methods.**

***Note to Practitioners*—This work was motivated by the need to develop a practical real-time motion planning algorithm for autonomous vehicles in narrow environments. Although learning-based sampling methods have been actively explored, their planning performance often degrades due to inaccurate predictions from the learned model, limiting their practicability and robustness. The *Learned Adaptive Anytime TargetTree-RRT** (LA3T*) algorithm addresses this problem by adaptively leveraging the learned model based on its reliability and integrating it with the target tree algorithm. Experiments conducted in various driving and parking scenarios demonstrate its practicality and robustness, notably improving success rates in worst-case situations and reducing path length. Furthermore, experiments in sensor noise and dynamic environments—particularly when combined with existing methods—highlight the algorithm's extensibility. LA3T* can be applied to various autonomous systems needing efficient, real-time motion planning in narrow spaces, such as other autonomous vehicles or robots. The algorithm has not yet been investigated in highly out-of-distribution scenarios. Future work will focus on improving performance in such scenarios and better handling more dynamic environments.**

***Index Terms*—Autonomous vehicles, real-time sampling-based planners, learning sampling distribution.**

I. Introduction

MOTION planning serves as a fundamental element in realizing autonomous vehicles. It involves planning a feasible and nearly optimal path from a start to a goal pose in various arbitrary environments. Despite significant progress, nonholonomic motion planning remains challenging. In simple scenarios like highway driving, locally following the lane is often sufficient. However, as the focus shifts to semi-structured or unstructured environments such as driving at alleyways, roads under construction, or parking lots, these narrow and cluttered spaces demand more computational effort for planning [1]. Due to constraints on vehicles' lateral velocity, a slight change in obstacle configurations, their turning radius, or steering speed often results in longer or more detoured paths.

Various planning algorithms have been developed for vehicle motion planning. Most algorithms organize planning and control in a hierarchical structure. First, they plan the optimal complete path. Then, the vehicle executes controls to follow this path. However, this approach intrinsically lacks real-time capability; if the planning time exceeds the control loop (e.g., 50 ms), the vehicle must stop and wait. This situation can easily occur, as narrow or cluttered spaces require a longer path than wide spaces and more frequent forward/backward direction switches, which demand significant planning time. Furthermore, waiting for a complete path can be inefficient, as the path may become infeasible due to varying or unseen obstacles, necessitating a complete re-planning.

To address real-time motion planning, various online sampling-based algorithms have been studied. These algorithms, known as the *anytime framework* [2], incrementally build a tree using random samples within the control loop. The vehicle then follows a segment of the best path in the tree, while the algorithm continuously optimizes the remaining path. The algorithms maintain a real-time capability and are intrinsically capable of handling unseen obstacles by updating

Received 12 August 2024; revised 22 January 2025 and 3 April 2025; accepted 19 May 2025. Date of publication 29 May 2025; date of current version 12 June 2025. This article was recommended for publication by Associate Editor K. Yu and Editor J. Yi upon evaluation of the reviewers' comments. This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grants funded by Korean Government [Ministry of Science and ICT (MSIT)] through the Core Technology Development of Open Simulator for Software-Defined Everything (SDx) Intelligent Application Development under Grant RS-2024-00459435 and through the Artificial Intelligence Graduate School Program at Seoul National University under Grant RS-2021-II211343. *(Corresponding author: Jaeheung Park.)*

Minsoo Kim and Arthur Esquerre-Pourtère are with the Department of Intelligence and Information, Graduate School of Convergence Science and Technology, Seoul National University, Seoul 08826, Republic of Korea (e-mail: msk930512@snu.ac.kr; tutur263@snu.ac.kr).

Jaeheung Park is with the Department of Intelligence and Information, Graduate School of Convergence Science and Technology, and ASRI, AIIS, Seoul National University, Seoul 08826, Republic of Korea, and also with the Advanced Institute of Convergence Technology, Suwon 16229, Republic of Korea (e-mail: park73@snu.ac.kr).

Digital Object Identifier 10.1109/TASE.2025.3574262

the tree with new information. Recent approaches use learned samples to further reduce path length [3], [4], [5], [6], [7]. They predict the optimal path using a neural network as a learned distribution and generate samples. To ensure completeness, these learned samples are employed alongside uniform samples at a specific ratio, known as the *biasing ratio*. Nevertheless, this learning-based framework suffers from inherent prediction inaccuracies of neural networks, leading to the following problems.

Firstly, planning performance is highly affected by the biasing ratio. Leveraging accurately learned distributions enables faster planning and shorter paths; however, a high ratio can cause failures in scenarios where the learned distribution is inaccurate, causing the algorithm to search for irrelevant regions. Prediction accuracy varies across scenarios, and finding the optimal ratio for each scenario is difficult. Although constantly using only a few learned samples might mitigate this, it limits the use of accurate predictions. Extensive data collection may address this issue by enhancing learning accuracy and using many learned samples. However, acquiring comprehensive datasets for various driving scenarios is costly and complex. Some prior studies set a constant ratio, often 50%, to balance exploitation and exploration [3], [6], [7], [8]. However, this constant ratio is not optimal for every test scenario.

Secondly, even well-trained neural networks often degrade planning performance within narrow environments. Tight, cluttered spaces near the goal necessitate precise samples, as tree expansions can easily lead to collisions. Even highly accurate neural networks struggle to generate samples that fully account for vehicle kinematics and other factors [9], requiring impractically large networks or datasets.

This work aims to address such performance degradation caused by the inherent prediction inaccuracies of neural networks, particularly within narrow and cluttered spaces. This paper proposes a novel real-time motion planning framework: the **L**earned **A**daptive Anytime **T**arget**T**ree-RR**T*** (**LA3T***). The LA3T* algorithm introduces the *adaptive biasing ratio* for sampling-based planners. Selecting a proper biasing ratio is formulated as a failure prediction problem. The key idea is to use the reliability of the learned distribution to approximate the optimal biasing ratio, adaptively adjusting it by accounting for the prediction accuracy of learned samples. Furthermore, the proposed algorithm integrates the target tree algorithm [10], [11] within the anytime framework. The goal pose is replaced by a target tree, a set of predefined near-optimal path segments.

The main contributions are summarized as follows.

- The adaptive biasing ratio improves planning performance by adaptively balancing exploitation and exploration in learned sampling-based planners. The proposed idea is straightforward, removes heuristic principles, and is easily applicable to any sampling-based planner.
- Incorporating the target tree algorithm efficiently reduces the computational effort required for planning in narrow goal regions. Additionally, the proposed framework is a novel extension of our previous works, integrating the target tree algorithm [10], [11] and the learning-based anytime framework [7].
- The proposed multi-head anytime predictor enables the anytime framework to successfully leverage the adaptive biasing ratio and target tree algorithm.
- Ablation studies in various simulated driving scenarios explore the effectiveness of each component within the framework. The proposed algorithm significantly increased the success rate and reduced the path length compared to other sampling-based algorithms. Additionally, experiments demonstrate that the LA3T* algorithm can be extended to handle various real-world conditions, such as sensor noises and dynamic environments, by integrating it with complementary modules. Finally, its application to autonomous valet parking tasks in a real vehicle further showcased its real-world applicability.

The remainder of the paper is structured as follows. Related works for real-time motion planning and failure handling are introduced in Section II. Section III describes the backgrounds of the anytime framework, the target tree algorithm, and the problem definition. Section IV explains the proposed motion planning framework in detail. In Section V, the experimental results and discussions are presented. Finally, the conclusion and the future work are presented in Section VI.

II. Related Works

A. Real-Time Motion Planning

Various types of real-time motion planning methods have been studied. Optimization-based methods have been applied to autonomous vehicles or drones by formulating real-time motion planning problems as optimal control problems (OCP). They continuously find optimal trajectories in every control loop and execute the control commands. To find the trajectory in real-time, these methods need to convexify not only the optimization problem but also environmental constraints, such as obstacles. One disadvantage is that this convexification can take a long time in cluttered spaces [4]. Furthermore, they often require good warm-starts or a coarse trajectory in advance to quickly solve the OCP [12], [13], [14], [15].

Learning-based methods use a neural network to replace planning and control modules. However, they require a massive dataset [16], [17], [18] to model the vehicle kinematics from scratch. Another method is the Monte-Carlo Tree Search (MCTS)-based planner. In each control loop, the MCTS-based planner simulates future actions (e.g., steering, acceleration, deceleration) and executes the best action to reach the goal [19], [20], [21]. To select the near-optimal action, those methods use the networks to guide the simulation and evaluate actions. However, substantial computational resources are required because high performance involves multiple neural network computations within a single control loop (e.g., 50 ms), which may limit practical applications. Additionally, using the kinematic model during the simulation can lead to sub-optimal actions due to the discrepancy from the actual model.

Real-time sampling-based methods, also known as the *anytime framework* [7], [22], [23], [24], operate as a *plan-with-execution* system. The robot tracks a segment (*committed path*) of the current best path while continuously searching

for and optimizing the remaining path [5], [7], [22], [23], [24], [25], [26]. Most studies employ a biasing sampling strategy to quickly find the optimal path. Closed-loop RRT [24], [25] uses biased sampling toward the center of the lane or narrow regions. RT-RRT* [26] employs rejection sampling to accelerate optimal path planning and continuously repairs a tree to handle unseen or dynamic obstacles. Learned Anytime-RRT* (LAT*) [7] uses a neural network to sequentially estimate the optimal path and bias samples toward it. The main disadvantage of these approaches is their heavy reliance on biased sampling, which must be carefully managed to enhance planning performance.

Choosing the best approach among these methods is challenging due to their respective pros and cons. Nevertheless, this study uses a sampling-based method as a baseline method due to its ease of implementation, relatively low computation, and ability to quickly replan the path by reusing the tree.

B. Handling Failures in Sampling-Based Planners

Few discussions addressed handling failures in sampling-based planners. Ichter et al. [3] conducted an in-depth analysis of the biasing ratio, demonstrating through case studies that using 50% learned and 50% uniform samples is generally effective. However, this constant ratio is not universally optimal and may fail to utilize the benefits of a learned distribution, sometimes even degrading planning performance. Gaebert and Thomas [27] proposed a fallback ratio to mitigate prediction inaccuracies. It begins with a high ratio, such as 90%, to leverage a well-learned distribution, then decreases to a lower value, around 10%, to handle possible errors in prediction. Nonetheless, this method still relies on heuristics for setting and adjusting ratios. Also, scenarios with precise distributions could benefit from a consistently higher ratio.

Some works handle failures by estimating model failures. Hecker et al. [28] predicted the model failures by assessing the complexity of a scene in autonomous driving. Similarly, Mohseni et al. [29] proposed estimating the model's prediction loss, anticipating failures during testing. Nevertheless, neither method discussed handling these failures in a motion planning context. Power and Berenson [30] evaluated how different a given input scenario is from the training dataset and adjusted the input to resemble the closest training scenario. Thus, the model is induced to hallucinate and accurately predict the learned distribution, even for scenarios not included in training. However, it may struggle with scenarios that are highly different from the training data. Guzzi et al. [31] estimated the probability of success for each sample and rejected those with lower probabilities. However, this approach can require many iterations to approximate the desired distribution closely.

III. Preliminaries

A. Anytime Framework

An anytime algorithm is defined as a planning algorithm characterized by asymptotic optimality [32], such as the optimal rapidly-exploring random tree (RRT*) [33]. It initially plans a feasible path, which may not be optimal, and incrementally improves it over time toward the optimal path. A real-time implementation of such algorithms is known as the anytime framework [7]. It comprises an initial and an iterative planning phase, as shown in Fig. 1.

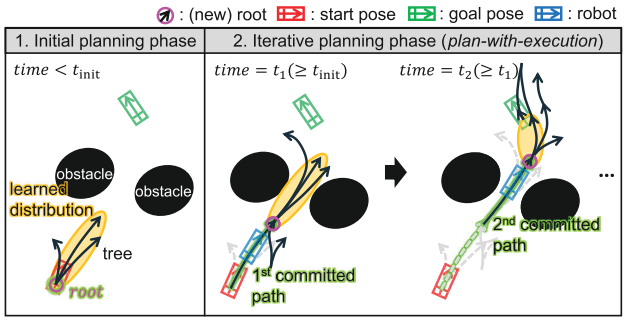


Fig. 1. Illustrations of the learning-based anytime framework [7].

