*remote sensing*

MDPI

*Article*

# FPS: Fast Path Planner Algorithm Based on Sparse Visibility Graph and Bidirectional Breadth-First Search

**Qunzhao Li [1], Fei Xie [1,*], Jing Zhao [2], Bing Xu [3], Jiquan Yang [1], Xixiang Liu [4] and Hongbo Suo [5]**

[1] School of Electrical and Automation Engineering, Nanjing Normal University, Nanjing 210023, China; 201846063@njnu.edu.cn (Q.L.); 63047@njnu.edu.cn (J.Y.)
[2] College of Automation & College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing 210023, China; zhaojing@njupt.edu.cn
[3] Department of Aeronautical and Aviation Engineering, The Hong Kong Polytechnic University, Hong Kong, China; pbing.xu@polyu.edu.hk
[4] College of Instrument Science & Engineering, Southeast University, Nanjing 210096, China; 101010902@seu.edu.cn
[5] Nanjing Zhongke Raycham Laser Technology Co., Ltd., Nanjing 210023, China; suohongbo@raycham.com
* Correspondence: xiefei@njnu.edu.cn

**Abstract:** The majority of planning algorithms used are based on the occupancy grid maps, but in complicated situations, the occupancy grid maps have a significant search overhead. This paper proposed a path planner based on the visibility graph (v-graph) for the mobile robot that uses sparse methods to speed up and simplify the construction of the v-graph. Firstly, the complementary grid framework is designed to reduce graph updating iteration costs during the data collection process in each data frame. Secondly, a filter approach based on the edge length and the number of vertices of the obstacle contour is proposed to reduce redundant nodes and edges in the v-graph. Thirdly, a bidirectional breadth-first search is combined into the path searching process in the proposed fast path planner algorithm in order to reduce the waste of exploring space. Finally, the simulation results indicate that the proposed sparse v-graph planner can significantly improve the efficiency of building the v-graph and reduce the time of path search. In highly convoluted unknown or partially known environments, our method is 40% faster than the FAR Planner and produces paths 25% shorter than it. Moreover, the physical experiment shows that the proposed path planner is faster than the FAR Planner in both the v-graph update process and laser process. The method proposed in this paper performs faster when seeking paths than the conventional method based on the occupancy grid.

**Keywords:** visibility graph; computational geometry; path planning; mapping



**Citation:** Li, Q.; Xie, F.; Zhao, J.; Xu, B.; Yang, J.; Liu, X.; Suo, H. FPS: Fast Path Planner Algorithm Based on Sparse Visibility Graph and Bidirectional Breadth-First Search. *Remote Sens.* **2022**, *14*, 3720. https://doi.org/10.3390/rs14153720

Academic Editor: Andrzej Stateczny

Received: 16 June 2022
Accepted: 31 July 2022
Published: 3 August 2022



## 1. Introduction

With the popularity of the robotics industry, simultaneous localization and mapping (SLAM) technology has developed rapidly. SLAM technology can be divided into three categories, i.e., LiDAR-SLAM [1–4], visual-SLAM [5–8] and LiDAR fusion visual SLAM [9–11], and it is widely used for robot navigation tasks. In the application of robot navigation, a by-product of SLAM is the map, including metric and topological maps. The metric map emphasizes accurately representing the positional relationships of objects, while the topological map emphasizes the relationships between map elements.

In smaller spaces, such as corridors and houses, occupancy grid maps [12] are preferred over topological maps. The topology maps, on the other hand, are more appropriate for path planning in large areas where the occupancy grid maps are computationally expensive. In this paper, the visibility graph [13], a topology-based type of map, will be constructed for route planning and navigation.

Path planning has been an emerging trend in research nowadays to cater to the needs of autonomous systems. The visibility graph (v-graph) is an efficient map representation

for path planning, which allows the robot to move from one node to another, but it has some drawbacks. Firstly, it is not only hard to map the visibility graph completely in the 3D world [14], but it is also difficult to extract the outlines of obstacles. Secondly, when the number of nodes in the graph increases, the associated edges will be doubled, which causes increased computational cost in terrain where complex obstacles exist [15]. In this case, it is necessary to simplify complex obstacles to reduce vertices and edges for path planning.

Most existing v-graph generation methods [16–21] store the pointcloud explored by the LiDAR into the local grid and then perform plane mapping to extract the vertices of the polygon. However, these algorithms, such as [17–20], face the following problems. Firstly, if the local grid is too sparse, the sampling accuracy will be decreased. On the other hand, if the local grid is dense, the shape of a polygon will be more accurately determined, but the amount of calculation will increase dramatically. Secondly, the quantity of vertices and edges affects how complicated the v-graph-based path search algorithm is. There will be a lot of redundant vertices and edges in maps with a lot of intricate barriers, which makes path search a time-consuming task. Finally, the large number of vertices and edges in the v-graph slows down its traversal speed, which leads to a decrease in the maintenance speed of the v-graph. Although WonheeLee et al. [21] proposed a v-graph-based obstacle avoidance strategy, they did not address the issue of dense v-graph in complicated scenes. A sparse v-graph-based path planner is proposed as a solution to these issues, which lowers the cost of v-graph maintenance, increases the effectiveness of v-graph building, and decreases space waste during the path search. Overall, the main contributions of the paper are summarized as follows:

