2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

InBi-RRT: Incremental Bidirectional Tree based Real-Time Path Planning/Replanning in Unknown Non-Convex Environments

Bo Cui[1], Yang Li[2,*], Weisheng Yan[1], ZhanWei Yang[1], Ao Feng[1] and Rongxin Cui[1,*]

***Abstract*— Real-time path planning in unknown non-convex environments is challenging, as obstacle updates can invalidate existing paths while narrow passages restrict feasible connectivity. This paper presents InBi-RRT, an incremental bidirectional tree-based framework that grows a reverse tree from the goal and maintains a reusable forward tree from the start. When the current path becomes invalid, a cost-guided expansion selectively extends the forward tree to establish collision-free connections with the reverse tree, followed by backtracking and lightweight path optimization for efficient repair. Simulation results in unknown and non-convex scenarios demonstrate that InBi-RRT achieves significantly faster replanning than baseline methods, being up to 5.5× faster than RT-RRT and 22× faster than RRT^{X}, with paths up to 19.8% shorter than RRT^{X} under the same sample count. Furthermore, real-world experiments in an indoor maze-like environment verify the practicality and robustness of the proposed planner in unknown non-convex scenarios.**

I. INTRODUCTION

Real-time path planning in unknown environments remains a fundamental challenge in robotics. A planner must not only generate an initial feasible path but also efficiently repair and optimize the path as obstacles change or new regions of the environment are revealed [1]. In typical non-convex scenarios—such as room–corridor combinations, underwater caves, or underground parking structures—the robot faces even greater challenges. These environments often exhibit concave, enclosing, or narrow geometrical features, which constrain straight-line connectivity and place higher demands on both the reliability and responsiveness of path planning [2].

Early online path planning approaches include graph-search and potential field methods. Classical algorithms such as D* and its variants [3], [4] incrementally update paths as new obstacles are discovered, while Lifelong Planning A* (LPAstar) [5] and Anytime A* [6] improve efficiency through heuristic and anytime strategies. Artificial Potential Field (APF) methods [7], [8] offer real-time responsiveness by treating obstacles as repulsive forces and the goal as an attractive potential, though they often suffer from local minima in complex environments. These methods laid the foundation for online replanning but face scalability and convergence issues in high-dimensional spaces.

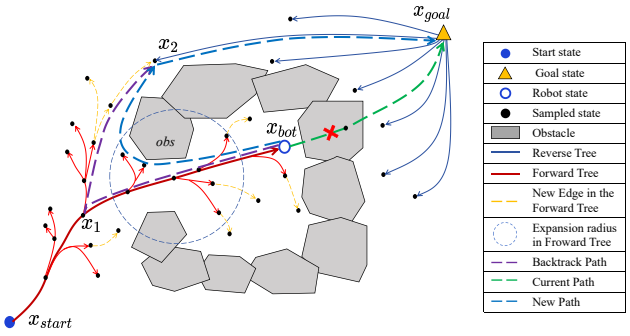


Fig. 1. Illustration of the InBi-RRT planning process. A reverse tree is first constructed from x_{goal} (blue solid lines) until a collision-free path (green dashed line) to x_{start} is found. The robot then departs from x_{start} with an initialized forward tree (thick red lines). When the current path becomes invalid due to collisions, the forward tree is incrementally expanded (thin red/yellow lines) in the same configuration space according to the cost function, while branches within the expansion radius are preserved (thin red lines). Once a collision-free connection between the forward and reverse trees is established, the backtracking mechanism generates a local path (purple dashed line) to repair the reverse tree, followed by a path optimization step that produces the final trajectory (blue dashed line). This process is repeated until the robot reaches x_{goal}.

Sampling-based path planning has emerged as a mainstream paradigm. In static environments, Rapidly-exploring Random Trees (RRT) [9] and their optimal variant RRT* [10] show strong scalability, while improvements such as Informed-RRT* [11], BIT* [12], and FMT* [13] accelerate convergence and improve path quality. In dynamic environments, extensions including E-RRT [14], GRIP [15], and ATS+EB [16] reuse partial structures or integrate elastic-band strategies for online replanning; however, frequent map/obstacle updates still trigger substantial re-propagation and rewiring of the search structure, which limits fast planning in practice [17].

The Dynamic RRT (DRRT) algorithm [18] introduced the idea of constructing a reverse tree rooted at the goal and repairing it by pruning invalid branches when obstacles change, thus requiring only a single tree to be maintained for replanning. Building on this concept, RRTX [19] proposed a reconnection cascade strategy that leverages stored neighborhood connectivity and iteratively updates affected subtrees, providing efficient propagation of changes with

This work was supported in part by Jing-Jin-Ji Regional Integrated Environmental Improvement-National Science and Technology Major Project under Grant 2025ZD1206800, in part by the National Natural Science Foundation of China (NSFC) under Grant U22A2066 and Grant U21B2047, in part by the National Natural Science Foundation of China under the Young Scientists Fund under Grant 62403384

[1] Bo Cui, Weisheng Yan, Zhanwei Yang, Ao Feng and Rongxin Cui are with the School of Marine Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China.

[2] Yang Li is with the Unmanned System Research Institute, Northwestern Polytechnical University, Xi'an 710072, China.

*Co-corresponding authors: Yang Li and Rongxin Cui.

e-mail: liyangusri@nwpu.edu.cn, r.cui@nwpu.edu.cn.

theoretical guarantees. More recently, RT-RRT [20] extended the reverse-tree framework by integrating a forward tree expanded from the robot's state, enabling faster online repair of invalid paths.

Reverse-tree-based methods such as RRTX and RT-RRT have thus shown favorable performance in unknown environments. Building on these ideas, we propose InBi-RRT (Incremental Bidirectional Rapidly-exploring Random Tree), an algorithm that incrementally constructs a forward tree to facilitate reverse tree repair. Specifically, a reverse tree $\mathcal{T}^-$ is first grown from x_{goal} to guide the robot's navigation. When the current path becomes invalid due to collisions, a forward tree $\mathcal{T}^+$ is incrementally expanded from the robot's current position within the same configuration space until a collision-free connection to $\mathcal{T}^-$ is established. This new connection is then integrated into $\mathcal{T}^-$, thereby repairing it incrementally and yielding an updated path (Fig. 1).

