# STGN: A Spatio-Temporal Graph Network for Real-Time and Generalizable Trajectory Planning

Runjiao Bao, Yongkang Xu, Chenhao Wang, Tianwei Niu, Junzheng Wang, and Shoukun Wang

***Abstract*—In dynamic and unstructured environments, mobile robots need to generate safe and efficient trajectories in real time, which poses significant challenges due to the uncertainty of surrounding obstacles. To address this, this article presents a real-time obstacle avoidance trajectory planning method, built upon a spatio-temporal graph network that integrates temporal modeling with graph attention mechanisms. The proposed network captures both temporal dynamics and spatial structural dependencies in dynamic environments by integrating a temporal information module based on long short-term memory (LSTM) and a spatial module based on relational graph attention networks (RGAT). On the whole, the approach follows a two-phase pipeline. In the offline phase, a high-quality trajectory dataset is constructed to represent the heterogeneous state graph of the robot and surrounding obstacles. Then the dataset is used to train the spatio-temporal network, which learns to map environment-state graphs to optimal control commands. In the online phase, the trained network is deployed on the robot to perform real-time perception, decision-making, and control, forming a closed-loop trajectory optimization process. Extensive experiments in both simulated and real-world scenarios demonstrate that the proposed method achieves high-quality trajectory planning, robust obstacle avoidance, and fast generalization under multi-obstacle and sudden disturbance conditions, while maintaining low computational overhead.**

***Note to Practitioners*—This article addresses the generalization challenges commonly encountered in traditional supervised learning-based obstacle avoidance methods. We propose a trajectory planning framework that leverages a spatiotemporal graph network to model dynamic interactions between a robot and its surrounding obstacles. This approach enables robust and adaptable behavior in complex, changing environments by explicitly capturing both spatial and temporal dependencies. The system is implemented on a four-wheel steering (4WS) robot for experimental validation. However, its modular design ensures straightforward transferability to a wide range of mobility platforms. The proposed method requires only basic obstacle position information, readily available from standard onboard sensors such as LiDAR or radar, and does not depend on raw sensor inputs or semantic maps, making it highly suitable for real-world deployment. It can be easily integrated into existing perception–planning–control pipelines, and future extensions may focus on incorporating richer semantic information and expanding to more diverse obstacle types.**

***Index Terms*—Mobile robot, trajectory planning, heterogeneous graph construction, long short-term memory (LSTM), relational graph attention networks (RGAT).**

Received 17 May 2025; revised 31 July 2025; accepted 23 September 2025. Date of publication 25 September 2025; date of current version 8 October 2025. This article was recommended for publication by Associate Editor S.-L. Chen and Editor P. Rocco upon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China under Grant 62473044 and in part by the Fundamental Research Funds for the Central Universities of China under Grant 2024CX06030. *(Corresponding author: Shoukun Wang.)*

The authors are with the National Key Laboratory of Autonomous Intelligent Unmanned Systems, School of Automation, Beijing Institute of Technology, Beijing 100081, China (e-mail: 3120230765@bit.edu.cn; yongkang.xu@bit.edu.cn; 3120230774@bit.edu.cn; ntwbit@bit.edu.cn; bitwsk@bit.edu.cn; wangjz@bit.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TASE.2025.3614472, provided by the authors.

Digital Object Identifier 10.1109/TASE.2025.3614472

## I. Introduction

THE rapid evolution of cyber-physical systems and intelligent transportation infrastructure has positioned autonomous navigation technology at the forefront of interdisciplinary research, synthesizing advancements in robotics, artificial intelligence, and control theory [1]. Modern mobile robots rely on a layered architecture where perception and state estimation provide situational awareness, and the planning-control stack translates this understanding into actions [2]. Within this paradigm, trajectory planning emerges as the pivotal intellectual challenge, tasked with generating dynamically feasible, collision-free paths that reconcile stringent safety constraints with mission-specific optimality criteria [3].

Existing path planning methods can be roughly categorized into four types: search-based, sampling-based, optimization-based, and learning-based algorithms. Search-based planners discretize the state space and search for a path within the resulting graph [4]. Representative examples include Dijkstra [5] and A* [6] algorithms. Anytime Repairing A* [7] extends the A* by introducing an inflation factor, improving efficiency at the cost of yielding only suboptimal solutions. In addition, conventional search methods often neglect robot kinematic constraints, potentially resulting in infeasible paths. To address this, Hybrid A* [8] integrates motion primitives that consider such constraints, enhancing the dynamic feasibility of the generated paths. Overall, when effective heuristics or high-quality motion primitives are available, search-based methods can perform well. However, these prerequisites are difficult to guarantee in complex real-world scenarios.

Sampling-based path planning algorithms construct a reachable graph by randomly sampling the state space, avoiding full discretization and offering good scalability. Representative methods include Rapidly-exploring Random Tree (RRT) [9], its improved version RRT* [10], and Probabilistic Roadmap (PRM) [11]. In practice, these algorithms often require enhancements to satisfy robot kinematic and optimization objectives [12]. Kinodynamic RRT* [13] accounts for kinematic constraints in its sampling strategy, but this significantly reduces planning efficiency. These methods can quickly produce feasible solutions in complex environments, they cannot guarantee optimality within limited computation time. Furthermore, most of these methods output discrete paths rather than trajectories. Therefore, an additional trajectory generation module is usually required to ensure dynamic feasibility and execution safety [14].

Optimization-based methods formulate path planning as a constrained optimization problem, generating optimal trajectories by minimizing a cost function subject to various constraints. These methods are advantageous in that they can simultaneously incorporate multiple cost metrics and complex constraints, resulting in more executable and practical trajectories. However, they typically require a good initial trajectory to start the optimization process; otherwise, they are prone to getting trapped in local minima. Particle Swarm Optimization (PSO) [15] is first used to generate an initial trajectory, which is then refined through nonlinear programming (NLP) to ensure feasibility and optimality [16]. Zhang et al. [17] combined Voronoi diagrams and an improved adaptive RRT* algorithm to find an initial path, finally solve an optimal control problem for trajectory refinement. Due to the high computational cost of optimization-based methods, they are difficult to apply in real-time scenarios.

In recent years, learning-based trajectory planning methods have attracted growing attention. Reinforcement learning (RL) continuously interacts with the environment and optimizes policies via trial and error, offering strong adaptability and robustness. For example, Choi et al. [18] utilized sequences of LiDAR frames to enable efficient navigation and obstacle avoidance in environments with both static and dynamic obstacles. Further research [19] incorporated different forms of image input, highlighting the critical role of depth information in navigation tasks. RL methods involve stochastic training, leading to high computational costs and slow convergence. Their performance hinges on well-designed reward functions, and the absence of reference solutions makes global optimality hard to ensure.

Supervised learning-based path planning methods primarily learn the mapping between perception states and optimal actions by imitating expert demonstrations. Banzhaf et al. [20] used a convolutional neural network (CNN) to predict the vehicle's future posture based on the current driving environment, assisting in generating appropriate control strategies. Chai et al. further explored the application of recurrent neural networks (RNN) in trajectory generation in [21] and [22], successfully learning the nonlinear mapping between vehicle states and optimal control. These studies collectively validate the strong modeling and real-time inference capabilities of neural networks in path planning tasks.

Current neural network-based trajectory planning research primarily focuses on structured static environments, struggling to effectively handle unstructured dynamic obstacles commonly found in real-world robot applications. Some studies have made preliminary explorations to establish adaptive planning mechanisms in dynamic environments. For example, Xing et al. [23] employed long short-term memory (LSTM) and attention architecture to fit optimal control strategies in dynamic environments. Liu and Yu [24] proposed a trajectory planning approach for vehicles by integrating B-spline optimization with deep neural network (DNN), enabling efficient and responsive obstacle avoidance. However, limited by fixed-dimensional input features, traditional supervised learning paradigms may experience safety degradation during the generalization, potentially leading to trajectory oscillations or collision risks.

TABLE I
EXPLANATIONS FOR THE ROBOT-RELATED PARAMETERS

| Relevant Notations | Definitions |
|---|---|
| $(x_p, y_p)$ | Coordinates of the robot's center point |
| $W_i$ | The $i$-th wheel of the robot |
| $(x_i, y_i)$ | Coordinates of the $i$-th wheel |
| $v_i$ | Velocity of wheel $W_i$ |
| $\eta_i$ | Steering angle of wheel $W_i$ |
| $2w$ | Overall width of the robot |
| $2l$ | Overall length of the robot |
| $v$ | velocity of the robot |
| $\theta$ | Heading angle of the robot |
| $L$ | Wheelbase (distance between axles) |
| $W$ | Track width (distance between wheels) |

To address the challenge of sudden moving obstacles during robot navigation, this article proposes a novel online obstacle avoidance trajectory planning method that integrates temporal modeling and graph attention mechanisms. While maintaining low computational cost, the proposed method demonstrates excellent trajectory quality and robustness in scenarios with multiple obstacles and sudden disturbances, and exhibits fast generalization capabilities. Specifically, the main contributions of this work are as follows.

1) A modeling method that uniformly represents obstacles in the environment and the robot's own state as a heterogeneous graph is proposed, laying the foundation for the spatiotemporal information fusion of subsequent graph neural networks.
2) A spatio-temporal graph network architecture is designed and implemented, which can simultaneously mine the historical dynamic characteristics of nodes and the spatial interaction relationship between multiple obstacles on the graph structure.
3) Experiments in various simulation and real-world settings validate the method's effectiveness in safe obstacle avoidance, optimal trajectory planning, and fast adaptation to complex environments.

The rest of this article is organized as follows. Section II formulates the time-optimal trajectory planning problem. Section III details the proposed online planning framework and its deployment scheme. Section IV presents extensive experimental results and comparative analyses with baseline methods. Finally, Section V draws conclusions.

## II. Obstacle Avoidance Problem Formulation

### A. 4WS Robot Kinematic Model

We make the following assumptions regarding the robot: the center of mass coincides with the geometric center, all four wheels have the same radius and mass, and sliding friction is

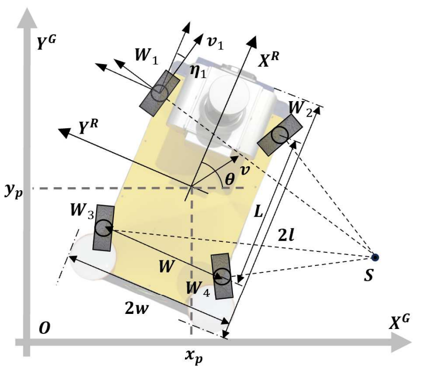


Fig. 1. Kinematic model of the 4WS robot.