- Compared to existing methods for storing a laser pointcloud, this paper proposes a complementary holed structure for iteratively updating the local grid. Basically, only half of the pointcloud data needs to be processed in each data frame to update the map. The pointcloud data are subjected to image blurring after planar mapping, and then key vertices are extracted from the blurred image.
- For obstacles with complex contours, this paper proposes a filtering method based on the edge length and the number of vertices of the obstacle contour. The method effectively reduces the number of vertices in the v-graph and the maintenance cost of the v-graph by performing vertex filtering on large complex obstacles. Since the v-graph and the path search algorithm are tightly coupled, the efficiency of the path search algorithm will also be improved.
- A bidirectional breadth-first search algorithm was introduced since exploring uncharted territory requires a lot of search space. In this paper, the edge between the goal point and the existing vertices in the v-graph is established by geometric checking. Therefore, the bidirectional breadth-first search algorithm could reduce the waste of exploration space in navigation.

## 2. Related Work

The current mainstream of path planning research is divided into the following categories: search-based, sampling (probability)-based, genetic algorithm (GA)-based, and learning-based. According to the planning results, it is further divided into complete planning algorithms and probabilistic complete planning algorithms.

Search-based planning methods: These methods mainly include Dijkstra [22] and its variants, such as A* [23], D* [24], etc. The Dijkstra and A* algorithms are often used to search on discrete grids. Such algorithms are re-initialized for each search cycle, thus taking a long time to plan routes. An incremental version of the Dijkstra-derived algorithm was proposed to reduce re-planning time by adjusting the local information to the planning result in the previous cycle. However, similar to D* Lite [25], when encountering complex environments, the computational load of the incremental algorithm to re-evaluate the current environment is even greater than that of A* without increment. Many improved A* algorithms have been proposed to decrease the memory space and achieve a better

trajectory [26,27], but they are all based on the occupancy grids and thus cannot avoid the shortcomings of the occupancy grids.

Random sampling planning methods: These methods mainly include rapidly expanding random tree (RRT)[28] and a series of its variants, such as RRT* [29], informed RRT* [30], and RRT-Connect [31]. These methods are designed for a known environment. Some algorithms derived from RRT are used for planning in an unknown or partially known environment. These methods must perform maintenance or regeneration of random trees frequently to account for newly observed environments.

Genetic algorithm methods (GA): This algorithm performs path planning through crossover and mutation of chromosomes. Tu et al. [32] proposed a method for generating collision-free paths based on GA. Chen et al. [33] proposed an improved GA to minimize the total distance of the UAV. The method in [34] combined deep learning and GA to design a followable path for multi-robots. Moreover, there are many different improvements to GA [35–37] to achieve better results. In the genetic algorithm, the choice of parameters such as crossover rate and mutation rate has a significant impact on the quality of the solution, but these values are typically chosen based on experience.

Planning by deep learning: These methods [38–40] require a large number of ground-truth labels for training. For example, the essence of training based on deep learning is to encode and decode environmental information. During testing, these methods can handle scenarios similar to the training environment; the essence of training based on reinforcement learning is to encode and map the information of the state of motion every time. During testing, the results are output according to the encoded motion state information. GuichaoLin et al. [41] proposed a collision-free model based on deep reinforcement learning to allow robots to avoid obstacles. Tutsoy et al. [42] provided a minimum time path for a balancing task through reinforcement learning, and [43] considered the energy consumption to design a general dynamic control model based on deep reinforcement learning. However, learning-based methods are inherently data-driven, and the magnitude of the data limits their ability to scale to different environments.

The majority of the aforementioned path planning techniques, including common search-based algorithms, RRT algorithms, and evolutionary algorithms, are based on occupancy grids. When the scope of the searched scene is large, the drawbacks of occupancy grids will constrain the speed of these algorithms, i.e., as shown in Figure 1, in a $10 \times 10$ grid map, the algorithm based on the occupancy grid map traversed 20 grids to reach the goal, while the algorithm based on the v-graph only traversed five vertices of the obstacle. This paper mainly studies the visibility graph-based and path-planning methods. Although the literature [13,44–46] studied the use of the visibility graph for robot navigation, and FAR Planner [19] applied the theoretical part to the exploration and navigation of the actual 3D world, the maintenance of a graph in complex situations is still expensive, and the way-point generated in unknown environments is easy to detour. In order to address these issues, a sparse v-graph path planner is proposed. This path planner enhances the efficiency of v-graph building while decreasing space waste and v-graph maintenance costs.

