doi:10.20944/preprints202104.0754.v1

RimJump*: Tangent based shortest path planning for cluttered 3D environment

Zhuo Yao*

***Abstract*—Path planning in 3D environment is a fundamental research area for robots and autonomous vehicles. Based on the principle "the shortest path consists of tangents", RimJump* is proposed as a tangent-based path planning method suitable for finding the shortest path (both off-ground and on-ground) in 3D space (e.g., octomap and point cloud) for mobile platform to follow. RimJump* searches the tangent graph in the form of a path tree and considers the geometrical properties of the locally shortest path. Therefore, the method can provide all of the locally shortest paths that connect the starting point and the target, including the globally shortest path. And the time cost of RimJump* is insensitive to map scale increases in comparison to methods that search the whole passable space rather than the surface of the obstacle, e.g., Dijkstra and A*. In the Results, RimJump* is compared with other methods in terms of path length and time cost.**

***Index Terms*—3D path planning, tangent, locally shortest path.**

I. Introduction

RECENTLY, autonomous mobile robots, such as UAV, quadruped robots, and biped robots, have attracted increasing attention for spreading their use in exploration, commercial delivery services, and military missions, as shown in Fig. 1. Path planning plays an important role in their application, resulting in increasing requirements for 3D path planning methods. The objective of path planning is to find the shortest collision-free path between two positions for robot and drones to follow, including off-ground (e.g., unmanned drones) path planning and on-ground (e.g., quadruped robot) path planning. However, efficient shortest path planning under cluttered 3D environments is still a challenging problem. This manuscript mainly focuses on both 3D path planning in static environments.

Graph search-based methods that use breadth-first search (BFS), which visits the closest nodes first and then proceeds to more distant nodes, to find the shortest path are widely used in practice. The method was first proposed by Dijkstra in [7], and several variants have since been reported. A*[13, 37, 10] is a heuristic method that reduces the computational cost by introducing an estimation of the future cost. Considering that the grid representation of maps constrains the possible headings of the paths, the paths formed by grid edges can be suboptimal. Theta* [25] generates paths without a heading constraint by checking whether a vertex has line-of-sight to another vertex. However, graph search algorithms use online point-to-point traversal. Thus, their time cost increases sharply

Zhuo Yao was graduated from Beijing Institute of Technology, Beijing, China. Now he is working as a robot navigation engineer.
*Corresponding author. E-mail: 1521232476@qq.com

Fig. 1. A: The NASA Mars Courage Rover. B: BigDog of Boston Dynamics, a rough-terrain cargo-hauling battlefield robot. C: M210 RTK V2 of DJI.

as the map size increases, which becomes a critical problem for a large-scale environment.

The idea of searching a workspace by random sampling is widely implemented in many areas, including path planning. Rapidly exploring random tree (RRT) [19, 31, 32, 36, 6, 15, 1] generates a path that connects the start and target by randomly searching a finite configuration space. Probabilistic road map (PRM) [18, 33, 3, 17] using a sampling-based approach builds a road map and then uses a graph search method (Dijkstra, A*, etc.) to find the lowest-cost path from it. Sampling-based path planning may not always find the optimal solution but can always find a feasible one.

A family of algorithms that can generate a near-optimal path has been proposed by imitating how humans or other natural creatures behave or think. Genetic algorithms (GAs)[14, 27, 9, 2, 11, 12] use the principles of natural selection to create increasingly better individuals via generational iterations and select the optimal individual in the last generation as the final result. Ant colony optimization (ACO) imitates the behavior of ants to find the shortest path from their colony to food by group cooperation [8, 5, 28, 24, 26]. By evaluating the "pheromone" density, the best path can therefore be achieved, which represents the intensity of the trail. Generally speaking, bioinspired path planning methods have strong global search capabilities, but the range of their application is limited by their heavy computational cost.

A visibility graph is a fundamental geometric structure that is useful in many applications, including path planning. Masehian[23] proposed a visibility graph, Voronoi diagram, and potential field integrated path planning method that has the potential to extend to 3D scenarios. Schøler[29, 30] generated a visibility graph from a mesh around an obstacle to obtain an approximately shortest path via graph search. Bygi[4] proposed a method to generate 3D visibility graphs that can be used in 3D path planning. Jiang[16] solved the problem of identifying visible edge sequences and determined the shortest path using the principle of minimum potential energy. The visibility graph does not need online point-by-point traversal nor considers the visibility between obstacles after the graph

doi:10.20944/preprints202104.0754.v1

was constructed. Nevertheless, a visibility graph needed to be reconstructed when the start and target changed while our method does not need to reconstruct the visibility graph. In addition, it is an algorithm limited to maps that only contain polygonal obstacles and cannot be applied on a voxel map directly, which limits its application range, while the proposed method can be applied on a voxel map directly.

Tangent-based path planning has been extensively studied in recent years and has been proven to be efficiently the shortest path under 2D maps efficiently. A tangent graph has a similar structure as a visibility graph, the major difference is it uses the tangent to represent the visibility between the grid on the surface and avoid the constraint about obstacle shape. Liu and Arimoto[20, 22, 21] first proposed the tangent graph (2D) based on the locally shortest path and used the graph search method (Dijkstra, A*, etc) to find the shortest path. The major difference between the tangent graph and the visibility graph is that the tangent graph can be applied to curved obstacles. Then, we proposed a tangent graph based on a 2D path planning method[35, 34] using a path tree to search the graph instead of a graph search. The approach can always find the shortest path, and the time cost is insensitive to the map scale increasing.

However, the above tangent-based path planning is limited to 2D environments, and the major problem is the definition of the 2D tangent is not suitable for a 3D environment. As the definition of the 2D tangent is based on 2D geometry elements. Inheriting the advantages of the tangent graph, this manuscript proposed a definition of tangent that is not limited by the dimensionality of the map and a path planning based on it, which can find the shortest path (off-ground and on-ground) from a 3D environment. Considering the globally shortest path may not be executable and all locally shortest paths contain all ways that connect the start and the target, an approximate solution of the globally optimal trajectory can be generated from them according to kinematics parameters and a trajectory cost function. The trajectory generation is suitable for robots with complex kinematics models, such as bipedal robots and quadruped robots.

The contributions of this paper are as follows: 1, a definition of tangent that is suitable for both voxel (3D) and grid (2D) map is presented; 2, the shortest path planning method (off-ground and on-ground) in Euclidean distance based on the above tangent.

The remainder of this paper is structured as follows: Section 2 includes the proposed algorithm and details about the implementation, including mathematical proof, pseudo code, and complexity analysis, described with several illustrative examples. In Section 3, the results of the proposed method and the comparison with other methods are presented and discussed. Final conclusions, potential applications, and prospective future research are provided in Section 4.

II. Algorithm and Implementation

RimJump* is a shortest path planning method based on the principle that the shortest path consists of tangents. Therefore, the definition of tangent and related proof of the principle is presented in the Mathematical fundamentals. Then, the implementation of RimJump* (both off-ground and on-ground) is described in the Algorithm and data structure section, including the tangent graph construction (offline), the adaptive Breadth-First-Search (online) for the shortest path.

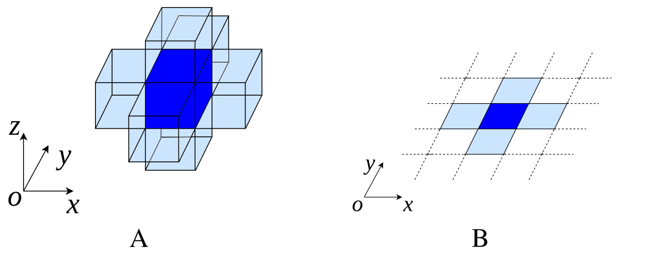


Fig. 2. These figures show the neighboring area of a voxel (3D) and grid (2D). Fig A for 3D and Fig B for 2D respectively. g is deep blue while *U(g)* is light blue.