1) Initial Phase: The anytime algorithm runs for a user-defined initial planning time, t_{init}. This duration is either set to the control loop [7], [24], [25] or determined by analyzing planning times in various scenarios [5]. After the initial planning time, the framework returns a segment of the current best path; an edge in the tree. This segment is defined as a *committed path* that the robot is about to follow. The best path is selected by a heuristic cost, such as the Euclidian distance from the node to the goal. If the algorithm finds a complete path to the goal, the first path segment (edge) on the complete path is selected as the committed path. The root is then changed to the end of this committed path.

2) Iterative Phase: The robot follows the committed path while the remaining tree is continuously optimized from the new root. When the robot arrives at the new root, the next segment (edge) of the current best path is selected as the next committed path based on heuristics. This process iteratively runs until the robot reaches the goal. Note that any sampling distribution can be used, including learned distributions. As shown in Fig. 1, the optimal path segments are sequentially predicted using the learned sampling distribution.

B. Problem Definition

This study Reference our previous work [7] to derive the problem definition. In the anytime framework, the resulting path, π, can be described as follows:

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

Here, the start and goal poses are denoted by q_s and q_g, respectively. q_k represents the node on the path, with $par(q_k)$ representing its parent node. $\pi_{c,k}(par(q_k), q_k)$ represents a committed path that a robot followed. Eq. (1) represents that the resulting path consists of a series of committed paths. Thus, the objective is to minimize the cost function until the robot reaches the goal, defined as:

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

Any user-defined cost function can be applied here if it satisfies the following property: $cost(\pi(q_0, q_1)) = cost(\pi(q_0, q_i)) +$

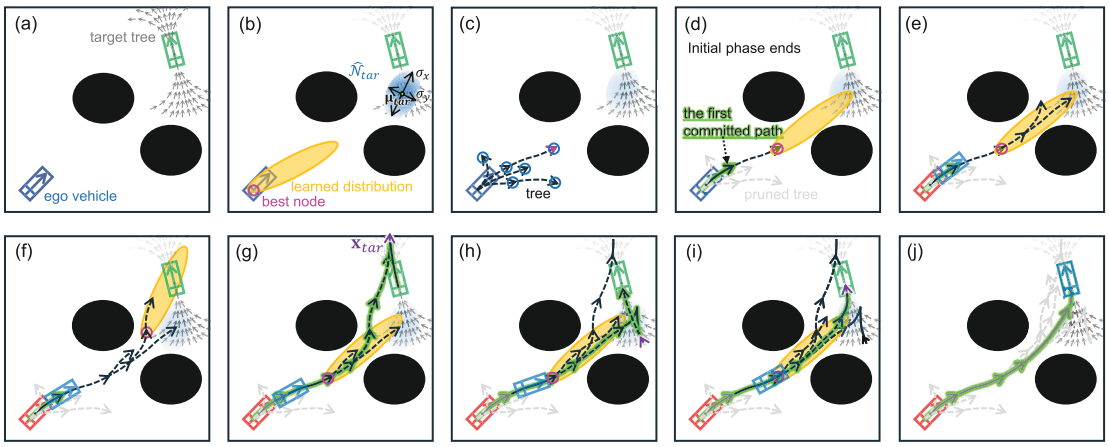


Fig. 2. Illustrations of learned adaptive anytime TargetTree-RRT*. The top-left alphabet represents the order of the illustrations.

$cost(\pi(q_i, q_1))$ for any node q_i on the path, $\pi(q_0, q_1)$. In this study, the path length is used as the cost function. Given Eqs. (1) and (2), the problem is defined as finding a sequence of optimal committed paths given arbitrary planning problems.

C. TargetTree-RRT* Algorithm

TargetTree-RRT* [10] is the anytime and complete planning algorithm for narrow environments. This approach replaces a goal pose with a set (*target tree*) of several candidate goals, positioned along predefined feasible paths close to the optimal path, as shown in Fig. 2(a). Following [10], the target tree is initialized to cover wide regions, considering obstacle configurations. In the planning step, the algorithm runs similarly to standard RRT*, but with different criteria for finding a path. A path is obtained when an RRT tree, $\mathcal{T}_{RRT}$, meets one of the candidate goals. When the planning step finishes, the lowest cost path among the obtained paths is returned, and its corresponding candidate goal is defined as the optimal target goal, $\mathbf{x}^*_{tar}$.

IV. Methodology

A. Real-Time Motion Planner: Learned Adaptive Anytime TargetTree-RRT* (LA3T*) Algorithm

The proposed framework extends the Learned Anytime-RRT* (LAT*) [7] by introducing the adaptive biasing ratio and incorporating the TargetTree-RRT* algorithm [10], [11]. The framework is depicted in Fig. 2, and Algorithm 1.

The proposed anytime framework receives obstacle information, a goal pose ($\mathbf{x}_{goal}$), an initial pose ($\mathbf{x}_{init}$), and parameters for the adaptive ratio and target tree. When planning starts, $\mathbf{x}_{goal}$ is replaced by a target tree, $\mathcal{T}_{target}$ (see Fig. 2(a)). The RRT tree ($\mathcal{T}_{RRT}$) is initialized with the initial pose as its root.

1) Initial Phase: Given the obstacle configurations, the root of $\mathcal{T}_{RRT}$, and the goal pose, the neural network outputs the learned distribution, $\hat{P}_{seg+1}$, for the first optimal path segment, the reliability of the distribution, and the biased range for $\mathcal{T}_{target}$. The biased range, $\hat{\mathcal{N}}_{tar}$, indicates where the optimal target goal, $\mathbf{x}^*_{tar}$, lies. The algorithm extends $\mathcal{T}_{RRT}$ using these outputs. The detailed sampling function will be provided in the next subsection. In this work, the initial planning time is set to

Algorithm 1 Learned Adaptive Anytime TargetTree-RRT* (**LA3T***)

Input: $\mathcal{X}_{free}, \mathbf{x}_{init}, \mathbf{x}_{goal}, \lambda_{\max}, \tau$

$\mathcal{T}_{target} \leftarrow$ InitializeTargetTree($\mathbf{x}_{goal}$)
$\mathcal{T}_{RRT}$.push_back($\mathbf{x}_{init}$)
while not GoalReachedByRobot() **do**
  UpdateObstacle()
  $\pi_{best}, \mathbf{x}_{best} \leftarrow$ GetBestPath&Node($\mathcal{T}_{RRT}$)
  $\hat{P}_{seg+1}, c, \hat{\mathcal{N}}_{tar} = f_{NN}(\mathcal{X}_{free}, \mathbf{x}_{goal}, \pi_{best}, \mathbf{x}_{best})$
  $\mathbf{x}_{rand} \leftarrow$ AdaptiveSampling($\hat{P}_{seg+1}$, c, $\lambda_{\max}$, τ, $\hat{\mathcal{N}}_{tar}$, $\mathcal{T}_{target}$)
  Extend($\mathbf{x}_{rand}$, $\mathcal{T}_{RRT}$, $\mathcal{T}_{target}$) // TargetTree-RRT*
  if InitPhaseDone **or** RobotReachRoot($\mathcal{T}_{RRT}$) **then**
    SelectCommittedPath($\mathcal{T}_{RRT}$)
    UpdateRootWithPruning($\mathcal{T}_{RRT}$)
  end if
end while

the control loop. When the initial planning time elapses, the best node is selected by a heuristic function, and the vehicle follows the first committed path, as shown in Figs. 2(c) and (d). The root is then updated to the end of this path.

This study uses the following heuristic function, motivated by the Hybrid-A* algorithm [34]:

$$\mathcal{H}(\mathbf{x}) = \max(\mathcal{H}_{non}(\mathbf{x}), \mathcal{H}_{hol}(\mathbf{x})), \qquad \forall \mathbf{x} \in \mathcal{T}_{RRT}. \tag{3}$$

$\mathcal{H}_{non}(\mathbf{x})$ is *nonholonomic-without-obstacles* heuristic that ignores obstacles while considering the nonholonomic constraints. $\mathcal{H}_{non}(\mathbf{x})$ is evaluated by the length of the hybrid curvature curve [35] between a given node $\mathbf{x}$ and the goal pose, assuming obstacle-free. $\mathcal{H}_{hol}(\mathbf{x})$ represents *holonomic-with-obstacles* heuristic. It ignores the nonholonomic constraints and is evaluated by the 2D path length generated by the A* algorithm from $\mathbf{x}$ to the goal, $\mathbf{x}_{goal}$, considering obstacles.

2) Iterative Phase: The framework continuously extends the RRT tree and searches for a path in the iterative phase, as shown in Figs. 2(e-i), while the vehicle tracks the path. The algorithm runs differently depending on whether a complete path is found. This operation is handled by Algorithm 2.

Algorithm 2 GetBestPath&Node()

Input: $\mathcal{T}_{RRT}$

1: **if** NotFoundCompletePath() **then**
2: $\quad \mathbf{x}_{best} \leftarrow$ GetBestNodeUsingHeuristic($\mathcal{T}_{RRT}$)
3: $\quad \pi_{best} \leftarrow$ GeneratePath($\mathbf{x}_{best}$)
4: $\quad$ **return:** $\pi_{best}, \mathbf{x}_{best}$
5: **else** {// Found a complete path}
6: $\quad \pi_c \leftarrow$ GetCommittedPath($\mathcal{T}_{RRT}$)
7: $\quad \mathbf{x}_{end} \leftarrow$ GetEndStateFromPath(π_c)
8: $\quad$ **return:** π_c, $\mathbf{x}_{end}$
9: **end if**

Algorithm 3 AdaptiveSampling()

Input: $\hat{P}_{seg+1}$, c, $\lambda_{\max}$, τ, $\hat{\mathcal{N}}_{tar}$, $\mathcal{T}_{target}$

1: $\lambda(c) = \min(\lambda_{\max}, c)$ // adaptive biasing ratio
2: coin $\leftarrow$ Rand() // $\in [0, 1]$
3: **if** coin <= τ **then**
4: $\quad \hat{P}_{TargetTree} \leftarrow$ GenerateBiasProb($\mathcal{T}_{target}$, $\hat{\mathcal{N}}_{tar}$)
5: $\quad \mathbf{x}_{rand} \leftarrow$ SampleFrom($\hat{P}_{TargetTree}$)
6: **else if** coin < $\tau + (1-\tau)\lambda(c)$ **then**
7: $\quad \mathbf{x}_{rand} \leftarrow$ SampleFrom($\hat{P}_{seg+1}$)
8: **else**
9: $\quad \mathbf{x}_{rand} \leftarrow$ UniformSampling()
10: **end if**
11: **return:** $\mathbf{x}_{rand}$

If there is no complete path, **LA3T*** focuses on finding a near-optimal complete path as quickly as possible by sequentially biasing samples towards the goal (see Figs. 2(d-f)). The best node is calculated among nodes on $\mathcal{T}_{RRT}$ and is returned with its corresponding path (lines 1-4 in Algorithm 2). This best node represents the node that is likely closer to the goal, as estimated by Eq. (3). In Figs. 2(d-f), the network estimates the learned distribution step-by-step given the best node.

Once a complete path is found, i.e., reaching one of the candidate goals, **LA3T*** intensively optimizes the next path segment from the committed path (see Figs. 2(g-i)). Thus, Algorithm 2 returns the committed path and the root, i.e., the end pose of the path. In Figs. 2(g-i), the network estimates the subsequent path segments (edges) from the root, and the planner continuously seeks a lower-cost next segment. Note that finding a complete path can also occur in the initial phase, though it is rare. When the vehicle reaches the new root, the next segment of the best path is selected as the committed path. The process repeats until the robot reaches the goal pose.

