EB-RRT: Optimal Motion Planning for Mobile Robots

Jiankun Wang, Max Q.-H. Meng, *Fellow, IEEE*, and Oussama Khatib, *Life Fellow, IEEE*

***Abstract*—In a human–robot coexisting environment, it is pivotal for a mobile service robot to arrive at the goal position safely and efficiently. In this article, an elastic band-based rapidly exploring random tree (EB-RRT) algorithm is proposed to achieve real-time optimal motion planning for the mobile robot in the dynamic environment, which can maintain a homotopy optimal trajectory based on current heuristic trajectory. Inspired by the EB method, we propose a hierarchical framework consisting of two planners. In the global planner, a time-based RRT algorithm is used to generate a feasible heuristic trajectory for a specific task in the dynamic environment. However, this heuristic trajectory is nonoptimal. In the dynamic replanner, the time-based nodes on the heuristic trajectory are updated due to the internal contraction force and the repulsive force from the obstacles. In this way, the heuristic trajectory is optimized continuously, and the final trajectory can be proved to be optimal in the homotopy class of the heuristic trajectory. Simulation experiments reveal that compared with two state-of-the-art algorithms, our proposed method can achieve better performance in dynamic environments.**

***Note to Practitioners*—The motivation of this work stems from the need to achieve real-time optimal motion planning for the mobile robot in the human–robot coexisting environment. Sampling-based algorithms are widely used in this area due to their good scalability and high efficiency. However, the generated trajectory is usually far from optimal. To obtain an optimized trajectory for the mobile robot in the dynamic environment with moving pedestrians, we propose the EB-RRT algorithm on the basis of the time-based RRT tree and the EB method. Depending on the time-based RRT tree, we quickly get a heuristic trajectory and guarantee the probabilistic completeness of our algorithm. Then, we optimize the heuristic trajectory similar to the EB method, which achieves the homotopy optimality of the final trajectory. We also take into account the nonholonomic constraints, and our proposed algorithm can be applied to most mobile robots to further improve their motion planning ability and the trajectory quality.**

***Index Terms*—Elastic band (EB), mobile robots, optimal motion planning, sampling-based motion planning.**

Manuscript received February 18, 2020; accepted April 7, 2020. Date of publication April 30, 2020; date of current version October 6, 2020. This article was recommended for publication by Associate Editor Y. Li and Editor K. Saitou upon evaluation of the reviewers' comments. This work was supported in part by the Hong Kong ITC ITSP Tier2 under Grant ITS/105/18FP and in part by the Hong Kong ITC MRP under Grant MRP/011/18. *(Corresponding authors: Max Q.-H. Meng; Oussama Khatib.)*

Jiankun Wang is with the Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: jkwang@ee.cuhk.edu.hk).

Max Q.-H. Meng is with the Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: max@ee.cuhk.edu.hk).

Oussama Khatib is with the Stanford Robotics Laboratory, Stanford University, Palo Alto, CA 94305 USA (e-mail: ok@cs.stanford.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2020.2987397

I. Introduction

In ROBOTICS, a fundamental task is to plan collision-free motions for the robot for some specific tasks among a number of static or moving obstacles [1]. Lots of efforts have been devoted to the motion planning problem over the last few decades. Many kinds of motion planning algorithms are proposed and have widespread success in different robotic applications, which can be broadly classified into three categories. The artificial potential field [2] algorithms find the feasible trajectory by following the direction of the steepest descent of the potential. However, they often end up in a local minimum. The grid-based algorithms, such as the A* [3], are resolution complete and resolution optimal, meaning that they can always find the optimal trajectory if it exists. However, they do not perform well as the problem scale increases. Sampling-based algorithms use the collision detector to compute whether a randomly sampled configuration is collision-free and construct a data structure to store collision-free trajectories. Therefore, these algorithms can avoid the exact geometric modeling of the configuration space. With this level of abstraction, sampling-based planners, such as the rapidly exploring random tree (RRT) [4], the probabilistic roadmap method (PRM) [5], and their many variants, are applicable to a wide variety of problems. It is noted that the sampling-based algorithms only provide a weaker form of completeness (also called probabilistic completeness). In addition, the generated trajectories from the RRT and the PRM are nonoptimal.

In a dynamic environment, the planned trajectory may become invalid due to the change in the current environment. Therefore, it is necessary for the planner to have the ability of real-time response to the current environment. The elastic band (EB) method [6] can be used to quickly adjust the current trajectory in the dynamic environment. As shown in Fig. 1(a), with a three-level framework, the heuristic trajectory from the path planner is updated by the EB module, and then, the final control command is obtained in the control planner. However, the EB method needs an initial trajectory and construction of the configuration space. In the dynamic environment, generating a feasible trajectory as quickly as possible is necessary for the path planner. When the working scenario is pretty large, such as an airport, constructing the configuration space costs a lot of computation resource and it becomes much harder to generate a feasible initial trajectory quickly. The sampling-based algorithms are suitable for their fewer requirements on computation resources and good scalability. Moreover, they can avoid the exact modeling of the configuration space. The time-based RRT algorithms, such as Risk-RRT [7], provide a solution to achieve real-time motion

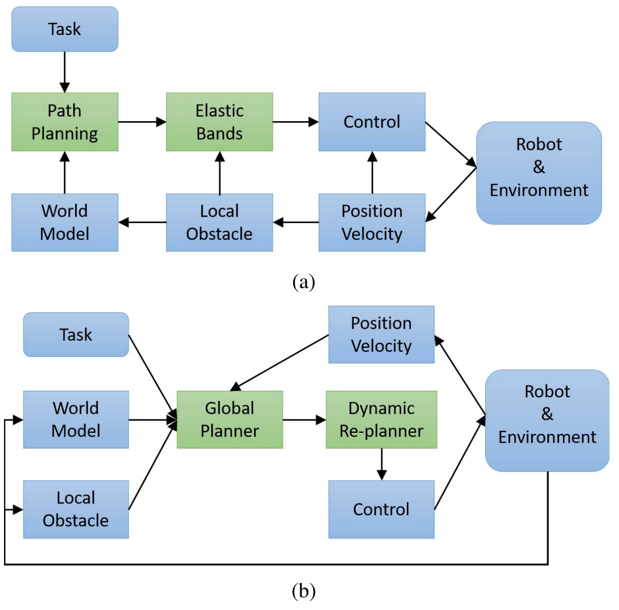


Fig. 1. Framework of the EB and the EB-RRT algorithm. (a) EB framework. (b) EB-RRT framework.

planning in the dynamic environment. However, the generated trajectories are far from optimal, and it is difficult to optimize these time-based trajectories in real time.

Therefore, in this article, by combining the EB method and the time-based RRT algorithm, we propose the EB-RRT algorithm to achieve real-time motion planning while optimizing the current trajectory. As shown in Fig. 1(b), for a specific task, the global planner generates the time-based RRT tree and a feasible initial trajectory according to the world model and the local obstacle. Then, the dynamic replanner optimizes the generated trajectory, and control commands are sent to the robot. The optimization procedure is similar to the EB method. Finally, the position and velocity of the robot will be updated, and this information will be transferred to the global planner again.

Our contributions are summarized as follows:

1) a hierarchical motion planning framework for mobile robot navigation in human–robot coexisting environments;
2) a real-time optimization strategy for the time-based RRT tree by combing the EB method;
3) a proof of the probabilistic completeness and homotopy optimality of the proposed EB-RRT algorithm.

The rest of this article is organized as follows. We introduce the related work in Section II and formulate the motion planning problem in Section III. Then, we describe our EB-RRT algorithm in Section IV. In the following, the results of the simulation experiments and some discussions are presented in Sections V and VI. Finally, we draw conclusions and discuss our future work in Section VII.

II. Related Work

The RRT algorithms have been widely used in robotic motion and path planning area since the 1990s. However, they converge surely to nonoptimal values. Karaman and Frazzoli [8] propose an asymptotically optimal algorithm named RRT*. However, the convergence speed of the RRT* algorithm is slow. Gammell *et al.* [9] propose the Informed-RRT* algorithm to achieve faster convergence. Salzman and Halperin [10] propose the low-bound tree RRT (LBT-RRT) to produce a high-quality trajectory more quickly. There are also other variants of the RRT and RRT* [11]–[15], which do some improvements from different aspects.

