https://doi.org/10.1109/LRA.2025.3544491

Adaptive Trajectory Learning with Obstacle Awareness for Motion Planning

Huaihang Zheng[1,3], Zimeng Tan[2], Junzheng Wang[1], and Mahdi Tavakoli[3], *Senior Member, IEEE*

***Abstract*—In motion planning, efficiently navigating from a start state to a goal state in spaces with narrow passages remains a significant challenge. Recently, learning-based methods have attracted considerable attention owing to their higher inference speeds compared to traditional approaches. However, the variability in state distribution on the expert path hinders the training of neural networks, while the overly dense states may lead to redundant decision iterations and unsatisfactory planning efficiency. In this paper, we present a novel deep learning framework for motion planning, termed Adaptive Trajectory Learning with Obstacle Awareness (ATOA). Instead of performing the conventional state-wise supervision that approaches the next state, we propose to learn the trajectory along the expert path. This mechanism not only mitigates the model's dependence on the expert paths but also has the potential to yield more effective planning solutions. Additionally, obstacle information is explicitly integrated by penalizing predictions with obstacle collisions. To further enhance the planning success rate, we introduce a confidence-driven path correction (CDPC) module to adjust the infeasible local paths. Extensive experiments demonstrate the effectiveness and superiority of ATOA compared to prior approaches in handling complex scenarios. We make code available at https://github.com/ZHHhang/ATOA.**

***Index Terms*—Deep learning methods, motion and path planning, adaptive trajectory learning, obstacle awareness**

I. Introduction

THE core issue in motion planning involves finding a path from a start state to a goal state within a space containing impassable regions. The generated feasible path is typically required to possess specific properties, such as minimal length and compliance with the robot's dynamics constraints. However, the complexity of the environment and constraints on the paths significantly impact the computational burden. In many cases, when goal states or surrounding environments change, actuators need to halt their current actions and wait for the motion planner to provide a new solution, which is undesirable for real-time control systems. To address this, considerable efforts have been dedicated to enhancing the computational efficiency of motion planners [1], [2].

A key technique for accelerating planning is leveraging prior experience to guide new planning tasks. This approach, referred to as Learning-Based Motion Planning (LBMP), was first introduced in [3]. In LBMP, classical planners or human demonstrations are used to solve predefined problems, with their solutions stored in a knowledge base. Once this repository is established, learning-based planners learn from the stored knowledge to generate feasible paths for new tasks. Initially, LBMP was employed to enhance sampling distributions and expedite the convergence of Sampling-Based Optimized Planners (SBOMP) [4]. Building upon this, Qureshi et al. [5] developed a deep learning framework that iteratively predicts subsequent states. Following this advancement, several planning neural networks tailored to specific scenarios have been proposed [6]–[8].

Despite remarkable progress, several challenges still remain in learning-based methods. First, a commonly used training scheme is to approach the next state on the expert path [5], [6], [9], where the expert path is typically generated by classical algorithms and represented as a series of discrete points. While classical algorithms yield near-optimal trajectories through substantial computational effort, an important observation is that a single trajectory can be represented by different sets of discrete states. This variability poses a challenge to the training of neural networks. Another issue is that mimicking the overly dense states on expert paths may result in redundant decision iterations, thereby reducing planning efficiency. Furthermore, since the expert paths typically navigate around obstacles to minimize path lengths, even small prediction errors can lead to obstacle collisions, which is unacceptable in practical applications [10]. That is, a single or few deviated predictions during the step-by-step inference can cause an overall failure. Therefore, an adaptive strategy to correct local paths involving obstacle collisions is crucial for effective online planning.

To address these challenges, we propose a deep learning framework called **A**daptive **T**rajectory Learning with **O**bstacle **A**wareness (ATOA) for motion planning. The key insight is to replace the conventional training objective of learning the next state on the expert path (referred to as path learning) with learning the trajectory along the expert path (referred to as trajectory learning). The latter approach enables the model to adaptively learn intermediate states along the expert trajectory, rather than strictly aligning with the discrete states on the expert path, which not only mitigates the model's dependence on the expert paths but also has the potential to yield more effective planning solutions. This mechanism is achieved by

Manuscript received: September, 18, 2024; Revised December, 22, 2024; Accepted February, 07, 2025.

This paper was recommended for publication by Editor A. Bera upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by the Canada Foundation for Innovation, in part by the Natural Sciences and Engineering Research Council of Canada, in part by the Canadian Institutes of Health Research, in part by Alberta Innovates, and in part by the National Natural Science Foundation of China under Grant 62173038.

[1]Huaihang Zheng and Junzheng Wang are with the Department of Automation, Beijing Institute of Technology, Beijing 100081, China hhzheng@bit.edu.cn; wangjz@bit.edu.cn

[2]Zimeng Tan is with the Beijing National Research Center for Information Science and Technology, Department of Automation, Tsinghua University, Beijing 100084, China tzm19@mails.tsinghua.edu.cn

[3]Huaihang Zheng and Mahdi Tavakoli are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton T6G 1H9, Alberta, Canada mahdi.tavakoli@ualberta.ca

Digital Object Identifier (DOI): see top of this page.

applying a state-trajectory deviation loss for trajectory approximation, a step length loss to maximize planning efficiency, and a multi-class classification loss guiding the moving direction. Additionally, an obstacle loss function is exploited to enhance the planner's spatial perception capabilities. During online planning, a confidence-driven path correction (CDPC) algorithm is introduced to reconstruct the infeasible local paths.

In summary, our main contributions are as follows:

1) We present a deep learning framework called ATOA for motion planning, which adaptively learns the trajectory along the expert path instead of the commonly used state-wise supervision.
2) We propose an obstacle loss function to explicitly integrate the obstacle information by penalizing predicted trajectories with obstacle collisions.
3) We propose a CDPC module to correct infeasible local paths during online planning by selecting alternative directions based on the predicted direction confidence, which further enhances the planning success rate.
4) Extensive experiments in complex environments with narrow passages demonstrate the effectiveness and superiority of ATOA compared to prior approaches.

II. RELATED WORK

Motion planning is a fundamental component for achieving autonomous movement in robotics. Two popular categories of classical algorithms are sampling-based planners [1], [11]–[13] and optimization-based planners [14]–[17]. These algorithms are widely used and can generate near-optimal solutions. However, as the dimensionality of the planning space increases, the computational complexity escalates significantly, often resulting in diminished performance.