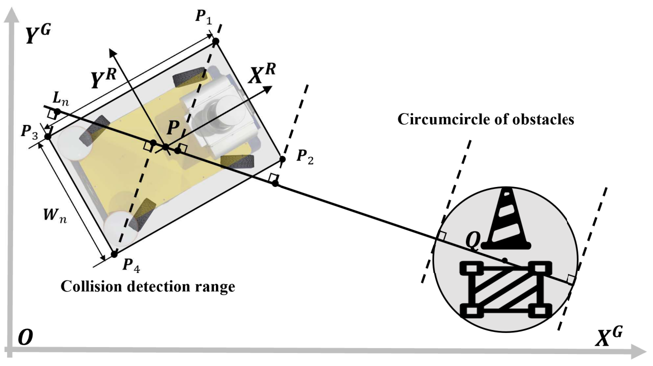


Fig. 2. Obstacle avoidance constraint illustration.

neglected. Table I provides explanations for the robot-related parameters. Under these assumptions, the kinematic model of the robot can be expressed as:

$$\begin{cases} \dot{x}_p = \dfrac{1}{4}\sum_{i=1}^{4} v\cos(\eta_i + \theta) \\ \dot{y}_p = \dfrac{1}{4}\sum_{i=1}^{4} v\sin(\eta_i + \theta) \\ \dot{\theta} = \sum_{i=1}^{4} \dfrac{v(-y_i^r\cos\eta_i + x_i^r\sin\eta_i)}{4x_i^{r2} + 4y_i^{r2}}, \end{cases} \tag{1}$$

where $x_i^r = (-1)^{\lfloor \frac{i-1}{2} \rfloor}L$, $y_i^r = (-1)^{i-1}W$.

In addition, during robot operation, the control and state variables must satisfy physical actuator constraints, formulated as:

$$X_{\min} \le X \le X_{\max}, \tag{2}$$

$$U_{\min} \le U \le U_{\max}, \tag{3}$$

$$\begin{cases} X(t_0) = X_0 \\ X(t_f) = X_f, \end{cases} \tag{4}$$

where $X = [x_p, y_p, \theta, v]$ and $U = [a, \eta]$ denotes the control input, where $a$ is the acceleration and $\eta$ is the steering angle. $t_0, t_f$represent the initial and terminal times, respectively.

In practice, Ackermann steering is widely used to reduce tire slip and steering load by aligning all wheels toward a common instantaneous center of rotation. The robot in this work adopts a four-wheel independent drive and steering (4WS) system, enabling full Ackermann steering, as illustrated in Fig. 1.

To ensure a unique instantaneous center of rotation, the steering angles and wheel speeds must satisfy the following constraints:

$$\begin{cases} \eta_1 = -\eta_3, \quad \eta_2 = -\eta_4 \\ v_1 = v_3, \quad v_2 = v_4 \\ \cot\eta_2 - \cot\eta_1 = \dfrac{2W}{L} \\ v_1\sin\eta_1 = v_2\sin\eta_2. \end{cases} \tag{5}$$

The controller provides a target forward velocity and a simplified equivalent steering angle $\{v, \eta\}$, from which the actual wheel inputs are computed as:

$$\begin{cases} \cot\eta_1 = \cot\eta - \dfrac{W}{L}, \quad \cot\eta_3 = \cot\eta + \dfrac{W}{L} \\ v_1 = \dfrac{v\tan\eta}{\sin\eta_1}, \quad v_3 = \dfrac{v\tan\eta}{\sin\eta_3}. \end{cases} \tag{6}$$

## B. Obstacle Avoidance Constraints

In real-world navigation, obstacles often have irregular shapes that are difficult to model with standard primitives. To maintain model simplicity, we adopt a conservative circular approximation, as illustrated in Fig. 2. For obstacles with strong anisotropy, a multi-circle composite model can be used to improve accuracy.

It is worth noting that traditional geometry-based obstacle avoidance strategies—such as those that calculate the perpendicular distance between the robot's corner points and the outer contour of obstacles—often introduce distance-based constraints dependent on the solution from the previous iteration. This can significantly reduce the solver's efficiency and stability. To address this issue, our formulation relies solely on constraints expressed in terms of optimization variables, thereby enhancing both model expressiveness and computational performance:

$$\begin{cases} (x_1, y_1) = (x_p + l\cos\theta - w\sin\theta,\ y_p + l\sin\theta + w\cos\theta) \\ (x_2, y_2) = (x_p - l\cos\theta - w\sin\theta,\ y_p - l\sin\theta + w\cos\theta) \\ (x_3, y_3) = (x_p + l\cos\theta + w\sin\theta,\ y_p + l\sin\theta - w\cos\theta) \\ (x_4, y_4) = (x_p - l\cos\theta + w\sin\theta,\ y_p - l\sin\theta - w\cos\theta). \end{cases} \tag{7}$$

Let the circular obstacle be centered at $Q(x_q, y_q)$ with radius $r$. The line connecting the robot center $P(x_p, y_p)$ to the obstacle center can be expressed as:

$$\Delta y \cdot (x - x_p) - \Delta x \cdot (y - y_p) = 0, \tag{8}$$

where $\Delta x = x_q - x_p$ and $\Delta y = y_q - y_p$.

The two tangents to the obstacle that are perpendicular to the line segment $PQ$ can be expressed as follows:

$$-\Delta y \cdot x + \Delta x \cdot y + (A \pm B) = 0, \tag{9}$$

where $A = -(x_q\Delta x + y_q\Delta y)$, and $B = r\sqrt{\Delta x^2 + \Delta y^2}$. To ensure obstacle avoidance, all robot corner points must lie on the same side of both tangent lines. This requirement is formulated as:

$$\begin{cases} L_i = x_i\Delta y + y_i\Delta x + (A + B) \\ R_i = x_i\Delta y + y_i\Delta x + (A - B) \\ L_i \cdot R_i \ge 0 \end{cases} , \forall i \in \{1, 2, 3, 4\}. \tag{10}$$

With these constraints, the robot is completely separated from the obstacle by at least one tangent line, thus avoiding the obstacle without introducing non-convex distance constraints.

### *C. Trajectory Planning Optimization Model*

To clearly formulate the trajectory planning task, we define a constrained optimization problem that balances motion efficiency, safety, and physical feasibility. The objective function $J$ includes costs during the motion and the terminal costs. To achieve the shortest travel time, we define the objective function $J$ as the final time $t_f$. To ensure feasibility and safety, the optimization incorporates multiple constraints, including robot dynamics, physical limits, boundary conditions, and collision avoidance. The complete formulation is given as:

$$\begin{aligned}\min_{X,U} \quad & J = \int_0^{t_f} L(X(t), U(t))dt + \Phi(x(t_f), t_f) \\ \text{subject to: } & \text{Kinematic constraints(Eq.(1))} \\ & \text{State constraints(Eq.(2))} \\ & \text{Actuator constraints(Eq.(3))} \\ & \text{Initial and terminal conditions(Eq.(4))} \\ & \text{Collision avoidance constraints(Eq.(10)).} \end{aligned} \tag{11}$$

## III. Trajectory Planner Framework

In Section II, we formulated a time-optimal trajectory optimization model for robot obstacle avoidance. This problem can be effectively solved using standard nonlinear optimization solvers such as IPOPT [25], yielding high-precision trajectories. However, such offline optimization-based methods suffer from limited real-time applicability and struggle to handle unexpected or dynamic obstacles encountered during robot operation.

To address this limitation, we propose an online trajectory planning framework based on spatio-temporal graph network architecture. This framework integrates real-time environmental information—such as the robot's current state and sudden obstacle appearances—into the planning process, enabling rapid generation of optimal feedback control actions. As a result, the proposed method achieves efficient real-time obstacle avoidance with low computational overhead. Moreover, it demonstrates strong generalization and transferability in environments with multiple dynamic obstacles.

### *A. Construction of Optimal Trajectory Dataset*

*1) Construction of Robot-Obstacle Topology Map:* Most existing deep learning-based obstacle avoidance algorithms adopt fixed-dimensional tensor input structures. This rigid feature representation exhibits significant limitations in dynamically changing obstacle distribution scenarios. Specifically, when the number of obstacles in the environment varies, conventional paradigms typically require preprocessing techniques such as zero-padding or data truncation. These approaches not only increase the complexity of feature engineering but also risk the loss of critical environmental information or the introduction of noise, ultimately compromising the robustness of the decision-making system.

To address these challenges, this study proposes a heterogeneous graph representation with the robot as the querying agent, wherein the robot and surrounding obstacles are modeled as a dynamic set of nodes. Due to the inherent flexibility of graph structures in handling variable node counts, the proposed method is naturally suited for scenarios with varying obstacle densities, thereby significantly enhancing the model's generalization and scalability [26].

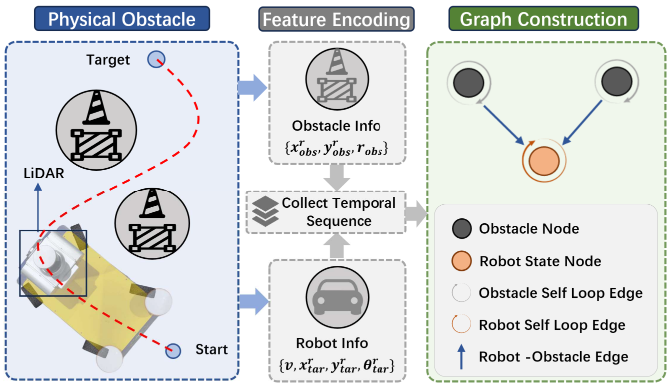


Fig. 3. Construction of topological maps between robot and obstacles.

To capture the spatiotemporal evolution during motion, each node is embedded with a time series of features. Specifically, the information of the robot node, denoted as $S_v$, is defined as:

$$\begin{aligned} S_v &= \{F_{state_t} \mid t \in \{0, 1, \cdots, t_N\}\} \\ F_{state_i} &= \{v, x^r_{tar}, y^r_{tar}, \theta^r_{tar}\}, \end{aligned} \tag{12}$$

where $v$ denotes the current velocity of the robot, and $x^r_{tar}$, $y^r_{tar}$, and $\theta^r_{tar}$ represent the relative position and orientation of the target point in the robot's local coordinate frame. $N$ represents the time horizon considered. Similarly, the obstacle node information $S_o$ is given by:

$$\begin{aligned} S_o &= \{F_{obs_t} \mid t \in \{0, 1, \cdots, t_N\}\} \\ F_{obs_i} &= \{x^r_{obs}, y^r_{obs}, r_{obs}\}, \end{aligned} \tag{13}$$

where $x^r_{obs}$ and $y^r_{obs}$ indicate the relative position of the obstacle in the robot's coordinate frame, and $r_{obs}$ is the radius of the obstacle's circumscribed circle. To enhance generalization in complex real-world environments, the state space is deliberately designed to avoid incorporating any variables related to the global coordinate system. All features are described relative to the robot's own frame, thereby improving adaptability across different environments and initial poses.

In constructing the graph structure, two node types are defined: robot state nodes and obstacle information nodes. To effectively capture the relationships among these nodes, three types of edges are introduced: robot-obstacle edges, obstacle self-loop edges, and robot self-loop edges, as illustrated in Fig. 3. Robot-obstacle edges serve as the primary information channel for modeling spatial interactions between the robot and each obstacle, which are crucial for informed avoidance decisions. The self-loop edges enable each node to retain and update its historical state within the graph neural network, thereby reinforcing temporal continuity and enriching feature expression.