For motion planning in dynamic environments, there are a number of different RRT algorithms. These algorithms can be divided into two categories: reactive algorithms and active algorithms. The reactive algorithms compute just one next motion in every instant based on the current condition, which means that they only need to consider the moving obstacles at the current time. Replaning is required whenever the environment changes, but they can cope with highly dynamic and unpredictable environments. Therefore, many variants of the RRT algorithms focus on quickly replanning new trajectories. Bruce and Veloso [16] propose execution extended RRT to achieve real-time randomized path planning. In [17], the multipartite RRT algorithm is presented to deal with the high-dimensional spaces and the time-constrained systems. Otte and Frazzoli [18] introduce the RRTX algorithm to achieve quick replanning in the unpredictably dynamic environment. However, the replanning algorithms are required to maintain a high frequency of updates to cope with moving obstacles.

The active algorithms utilize the trajectory prediction of the moving obstacles in the dynamic environment and generate the trajectory, which can actively avoid the moving obstacles in the future. Ferguson and Stentz [19] propose anytime RRTs to repair the tree branches. Due to the influence of the moving obstacles, the invalid nodes are deleted, and new nodes are searched to connect the root with the branches that have been isolated. In partial motion planning [20], time constraints are explicitly taken into account. The node that is safe with respect to the known trajectories of moving obstacles will be added to the tree. At the same time, another partial path is generated from the end of the previously generated path. Inspired by these two methods, Fulgenzi *et al.* [7] propose a time-based RRT algorithm named Risk-RRT to link the planning and navigation methods with a probabilistic collision risk function. However, the optimization of the planned trajectory is not considered. Compared with the reactive algorithms, the active algorithms can guarantee global optimization to an extent, but a fast and efficient optimization of the planned trajectory is required to deal with the dynamic environment.

Both the reactive and active algorithms are widely used in the robot motion planning field, and they focus on different emphases. In this article, we concentrate on the further improvement of the planned trajectory for the active algorithms.

III. Problem Formulation

In this section, we first briefly formulate the motion planning problem and then use the Risk-RRT as an example to introduce the time-based RRT trees.

A. Motion Planning

The basic motion planning problem, known as the piano mover's problem, can be defined as follows. Let $\mathcal{X} \in \mathbb{R}^n$

be the state space. The obstacle space and the free space are denoted as $\mathcal{X}_{\text{obs}}$ and $\mathcal{X}_{\text{free}}$, respectively. In a dynamic environment, the state space is time-varying, so they are denoted as $\mathcal{X}_{\text{obs}}(t)$ and $\mathcal{X}_{\text{free}}(t)$. Let x_{init} be the initial state and x_{goal} be the goal state. Usually, a goal region $\mathcal{G}(x_{\text{goal}}, t) = \{x \in \mathcal{X}_{\text{free}} | \; ||x - x_{\text{goal}}|| < r\}$ is used. The aim is to compute a trajectory $\sigma : [0, T] \to \mathcal{X}_{\text{free}}$ such that $\sigma(0) = x_{\text{init}}$ and $\sigma(T) \in \mathcal{G}(x_{\text{goal}}, T)$.

Let $\mathcal{U} \in \mathbb{R}^m$ be the control space for the robot. The planned trajectory corresponds to a set of control inputs $u : [0, T] \to \mathcal{U}$. The robot dynamics is represented as

$$\sigma(t+1) = g(\sigma(t), u(t)) \tag{1}$$

where $\sigma(t)$ and $\sigma(t+1)$ are two adjacent states.

Let Σ be the set of all feasible trajectories. The cost function $c(\sigma)$ maps each feasible trajectory σ to a positive real number $\mathbb{R}$. Therefore, the optimal motion planning problem can be defined as

$$\begin{aligned} \sigma^* = \; & \arg\min_{\sigma \in \Sigma} c(\sigma) \\ & \text{s.t. } \sigma(0) = x_{\text{init}} \\ & \quad \sigma(T) \in \mathcal{G}(x_{\text{goal}}, T) \\ & \quad \sigma(t) \in \mathcal{X}_{\text{free}}(t) \quad \forall t \in [0, T]. \end{aligned} \tag{2}$$

In this article, the cost function between two states is defined as follows:

$$\text{Cost}(x_1, x_2, v_1) = \alpha_1 ||x_2 - x_1|| + \alpha_2 \arccos \frac{\overrightarrow{v_1} \cdot \overrightarrow{x_1 x_2}}{|\overrightarrow{v_1}||\overrightarrow{x_1 x_2}|} \tag{3}$$

where α_1 and α_2 are two predefined constants, v_1 is the robot linear velocity at x_1, $|| \cdot ||$ is the Euclidean distance between x_1 and x_2, and $\overrightarrow{x_1 \; x_2}$ is the vector from x_1 to x_2. α_1 and α_2 are used to balance the effect of the Euclidean distance and angle difference. In this article, $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$. Therefore, the cost function of the planned trajectory is

$$c(\sigma) = \sum_{t=0}^{T-1} \text{Cost}(\sigma(t), \sigma(t+1), v(t)). \tag{4}$$

B. Time-Based RRT Tree

The basic structure of a time-based tree and its reconnection process are shown in Fig. 2. The information on each node can be defined as a tuple $(x, x_p, x_c, n, t, u, \mathcal{U}, P(t))$.

1) The state (position and orientation) of the robot. For simplification, we also use x to denote the state.
2) The parent and child node, x_p and x_c, of the current node. For the node x_2 in Fig. 2, x_0 and x_5 are its parent and child node, respectively.
3) The depth n, which measures the *depth* distance between the root node and the current node. In Fig. 2, $n(x_0) = 0$ and $n(x_3) = 2$.
4) The timestamp $t = t_0 + n * \Delta t$. The tree root x_0 has a timestamp t_0, and the time increment between two nodes is Δt.
5) The control input u and possible control outputs $\mathcal{U}$. With u, the robot can move from the state of its parent node to current state: $x = g(x_p, u)$. $\mathcal{U}$ means all the possible control outputs from the state of the current node.
6) The probabilistic collision risk $P(t)$ [7] at timestamp t due to the static and moving obstacles

$$P(t) = P_s + (1 - P_s) \cdot P_d(t) \tag{5}$$

and

$$P_d(t) = 1 - \prod_{m=1}^{M} (1 - P_d(p_m(t))) \tag{6}$$

where P_s denotes the collision probability due to the static obstacles, $P_d(t)$ denotes the collision probability due to the moving obstacles at timestamp t (the trajectory of moving obstacle is represented by the Gaussian process), and $P_d(p_m(t))$ denotes the collision probability due to the planned trajectory of the moving obstacle p_m at timestamp t.

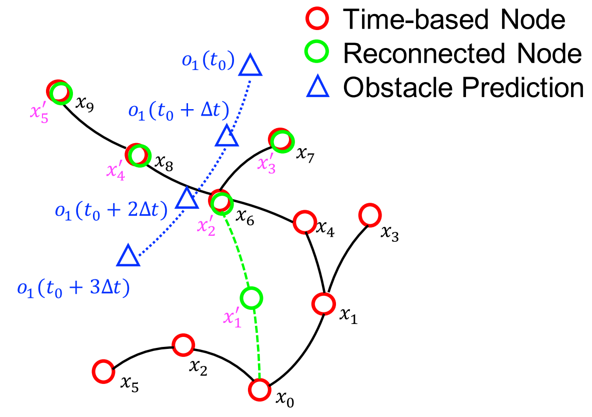


Fig. 2. Reconnection of the time-based tree in dynamic environment.

Just like the basic RRT tree, the time-based RRT tree continuously grows by sampling random nodes. However, it deletes the node that has a high probabilistic collision risk or has a smaller timestamp than that of the current tree root. Obviously, the generated trajectory is nonoptimal, e.g., the trajectory σ ($\sigma(0) = x_0, \sigma(T) = x_9$) in Fig. 2. To optimize the generated trajectory, an intuitive idea is to reconnect the nodes on the trajectory with less cost. However, the reconnection process is difficult.