In recent years, considerable interest has been shown in integrating machine learning into motion planning. Some studies focus on optimizing specific modules in classical planners [18], [19]. For example, Tenhumberg et al. [18] utilized network predictions to warm-start an optimization-based planner. Other researchers [20], [21] devoted to employing learning-based method to bias sampling distribution in sampling-based planners.

Beyond replacing specific modules in classical planners, many researchers construct planners directly using neural networks. This approach is known as Neural Motion Planning (NMP) [22], [23]. Unlike classical algorithms, NMP is less constrained by the dimensionality of the planning space and enables rapid online planning at the expense of offline training time. A mainstream approach in NMP is the supervised learning-based planner [5]–[8], [22], which mimics the paths generated by classical algorithms. As a pioneer, Qureshi et al. [5] proposed an iterative planner network called MPNet. Building upon this, MPC-MPNet [6] combined Model Predictive Control (MPC), to address dynamically constrained problems. Despite their strengths, the effectiveness of these methods relies on the quality and quantity of expert paths. Another category of NMP methods is reinforcement learning-based planner, which has recently made breakthroughs in multi-dimensional continuous motion planning [24]–[26]. However, these methods face the challenge of escalating training difficulty as planning accuracy and degrees of freedom increase.

III. PROBLEM DEFINITION

Following the definition in [27], let the n-dimensional state space $X \subset \mathbb{R}^n$ consists of obstacle states $X_{\text{obs}} \subset X$ and permissible states $X_{\text{free}} = X \setminus X_{\text{obs}}$. The goal of motion planning is to identify a feasible path starting from a start state $x_{\text{start}} \in X_{\text{free}}$ to a goal state $x_{\text{goal}} \in X_{\text{free}}$, ensuring no intersection with X_{obs}. In this paper, we represent the feasible path as a tree structure $\mathcal{T} = (V, E)$, where the vertices V are selected from X_{free} and the edges E represent local paths connecting pairs of vertices.

In practice, motion planning task typically begins in workspace, as it is more intuitive for human perception. The workspace $W \subset \mathbb{R}^m$, also known as the environment, includes obstacle regions W_{obs} and free regions $W_{\text{free}} = W \setminus W_{\text{obs}}$. The start and goal points w_{start} and w_{goal} are initially specified in W and subsequently mapped to the corresponding states x_{start} and x_{goal} in the state space. The planner then constructs a feasible tree $\mathcal{T}$ connecting x_{start} to x_{goal}, ensuring obstacle avoidance.

IV. ADAPTIVE TRAJECTORY LEARNING WITH OBSTACLE AWARENESS FRAMEWORK

In this section, we present the Adaptive Trajectory Learning with Obstacle Awareness (ATOA), which involves two phases: (1) offline training with abundant pre-generated environments and expert paths, and (2) online planning for adaptive prediction using a confidence-based search strategy.

A. Offline Training

1) Network Architecture: As illustrated in Fig. 1, ATOA consists of an environment encoder and a neural planner. The encoder network E embeds the environment information into a highly nonlinear latent space Z:

$$E(\mathcal{M}) \rightarrow Z. \tag{1}$$

The input environment is formulated as a binary map $\mathcal{M}$, with obstacle areas set to 1 and feasible areas set to 0. Based on the convolutional neural network (CNN) architecture, the encoder network involves five convolutional blocks and five max-pooling operations alternatively, followed by three fully connected layers. The convolutional layers progressively capture the fine-grained spatial details and abstract context.

The planner network, consisting of multi-layer perceptrons (MLPs), takes Z as input and incrementally predicts a feasible path. Detailed task definition are as follows.

2) Adaptive Trajectory Learning: The objective of ATOA is to estimate a feasible path from the start state $\boldsymbol{x}_{\text{start}}$ to the goal state $\boldsymbol{x}_{\text{goal}}$, which is achieved step-by-step. That is, the network is responsible for predicting an intermediate state $\tilde{\boldsymbol{x}}$ from the current state $\boldsymbol{x}_c$ towards $\boldsymbol{x}_{\text{goal}}$. Instead of the commonly used state-wise supervision that enforces alignment between $\tilde{\boldsymbol{x}}$ and the next state on the expert path (termed path learning) [5], [7], [9], we propose to infer the trajectory along the expert path (termed trajectory learning). Trajectory learning enables the

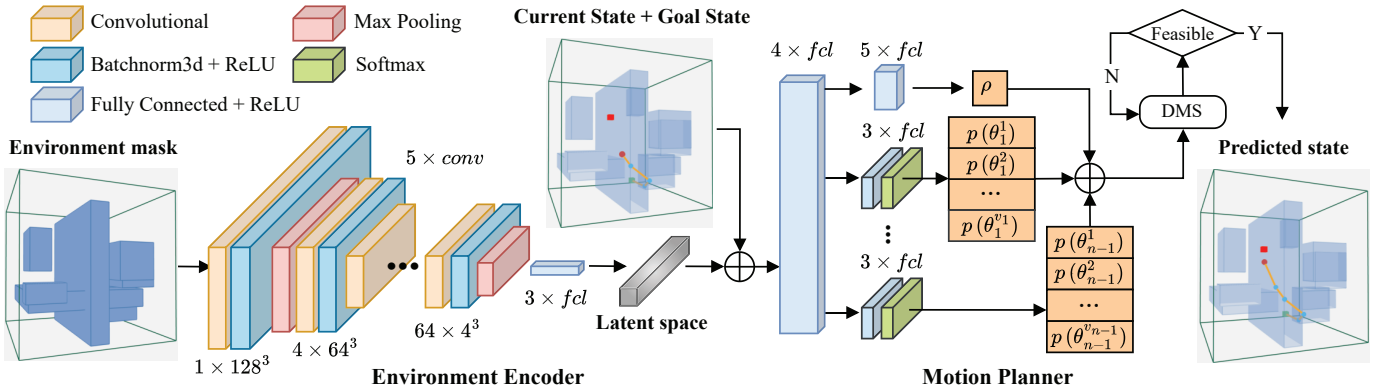

Fig. 1. Architecture of the proposed ATOA consisting of an environment encoder and a motion planner network. The n-dimensional state displacement is described using hyperspherical coordinates. The planner network is responsible for norm regression and $(n-1)$ direction angle classification simultaneously. $p(\theta_i^j)(i \in [1, n-1], j \in [1, v_i])$ denotes the j-th predicted probability of i-th direction angle, where the i-th direction angle is divided into v_i bins.

network to adaptively learn intermediate states, which not only facilitates the model's acquisition of the intrinsic properties of trajectories but also has the potential to yield more effective planning solutions.