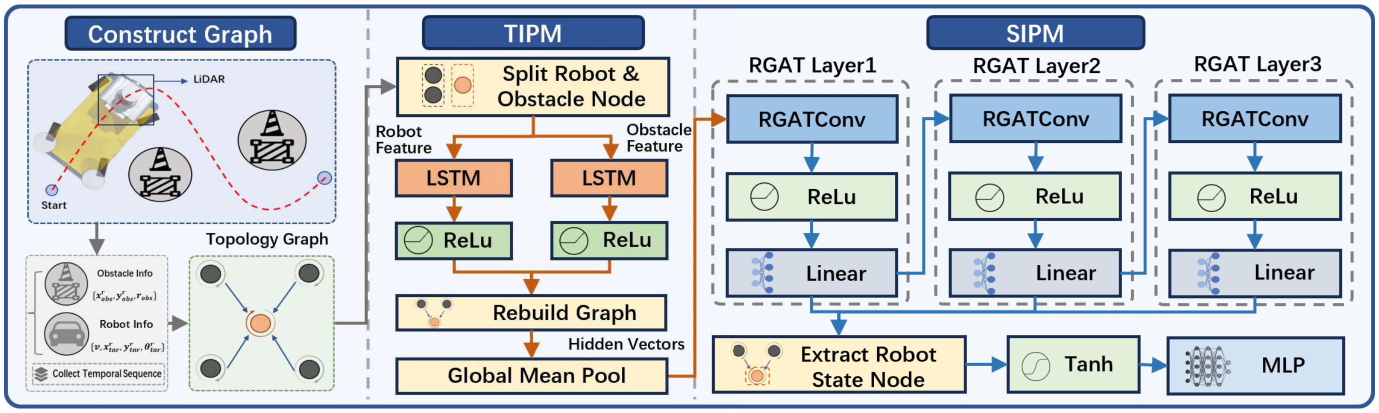


Fig. 4. Overall framework of the proposed spatio-temporal graph network for obstacle-avoidance trajectory planning.

It is worth noting that no inter-obstacle edges are established in this framework, as obstacle-to-obstacle interactions are generally negligible in most avoidance scenarios. The focus of this work is to model the relative relationships between the robot and its surrounding obstacles, rather than capturing the overall structure of the obstacle field. Furthermore, all spatial information is represented in terms of relative positions, minimizing reliance on prior map knowledge.

*2) Data Generation Pipeline:* We systematically construct an optimal trajectory dataset to train an online trajectory planner. To enhance data diversity and improve model robustness, Gaussian noise is introduced to both the initial state of the robot and the obstacle size parameters during the generation of each trajectory. The perturbation process is described as follows:

$$\begin{cases} X(t_0) = X_0 + \xi_i \\ r_{\text{obs}} = M_i R_i \\ M_i \sim \text{Bernoulli}(0.5) \end{cases}, \quad i = 1, \dots, N, \tag{14}$$

where $\mathcal{U}$ denotes a uniform distribution, $\xi_i \sim \mathcal{U}(\xi_{\min}, \xi_{\max})$ represents the stochastic perturbation to the robot's initial state, and $R_i \sim \mathcal{U}(r_{\min}, r_{\max})$ denotes the randomly sampled obstacle radius. $N$ is the number of trajectory samples generated. When $M_i = 0$, the environment is considered obstacle-free and the corresponding obstacle information is zero padded.

To address the optimal trajectory planning problem, the continuous-time optimal control formulation is transformed into an NLP problem. The motion time horizon $[0, t_f]$ is discretized into $N_k$ time nodes $\{t_k\}_{k=1}^{N_k}$, where $t_1 = 0$ and $t_{N_k} = t_f$. The state variables $X(t_k)$ and control inputs $U(t_k)$ are parameterized into discrete sequences $x_k$ and $u_k$, respectively. The system dynamics, path constraints, and objective function are then reformulated into algebraic expressions based on these variables.

Considering the potential convergence issues caused by injected noise, a two-stage optimization framework is adopted. First, PSO algorithm is used to rapidly generate an initial guess. Then, the IPOPT solver is employed to solve the resulting NLP problem, significantly accelerating the dataset construction. The complete trajectory generation procedure is summarized in Algorithm 1.

**Algorithm 1** Optimal Trajectory Dataset Generation

```
1:  **Input**: dataset size $N_d$, dataset $\mathbb{D} = \emptyset$, max iterations number $H_{\max}$ for PSO initial, max iterations number $G_{\max}$ and convergence tolerance $\epsilon$ for IPOPT solver
2:  **Output**: the final trajectory dataset $\mathbb{D}$
3:  Generate time nodes $\{t_k\}_{k=1}^{N_k}$
4:  **while** $\|\mathbb{D}\| < N_d$ **do**
5:      Randomly generate initial noise $\xi_i, M_i, R_i$
6:      Assign noise to initial state and form trajectory optimization model (Eq. (11))
7:      Discretize the model as a NLP problem:
            $\min_{X_k, U_k} \quad J = \sum_{K=0}^{N_k} L(x_k, u_k)dt + \Phi(x_{N_k}, t_k)$
8:      Warm-start via PSO (max iterations $H_{\max}$)
9:      Solve NLP with IPOPT (max iterations $G_{\max}$)
10:     **if** gradient diff $\le \epsilon$ **and** current iteration $\le G_{\max}$ **then**
11:         Extract optimal trajectory $\{x_k^*, u_k^*\}_{k=1}^{N_k}$
12:         Construct heterogeneous graph structure $G_i$ based on trajectory and obstacle information
13:         Add $(\{x_k^*, u_k^*\}, G_i)$ to dataset $\mathbb{D}$
14:     **else**
15:         Discard current trajectory
16:     **end if**
17: **end while**
18: **return** $\mathbb{D}$
```

## B. Network Framework

The proposed network architecture is illustrated in Fig. 4. The overall structure comprises two main modules: the temporal information processing (TIP) module and the spatial information processing (SIP) module. The TIP module is designed to receive the temporal feature sequences of each node. It first applies a linear transformation to map the input into a suitable feature space, and then utilizes LSTM [27] to model the temporal dependencies, producing a set of feature vectors with temporal relevance. The SIP module further processes the node features output by the TIP module using a Relational Graph Attention Network (RGAT) [28]. By incorporating the structural information of the original heterogeneous graph, it leverages a relation-aware attention mechanism to model the spatial interactions between obstacle nodes and robot state nodes, ultimately regressing to the optimal control command for the robot.

*1) Temporal Information Processing Module:* The TIP module is designed to model the temporal dynamic features of nodes in the heterogeneous graph. Specifically, given a heterogeneous graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{R}$ denote the sets of nodes, edges, and relation types respectively, each node $i \in \mathcal{V}$ is associated with a temporal feature sequence $\mathbf{h}_i \in \mathbb{R}^{T \times d}$, where $T$ is the temporal length and $d$ is the feature dimension. The specific construction method of the heterogeneous graph is given in Section III-A.1.

To process these time-series features, TIP first applies type-specific linear projections to unify the feature dimensions of different semantic node types. Specifically, we employ two distinct sets of linear transformation weights for vehicle nodes and obstacle nodes to address the heterogeneity in their information representations. The type-specific transformation is defined as:

$$\tilde{\mathbf{h}}_i^{(t)} = \mathbf{W}^{(type(i))}\mathbf{h}_i^{(t)} + \mathbf{b}^{(type(i))}, \tag{15}$$

where $\mathbf{h}_i^{(t)}$ is the feature vector of node $i$ at time step $t$, $\mathbf{W}^{(type(i))}$ and $\mathbf{b}^{(type(i))}$ are the type-specific weight matrix and bias vector respectively. This step mitigates the interference caused by semantic heterogeneity while preserving high-level information. The transformed sequences are then passed into a LSTM network to capture temporal dependencies and dynamic patterns over time.

RNNs use feedback connections and cyclic structures to extract features from sequence data. LSTM, a variant, introduces gating mechanisms to address gradient vanishing and explosion, outperforming other networks in sequence tasks. Fig. 5 illustrates LSTM's structure. LSTM comprises repeating units with input, forget, and output gates, plus cell states to store long-term information. Gates control information flow via bounded activation functions, enabling selective learning, memory, and forgetting. The update equation of LSTM cell is as follows:

$$\begin{cases} i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\ f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\ \tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\ C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \\ h_t = o_t \odot \tanh\left(C_t\right), \end{cases} \tag{16}$$

where $i_t$, $f_t$, $o_t$ represent the input gate, forget gate and output gate, respectively, $\tilde{C}_t$, $C_t$, $h_t$ represent the hidden state, candidate state and cell state, respectively. Besides, $W_f$, $W_i$, $W_C$, $W_o$ are the weight matrices, and $b_f$, $b_i$, $b_C$, $b_o$ are the bias vectors, which control the information flow in LSTM. Additionally, $\sigma(\bullet)$ represents the sigmoid function, and $\odot$ denotes the element-wise product operation.

For each node $i$, the LSTM processes the temporal sequence $\{\tilde{\mathbf{h}}_i^{(1)}, \tilde{\mathbf{h}}_i^{(2)}, \ldots, \tilde{\mathbf{h}}_i^{(T)}\}$, producing a sequence of hidden vectors. Global average pooling is then applied to aggregate these vectors into a single fixed-dimensional vector, capturing the temporal representation. Notably, the graph structure and connectivity remain unchanged during this process, with only node features updated, enabling effective spatial modeling while preserving temporal dynamics.

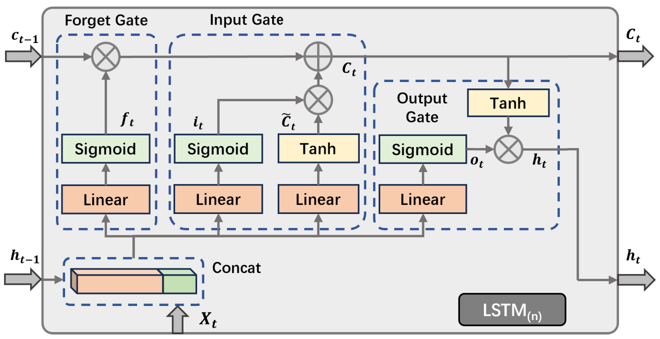


Fig. 5. LSTM network structure.

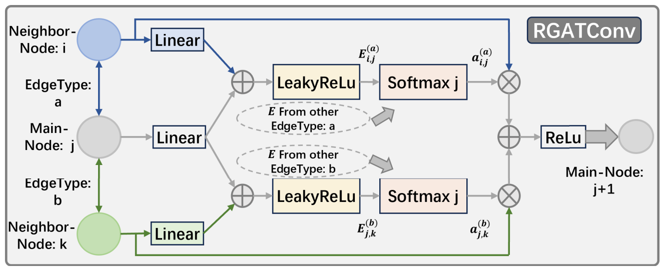


Fig. 6. RGAT network structure.

The output of the TIP module for node $i$ is used to replace the original node feature vector $\mathbf{h}_i$, summarized as:

$$\mathbf{z}_i = \frac{1}{T}\sum_{t=1}^{T} \mathrm{LSTM}\left(\mathbf{W}^{(type(i))}\mathbf{h}_i^{(t)} + \mathbf{b}^{(type(i))}\right). \tag{17}$$

*2) Spatial Information Processing Module:* The main function of the SIP module is to model the spatial structural relationships between different nodes in the heterogeneous graph and generate spatially-aware representation vectors by incorporating the temporal features of the nodes. To effectively model the influence of different edge types in graph-structured data, we adopt the RGAT, which extends the standard Graph Attention Network (GAT) [29] by introducing relation-specific attention mechanisms.