As shown in Fig. 2, all branches related to the reconnected nodes need to update their information, which brings a large increment on the computation. For example, x_6 is reconnected to the tree root through the new node x_1'. Then, x_6 becomes x_2', and all information needs to be updated. Similarly, the nodes x_7, x_8, and x_9 are also influenced by the reconnection process, and they are updated to be x_3', x_4', and x_5', respectively. In addition, some branches need to be deleted after the reconnection. As shown in Fig. 2, the edge linking x_4 to x_2' needs to be deleted because x_4 and x_2' have the same timestamp. In terms of the obstacle prediction, at the timestamp $t = t_0 + 2\Delta t$, x_2' has a high collision risk so all branches generated from x_2' need to be deleted. Therefore, many useful branches are wasted, and we need to explore again the areas covered by these branches, which brings some repeated work and increases the computation.

Algorithm 1 EB-RRT: Global Planner

```
Input  : $x_r$, $\mathcal{G}(x_{goal}, t)$ and $Map$
Output: $\sigma$
1  $\sigma = \emptyset,\ \mathcal{T} = \emptyset$;
2  $x_{init} = x_r,\ goal = \mathcal{G}(x_{goal}, t)$;
3  $t_0 \leftarrow$ clock();
4  while ($goal\ not\ reached$) do
5      observe($x_r$);
6      delete unreachable nodes($\mathcal{T}, x_r, t_0$);
7      observe($moving\_obs$);
8      $t_0 \leftarrow$ clock();
9      predict $moving\_obs$ at $t_0, \dots, t_0 + N \cdot \Delta t$;
10     if ($environment\ different$) then
11         update($Map, \mathcal{T}, x_r, moving\_obs$);
12     while ($clock() < t_0 + \Delta t$) do
13         grow($\mathcal{T}$);
14     $\sigma_{heu}$ = choose best trajectory in $\mathcal{T}$;
15     $\sigma \leftarrow$ Dynamic Re-planner($\sigma_{heu}$);
16     $t_0 \leftarrow$ clock();
17     if ($\sigma == \emptyset$) then
18         brake();
19     else
20         move along $\sigma$ for one step;
```

Algorithm 2 EB-RRT: Grow($\mathcal{T}$)

```
Input  : $\mathcal{T}$
Output: New $\mathcal{T}$ after growing
1  $x_{rand} \leftarrow$ randomSampling();
2  $x_{nearest} \leftarrow$ chooseParent($x_{rand}, \mathcal{T}$);
3  $x_{new} \leftarrow$ steer($x_{nearest}, x_{rand}$);
4  if $obstacleFree(x_{nearest}, x_{new})$ then
5      $P(t) \leftarrow$ computeRisk($x_{new}, t$);
6      if $P(t) < P_{threshold}$ then
7          $\mathcal{T} \leftarrow$ extend($\mathcal{T}, x_{new}$);
```

Algorithm 3 EB-RRT: Dynamic Replanner

```
Input  : Heuristic trajectory $\sigma_{heu}$
Output: Optimized trajectory $\sigma$
1  $can\_optimize \leftarrow$ geodesicCheck($\sigma_{heu}$);
2  $\sigma_{tmp} = \sigma_{heu}$;
3  if ($can\_optimize == false$) then
4      $\sigma = \sigma_{tmp}$;
5      return $\sigma$;
6  while ($can\_optimize == true$) do
7      $\sigma$.clear();
8      $\sigma$.add($\sigma_{tmp}$.at(0));
9      for ($i = 1; i < \sigma_{tmp}.size() - 1; i++$) do
10         $x_{i-1} \leftarrow \sigma_{tmp}$.at($i-1$);
11         $x_i \leftarrow \sigma_{tmp}$.at($i$);
12         $x_{i+1} \leftarrow \sigma_{tmp}$.at($i+1$);
13         $f = k \cdot (\frac{x_{i-1}-x_i}{||x_{i-1}-x_i||} + \frac{x_{i+1}-x_i}{||x_{i+1}-x_i||})$;
14         $x_{new} = x_i + \alpha f$;
15         $x_{free} \leftarrow \sigma$.end();
16         $b_{control} \leftarrow$ controlCheck($x_{free}, x_{new}, x_{i+1}$);
17         $P(t) \leftarrow$ computeRisk($x_{new}, t$);
18         if $P(t) < P_{threshold}$ && $b_{control} == true$) then
19             $\sigma$.add($x_{new}$);
20         else
21             $\sigma$.clear();
22             $\sigma = \sigma_{tmp}$;
23             $can\_optimize = false$;
24             return $\sigma$;
25     $x_{end} \leftarrow \sigma_{tmp}$.end();
26     $\sigma$.add($x_{end}$);
27     $\sigma_{tmp}$.clear();
28     $\sigma_{tmp} = \sigma$;
29     $can\_optimize \leftarrow$ geodesicCheck($\sigma$);
30 return $\sigma$;
```

In order to avoid the aforementioned problems, we propose the EB-RRT algorithm with the two-level planner. The generation of the initial feasible trajectory and its optimization is implemented in the global planner and dynamic replanner, respectively. The structure of the original tree is not destroyed, and the dynamic replanner will output control commands to direct the robot to move in the dynamic environment.

IV. EB-RRT Algorithm

In this section, we introduce the EB-RRT algorithm on the basis of the Risk-RRT algorithm [7] in Section IV-A. Then, the completeness and optimality of the EB-RRT algorithm are proved in Section IV-B.

A. EB-RRT

As shown in Algorithms 1 and 3, the EB-RRT algorithm consists of two planners. The global planner focuses on perceiving the dynamic environment and planning a feasible heuristic trajectory, while the dynamic replanner is responsible for optimizing the heuristic trajectory. Finally, the robot is directed to move along the optimized trajectory. In fact, the robot has a limited time to perform trajectory planning in a dynamic environment because of the moving obstacles. Therefore, the partial motion planning method [20] is used to implement the trajectory planning and robot execution in parallel. In each time step, the robot x_r moves along the generated partial trajectory σ for one step, while the environment Map, the time-based tree $\mathcal{T}$, the heuristic trajectory σ_{heu}, and the optimized trajectory σ are all updated with the information from the robot perception. When the time step is over, σ is used for robot execution, and a new round of planning and execution begins.

In Algorithm 1, the robot first perceives the current environment and obtains its state x_r. The nodes whose timestamps are earlier than t_0 and their child nodes are deleted. In the following, the moving obstacles are predicted in N time steps, where N is the maximum depth of the time-based tree.

The environment changes due to the moving obstacles and the robot movement, and then, all related variables are updated. Then, $\mathcal{T}$ grows during the time step Δt, and the details of the growing process are provided in Algorithm 2. First, the global planner randomly samples a node in the whole state space as x_{rand}. Second, a node located on the existing tree $\mathcal{T}$ is selected as the nearest node x_{nearest} to try connecting x_{rand} in terms of (3). In the following, x_{rand} is adjusted to x_{new} through a steering function. Third, if there is no obstacle between x_{new} and x_{nearest} and the probabilistic collision risk of x_{new} at timestamp t does not exceed the threshold $P_{\text{threshold}}$, then x_{new} is connected, and $\mathcal{T}$ is extended to become a new $\mathcal{T}$. By querying the current tree, a heuristic trajectory σ_{heu} is obtained. The dynamic replanner optimizes σ_{heu} and outputs the optimized trajectory σ, which directs the robot to move for one-time step. The whole procedure is iterated until the robot reaches the goal region. The function **brake()** indicates that if there is no feasible trajectory, the robot will stop in place until a newly feasible trajectory is generated.

In Algorithm 3, the dynamic replanner optimizes the heuristic trajectory σ_{heu} based on the EB method [6]. It is noted that different from the general reconnection method, the optimization procedure does not affect the time-based tree structure. With σ_{heu}, the dynamic replanner obtains a new trajectory σ to direct the robot navigation, while the time-based tree grows continuously to update σ_{heu}. In this way, no nodes on the tree are wasted, and no extra computation is required compared with the general reconnection method.

In the EB method, a trajectory is considered as an EB consisting of a series of bubbles. Then, with the internal contraction force and the external repulsive force from the obstacles, the EB is deformed until equilibrium. In the EB-RRT algorithm, the heuristic trajectory σ_{heu} is also optimized using the internal contraction force and the external repulsive force.