Specifically, the state displacement from the current state $\boldsymbol{x}_c$ to the predicted state $\tilde{\boldsymbol{x}}$ is described using hyperspherical coordinates, and the network is responsible for norm ρ regression and direction angle $\boldsymbol{\theta}$ classification simultaneously. Adaptive trajectory learning raises three key conditions: (1) The predicted state $\tilde{\boldsymbol{x}}$ is required to fall on the expert trajectory while not necessarily aligned with the next state on the expert path. (2) The direction $\boldsymbol{\theta}$ should be oriented according to the reference direction on the expert path. (3) The predicted state is expected to have the maximum possible step length ρ to minimize the number of iterations required to reach the goal state. These conditions are achieved through carefully designed loss functions, which are detailed below.

a) Trajectory Approximation: The intermediate state $\tilde{\boldsymbol{x}}$ is encouraged to approach the expert trajectory $\mathcal{E}$, which is approximated as a series of discrete points $\boldsymbol{e}$ through linear interpolation between neighboring states on the expert path. The supervision is applied to the distance between the predicted state $\tilde{\boldsymbol{x}}$ and the nearest interpolated point $\boldsymbol{e}$:

$$\mathcal{L}_{\text{tra}} = \min_{\boldsymbol{e} \in \mathcal{E}} \|\tilde{\boldsymbol{x}} - \boldsymbol{e}\|_2. \tag{2}$$

b) Directional Consistency: The directional consistency is built upon the assumption that the expert path indicates the step-to-step movement direction from the start state to the goal state. Specifically, the direction from the current state $\boldsymbol{x}_c$ to the intermediate state $\tilde{\boldsymbol{x}}$ should align with the direction to the next state $\boldsymbol{x}_{c+1}$ on the expert path. In an n-dimensional state, the displacement between two states is decomposed into a Euclidean norm ρ and $(n-1)$ direction angles $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_{n-1}]$. By equally dividing the feasible range of i-th angle θ_i into ν_i bins, we convert the direction prediction into a multi-class classification task, applying a cross-entropy loss function to each direction:

$$\mathcal{L}_{\text{dir}} = \sum_{i=1}^{n-1} \mathcal{L}_{\text{dir}}^i = -\sum_{i=1}^{n-1} \sum_{j=1}^{\nu_i} y_i^j \log p(\theta_i^j), \tag{3}$$

where $p(\theta_i^j)$ is the predicted probability for the j-th class of the i-th angle, and y_i^j denotes the ground truth label that set to 1 for the correct class and 0 for the others. The classification mechanism provides candidate directions along with confidence probabilities, enabling the search strategy to explore alternative feasible solutions when encountering impassable sections, as detailed in Section IV-B2.

c) Maximizing Forward Efficiency: The objective is to maximize the ratio of the distance between $\boldsymbol{x}_c$ and $\tilde{\boldsymbol{x}}$ to the remaining path length from $\boldsymbol{x}_c$ to $\boldsymbol{x}_{\text{goal}}$ on the expert path, within the interval $(0, 1]$. This strategy allows for the largest possible step length, thereby reaching the goal state with fewer iterations. The loss is defined using a nonlinear function, where the loss value decreases as the ratio approaches 1 and increases when the ratio exceeds 1 to penalize outliers:

$$\mathcal{L}_{\text{norm}} = \log(1 + \exp(-k_1(\delta - 1))) + \alpha(\delta - 1), \tag{4}$$

$$\alpha = \begin{cases} 0, & \text{if } 0 < \delta \leq 1, \\ k_2, & \text{if } \delta > 1, \end{cases} \tag{5}$$

$$\delta = \frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}_c\|_2}{\sum_{i=0}^{p-1} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|_2}, \tag{6}$$

where $\boldsymbol{x}_0 = \boldsymbol{x}_c$ and $\boldsymbol{x}_p = \boldsymbol{x}_{\text{goal}}$, with $(p-1)$ intermediate states on the expert path. k_1 and k_2 control the sensitivity of the decrease and increase in values, respectively.

3) Obstacle Perception Module: Another crucial factor in motion planning is obstacle perception. To enhance the spatial modeling capability of the planner, we explicitly incorporate obstacle information by penalizing predicted trajectories with obstacle collisions. Specifically, obstacle regions W_{obs} defined in the workspace are projected into the state space to form X_{obs}. The boundaries $\mathscr{A}$ of X_{obs} are extracted to distinguish the interior and exterior of the obstacle regions. The predicted trajectory is formulated as a series of interpolated states $\boldsymbol{x}_p$ between the current state $\boldsymbol{x}_c$ and the predicted state $\tilde{\boldsymbol{x}}$.

Subsequently, the obstacle loss function is defined as follows:

$$\mathcal{L}_{\text{obs}} = \frac{1}{N_p}\sum_{p=1}^{N_p}\exp\left(-k_3\cdot\phi(\boldsymbol{x}_p)\min_{\boldsymbol{x}_o\in\mathscr{A}} d(\boldsymbol{x}_p,\boldsymbol{x}_o)\right), \quad (7)$$

$$\phi(\boldsymbol{x}) = \begin{cases} 1, & \mathcal{M}(\boldsymbol{x})=1, \\ -1, & \mathcal{M}(\boldsymbol{x})=0, \end{cases} \quad (8)$$

where N_p is the number of interpolated states. $d(\boldsymbol{x}_p, \boldsymbol{x}_o)$ computes the Euclidean distance between each interpolated state $\boldsymbol{x}_p$ and the obstacle boundary point $\boldsymbol{x}_o$. The sign function $\phi(\cdot)$ distinguishes between the interior and exterior of obstacles, assigning higher costs to states within obstacles.

4) Total Loss: The total loss is derived by a linear combination of the individual losses, with λ_1, λ_2, and λ_3 balancing the contribution of each component:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{tra}} + \lambda_1\mathcal{L}_{\text{dir}} + \lambda_2\mathcal{L}_{\text{norm}} + \lambda_3\mathcal{L}_{\text{obs}}. \quad (9)$$

B. Online Planning

This section introduces the online planning phase of ATOA, as summarized in Algorithm 1. It involves several key components: bidirectional path generation by the well-trained planner network (PN), a direction and magnitude selection (DMS) module, a state collision detection module (named as IsFeasible module), and a confidence-driven path correction (CDPC) strategy for reconstructing infeasible local paths, which is outlined in Algorithm 2. We note that the input to the IsFeasible module can be either a single state or a local path, with the latter consisting of a series of discrete states, each of which is checked to determine whether it lies within the obstacle region $\mathcal{X}_{\text{obs}}$.