Beyond this basic mechanism, we introduce an additional innovation: instead of discarding forward trees $\mathcal{T}^+$ after each repair attempt, we preserve them and progressively accumulate a global forward tree $\mathcal{T}^+$ rooted at x_{start}. When subsequent collisions occur, this $\mathcal{T}^+$ is further incrementally expanded outward. Such a design is particularly advantageous in non-convex environments (e.g., rooms or caves with a single exit), where previously constructed branches near the entrance can be rapidly reused. As a result, the robot can quickly reconstruct feasible connections and efficiently retreat from challenging regions.

The distinctions of InBi-RRT compared to prior reverse-tree methods, particularly RT-RRT, constitute the main contributions of this work and are summarized as follows:

- A real-time planning framework based on bidirectional trees, where the key idea is to incrementally expand a forward tree to search for feasible connections, thereby enabling progressive repair and update of the reverse tree.
- A new cost function that jointly considers factors such as proximity to the robot, goal direction, local connection length, affinity to the existing forward tree, and tip expansion preference, guiding selective and efficient forward-tree growth.
- Extensive simulation studies validating the effectiveness of the proposed algorithm and demonstrating superior performance in non-convex and dynamic environments compared to baseline methods.

II. The InBi-RRT Algorithm

A. Problem Definition

Let $\mathcal{C}$ be the configuration space, consisting of the free region $\mathcal{C}_{\text{free}}$ and the obstacle region $\mathcal{C}_{\text{obs}}$:

$$\mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}}, \quad \mathcal{C}_{\text{free}} \cap \mathcal{C}_{\text{obs}} = \emptyset.$$

At time t, the robot state is $x_{\text{bot}}(t) \in \mathcal{C}_{\text{free}}$ with start x_{start} and goal x_{goal}. A path is denoted by $\pi_t = \{x_{\text{bot}}(t), x_1, \ldots, x_{\text{goal}}\}$, where each segment $\sigma(x_i, x_{i+1}) \subseteq \mathcal{C}_{\text{free}}$.

The environment is dynamic, i.e., $\mathcal{C}_{\text{obs}}(t)$ varies with time. Changes in $\mathcal{C}_{\text{obs}}(t)$ can render the current path π_t infeasible. The problem of interest in this paper is to design an algorithm that finds an initial feasible path and updates π_t online so that it always remains collision-free and executable in real time.

B. Terminology and Function Definitions

Before presenting the detailed algorithms, we summarize the key functions and operations used throughout the pseudocode.

- **GetPath**($\mathcal{T}, x_{\textbf{bot}}$): Extracts a collision-free path from the robot state x_{bot} to the goal within the tree $\mathcal{T}$.
- **needsReplanning**($\sigma(x_{\textbf{bot}}(t), x_{\textbf{goal}}), \mathcal{T}$): Checks whether the current path in $\mathcal{T}$ has become invalid due to collisions.
- **SampleFree**(): Uniformly samples a collision-free state in $\mathcal{C}_{\text{free}}$.
- **Nearest**($\mathcal{T}, x$): Returns the nearest node in tree $\mathcal{T}$ to the query state x.
- **Steer**($x_{\textbf{nearest}}, x_{\textbf{rand}}, \varepsilon$): Generates a new node by moving from x_{nearest} toward x_{rand} with step size ε.
- **Ancestors**($\mathcal{T}, x$): Returns the ordered set of ancestor nodes of x in tree $\mathcal{T}$, up to the root.
- **CollisionFree**($\sigma(x_1, x_2)$, $\mathcal{C}_{\textbf{free}}$): Determines whether the entire local path segment $\sigma(x_1, x_2)$ lies within the free region $\mathcal{C}_{\text{free}}$.
- **Sort**($X_{\textbf{cost}}$): Sorts candidate nodes by their cost in ascending order.
- **FirstCommon**($A_{\textbf{new}}, A_{\textbf{bot}}$): Finds the first common ancestor shared by two ancestor sets.

For clarity, we use $\mathcal{T}.V$ to denote the node set of tree $\mathcal{T}$, and $\mathcal{T}$.parent(x) to denote the parent of node x.

C. Overview

Algorithm 1 InBi-RRT

1: $x_{\text{bot}}(t_0) \leftarrow x_{\text{start}}$
2: $\mathcal{T}^- \leftarrow$CreateReverseTree$^-(x_{\text{goal}}, \mathcal{C})$
3: $\mathcal{T}^+.\mathcal{V} \leftarrow x_{\text{bot}}(t_0)$
4: $\pi_{t_0} \leftarrow$GetPath$(\mathcal{T}^-, x_{\text{bot}})$
5: **while** $x_{\text{bot}}(t) \neq x_{\text{goal}}$ **do**
6: **if** *obstacles have changed* **then**
7: updateObstacles()
8: **end if**
9: **if** *robot is moving* **then**
10: $x_{\text{bot}}(t) \leftarrow$ updateRobot$(x_{\text{bot}}(t))$
11: $\mathcal{T}^+.\mathcal{V} \leftarrow x_{\text{bot}}(t)$
12: $\mathcal{T}^+$.parent$(x_{\text{bot}}(t)) \leftarrow x_{\text{bot}}(t-1)$
13: **end if**
14: **if** needsReplanning$(\sigma(x_{\text{bot}}(t), x_{\text{goal}}), \mathcal{T}^-)$ **then**
15: UpdateForwardTree()
16: $\pi_t \leftarrow$GetPath$(\mathcal{T}^-, x_{\text{bot}})$
17: **end if**
18: $\pi_t^* \leftarrow$ PruneRewire$(\mathcal{T}^-, x_{\text{bot}}(t), x_{\text{goal}}, D_{\text{Threshold}})$
19: **end while**

Algorithm 1 illustrates the overall workflow of the proposed InBi-RRT framework. The method integrates incre-

mental forward-tree expansion into reverse-tree-based planning: on the one hand, the forward tree is incrementally updated as the robot moves through the configuration space; on the other hand, the reverse tree is incrementally repaired by establishing collision-free connections with the forward tree, thereby enabling real-time path adaptation in dynamic and complex environments.