For each relation type $r \in \mathcal{R}$, RGAT computes attention scores independently to weigh the importance of neighboring nodes. Its structure is shown in Fig. 6. For each node $i$, and each of its neighbors $j \in \mathcal{N}_i^r$ under relation $r$, the attention coefficient is computed as:

$$\alpha_{ij}^{(r)} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}_r^\top[\mathbf{W}_r\mathbf{z}_i \,\|\, \mathbf{W}_r\mathbf{z}_j]\right)\right)}{\sum_k \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}_r^\top[\mathbf{W}_r\mathbf{z}_i \,\|\, \mathbf{W}_r\mathbf{z}_k]\right)\right)}, \tag{18}$$

where $k \in \mathcal{N}_i^r$, $\|$ denotes concatenation, and $\mathbf{W}_r$, $\mathbf{a}_r$ are learnable parameters specific to relation type $r$. The updated representation of node $i$ is then computed as the sum of the weighted messages from all relations:

$$\mathbf{z}_i' = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r} \alpha_{ij}^{(r)}\mathbf{W}_r\mathbf{z}_j\right), \tag{19}$$

where $\sigma$ is a non-linear activation function (e.g., ReLU). This mechanism enables RGAT to model the heterogeneity of information propagation across different relation types, leading to more expressive node representations.

For a single RGAT layer, the output feature vector $\mathbf{k}_i$ represents the representation of node $i$ at that layer, which can be formalized based on the RGAT module output $\mathbf{z}_i'$ as:

$$\mathbf{k}_i = \sigma(\mathbf{W}_n \mathbf{z}_i' + \mathbf{b}_n), \tag{20}$$

where $\mathbf{W}_n, \mathbf{b}_n$ denote the linear transformation parameters of the $n$-th RGAT layer. Through the cascaded structure of three RGAT layers, the output of the previous layer serves as the input to the subsequent layer, sequentially generating three hierarchical features $\{\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3\}$. This layered stacking mechanism enhances the expressive power of node representations by progressively integrating spatial dependencies from local to global scales.

Subsequently, we generate the fused feature vector $\mathbf{k}_i'$ for each node through element-wise summation. To avoid interference from redundant information in control decision-making, we only extract the fused feature vector $\mathbf{k}_{i_{\text{robot}}}'$ of the node representing the robot state (denoted as $i_{\text{robot}}$) in the graph, and generate control commands through multilayer perceptron (MLP) regression:

$$\mathbf{u}_t = \text{MLP}\left(\mathbf{k}_{i_{\text{robot}}}'\right), \tag{21}$$

where $\mathbf{u}_t$ represents the control command output.

The TIP module focuses on modeling the historical state of each node, suitable for various temporal feature inputs, while the SIP module independently handles spatial structure modeling, adaptable to different types of heterogeneous graph topologies. This clear modular division facilitates independent development and optimization of each submodule and enhances the network's transferability and scalability across different task scenarios.

### *C. Network Training Policy*

Before training the network, we apply Min-Max normalization to the optimal control values in the dataset based on the upper and lower bounds of the robot control limits [30]. This maps the original values into the range $[-1, 1]$, which improves numerical stability and accelerates model convergence. The normalization is defined as:

$$\hat{u}^* = \frac{2(u^* - u_{\min})}{u_{\max} - u_{\min}} - 1, \tag{22}$$

where $u_{\min}$ and $u_{\max}$ denote the minimum and maximum values of the control constraints, $u^*$ represents the original control value in the dataset, and $\hat{u}^*$ is the normalized output.

During the training phase, the mean squared error (MSE) is employed as the loss function to measure the discrepancy between the predicted and target control values. The loss function is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_j}\sum_{j=1}^{n_j}\left(\hat{u}^*(t_j) - \hat{u}(t_j)\right)^2, \tag{23}$$

where $N$ is the number of training samples, $n_j$ is the length of the trajectory sequence for the $i$-th sample, $\hat{u}(t_j)$ is the predicted control value at time step $t_j$, and $\hat{u}^*(t_j)$ is the corresponding normalized target control value. To optimize the model parameters, we adopt the Adam optimizer [31], which combines the benefits of adaptive learning rate techniques and stochastic gradient descent (SGD), achieving both stable training and improved convergence efficiency.

To further improve the accuracy of trajectory planning, we design and introduce an enhanced training mechanism. The core idea of this strategy is to perform targeted retraining on the updated training dataset. Specifically, during the training process, we continuously monitor the average loss within an epoch. If the loss of certain samples significantly exceeds this average—such as surpassing a predefined multiple threshold—those poorly performing samples are subjected to additional training after the current epoch ends.

**Algorithm 2** Enhanced Training Mechanism

```
1:  Input: Initial training set $\mathcal{D}_{\text{init}}$, base model $\theta_0$, threshold multiplier $\gamma$
2:  Output: the optimal network parameters $\theta^*$
3:  $\theta \leftarrow \theta_0$, $\mathcal{D}_{\text{aug}} \leftarrow \mathcal{D}_{\text{init}}$
4:  for epoch $k = 1$ to $E_{\text{total}}$ do
5:      Record epoch loss sequence $\mathcal{L}_k = \{\}$
6:      for each batch $(x_i, \hat{u}_i^*) \in \mathcal{D}_{\text{aug}}$ do
7:          Forward propagate: $\hat{u}_i \leftarrow f_\theta(x_i)$
8:          Compute loss: $\ell_i \leftarrow \mathcal{L}(\hat{u}_i, \hat{u}_i^*)$
9:          $\mathcal{L}_k$.append($\ell_i$)
10:         Backward propagate to update $\theta$
11:     end for
12:     Compute moving average loss $\bar{\mathcal{L}}_e$
13:     Filter hard samples: $\mathcal{D}_{\text{h}} \leftarrow \{(x_i, \hat{u}_i^*) | \ell_i > \gamma \bar{\ell}_e\}$
14:     if $\mathcal{D}_{\text{h}} \neq \emptyset$ then
15:         Enhanced train the network with $\mathcal{D}_{\text{h}}$
16:     end if
17:     Update dataset: $\mathcal{D}_{\text{aug}} \leftarrow \mathcal{D}_{\text{aug}} \cup \mathcal{D}_{\text{h}}$
18: end for
19: return $\theta^*$
```

In this way, the enhanced training mechanism encourages the network to focus more on hard-to-fit samples, thereby improving the overall robustness and generalization ability of the model. The detailed procedure of this enhanced training mechanism is summarized in Algorithm 2.

### *D. Real-Time Application of the Planner*

The aforementioned dataset construction and model training methods are employed for the offline development of the trajectory planner. Upon completion of offline training, the trained STGN is deployed onto the robot platform as the core component of the real-time trajectory planning module. The complete online deployment workflow is illustrated in Fig. 7.

In practical deployment, the robot acquires environmental information such as the position and shape of surrounding obstacles through perception sensors like LiDAR and cameras, and obtains its own current state via localization devices such as GPS and IMU. To ensure consistency and generalizability of the model input, all such information must be transformed into the local coordinate frame before being integrated into the

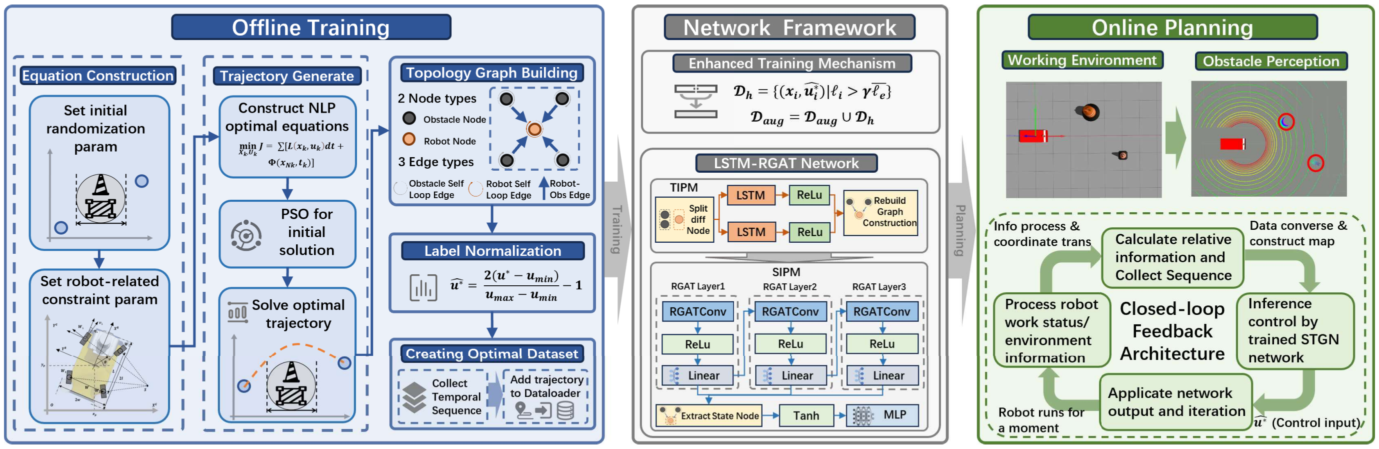


Fig. 7. Pipeline of the proposed method, including training process, network architecture, and deployment in obstacle-avoidance tasks.

temporal heterogeneous graph. This coordinate transformation can be expressed as:

$$\begin{bmatrix} x_{\text{local}} \\ y_{\text{local}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & x_{\text{robot}} \\ -\sin(\theta) & \cos(\theta) & y_{\text{robot}} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{\text{global}} \\ y_{\text{global}} \\ 1 \end{bmatrix}, \tag{24}$$

where $(x_{\text{global}}, y_{\text{global}})$ represents the coordinates of the point to be transformed in the global frame, $(x_{\text{robot}}, y_{\text{robot}}, \theta)$ denotes the robot's position and heading angle in the global frame, and $(x_{\text{local}}, y_{\text{local}})$ is the resulting position in the local coordinate frame. This transformation strategy enables the network to process obstacles from arbitrary directions using a unified input format, significantly enhancing its adaptability and robustness in real-world complex environments.

Unlike conventional trajectory planning methods, which typically adopt a two-stage "plan-then-track" pipeline, the proposed approach employs a stepwise, iterative closed-loop control mechanism. At each time step, the network generates a control command in real time based on the current robot state and surrounding environmental information. After executing the command and updating the robot's state, a new graph is reconstructed and fed back into the network for the next decision. By continuously repeating this process, the system maintains an ongoing response to dynamic environments.

This strategy forms a closed-loop feedback architecture, which offers high robustness and online adaptability, enabling stable trajectory generation in non-static and uncertain scenarios. Notably, although the method is naturally suited for online step-by-step control, it can also generate a complete trajectory via forward integration of the control commands using the robot's kinematic model, thus remaining compatible with traditional planning frameworks.

## IV. Experiment Results and Discussion

This section presents the simulation and real-world experiment results of the online trajectory planning method based on the STGN under various scenarios, including environments without obstacles, with known static obstacles, with sudden static obstacles, and with sudden moving obstacles. In addition, the method's rapid transferability is also validated.

TABLE II
Robot-Related Parameters

| Parameters | Values | Parameters | Values |
|---|---|---|---|
| $W$ | $0.52\ (m)$ | $L$ | $0.68\ (m)$ |
| $2w$ | $0.62\ (m)$ | $2l$ | $1.00\ (m)$ |
| $\lvert v_{\max}\rvert$ | $2\ (m/s)$ | $a_{\max}$ | $5\ (m/s^2)$ |
| $\lvert \eta_{\max}\rvert$ | $\pi/6\ (rad)$ | $\dot{\eta}_{\max}$ | $\pi\ (rad/s)$ |

The experimental results demonstrate the effectiveness and real-time performance of the proposed method in trajectory planning across diverse scenarios.