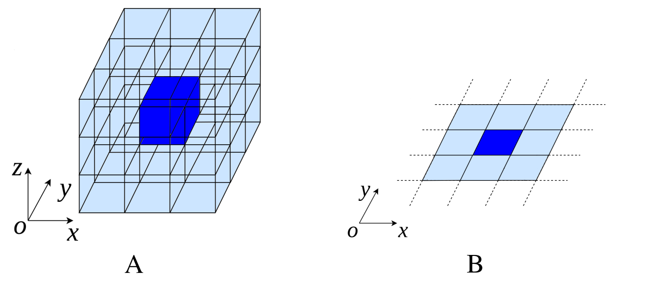


Fig. 3. These figures show the frontier area of a voxel (3D) and grid (2D). A for 3D and B for 2D, respectively. g is deep blue, while *Fr(g)* is light blue.

A. *Mathematical fundamental*

This section focuses on proving that the shortest path consists of tangents and only considers the situation of the connection between the start and target collide with obstacles. It is obvious that the shortest path is the connection if the connection doesn't collide with any obstacles. The distance between two points is calculated in the Euclidean distance, where A_i, B_i is the coordinate of point A, B and n is the dimension:

$$EuclideanDistance = \sqrt{\sum_{i=0}^{n-1}(A_i - B_i)^2} \tag{1}$$

Space V is a finite static discrete n-dimensional space, where $n = 2$ or 3. And denote the set that consists of all occupied grids as O and the set of all free grids as F.

$$O \cap F = \emptyset, O \cup F = V \tag{2}$$

Assume A, B are two grids, denote $l(A, B)$ as the set of all grids path through by the line AB. If $\exists(g) \in l(A, B), g \in O$, line AB collide with obstacles; otherwise, it does not.

The resulting path is noted as an ordered set of grids (waypoints) that all grids in it are free to pass and the line that connects adjacent elements in the path cannot pass through any occupied grid.

A globally shortest path is locally shortest [20] while a locally shortest path may not be the globally shortest path.

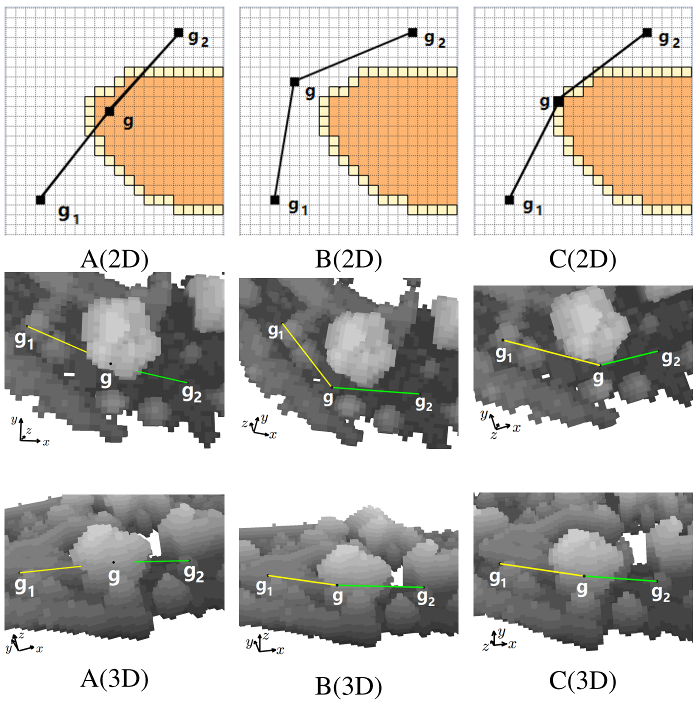


Fig. 4. These figures show examples (2D and 3D) of three possible states for a point g that is neither the start nor the target in the path. In 2D examples, the deep yellow area represents the obstacle, while the light yellow area represents the surface. In 3D examples (from two perspectives), the gray voxel represents the obstacle, while the light gray area represents the higher voxels, as same as following.

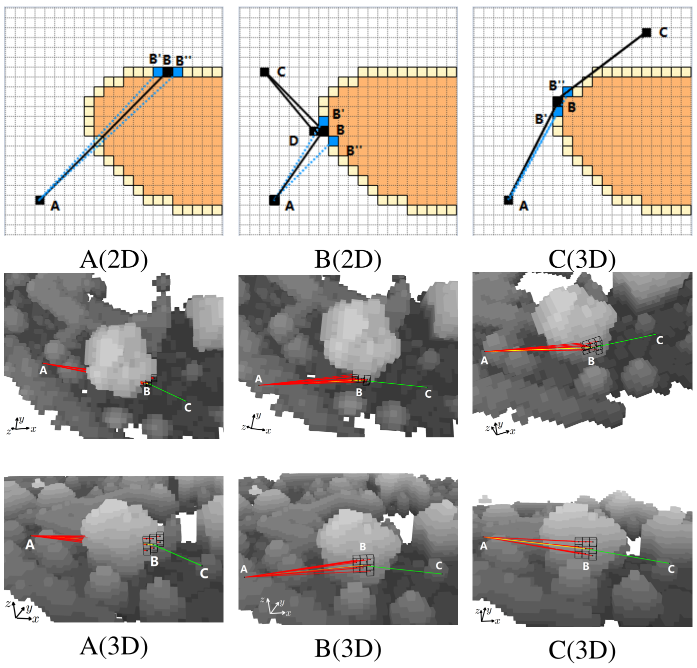


Fig. 5. These figures show examples (2D and 3D) of three possible states for a point B which belongs to a path and on the surface. In 2D examples, B' and B'' are $\{Fr(B) \cap S\}$. In 3D examples, the transparent voxels is $\{Fr(B) \cap S\}$. Fig A shows $\forall g \in \{Fr(B) \cap S\}$, $l(A, g)$ collide with obstacles locally, which contradicts the definition of a path. Fig B shows all $l(Ag)$ is not collide with obstacles locally, and it is feasible to construct a shorter path by replacing B with a point D in $\{Fr(B) \cap F\}$, enable $\left\|\overrightarrow{AD}\right\| + \left\|\overrightarrow{CD}\right\| < \left\|\overrightarrow{AB}\right\| + \left\|\overrightarrow{BC}\right\|$. Fig C shows a path that consists of tangents.

Definition 1. ***Neighborhood***: *$U(g)$ is defined as the neighborhood of grid g, with coordinate $(g_0, g_1, \cdots, g_{n-1})$. The neighborhood of a grid is defined as the nearest $2n$ grids to g, as shown in Fig. 2.*

$$U(g) = \underbrace{\{(g_0 \pm 1, g_1, \cdots, g_{n-1}), \cdots, (g_0, g_1, \cdots, g_{n-1} \pm 1)\}}_{n} \tag{3}$$

Definition 2. ***Frontier***:*$Fr(g)$ is defined as the frontier of grid g, with a coordinate of $(g_0, g_1, \cdots, g_{n-1})$. The frontier of a grid is defined as the nearest $3^n - 1$ grids to g, where $n = 2$ or 3, as shown in Fig. 3.*

$$\forall g' \in Fr(g), \forall i \in \{0, 1, \cdots, n-1\}, |g_i - g'_i| <= 1, g' \neq g \tag{4}$$

Definition 3. ***Surface***: *if a grid is free to pass but there is an occupied grid in its neighborhood, define it as a grid on the surface of obstacle. Denote the set consists of all grids on surfaces as S, if $g \in F, \exists g' \in U(g), g' \in O$, then $g \in S$.*

Definition 4. ***Locally shortest path***: *if it is impossible to get a shorter path by replacing its waypoint with any cell in its neighborhood, the path is locally shortest.*

Theorem 1. *Every grid of the locally shortest path must be on the surface, except the start and target. For $\forall g \in \hat{P}$, if $g! = g_{start}$ and $g! = g_{target}$.*