First, the global planner continuously samples nodes to construct a trajectory. Based on the collision detection algorithm, each node is collision-free in the current configuration space. In addition, as we mentioned in Section III-B, each node x has a property called the probabilistic collision risk $P(t)$ at any timestamp. In the EB method, the boundary of the bubble cannot touch the obstacles in the deformation process. Similarly, in the EB-RRT algorithm, the function **computeRisk**(x_{new}, t) in Line 17 computes the probabilistic collision risk for each node. If the probabilistic collision risk is larger than the threshold, the corresponding node will be not added to the optimized trajectory σ. It guarantees that each node on σ is safe for robot navigation. Therefore, the probabilistic collision risk plays a role of external repulsive force to make the node far away from the obstacles.

Second, we apply the internal contraction force in these nodes located on the heuristic trajectory, which is continuously deformed until equilibrium. The internal contraction force f is defined as

$$f = k \cdot \left(\frac{x_{i-1} - x_i}{||x_{i-1} - x_i||} + \frac{x_{i+1} - x_i}{||x_{i+1} - x_i||} \right) \tag{7}$$

and then each node located on the heuristic trajectory σ_{heu} is continuously adjusted in terms of the following equation:

$$x_{\text{new}} = x_i + \alpha f \tag{8}$$

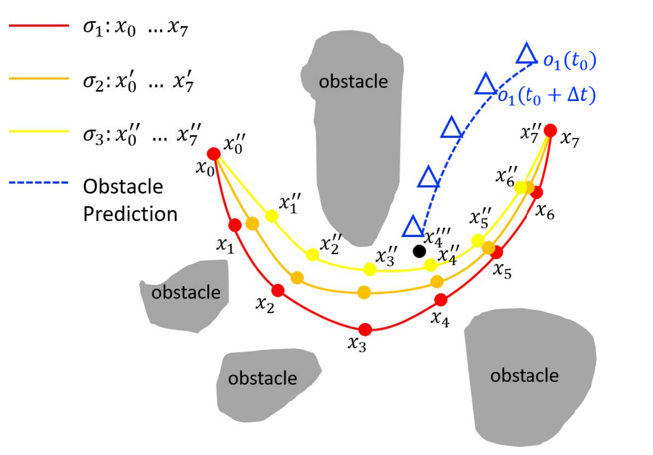


Fig. 3. Illustration of the EB-RRT algorithm. The gray polygons and blue triangles denote the static and moving obstacles, respectively. Three lines with different colors denote the generated trajectory in the optimization process.

where k and α are two positive scaling factors. In this article, $k = 1$ and $\alpha = 1$. Theoretically, increasing the value of k and α should accelerate the optimization process of motion planning. However, when we enlarge k to 3 and 5, or enlarge α to 3 and 5, or enlarge them together, the improvement of the program time cost is smaller than 1%. The reason is that the heuristic trajectory σ_{heu} consists of a small number of nodes, and the execution speed of the program is very fast. The physical interpretation of the internal contraction force is a series of springs between the nodes. Consider σ_{heu} as an EB, and the force from each spring is normalized to reflect a uniform tension along σ_{heu}. Then, x_i is updated to x_{new} using f.

Fig. 3 illustrates the optimization process. The red line, orange line, and yellow line denote the heuristic trajectory σ_1, the intermediate trajectory σ_2, and the final trajectory σ_3, respectively. In the first round, σ_1 is updated to σ_2 with the internal contraction force f. Similarly, σ_2 is updated to σ_3 in the second round. At the timestamp $t_0 + 4\Delta t$, the black node x'''_4 is detected with a high probabilistic collision risk due to the prediction of the moving obstacle. Therefore, x'''_4 is not valid, and the optimization process is terminated. σ_3 becomes the final optimized trajectory.

The functions **geodesicCheck**(σ_{heu}) in Line 1 and **controlCheck**$(x_{\text{free}}, x_{\text{new}}, x_{i+1})$ in Line 16 are also associated with the optimization process. In general, any optimal trajectory in a metric space is a geodesic. Thus, in the function **geodesicCheck**(σ_{heu}), if the internal contraction force f satisfies the following condition:

$$f_i < f_{\text{threshold}} \quad \forall i \in [1, I - 1] \tag{9}$$

where $I - 1$ denotes the number of nodes on the heuristic trajectory σ_{heu}, it means that the current trajectory is already geodesic. Then, the function **geodesicCheck**(σ_{heu}) will return false. The function **controlCheck**$(x_{\text{free}}, x_{\text{new}}, x_{i+1})$ detects the feasibility that the robot moves from x_{free} to x_{i+1} via x_{new}. If the required linear velocity, linear acceleration, angular velocity, and angular acceleration do not exceed the limits, it means that robot can successfully achieve the aforementioned movement. Then, the function **controlCheck**$(x_{\text{free}}, x_{\text{new}}, x_{i+1})$ returns true. In this article, the constraints include the linear velocity $v \in [0, 1.0]$ (m/s), the linear acceleration $a \in [0, 1.0]$ (m/s^2), the angular velocity $\omega \in [0, 0.5]$ (rad/s), and the angular acceleration $\alpha \in [0, 0.5]$ (rad/s^2).

Therefore, the optimization process will be terminated due to three reasons.

1) The internal contraction force between any two neighbor nodes on the trajectory is less than $f_{\text{threshold}}$, which means that the function **geodesicCheck**(σ_{heu}) returns false.
2) The kinematic and dynamic constraints make the optimization process not going further, which means that the function **controlCheck**($x_{\text{free}}, x_{\text{new}}, x_{i+1}$) returns false.
3) The probabilistic collision risk of the new generated node exceeds the threshold, which means that the value of $P(t)$ from the function **computeRisk**(x_{new}, t) is larger than $P_{\text{threshold}}$ and the generated node x_{new} is not valid.

We can see that in the whole optimization process, the structure of the original time-based tree does not change, and no branches are wasted. In addition, no extra computation is required to modify the node information on the time-based tree. Compared with general reconnection methods, we obtain a relatively independent final trajectory based on σ_{heu} and avoid the aforementioned problems.

B. Completeness and Optimality Analysis

As we all know, the RRT algorithm is probabilistic complete, which means that the probability of finding a feasible trajectory approaches to one if it exists with the number of iterations approaching infinity. In this article, the EB-RRT algorithm uses the RRT trajectory as the heuristic trajectory. Therefore, the EB-RRT algorithm retains the same probabilistic completeness as that of the RRT.

In [8] and [21], the proposed RRT* algorithm is proven to be asymptotically optimal. The EB-RRT algorithm also has a similar property, namely homotopy optimality. In the robot navigation progress, the heuristic trajectory is partially planned in each time step. Thus, the homotopy optimality of the EB-RRT algorithm refers to the current partially planned trajectory instead of the whole trajectory connecting the start and goal positions. In other words, based on the current heuristic trajectory, the EB-RRT algorithm can generate an optimized trajectory guaranteeing the homotopy optimality. In the following, we prove that the EB-RRT algorithm is optimal in the homotopy class of the current heuristic trajectory.

First, we give the definition of the homotopy class of trajectories and the homotopy optimal trajectory.

Definition 1 (Homotopy Class of Trajectories [22]): Two trajectories, σ_1 and σ_2, are said to be in the same **Homotopy Class** iff one can be smoothly deformed into the other without intersecting obstacles. Otherwise, they belong to different homotopy classes.

Let $\hat{\Sigma}$ denote a homotopy class of trajectories, and $\hat{\sigma}$ is an arbitrary trajectory in $\hat{\Sigma}$. A homotopy optimal trajectory $\hat{\sigma}^*$ is defined as

$$\hat{\sigma}^* = \arg\min_{\hat{\sigma}\in\hat{\Sigma}} c(\hat{\sigma})$$

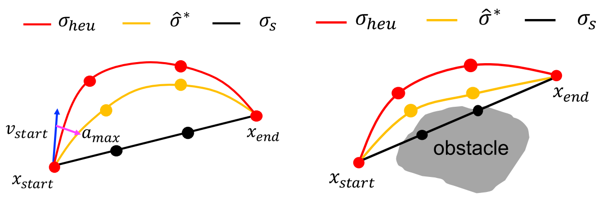


Fig. 4. Two types of the final optimized trajectory. The blue arrow denotes the current velocity of the robot at x_{start}, and the magenta arrow denotes the maximum acceleration of the robot. (a) Acceleration constraint. (b) Obstacle constraint.

$$\begin{aligned} \text{s.t. } & \hat{\sigma}(0) = x_{\text{init}} \\ & \hat{\sigma}(T) \in \mathcal{G}(x_{\text{goal}}, T) \\ & \hat{\sigma}(t) \in \mathcal{X}_{\text{free}}(t) \quad \forall t \in [0, T]. \end{aligned} \tag{10}$$

Then, we need to prove the following lemma.