### A. Experiment Preparation and Related Parameters

To support the proposed online trajectory planning framework, a high-quality dataset of optimal obstacle avoidance trajectories for robots was constructed. In order to formulate the trajectory optimization problem (Eq. (11)), robot-related parameters and task-specific parameters are detailed in Table II and Table III, respectively.

As described in Section III-A, this dataset was generated via a Monte Carlo approach by introducing randomness into both the initial robot state $x_0$ and the obstacle radius $R_{\text{obs}}$. Specifically, random perturbations $\boldsymbol{\xi}_i = [\xi_{px}, \xi_{py}, \xi_\theta, \xi_v]^T$ were applied to the robot's position, orientation, and velocity to enrich the diversity of motion scenarios. These perturbations follow the distributions:

$$\begin{cases} \xi_{px} \sim \mathcal{U}(-1.5, 1.5) \\ \xi_{py} \sim \mathcal{U}(-1, 1) \\ \xi_\theta \sim \mathcal{U}(-\pi/6, \pi/6) \\ \xi_v \sim \mathcal{U}(0, 2) \\ R_{\text{obs}} \sim \mathcal{U}(0.2, 1). \end{cases} \tag{25}$$

Based on the numerical trajectory optimization method presented in Algorithm 1, 1,000 time-optimal trajectories were generated for randomized obstacle-free and single-obstacle scenarios. Additionally, by systematically adjusting the robot's initial state, we simulate various potential collision scenarios, further increasing the dataset's diversity.

TABLE III
OBSTACLE AVOIDANCE TASK PARAMETERS

| Parameters | Values | Parameters | Values |
|---|---|---|---|
| $p_{x_0}$ | $0\ (m)$ | $p_{x_f}$ | $8\ (m)$ |
| $p_{y_0}$ | $0\ (m)$ | $p_{y_f}$ | $0\ (m)$ |
| $\theta_0$ | $0\ (rad)$ | $\theta_f$ | $0\ (rad)$ |
| $v_0$ | $0\ (m/s)$ | $v_f$ | $0\ (m/s)$ |
| $x_{\text{obs}}$ | $4\ (m)$ | $y_{\text{obs}}$ | $0\ (m)$ |

TABLE IV
HYPERPARAMETER SETTINGS FOR NETWORK TRAINING

| Parameters | Configurations |
|---|---|
| Input sequence length | 4 |
| LSTM input size | 64 |
| LSTM hidden size | 128 |
| RGAT hidden size | 128 |
| RGAT attention heads | 1 |
| Batch size for training | 256 |
| Learning rate | 1e-3 |
| Optimizer for training | Adam |
| Number of training epochs | 1000 |

The STGN is designed to learn the mapping between the heterogeneous graph representation $x_{\text{in}}$, which encodes robot–obstacle interaction, and the corresponding optimal control command $u$. The performance of the model is significantly influenced by the network architecture, training setup, and hyperparameters. Therefore, we conducted extensive training and evaluation across multiple configurations to determine the optimal network used in this study. The final set of hyperparameters is summarized in Table IV.

Offline training was performed on an NVIDIA L40 GPU. The real-time inference and control processes were executed on a device equipped with an Intel i7-13700H CPU and an RTX 4060 GPU, running the Ubuntu operating system.

## B. Performance Comparison Experiment

To evaluate the performance of the proposed method, we conducted a comparative study of STGN planner, traditional trajectory optimization methods, DNN-based, LSTM-based, and LSTM-Attention-based [23] trajectory planning methods. For a comprehensive evaluation, we randomly generated 100 trajectories in both obstacle-free and single-obstacle environments, then used Root Mean Square Error (RMSE) to evaluate the network's output and trajectory state variables. The RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\epsilon_{true_i} - \epsilon_i)^2}, \tag{26}$$

where $\epsilon_{true_i}$ represents the true value, $\epsilon_i$ represents the predicted value, and $N$ is the number of data points.

Additionally, to more accurately assess the method's performance in real-world scenarios, we selected the mean and extreme values of the lateral error and longitudinal error as supplementary metrics. Let the robot's position be $(x(t), y(t), \theta(t))$, and the reference trajectory's position be $(x_r(t), y_r(t), \theta_r(t))$. The lateral errors $e_{\text{lat}}(t)$ and longitudinal errors $e_{\text{long}}(t)$ are calculated via a rotation matrix:

$$\begin{bmatrix} e_{\text{long}}(t) \\ e_{\text{lat}}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta_r(t) & \sin\theta_r(t) \\ -\sin\theta_r(t) & \cos\theta_r(t) \end{bmatrix} \cdot \begin{bmatrix} x(t) - x_r(t) \\ y(t) - y_r(t) \end{bmatrix} \tag{27}$$

Table V presents the experimental results with corresponding standard deviations. Our method demonstrates superior performance across most evaluation metrics, achieving both lower mean errors and reduced variance compared to baseline approaches.

To more intuitively demonstrate the effectiveness of the proposed method, several representative scenarios were selected for visualization. Fig. 8 illustrates the state evolution and control input curves of the proposed method under a representative single-obstacle scenario. Subsequently, Fig. 9 compares the obstacle avoidance trajectories generated by different methods, where Fig. 9(b) corresponds to the scenario shown in Fig. 8. In the obstacle-free scenario, all three methods were able to generate smooth trajectories toward the target point; however, the trajectory produced by our method showed the closest alignment with the optimal trajectory. In the single-obstacle scenario, the performance differences among the methods became more pronounced, with our approach still exhibiting superior proximity and stability.

In the experiments, the DNN-based method frequently exhibited trajectory deviations or even penetrations through obstacles, indicating a clear risk of collision and limited capability in spatial perception and control. The LSTM-based approach showed steady-state errors during trajectory tracking, especially when turning, where it struggled to closely follow the optimal path—highlighting its insufficient ability to model dynamic changes. Although the LSTM-Attention method performed well in terms of state tracking, it exhibited slight lag in scenarios involving rapidly changing control commands. In contrast, the proposed STGN method consistently achieved the best performance across nearly all evaluation metrics. This demonstrates that by integrating temporal modeling with graph-structured spatial representation, the STGN framework can produce more accurate state estimation and more reliable control decisions under obstacle disturbances.

To further validate the effectiveness of the proposed graph module, we conducted performance comparison experiments between the STGN method and the homogeneous graph processing modules commonly used in the trajectory planning field [26], [32], [33], such as GCN [34] and GAT [29]. The experimental results, along with their corresponding standard deviations, are presented in Table VI. The results demonstrate that the STGN method significantly outperforms GCN and GAT across all evaluation metrics, exhibiting superior performance. Notably, while homogeneous graph modules retain some generalization capability for graph structures, their accuracy is slightly inferior to that of the LSTM-Attention method. This highlights that, in the heterogeneous interaction scenario of obstacle avoidance tasks, traditional homogeneous graph modules struggle to effectively extract spatial features and their interactions within the environment. In contrast, by integrating temporal modeling with spatial representations of heterogeneous graph structures, STGN efficiently captures

TABLE V
COMPREHENSIVE COMPARISON OF TRAJECTORY ERROR AND STATE ERROR

| Method | Trajectory Errors (mean ± std) ($\times 10^{-2}$) | | | | State RMSE (mean ± std) ($\times 10^{-2}$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Longitudinal (m) | | Lateral (m) | | X (m) | Y (m) | $\theta$ (rad) | V (m/s) | a (m/s²) | $\omega$ (rad/s) |
| | Mean | Max | Mean | Max | | | | | | |
| DNN | 1.93±0.76 | 6.33±3.36 | 1.99±2.21 | 4.23±9.03 | 7.06±8.17 | 4.37±4.52 | 4.22±3.32 | 4.95±3.09 | 2.75±2.12 | 6.77±5.22 |
| LSTM | 0.97±0.81 | 3.13±1.79 | 1.39±2.00 | 3.85±5.08 | 6.21±7.96 | 4.58±3.67 | 4.39±2.41 | 3.27±2.59 | 2.87±1.46 | 6.59±3.66 |
| LSTM-ATT | 0.97±1.04 | 3.18±1.49 | 0.89±1.07 | 2.55±2.23 | 5.73±7.74 | 3.92±3.76 | 3.89±3.31 | 3.14±3.26 | 3.23±2.43 | 5.18±2.91 |
| STGN | **0.81±0.89** | **2.62±1.42** | **0.81±0.87** | **2.38±1.34** | **5.34±6.95** | **4.24±3.72** | **3.78±2.99** | **2.82±2.55** | **2.71±2.38** | **4.80±3.21** |

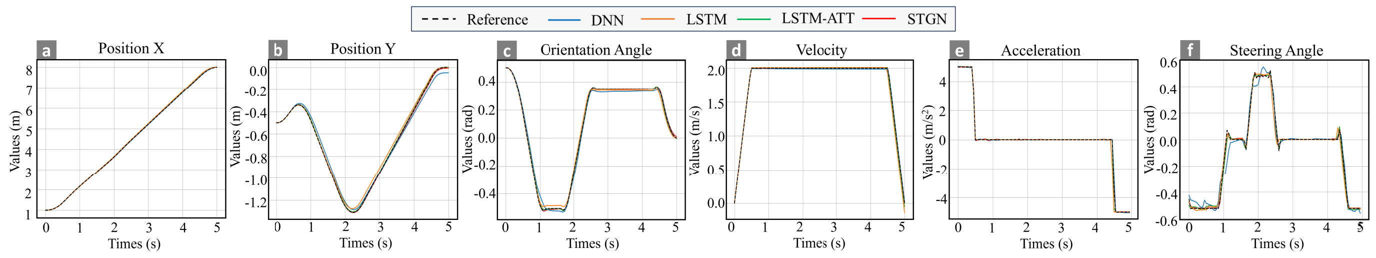


Fig. 8. Visualization of robot's states and control inputs corresponding to the planned trajectories. (a-d) Robot states. (e-f) Control inputs.

TABLE VI
PERFORMANCE COMPARISON OF GRAPH-BASED NETWORK METHODS

| Method | Trajectory Errors (mean ± std) ($\times 10^{-2}$) | | | | State RMSE (mean ± std) ($\times 10^{-2}$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Longitudinal (m) | | Lateral (m) | | X (m) | Y (m) | $\theta$ (rad) | V (m/s) | a (m/s²) | $\omega$ (rad/s) |
| | Mean | Max | Mean | Max | | | | | | |
| LSTM-GCN | 1.06±1.24 | 3.34±2.26 | 1.58±2.16 | 3.65±4.41 | 5.43±8.13 | 4.23±3.98 | 4.37±3.76 | 3.31±2.86 | 3.21±1.78 | 5.43±3.83 |
| LSTM-GAT | 0.88±0.88 | 2.58±2.24 | 1.19±1.35 | 2.66±3.15 | 5.03±7.04 | 3.79±4.41 | 3.97±3.63 | 3.11±2.43 | 3.14±1.61 | 5.04±3.23 |
| STGN | **0.81±0.89** | **2.62±1.42** | **0.81±0.87** | **2.38±1.34** | **5.34±6.95** | **4.24±3.72** | **3.78±2.99** | **2.82±2.55** | **2.71±2.38** | **4.80±3.21** |

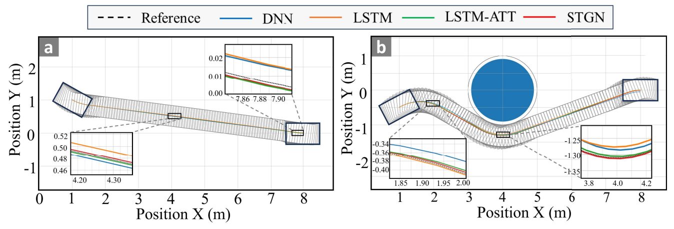


Fig. 9. Trajectory planning results in different environments. (a) Obstacle-free scenario. (b) With obstacles scenario.