Proof 1. *There are two possible states if a grid g is not on the surface: 1, $g \in O$, as shown in Fig. 4A, and this contradicts the definition of a path and 2, $g \in (F - S)$. Assume that two adjacent grids of g in the locally shortest path are g_1 and g_2, as shown in Fig. 4B, $\exists g_3 \in U(g)$, $\|\overrightarrow{g_1g_3}\| + \|\overrightarrow{g_2g_3}\| < \|\overrightarrow{g_1g}\| + \|\overrightarrow{g_2g}\|$. This contradicts the definition of locally shortest path. Therefore, when g belongt to a locally shortest path, if $g! = g_{start}$ and $g! = g_{target}$, then $g \in S$, as shown in Fig. 4C.*

Definition 5. ***Line AB is collide with obstacles locally***: *if A is on the surface, then $l(A, B)$ is collide with $\{Fr(A) \cap O\}$ locally; and if B is on the surface, then $l(A, B)$ is collide with $\{Fr(B) \cap O\}$ locally.*

Definition 6. ***Tangent***: *1, there is at least one end of the tangent on the surface; 2, for a tangent AB, $A \in F, B \in F$, $l(A, B)$ is collide with no obstacles; and 3, for any grid on the surface (A or B or both), denote it as A, and denote the other as B. There must be $\exists g_1 \in Fr(A) \cap S$, $l(B, g_1)$ obstacle is collide with obstacle locally and $\exists g_2 \in Fr(A) \cap S$, $l(B, g_2)$ does not collide with obstacles.*

Theorem 2. *The locally shortest path consists of tangents: if a line in a path is not a tangent line, the path is not locally shortest. An example of a tangent is shown in Fig. 5C. The only exception is that both ends of the line are not on the surface, meaning $l(g_{start}, g_{target})$ is collide with no obstacles, which is easy to distinguish.*

Algorithm 1: Construction of the tangent graph

```
Input: $O$, $F$, $S$
Output: $G$
1 // $G$: the tangent graph
2 // $p_1, p_2$: grids that on surface
3 for $p_1 \in S$ do
4     for $p_2 \in (S - p_1)$ do
5         if $l(p_1, p_2)$ is a tangent && $l(p_1, p_2)$ collide with no obstacles then
6             add $l(p_1, p_2)$ to $G$;
7 return $G$;
```

Proof 2. *According to the definition of tangent and the waypoint of the shortest path must be on the surface, there are two possible situations for a line in the shortest path except for the tangent, assuming that the ends of the line are A and B, and B is on the surface:*

1, $\forall g \in \{Fr(B) \cap S\}$. $l(A, g)$ collide with obstacles locally; that means the $l(A, B)$ also collide with an obstacle, as shown in Fig. 5A, which contradicts the definition of the shortest path;

2, $\forall g \in \{Fr(B) \cap S\}$, $l(A, g)$ does not collide with any obstacles locally, and thus, it is feasible to construct a shorter path by replacing B with a grid in $\{Fr(B) \cap F\}$, as shown in Fig. 5B. This contradicts the definition of the locally shortest path.

So the locally shortest path consists of tangents.

As all segments in the globally shortest path are locally shortest [22], the globally shortest path consists of tangents. RimJump* cannot get the shortest path directly, but it selects the globally shortest path from all of the locally shortest paths. Related details will be described in the next section.

To accelerate the adaptive BFS, RimJump* uses the available range [34] which limits the range of select tangents.

Theorem 3. *Available range: For two adjacent tangents on the shortest path, the previous tangent limits the orientation of the current one. The angle between two tangents must be greater than or equal to 90 degrees, and the new tangent is closer to the obstacle than the previous one. As the available range limits the relationship between three continuous points in the shortest path, and three points determine a plane, this theorem can also be applied to a 3D environment.*

Proof 3. *This theorem has been proven in [34].*

In summary, the proof of the shortest path consists of tangents consists of two steps: 1, limit the range of the waypoint of the shortest path to the surface, and 2, prove that the connection between these points must be tangent. The implementation of how to construct the tangent graph and how to find the shortest collision-free path will be described below.

B. Algorithm and data structure

The method consist of two parts: 1, construction of the tangent graph (*offline*); 2, searching for the shortest path (*online*); and 3, generating an approximately optimal trajectory (*online*). Considering that the tangent graph is unchanged for a static map, we need to determine the tangent graph only once. The process of RimJump* is shown in Fig. 6.

Algorithm 2: Adaptive BFS for the shortest path

```
Input: $start$, $target$, $G$, $O$, $F$
Output: the shortest path (globally or locally is
        determined by how to branch cutting
        conditions)
1  // $G$: the tangent graph
2  $initPaths = \phi$;
3  for $p \in G$ do
4      if $l(start, p)$ collide with no obstacles and $l(start, p)$ is a tangent then
5          $initPaths \leftarrow l(start, p)$;
6  $paths = initPaths$;
7  while not all $path \in paths$ reach $target$ do
8      $newPaths = \phi$;
9      for $path \in paths$ do
10         $newPaths \leftarrow$ new paths created from $path$;
11     if any $path \in newPaths$ reach the target then
12         $maxLength$ = length of the shortest finished path in $paths$;
13     else
14         $maxLength = \infty$;
15     $paths$ = $newPaths$;
16 return $paths$;
```

1) Construction of the tangent graph: Generally, path planning is required to provide the shortest path that maintains a minimum distance from obstacles, which is called the **safe radius**, as well as our method. This section consists of three steps: 1, expand the boundary of obstacles to ensure safety, considering the safe radius; 2, detect surfaces according to the definition of a surface; and 3, determine the tangent graph according to the definition of a tangent, as shown in Algorithm 1. Since the definition of tangents is independent of the dimension of a map, both 2D and 3D tangent graphs can be constructed via the same algorithm.

2) Adaptive BFS for the shortest path: This section focuses on how to generate the shortest path from the tangent graph. A visibility graph or similar method uses Breadth First Search (BFS) to find the shortest path from the graph. The general BFS uses a queue to store current nodes while our method uses a path tree (proposed in our previous work [34]), which is an effective data structure to store information for the acceleration of searching, as shown in Fig. 7. The nodes in the path tree represent tangent points, and the edges represent tangent between nodes. Furthermore, it is possible for a parent node to have more than two child nodes in a path tree. When RimJump* finds all finished paths (in other words, all leaf nodes reach target), terminate searching and select the shortest one as the result. And this section only considers the geometrical requirement of the shortest path, and dynamics or mechanics are not considered, as well as on-ground path

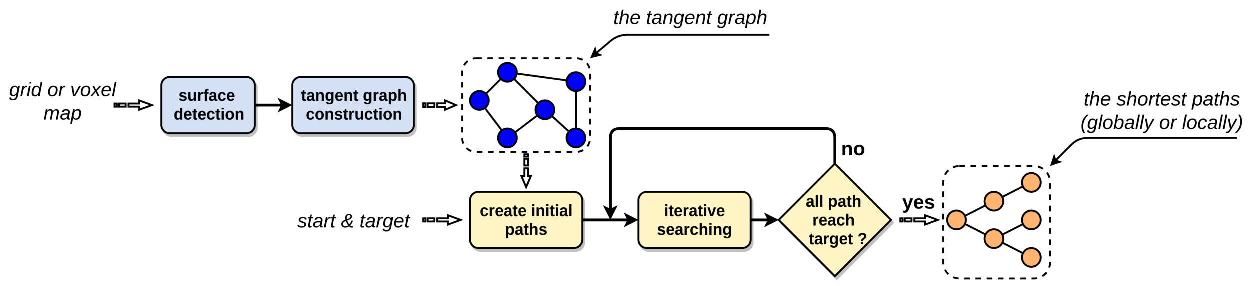


Fig. 6. This is the main process of RimJump*, and it consists of two sections: 1, construction of the tangent graph (offline, light blue) and 2, adaptive BFS searching. In other words, iterative search (online, light yellow).

planning.

To reduce the size of the path tree and accelerate the searching, three branch cutting conditions related to the geometric nature of the shortest path are introduced: 1, the available range [34] that limit the orientation of tangent during tangent graph searching; 2, the graph of the path length, if a node in the tangent graph is visited by a shorter path, then all paths created from the longer path are removed from the path tree, and any paths longer than the shortest finished path are removed; and 3, there are no loops in the shortest path, and a path containing the same node twice or more cannot be the shortest path. Therefore, if two non-adjacent nodes in the path are visible (in other words, their connection collides with no obstacles, there is a loop in the path), the path should be removed, except when the two nodes are the start and target.