B. Adaptive Biasing Ratio

The adaptive biasing ratio is a novel mechanism that adaptively adjusts the biasing ratio for learned distributions. The main idea is to approximate the optimal biasing ratio using the correlation between the reliability of the learned distribution and the optimal biasing ratio. The adaptive ratio increases for highly reliable sampling distributions and decreases for less reliable ones. This study employs the confidence learning method from the classification domain [36] due to its simplicity and ability to infer reliability without extra data or additional training. Indeed, high confidence indicates a reliable distribution, which is correlated with requiring a high biasing ratio, and conversely. Other state-of-the-art confidence estimation methods could replace this reliability estimation.

The adaptive biasing ratio, $\lambda(c)$, is calculated using the confidence, c, as follows.

$$\lambda(c) = \min(\lambda_{\max}, c), \quad \forall c, \lambda_{\max} \in [0, 1]. \tag{4}$$

Here, $\lambda_{\max}$ denotes the maximum biasing ratio, ensuring completeness and optimality for the planners. In our experiments, $\lambda_{\max}$ is set to 0.95. This ratio is used within the sampling function in our framework as described in Algorithm 3.

First, the adaptive ratio, $\lambda(c)$, is calculated. A random sample is then drawn from either the target tree, the learned distribution, or the uniform distribution. With a probability of τ, the random sample is drawn from the target tree. Instead of being sampled uniformly across the entire target tree, candidate goals are preferentially sampled closer to the biased range with higher probability (see Fig. 2(b)). Note that τ is a hyperparameter set to 0.1 for our experiments. For the remaining $1 - \tau$ probability, the sample is drawn from the learned distribution with a probability of $\lambda(c)$. The higher the prediction accuracy, the more samples are drawn from this distribution. If neither condition is met, the sample is drawn from the uniform distribution to balance out less accurate learned distributions.

C. Multi-Heads Anytime Predictor

The proposed neural network model, *multi-heads anytime predictor*, incorporates confidence learning. The novel contribution lies in integrating a confidence head [36] and the target goal head into the sampling distribution estimator to leverage the adaptive biasing ratio and the target tree. The estimator is based on segmentation-based models from [4], [6], and [7], which shows a significant performance for nonholonomic robots.

The *anytime predictor* consists of one encoder and three decoders, as shown in Fig. 3. The encoder takes two inputs: a 5-channel input grid map and conditional information. The first two channels represent an occupancy grid map, identifying cells as occupied or unknown. The third channel represents a path that the robot follows, either the best or the committed path according to Algorithm 2. Also, the end pose of this path and the goal pose are represented by squares containing position and orientation information at channels 4 and 5. The encoder processes the input grid map through convolutional blocks, and at an intermediate stage, the conditional information is concatenated as new channels. Each channel contains either position or orientation information of the end pose and the goal pose, filling the entire cells with one of the following values; $x, y, cos(\theta), sin(\theta)$. This conditional information prevents the convolutional blocks from easily losing the detailed cell values of the end pose and goal pose information. Finally, the encoder outputs a context vector, taken into each head.

1) Prediction and Confidence Heads: The prediction and confidence heads output the sampling distribution and its confidence. Learning confidence is derived from the *ask for label*

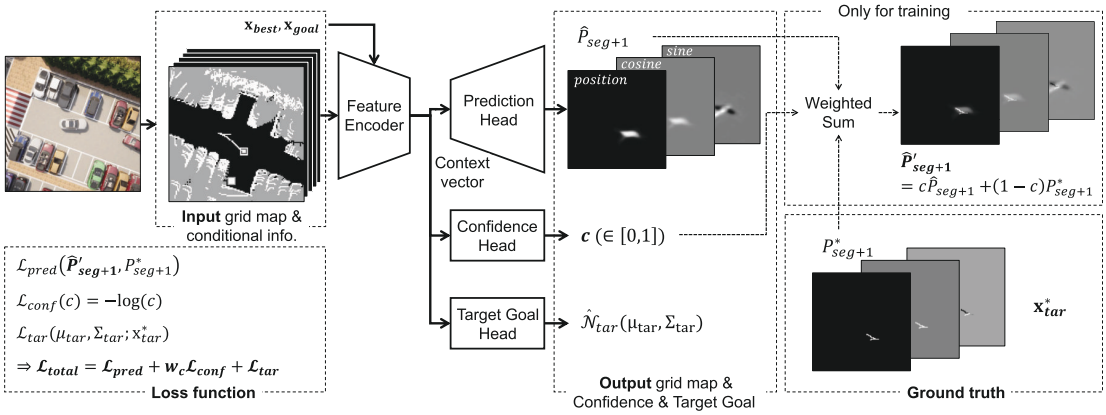


Fig. 3. Illustrations of multi-heads anytime predictor and its training pipeline.

approach [36] in the classification domain. During training, the neural network assesses its confidence for each inference, reflecting its reliance on labels. Indeed, the prediction is combined with the label based on its confidence level before computing the loss. Concurrently, the network incurs penalties based on its confidence; lower confidence induces a higher penalty. Ideally, the network learns to predict without relying on labels, except when facing uncertainty in its predictions.

The prediction head estimates the distribution for the next optimal path segment, $\hat{P}_{seg+1}$, represented by a 4-channel grid map. The first two channels contain the probability of each grid cell belonging to the next optimal segment. By applying the softmax function to each cell across these channels, the probability that each cell belongs to the segment is obtained. The last two channels contain the cosine and sine values of the vehicle's orientation on the cell. Concurrently, the confidence head outputs the confidence as a scalar value ($c \in [0, 1]$). To learn the confidence, $\hat{P}_{seg+1}$ is corrected with the ground truth grid map, P^*_{seg+1}, using c as a weighting factor.

$$\hat{P}'_{seg+1} = c\hat{P}_{seg+1} + (1-c)P^*_{seg+1}. \tag{5}$$

This corrected distribution, $\hat{P}'_{seg+1}$, is then utilized to compute the prediction loss as follows.

$$\mathcal{L}_{pred}(\hat{P}'_{seg+1}, P^*_{seg+1}) = \sum_{g \in G}(f_{ce}(g)L_{ce}(g) + f_{mse}(g)L_{mse}(g)), \tag{6}$$

where G denotes the set of all corresponding grid cells across both grid maps, $\hat{P}'_{seg+1}$ and P^*_{seg+1}, with each cell represented by g. $\mathcal{L}_{ce}(g)$ is the cell-wise cross-entropy loss, classifying whether the optimal path passes through the cell (the first two channels). $\mathcal{L}_{mse}(g)$ is the cell-wise mean-squared error loss for regressing the orientation (cosine and sine) of the cell when the optimal path lies on it (the other two channels). The weighting function $f_\bullet(g)$ applies a weight ($w_\bullet \geq 1$) to a cell on the path and 1 to others because most cells lack path information, which can slow training. Higher weights for cells on the optimal path can speed up training [7].

To avoid the network to reduce prediction loss by outputting $c = 0$ and using the full ground truth grid map, a logarithmic penalty is used as confidence loss: $\mathcal{L}_{conf} = -log(c)$. Finally, the loss for prediction and confidence heads is as follows: $\mathcal{L}_{pred} + w_c\mathcal{L}_{conf}$. w_c denotes the weight. Given the Eq. (6) and the confidence loss, high confidence ($c \to 1$) results in $\hat{P}'_{seg+1} \to \hat{P}_{seg+1}$ and in a small $\mathcal{L}_{conf}$. When confidence is low ($c \to 0$), $\hat{P}'_{seg+1}$ approximates P^*_{seg+1}, and $\mathcal{L}_{conf}$ increases. Thus, the model lowers the loss when it accurately distinguishes the outputs that are more likely to be corrected.

The approach is straightforward but requires specific techniques during training to achieve accuracy for both the learned distribution and confidence. Thus, this study refers to the original confidence learning [36] and employs the following techniques. Firstly, a budget parameter, β, represents the allowable confidence loss and prevents the model from assigning the same confidence to all inputs during training. β automatically adjusts the confidence loss weight, w_c, after each training step, guiding the confidence loss close to β. If $\mathcal{L}_{conf} > \beta$ then w_c increases, making it more costly to ask for labels; conversely, when $\mathcal{L}_{conf} < \beta$, w_c decreases. In this study, β was set to 0.7, which resulted in the best performance for our algorithm. Second, the corrected distribution in Eq. (5) is applied to only half of the batch, with half of the confidence values randomly set to 1. Since inaccurate learned predictions are always corrected using the label, the network may struggle to learn the accurate distribution and distinguish fine differences between learned distributions during inference. This technique forces the model to learn from its inaccurate predictions rather than relying solely on the label, resulting in improved predictions and confidence estimation.

2) Target Goal Head: The target goal head estimates the optimal target goal, $\mathbf{x}^*_{tar}$, as the biased range $\hat{\mathcal{N}}_{tar}$ using Gaussian regression; mean (μ_{tar}) and covariance (Σ_{tar}).

In a previous work [11], the biased range was predicted under the assumption of a diagonal covariance matrix. Unlike this, this study considers the non-diagonal covariance matrix, allowing for more accurate target goal estimation. To efficiently learn the non-diagonal covariance, the head outputs the mean (μ_x, μ_y), variance (σ_x^2, σ_y^2), and rotation angle, θ_{xy}, for Gaussian distribution. It is difficult to constrain the network to satisfy conditions necessary for a covariance matrix, such as having the same value for non-diagonal elements or maintaining the positive definite property. Thus, the Gaussian distribution is computed using the diagonal covariance (μ_{tar}, Σ_{xy}) and rotated by θ_{xy}. As a result, μ_{tar} and Σ_{tar} are

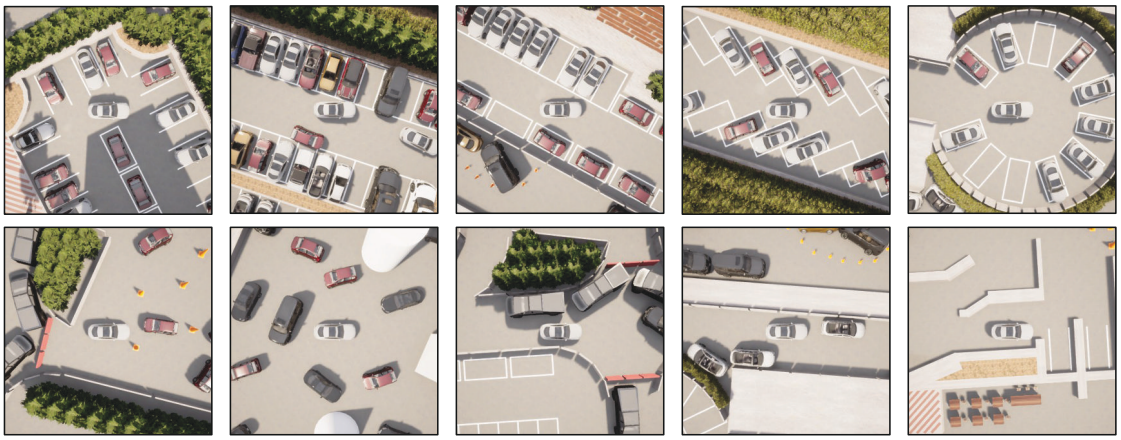


Fig. 4. Examples of driving scenarios. The top figures represent parking scenarios. The bottom figures include scenarios of driving in cluttered spaces.