Lemma 1: The generated trajectory $\hat{\sigma}^*$ from the EB-RRT algorithm is the optimal trajectory in the homotopy class consisting of the heuristic trajectory.

Proof: As shown in Algorithm 2, we use the EB method to optimize the heuristic trajectory. When the optimization process is over, we obtain an optimized trajectory $\hat{\sigma}^*$. In fact, the shape of the heuristic trajectory has a big impact on the optimization process. Thus, we discuss the homotopy optimality according to different shapes of the heuristic trajectory.

First Case: The Heuristic Trajectory Is a Geodesic: For two arbitrary nodes, a geodesic connecting them is the optimized trajectory in its homotopy class if there is no obstacle between them. Therefore, if the heuristic trajectory is a geodesic, the function **geodesicCheck**(σ_{heu}) returns false. Therefore, the optimization process is not executed, and the final trajectory $\hat{\sigma}^*$ is the same as the heuristic trajectory. Since the heuristic trajectory is optimal in its homotopy class, $\hat{\sigma}^*$ is optimal in its homotopy class.

Second Case: The Heuristic Trajectory Is ***Not*** *a Geodesic:* Generally, the trajectory between two nodes is not a geodesic due to the kinematic and dynamic constraints or the existence of the obstacles. As shown in Fig. 4, there are two types of final optimized trajectory. Here, we use reduction to absurdity to prove the homotopy optimality of $\hat{\sigma}^*$ in its homotopy class. Suppose that there is another trajectory $\hat{\sigma}_z$ such that $\hat{\sigma}_z \neq \hat{\sigma}^*$ and $c(\hat{\sigma}_z) < c(\hat{\sigma}^*)$, where $c()$ refers to the cost function defined in Section III-A.

In Fig. 4(a), the optimization process is terminated due to the maximum acceleration constraint. Because $c(\hat{\sigma})_z < c(\hat{\sigma}^*)$, $\hat{\sigma}_z$ must be located between $\hat{\sigma}^*$ and σ_s, where σ_s is a geodesic connecting the start and end nodes. Obviously, $\hat{\sigma}_z$ does not satisfy the kinematic and dynamic constraints. Therefore, such a trajectory does not exist, which is conflicted with the hypothesis. In Fig. 4(b), the optimization process is terminated because the probabilistic collision risk of the newly generated node exceeds the threshold. Again, if $\hat{\sigma}_z$ exists, it should be located between $\hat{\sigma}^*$ and σ_s. However, the nodes on $\hat{\sigma}_z$ cannot pass the collision check. Then, such a trajectory does not exist, and the hypothesis is not true.

In conclusion, we prove that $\hat{\sigma}^*$ is the optimal trajectory in its homotopy class.

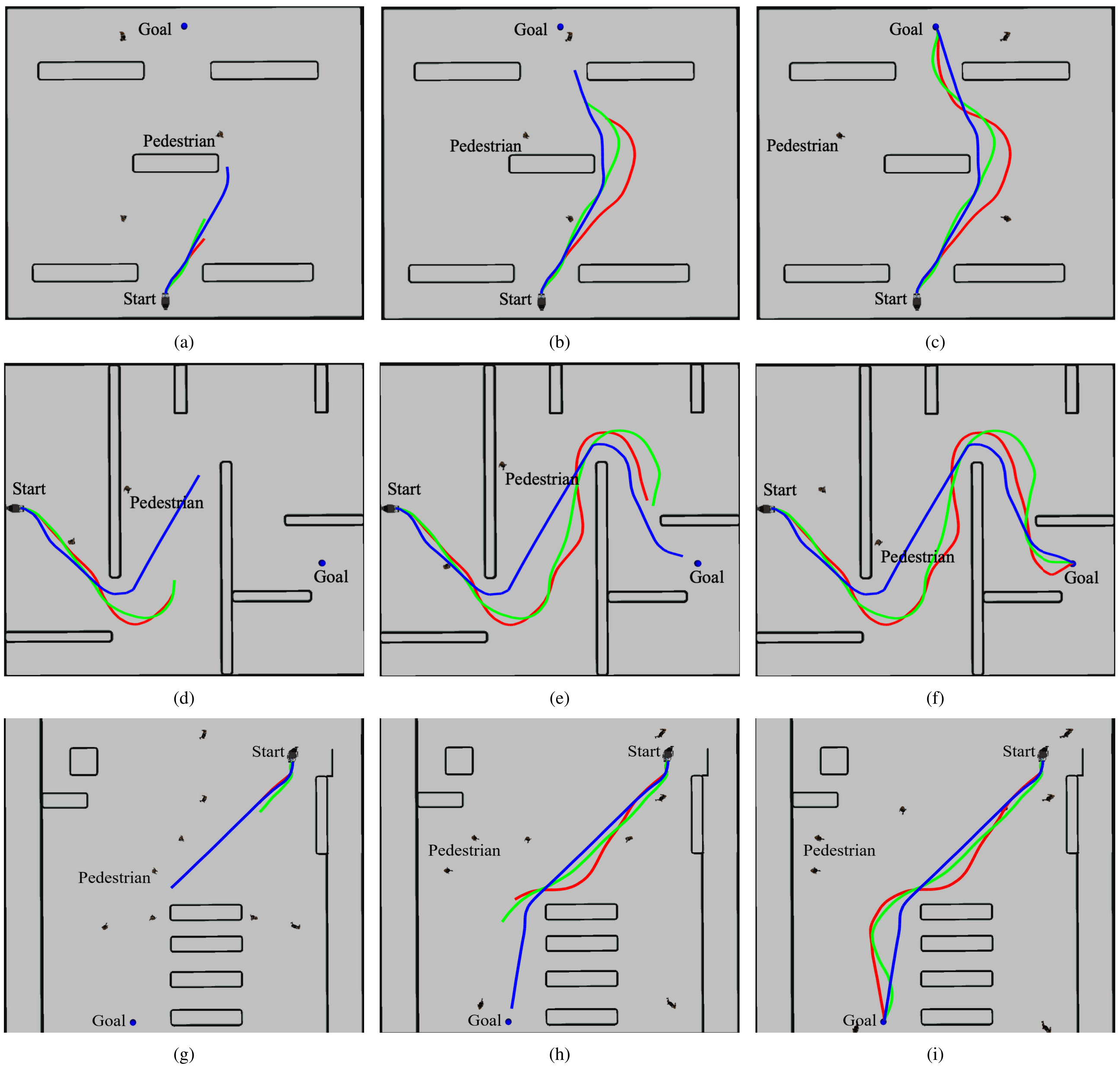


Fig. 5. Simulation scenarios. Each scenario consists of some obstacles and pedestrians, which moves around the current scenario. The initial poses of the robot in three scenarios are $(-1.5, -10, 90°)$, $(-12, 1, 0°)$, and $(8, 9, -90°)$, while the coordinates of the goal are $(0, 10)$, $(10, -3)$ and $(-4, -12)$, respectively. The red, green, and blue lines denote the trajectory generated by Risk-RRT algorithm, Risk-RRT* algorithm, and our proposed EB-RRT algorithm at different timestamps, respectively. (a) Scenario 1, $t = 10.0$ s. (b) Scenario 1, $t = 20.0$ s. (c) Scenario 1, $t = 29.4$ s. (d) Scenario 2, $t = 20.0$ s. (e) Scenario 2, $t = 40.0$ s. (f) Scenario 2, $t = 51.2$ s. (g) Scenario 3, $t = 12.0$ s. (h) Scenario 3, $t = 24.0$ s. (i) Scenario 3, $t = 34.8$ s.