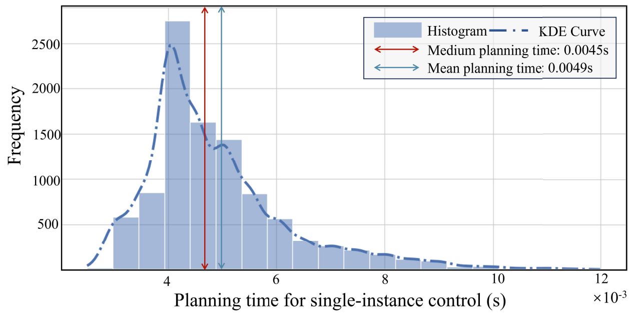


Fig. 10. Statistical analysis of single-instance control time using the STGN trajectory planner.

complex environmental dynamics, enabling more accurate state estimation and more robust control decisions.

## C. Real-Time Analysis of the Planner

To evaluate the real-time performance of the proposed obstacle avoidance trajectory planning method, we conducted a statistical analysis over 100 complete planning cycles using STGN, with each cycle consisting of 100 trajectory points. Fig. 10 illustrates the distribution of single-step planning times. The reported timing includes the construction of the robot's local coordinate frame, transformation of environmental information into this frame, neural network inference to generate control commands, numerical integration to compute the robot's next state, and the subsequent state update. While the method incurs extra computational cost from graph construction and network complexity, experiments confirm that its planning latency is low enough to support real-time control at 200 Hz. This indicates that the method maintains satisfactory real-time performance and is suitable for most online trajectory planning applications.

In addition, we compared the performance of the proposed STGN planner with that of optimization-based planners in terms of the time required to generate complete trajectories. A statistical analysis was conducted, and the results are shown

TABLE VII
PLANNING TIME FOR COMPLETE TRAJECTORY GENERATION

| Method | Planning Time (s) | | |
|---|---|---|---|
| | Mean | Median | Max |
| Optimization (no initial guess) | 4.1242 | 3.9856 | 8.7634 |
| Optimization (Hybrid A* initialization) | 1.0636 | 1.0284 | 3.2059 |
| STGN Planner | **0.4978** | **0.4904** | **0.5412** |

in Table VII. From the data, it can be observed that the optimization-based planner without an initial guess requires the longest average planning time. This indicates that the optimization process is computationally intensive in the absence of prior information, likely due to the need for extensive search and iterative refinement to obtain a feasible trajectory.

In contrast, the optimization-based planner with an initial guess significantly reduces planning time. This improvement can be attributed to the ability of the initial guess to constrain the search space and accelerate convergence. For the offline dataset construction phase, the initial guess is generated using Particle Swarm Optimization (PSO), which offers high-quality solutions but incurs higher computational cost. In experiment, however, the initial guess is obtained using the Hybrid A* algorithm, which generates kinematically feasible trajectories within a short time (typically $\leq 0.3$ s). Although the solution quality of Hybrid A* may be inferior to PSO, it provides a suitable warm start for subsequent nonlinear optimization.

Notably, the proposed STGN planner achieves the best performance in terms of planning efficiency. It takes an average of only 0.4978 s to generate a complete trajectory, representing a 53.2% reduction in planning time compared to the optimization-based planner with an initial guess. These results further validate the real-time capability of the proposed STGN planner for trajectory planning tasks, demonstrating its potential for fast and reliable operation in dynamic environments.

## D. *Obstacle Avoidance Results for Sudden Obstacles*

To thoroughly evaluate the effectiveness and adaptability of the proposed trajectory planning method under unexpected conditions, a series of sudden obstacle avoidance experiments were conducted. These experiments simulate realistic scenarios in which a robot encounters unforeseen static or dynamic obstacles during operation—obstacles that were not present during the initial planning phase. The objective is to assess the proposed method's capability to respond promptly to sudden disturbances, ensure safe obstacle avoidance, and maintain stability and robustness in task execution.

*1) Results for Static Sudden Obstacles:* To further validate the effectiveness of the proposed trajectory planning method, a test scenario was designed as illustrated in Fig. 11. In this scenario, robot initially follows a predefined, obstacle-free trajectory. However, during execution, an unexpected obstacle suddenly appears in the robot's path. This setup is intended to evaluate whether the robot can respond promptly and perform safe, real-time obstacle avoidance maneuvers.

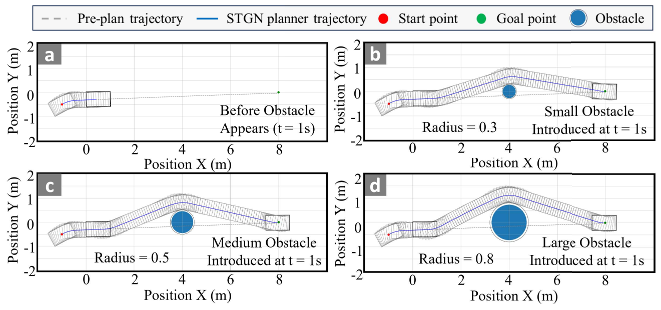


Fig. 11. Obstacle avoidance for sudden obstacles of varying sizes (The initial state of robot is [-1, -0.5, 0.5, 0]). (a) Pre-planned trajectory without obstacles. (b-d) Re-planned trajectories with obstacles of different radius.

After the robot follows the original planned trajectory for 1s, an obstacle of varying size abruptly emerges at the coordinate [4, 0], posing a potential collision threat. At this moment, the proposed trajectory planner is triggered to generate a new collision-free path in real time. The resulting trajectory demonstrates that the robot successfully avoids the unexpected obstacle and continues toward its goal. These results confirm the system's ability to react quickly and safely under dynamic and uncertain conditions.

*2) Results for Moving Sudden Obstacles:* In real-world scenarios, sudden obstacles are not always static; common obstacles such as pedestrians often continue moving after appearing on the pre-planned trajectory. To further evaluate the proposed method's capability to handle moving obstacles, we extended the static obstacle experiments by introducing sudden obstacles that move along either the x-axis or the y-axis. Considering that any velocity vector in a two-dimensional plane can be decomposed into two orthogonal components, this setup effectively covers typical obstacle motion patterns.

The experimental results are shown in Fig. 12. When the robot detects a sudden obstacle moving along the x-axis, the proposed STGN trajectory planner autonomously adjusts the path by increasing the robot's displacement along the x-axis, successfully avoiding the obstacle. Similarly, when an obstacle appears and moves along the y-axis, the robot autonomously steers away from the obstacle's moving direction and increases its lateral displacement to ensure safety. These results further validate the effectiveness and adaptability of the proposed method in handling sudden moving obstacles under dynamic conditions.

## E. *Generalization Experiments With Multiple Obstacles*

Traditional deep learning-based planners often require fixed-size inputs and struggle to adapt when the environment changes, such as moving from single to multiple obstacles. These models typically focus only on the nearest obstacle, limiting both safety and optimality. In contrast, our graph-based approach treats obstacles as graph nodes, allowing flexible input sizes and enabling safe, efficient adaptation to varying environments.

To further validate the transferability of our method, we extended the previous model by generating a new small-scale optimal trajectory dataset involving two obstacles. The

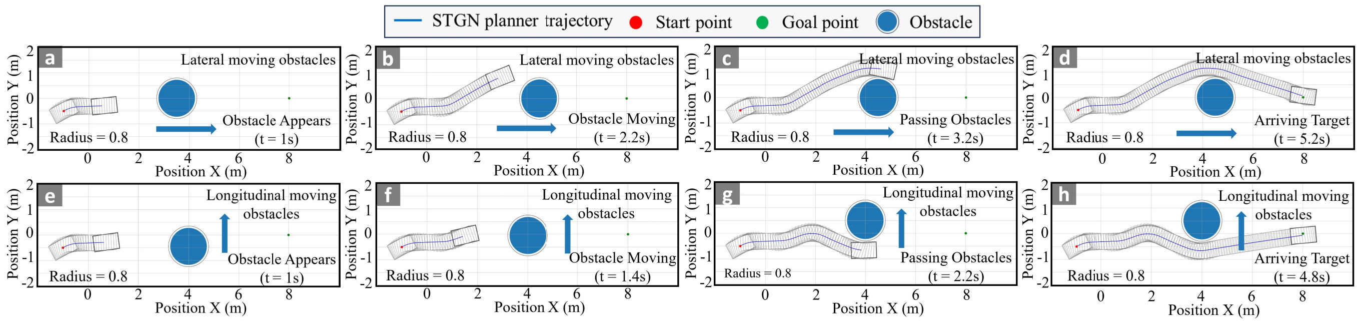


Fig. 12. Obstacle avoidance for sudden moving obstacles of varying sides (The initial state of robot is [-1, -0.5, 0.5, 0]). (a-d) Horizontally moving obstacle scene. (e-h) Vertically moving obstacle scene.

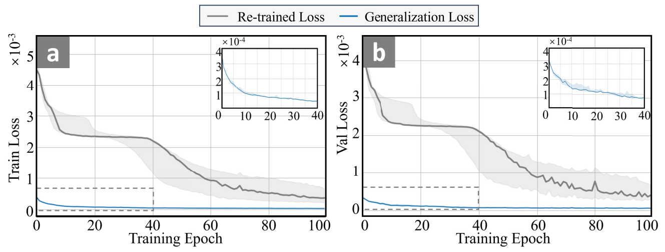


Fig. 13. Training convergence behavior with and without transfer learning. (a) Train loss curves. (b) Test loss curves.

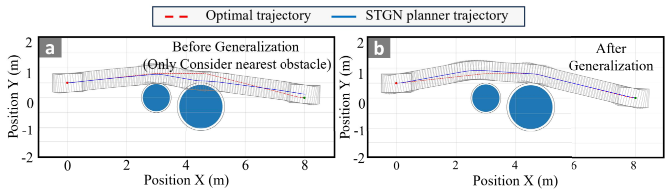


Fig. 14. Comparison of transferred and baseline networks in handling multiple obstacles. (a) Before generalization. (b) After generalization.

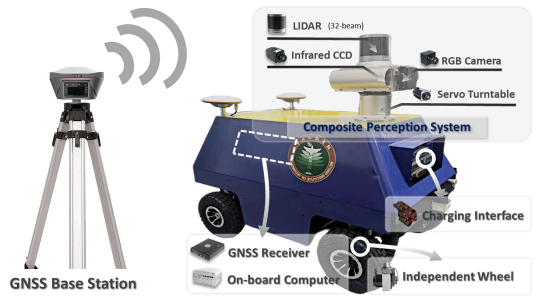


Fig. 15. Illustration of the robot experiment platform.

robot's initialization parameters and the obstacle radius range remained consistent with the earlier setup. Specifically, the center of obstacle 1 was randomly placed with its x-coordinate in $[2.5, 4]$ m and y-coordinate in $[-1.5, 1.5]$ m, while the center of obstacle 2 was placed with its x-coordinate in $[4, 5.5]$ m and y-coordinate in $[-1.5, 1.5]$ m. This dataset contains 200 trajectories, providing a basis for evaluating the method's adaptability to multi-obstacle scenarios.