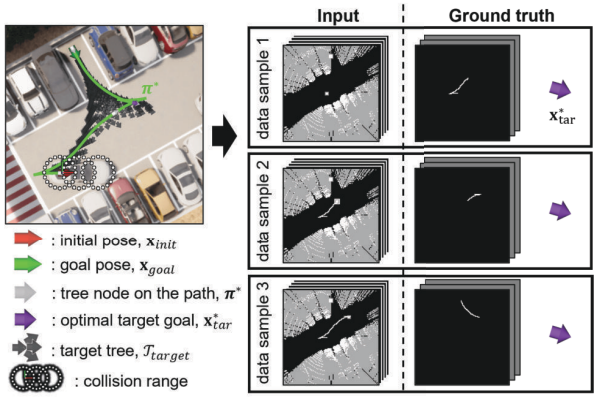


Fig. 5. Illustrations of the dataset generation.

computed as follows:

$$\mu_{tar} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \ \Sigma_{tar} = R\Sigma_{xy}R^\top,$$
$$\text{where } \Sigma_{xy} = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}, \ R = \begin{pmatrix} \cos\theta_{xy} & -\sin\theta_{xy} \\ \sin\theta_{xy} & \cos\theta_{xy} \end{pmatrix}. \quad (7)$$

Considering Eq. (7), the loss function is derived as follows:

$$\mathcal{L}_{tar}(\mu_{tar}, \Sigma_{tar}; \mathbf{x}^*_{tar}) = \frac{1}{2}\log\det(\Sigma_{tar}) + \frac{1}{2}(\mathbf{x}^*_{tar} - \mu_{tar})^\top \Sigma^{-1}_{tar}(\mathbf{x}^*_{tar} - \mu_{tar}) \quad (8)$$

3) Loss Function for Training: The final loss function is defined as follows:

$$\mathcal{L}_{total} = \mathcal{L}_{pred} + w_c\mathcal{L}_{conf} + \mathcal{L}_{tar}. \quad (9)$$

Finally, the model learns to estimate the next optimal path segment as a sampling distribution, its accuracy (reliability), and the biased range for the target tree.

V. Experiments

A. Dataset Generation

The dataset generation process is depicted in Fig. 5. The dataset was collected in various driving scenarios using the CARLA simulator [37], as shown in Fig. 4. Each scenario is categorized into one of the following types: 1) perpendicular parking, 2) parallel parking, 3) front-angle parking, 4) driving in highly cluttered spaces, and 5) long-distance driving. Long-distance driving is defined as driving at least 30 meters within a scenario of driving in cluttered spaces. To generate challenging parking scenarios, two irregularly positioned vehicles are randomly spawned on the road.

The dataset was gathered by organizing planning and control in a hierarchical structure. To efficiently handle narrow spaces, the TargetTree-RRT* algorithm [10] was used to generate the optimal path, π^*. The path is divided into the optimal path segments ($\pi^*_{seg,k}$) based on the tree nodes. For example, the optimal path in Fig. 5 consists of three segments (edges). A 3D-LiDAR sensor constructs an occupancy grid map (OGM) as 160×160 cells with 0.2 m/px resolution using a heightmap algorithm [38]; white, gray, and black cells represent occupied, unknown, and free grid cells. The best node and goal information are marked as 7×7 cell squares. The ego vehicle tracked the segments, collecting data every 0.5 m.

As the network estimates the next path segment given the committed path, various pairs of committed paths and segments were collected at each collection step. The number of data samples varies depending on the number of path segments. In Fig. 5, three pieces of data are gathered. In the first data sample, the best node (equivalent to the initial pose) and the goal pose are collected along with the OGM as inputs, with the first segment and the optimal target goal, $\mathbf{x}^*_{tar}$, as ground truths. In the second sample, the first path segment (equivalent to the committed path), as well as its end pose and the goal pose, is collected, with the second segment and $\mathbf{x}^*_{tar}$ as the ground truths. In the third sample, the first and second segments are treated as one committed path, and the third segment becomes the ground truth. In total, 254,454 data samples were collected from 4,000 scenarios. Of these scenarios, 85% were used for training and 15% for validation.

B. Experiments in Simulation

1) Settings: To thoroughly explore the effectiveness of each module in the proposed framework, we conducted ablation studies in simulated scenarios, adopting the adaptive biasing ratio and the target tree step-by-step. For each scenario type, 20 test scenarios were randomly generated. For ablation studies, the following algorithms were compared:

- *RT-RRT** [26]: The anytime framework (III-A) with informed sampling [39] and the fixed-node approach [40]. Informed sampling rejects random samples unlikely to improve the current best cost. The fixed-node approach prunes redundant nodes from the tree when it reaches its maximum size, reducing re-wiring computations.
- *LAT** [7] *(Constant)*: The learning-based anytime framework (III-A). The path segment prediction for biasing samples (see Fig. 1) is used. Several constant ratios are tested: 0.1, 0.3, 0.5, 0.7, and 0.9.
- *LAT** ***(Adapt)***: *LAT** [7] with the adaptive biasing ratio.
- *LAT** ***(Adapt)*** *w/ DO* [41]: *LAT** ***(Adapt)*** with the desired orientation (DO) sampling. DO sampling uses magnetic and potential fields to assign the desired orientation to random samples, handling narrow spaces for nonholonomic robots. In this context, DO sampling sets the desired orientation for half of the uniform samples for *LAT** ***(Adapt)***.
- *LA3T** ***(Ours** w/o BiasTT)*: The proposed LA3T* algorithm without using the biased range. Thus, it uniformly samples from the target tree (line 4 in Algorithm 3).
- *LA3T** ***(Ours** w BiasTT)*: The proposed LA3T* algorithm.

All methods used the Hybrid Curvature steering [35] for tree extension to plan a continuous-curvature path. The maximum curvature and its rate were set to 1/6.0 m^{-1} and 0.15 m^{-2}. In sampling from the learned distribution (see Algorithm 3), the positions of random samples are drawn from the probability grid map generated using the softmax function across the cells in the first two channels of $\hat{P}_{seg+1}$. The orientation is assigned to the samples by calculating **atan2**$(\sin(\theta), \cos(\theta))$ using the last two channels. Thus, each sample is represented as a pose on the predicted path segment. Collision checking was performed using circles that cover the full-size vehicle (5.255 m $\times$1.899 m) with a 20 cm safety margin and the occupied cells (see Fig. 5). Given the feasible path, the Kanayama steering controller [42] was used for path tracking, with a reference velocity of 3.6 km/h adjusted according to the curvature, cusps, and goal [10]. The control loop was set to 50 ms and the ego vehicle's localization is given.

The following two metrics were considered:

$$\text{Success Rate} = \frac{\Sigma_N \mathbf{1}_{length \leq Thres}}{N} \times 100\,[\%] \tag{10}$$

$$\text{Norm. Cost} = \frac{cost(\pi(q_s, q_g))}{cost(\pi^*(q_s, q_g))} \tag{11}$$

The success rate measures how often the algorithm reaches the goal with a path length below a specified threshold, indicating the frequency of successful real-time planning. A low success rate means the algorithm fails to find a complete path and revisits the same states multiple times (*scrub motions*) [43]. In this study, the threshold was set to three times the optimal path length for each scenario. The normalized (norm.) cost indicates how close the resulting path (Eq. (1)) is to the optimal path, π^*, among successful trials. This metric demonstrates how effectively the planner optimizes its cost function. Each algorithm was executed 10 times per test scenario, and the mean result was reported with a 95% *confidence interval*.

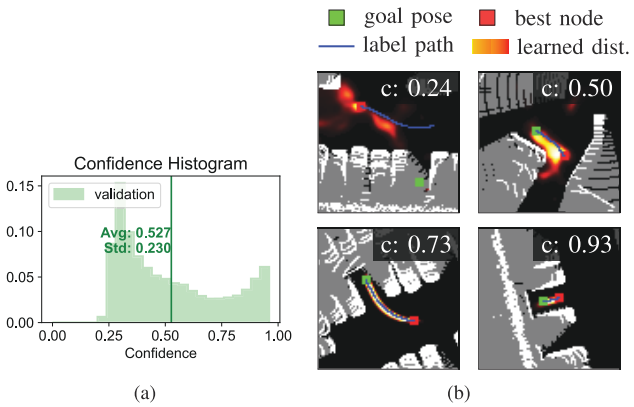


Fig. 6. Illustrations for confidence learning: (a) Confidence histogram in the validation set and (b) Examples.

2) Results of Confidence Learning: Firstly, the effectiveness of the proposed model in evaluating the reliability of its learned distribution by confidence was examined. The results are displayed in Fig. 6. As shown in the histogram and results of learned distributions, the model identified the extent of its prediction accuracy via confidence levels. For instance, in the top-left scenario, the prediction of the learned distribution shows low accuracy. Thus, the anytime predictor evaluates this prediction with low reliability (confidence). Consequently, the planner assigns a low biasing ratio for this situation. In contrast, as shown in the bottom figures, the prediction of the learned distribution is highly accurate. Therefore, the anytime predictor estimates higher confidence, leading to a higher biasing ratio in the planner.

One limitation is that confidence cannot approach zero because the confidence loss gives an infinite value for zero confidence. This prevents the model from learning to output zero confidence in extremely inaccurate prediction results, such as when facing an out-of-distribution scenario. Although employing other confidence learning methods may address this, they have limitations such as the confidence value not being restricted within [0, 1], or requiring out-of-distribution scenarios as a distinct dataset during training. In practice, adopting the method from [36] into our model demonstrated effective planning performance throughout the experiments.

3) Results of Ablation Studies: The main experimental results are shown in Tables I and IV. Table I describes the effectiveness of the adaptive biasing ratio against constant biasing ratios. Table IV shows the effectiveness of the proposed planner, **LA3T***. Min Success Rate represents the success rate in the worst-case scenario and Low 25% Norm. Cost represents the bottom 25% of the normalized cost.

a) Effectiveness of adaptive biasing: The adaptive biasing ratio mostly demonstrated the highest success rate and lowest path cost. A key advantage of adaptive biasing is its robust performance, which relieves the trade-off between success rate and path cost. To further analyze this result, two different scenarios are illustrated in Figs. 7(a) and (b). Fig. 7(a) shows a scenario with a highly accurate learned distribution from the start, while Fig. 7(b) depicts a scenario with low prediction accuracy. The experimental results of both scenarios

TABLE I

EXPERIMENTAL RESULTS COMPARING DIFFERENT CONSTANT RATIOS AND THE PROPOSED ADAPTIVE BIASING RATIO

Scenario Type	Planning Algorithms	Success Rate [%] ↑	Min. Success Rate [%] ↑	Norm. Cost [-] ↓	Low 25% Norm. Cost [-] ↓
Perpendicular Parking	*LAT* (Constant:0.9)*	92.5	20.0	1.48 ± 0.04	1.58
	LAT (Constant:0.7)*	93.0	20.0	**1.47** ± 0.04	**1.57**
	LAT (Constant:0.5)* [7]	93.5	20.0	1.52 ± 0.04	1.63
	LAT (Constant:0.3)*	**97.5**	60.0	1.58 ± 0.05	1.68
	LAT (Constant:0.1)*	96.0	**70.0**	1.68 ± 0.06	1.82
	LAT (**Adapt**)*	**97.5**	60.0	**1.47** ± 0.04	1.60
Parallel Parking	*LAT* (Constant:0.9)*	66.5	0.0	1.73 ± 0.07	1.87
	LAT (Constant:0.7)*	69.0	0.0	1.80 ± 0.08	1.95
	LAT (Constant:0.5)* [7]	69.0	0.0	1.78 ± 0.06	1.90
	LAT (Constant:0.3)*	67.0	0.0	1.82 ± 0.07	1.97
	LAT (Constant:0.1)*	70.0	0.0	1.81 ± 0.08	2.02
	LAT (**Adapt**)*	**73.5**	0.0	**1.72** ± 0.07	**1.82**
Front Angle Parking	*LAT* (Constant:0.9)*	90.5	60.0	**1.51** ± 0.07	**1.61**
	LAT (Constant:0.7)*	94.0	60.0	**1.51** ± 0.07	**1.61**
	LAT (Constant:0.5)* [7]	93.5	**80.0**	1.54 ± 0.06	1.63
	LAT (Constant:0.3)*	95.5	70.0	1.60 ± 0.07	1.75
	LAT (Constant:0.1)*	96.5	70.0	1.60 ± 0.07	1.75
	LAT (**Adapt**)*	**97.0**	**80.0**	**1.51** ± 0.07	1.62
Driving at Cluttered Spaces	*LAT* (Constant:0.9)*	93.0	60.0	1.61 ± 0.07	1.77
	LAT (Constant:0.7)*	95.5	70.0	1.57 ± 0.06	1.71
	LAT (Constant:0.5)* [7]	97.0	70.0	1.60 ± 0.06	1.76
	LAT (Constant:0.3)*	98.0	**90.0**	1.65 ± 0.06	1.79
	LAT (Constant:0.1)*	**99.0**	**90.0**	1.70 ± 0.07	1.91
	LAT (**Adapt**)*	98.5	**90.0**	**1.56** ± 0.07	**1.70**
Long-way Driving	*LAT* (Constant:0.9)*	82.0	0.0	**1.47** ± 0.05	**1.53**
	LAT (Constant:0.7)*	90.0	40.0	1.59 ± 0.06	1.70
	LAT (Constant:0.5)* [7]	92.0	40.0	1.56 ± 0.06	1.68
	LAT (Constant:0.3)*	93.0	30.0	1.68 ± 0.07	1.82
	LAT (Constant:0.1)*	91.5	60.0	1.71 ± 0.07	1.89
	LAT (**Adapt**)*	**94.5**	**70.0**	1.56 ± 0.06	1.67