1) Preliminary Bidirectional Path Generation (Alg. 1, Lines 1-16): Taking the start state $\boldsymbol{x}_{\text{start}}$, the goal state $\boldsymbol{x}_{\text{goal}}$, and the latent embeddings Z as inputs, the planner network (PN) conducts bidirectional path generation. This process is represented by three trees: $\mathcal{T}_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ to grow forwards from the start, $\mathcal{T}_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ to grow backwards from the goal, and $\mathcal{T}_{\text{total}} = (V_{\text{total}}, E_{\text{total}})$ to combine $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{R}}$. The states and the local path connecting corresponding states form the vertices V and edges E, respectively.

The trees $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{R}}$ are expanded alternately over a total of N_{pre} iterations (Lines 5-14). This is achieved by swapping the roles of $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{R}}$ in the end of each iteration. Specifically, the output of PN consists of a step length ρ and a direction probability matrix $\boldsymbol{P} = \{\boldsymbol{p}(\theta_1), \boldsymbol{p}(\theta_2), \ldots, \boldsymbol{p}(\theta_{n-1})\}^T$, with $(n-1)$ angles. Each $\boldsymbol{p}(\theta_i)$ is composed of $\{p(\theta_i^1), p(\theta_i^2), \ldots, p(\theta_i^{\nu_i})\}$, where ν_i is the number of divisions of the i-th angle and $p(\cdot)$ denotes the estimated probability. The angle combination $\boldsymbol{\theta}_{\max}$ with the highest probability for each dimension (denoted as $r = 0$) along with the estimated norm ρ forms the predicted state displacement, pointing from the current state to the actual state $\boldsymbol{x}_{\text{new}}$ ("Translate" function). $\boldsymbol{x}_{\text{new}}$ is then added to the corresponding tree (Lines 7-10).

The process terminates if a feasible straight-line path exists between $V_{\mathcal{F}}^{\text{end}}$ and $V_{\mathcal{R}}^{\text{end}}$, which is checked by the IsFeasible module (i.e., IsFeasible $\left(V_{\mathcal{F}}^{\text{end}}, V_{\mathcal{R}}^{\text{end}}\right)$, Lines 12 and 13). The tree $\mathcal{T}_{\text{total}}$, formed by combining $\mathcal{T}_{\mathcal{F}}$ and $\mathcal{T}_{\mathcal{R}}$, is returned as the final predicted path if all edges pass the IsFeasible module without obstacle collisions (Lines 15 and 16). Otherwise, the CDPC module is executed for local path correction (Lines 17-20).

Algorithm 1: ATOA($x_{\text{start}}, x_{\text{goal}}, \mathcal{M}$)

1 $E(\mathcal{M}) \to Z$;
2 $\mathcal{T}_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}}), \mathcal{T}_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}}), \mathcal{T}_{\text{total}} = (V_{\text{total}}, E_{\text{total}})$;
3 $E_{\mathcal{F}} \leftarrow \emptyset, E_{\mathcal{R}} \leftarrow \emptyset, V_{\text{total}} \leftarrow \emptyset, E_{\text{total}} \leftarrow \emptyset$;
4 $V_{\mathcal{F}} \leftarrow \{\boldsymbol{x}_{\text{start}}\}, V_{\mathcal{R}} \leftarrow \{\boldsymbol{x}_{\text{goal}}\}, r = 0$;
5 **for** $i = 0$ ***to*** N_{pre} **do**
6 $\{\rho, \boldsymbol{P}\} \leftarrow \text{PN}(Z, V_{\mathcal{F}}^{\text{end}}, V_{\mathcal{R}}^{\text{end}})$;
7 $\{\rho, \boldsymbol{\theta}_{\max}\} \leftarrow \text{DMS}(\rho, \boldsymbol{P}, r)$;
8 $\boldsymbol{x}_{\text{new}} \leftarrow \text{Translate}(\rho, \boldsymbol{\theta}_{\max})$;
9 $V_{\mathcal{F}} \leftarrow V_{\mathcal{F}} \cup \{\boldsymbol{x}_{\text{new}}\}$;
10 $E_{\mathcal{F}} \leftarrow E_{\mathcal{F}} \cup \{(V_{\mathcal{F}}^{\text{end}}, \boldsymbol{x}_{\text{new}})\}$;
11 $\mathcal{T}_{\text{total}} \leftarrow \text{Concatenate}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{R}})$;
12 **if** *IsFeasible*$(V_{\mathcal{F}}^{end}, V_{\mathcal{R}}^{end})$ **then**
13 **break;**
14 SWAP$(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{R}})$;
15 **if** *IsFeasible*(E_{total}) **then**
16 **return** $\mathcal{T}_{\text{total}}$;
17 **else**
18 $\mathcal{T}_{\text{total}} \leftarrow \text{CDPC}(\mathcal{T}_{\text{total}}, Z)$;
19 **if** *IsFeasible*(E_{total}) **then**
20 **return** $\mathcal{T}_{\text{total}}$;
21 $\mathcal{T}_{\text{new}} = (V_{\text{new}}, E_{\text{new}}), V_{\text{new}} \leftarrow \emptyset, E_{\text{new}} \leftarrow \emptyset$;
22 **for** $i = 0$ ***to*** *length*$(V_{total}) - 1$ **do**
23 **if** *IsFeasible*$(V_{total}^{i}, V_{total}^{i+1})$ **then**
24 $V_{\text{new}} \leftarrow V_{\text{new}} \cup \{V_{\text{total}}^{i}, V_{\text{total}}^{i+1}\}$;
25 $E_{\text{new}} \leftarrow E_{\text{new}} \cup E_{\text{total}}^{i}$;
26 **else**
27 $\mathcal{T}_{\text{new}} \leftarrow \mathcal{T}_{\text{new}} \cup \text{BIT}^*(V_{\text{total}}^{i}, V_{\text{total}}^{i+1}, \mathcal{M})$;
28 **if** *IsFeasible*(E_{new}) **then**
29 **return** $\mathcal{T}_{\text{new}}$;
30 **return** $\emptyset$;

2) Confidence-Driven Path Correction (Alg. 1, Lines 17-20 + Alg. 2): In this step, the IsFeasible module detects and remove vertices in V_{total} that fall within the obstacle region (Alg. 2, Lines 5-7). The edges that do not collide with obstacles, along with their corresponding valid vertices, are directly added to $\mathcal{T}_{\text{new}}$ (Alg. 2, Lines 8-12). The remaining unconnected vertices are subsequently processed for reconstructing the infeasible local paths (Alg. 2, Lines 15-30).