Branch cutting conditions and path tree is the major difference between our search method and general Breadth First Search, related details can be found in Algorithm 2 and 3.

If we consider branch cutting conditions 1 and 2, the globally shortest path can be found, as shown in Fig. 9A. When condition 2 is replaced by condition 3, all locally shortest paths can be found. Generally speaking, the path tree obtained when searching the locally shortest paths is larger than that obtained when searching the globally shortest path. Notably, the path length graph condition can avoid loops, and thus the method searches the globally shortest path without requiring condition 3. Searching the globally shortest path in 3D will be demonstrated in the supplementary video. If we consider branch cutting conditions 1 and 3, all locally shortest paths can be found.

When it comes to on-ground path planning, the theorems that the turning point of the shortest path must be on the surface of an obstacle and the shortest path consists of tangents still hold. The proof is as follows: imagine that there is a virtual surface on the ground and the surface keeps a constant distance to the cell below. The objective of on-ground path planning can be transformed into a problem that finds the shortest path between the virtual surface and the ground. And the problem can be solved via the off-ground version of RimJump*.

So some tangent that beyond the slope of climbing or over the height of colliding with obstacles should be removed, and two extra branch cutting rules are introduced: 1, considering the maximum slope of climbing or downgrade of the vehicle, the angle between the plain ground and the tangent (in other words, the slope of a path) should be less than a threshold and 2, considering the maximum height of colliding with obstacles of the vehicle, the maximum distance between each voxel the tangent passes through and the ground below should be less than a threshold, as shown in Fig. 8. Considering the above constraints, some branch-cutting rules need to be removed, including 1 and 3. Therefore RimJump* (on-ground) cannot provide all locally shortest paths for the problem at this stage, but it can provide the globally shortest path under the two constraints of on-ground 3D path planning, as shown in Fig. 9B. In conclusion, the difference between RimJump* (off-ground) and RimJump (on-ground) is that the latter one uses different branch cutting rules.

Considering that the start and target are not always on the tangent graph, it is necessary to add the tangents of the start and target to the tangent graph before searching. This step involves the creations of the initial path. Except for the creation of the initial path, there is no online collision checking between nodes in the graph that have already been checked when constructing the graph, which reduces computational cost compares to path planning that contains online collision checking (e.g., Theta*, RRT).

C. Time and space complexity

This section focuses on the complexity of RimJump*, considering the difference between the off-ground and the on-ground version of RimJump* is cutting branch rules, they have the same time complexity and space complexity, as described below.

The time complexity of constructing the tangent graph, which is described in Algorithm 1, consists of two parts: 1, checking every surface point to another surface points, with $O(C_n^2) = O(n^2)$ time complexity and 2, collision checking and tangent checking for each pair of surface points, with $O(n)$ + $O(1)$ = $O(n)$ time complexity. Therefore, the total time complexity of constructing the tangent graph is $O(n^2) * O(n)$ = $O(n^3)$.

The time complexity of the adaptive BFS search consists of two components: 1, the doubled loop of BFS, described in Algorithm 2, with $O(n)$*($O(n)$ + $O(n)$) = $O(n^2)$ complexity and 2, creating new paths from the previous path, as described in Algorithm 3, determined by branch cutting conditions. The time complexity of checking whether there is a loop in a path

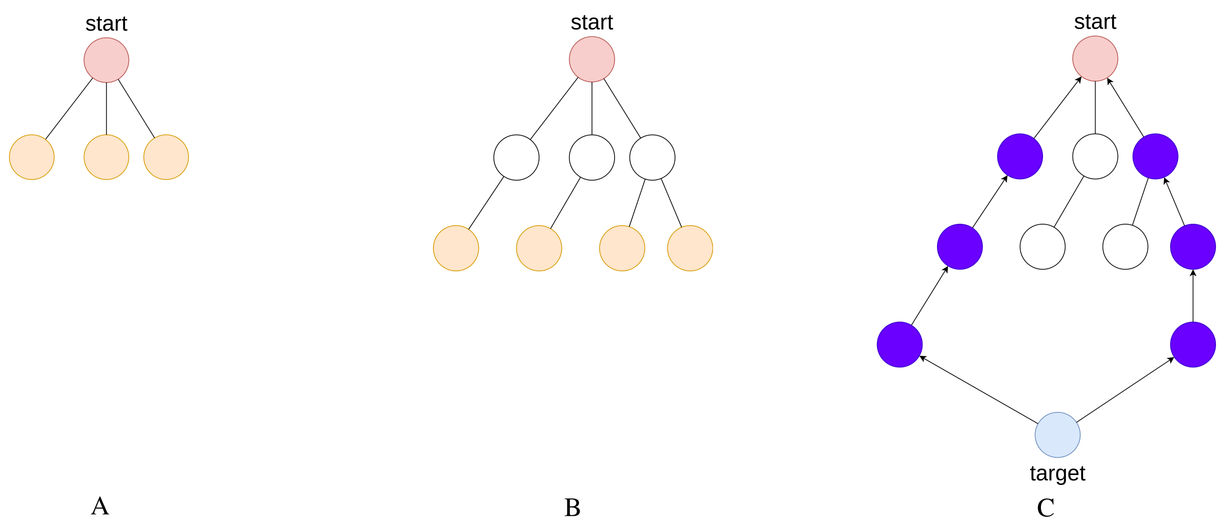


Fig. 7. These figures show examples of three status of the path tree during online searching: A, create initial paths; B, create new paths from previous path and C, all current paths reach the target. In these figures, yellow nodes represent nodes that are used to create new paths (in other words, all leaf nodes that do not reach the target) and deep blue nodes represent nodes that forms the finished path.

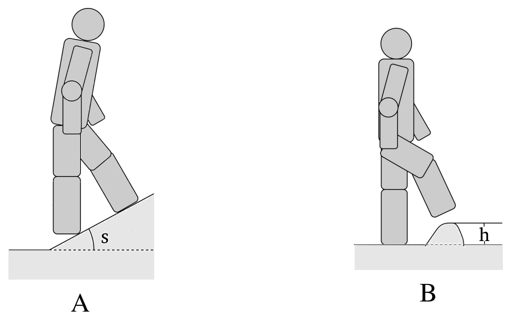


Fig. 8. These figures show the two constraints of on-ground path planning, A and B show the slope of climbing s and the height of colliding with obstacles h of the bipedal robot, respectively.

is $O(n)$, and the complexity of the two other branch cutting criteria, namely, the path length graph is $O(1)$ (the tangent graph's node record the shortest path that reaches it), and the available range is $O(1)$. Thus, searching for the locally shortest path has $O(n^4)$ time complexity, and globally shortest paths have $O(n^3)$ time complexity.

The space complexity has two main components, the tangent graph and the path tree. Assuming that there are n nodes in the tangent graph. The tangent graph consists of the nodes of tangents and edges between nodes, and thus, it has $O(n)$ + $O(C_n^2)$ = $O(n^2)$ space complexity.

The size of the path tree is determined by the number of nodes, but the number of nodes is different when searching for the globally and locally shortest paths. Due to the limitations of the path length graph, a node (equivalent to a tangent point) cannot appear in the path tree more than once when searching for the globally shortest path, and thus, the path tree has $O(n^2)$ space complexity when searching for the globally shortest path. In contrast, when searching for the locally shortest path, a node may appear twice or more, and thus, the path tree has $O(n^3)$ space complexity, as described in Algorithm 2 and 3.

In conclusion, the time complexity and the space complexity of RimJump* are shown In Table I.