The main program is presented in Algorithm 1, while the supporting subroutines are detailed in subsequent algorithms. Initially, the robot state is set to x_{start}, and the reverse tree $\mathcal{T}^-$ is constructed from x_{goal}, while a forward tree $\mathcal{T}^+$ rooted at x_{start} is also initialized . The initial path π_{t_0} is then extracted from the reverse tree(line 1-4).

During navigation, the main loop (lines 6-18) iteratively updates the configuration space whenever new obstacles are detected or the robot moves. The forward tree is incrementally expanded along the robot's trajectory to accumulate reachable regions(lines 9-13). If the path $\sigma(x_{\text{bot}}(t), x_{\text{goal}})$ within the reverse tree $\mathcal{T}^-$ becomes invalid, the `updateForwardTree()` function incrementally grows $\mathcal{T}^+$ until a collision-free connection with $\mathcal{T}^-$ is established (line 14-17) . This connection is integrated into $\mathcal{T}^-$ to repair the reverse tree, after which the `PruneRewire()` function (line 18) refines the resulting trajectory. The process repeats until the robot reaches the goal state.

D. Construction of $\mathcal{T}^-$

A reverse tree $\mathcal{T}^-$ rooted at x_{goal} can be incrementally repaired as obstacles change, making it effective in dynamic environments. Similar to RT-RRT, we avoid fixed-radius neighbor searches by leveraging ancestor states of x_{nearest} as candidate parents. This strategy reduces both planning time and path cost by exploiting the hierarchical structure of the tree. Algorithm 2 outlines this process: after generating a new state x_{new}, its connection is validated against ancestor states, and the oldest collision-free ancestor is chosen as parent to expand $\mathcal{T}^-$.

Algorithm 2 CreateReverseTree$^-$ $(x_{\text{goal}}, \mathcal{C})$

1: $\mathcal{T}^-.\mathcal{V} \leftarrow x_{\text{goal}}$
2: **for** $i = 1$ **to** n_{samples} **do**
3: $x_{\text{rand}} \leftarrow \text{SampleFree}()$
4: $x_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}^-, x_{\text{rand}})$
5: $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}}, \varepsilon)$
6: **for each** $x_{\text{ancestor}} \in \text{Ancestors}(x_{\text{nearest}})$ **do**
7: **if** $\text{CollisionFree}(\sigma(x_{\text{new}}, x_{\text{ancestor}}), \mathcal{C}_{\text{free}})$ **then**
8: $\mathcal{T}^-.\mathcal{V} \leftarrow x_{\text{new}}$
9: $\mathcal{T}^-.\text{parent}(x_{\text{new}}) \leftarrow x_{\text{ancestor}}$
10: **break**
11: **end if**
12: **end for**
13: **end for**

E. Construction of $\mathcal{T}^+$

The key distinction from RT-RRT lies in the construction of the forward tree. In RT-RRT, a new random tree is reconstructed from the robot's current position whenever an obstacle invalidates the path. By contrast, InBi-RRT maintains a single global forward tree. During replanning, a newly designed cost function guides selective expansion and update of this tree, while previously constructed information is preserved for incremental growth. This design enables efficient reuse of existing forward-tree branches, which is particularly advantageous in non-convex environments (e.g., rooms or caves), allowing the robot to quickly discover escape paths (as shown in Fig. 2).

The forward-tree construction mainly consists of the following three components:

- an incremental update mechanism that enables progressive expansion while preserving previously explored branches;
- a new cost function that effectively guides the forward-tree growth;
- a path backtracking strategy that, after establishing a connection with the reverse tree, traces back to the current state and integrates the result into the reverse tree for global path update.

Algorithm 3 UpdateForwardTree$^+$ $(\mathcal{T}^-.\mathcal{V}, x_{\text{bot}}(t))$

1: $\mathcal{T}^* \leftarrow \mathcal{T}^+.copy()$
2: $X_{\text{cost}} \leftarrow \text{ExpansionCost}(\mathcal{T}^-, \mathcal{T}^+, x_{\text{goal}}, x_{\text{bot}}(t))$
3: $X_{\text{sort}} \leftarrow \text{Sort}(X_{\text{cost}})$
4: **for each** $x_{\text{new}} \in X_{\text{sort}}$ **do**
5: $x_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}^*.\mathcal{V}, x_{\text{new}})$
6: **if** $\text{CollisionFree}(\sigma(x_{\text{nearest}}, x_{\text{new}}), \mathcal{C}_{\text{free}})$ **then**
7: $\mathcal{T}^*.\mathcal{V} \leftarrow \{x_{\text{new}}\}$
8: $\mathcal{T}^*.\text{parent}(x_{\text{new}}) \leftarrow x_{\text{nearest}}$
9: **if** $|\sigma(x_{\text{new}}, x_{\text{bot}}(t))| < D_{\text{sensing}}$ **then**
10: $\mathcal{T}^+.\mathcal{V} \leftarrow \{x_{\text{new}}\}$
11: $\mathcal{T}^+.\text{parent}(x_{\text{new}}) \leftarrow x_{\text{nearest}}$
12: **end if**
13: **end if**
14: **if** $\text{CollisionFree}(\sigma(x_{\text{new}}, x_{\text{goal}})$ in $\mathcal{T}^-, \mathcal{C}_{\text{free}})$ **then**
15: $\text{BacktrackPathToNode}(x_{\text{new}}, x_{\text{bot}}(t), \mathcal{T}^-, \mathcal{T}^*)$
16: BREAK
17: **end if**
18: **end for**

Algorithm 3 describes the incremental update and path rewriting mechanism of the forward tree during replanning. First, the current forward tree is duplicated to generate a temporary tree $\mathcal{T}^*$ (line 1). This temporary tree is initialized from the existing forward tree and continues to expand into the space. However, its branches cannot be guaranteed to remain collision-free during the robot's subsequent motion, especially beyond the sensor detection range. Therefore, $\mathcal{T}^*$ mainly serves to quickly search for collision-free paths at the current moment and selectively retain relatively safe branches (typically those within the detection radius).