According to the conclusions from the first and second cases, the proof is finished.

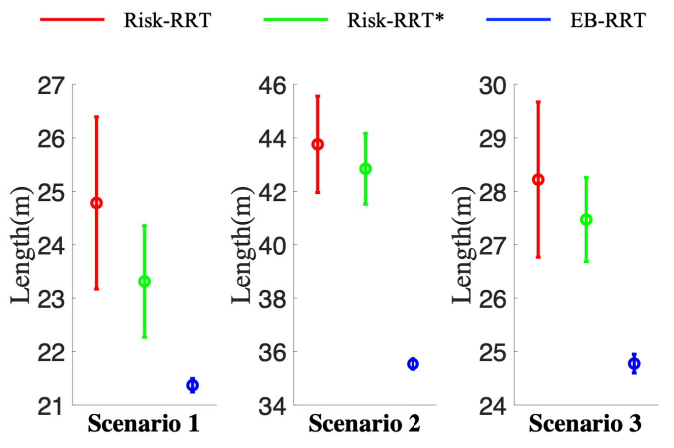
V. Results of Simulation Experiments

In this section, we conduct the simulation experiments in three different scenarios (as shown in Fig. 5) on the basis of robot operating system (ROS). We use Ubuntu 16.04 on an Intel i5-4590 with 8-GB RAM as our experimental platform. The robot parameters in the simulation experiments are listed in the following table:

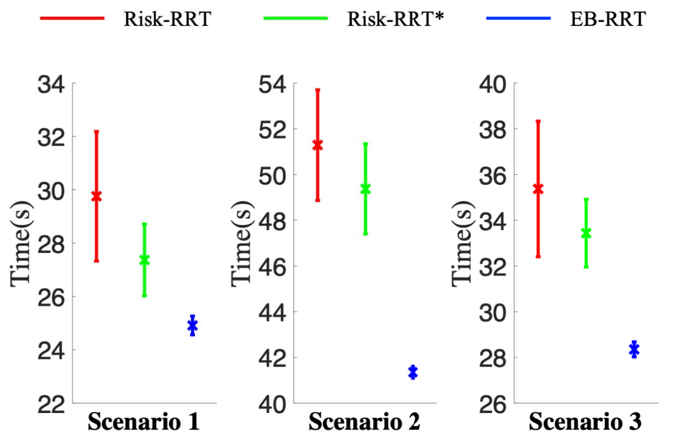
v_m (m/s)	a_m (m/s^2)	ω_m (rad/s)	α_m (rad/s^2)
1.0	1.0	0.5	0.5

Here, v_m, a_m, ω_m, and α_m represent the maximum linear velocity, the maximum linear acceleration, the maximum angular velocity, and the maximum angular acceleration, respectively. As shown in Fig. 5, the sizes of both scenario 1 and scenario 2 are 26 m × 23 m, and the size of scenario 3 is 28 m × 30 m. Three, two and eight pedestrians in each scenario are walking forth and back with the specified velocity, $v_p = 0.5$ m/s. The goal region is defined as $\mathcal{G}(x_{\text{goal}}, t) = \{x \in \mathcal{X}_{\text{free}} \big| \, ||x - x_{\text{goal}}|| < 0.2 \text{ m}\}$. In other words, the robot is defined as having arrived at its goal region when its distance to the goal position is smaller than 0.2 m.

We compare our proposed EB-RRT algorithm with two state-of-the-art risk-based motion planning algorithms, Risk-RRT and Risk-RRT*. In each scenario, the execution is



(a)



(b)

Fig. 6. Simulation experimental results in different scenarios. (a) Trajectory length. (b) Navigation time cost. The circle and cross mark denote the mean value, and the line denotes the standard deviation.

repeated 50 times. We use the navigation time (including the planning time and the robot execution time) and the length of the generated trajectory as the metrics to evaluate the performance of each algorithm.

A. Simulation Experiment Results

The experimental performance of 50 simulation trials is shown in Fig. 6. First, Fig. 6(a) shows the length of the generated trajectory of three different algorithms in three different scenarios. The average trajectory lengths of the EB-RRT algorithm are the shortest among these three methods. Also, the smallest deviation means that the EB-RRT algorithm performs most stably. Second, Fig. 6(b) shows the navigation time cost of three different algorithms in three different scenarios. Compared with the Risk-RRT and Risk-RRT* algorithms, the EB-RRT algorithm uses the shortest time, and the standard deviation of the time cost is also the smallest.

B. Analysis of Simulation Experiment Results

The Risk-RRT* algorithm, as a traditional reconnection method, reduces the trajectory length, but this improvement is limited. Since the robot moves in a dynamic environment, dynamic motion planning requires a real-time response. Therefore, the time allocated for the reconnection process is limited. However, as mentioned in Section III, the Risk-RRT* algorithm needs to do much extra computation to finish the whole reconnection process step by step, which is similar to the RRT* algorithm. Thus, the reconnection process cannot be finished completely in a limited time step, and the improvement is not significant. In Fig. 5, we can also see that in the early time, the Risk-RRT* algorithm performs very well because there are only a small number of nodes and the reconnection process can be finished easily. As the number of nodes increases, the Risk-RRT* does not perform well. However, the EB-RRT algorithm always performs well in the whole navigation process. Depending on the heuristic trajectory, the EB-RRT algorithm quickly obtains an optimized trajectory with the EB method, and the number of nodes (the length of the heuristic trajectory) does not affect the optimization process. This optimized trajectory is independent of the Risk-RRT time-based tree structure, so the tree structure is not affected and no extra computation is required compared with the Risk-RRT* algorithm. Therefore, we can find that the length of the final trajectory generated from the EB-RRT algorithm is much shorter than those from the Risk-RRT and the Risk-RRT* algorithm.

We can also find that in different trials, the performance of the EB-RRT algorithm is different. The reason is that the EB optimization process is based on the partially planned heuristic trajectory on each timestamp and the heuristic trajectories at the same timestamp in different trials are different. The standard deviation of the navigation time cost and the length of the final trajectory indicates that the difference of the heuristic trajectories has an impact on the EB optimization process. In conclusion, compared with the other state-of-the-art algorithms, the EB-RRT algorithm costs the least navigation time and generates the shortest trajectory, which reveals the efficiency and effectiveness of our proposed EB-RRT algorithm.

VI. Discussion

In the time-based RRT tree algorithm, there are mainly three predefined parameters, including the maximum depth N, the time step Δt, and the maximum velocity $v_{\max}$. In order to show the robustness of our algorithm, we further discuss the performance of the Risk-RRT, the Risk-RRT*, and the EB-RRT algorithms under different parameter settings via simulation experiments. We carry out the simulation experiments in scenario 1, as shown in Fig. 5. The start state is $(-1.5, -10, 90°)$, and the goal region is a circle centered at $(0, 10)$ with a radius 0.2 m. For each different parameter setting, 50 times of repeated experiments are carried out, and the navigation time and the length of the generated trajectory are used to evaluate the performance of different algorithms.

In addition, we also test our algorithm in a more challenging environment to further evaluate its performance.