Algorithm 3: Create new paths from the previous path

```
Input: $O, F, G, target, path$
Output: $newPaths$
1  // $path$: a previous path
2  // $newPaths$: new paths created from $path$
3  $newPaths = \phi$;
4  if $connection$ of the $target$ and the last point of $path$
     collide with no obstacles then
5      $newPath$ = $path \leftarrow target$;
6      $newPaths \leftarrow newPath$;
7      return $newPaths$;
8  $tangentCandidates$ = tangents in $G$ that one of the
     two end points is the last point of $path$;
9  for $tangent \in tangentCandidates$ do
10     if $tangent$ meet no conditions of cutting branch
         then
11         $newPath$ = $path \leftarrow tangent$;
12         $newPaths \leftarrow newPath$;
13 return $newPaths$;
```

TABLE I
RimJump*'s time and space complexity

index	Tangent graph construction	search globally shortest path	search locally shortest path
Time	$O(n^3)$	$O(n^3)$	$O(n^4)$
Space	$O(n^2)$	$O(n^2)$	$O(n^3)$

In comparison between RimJump* (searching globally shortest path) and other algorithms, we mainly focus on the time complexity of search the globally shortest path. The time complexity of an algorithm quantifies the amount of time as a function of the size of the input to the problem. But the raw time complexity is measured in different size of the input, as

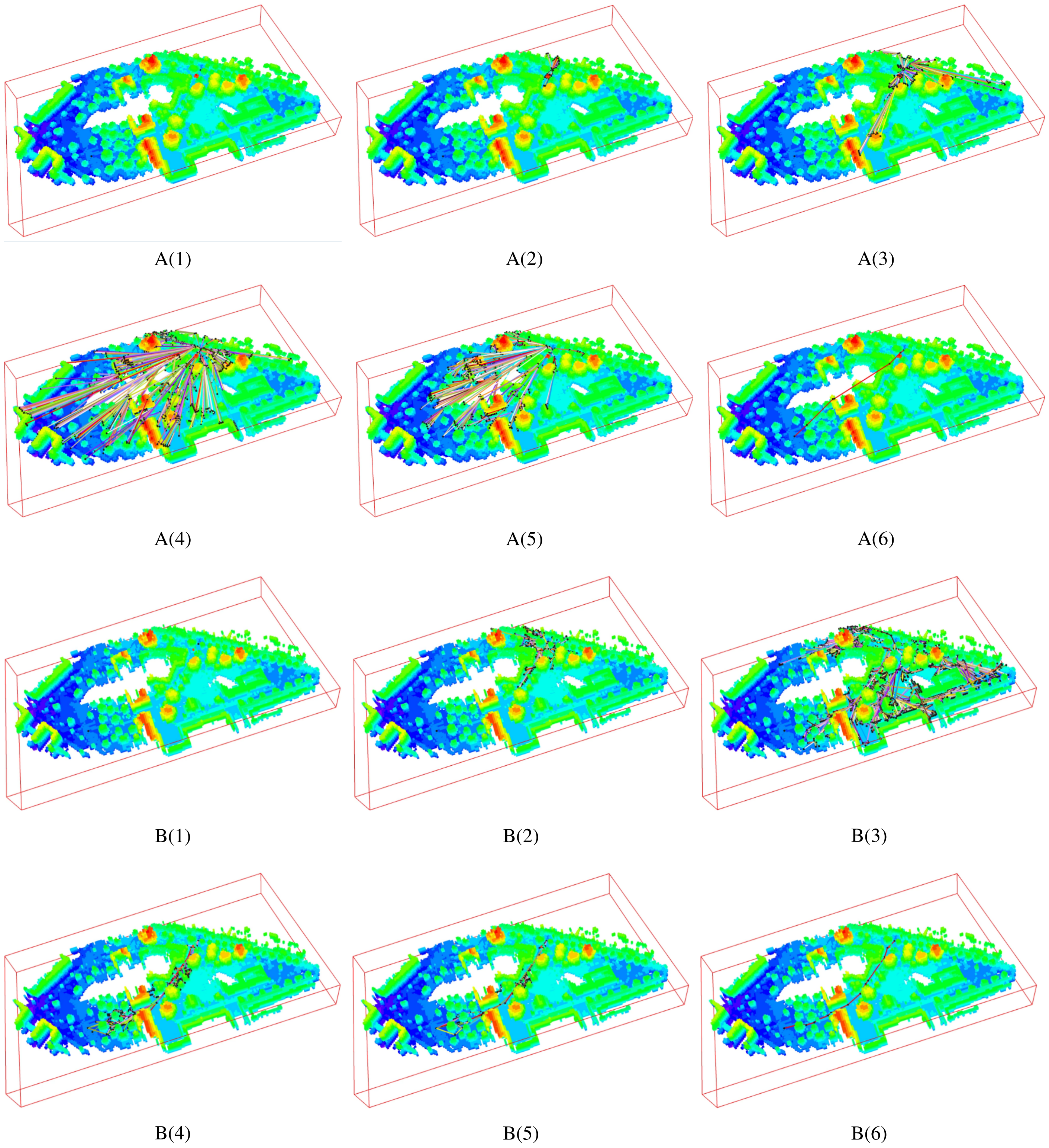


Fig. 9. These figures represent how RimJump* searches the off-ground shortest paths (A) and the on-ground shortest path (B) under a 3D map, in (1) $\rightarrow$ (6) order, and more red represents higher voxels while blue means lower voxels. They shared the same start and target, but generate different result paths. Related videos will be found in the supplementary material.

shown in Table II. So the size of input must increase at the same level to ensure a fair comparison. Denote the scale of the space is increased at speed s, so the volume is increased at s^3 and the area of the surface is increased at s^2. For node-based algorithms(e.g., Dijkstra, A*), sampling-based algorithms(e.g., RRT), and bio-inspired algorithms(e.g., genetic algorithm and ant colony optimization), their size of the input is determined by the volume of free space, so their size of the input is increased at s^3. RimJump*'s size of the input is determined by the size of the tangent graph which is determined by the area of the surface, so RimJump* and the visibility graph's size of input is increased at s^2. The complexity that under the same of the size of the input is shown in But the raw time complexity is measured in different size of the input, as shown in Table III. It is obvious that RimJump* has one of the least time complexity under the same size of the input.

TABLE II
TIME COMPLEXITY UNDER DIFFERENT SIZE OF INPUT

type	RimJump*	Dijkstra	A*	Theta*
complexity	$O(n^3)$	$O(n^2)$	$O(n^2 \log n)$	$O(n^3 \log n)$
type	RRT	Visibility Graph	Genetic Algorithm	Ant Colony Optimization
complexity	$O(n^2)$	$O(n^3)$	$> O(n^2)$	$O(n^3)$

TABLE III
TIME COMPLEXITY UNDER THE SAME SIZE OF INPUT

type	RimJump*	Dijkstra	A*	Theta*
complexity	$O(s^6)$	$O(s^6)$	$O(s^6 \log n)$	$O(O(s^9 \log s))$
type	RRT	Visibility Graph	Genetic Algorithm	Ant Colony Optimization
complexity	$O(s^6)$	$O(s^6)$	$> O(s^6)$	$O(s^9)$

III. RESULTS

The performance of RimJump* is evaluated by a comparison between RimJump* and Dijkstra, A*, Theta* and RRT. RimJump* can be applied to both 2D and 3D environments, but there is lot of work focus on path planning under the 2D environment: thus, only comparisons about 3D path planning (off-ground) are included in the results. The result of the on-ground 3D path planning of RimJump* will be demonstrated.

The experimental section mainly focuses on the following questions: 1, does RimJump* (off-ground path planning) find the shortest path under a cluttered 3D map; 2, what is the relationship between RimJump*'s time cost and distance between the start and target when compared to other algorithms.

All experiments were conducted on a computer with a 3.60 GHz Intel Xeon(R) W-2123 CPU and 15.3 GB RAM. We used the best available third-party implementations of our knowledge of the other algorithms to perform the comparison. Related videos can be found in the supplementary materials.