Fig. 13 illustrates the convergence behavior of the network during the transfer learning process. Compared to training the model from scratch, applying transfer learning enables the network to reach convergence within a significantly smaller number of epochs. This highlights the efficiency of the proposed framework in adapting to new, more complex multi-obstacle environments without requiring extensive retraining.

Fig. 14 compares the trajectory planning performance of the transferred network with that of a baseline network that only considers the nearest obstacle for avoidance. Although a slight degradation in local optimality is observed after transfer, the network successfully takes multiple obstacles into account simultaneously, generating trajectories that maintain safe distances from all surrounding obstacles. In contrast, the baseline network, constrained by its nearest-obstacle-only strategy, often exhibits suboptimal behavior, such as unnecessary detours and reduced clearance. Furthermore, when a new obstacle suddenly appears after bypassing an initial obstacle, the baseline network may even fail to react appropriately, leading to potential collisions.

These results demonstrate that the proposed graph-based method, empowered by transfer learning, not only significantly accelerates the adaptation process but also ensures a high level of safety and optimality in complex, dynamic environments. The ability to efficiently migrate to multi-obstacle scenarios further confirms the generalization and robustness of our approach.

## F. Results of Real-World Experiments

*1) Experimental Platform:* To validate the proposed trajectory planning method in real-world scenarios, we conducted experiments on a real-world robot platform. The robot's onboard computer is equipped with an Intel Core i7-13700H CPU, a GeForce GTX 2060 GPU, and 16 GB of RAM. It runs the Ubuntu operating system with the ROS middleware, and is configured with PyTorch 2.1.0 and CUDA 12.7, enabling efficient execution of control commands.

To ensure consistency between real-world and simulation results, key parameters of the real robot are kept identical to those in the simulation model, as detailed in Table II. In addition, the robot has a wheel radius of 0.13 m and a total weight of 230 kg. The robotic platform used in our

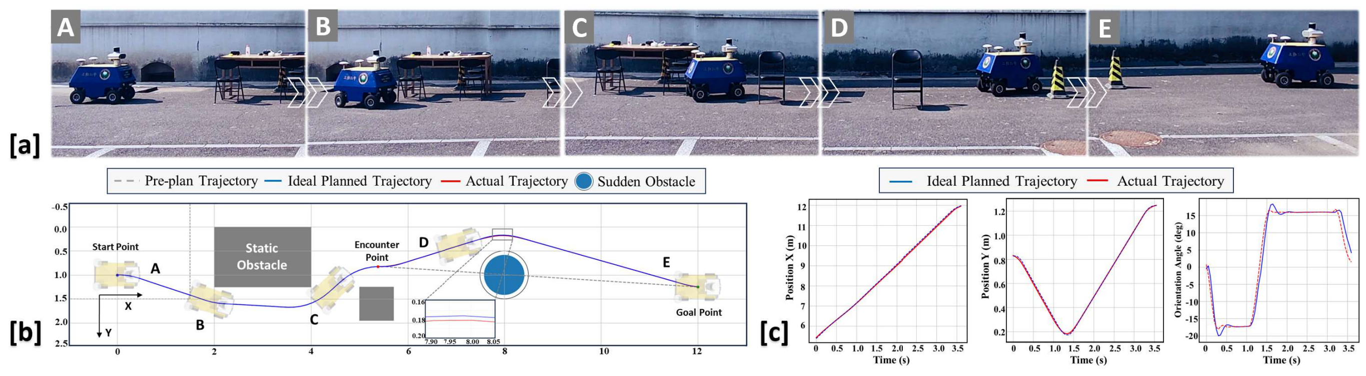


Fig. 16. Single obstacle avoidance experiment: (a) Real-world navigation. (b) Visualization of ideal/actual trajectory. (c) State comparison curves.

experiments is shown in Fig. 15. To prevent lateral slipping, the wheels are fitted with anti-slip tires.

During the experiments, a partial map of the static environment was pre-built using SLAM techniques. For handling sudden obstacles, real-time point cloud data were collected using the onboard LiDAR. To simulate the robot's perception range, only points within a 5-meter radius were retained. These points were further processed with Kalman filtering and motion compensation using fused IMU data. The processed point cloud was then compared with the static obstacle map to identify newly emerging obstacles.

After filtering out known regions, the remaining point cloud is clustered, and a minimum bounding circle is fitted to each new obstacle. To ensure safety, we add an extra $0.1\times$ inflation to the radius of the fitted circle. The LiDAR used in the experiment operates at a frequency of 20 Hz, enabling rapid detection of sudden obstacles. Global positioning is provided by a real-time kinematic (RTK) differential GPS module with an accuracy of $\pm 0.8$ cm. Furthermore, IMU data were fused to suppress localization drift during operation.

*2) Single Sudden Obstacles:* To validate the proposed method's ability to handle sudden static obstacles, a physical experiment was conducted using a predefined trajectory in a controlled environment, as illustrated in Fig. 16(a). The robot initially follows the planned path while bypassing a static obstacle. As the robot passed the original obstacle and reached the red dot location—about to exit the blind zone—we placed a new obstacle directly in its path to simulate a sudden appearance. To avoid sensor interference from human movement, the obstacle was pre-positioned, and the perception and avoidance modules were activated only when the robot reached the designated location.

Fig. 16(b) shows a comparison between the pre-planned trajectory, the trajectory planned under ideal conditions, and the actual trajectory followed by the robot. Despite slight deviations due to mechanical execution, the robot successfully re-routed and avoided the obstacle, maintaining a collision-free trajectory. Fig. 16(c) presents time-series plots of the robot's position and heading. The actual states closely follow the planned values, demonstrating the system's planning accuracy and robustness to disturbances introduced by unexpected environmental changes.

TABLE VIII
TRAJECTORY ERROR STATISTICS FOR OBSTACLE AVOIDANCE SCENARIOS

| Scenario | Longitudinal (m) | | Lateral (m) | | Theta (rad) | |
|---|---|---|---|---|---|---|
| | mean | max | mean | max | mean | max |
| Single Obstacle | 0.008 | 0.017 | 0.018 | 0.034 | 0.047 | 0.066 |
| Dual Obstacles | 0.011 | 0.021 | 0.038 | 0.068 | 0.040 | 0.083 |
| Multi Obstacles | 0.026 | 0.054 | 0.041 | 0.086 | 0.051 | 0.143 |

*3) Dual Sudden Obstacles:* In this scenario, two obstacles were placed at different positions along the robot's path, with partial occlusion between them, simulating a more complex obstacle distribution. The initial state of the robot was the same as in previous experiments, but this time, it had to consider the impact of both obstacles when entering the critical region. During the experiment, the LiDAR accurately captured the outlines of both obstacles. The proposed graph-structured network automatically constructed a relational graph of the obstacles, assessed their combined influence on the trajectory, and generated a smooth path that avoided them both. Fig. 17(a) captures the robot's actual avoidance process, while Fig. 17(b) shows a comparison between the ideal generated trajectory and the executed trajectory. As seen in Fig. 17(c), the robot maintained stable performance throughout the maneuver, further demonstrating the method's robustness and generalizability.

*4) Multiple Sudden Obstacles:* To further evaluate the algorithm's planning performance in environments with multiple obstacles, we designed an experimental setup featuring several densely placed static obstacles. The robot was required to navigate from a start point to a target point by autonomously avoiding these obstacles without any pre-defined path. Fig. 18(a) illustrates the obstacle avoidance process, Fig. 18(b) shows the comparison between the ideal path and the actual execution trajectory, and Fig. 18(c) presents the time series of the robot's position and heading during navigation. The results demonstrate that even in complex, cluttered environments, the proposed method can generate reasonable and effective navigation trajectories, exhibiting strong robustness and generalization capabilities.

*5) Planning Error Analysis:* To further evaluate the accuracy and robustness of the proposed trajectory planning and tracking system, we conducted a quantitative analysis of the deviation between the ideal trajectory and the actual executed

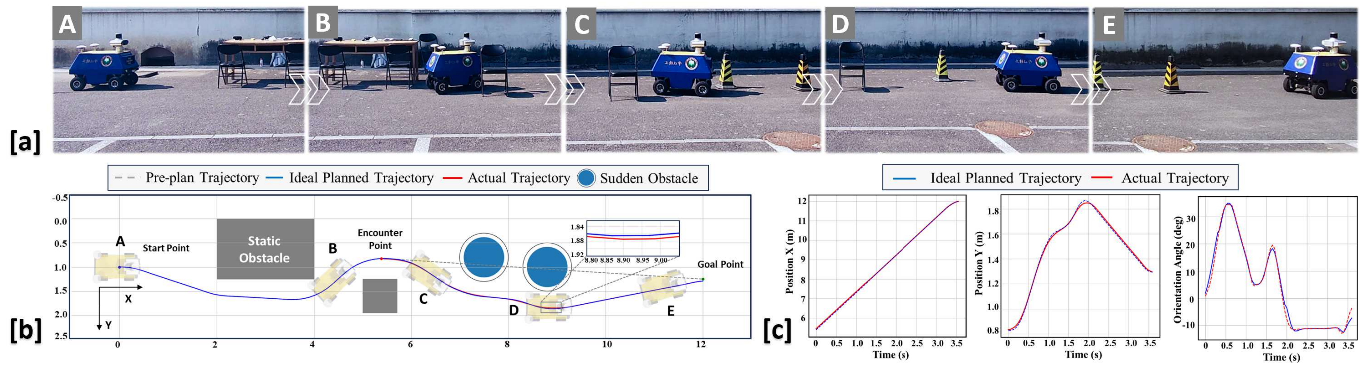


Fig. 17. Dual obstacle avoidance experiment: (a) Real-world navigation. (b) Visualization of ideal/actual trajectory. (c) State comparison curves.

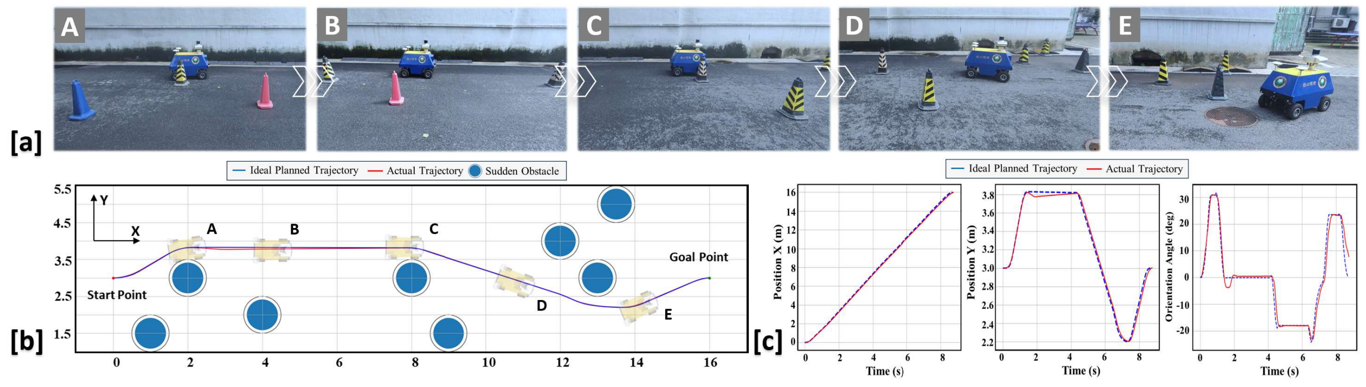


Fig. 18. Multiple obstacle avoidance experiment: (a) Real-world navigation. (b) Visualization of ideal/actual trajectory. (c) State comparison curves.

trajectory. Since the time steps of the two trajectories may not align, we employed interpolation to extract reference points from the ideal trajectory corresponding to the timestamps of the actual trajectory for error evaluation. Table VIII presents the error metrics across three experimental scenarios.