A. Different Maximum Depth N

The maximum depth N is the maximum growth depth of the tree from the current node. As mentioned earlier,

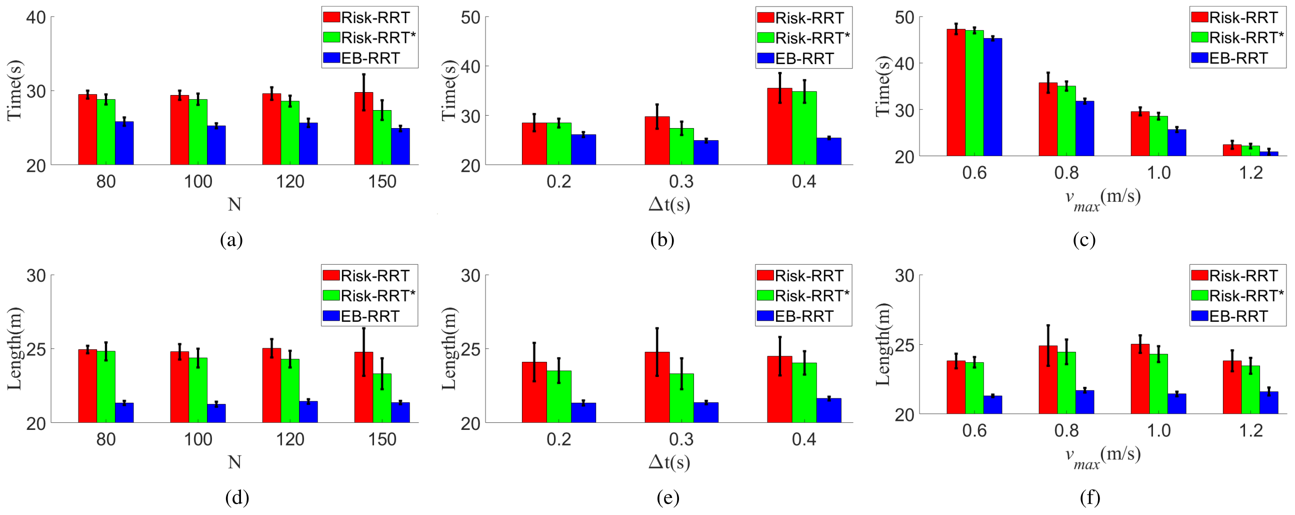


Fig. 7. Simulation experimental results in scenario 1 under a different maximum depth N, time step Δt, and maximum velocity $v_{\max}$. The bar denotes the mean value, and the line denotes the standard variation of 50 times simulation experiments. (a) Navigation time under a different value of N. (b) Navigation time under a different value of Δt. (c) Navigation time under a different value of $v_{\max}$. (d) Trajectory length under a different value of N. (e) Trajectory length under a different value of Δt. (f) Trajectory length under a different value of $v_{\max}$.

the probabilistic collision check will be implemented in a limited period $[t, t + N * \Delta t]$. If N is small, the result of the probabilistic collision check is more confident, but the tree can only grow within a small size. As shown in Fig. 5(a), (d), and (g), all algorithms can only provide a partially generated trajectory due to the limitation of the maximum depth N. If N is large, a big collision check estimation may cause a failure of the motion planning, but the tree can grow within a big size. Therefore, it is a tradeoff to set the value of the maximum depth N in terms of the quality of the trajectory and the accuracy of the collision check estimation.

In Fig. 7(a) and (d), four different maximum depths N are used to evaluate the performance of three algorithms. The bar denotes the mean value, and the line denotes the standard variation of 50 times simulation experiments. In general, when $N = 80$, 100, and 120, the performance of the three algorithms does not change too much. However, when N increases to 150, the standard variations of the Risk-RRT and the Risk-RRT* become larger obviously. The reason is that when N becomes larger, the accuracy of the collision check estimation begins to decline, and the length of the generated trajectory is longer at the present time; then, the quality of the trajectory cannot be guaranteed. However, the EB-RRT still performs well when $N = 150$. Moreover, the navigation time decreases a little because the EB method can optimize a relatively long trajectory at each timestamp and the optimization result becomes better.

In conclusion, under a different maximum depth N, the EB-RRT always achieves the best performance compared with the other algorithms, which reveals the robustness of the EB-RRT algorithm.

B. Different Time Step Δt

First, the time step Δt is the time increment between two nodes. With the same maximum velocity $v_{\max}$, when the Δt is large, the distance between two nodes also becomes large. Second, Δt also denotes the planning time in the navigation process. In the time-based tree algorithm, the planning and execution processes are carried out in parallel. Within Δt, the robot moves from one node to its child node, while the trajectory is updated using the latest information. In order to get a real-time response, the time step Δt is required to be set to a small value.

In Fig. 7(b) and (e), three different time step Δt are used to evaluate the performance of three algorithms. The bar denotes the mean value, and the line denotes the standard variation of 50 times simulation experiments. We can find that the Risk-RRT and Risk-RRT* algorithms are sensitive to the change of Δt. When $\Delta t = 0.4s$, both the mean value and the standard variation of the navigation time cost obviously increase. The reason is that the generated trajectory becomes rough when Δt is large and the quality of the generated trajectory declines. However, the change of Δt has little effect on the EB-RRT algorithm. Among these three algorithms, the EB-RRT algorithm achieves the best performance, which shows the robustness of the EB-RRT algorithm again.

C. Different Maximum Velocity v_{max}

The maximum velocity $v_{\max}$ is an important parameter in the time-based tree algorithm. It directly affects the navigation time cost. Also, when $v_{\max}$ becomes larger, it has a higher requirement for the robot to maintain a real-time response.

In Fig. 7(c) and (f), four different maximum velocities $v_{\max}$ are used to evaluate the performance of three algorithms. The bar denotes the mean value, and the line denotes the standard variation of 50 times simulation experiments. As for the length of the generated trajectory, all algorithms are not very sensitive to the change of $v_{\max}$. The navigation time cost naturally decreases as $v_{\max}$ increases. We can find that the EB-RRT algorithm always performs best in the navigation time cost and the trajectory length for each different maximum velocity $v_{\max}$, which reveals the stability and robustness of the EB-RRT algorithm.

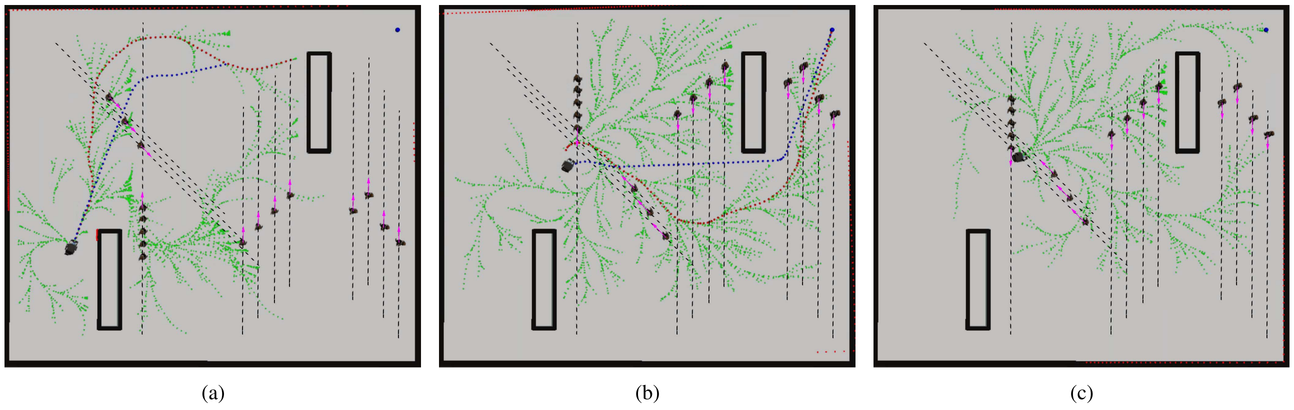


Fig. 8. Simulation experiment in a challenging environment. The red and blue dotted lines denote the heuristic trajectory and optimized trajectory, respectively; 16 pedestrians are moving along the black dotted line, and the magenta arrow denotes the current moving direction. (a) $t = 10.1$ s. (b) $t = 27.2$ s. (c) $t = 31.4$ s.

D. Simulation Experiment in a Challenging Scenario

In order to further reveal the robustness of our algorithm, we test our algorithm in a more challenging scenario, crowded environment. As shown in Fig. 8, 16 pedestrians are adopted to demonstrate the crowded environment. They move along the black dotted line, and the magenta arrow denotes the current moving direction. It is noted that their moving direction changes in Fig. 8(c). We test our algorithm 50 times, and the success rate is 86%. For the other 14% of trails that are not successful, the reason is that the global planner cannot compute a feasible trajectory due to insufficient reaction time. Therefore, the robot cannot avoid the collision, which is considered as a failure.

In our algorithm, the global planner continuously searches the feasible trajectory at each iteration. When there is no feasible trajectory due to the influence of the moving pedestrian, the robot will stay in place until the global planner finds a new feasible trajectory. However, sometimes, the global planner will fail due to insufficient reaction time before the collision. A failure example is shown in Fig. 8. First, in Fig. 8(a) and (b), the planner can generate a collision-free trajectory for the robot with the trajectory prediction for the moving pedestrians, where a probabilistic prediction method using a Gaussian process is implemented. However, in Fig. 8(c), there is a sudden reversal of motion of the pedestrians, and the robot cannot quickly make a correct prediction. Therefore, the collision is not avoided, and path planning fails.