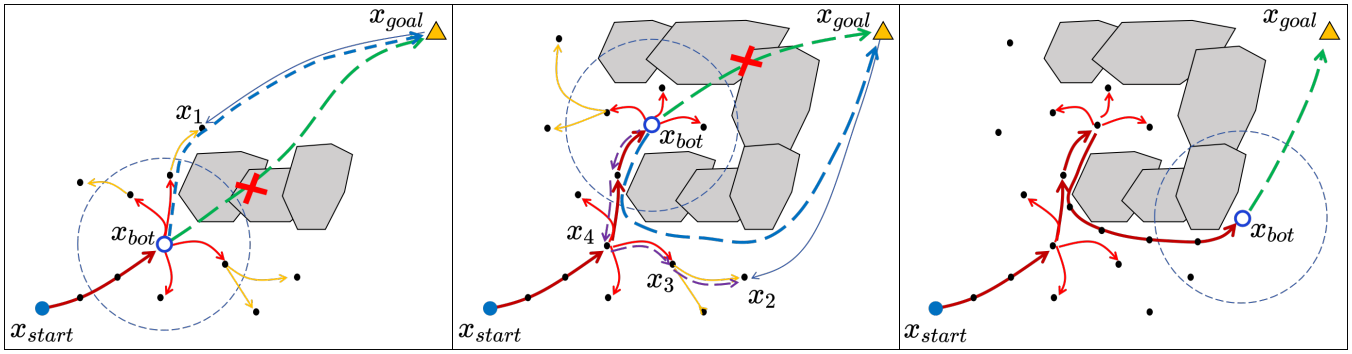


Fig. 2. Forward-tree update and reverse-tree repair process (notations are consistent with Fig. 1; for clarity, only the key branches of the reverse tree are shown). (a) The robot departs from the start and incrementally constructs the forward tree $\mathcal{T}^+$ (red solid lines). When the current path (green dashed line) collides with a newly detected obstacle, InBi-RRT starts updating the forward tree $\mathcal{T}^+$ (red/yellow solid lines) until a collision-free connection to the reverse tree $\mathcal{T}^-$ is found at node x_1. The connection $x_{\text{goal}} - x_1 - x_{\text{bot}}$ is then merged into the $\mathcal{T}^-$, guiding the robot along a new collision-free path (blue dashed line). Meanwhile, branches of $\mathcal{T}^+$ within the sensing range are preserved (red solid lines) for efficient reuse. (b) As the robot proceeds, another obstacle is detected. InBi-RRT again updates the forward tree $\mathcal{T}^+$ and finds a new node x_2 near the previously expanded node x_3, establishing a collision-free connection to the reverse tree $\mathcal{T}^-$. The backtracking mechanism retrieves the local path $x_{\text{bot}} - x_4 - x_3 - x_2$ (purple dashed line) and integrates it into the reverse tree. Subsequently, the prune rewire procedure optimizes the trajectory (blue dashed line), reducing overall path cost. (c) The robot escapes the non-convex obstacle region under the guidance of InBi-RRT. The forward tree $\mathcal{T}^+$ (red solid lines) grows along the robot's trajectory as its backbone, and the above update-and-repair process repeats until the robot successfully reaches the goal.

Subsequently, the `ExpansionCost` function evaluates the expansion priority of candidate nodes derived from $\mathcal{T}^-.V$ and the current robot state $x_{\text{bot}}(t)$, producing a sorted sequence X_{sort}.(lines 2-3)

Next, the nearest neighbor in $\mathcal{T}^*$ is identified; if a collision-free local path exists, the node and its parent relation are added into $\mathcal{T}^*$ (lines 5-8). At the same time, when the candidate node lies within the robot's safe sensing distance D_{sensing}, it is promoted to the global forward tree $\mathcal{T}^+$ (lines 9-12) to enable sustainable reuse.

Furthermore, if a candidate node establishes a feasible connection with the reverse tree, the algorithm invokes `BacktrackPathToNode` to extract a local path from the robot's current position to this node within $\mathcal{T}^*$ and merges this backtracked path into the reverse tree $\mathcal{T}^-$ (line 15).

Overall, this mechanism achieves selective expansion of the forward tree under the guidance of cost evaluation, while incrementally reusing historical information to ensure efficient replanning in dynamic and non-convex environments.

1) ExpansionCost: The procedure `ExpansionCost` assigns an expansion priority to each candidate node by computing a composite cost. Given a candidate set $E = \mathcal{T}^-.V$, the cost of node $i \in E$ is defined as

$$J(i) = w_{\text{curr}}\, d_{\text{curr}}(i) \;+\; w_{\text{dir}}\, \Delta_{\text{dir}}(i) \;+\; w_{\text{conn}}\, d_{\text{conn}}(i) \\ +\; w_{\text{ft}}\, \phi_{\text{ft}}(i) \;-\; w_{\text{tip}}\, b_{\text{tip}}(i), \tag{1}$$

where each term captures a specific heuristic property, and $w_\cdot$ are weighting coefficients. To avoid over-expansion into unreliable regions, the overall cost is further scaled as

$$J(i) \leftarrow J(i) \times \begin{cases} \text{scale}_{\text{far}}, & d_{\text{curr}}(i) \le D_{\text{sensing}}, \\ \text{scale}_{\text{out}}, & d_{\text{curr}}(i) > D_{\text{sensing}}, \end{cases} \tag{2}$$

where D_{sensing} is the safe sensing radius.

a) Distance to current node.:

$$d_{\text{curr}}(i) = \|x_i - x_{\text{bot}}(t)\|_2,$$

favoring nodes closer to the robot's current position to ensure feasible short-term progress.

b) Direction to goal.: Let u_{go} be the normalized direction from $x_{\text{bot}}(t)$ to x_{goal}, and $v_i = x_i - x_{\text{bot}}(t)$. The directional mismatch is

$$\Delta_{\text{dir}}(i) = 1 - \frac{v_i^\top u_{\text{go}}}{\|v_i\|_2 + \varepsilon},$$

penalizing nodes deviating from the goal direction.

c) Connection length.:

$$d_{\text{conn}}(i) = \|x_i - x_{\text{parent}(i)}\|_2,$$

which encourages short local connections and reduces the risk of invalid long edges.

d) Forward-tree affinity.: Let $\mathcal{T}^+.V$ denote the set of nodes in the existing forward tree. For candidate i, the affinity penalty is

$$\phi_{\text{ft}}(i) = \frac{d_{\text{ft}}(i)}{d_{\text{ft}}(i) + \sigma}, \quad d_{\text{ft}}(i) = \min_{j \in \mathcal{T}^+.V} \|x_i - x_j\|_2,$$

where $\sigma = 0.75\, r_{\text{radius}}$. This term promotes candidates close to the forward tree, encouraging reuse of reliable structures.

e) Forward-tree tip bonus.: For candidates already in the forward tree, a bonus $b_{\text{tip}}(i) = 1$ is assigned if their degree (number of neighbors) is no greater than one, and 0 otherwise. This promotes the expansion of tree tips, which are more likely to lead exploration into unexplored regions.

f) Non-local scaling.: When $d_{\text{curr}}(i) > D_{\text{sensing}}$, the cost is amplified by $\text{scale}_{\text{out}} > 1$, reducing the priority of nodes outside the safe sensing radius.