In the single-obstacle and double-obstacle scenarios, the environment is relatively simple and the obstacle perception remains stable. As a result, the trajectory deviation is minor, with most of the error attributed to the robot's execution inaccuracy. In the multi-obstacle scenario, however, the robot's limited sensing radius prevents it from perceiving the entire environment at once. As the robot moves forward, it continually updates its planned trajectory based on newly perceived obstacles. This leads to larger deviations compared to the ideal trajectory generated under fully known obstacle conditions. Nevertheless, the resulting trajectory still ensures obstacle avoidance and guides the robot efficiently to the goal. These results demonstrate the system's strong adaptability to different sudden environments, as well as its potential for real-world deployment.

## V. Conclusion

This article proposes a real-time obstacle avoidance trajectory planning method for robots by integrating temporal modeling and graph attention mechanisms, aiming to ensure rapid and robust responses in dynamic environments with sudden and moving obstacles. Offline, a heterogeneous graph representation is constructed to uniformly model robot and obstacle states, and a spatio-temporal graph network is trained using a dataset of high-quality obstacle avoidance trajectories. This network captures both temporal dynamics and spatial interactions, enabling the learning of an effective mapping from environmental states to control commands. Online, the trained model is deployed to perceive real-time states, generate avoidance trajectories quickly. Empowered by transfer learning, the proposed graph-based method significantly accelerates adaptation to multi-obstacle scenarios, demonstrating strong potential for robust and efficient trajectory planning.

Future work will focus on enhancing the robustness and adaptability of the proposed planning framework. We plan to improve obstacle representation by adopting polygonal or deep feature encoding methods instead of the bounding circle approach, aiming to reduce overly conservative paths and better handle irregularly shaped obstacles. Additionally, we intend to incorporate error modeling into the planning process by integrating control uncertainty quantification to address vehicle execution errors, thereby improving safety in constrained or high-risk scenarios. Another key direction is to extend trajectory generation from kinematic to dynamic models, accounting for vehicle mass, inertia, and robot-ground friction to mitigate sideslip in extreme scenarios [35], ensuring the physical feasibility of generated trajectories.

## References

[1] Y. Zhang, Z. Ai, J. Chen, T. You, C. Du, and L. Deng, "Energy-saving optimization and control of autonomous electric vehicles with considering multiconstraints," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 10869–10881, Oct. 2022.

[2] Y. Xu, R. Bao, L. Zhang, J. Wang, and S. Wang, "Embodied intelligence in RO/RO logistic terminal: Autonomous intelligent transportation robot architecture," *Sci. China Inf. Sci.*, vol. 68, no. 5, May 2025, Art. no. 150210.

[3] X. Liu, Y. Wang, K. Jiang, Z. Zhou, K. Nam, and C. Yin, "Interactive trajectory prediction using a driving risk map-integrated deep learning method for surrounding vehicles on highways," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 19076–19087, Oct. 2022.

[4] B. Paden, M. Cáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Vehicles*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[5] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[6] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Feb. 1968.

[7] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA: Anytime A with provable bounds on sub-optimality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 16, Aug. 2003, pp. 767–774.

[8] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proc. AAAI Workshop*, vol. 1001, no. 48105, 2008, pp. 18–80.

[9] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Dept. Comput. Sci., Iowa State Univ., Ames, IA, USA, Tech. Rep. 9811, 1998.

[10] 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.

[11] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Apr. 1996.

[12] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2011, pp. 2640–2645.

[13] D. J. Webb and J. van den Berg, "Kinodynamic RRT: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 5054–5061.

[14] J. Ortiz-Haro, W. Hönig, V. N. Hartmann, and M. Toussaint, "IDb—A: Iterative search and optimization for optimal kinodynamic motion planning," *IEEE Trans. Robot.*, vol. 41, pp. 2031–2049, 2025.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, Perth, WA, Australia, 1995, pp. 1942–1948.

[16] R. Chai, A. Tsourdos, A. Savvaris, S. Chai, and Y. Xia, "Two-stage trajectory optimization for autonomous ground vehicles parking maneuver," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 3899–3909, Jul. 2019.

[17] R. Zhang, R. Chai, S. Chai, Y. Xia, and A. Tsourdos, "Design and practical implementation of a high efficiency two-layer trajectory planning method for AGV," *IEEE Trans. Ind. Electron.*, vol. 71, no. 2, pp. 1811–1822, Feb. 2024.

[18] J. Choi, G. Lee, and C. Lee, "Reinforcement learning-based dynamic obstacle avoidance and integration of path planning," *Intell. Service Robot.*, vol. 14, no. 5, pp. 663–677, Nov. 2021.

[19] K. Garmon and Y. Wang, "Vision based leader-follower control of wheeled mobile robots using reinforcement learning and deep learning," in *Proc. 9th Int. Symp. Syst. Secur., Saf., Rel. (ISSSR)*, Jun. 2023, pp. 440–441.

[20] 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.

[21] R. Chai, A. Tsourdos, A. Savvaris, S. Chai, Y. Xia, and C. L. P. Chen, "Design and implementation of deep neural network-based control for automatic parking maneuver process," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 4, pp. 1400–1413, Apr. 2022.

[22] R. Chai, D. Liu, T. Liu, A. Tsourdos, Y. Xia, and S. Chai, "Deep learning-based trajectory planning and control for autonomous ground vehicle parking maneuver," *IEEE Trans. Autom. Sci. Eng. (from July 2004)*, vol. 20, no. 3, pp. 1633–1647, Jul. 2023.

[23] Z. Xing, R. Chai, K. Chen, Y. Xia, and S. Chai, "Online trajectory planning method for autonomous ground vehicles confronting sudden and moving obstacles based on LSTM-attention network," *IEEE Trans. Cybern.*, vol. 55, no. 1, pp. 421–435, Jan. 2025.

[24] J. Liu and M. Yu, "Deep learning-based autopilot vehicle trajectory planning," in *Proc. 8th Int. Conf. Intell. Comput. Signal Process. (ICSP)*, Apr. 2023, pp. 929–933.

[25] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

[26] C. Chen, X. Chen, Y. Yang, and P. Hang, "Sparse attention graph convolution network for vehicle trajectory prediction," *IEEE Trans. Veh. Technol.*, vol. 73, no. 12, pp. 18294–18306, Dec. 2024.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[28] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational graph attention networks," 2019, *arXiv:1904.05811*.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lió, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[30] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Deep adaptive input normalization for time series forecasting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3760–3765, Sep. 2020.

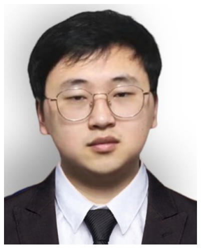
[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[32] Q. Dong, Z. Yan, K. Nakano, X. Ji, and Y. Liu, "Graph-based scenario-adaptive lane-changing trajectory planning for autonomous driving," *IEEE Robot. Autom. Lett.*, vol. 8, no. 9, pp. 5688–5695, Sep. 2023.

[33] Y. Wu, W. Liao, W. Yu, G. Zeng, Y. Shang, and X. L. Dong, "Multi-task learning for ship trajectory prediction and motion planning via node relationship modeling," *IEEE Trans. Intell. Transp. Syst.*, vol. 26, no. 9, pp. 13051–13064, Sep. 2025.

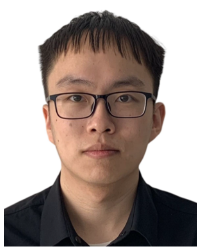
[34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[35] A. Abubakar, Y. Zweiri, A. Haddad, M. Yakubu, R. Alhammadi, and L. Seneviratne, "Physics-informed LSTM-based delay compensation framework for teleoperated UGVs," 2024, *arXiv:2402.16587*.



**Runjiao Bao** received the B.Eng. degree from Jilin University, China, in 2023. He is currently pursuing the M.S. degree in control science and engineering with the State Key Laboratory of Intelligent Control and Decision of Complex Systems, School of Automation, Beijing Institute of Technology, China. He is a member with the State Key Laboratory of Intelligent Control and Decision of Complex Systems, School of Automation, Beijing Institute of Technology. His research interests include trajectory planning and control using deep reinforcement learning and path planning using geometric optimization methods.

**Yongkang Xu** received the M.Sc. degree in mechatronic engineering from Central South University, Changsha, China, in 2021. He is pursuing the Ph.D. degree in control science and engineering with the State Key Laboratory of Intelligent Control and Decision of Complex Systems, School of Automation, Beijing Institute of Technology, China, where he is participating in the air-ground collaborative robots. He is a member with State Key Laboratory of Intelligent Control and Decision of Complex Systems, School of Automation, Beijing Institute of Technology. His research interests include environmental awareness, mobile robotics, motion control, neural networks, and autonomous navigation.



**Chenhao Wang** received the B.S. degree in automation from the School of Automation, Beijing Institute of Technology, Beijing, China, in 2023. He is currently pursuing the M.S. degree in control science and engineering with Beijing Institute of Technology. His current research interests include signal processing, deep learning, transfer learning, and intelligent fault diagnosis.

**Tianwei Niu** received the M.S. degree in mechanical engineering from the University of Science and Technology Beijing. He is currently pursuing the Ph.D. degree with the School of Automation, Beijing Institute of Technology, China. He is a member of the State Key Laboratory of Intelligent Control and Decision of Complex Systems. His research interests include localization and motion planning for field robots.

**Junzheng Wang** received the Ph.D. degree in control science and engineering from Beijing Institute of Technology, Beijing, China, in 1994. He is the Deputy Director of the State Key Laboratory of Intelligent Control and Decision of Complex Systems, the Director of the Key Laboratory of Servo Motion System Drive and Control, and the Dean of the Graduate School of Beijing Institute of Technology, where he is a Professor and a Ph.D. Supervisor. His current research interests include motion control, electric hydraulic servo system, and dynamic target detection and tracking based on image technology. He is the Standing Director of Chinese Association of Automation, and a Senior Member of Chinese Mechanical Engineering Society and Chinese Society for Measurement. He received the Second Award from the National Scientific and Technological Progress (No.1) of China.

**Shoukun Wang** received the B.S., M.S., and Ph.D. degrees from the Department of Automation, Beijing Institute of Technology, Beijing, China, in 1999, 2002, 2004, respectively. He has been entered with the Department of Electronics and Computer Engineering, Purdue University, West Lafayette, IN, USA, as a Visiting Scholar. He has been teaching with the School of Automation, Beijing Institute of Technology, since 2004. He has participated in over 30 scientific research projects since 2001, which mainly belong to measurement and servo control. His main research interests include electrical-hydraulic algorithms, robot locomotion control, and visual servo. His research interests include sensors, measurement, and electrohydraulic control.