Conditioned on the latent embeddings Z, CDPC follows a bidirectional generation process similar to the preliminary planning stage, taking unconnected adjacent vertices as new start and goal state pairs. In each iteration, the DMS module estimates the local path through a confidence-based search strategy. Specifically, the angle probabilities $\boldsymbol{p}(\theta_i)$ in $\boldsymbol{P}$ are considered as independent distributions. The total probability for each angle combination is calculated by multiplying the

probabilities of its components. These probabilities are then sorted in descending order, and the r-th highest probability is selected. A threshold $r_{\max}$ is set to limit the number of search attempts. Particularly, the predicted norm ρ is adaptively adjusted to explore the space more cautiously with the lower confidence (i.e., higher r), which is accomplished through a dynamic weight w (Alg. 2, Lines 15-17):

$$\tilde{\rho} = w \cdot \rho, \tag{10}$$

$$w = \left[(0.5 + k_4) + (0.5 - k_4) \sin\left(\frac{r}{r_{\max}} \cdot \pi + \frac{\pi}{2} \right) \right], \tag{11}$$

where $k_4 = 0.01$ controls the maximum attenuation amplitude, which is reached when $r = r_{\max}$. Subsequently, the selected angle combination $\tilde{\boldsymbol{\theta}}$ and the step length $\tilde{\rho}$ are combined to form the predicted vertex. If a vertex or its corresponding edge fails the collision detection by IsFeasible module, the new vertex is discarded. The r value is then incremented by one, and the next DMS exploration is awaited (Alg. 2, Lines 19-23). Finally, CDPC outputs the corrected tree $\mathcal{T}_{\text{new}}$.

3) Replanning and Probabilistic Completeness Guarantee (Alg. 1, Lines 21-29): Due to the difficulty of ensuring the completeness of neural planners [10], planning failures may occur in some cases. Therefore, we employ the probabilistically complete BIT* to replan the segments where the CDPC module fails to find a feasible path. This scheme ensures that BIT* operates efficiently without consuming excessive time.

Algorithm 2: CDPC($\mathcal{T}_{\text{total}}, Z$)

1 $\mathcal{T}_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}}), \mathcal{T}_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$;
2 $E_{\mathcal{F}} \leftarrow \emptyset, E_{\mathcal{R}} \leftarrow \emptyset$;
3 $\mathcal{T}_{\text{new}} = (V_{\text{new}}, E_{\text{new}})$;
4 $V_{\text{new}} \leftarrow \emptyset, E_{\text{new}} \leftarrow \emptyset$;
5 **for** $i = 0$ ***to*** *length*(V_{total}) **do**
6 **if** *IsFeasible*(V^i_{total}) **then**
7 $V_{\text{total}} \leftarrow V_{\text{total}} \setminus \{V^i_{\text{total}}\}$;
8 **for** $i = 0$ ***to*** *length*$(V_{total}) - 1$ **do**
9 **if** *IsFeasible*$(V^i_{total}, V^{i+1}_{total})$ **then**
10 $V_{\text{new}} \leftarrow V_{\text{new}} \cup \{V^i_{\text{total}}, V^{i+1}_{\text{total}}\}$;
11 $E_{\text{new}} \leftarrow E_{\text{new}} \cup E^i_{\text{total}}$;
12 **continue;**
13 $V_{\mathcal{F}} \leftarrow \{V^i_{\text{total}}\}, V_{\mathcal{R}} \leftarrow \{V^{i+1}_{\text{total}}\}$;
14 $r = 1$;
15 **for** $j = 0$ ***to*** N_{rec} **do**
16 $\{\rho, \boldsymbol{P}\} \leftarrow \text{NP}(Z, V^{\text{end}}_{\mathcal{F}}, V^{\text{end}}_{\mathcal{R}})$;
17 $\{\tilde{\rho}, \tilde{\boldsymbol{\theta}}\} \leftarrow \text{DMS}(\rho, \boldsymbol{P}, r)$;
18 $\tilde{\boldsymbol{x}}_{\text{new}} \leftarrow \text{Translate}(\tilde{\rho}, \tilde{\boldsymbol{\theta}})$;
19 **if** *IsFeasible*$(\tilde{\boldsymbol{x}}_{new})$ ***or not*** *IsFeasible*$(V^{end}_{\mathcal{F}}, \tilde{\boldsymbol{x}}_{new})$ **then**
20 $r = r + 1$;
21 **if** $r > r_{max}$ **then**
22 **break;**
23 **continue;**
24 $V_{\mathcal{F}} \leftarrow V_{\mathcal{F}} \cup \{\tilde{\boldsymbol{x}}_{\text{new}}\}$;
25 $E_{\mathcal{F}} \leftarrow E_{\mathcal{F}} \cup \{(V^{\text{end}}_{\mathcal{F}}, \tilde{\boldsymbol{x}}_{\text{new}})\}$;
26 $\mathcal{T}_{\text{new}} \leftarrow \text{concatenate}(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{R}})$;
27 $r = 0$;
28 **if** *IsFeasible*$(V^{end}_{\mathcal{F}}, V^{end}_{\mathcal{R}})$ **then**
29 **break;**
30 SWAP$(\mathcal{T}_{\mathcal{F}}, \mathcal{T}_{\mathcal{R}})$;
31 **return** $\mathcal{T}_{\text{new}}$;

V. Experiments

In this section, we firstly evaluate the proposed method in the 2D and 3D narrow passage environments. Ablation studies are conducted on the training objectives and training data preprocessing. Then, we extend ATOA to high-DOF robots in both real and simulation scenarios.

A. Implementation Details

1) 2D/3D Dataset: We established 35 environment in both 2D and 3D scanerios, each facing the challenge of "narrow passages". We randomly generated 5,000 collision-free start-goal pairs in each environment. In the first 30 environments, feasible paths for the first 4,000 pairs were planned using BIT* with a 5-second planning time limit per path and allocated for training. The remaining 1,000 pairs in the first 30 environments, along with all pairs in the last 5 environments, were designated for testing.