Overall, `ExpansionCost` integrates proximity, goal direction, local connectivity, affinity to the existing forward

tree, and tip preference, while penalizing distant nodes. This design enables selective and incremental reuse of forward-tree information, ensuring efficient replanning in dynamic and non-convex environments.

2) BacktrackPathToNode: Algorithm 4 presents the `BacktrackPathToNode` procedure, which integrates a newly discovered feasible connection into the forward tree $\mathcal{T}^+$. Once a candidate node x_{new} is found to connect collision-free with the reverse tree $\mathcal{T}^-$, the procedure locates their nearest common ancestor x_{com} shared with the current robot state $x_{\text{bot}}(t)$ in the search tree $\mathcal{T}$. The parent–child relations along the path from x_{new} to x_{com} are then reversed to ensure consistency with the orientation of $\mathcal{T}^+$. By linking x_{new} as the parent of $x_{\text{bot}}(t)$ and promoting the resulting chain into $\mathcal{T}^+$, the forward tree is incrementally expanded. Finally, a valid path from the global goal to $x_{\text{bot}}(t)$ is obtained by tracing through the updated $\mathcal{T}^+$. This mechanism enables incremental reuse of forward-tree information while ensuring a globally consistent trajectory in dynamic and non-convex environments.

Algorithm 4 BacktrackPathToNode($x_{\text{bot}}(t)$, x_{new}, $\mathcal{T}^-$, $\mathcal{T}^+$)

$A_{\text{bot}} \leftarrow$ Ancestors($\mathcal{T}^+$, $x_{\text{bot}}(t)$)
$A_{\text{new}} \leftarrow$ Ancestors($\mathcal{T}^+$, x_{new})
$x_{\text{com}} \leftarrow$ FirstCommon(A_{new}, A_{bot})
$P_{\text{com}\to\text{new}} \leftarrow$ Path(x_{com}, x_{new}, $\mathcal{T}^+$)
$prev \leftarrow x_{\text{new}}$
for each $v \in$ Reverse($P_{\text{com}\to\text{new}}$) **do**
  $\mathcal{T}^+$.parent(v) $\leftarrow prev$
  $prev \leftarrow v$
end for
$P_{\text{new}\to\text{bot}} \leftarrow P_{\text{new}\to\text{com}} \cup P_{\text{com}\to\text{bot}}$
for each $x \in P_{\text{new}\to\text{bot}}$ **do**
  $\mathcal{T}^-$.V $\leftarrow x$
  $\mathcal{T}^-$.parent(u) $\leftarrow \mathcal{T}^+$.parent(x)
end for

F. Prune Rewire

Algorithm 5 PruneRewire($\mathcal{T}^-$, $x_{\text{bot}}(t)$, x_{goal}, $D_{\text{Threshold}}$)

$A_{\text{goal}} \leftarrow$ Ancestors($\mathcal{T}^-$, x_{goal})
$x_{\text{far}} \leftarrow x_{\text{goal}}$
for each $x \in A_{\text{goal}}$ in order from x_{goal} to $x_{\text{bot}}(t)$ **do**
  if CollisionFree($x_{\text{bot}}(t)$, x, $\mathcal{C}_{\text{free}}$) **then**
    $\mathcal{T}^-$.parent(x) $\leftarrow x_{\text{bot}}(t)$
  else
    break
  end if
end for
$\pi_t^* \leftarrow$OptimizePath($\mathcal{T}^-$, $D_{\text{Threshold}}$)
RETURN π_t^*

Before optimization, InBi-RRT prunes along the current path by directly linking the farthest collision-free ancestor to x_{bot}. `OptimizePath` then rewrites parents until convergence (same routine as RT-RRT).This additional pruning is necessary because, unlike RT-RRT where forward-tree construction already accounts for historical nodes, InBi-RRT accelerates planning by omitting forward-branch optimization and applies optimization only on the final path(Algorithm 5).

III. Simulations and Experiments

To validate the effectiveness and advantages of the proposed algorithm, three sets of experiments were conducted. The first experiment was performed in a large-scale unknown environment of $1000\,\text{m} \times 1000\,\text{m}$ to evaluate the algorithm's capability for real-time path planning in unknown and non-convex scenarios. The second experiment compared the proposed method with RT-RRT and RRT$^{\text{X}}$ to further demonstrate its improvements in planning efficiency and path quality. The third experiment was carried out in a real-world indoor maze-like environment to verify the algorithm's practicality and robustness in unknown non-convex scenarios.

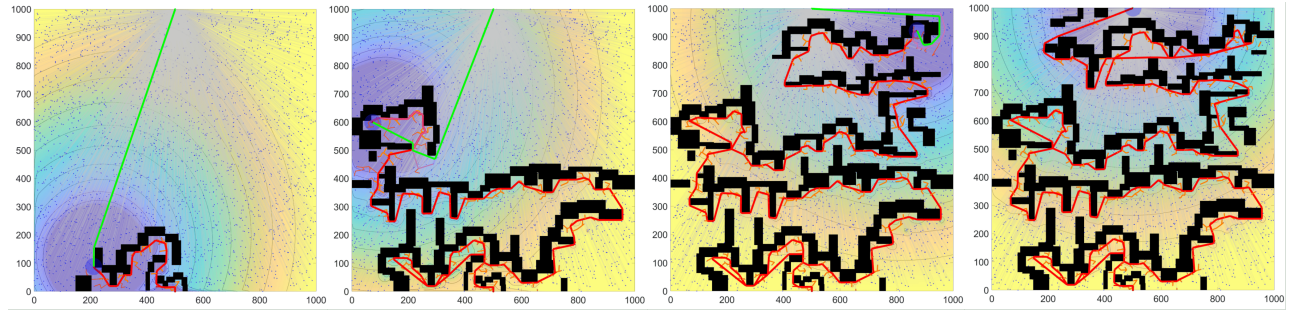
A. Simulations in Unknown Environments

We evaluated the proposed algorithm in two representative scenarios. The environment size was $1000\,\text{m} \times 1000\,\text{m}$. In the tests, the robot's velocity was set to $5\,\text{m/s}$, the sensing radius was $30\,\text{m}$, the start state was located at $(500\text{m}, 0\text{m})$, the goal at $(500\text{m}, 1000\text{m})$, and the number of samples was 3000. In Scenario 1, the environment contained no prior information and included maze unknown obstacles. In Scenario 2, the environment included two known obstacles along with additional unknown non-convex obstacles.