TABLE II

COMPARISON OF DIFFERENT BIASING RATIOS IN THE **Accurate** PREDICTION SCENARIO

Biasing ratio	***Adapt***	0.1	0.3	0.5	0.7	0.9
$\text{Length}_{\text{path}}$ [m]	**15.4**	20.4	17.2	16.2	15.8	16.5
$\text{Time}_{\text{track}}$ [s]	**28.4**	36.1	31.7	29.0	28.6	29.7
$\text{Time}_{\text{complete path}}$ [s]	**0.1**	2.7	**0.1**	0.2	**0.1**	**0.1**

**The optimal path length*: 11.8 m

TABLE III

COMPARISON OF DIFFERENT BIASING RATIOS IN THE **Inaccurate** PREDICTION SCENARIO

Biasing ratio	***Adapt***	0.1	0.3	0.5	0.7	0.9
$\text{Length}_{\text{path}}$ [m]	47.7	**46.6**	48.2	51.9	54.5	63.1
$\text{Time}_{\text{track}}$ [s]	94.0	**92.2**	96.4	102.2	103.3	128.7
$\text{Time}_{\text{complete path}}$ [s]	10.8	**5.2**	15.6	23.7	19.3	50.7

**The optimal path length*: 34.3 m

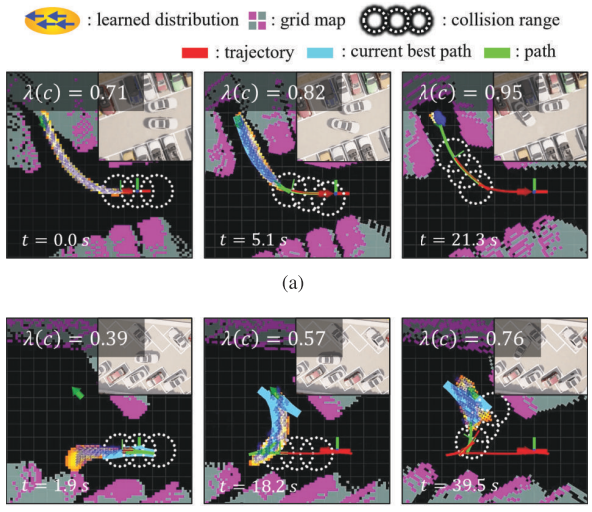


Fig. 7. Snapshots of LAT* (***Adapt***) in two different scenarios. (a) Scenario with accurate prediction. (b) Scenario with inaccurate prediction.

are detailed in Tables II and III, covering mean tracked path length, path tracking time, and time to find a complete path.

In the scenario with accurate predictions, the adaptive ratio starts higher (approximately 0.71). Thus, the algorithm used more learned samples initially, i.e., adaptively focusing on exploitation. This behavior achieved a relatively short resulting path, similar to using a high constant ratio, such as 0.7 (see Table II). In contrast, constantly using low biasing ratios, such as 0.1 and 0.3, increased the path length.

In the scenario with inaccurate predictions (Fig. 7(b)) at the beginning, the adaptive ratio starts lower. Thus, the algorithm initially focuses on exploration to compensate for initial inaccuracies and encourage uniform sampling. When the learned distribution is more reliable, adaptive biasing uses more learned samples to find a lower-cost path. This can be shown that the prediction accuracy increases (see the middle and right figures in Fig. 7), and the model encourages exploitation, resulting in a shorter best path. Note that, in such scenarios, constantly using a high ratio significantly increased path length and tracking time (see Table III), raising the probability of the vehicle failing to reach the goal. This can be seen in Table I where a high constant ratio shows a relatively low success rate. Overall, adaptive biasing improved planning performance by maintaining a high success rate and low path length by adjusting the ratio according to prediction accuracy.

TABLE IV

EXPERIMENTAL RESULTS COMPARING THE PROPOSED FRAMEWORK WITH THE METHODS TO ADDRESS THE NARROW SPACES

Scenario Type	Planning Algorithms	Success Rate [%] ↑	Min. Success Rate [%] ↑	Norm. Cost [-] ↓	Low 25% Norm. Cost [-] ↓
Perpendicular Parking	*RT-RRT** [26]	88.5	40.0	1.89 ± 0.06	2.12
	LAT (Constant:0.5)* [7]	93.5	20.0	1.52 ± 0.04	1.63
	LAT (**Adapt**)*	97.5	60.0	1.47 ± 0.04	1.60
	LAT (**Adapt**) w/ DO* [41]	**98.5**	70.0	1.50 ± 0.05	1.62
	LA3T (**Ours** w/o BiasTT)*	**98.5**	**80.0**	1.49 ± 0.05	1.64
	LA3T (**Ours** w/ BiasTT)*	98.0	70.0	**1.40** ± 0.05	**1.54**
Parallel Parking	*RT-RRT** [26]	50.5	0.0	2.04 ± 0.08	2.20
	LAT (Constant:0.5)* [7]	69.0	0.0	1.78 ± 0.06	1.90
	LAT (**Adapt**)*	73.5	0.0	1.72 ± 0.07	1.82
	LAT (**Adapt**) w/ DO* [41]	77.5	0.0	1.80 ± 0.07	1.93
	LA3T (**Ours** w/o BiasTT)*	**99.5**	**90.0**	1.48 ± 0.05	1.59
	LA3T (**Ours** w/ BiasTT)*	**99.5**	**90.0**	**1.40** ± 0.04	**1.48**
Front Angle Parking	*RT-RRT** [26]	89.0	20.0	1.81 ± 0.07	2.03
	LAT (Constant:0.5)* [7]	93.5	80.0	1.54 ± 0.06	1.63
	LAT (**Adapt**)*	97.0	80.0	1.51 ± 0.07	1.62
	LAT (**Adapt**) w/ DO* [41]	98.0	70.0	1.51 ± 0.06	1.60
	LA3T (**Ours** w/o BiasTT)*	**99.0**	**90.0**	1.51 ± 0.06	1.61
	LA3T (**Ours** w/ BiasTT)*	**99.0**	**90.0**	**1.43** ± 0.05	**1.56**
Driving at Cluttered Spaces	*RT-RRT** [26]	89.5	30.0	1.86 ± 0.06	2.09
	LAT (Constant:0.5)* [7]	97.0	70.0	1.60 ± 0.06	1.76
	LAT (**Adapt**)*	98.5	90.0	1.56 ± 0.07	1.70
	LAT (**Adapt**) w/ DO* [41]	97.5	80.0	1.58 ± 0.06	1.67
	LA3T (**Ours** w/o BiasTT)*	99.5	90.0	1.51 ± 0.06	1.63
	LA3T (**Ours** w/ BiasTT)*	**100.0**	**100.0**	**1.47** ± 0.05	**1.60**
Long-way Driving	*RT-RRT** [26]	83.0	30.0	1.90 ± 0.07	2.13
	LAT (Constant:0.5)* [7]	92.0	40.0	1.56 ± 0.06	1.68
	LAT (**Adapt**)*	94.5	**70.0**	1.56 ± 0.06	1.67
	LAT (**Adapt**) w/ DO* [41]	94.5	**70.0**	1.58 ± 0.06	1.74
	LA3T (**Ours** w/o BiasTT)*	96.0	**70.0**	**1.48** ± 0.05	1.61
	LA3T (**Ours** w/ BiasTT)*	**97.5**	**70.0**	1.49 ± 0.05	**1.58**

(a)

(b)

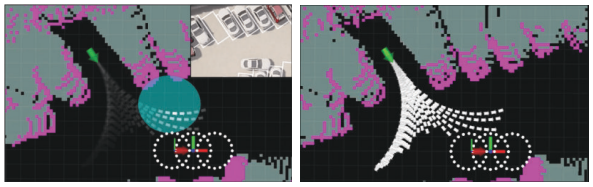
Fig. 8. Examples of snapshots for (a) LA3T* (***Ours***) and (b) LAT* (***Adapt***).

One limitation is that none of the methods achieved both a high success rate and low path cost in parallel parking. This scenario requires frequent forward/backward switches to adjust its pose within a tight space, requiring highly accurate samples to avoid collisions while extending a tree. This can be challenging even for well-trained models, as the network cannot guarantee 100% accuracy. Such environments may also lead to overconfidence, where the algorithm incorrectly assigns high confidence, assuming the learned distribution closely matches the true distribution.

b) Effectiveness of target tree: As shown by the comparison between LA3T* (**Ours**) and LAT* (***Adapt***), incorporating the target tree efficiently addressed the problem of narrow regions. In particular, this technique showcased high performance in parallel parking scenarios, with the mean success rate increasing by 26.0% and the norm. cost decreasing from approximately 1.72 to 1.40. Given that the minimum achievable cost is 1.0, this reduction of 0.32 represents roughly 44% of the maximum possible decrease of 0.72. Furthermore, adopting a target tree achieved a 90% success rate in the worst-case parallel parking scenario. This performance gain is also observed in other types of scenarios. The main strength is that the planner can easily find the near-optimal path without searching narrow regions around the goal, as shown in Fig. 8(a). This approach is practical as it achieves high performance in such regions without requiring additional datasets or extensive neural network training to handle problems needing extremely accurate samples.

Compared to DO sampling, incorporating the target tree outperforms in handling narrow regions. One disadvantage of DO sampling is requiring extensive tuning to weight each factor from the magnetic and potential fields. Depending on the scenarios, the desired orientation can lead to a local minima. Additionally, as DO sampling only accounts for orientation in a circular arc, continuous curvature planning cannot be captured by these fields.

c) Effectiveness of biased range: As observed in Table IV, the target goal head within our framework, LA3T* (**Ours** w/ BiasTT) provided a higher success rate and lower path cost compared to not using a biased range, LA3T* (**Ours** w/o BiasTT). Some candidate goals on the target tree can be far from the optimal path. For instance, as shown in Fig. 9(b), sampling candidate goals with orientations opposite to the vehicle's orientation can waste the sampling process. Instead, sampling around regions where the optimal target goal is likely



(a) (b)

Fig. 9. Illustrations of the probability of sampling on the target tree: (a) LA3T* and (b) LA3T* without the biased range. The transparency of the candidate goals on the target tree represents the probability of being sampled.

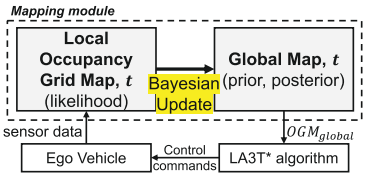


Fig. 10. Illustration of the mapping-integrated framework.

to lie is more efficient. Consequently, the biased range often allows the framework to find a path closer to optimal.

Overall, the proposed real-time framework recorded an average 9.8% increase in the success rate, compared to the LAT* (Constant: 0.5) [7] baseline. Furthermore, the results showed a decrease in the path cost of at least 20.4% (front driving) and at most 48.7% (parallel parking), showcasing a 31.9% average decrease rate in path cost when considering the minimum achievable cost of 1.0.