In order to solve this problem, we consider two improved methods in future work. The first method is taking into account the uncertainty in the path planning process. The uncertainty includes the pedestrian trajectory prediction uncertainty, measurement uncertainty from the sensors, and robot execution uncertainty, among others. These uncertainties should be considered to generate a safe trajectory. The second method is increasing the clearance between the robot and the obstacles in the path planning process. In this way, the robot can get more time to avoid the coming collision. However, the trajectory length will also increase. More time and energy are required for the robot to achieve the designated path planning work. Thus, it is a tradeoff between increasing the clearance and maintaining an energy-optimal or time-optimal trajectory.

VII. Conclusion and Future Work

In this article, the EB-RRT algorithm consisting of two planners is proposed to deal with the mobile robot motion planning in dynamic environments. In the global planner, a heuristic trajectory is obtained, and in the dynamic replanner, the heuristic trajectory is optimized with the EB method. The final generated trajectory has been proven to be homotopy optimal. Simulation experiments indicate that our algorithm achieves the best performance in the navigation time cost and the trajectory length compared with the other state-of-the-art algorithms.

Essentially, in the EB-RRT algorithm, the EB optimization process provides a new cost function to guide the tree growth. However, different from the conventional method where a large number of old nodes are deleted and many new nodes are added, our algorithm only adds some new nodes along the generated trajectory to the tree to form a better trajectory. It not only considers the Euclidean distance and the angle difference between any two neighbor nodes on the trajectory but also takes into account the internal contraction force and external repulsive force acting on the current trajectory.

Recently, the inverse reinforcement learning methods [23]–[25] are widely used to learn a new cost function from the demonstrations in the robot social navigation. In the future, we plan to apply the inverse reinforcement learning method into our global planner to guide the generation of the initial trajectory. The learned cost function can satisfy different requirements in different scenarios. For example, sometimes, the clearance with the obstacles is more important, or the smooth level of the trajectory is required. For different initial trajectories, we still can use our dynamic replanner to optimize it. Naturally, another intuitive prospect is to use the reinforcement learning method to imitate humans to optimize the initial trajectory.

Acknowledgment

This work was finished when the first author visited the Robotics Laboratory, Stanford University, whose hosting is gratefully acknowledged. The authors also appreciate Shameek Ganguly and Margot Vulliez's advice on this work.

REFERENCES

[1] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Germany: Springer, 2016.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

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

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

[5] 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, Aug. 1996.

[6] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1993, pp. 802–807.

[7] C. Fulgenzi, A. Spalanzani, C. Laugier, and C. Tay, "Risk based motion planning and navigation in uncertain dynamic environment," Res. Rep., 2010, p. 14. [Online]. Available: https://hal.inria.fr/inria-00526601

[8] 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.

[9] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 2997–3004.

[10] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality motion planning," *IEEE Trans. Robot.*, vol. 32, no. 3, pp. 473–483, Jun. 2016.

[11] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural RRT*: Learning-based optimal path planning," *IEEE Trans. Autom. Sci. Eng.*, early access, Mar. 16, 2020, doi: 10.1109/TASE.2020.2976560.

[12] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 3067–3074.

[13] J. Wang, W. Chi, M. Shao, and M. Q.-H. Meng, "Finding a high-quality initial solution for the RRTs algorithms in 2D environments," *Robotica*, vol. 37, no. 10, pp. 1677–1694, 2019.

[14] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT-smart: Rapid convergence implementation of RRT towards optimal solution," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2012, pp. 1651–1656.

[15] J. Wang and M. Q.-H. Meng, "Optimal path planning using generalized Voronoi graph and multiple potential functions," *IEEE Trans. Ind. Electron.*, early access, Jan. 1, 2020, doi: 10.1109/TIE.2019.2962425.

[16] J. Bruce and M. M. Veloso, "Real-time randomized path planning for robot navigation," in *Robot Soccer World Cup*. Berlin, Germany: Springer, 2002, pp. 288–295.

[17] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2007, pp. 1603–1609.

[18] M. Otte and E. Frazzoli, "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 797–822, Jun. 2016.

[19] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 5369–5375.

[20] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Aug. 2005, pp. 2210–2215.

[21] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, and M. Pavone, "Revisiting the asymptotic optimality of RRT*," 2019, *arXiv:1909.09688*. [Online]. Available: http://arxiv.org/abs/1909.09688

[22] S. Bhattacharya, V. Kumar, and M. Likhachev, "Search-based path planning with homotopy class constraints," in *Proc. 3rd Annu. Symp. Combinat. Search*, 2010, pp. 1–8.

[23] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 729–736.

[24] K. Shiarlis, J. Messias, and S. Whiteson, "Rapidly exploring learning trees," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 1541–1548.

[25] N. Pérez-Higueras, F. Caballero, and L. Merino, "Teaching robot navigation behaviors to optimal RRT planners," *Int. J. Social Robot.*, vol. 10, no. 2, pp. 235–249, Apr. 2018.

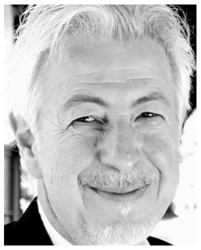
Jiankun Wang received the B.E. degree in automation from Shandong University, Jinan, China, in 2015, and the Ph.D. degree from the Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong, in 2019.

During his Ph.D. degree, he spent six months at Stanford University, Palo Alto, CA, USA, as a Visiting Student Scholar, supervised by Prof. Oussama Khatib. He is currently a Post-Doctoral Fellow with the Department of Electronic Engineering, The Chinese University of Hong Kong. His current research interests include motion planning and control, human–robot interaction, and machine learning in robotics.

Max Q.-H. Meng (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Victoria, Victoria, BC, Canada, in 1992.

He is a Professor of electronic engineering with The Chinese University of Hong Kong, Hong Kong, since 2002, after working for ten years with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada, as the Director of the Advanced Robotics and Teleoperation Laboratory and holding the positions of an Assistant Professor in 1994, an Associate Professor in 1998, and a Professor in 2000. He holds honorary positions as a Distinguished Professor with the State Key Laboratory of Robotics and Systems, Harbin Institute of Technology, Harbin, China, a Distinguished Provincial Professor with the Henan University of Science and Technology, Luoyang, China, and the Honorary Dean of the School of Control Science and Engineering, Shandong University, Jinan, China. He has published more than 500 journal articles and conference papers and served on many editorial boards. His research interests include robotics, perception and sensing, human–robot interaction, active medical devices, biosensors and sensor networks, and adaptive and intelligent systems.

Dr. Meng has been serving as an Elected Member of the Administrative Committee of the IEEE Robotics and Automation Society. He received the IEEE Third Millennium Medal Award.



Oussama Khatib (Life Fellow, IEEE) received the Ph.D. degree in electrical engineering from Sup'Aero, Toulouse, France, in 1980.

He is currently a Professor of computer science with Stanford University, Palo Alto, CA, USA. His work on advanced robotics focuses on methodologies and technologies in human-centered robotics, including humanoid control architectures, human motion synthesis, interactive dynamic simulation, haptics, and human-friendly robot design. He is a Coeditor of the *Springer Tracts in Advanced Robotics* Series. He coedited the *Springer Handbook of Robotics*, which received the PROSE Award.

Dr. Khatib was a recipient of the Japan Robot Association (JARA) Award in Research and Development. In 2010, he received the IEEE RAS Pioneer Award in Robotics and Automation for his fundamental pioneering contributions in robotics research, visionary leadership, and life-long commitment to the field. He received the 2013 IEEE RAS Distinguished Service Award in recognition of his vision and leadership for the Robotics and Automation Society, in establishing and sustaining conferences in robotics and related areas, publishing influential monographs and handbooks and training and mentoring the next generation of leaders in robotics education and research. In 2014, he received the 2014 IEEE RAS George Saridis Leadership Award in Robotics and Automation. He has served on the editorial boards of several journals as well as the chair or the co-chair of numerous international conferences. He is also the President of the International Foundation of Robotics Research (IFRR). He has served as a Distinguished Lecturer.