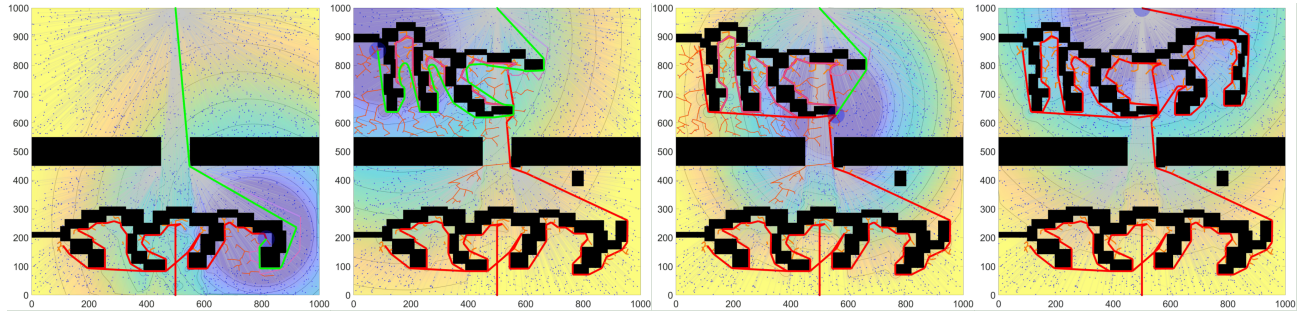
The experimental results are illustrated in Fig. 3, where (a)-(d) correspond to partial processes in Scenario 1 and (e)-(h) show partial outcomes in Scenario 2. The InBi-RRT algorithm guides the robot toward the goal via the reverse tree while incrementally constructing a forward tree (red). Selected branches (yellow) are preserved for reuse in subsequent replanning. During path planning, the forward tree is expanded under a cost function that incorporates not only goal heuristics and sample distributions but also incentives for existing branches and terminal nodes. This design enables the algorithm to exploit local historical information effectively, thereby allowing the robot to quickly identify feasible detours in non-convex regions. Meanwhile, the path backtracking mechanism and path optimization module further refine the trajectory to reduce the overall path cost.

In Scenario 1, which contains large-scale unknown obstacles with maze-like structures, InBi-RRT successfully completes the navigation task. When the robot becomes trapped in a non-convex obstacle (Fig. 3(b)), the backtracking mechanism rapidly generates a feasible path from the current state to the goal (pink curve), though it may not be optimal. The optimization module then progressively refines this path into a shorter and smoother trajectory (green curve), combining fast responsiveness with path quality improvement.

In Scenario 2, two known obstacles are explicitly incorporated into the construction of the forward tree (shown as rectangles), accelerating online planning. When the robot enters a complex non-convex region (Fig. 3(f)), the initial



(a) Step 153 (b) Step 815 (c) Step 1360 (d) Step 1647



(e) Step 524 (f) Step 1215 (g) Step 1351 (h) Step 1708

Fig. 3. The experiment demonstrates real-time obstacle avoidance in a $1000\,\mathrm{m} \times 1000\,\mathrm{m}$ unknown environment. A reverse tree $\mathcal{T}^-$ rooted at x_{goal} (blue dots and gray lines) guides the robot forward (green line), while visited states incrementally form a forward tree $\mathcal{T}^+$ (red lines). When obstacles (black rectangles) are detected within the sensing range (blue circles), the forward tree $\mathcal{T}^+$ is expanded under the guidance of the cost function (colored background). Some branches are selectively preserved (yellow lines) for future reuse, while others are pruned (orange lines). Once a collision-free connection is found, path `BacktrackPathToNode` (pink line) and `PruneRewire` (green line) ensure that the robot safely reaches the x_{goal}.

backtracking path tends to traverse multiple non-convex structures. However, as the execution proceeds, the optimization module continuously adjusts the trajectory and avoids revisiting non-convex traps, ultimately leading to the shortest path from the backtracking point to the goal(Fig. 3(g)).

Overall, the results demonstrate that InBi-RRT achieves both real-time performance and robustness in large-scale unknown environments as well as in hybrid scenarios with known obstacles. In particular, in non-convex environments, the algorithm efficiently reuses historical forward-tree structures to rapidly generate feasible detours, while the optimization mechanism ensures improved path quality, highlighting the effectiveness of the proposed approach for real-time obstacle avoidance and replanning in complex settings.