C. Extensions for Practical Use in Real-World Conditions

This section demonstrates the applicability of the proposed LA3T* algorithm in real-world conditions by extending it with complementary modules. Rather than introducing new algorithms, the goal is to validate that LA3T* can be effectively integrated with existing approaches to address common challenges in practice. Two additional extensions are introduced in the presence of 1) sensor noise and 2) dynamic obstacles.

1) Sensor Noises: Perception uncertainty induced by sensor noise significantly affects the efficiency of planning algorithms. In snowy weather, LiDAR may produce noisy point clouds, resulting in sudden ghost objects. Cost-effective sensors, such as cameras, can struggle to detect obstacle boundaries, causing inconsistencies in occupancy grid maps. While improving perception is a direct approach to mitigating these issues, it is beyond the scope of this work. Instead, the proposed LA3T* is extended into a mapping-integrated framework that incorporates a mapping module [44] for real-time planning and mapping, as illustrated in Fig. 10.

The mapping-integrated framework incrementally builds a global occupancy grid map by continuously integrating local occupancy maps during operation. Initially, the global map is filled with unknown occupancy probabilities. As the ego-vehicle moves, it obtains a local occupancy map from onboard sensors. This local map is then used to update the corresponding cells in the global map. Here, each cell's occupancy probability is updated over time using a Bayesian approach [44], where new local observations incrementally refine the global map. Simultaneously, LA3T* continuously plans a feasible path using the updated global map. The framework

(a) (b) (c)

Fig. 11. Impact of the mapping-integrated framework under snowy weather (10 mm/hr): (a) Raw point cloud, (b) Local occupancy grid map, and (c) Global occupancy grid map constructed by the mapping module.

TABLE V

PERFORMANCE COMPARISON WITH/WITHOUT MAPPING UNDER ADVERSE WEATHER

	Snow (10 mm/hr)		**Rain** (15 mm/hr)	
	w/o	**w/ Map**	w/o	**w/ Map**
Num. of Ghost cells [.] ↓	7.5 ± 1.9	**0.0 ± 0.0**	1.5 ± 0.5	**0.0 ± 0.0**

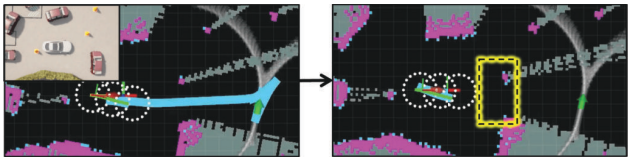


Fig. 12. Illustrations for planning instability caused by boundary noise. The cyan cells represent the occupied cell by artificial noises.

is different from the setup in Section V-B, where the LA3T* algorithm operates solely on the local occupancy grid map.

The framework was evaluated under two types of sensor noise: 1) adverse weather conditions and 2) obstacle boundary noise. The experiments assume static environments, and the localization of the ego-vehicle is given.

a) Results under adverse weather: Adverse weather such as heavy snow or rain can degrade LiDAR performance due to light scattering and refraction caused by precipitation and airborne particles. These effects often result in misreadings, producing ghost cells in the occupancy grid map, as shown in Figs. 11(a) and (b). To evaluate the robustness of the framework under such conditions, realistic weather-induced noise was simulated using the physics-based approach proposed in [45] and [46], which models the effects of adverse weather on point cloud data. Experiments were conducted across ten different scenarios, and the mean number of ghost cells was recorded. The results are summarized in Table V.

The mapping-integrated framework demonstrated strong robustness in adverse weather conditions. As shown in the table, the framework yielded an average of 0.0 ghost cells. This result is attributed to the Bayesian update employed during the global map construction, which effectively filters out ghost cells, as observed in Fig. 11(c). Consequently, the framework enables the LA3T* algorithm to perform efficient real-time motion planning even under adverse weather conditions, particularly during heavy snowfall. In contrast, the absence of such a mapping module allows ghost cells to persist in the local map, potentially resulting in inefficient planning due to false obstacle detections.

b) Results under boundary noises: Cameras and other cost-effective sensors often produce less accurate results at obstacle boundaries, leading to inconsistencies in occupancy grid maps (OGMs) [44]. These inconsistencies can cause instability during planning. As shown in Fig. 12, minor

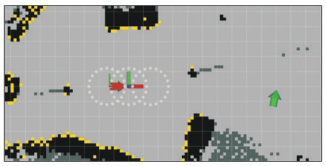


Fig. 13. Impact of the *mapping*-integrated framework under boundary noises. The gold-colored grid cell represents the removed noise cell by *mapping*. The experimental video is available at: https://youtu.be/fo9KKU9K_w4.

TABLE VI

PERFORMANCE COMPARISON WITH/WITHOUT MAPPING UNDER DIFFERENT BOUNDARY NOISE PROBABILITY (NP)

	Artificial Noise ×		NP: 0.25		NP: 0.5		NP: 0.75	
	w/o	**w/ Map**	w/o	**w/ Map**	w/o	**w/ Map**	w/o	**w/ Map**
Increased Len. [%] ↓	+0.0	**-10.5**	+9.1	**-6.7**	+12.1	**-1.5**	+20.0	**+2.4**
Infeasible Cnt. [.] ↓	4.9	**0.4**	19.7	**0.4**	20.5	**0.4**	22.6	**0.5**

fluctuations in a single grid on the boundary cause the obtained feasible path (cyan-colored path) to become infeasible. To evaluate the framework's robustness, artificial noise was added to OGM boundaries, with the noise probability controlling the likelihood of boundary inconsistencies. The relative increase in trajectory length was measured, and the number of infeasibilities—cases where a previously feasible path became infeasible due to noise—was counted. The evaluation was tested in ten scenarios, with mean results recorded.

The results are summarized in Table VI under different noise probabilities. Firstly, it is observed that sensor noise does not significantly affect the *anytime predictor*. This is because the convolutional blocks generalize features [47], minimizing the impact of boundary noise on both confidence and the learned distribution results. Furthermore, as shown in **w/ Map** in the table, the mapping-integrated framework considerably mitigates the effect of sensor noise. On average, the trajectory length decreased by 4.1%, and infeasibility counts dropped by 96%. A notable finding is that the performance improved even compared to cases without artificial noise. This is because occupancy grid maps constructed using LiDAR inherently exhibit inconsistencies at obstacle boundaries. The mapping process addresses these inconsistencies, enabling planning under more stable conditions. In environments with artificial noise, the framework removes the most noisy grid cells in the global grid map, as illustrated in Fig. 13.

The results highlight that the LA3T* can be extended to perform robustly under sensor noises. However, its performance remains limited in dynamic environments, where more advanced mapping modules could be explored. Distinguishing between static and dynamic objects, for example, by maintaining separate map layers, could help prevent the global map from becoming outdated [48].

The next subsection presents a separate extension of the LA3T* algorithm for motion planning in dynamic environments. Note that, in Section V-C2 (dynamic obstacles), the mapping-integrated framework was not used.

2) Dynamic Obstacles: Even in low-speed driving and parking scenarios, ensuring safety around vulnerable road users, such as pedestrians and cyclists, and quickly replanning paths is critical. In this experiment, the LA3T* algorithm is extended to dynamic environments by integrating it with a safety controller based on Model Predictive Control (MPC).

TABLE VII

PERFORMANCE EVALUATION IN DYNAMIC ENVIRONMENTS

Num. of Dynamic Obs.	*1*	*2*	*3*	*4*
Mean Collision [.] ↓	**0.0**	**0.0**	**0.0**	**0.0**
Num. of Stop [.] ↓	1.7 ± 0.9	2.8 ± 1.0	3.6 ± 1.2	4.9 ± 2.0

The MPC-based controller [49], [50] computes a control input by minimizing a cost function over a prediction horizon while considering vehicle dynamics and constraints. The optimization problem at each time step is formulated as:

$$\min_{\mathbf{X},\mathbf{U}} \sum_{k=0}^{N-1} \mathbf{e}_k^T Q \mathbf{e}_k + \sum_{k=1}^{N-1} (\mathbf{u}_k - \mathbf{u}_{k-1})^T R (\mathbf{u}_k - \mathbf{u}_{k-1}) \qquad (12)$$

The first term represents the state error cost, and the second term considers the control effort. N represents a prediction horizon, and Q and R represent weight matrices. $\mathbf{e}_k$ represents the state error between the predicted state and the reference state within the horizon.

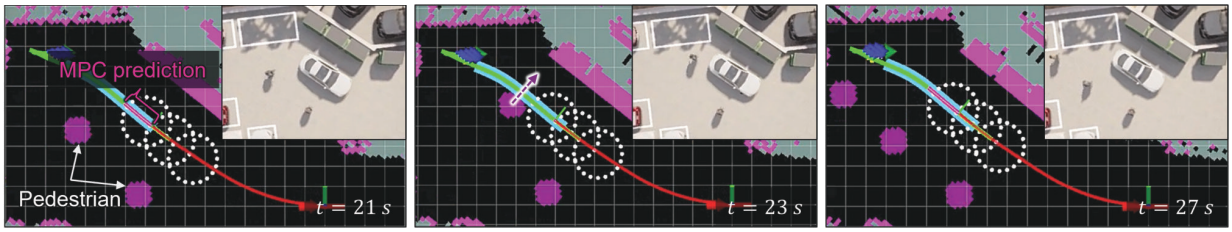
To ensure collision-free behavior and enable stop-and-go maneuvers, the reference path is constructed as the feasible portion of the committed path. For instance, when pedestrians cause the committed path to become infeasible, the reference path is defined as the portion up to the pedestrian without causing a collision. This portion is conservatively determined by considering the vehicle's current velocity, maximum allowable acceleration, and the pedestrians' near future position based on their velocity, prioritizing safety. Thus, the vehicle decelerates smoothly and stops safely in response to dynamic obstacles.

Furthermore, inspired by [51], the LA3T* algorithm avoids pruning the tree when it becomes infeasible due to dynamic obstacles. This is because, if the tree were pruned for dynamic obstacles, excessive pruning would occur. This approach mitigates over-aggressive pruning and prevents unnecessary re-searching of already discovered paths.

The experiment was conducted across twelve dynamic scenarios, each involving varying numbers of pedestrians or cyclists. Perception of these vulnerable road users is assumed to be perfect. The results are summarized in Table VII, where the scenarios are grouped based on the number of dynamic obstacles (three scenarios per group). Num. of Stop represents the number of a full stop due to dynamic obstacles.

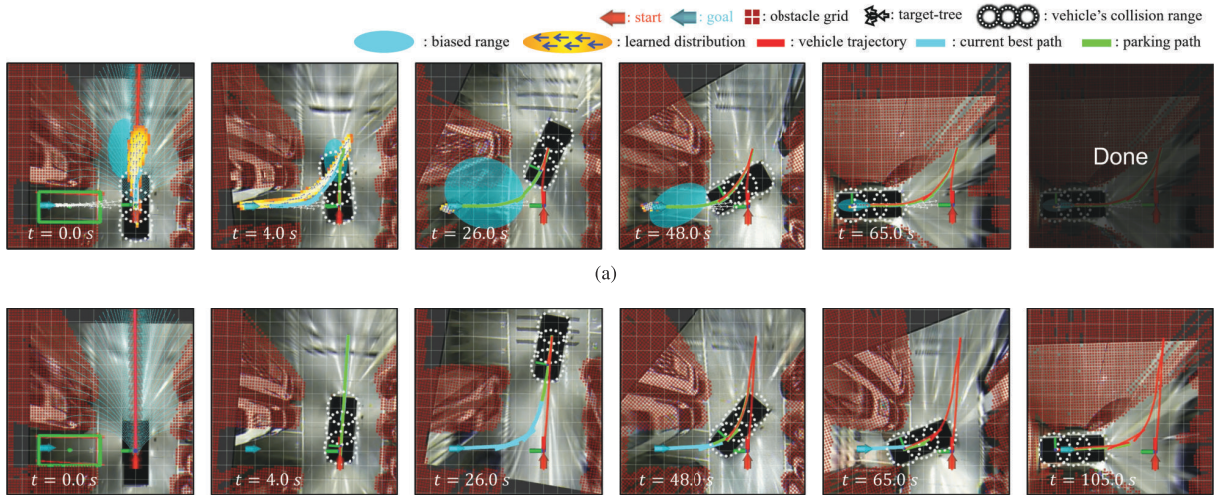
a) Results: The integration of the LA3T* algorithm with the safety controller consistently enables safe maneuvering in dynamic environments. As summarized in Table VII, the integration achieved zero collisions across all the scenarios, regardless of the number of dynamic obstacles. In particular, the extension guarantees safety even in dense environments: as the number of dynamic obstacles increases, the system adopts a more conservative stop-and-go strategy, reflected by an increase in full stops from 1.7 to 4.9.