A. Comparison with other off-ground 3D path planning

In this section, we apply RimJump* and the other four algorithms to an octomap created from real 3D scans obtained from the dataset mentioned above, the map contains $184 \times 105 \times 19 = 3.67 \times 10^5$ voxels, and 2.62×10^4 voxels are on the surface. The input is 12 pairs of starts and targets, for which the shortest distance between them is increasing. As shown in Table IV, the index 1 is the shortest, while the index 12 is the longest, as shown in Fig. 11A. This setting aims to show how their time cost changes as the distance between the start and target increases. The octomap is shown in Fig. 10A, and an comparison example is shown Fig. 10B. The results of the comparison are shown in Fig. 11.

TABLE IV
TEN PAIRS OF COORDINATES (VOXELS) USED IN PATH PLANNING COMPARISON

index	start	target	index	start	target
1	$(151, 55, 8)$	$(177, 64, 7)$	7	$(56, 34, 12)$	$(177, 64, 7)$
2	$(104, 38, 10)$	$(147, 37, 7)$	8	$(23, 79, 10)$	$(177, 64, 7)$
3	$(85, 50, 12)$	$(147, 37, 7)$	9	$(14, 18, 8)$	$(177, 64, 7)$
4	$(73, 56, 12)$	$(147, 37, 7)$	10	$(7, 15, 7)$	$(180, 95, 7)$
5	$(72, 34, 15)$	$(177, 64, 7)$	11	$(9, 9, 10)$	$(182, 100, 7)$
6	$(69, 71, 11)$	$(177, 64, 7)$	12	$(2, 3, 8)$	$(182, 101, 5)$

As shown in Fig. 11A, both RimJump* and Theta* produce the same path lengths. Theta* has been proven to always find the shortest path in [25], so RimJump* can also find the shortest path. This proves the principle: "the shortest path between two points consists of tangents". Both Dijkstra and A* measured the length of the path in Manhattan distance, where A_i, B_i is the coordinate of point A, B:

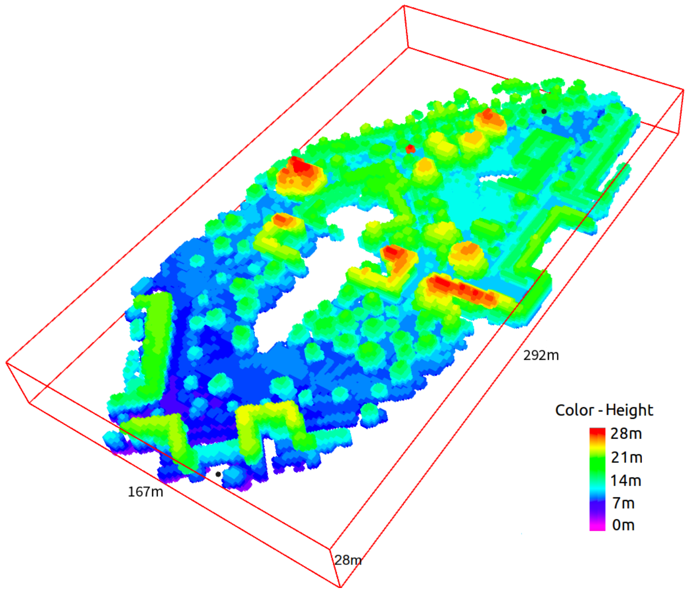
$$ManhattanDistance = \sum_{i=0}^{n-1} \|(A_i - B_i)\| \tag{5}$$

resulting in that they cannot find the shortest path in Euclidean distance. For RRT, due to the random sampling, it is the fastest method in comparison, and it can always find a feasible path but not the shortest one.

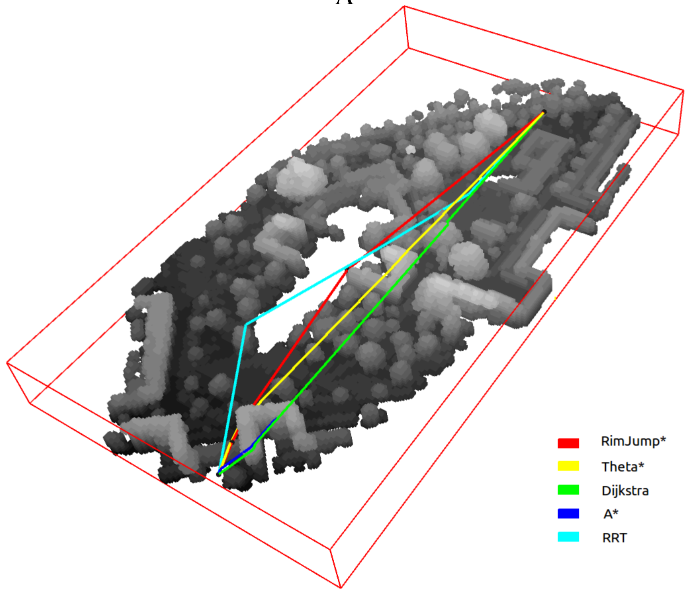
When it comes to the time cost aspect, as shown in Fig. 11B, Theta* and Dijkstra cost more time than other algorithms in most tests, and RRT costs the least amount of time. A* is a heuristic graph search method while Dijkstra is a Breadth-First-Search-based method that searches in all directions. Thus, A* generally costs less time than Dijkstra. A* costs only a little time in short distance path planning; however, its time cost increases dramatically as the distance between the start and target increases. Dijkstra and A*'s time cost keeps increasing until all free voxel is visited.

These phenomena resulted from the nature of the graph searching method; the more nodes, the more time cost. On one hand, as the distance increases, the number of free voxels increases heavily, which results in dramatic increases in time cost. On the other hand, the graph search method takes a few times if the number of free voxels is less.

As shown in Fig. 11B, the time cost of RimJump* is increasing slowly as the distance becomes larger. To explain

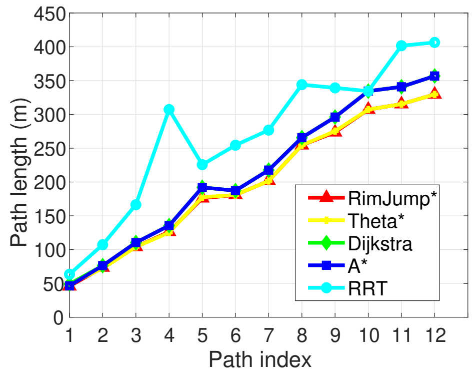


A

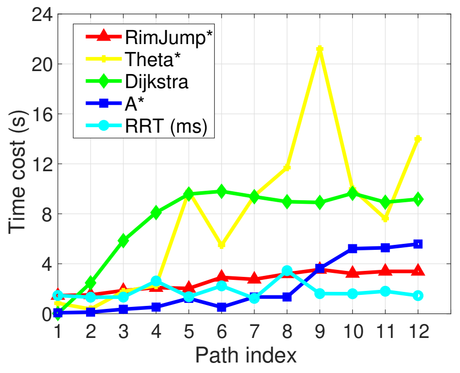


B

Fig. 10. A: The octomap used in the large-scale map experiment. More red represents higher voxels while blue means lower voxels. The octomap was acquired from the campus of the University of Freiburg, with the following size: 292 m x 167 m x 28 m. RimJump* generates a tangent graph with 2.3×10^4 nodes and 1.57×10^6 vertexes from the octomap. B: Path planning results of path planning methods. As the globally shortest path may not be unique, RimJump* and Theta* generate different path, but they have the same length.



A



B

Fig. 11. A shows the length of RimJump* and the other four algorithms under 12 pairs of starts and targets. B shows related time cost of path planning. It is noteworthy that the unit of time cost of RRT is milliseconds while the units of the other algorithms are seconds.