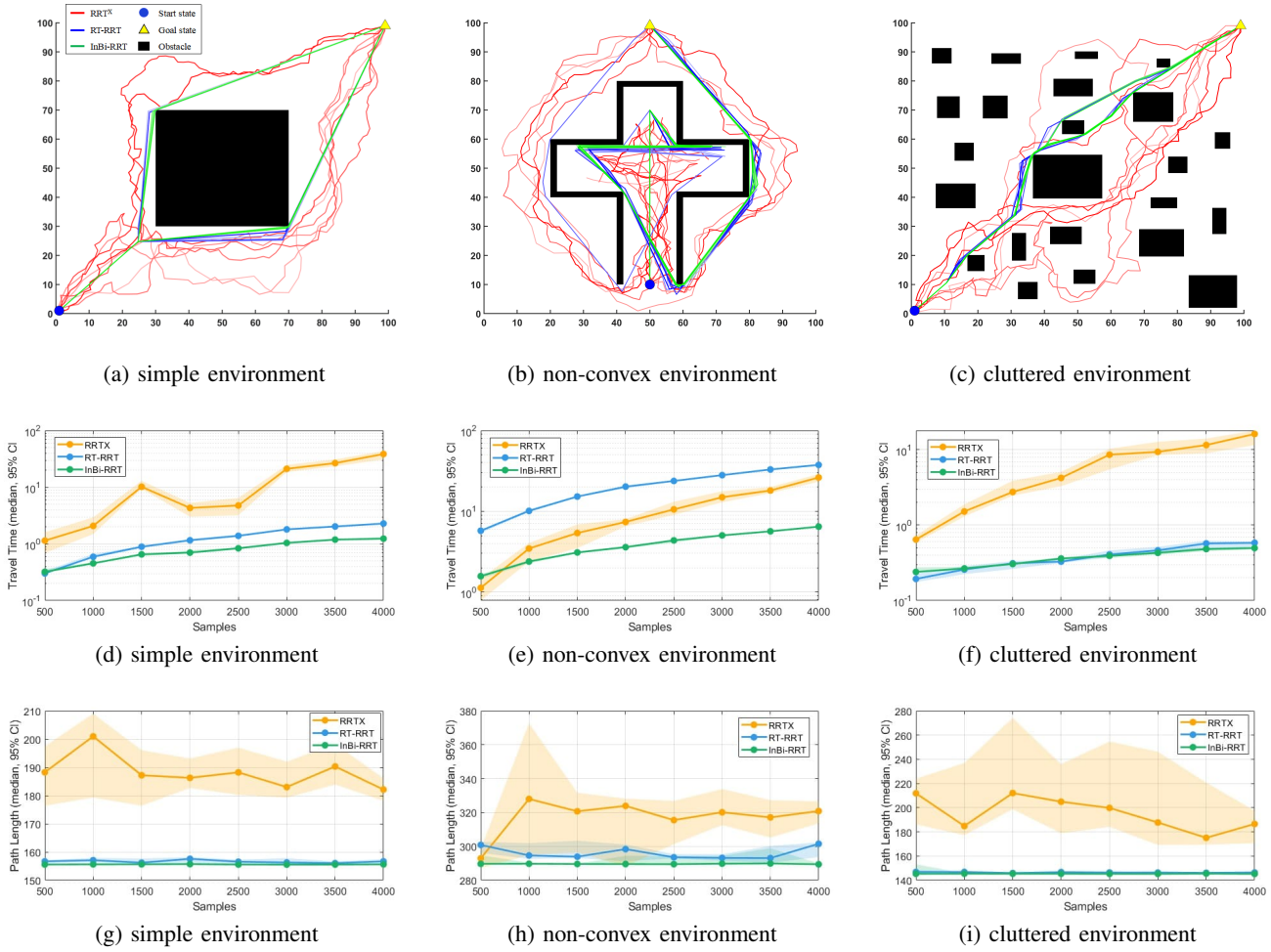
B. Comparative Experiments

In this section, we compare three representative reverse-tree algorithms: RRT$^{\text{X}}$, RT-RRT, and InBi-RRT, across three typical environments (Fig. 4(a)-(c)). The number of samples was used as the varying parameter, starting from 500 and increasing in increments of 500 up to 4000. For each sampling setting, all three algorithms were executed independently 10 times, and the results were evaluated in terms of path length and travel time. The experimental parameters were configured as follows: the environment size was $100\text{m} \times 100\text{m}$, the robot velocity was set to $5\,\text{m/s}$, the sensing radius was $10\,\text{m}$, and all obstacles were unknown. In scenarios a and c, the start state was $(1\text{m},\ 1\text{m})$ and the goal state was $(99\text{m},\ 99\text{m})$, while in scenario b, the start state was $(50\text{m},\ 10\text{m})$ and the goal state was $(50\text{m},\ 99\text{m})$. The variation trends of travel time under different sample sizes are shown in Fig. 4(d)-(f), and the corresponding path length trends are shown in Fig. 4(g)-(i).

In Fig. 4(a)-(c), the colored trajectories represent the robot paths generated under different sample sizes by InBi-RRT (green), RT-RRT (blue), and RRT$^{\text{X}}$ (red). The color intensity indicates the number of samples. The robot starts from the same initial position (blue circle) and moves toward the goal (yellow triangle) under the guidance of different algorithms. As illustrated in Fig. 4(a)-(c), both RT-RRT and InBi-RRT benefit from the path optimization strategy, resulting in overall higher path quality compared to RRT$^{\text{X}}$.

Figures 4(d)–(f) present the travel-time statistics of the three algorithms under different environments, with the vertical axis plotted on a logarithmic scale. Taking 3000 samples as an example, the results show that in the simple environment (a), the travel time of InBi-RRT is $22\times$ faster than RRTX and $1.76\times$ faster than RT-RRT. In the complex non-convex environment (b), InBi-RRT is $3\times$ faster than RRTX and $5.5\times$ faster than RT-RRT. In the cluttered environment (c), InBi-RRT is $20\times$ faster than RRTX and comparable to RT-RRT.

These differences can be attributed to the distinct repair strategies of the reverse tree in each algorithm. RRT$^{\text{X}}$ relies on globally extending the residual reverse tree after obstacles



(a) simple environment (b) non-convex environment (c) cluttered environment

(d) simple environment (e) non-convex environment (f) cluttered environment

(g) simple environment (h) non-convex environment (i) cluttered environment

Fig. 4. Paths traveled by the robot under RRTX (red), RT-RRT (blue) and InBi-RRT (green) for different sample counts. For each method, the trajectory color darkens as the number of samples increases.

appear and incrementally updating neighbor nodes in a top-down manner from the goal. This global repair strategy does not fully exploit the robot's current state information, leading to relatively high computational cost. In contrast, both RT-RRT and InBi-RRT adopt a reverse-tree-guided, forward-tree-repair structure, where local incremental expansions are triggered from the robot's current position. Once a feasible connection is found, it is integrated into the reverse tree, significantly reducing repair time.

Compared with RT-RRT, InBi-RRT achieves higher efficiency mainly for two reasons. First, in RT-RRT, every newly added branch of the forward tree requires optimization and parent rewiring, whereas InBi-RRT expands branches primarily by local extension without additional rewiring, saving substantial computation. This advantage is particularly evident in environments with large obstacles, where RT-RRT requires additional computation for branch optimization and parent rewiring (Fig. 4d). However, in the cluttered environment (c), where obstacles are relatively small and each expansion involves fewer nodes, both RT-RRT and InBi-RRT can quickly find feasible paths, and thus their time difference becomes less pronounced.