Fig. 14 illustrates an example result. The purple circles represent pedestrians, while the pink trajectory extending from the vehicle's rear axle denotes the prediction path generated by the MPC. Initially, the vehicle accurately tracks the committed path (green trajectory). In Fig. 14(b), a pedestrian suddenly crosses the committed path, creating a potential collision risk. In response, the MPC controller calculates control commands to gradually reduce the vehicle's speed until it completely



(a) (b) (c)

Fig. 14. Snapshots for the extension of LA3T* in dynamic situations at (a) 21 s, (b) 23 s, and (c) 27 s. The experimental videos can be accessed at the following link: https://youtu.be/EsW39MsDchY.



(a)

(b)

Fig. 15. Snapshots from the real vehicle test in the indoor parking lot using (a) LA3T* (**Ours**), (b) RT-RRT* (baseline). The experimental videos can be accessed at the following link: https://youtu.be/8Q_eVhyxE7w.

stops. Once the path is clear, the vehicle gradually accelerates and resumes following the committed path.

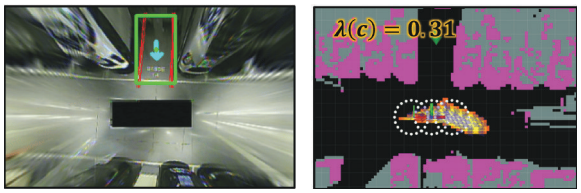
Further improvements could focus on faster goal-reaching strategies by interacting with dynamic obstacles or utilizing more advanced trajectory prediction methods [52]. This would not only preserve safety but also reduce overly conservative behavior.

D. Experiments in Real Vehicle

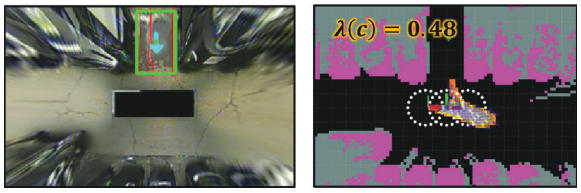
1) Settings: To evaluate the applicability within a real vehicle, LA3T* (**Ours**) was implemented in real-world indoor/outdoor autonomous valet parking tasks and qualitatively analyzed. The proposed framework was applied to the motion planning module in a parking phase.

2) Results: The results are depicted in Fig. 15. The proposed LA3T* algorithm successfully completed the autonomous valet parking task; parking the vehicle faster and with a shorter trajectory than the RT-RRT* baseline [26]. The main advantage of LA3T* appeared in the initial planning phase, where it quickly found the first near-optimal committed path (first column of Fig. 15). Furthermore, the target tree allowed the planner to avoid searching for a narrow region around the goal, thereby easily finding a complete path within a short time. In contrast, RT-RRT* followed a less optimal path and took longer to find a complete path.

This real-world experiment demonstrates the practicability of the proposed algorithm. First, the anytime framework allows the vehicle to keep moving without stopping, improving efficiency and adapting quickly to unseen obstacles, which frequently happen in the real world. Second, the input and output configurations of the anytime predictor have proven the framework's transferability, providing accurate predictions for real-world scenarios even when trained in simulation. Furthermore, the adaptive ratio intrinsically handles sim-to-real transitions by capturing decreasing prediction accuracy, as shown in Fig. 16. In Fig. 16(a) right, noises at the vehicle's



(a)



(b)

Fig. 16. Illustrations of real-world (a) indoor and (b) outdoor parking. The left figure displays a top-down view via a surround-view monitor. The right figure displays the distributions and grid maps generated by 3D-LiDAR.

front on the grid map lead to a less accurate distribution as some biased samples are infeasible. The model has learned to capture such inaccuracies with low confidence. In contrast, in Fig. 16(b), the model estimates a more accurate distribution, resulting in a higher biasing ratio used for planning. Lastly, adopting the target tree allows the vehicle to reduce tracking errors and park accurately [10]. Since the path generally contains longer straight segments near the goal, correcting tracking errors can be easier [53]. This can be observed in the uploaded video (see Fig. 15).

VI. Conclusion

This paper proposes the LA3T* algorithm, a robust real-time sampling-based motion planner for autonomous vehicles. It addresses performance degradation in planning caused by the inherent inaccuracy of the learned distribution. The key idea is to learn the reliability of the learned distribution to approximate the optimal biasing ratio. Furthermore, incorporating the target tree algorithm and the biased range effectively handles the narrow and cluttered spaces. Numerical ablation studies in various parking and driving scenarios explored the planner's effectiveness. Compared to the existing learning-based anytime framework, the proposed algorithm increased the success rate by an average of 9.8% and improved path cost by an average of 31.9% when considering the possible decrease rate. Additionally, the algorithm demonstrated its extensibility under sensor noise and dynamic environments when integrated with complementary modules. Finally, real-world experiments validated the framework's applicability, outperforming the RT-RRT* baseline in autonomous valet parking tasks.

Despite significant advances, there are some limitations. One limitation is the algorithm's inability to handle performance degradation in highly out-of-distribution scenarios far from the training data. The current confidence learning method cannot output zero confidence for that scenario. This remains for future work. Another direction for future work is enhancing the algorithm's ability to handle uncertain and dynamic environments. Although preliminary experiments have been conducted in such scenarios, future studies could focus on optimizing performance, such as incorporating the predicted trajectories or intentions of other agents into the selection of the committed path.

Appendix A
Model Architecture and Hyperparameters for Training

The detailed model architecture and hyperparameters used for training are presented in Tables VIII and IX. A convolution block consists of 3×3 kernel layers with stride 1, adjusting the input channel size to double or half. Each block is followed by either a 2×2 max pooling layer or a 2×2 unpooling layer, with a batch normalization layer and a ReLU activation function. For training, a cosine annealing with warm restarts was used for scheduling the learning rate with the following parameters: first restart at 10 epochs, multiplier factor of 2, and a minimum learning rate of 1e-5.

TABLE VIII
Network Architecture

Network design	Value
Shape of Input grid maps	$(160 \times 160 \times 5)$
Number of conv blocks in Encoder	4
Number of conv blocks in Prediction Head	4
Shape of Output ($\hat{P}_{seg+1}$)	$(160 \times 160 \times 4)$
Number of layers in Confidence Head (CH)	2
Number of nodes in each layer in CH	[128, 32]
Number of layers in Target Goal Head (TH)	2
Number of nodes in each layer in TH	[128, 32]

TABLE IX
Hyperparameters for Training

Hyperparameter	Decision
Epoch	300
Batch size	256
Initial learning rate	1e-4
Optimizer	Adam (0.9 0.999)
$w_{\bullet\in\{ce,mse\}}$	20
Budget parameter (β)	0.7
Initial w_c	0.1

Appendix B
Hyperparameters Optimization

A. Hyperparameters for Training

To determine the proper hyperparameters, the model was analyzed with several different parameters on a validation dataset. Two metrics were considered [4]: 1) The average path deviation, D. 2) The maximum prediction gap, G. The average path deviation calculates the accuracy of the N biased samples $\hat{\mathbf{x}}^{[i]}_{\text{pred}}$ relative to the label path segment, $\pi^*_{seg,k}$ as follows:

$$D(\pi^*_{seg,k}, \{\hat{\mathbf{x}}^{[i]}_{\text{pred}}\}_N) = \frac{1}{N}\sum_{i=1}^{N} \min_{\mathbf{x}\in\pi^*_{seg,k}} d(\mathbf{x}, \hat{\mathbf{x}}^{[i]}_{\text{pred}}). \tag{13}$$

$d(\cdot)$ represents the distance between the sample and a pose on $\pi^*_{seg,k}$. This distance is a weighted sum of the Euclidean distance and angular difference. It is calculated as follows: $d(\mathbf{x}, \hat{\mathbf{x}}^{[i]}_{\text{pred}}) = w_{\text{pos}}\|\mathbf{x}_{\text{pos}} - \hat{\mathbf{x}}^{[i]}_{\text{pred,pos}}\| + w_\theta|\theta - \hat{\theta}^{[i]}_{\text{pred}}|$. w_{pos} and w_θ were set to 0.35 and 0.65 to balance the different units.

The maximum prediction gap G computes the maximum discrepancy in-between biased samples. That is, this metric assesses how evenly biased samples are distributed along the path. First, the biased sample is projected as follows: $\hat{\mathbf{x}}^{[i]}_{\text{ref}} = \arg\min_{\mathbf{x}\in\mathcal{X}} d(\mathbf{x}, \hat{\mathbf{x}}^{[i]}_{\text{pred}})$. This projected sample, $\hat{\mathbf{x}}^{[i]}_{\text{ref}}$, represents the closest pose of the biased sample $\hat{\mathbf{x}}^{[i]}_{\text{pred}}$ on the path, $\mathcal{X}$. These reference poses are sorted according to their position, $s^{[i]}_{\text{ref}}$, along the path, and included in the list $\mathcal{X}_{\text{ref}}$. The maximum prediction gap is then calculated as

$$G(\mathcal{X}, \mathcal{X}_{\text{ref}}) = \frac{\max_{i=1:N-1}\left(s^{[i+1]}_{\text{ref}} - s^{[i]}_{\text{ref}}\right)}{s_{\mathcal{T}}} \times 100[\%], \tag{14}$$

where $s_{\mathcal{T}}$ represents the length of the label path.

Hyperparameter optimization was performed using these two metrics. This quantitative comparison was conducted on 200 biased samples from each scenario in the validation dataset. The results are displayed in Table X. Additionally, Fig. 17 presents the validation loss for the target goal head prediction. A learning rate of 1e-4 was chosen for its lowest

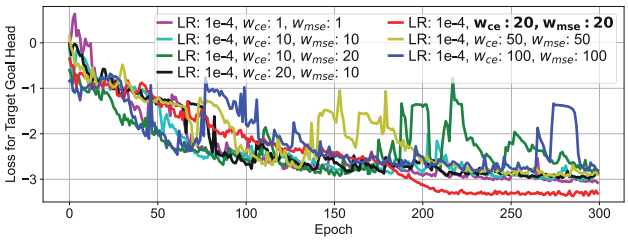


Fig. 17. Comparisons of the validation loss of $\mathcal{L}_{tar}$ with the hyperparameters set among different w_{ce} and w_{mse}.

TABLE X

HYPERPARAMETERS OPTIMIZATION: THE VALUES ARE PRESENTED WITH THEIR MEAN AND STANDARD DEVIATION

LR	w_{ce}	w_{mse}	D [-]	G [%]
1e-3			0.67 ± 0.52	15.55 ± 16.11
1e-4	**20**	**20**	**0.59 ± 0.46**	**15.01 ± 16.47**
1e-5			0.63 ± 0.45	15.00 ± 16.18
	1	1	0.90 ± 0.68	15.78 ± 16.66
	10	10	0.62 ± 0.46	15.61 ± 16.59
	10	20	0.60 ± 0.47	16.03 ± 16.97
1e-4	20	10	0.67 ± 0.45	15.41 ± 16.79
	20	**20**	**0.59 ± 0.46**	**15.01 ± 16.47**
	50	50	0.63 ± 0.45	15.00 ± 16.43
	100	100	0.73 ± 0.49	14.81 ± 15.95

path deviation, D, which is directly related to the prediction accuracy of the learned distribution. Finally, among different values tested for w_{ce} and w_{mse}, the model trained with both w_{ce} and w_{mse} set to 20 was selected.