2) High-DOF Robot Dataset: We evaluated the proposed method on Panda and Fetch robotic manipulators. For the Panda robot, we established 20 environments in a laboratory setting, with four open-top boxes randomly placed. We fixed the last joint of the Panda robot and transformed the system into a 6-DOF manipulator, resulting in a constrained planning space. In each environment, we randomly sampled 50 end-effector positions and computed collision-free configurations via inverse kinematics. These configurations were paired to form planning problems, with expert paths generated using BIT*. Based on path lengths, the top 800 paths from the first 15 environments were selected as the training set, and the top 200 paths from the last 5 environments formed the test set. The environment was represented as a voxel occupancy grid, serving as the input of the network.

For the 8-DOF Fetch robot, two challenging datasets, "Thin-Shelf" and "Cage" were generated on the MotionBenchMaker [29] simulation platform. Each dataset comprises 40 environments, with 400 expert paths per environment planned by BIT*, which were divided into a training set (30 environments) and a test set (10 environments). The start state was defined as the home (trunk) position of the robot, while the goal state was a valid inverse kinematics placing the end-effector in a grasping pose relative to a target object. Among different environments, we varied the XY coordinates and related orientation of the obstacles to the robot. Particularly, in the test set, we partly customized the sizes of obstacles to enhance planning complexity. Within each environment, variations in the target object's placement corresponded to different expert paths.

3) Network Architecture: The encoder consists of 5 convolutional layers. The planner comprises a shared feature

TABLE I
COMPARISON OF BIT*, MPNET-SD, MPNET-LD, AND ATOA IN 2D AND 3D NARROW PASSAGE ENVIRONMENTS. RESULTS ON SEEN AND UNSEEN TEST SETS ARE SHOWN OUTSIDE AND INSIDE PARENTHESES, RESPECTIVELY. THE PERFORMANCE IS EVALUATED USING SUCCESS RATE (%), PATH LENGTH, AND PLANNING TIME COST (IN SECONDS).

	Method	Success Rate	Path Length	Time Cost
2D	BIT*	100.00 (100.00)	47.02 (47.96)	1.81 (1.85)
	MPNet-SD	53.25 (50.18)	—	—
	MPNet-LD	82.33 (79.37)	44.33 (47.26)	1.36 (1.98)
	ATOA	99.74 (99.10)	43.26 (44.17)	0.21 (0.24)
3D	BIT*	100.00 (100.00)	40.54 (42.09)	6.32 (6.19)
	MPNet-SD	57.69 (42.63)	—	—
	MPNet-LD	84.36 (81.35)	38.93 (39.12)	1.72 (2.28)
	ATOA	99.45 (99.31)	36.95 (38.32)	0.26 (0.40)

extraction module, a norm regression module, and a multi-head classification module to simultaneously regress the step length and the $(n-1)$ direction angles. A Softmax operation is appended at the end of each classification head. The number of classes v_i for the i-th head varies depending on the feasible range of the direction angle in the system. During inference, the center of the selected classification bin is used as the predicted angle.

4) Hardware and Training Process: The experiments were conducted on a 2.20 GHz 6-core Intel Core i7-8750H mobile processor with 16GB RAM and an NVIDIA RTX 2060. The neural networks were implemented using PyTorch and trained on an NVIDIA RTX 3090. An Adam optimizer with a learning rate of 0.0001 was employed for training.

B. Experiments in 2D/3D Datasets

In this section, we evaluate ATOA in both 2D and 3D scenarios across seen and unseen environments. The "seen" category refers to new start and goal state pairs within environments encountered during training, whereas the "unseen" category involves environments that were not included in the training set. We compare the proposed method against an advanced classical planner **BIT*** [1] and a learning-based planner **MPNet** [5]. Specifically, BIT* was set to optimize the path cost to within 110% of the ATOA path cost. If ATOA fails to plan, there are no restrictions on BIT*'s path cost. MPNet adopts the version described in [5], which uses a voxel grid as the environment input. The ATOA and MPNet (denoted as MPNet-SD) were trained using 200 randomly selected paths from each environment in the training dataset. To maintain consistency with the original version [5], we retrained MPNet with 4,000 paths per environment (denoted as MPNet-LD). Additionally, the classical planners in both ATOA (as described in Section IV-B3) and MPNet were excluded to ensure a fair comparison.

The planning performance is evaluated using the success rate (%), path length, and time cost (in seconds), as listed in Table I. The results on the unseen test set are shown in parentheses. Due to the low success rate of MPNet-SD, other metrics are not recorded, as a large number of difficult cases with longer path lengths and more inference iterations were not included in the analysis. It can be observed that:

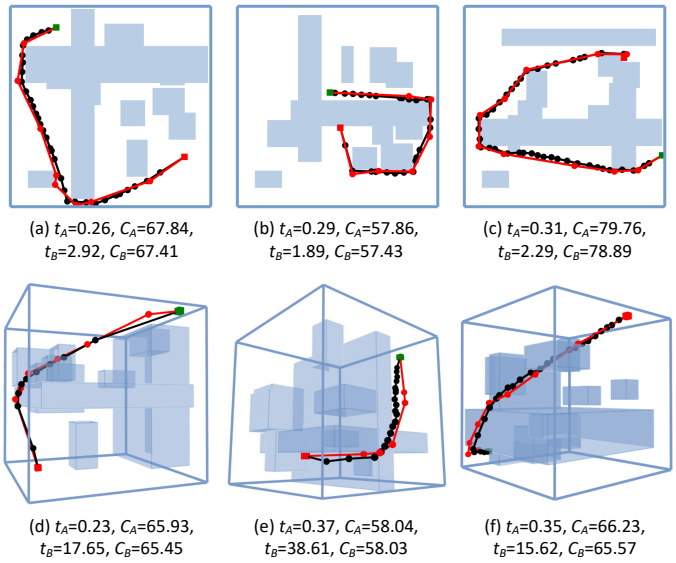

Fig. 2. Qualitative comparison of ATOA (red) and BIT* (black) in 2D and 3D challenging environments. The planning times (t) and path costs (C) are reported in each case, with red and green squares representing the start and goal states, respectively. In these tasks, BIT* stops once it finds a path with a cost no greater than that of the path generated by ATOA.