Second, InBi-RRT selectively preserves portions of the historical forward tree. This design enables the algorithm to reuse existing branches when entering non-convex regions, allowing it to quickly identify escape paths. In contrast, RT-RRT reconstructs a new forward tree from the current position whenever an obstacle is encountered, repeatedly rebuilding trees within confined regions and exhaustively exploring internal samples—thereby resulting in much slower performance (as in Fig. 4e).

Figures 4(g)–(i) show the path length trends of the three algorithms across different scenarios. RRT^X is asymptotically optimal but lacks explicit path optimization strategies, which leads to suboptimal paths when the number of samples is small. As the sample size increases, the path cost gradually decreases, but this improvement comes at the expense of substantially higher computation, making it difficult to balance efficiency and path quality. In contrast, InBi-RRT leverages the path backtracking mechanism together with a path optimization strategy similar to RT-RRT, resulting in consistently shorter paths. Both InBi-RRT and RT-RRT achieve significant reductions in path cost under different sampling settings. For instance, with 3000 samples, InBi-

RRT produces path lengths comparable to RT-RRT and shorter than RRT^X by approximately 19.8%, 9.0%, and 17.8% in the three scenarios, respectively.

In summary, by combining bidirectional incremental repair with selective reuse of forward-tree structures, InBi-RRT consistently reduces computation time and path cost. The algorithm outperforms RT-RRT and RRT^X in large-scale non-convex environments and achieves similar efficiency to RT-RRT in cluttered cases, demonstrating both real-time capability and robustness.

C. Real-World Experiments

To further validate the effectiveness of the proposed algorithm, real-world experiments were conducted. The experimental platform was a tethered mobile robot (Pibot) equipped with a LiDAR sensor, with a detection radius of 1 m. An indoor maze-like environment was constructed using cardboard walls and concrete blocks (Fig. 5), which contains narrow passages and non-convex corridors, posing significant challenges for real-time path planning. The robot was required to move from a predefined start position to a goal point located at the opposite end of the corridor, while the overall environment was completely unknown to it. During navigation, InBi-RRT was able to continuously replan paths in response to newly detected obstacles, allowing the robot to traverse the complex environment without collisions (Figs. 5(a)–(d)). The results demonstrate that the proposed planner can effectively handle environmental uncertainty and achieve efficient path planning in non-convex scenarios.

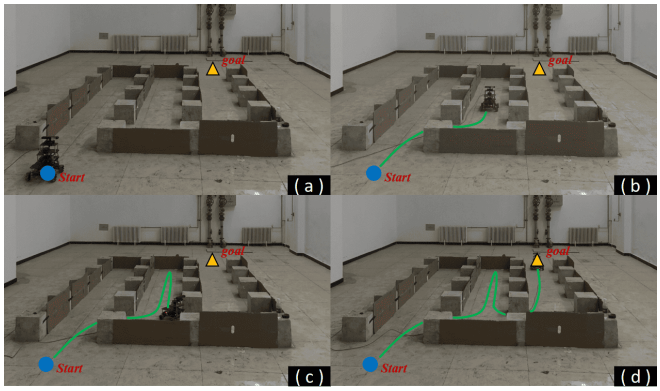


Fig. 5. Real-world experiment conducted in an indoor maze-like environment. Cardboard walls and concrete blocks were arranged to form narrow passages and non-convex corridors. (a)–(d) illustrate the robot navigating from the start point to the goal while InBi-RRT replans paths in response to newly detected obstacles, enabling successful traversal without collisions.

IV. CONCLUSIONS

We presented InBi-RRT, a real-time path planner that maintains a single reusable forward tree and incrementally repairs the reverse tree through cost-guided expansion, backtracking, and lightweight optimization. Extensive simulations in unknown non-convex and cluttered environments demonstrated up to 5.5 $\times$ and 22 $\times$ faster replanning compared to RT-RRT and RRT^X, respectively, while achieving 9%–19.8% shorter paths than RRT^X. Real-world experiments further verified the robustness of the proposed approach. Future work will focus on extending InBi-RRT to address kinodynamic constraints and broader robotic platforms.

REFERENCES

[1] B. Zhu, J. He, Z. Yuan, and F. Gao, "Probabilistic path planning for wheel-legged rover in dense environment based on extended mdp and configuration topology analysis," *IEEE Transactions on Robotics*, 2025.

[2] V. K. Adajania, S. Zhou, A. K. Singh, and A. P. Schoellig, "Amswarmx: Safe swarm coordination in complex environments via implicit non-convex decomposition of the obstacle-free space," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 555–14 561.

[3] A. Stentz *et al.*, "The focussed d* algorithm for real-time replanning," in *IJCAI*, vol. 95, 1995, pp. 1652–1659.

[4] R. Dechter, M. Kearns, R. Sutton, and S. H. Form, "Eighteenth national conference on artificial intelligence." American Association for Artificial Intelligence, 2002.

[5] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[6] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm." in *ICAPS*, vol. 5, 2005, pp. 262–271.

[7] C. W. Warren, "Global path planning using artificial potential fields," in *Proceedings, 1989 International Conference on Robotics and Automation*. Ieee, 1989, pp. 316–321.

[8] N. Malone, H.-T. Chiang, K. Lesser, M. Oishi, and L. Tapia, "Hybrid dynamic moving obstacle avoidance using a stochastic reachable set-based potential field," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1124–1138, 2017.

[9] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2014, pp. 2997–3004.

[12] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[13] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[14] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 3. IEEE, 2002, pp. 2383–2388.

[15] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 704–710.

[16] K. Hauser, "On responsiveness, safety, and completeness in real-time motion planning," *Autonomous Robots*, vol. 32, no. 1, pp. 35–48, 2012.

[17] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419839596, 2019.

[18] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1243–1248.

[19] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[20] B. Cui, R. Cui, W. Yan, Y. Wang, and S. Zhang, "Rt-rrt: Reverse tree guided real-time path planning/replanning in unpredictable dynamic environments," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 5380–5387.