B. Hyperparameters for Planning

The main parameters for planning, such as τ and $\lambda_{\max}$, were selected based on the following considerations.

For τ, the probability of sampling from the target tree, the value was set to 0.1 based on insights from prior works in sampling-based planning [54], [55], [56]. The target tree can be treated as a goal region, and the prior works commonly suggest that goal sampling probabilities typically fall within the range of [0.01, 0.1]. Given that the target tree represents a larger area than a single goal, a higher sampling probability was considered appropriate, leading to the selection of $\tau = 0.1$. While τ could be set to larger values, we observed that excessive sampling from the target tree negatively impacts planning performance in long-distance driving scenarios.

For $\lambda_{\max}$, which must be within [0, 1], setting this value close to 1 is beneficial for the proposed LA3T* algorithm. Lower values, such as 0.8 or 0.7, reduce the effectiveness of the adaptive biasing ratio, which adaptively prioritizes accurate learned samples and minimizes the use of inaccurate ones. Therefore, we set $\lambda_{\max}$ to 0.95, ensuring at least 5% uniform sampling while maintaining effective use of the adaptive biasing ratio. In the experiments, this setting has consistently resulted in substantial improvements in performance.

REFERENCES

[1] J. Ahn, M. Kim, and J. Park, "Autonomous driving using imitation learning with look ahead point for semi structured environments," *Sci. Rep.*, vol. 12, no. 1, p. 21285, Dec. 2022.

[2] H.-C. Yang, J. Lim, and S. Yoon, "Anytime RRBT for handling uncertainty and dynamic objects," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 4786–4793.

[3] B. Ichter, J. M. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 7087–7094.

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

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

[6] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural RRT*: Learning-based optimal path planning," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1748–1758, Oct. 2020.

[7] M. Kim, S. Shin, J. Ahn, and J. Park, "Real-time motion planning framework for autonomous vehicles with learned committed trajectory distribution," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2023, pp. 4797–4803.

[8] C. Chamzas, Z. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki, "Learning sampling distributions using local 3D workspace decompositions for motion planning in high dimensions," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 1283–1289.

[9] C.-Z. Pan, X.-Z. Lai, S. X. Yang, and M. Wu, "An efficient neural network approach to tracking control of an autonomous surface vehicle with unknown dynamics," *Expert Syst. Appl.*, vol. 40, no. 5, pp. 1629–1635, Apr. 2013.

[10] M. Kim, J. Ahn, and J. Park, "TargetTree-RRT*: Continuous-curvature path planning algorithm for autonomous parking in complex environments," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 1, pp. 1–12, Jul. 2022.

[11] J. Ahn, M. Kim, and J. Park, "Biased target-tree* algorithm with RRT* for reducing parking path planning time," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2023, pp. 1–6.

[12] B. Li, Z. Yin, Y. Ouyang, Y. Zhang, X. Zhong, and S. Tang, "Online trajectory replanning for sudden environmental changes during automated parking: A parallel stitching method," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 748–757, Sep. 2022.

[13] B. Li et al., "Optimization-based trajectory planning for autonomous parking with irregularly placed obstacles: A lightweight iterative framework," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11970–11981, Aug. 2022.

[14] R. Lattarulo and J. Perez, "Fast real-time trajectory planning method with 3rd-order curve optimization for automated vehicles," in *Proc. IEEE 23rd Int. Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2020, pp. 1–6.

[15] M. Chen et al., "FaSTrack: A modular framework for real-time motion planning and guaranteed safe tracking," *IEEE Trans. Autom. Control*, vol. 66, no. 12, pp. 5861–5876, Dec. 2021.

[16] S. Rathour, V. John, M. K. Nithilan, and S. Mita, "Vision and dead reckoning-based end-to-end parking for autonomous vehicles," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 2182–2187.

[17] P. Zhang et al., "Reinforcement learning-based end-to-end parking for automatic parking system," *Sensors*, vol. 19, no. 18, p. 3996, Sep. 2019.

[18] Y. Yang, D. Chen, T. Qin, X. Mu, C. Xu, and M. Yang, "E2E parking: Autonomous parking by the end-to-end neural network on the CARLA simulator," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2024, pp. 2375–2382.

[19] L. Lei et al., "KB-tree: Learnable and continuous Monte-Carlo tree search for autonomous driving planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 4493–4500.

[20] P. Cai and D. Hsu, "Closing the planning–learning loop with application to autonomous driving," *IEEE Trans. Robot.*, vol. 39, no. 2, pp. 998–1011, Apr. 2023.

[21] 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 Robot. Autom. Lett.*, vol. 7, no. 2, pp. 2226–2233, Apr. 2022.

[22] R. Luna, I. A. Sucan, M. Moll, and L. E. Kavraki, "Anytime solution optimization for sampling-based motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 5068–5074.

[23] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1478–1483.

[24] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1105–1118, Sep. 2009.

[25] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 4991–4996.

[26] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT* a real-time path planning algorithm based on RRT," in *Proc. 8th ACM SIGGRAPH Conf. Motion Games*, 2015, pp. 113–118.

[27] C. Gaebert and U. Thomas, "Learning-based adaptive sampling for manipulator motion planning," in *Proc. IEEE 18th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2022, pp. 715–721.

[28] S. Hecker, D. Dai, and L. Van Gool, "Failure prediction for autonomous driving," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 1792–1799.

[29] S. Mohseni, A. Jagadeesh, and Z. Wang, "Predicting model failure using saliency maps in autonomous driving systems," 2019, *arXiv:1905.07679*.

[30] T. Power and D. Berenson, "Variational inference MPC using normalizing flows and out-of-distribution projection," in *Proc. Robotics, Sci. Syst.*, New York City, NY, USA, Jun. 2022, pp. 1–10.

[31] J. Guzzi, R. O. Chavez-Garcia, M. Nava, L. M. Gambardella, and A. Giusti, "Path planning with local motion estimations," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2586–2593, Apr. 2020.

[32] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, and M. Pavone, "Revisiting the asymptotic optimality of RRT," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 2189–2195.

[33] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[34] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot. Res.*, vol. 29, no. 5, pp. 485–501, Apr. 2010.

[35] H. Banzhaf, L. Palmieri, D. Nienhüser, T. Schamm, S. Knoop, and J. M. Zöllner, "Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2017, pp. 1–8.

[36] T. DeVries and G. W. Taylor, "Learning confidence for out-of-distribution detection in neural networks," 2018, *arXiv:1802.04865*.

[37] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. Conf. Robot Learn.*, 2017, pp. 1–16.

[38] R Community. (2014). *Velodyne_Height_Map–Ros Wiki*. Accessed: Mar. 23, 2025. [Online]. Available: https://wiki.ros.org/velodyne_height_map

[39] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Informed sampling for asymptotically optimal path planning," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 966–984, Aug. 2018.

[40] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2013, pp. 354–359.

[41] S. Shin, J. Ahn, and J. Park, "Desired orientation RRT (DO-RRT) for autonomous vehicle in narrow cluttered spaces," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 4736–4741.

[42] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1990, pp. 384–389.

[43] B. Cserna, M. Bogochow, S. Chambers, M. Tremblay, S. Katt, and W. Ruml, "Anytime versus real-time heuristic search for on-line planning," in *Proc. Int. Symp. Combinat. Search*, 2016, vol. 7, no. 1, pp. 131–132.

[44] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Auton. Robots*, vol. 15, pp. 111–127, Sep. 2003.

[45] D. Hegde, V. Kilic, V. Sindagi, A. B. Cooper, M. Foster, and V. M. Patel, "Source-free unsupervised domain adaptation for 3D object detection in adverse weather," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 6973–6980.

[46] V. Kilic, D. Hegde, A. B. Cooper, V. M. Patel, and M. Foster, "LiDAR light scattering augmentation (LISA): Physics-based simulation of adverse weather conditions for 3D object detection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2025, pp. 1–5.

[47] G. Chen et al., "Robot navigation with map-based deep reinforcement learning," in *Proc. IEEE Int. Conf. Netw., Sens. Control (ICNSC)*, Oct. 2020, pp. 1–6.

[48] Z. Zheng, S. Lin, and C. Yang, "RLD-SLAM: A robust lightweight VI-SLAM for dynamic environments leveraging semantics and motion information," *IEEE Trans. Ind. Electron.*, vol. 71, no. 11, pp. 14328–14338, Nov. 2024.

[49] B. Yi, P. Bender, F. Bonarens, and C. Stiller, "Model predictive trajectory planning for automated driving," *IEEE Trans. Intell. Vehicles*, vol. 4, no. 1, pp. 24–38, Mar. 2019.

[50] M. Kim, D. Lee, J. Ahn, M. Kim, and J. Park, "Model predictive control method for autonomous vehicles using time-varying and non-uniformly spaced horizon," *IEEE Access*, vol. 9, pp. 86475–86487, 2021.

[51] Z. Shen, J. P. Wilson, S. Gupta, and R. Harvey, "SMART: Self-morphing adaptive replanning tree," *IEEE Robot. Autom. Lett.*, vol. 8, no. 11, pp. 7312–7319, Nov. 2023, doi: 10.1109/LRA.2023.3315210.

[52] L. Lindemann, M. Cleaveland, G. Shim, and G. J. Pappas, "Safe planning in dynamic environments using conformal prediction," *IEEE Robot. Autom. Lett.*, vol. 8, no. 8, pp. 5116–5123, Aug. 2023.

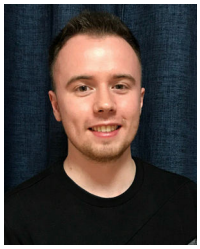
[53] R. Rajamani, *Vehicle Dynamics and Control*. New York, NY, USA: Springer, 2011. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4614-1433-9

[54] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[55] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[56] G. Kang et al., "Sampling-based path planning with goal oriented sampling," in *Proc. IEEE Int. Conf. Adv. Intell. Mechatronics (AIM)*, Jul. 2016, pp. 1285–1290.

Minsoo Kim received the B.S. degree in mechanical engineering from Hanyang University, Seoul, South Korea, in 2018, and the Ph.D. degree in intelligent systems from the Department of Intelligence and Information, Seoul National University, Seoul, in 2025. He is currently a Robotics Researcher at Dexterity, Inc., Redwood City, CA, USA. His research interests include autonomous vehicles, motion planning for nonholonomic robots, data-driven motion planning, and autonomous valet parking.



Arthur Esquerre-Pourtère received the B.S. degree in computer science from the Université de Lille, France, in 2017, and the M.S. degree in artificial intelligence from Sorbonne Université, Paris, France, in 2021. He is currently pursuing the Ph.D. degree in intelligent systems with the Department of Intelligence and Information, Seoul National University, Seoul, South Korea. His research interests include scenario generation for robots, deep-learning applied to robotics, autonomous vehicles, and data-driven motion planning.

Jaeheung Park (Senior Member, IEEE) received the B.S. and M.S. degrees in aerospace engineering from Seoul National University, Seoul, South Korea, in 1995 and 1999, respectively, and the Ph.D. degree in aeronautics and astronautics from Stanford University, CA, USA, in 2006. From 2006 to 2009, he was a Post-Doctoral Researcher and later a Research Associate with the Stanford Artificial Intelligence Laboratory. From 2007 to 2008, he worked part-time with Hansen Medical Inc., USA, a medical robotics company. Since 2009, he has been a Professor at the Graduate School of Convergence Science and Technology, Seoul National University. His research interests include robot-environment interaction, contact force control, robust haptic teleoperation, multi-contact control, whole-body dynamic control, autonomous vehicle, biomechanics, and medical robotics.