(1) ATOA achieves a higher success rate than MPNet and is comparable to BIT*, which is attributed to the CDPC module that corrects infeasible local paths by adaptively exploring directions based on probabilities. (2) Compared to MPNet-LD, ATOA demonstrates improved planning efficiency with fewer inference iterations (e.g., 22.62 for ATOA and 262.76 for MPNet-LD on the unseen test set of 3D dataset) by predicting the maximum possible step length. Under the 110% ATOA's path length constraint, BIT* requires significantly more runtime to achieve satisfactory solutions, with runtime increasing substantially as space dimensionality grows. (3) ATOA requires less training data (around 5% of MPNet-LD), indicating that the training objective of learning trajectory can be more suitable for networks. (4) ATOA exhibits robust generalization performance in unseen environments.

Fig. 2 presents challenging test cases where MPNet fails to generate a feasible path within the limit of 1,000 iterations. The red paths denote ATOA's results, while the black paths depict BIT*'s results when it first finds a solution with a path length not exceeding that of ATOA. ATOA effectively handles the complex environments and achieves significant improvements in planning speed compared to BIT* with nearly identical path lengths, especially in 3D settings.

C. Ablation Study

In this section, we conducted ablation studies on the unseen test set of 3D dataset.

1) Training Objective: To verify the effectiveness of the proposed method, we compared several variants of ATOA with different training objectives: (I) regressing the state displacement vector, (II) regressing the norm and unit direction vector separately, (III) building upon (II), modeling the direction as a $(n-1)$-dimensional Gaussian distribution and estimating the

TABLE II
ABLATION STUDY OF TRAINING OBJECTIVE ON THE 3D DATASET. THE SUPERSCRIPT "⋆" DENOTES AOTA WITHOUT THE CDPC MODULE. SEE SECTION V-C1 FOR THE DETAILED SETTINGS OF VARIANTS.

Variant	Replanning	Success Rate	Path Length	Time Cost
I	✗	73.12	40.45	3.26
II	✗	75.32	39.53	1.06
III	✓	91.42	38.76	1.77
ATOA⋆	✗	80.18	38.05	0.38
ATOA	✓	99.31	38.32	0.40

TABLE III
ABLATION STUDY OF TRAINING DATA PREPROCESSING.

Method	Preprocessing	Success Rate	Path Length	Time Cost
MPNet-LD	✗	81.35	39.12	2.28
	✓	89.77	38.21	1.03
ATOA	✗	99.31	38.32	0.40
	✓	99.68	38.15	0.39

controlling parameters, and (IV) based on (II), converting the direction regression to a multi-class classification task (i.e., ATOA). The step length maximization loss (Eq. 4) is applied in (II), (III), and (IV). Note that variant (I) and (II) do not incorporate a replanning algorithm, as they cannot provide alternative solutions. In variant (III), replanning mechanism is achieved by resampling from the learned distribution. Both performance of AOTA with and without the CDPC module is reported, as shown in Table II. It can be observed that: (1) Compared to the other objectives, direction classification task (without CDPC module) achieves a sight performance gain, which can be more favorable to the training of network. (2) The replanning algorithm brings significant improvement in success rate by reconstructing the infeasible predictions. (3) The distribution-based approach (variant III) requires more inference time than our method, as replanning via sampling from the learned distribution introduces greater randomness.

2) Preprocessing of Training Data: Preprocessing training data is a potential alternative to obtain effective solutions with fewer intermediate states. We simplified the expert paths similar to the shortcutting operation in the Open Motion Planning Library (OMPL) [28]. After preprocessing, redundant states on the expert paths are removed and the expert trajectories are redefined with fewer states.

We compared the performance of MPNet [5] and AOTA using both the original and preprocessed training data. The results are shown in Table III, which indicates that: (1) Preprocessing of training data leads to performance improvements for both methods, particularly in terms of path length and time cost metrics. (2) Even with the preprocessed training data, AOTA demonstrates superior performance by directly learning the reference trajectories and transcending the constrains of expert paths. (3) The performance gains in AOTA are smaller than MPNet, suggesting that ATOA is less dependent on the quality of training data.

TABLE IV
EXPERIMENTS ON PANDA (REAL MANUSCRIPT) AND FETCH (SIMULATION PLATFORM) ROBOTS. RRTC† DENOTES THE RRTC ALGORITHM WITH LOCAL SIMPLIFICATION. FOR FETCH DATASET, RESULTS ON THE "THIN-SHELF" AND "CAGE" DATASETS ARE SHOWN OUTSIDE AND INSIDE PARENTHESES, RESPECTIVELY.

	Method	Success Rate	Path Length	Time Cost
Panda	RRTC†	88.10	78.34	13.54
	FLAME	89.80	87.12	12.48
	FIRE	92.30	83.34	8.98
	MPNet-LD	83.50	63.41	7.63
	ATOA	95.50	56.09	1.75
Fetch	RRTC†	75.30 (83.78)	49.48 (57.63)	43.15 (17.02)
	FLAME	79.35 (88.65)	45.38 (55.25)	34.44 (15.58)
	FIRE	84.35 (92.78)	44.38 (55.77)	13.82 (9.71)
	MPNet-LD	68.10 (80.05)	43.91 (42.36)	9.26 (6.23)
	ATOA	90.28 (94.33)	38.84 (39.58)	2.88 (2.47)

D. Experiments on High-DOF Robots

We further explored the effectiveness of ATOA in high-dimensional state spaces on the Panda (in laboratory environment) and Fetch (on MotionBenchMaker [29] simulation platform) robots. In addition to MPNet-LD, the proposed method is compared with several state-of-the-art approaches: (1) RRT-Connect (**RRTC**) [11] with local simplification, which is a sampling-based bidirectional planner. The local simplification is applied using the PathSimplifier operation in OMPL [28]. Although the inclusion of local simplification may slightly increase the planning time of RRT-Connect, it significantly reduces the average path length cost generated by RRT-Connect. (2) **FLAME** [20], an experience-based algorithm that leverages an octree-based environment decomposition to estimate a global biased sampling distribution. (3) **FIRE** [21], inheriting the insights from FLAME, which learns a similarity function by extracting local representations from past solution paths. FLAME and FIRE are both combined with a sampling-based planner RRTC, where the learned distributions serve as the basis for state sampling.