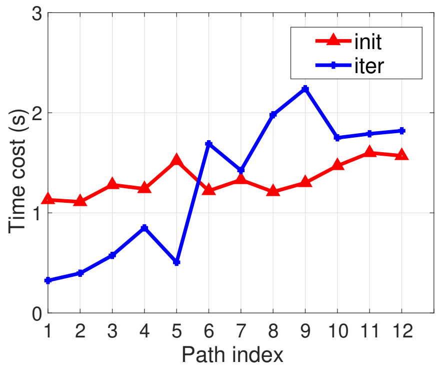
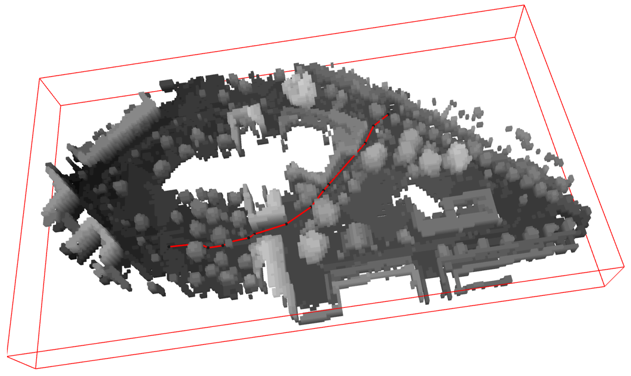


Fig. 12. This figure shows the time cost of creating the initial path (labeled in *init*) and the adaptive BFS (labeled in *iter*) of RimJump* when input the 12 pairs of starts and targets are input. Dijkstra's path length is the same as A*'s, while RimJump*'s path length is the same as Theta*'s.

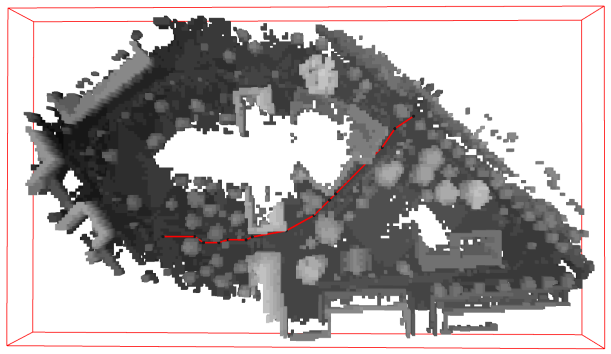
the phenomenon, we need to analyze the composition of the time cost of RimJump*. As mentioned in Algorithm 2, the time cost of RimJump* consists of two parts: 1, create the initial path from the tangent graph and 2, iterative searching until all paths in the path tree reach the target. The time cost of the two parts is listed in Fig. 12.

It is obvious that the time cost of generating the initial path is stable no matter the distance between the start and target. This is because when creating the initial path, RimJump* checks whether the connection line between the start and all nodes in the tangent graph is tangent and whether the connection line collides with obstacles. As the size of the tangent graph is unchanged, the time cost of creating the initial path is stable under the same map.

When it comes to the adaptive BFS, the tangent graph only contains tangent points, while the graph A* search contains all passable voxels between the start and target point. In other words, the size of the tangent graph is mainly influenced by the area of the surface, while the size of the graph that Dijkstra and A* search are mainly influenced by the volume of the space.



A

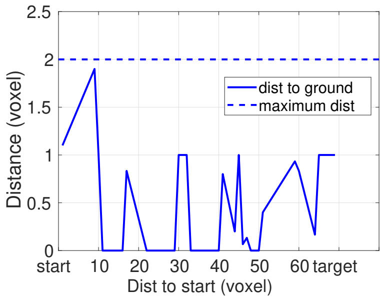


B

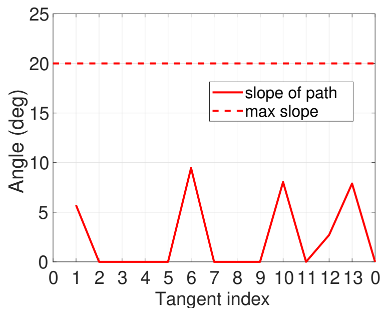
Fig. 13. This is the resultant path (the same path under two different perspectives) of the on-ground version of RimJump*. Related videos will be found in the supplementary material. The path meets the requirement that it cannot leave the ground too much. It takes 0.52s to create initial paths and 0.54s to search the shortest path.

Generally, if the scale of the space is increased at speed s, the area of the surface is increases at speed s^2 and the volume increases at speed s^3. Thus, eventually, RimJump* will be faster than graph search-based path planning as the scale of space keeps increasing. Considering that random sampling-based methods also search all passable space in the map of Dijkstra and A*, it is reasonable to presume that they run slower than RimJump* too if the scale of the map is big enough. Additionally, RimJump* reduces the time cost of BFS by considering the geometric nature (the three cutting branch rules) of the shortest path, and thus, the time cost increases slowly as the scale of the map increases.

In conclusion, RimJump* can always find the shortest path, and its time cost is mainly determined by the size of the tangent graph and is insensitive as the map scale increases, which means that the time cost is insensitive to the increased distance between the start and target increases and is mainly determined by the number of tangents between the start and target. Especially, RimJump* is suitable for path planning in large-scale 3D environments with sparse obstacles (e.g., downtown and mountains) which is not suitable for graph search-based methods. Since the mentioned environment contains lots of free voxels, the point-by-point traversal method will cost more time than environment that contains fewer free voxels. However, the tangent graph in the environment is relatively small, and thus, RimJump* is faster than the graph search method.



A



B

Fig. 14. Figure A shows the distance between the tangent and the below-ground changes, and Figure B shows the angle between the tangent of the path and the horizontal plane changes as the distance to the start increases. The distance to the ground is less than 2 voxels and the slope of the tangents of the path is under the limitation of 20 degrees.

B. *On-ground 3D path planning*

This section demonstrates RimJump*'s result of on-ground path planning. This section use the same map for comparison with other off-ground path planning use. Two coordinates on the ground are selected as the start $(57, 33, 8)$ and the target $(138, 72, 7)$, and the resulting paths are shown in Fig. 13. Considering the constraint of on-ground path planning, the height that the path colliding with obstacles was set to 2 voxels, and the maximum slope of climbing or downgrade was set to 20 degrees. Then RimJump* (on-ground) generates a tangent graph that contains 7897 nodes and 1.04×10^5 vertexes, which is less than the off-ground version. How the distance between the path and the below-ground changes and the slope of the tangents of the path (RimJump*'s resultant path is consists of tangents) are shown in Fig. 14. As shown in these figures, the path meets the two constraints of on-ground path planning.

IV. Conclusion and Future Works

Based on the principle that the shortest collision-free path consists of tangents, RimJump* was proposed as a shortest path planning in discrete space (e.g., octomap) that provides a path for the robot to follow. The tangent-based approach can be applied to 2D and 3D environments (off-ground and on-ground). The off-ground and on-ground versions of RimJump* are based on the same mathematical fundamental; the only difference is that the on-ground version uses extra branch cutting rules to limit the resulting path to ground. To accelerate the adaptive BFS, three branch cutting conditions were applied to the adaptive BFS. These branch cutting conditions substantially reduce the time cost of searching the shortest path.

There are three major advantages of RimJump*: 1, it avoids almost all online collision checking, which costs lots of time under large-scale maps. Since the area of the surface increases slower than the volume as the scale increases, RimJump* has low complexity than other methods that search the whole passable space; 2, it uses an adaptive Breadth-First Search which reduces the time cost of searching the graph; 3, it determines the shortest path (both on-ground and off-ground) in Euclidean distance in discrete space. As shown in the Results, RimJump* and Theta* find the path with the same length under various tests, which means the definition of tangent meets our expectations.

RimJump* suffers from several disadvantages; it takes considerable time (tens of minutes or several hours) to construct the tangent graph and the size of the graph is influenced by the degree of fragmentation of obstacle surfaces. Since the 3D tangent graph contains many nodes, finding all locally shortest paths in a 3D scenario is time-consuming. Since RimJump* is a tangent-based method without online point-by-point traversal, it cannot be applied to maps with various cost values. For RimJump*, a grid (or voxel) has only two states, free to pass or impassable (occupied). There remains room for improvement in the current implementation of RimJump*, and we will open the source code to enable anyone the opportunity to improve it.

Considering that the definition of the tangent is independent of the dimension, exploring tangent-based path planning under higher-dimensional (≥ 4) space is an interesting question. The current definition of the tangent can be expanded to high-dimensional space, except for the available range condition of branch cutting, which includes the definition of a plane surface under high-dimensional space.