Quantitative comparisons are listed in Table IV, with several predictions by our method illustrated in Fig. 3. Particularly, sampling-based methods (RRTC, FLAME, and FIRE) are theoretically capable of achieving a 100% success rate due to their asymptotic completeness, but this is not achievable in practical scenarios. Therefore, we set a timeout threshold and consider planning to have failed if the planning time exceeds this limit. It can be observed that the sampling-based methods (RRTC, FLAME, and FIRE) typically exhibit longer path lengths, as they do not continuously optimize for path quality. In contrast, neural networks demonstrate superior environmental perception and long-range modeling capabilities, showcasing potential for quickly generating initial trajectories. The replanning algorithm further enhances the planning performance by correcting infeasible local paths. Experimental results demonstrate the superiority and robustness of the proposed method, especially challenging scenarios.

VI. CONCLUSIONS

This paper presents a deep learning framework called Adaptive Trajectory Learning with Obstacle Awareness (ATOA)

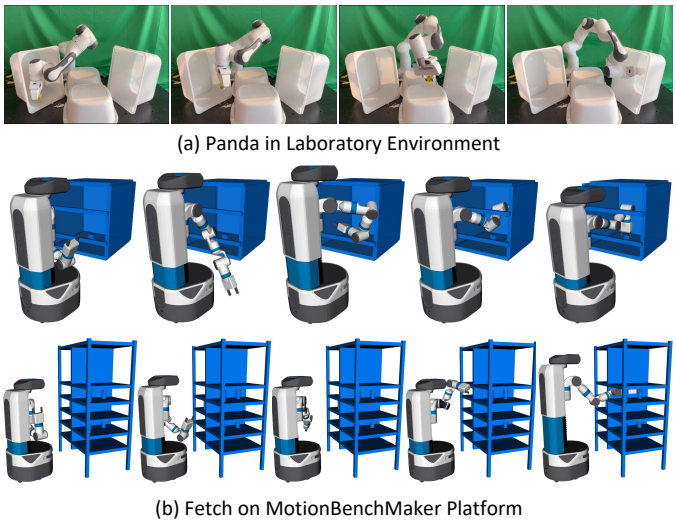

Fig. 3. Qualitative results by ATOA on a) Panda in laboratory environment and b) Fetch on the MotionBenchMaker simulation platform.

for motion planning. By replacing the conventional state-wise training objective with trajectory learning, ATOA allows the network to adaptively predict intermediate states, with the potential for more efficient planning solutions. Obstacle information is explicitly integrated by penalizing predictions with obstacle collisions. CDPC module resolves infeasible paths by exploring alternative routes based on direction confidences. Experiments in both real and simulated experiments demonstrate the superiority and effectiveness of ATOA. Future work will include extending our method to partially observable environments and multi-agent systems.

REFERENCES

[1] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search," *Int. J. Robot. Res.*, vol. 39, no. 5, pp. 543–567, Apr. 2020.

[2] A. Knobloch, N. Vahrenkamp, M. Wächter and T. Asfour, "Distance-aware dynamically weighted roadmaps for motion planning in unknown environments," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2016–2023, Jul. 2018

[3] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3671–3678.

[4] H. Ma et al., "Enhance connectivity of promising regions for sampling-based path planning," *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 3, pp. 1997–2010, Jul. 2023.

[5] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Trans. Robot.*, vol. 37, no. 1, pp. 48–66, Feb. 2021.

[6] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4496–4503, Jul. 2021.

[7] J. J. Johnson, et al., "Dynamically constrained motion planning networks for non-holonomic robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 6937–6943.

[8] G. Palomares, I. Becerra, and R. Murrieta-Cid, "Control inference neural network for motion planning with dynamical systems," *IEEE Robot. Autom. Lett.*, vol. 8, no. 12, pp. 8224–8231, Dec. 2023.

[9] F. Meng, L. Chen, H. Ma, J. Wang, and M. Q.-H. Meng, "NR-RRT: Neural risk-aware near-optimal path planning in uncertain nonconvex environments," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 1, pp. 135–146, Jan. 2024.

[10] T. Barbie and S. Mukai, "ITIRRT: A decoupled framework for the integration of machine learning into path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023, pp. 11313–11320.

[11] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, pp. 995–1001.

[12] A. M. Tahmasbi, M. S. Faghfoorian, S. Khodaygan, and A. Bera, "Zonal RL-RRT: Integrated RL-RRT path planning with collision probability and zone connectivity," *arXiv preprint arXiv:2410.24205*, Oct. 2024. [Online].

[13] M. P. Strub and J. D. Gammell, "Adaptively informed trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 3191–3198.

[14] M. Zucker et al., "CHOMP: Covariant hamiltonian optimization for motion planning," *Int. J. Robot. Res.*, vol. 32, no. 9-10, pp. 1164–1193, Aug. 2013.

[15] D. Kularatne, H. Hajieghrary, and M. A. Hsieh, "Optimal path planning in time-varying flows with forecasting uncertainties," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 4857–4864.

[16] J. Schulman et al., "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1251–1270, Aug. 2014.

[17] B. Sundaralingam, et al., "CuRobo: Parallelized collision-free robot motion generation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2023, pp. 8112–8119.

[18] J. Tenhumberg, D. Burschka, and B. Bäuml, "Speeding up optimization-based motion planning through deep learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 7182–7189.

[19] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1096–1114, Aug. 2020.

[20] 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.*, 2021, pp. 1283–1289.

[21] C. Chamzas, A. Cullen, A. Shrivastava, and L. E. Kavraki, "Learning to retrieve relevant experiences for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2022: 7233–7240.

[22] A. Fishman, et al., "Motion policy networks," in *Proc. Conf. Robot Learn.*, PMLR, 2023, pp. 967–977.

[23] W. Zhi, T. Zhang, and M. Johnson-Roberson, "Instructing robots by sketching: Learning from demonstration via probabilistic diagrammatic teaching," *arXiv preprint arXiv:2309.03835*, Sep. 2024. [Online].

[24] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 5113–5120.

[25] S. Safaoui, A. P. Vinod, A. Chakrabarty, R. Quirynen, N. Yoshikawa, and S. Di Cairano, "Safe multiagent motion planning under uncertainty for drones using filtered reinforcement learning," *IEEE Trans. Robot.*, vol. 40, pp. 2529–2542, Apr. 2024.

[26] P. Rousseas, C. P. Bechlioulis, and K. J. Kyriakopoulos, "Optimal motion planning in unknown workspaces using integral reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 6926–6933, Jul. 2022.

[27] 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.

[28] 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.

[29] C. Chamzas, et al., "MotionBenchMaker: A tool to generate and benchmark motion planning datasets," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 882–889, Dec. 2021.