A potential application of RimJump* (on-ground) is the navigation of bipedal or quadruped robots equipped with visual sensors or 3D lidar. The global path planning is implemented by RimJump*, to provide the shortest path that under the two constraints of on-ground path planning for the robot to follow. An example of path planning for a robot that can not leave the ground is shown in Fig. 13. Assuming that the robot already has a map of the nearby environment, it can walk on an uneven surface via 3D RimJump*.

Another potential application of RimJump* (off-ground) is commercial package delivery systems consisting of drones (e.g, PrimeAir of Amazon and Wing of Google). The global path planning of drones, which needs a large-scale map of a city and substantial calculation, can be implemented by RimJump* (off-ground) in the cloud. When a drone is required to fly to a new position, RimJump* determines the shortest path in a short time. A server on the cloud may provide navigation service for thousands of drones.

Acknowledgment

We thank Armin Hornung for the octomap used in 3D comparison and Dirk Haehnel for the map used in 2D trajectory generation. These datasets played an important role in our experiment. We also thank our colleagues for their inspiring questions and constructive opinions, especially Zhe Xu, Xiaozhu Ju, and Yijie Guo. The first author thanks Lu Zhu for her encouragement and support during the consummation of RimJump*.

References

[1] Chengren Yuan A et al. “An efficient RRT cache method in dynamic environments for path planning”. In: *Robotics and Autonomous Systems* (2020).

[2] R. A. Saeed A, Diego Reforgiato Recupero A, and Paolo Remagnino B. “A Boundary Node Method for path planning of mobile robots”. In: *Robotics and Autonomous Systems* 123 (2019).

[3] Nancy M Amato et al. “OBPRM: An obstacle-based PRM for 3D workspaces”. In: *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*. 1998, pp. 155–168.

[4] Mojtaba Nouri Bygi and Mohammad Ghodsi. “3D visibility graph”. In: *Computational Science and its Applications, Kuala Lampur* (2007).

[5] Yang Chen et al. “Relative coordination 3D trajectory generation based on the trimmed ACO”. In: *2010 International Conference on Electrical and Control Engineering*. IEEE. 2010, pp. 1531–1536.

[6] Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. “RRT*-AR: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3947–3952.

[7] E. W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numer. Math.* 1.1 (1959), pp. 269–271.

[8] Haibin Duan et al. “Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm”. In: *Simulation Modelling Practice and Theory* 18.8 (2010), pp. 1104–1115.

[9] Cyril Fonlupt, Philippe Preux, and Denis Robilliard. “Preventing premature convergence via cooperating genetic algorithms”. In: *Proc. of the Workshop on Foundations of Genetic Algorithms*. Citeseer. 1993.

[10] Bing Fu et al. “An improved A* algorithm for the industrial robot path planning with high success rate and short length”. In: *Robotics and Autonomous Systems* 106 (2018).

[11] Abdurrahman Hacıogĝlu et al. “Transonic airfoil design and optimisation by using vibrational genetic algorithm”. In: *Aircraft Engineering and Aerospace Technology* (2003).

[12] Abdurrahman Hacioĝlu and İbrahim Özkol. “Vibrational genetic algorithm as a new concept in airfoil design”. In: *Aircraft Engineering and Aerospace Technology* 74.3 (2002), pp. 228–236.

[13] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE Trans. Syst. Sci. Cybern.* 4.2 (1968), pp. 100–107.

[14] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[15] Lucas Janson, Brian Ichter, and Marco Pavone. "Deterministic sampling-based motion planning: Optimality, complexity, and performance". In: *The International Journal of Robotics Research* 37.1 (2018), pp. 46–61. DOI: 10.1177/0278364917714338. eprint: https://doi.org/10.1177/0278364917714338. URL: https://doi.org/10.1177/0278364917714338.

[16] Kaichun Jiang, LS Seneviratne, and SWE Earles. "Finding the 3D shortest path with visibility graph and minimum potential energy". In: *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*. Vol. 1. IEEE. 1993, pp. 679–684.

[17] Lydia Kavraki and J-C Latombe. "Randomized preprocessing of configuration for fast path planning". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE. 1994, pp. 2138–2145.

[18] Lydia E Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE. Trans. Rob. Autom.* 12.4 (1996), pp. 566–580.

[19] Steven M LaValle. *Rapidly-exploring random trees: A new tool for path planning*. Citeseer, 1998.

[20] Y-H Liu and Suguru Arimoto. "Proposal of tangent graph and extended tangent graph for path planning of mobile robots". In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. IEEE. 1991, pp. 312–317.

[21] Yun-Hui Liu and Suguru Arimoto. "Computation of the tangent graph of polygonal obstacles by moving-line processing". In: *IEEE Transactions on Robotics and Automation* 10.6 (1994), pp. 823–830.

[22] Yun-Hui Liu and Suguru Arimoto. "Path planning using a tangent graph for mobile robots among polygonal and curved obstacles: Communication". In: *The International Journal of Robotics Research* 11.4 (1992), pp. 376–382.

[23] Ellips Masehian and MR Amin-Naseri. "A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning". In: *Journal of Robotic Systems* 21.6 (2004), pp. 275–300.

[24] Seyedali Mirjalili, Jin Song Dong, and Andrew Lewis. "Ant colony optimizer: Theory, literature review, and application in AUV path planning". English. In: *Studies in Computational Intelligence*. Studies in Computational Intelligence. Springer Verlag, Jan. 2020, pp. 7–21. DOI: 10.1007/978-3-030-12127-3_2.

[25] Alex Nash et al. "Theta*: Any-angle path planning on grids". In: *AAAI*. Vol. 7. 2007, pp. 1177–1183.

[26] Chong Pan et al. "Path Planning of Mobile Robot Based on an Improved Ant Colony Algorithm". In: *Convergent Cognitive Information Technologies*. Ed. by Vladimir Sukhomlin and Elena Zubareva. Cham: Springer International Publishing, 2020, pp. 132–141. ISBN: 978-3-030-37436-5.

[27] Y Volkan Pehlivanoglu. "A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV". In: *Aerospace Science and Technology* 16.1 (2012), pp. 47–55.

[28] Ahmed Yousuf Saber and Abdulaziz Mohammed Alshareef. "Scalable unit commitment by memory-bounded ant colony optimization with A local search". In: *International Journal of Electrical Power & Energy Systems* 30.6-7 (2008), pp. 403–414.

[29] Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard. "Configuration space and visibility graph generation from geometric workspaces for uavs". In: *AIAA Guidance, Navigation, and Control Conference*. 2011, p. 6416.

[30] Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard. "Generating approximative minimum length paths in 3D for UAVs". In: *2012 IEEE Intelligent Vehicles Symposium*. IEEE. 2012, pp. 229–233.

[31] John Schulman et al. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization." In: *Robotics: science and systems*. Vol. 9. 1. Citeseer. 2013, pp. 1–10.

[32] Glenn Wagner, Minsu Kang, and Howie Choset. "Probabilistic path planning for multiple robots with subdimensional expansion". In: (2012), pp. 2886–2892.

[33] Fei Yan, Yi-Sha Liu, and Ji-Zhong Xiao. "Path planning in complex 3D environments using a probabilistic roadmap method". In: *International Journal of Automation and computing* 10.6 (2013), pp. 525–533.

[34] Zhuo Yao et al. "ReinforcedRimJump: Tangent-based shortest-path planning for two-dimensional maps". In: *IEEE Transactions on Industrial Informatics* (2019).

[35] Zhuo Yao et al. "RimJump: Edge-based Shortest Path Planning for a 2D Map". In: *Robotica* 37.4 (2019), pp. 641–655.

[36] Anna Yershova et al. "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain". In: 2005.

[37] Xunyu Zhong et al. "Hybrid Path Planning Based on Safe A* Algorithm and Adaptive Window Approach for Mobile Robot in Large-Scale Dynamic Environment". In: *Journal of Intelligent and Robotic Systems* (2020), pp. 1–13.