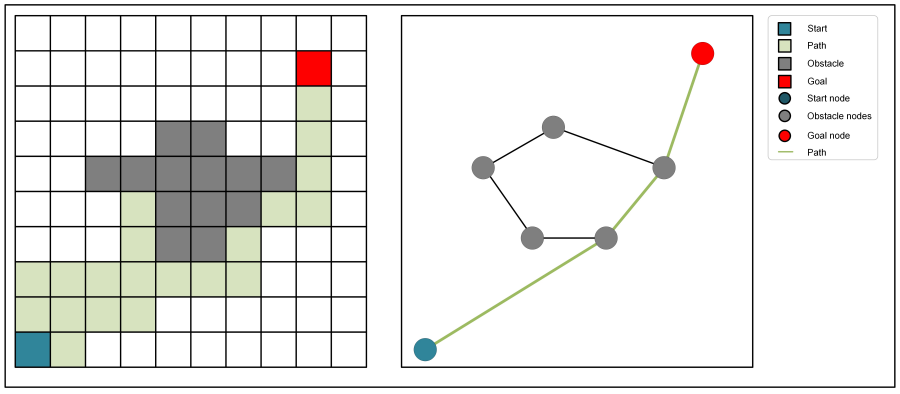



**Figure 1.** Comparison of the occupancy grids and the visibility graph.

Similar to FAR Planner [19], the pointcloud is extracted from obstacles and mapped into polygons, from which vertices and edges are extracted to construct the v-graph for navigation. The improvement of our approach is that each data frame of pointcloud in the local area does not fully participate in the construction of the global layer. The complementary hole structure for iteratively updating the local grid is used to store the pointcloud information in the current local area, which means that only half of the points in each data frame need to be processed each time. The method continuously updates the pointcloud information on each data frame until a global map is formed. Compared with the original algorithm of FAR Planner, the proposed algorithm can reach the target point within a shorter distance and take less time.

In simulation experiments, the feasibility of the method is evaluated through the simulated physical environment. The environment of the simulation experiment includes medium-scale, complex-scale, and large-scale environments in the Autonomous Exploration Development Environment provided by CMU [47], and medium-scale indoor environments and complex large-scale indoor environments provided by Matterport3D [48]. In the physical experiment, the LiDAR, and an Inertial Measurement Unit (IMU) are coupled to generate state estimation of the mobile robot [4], and the proposed replacednavigationnavgiation algorithm will be tested in a real garage.

## 3. Sparse Visibility Graph-Based Path Planner

Define $\mathcal{Q} \subset \mathbb{R}^3$ as the robot navigation space, and $\mathcal{S} \subset \mathcal{Q}$ as the sensor data from obstacles. A down-sample strategy is used to update and maintain the v-graph, denoted as $\mathcal{G}$, and the grid to store pointcloud is denoted as $\mathcal{L}$. Define the position of robot as $P_{robot} \in \mathcal{Q}$, the goal $P_{goal} \in \mathcal{Q}$.

The flow chart of the path planner proposed in the paper is shown in Figure 2. And the process consists of three parts: (1) generating the geometric contours of obstacles by LiDAR-to-plane mapping; (2) aggregating and simplifying complex obstacle information to maintain the v-graph at a low cost; and (3) searching for nodes and edges to generate the path from the start point to the goal through the v-graph.

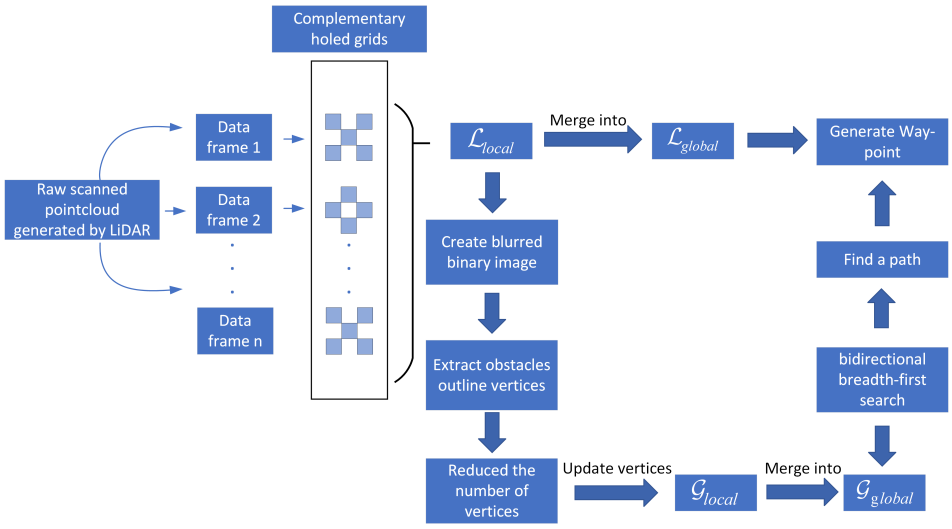



**Figure 2.** The main flow chart of the path planner based on the v-graph.

### *3.1. Pointcloud Extraction Structure*

We denote the process of extracting and mapping the pointcloud to geometric contours as extract $\left\{ P^k_{cloud} \subset \mathcal{Q} \,|k \in Z^+ \right\}$, and the grid as *Grid*, respectively. In most laser-based SLAM, grids are used for accessibility analysis, which means that pointcloud information needs to be recorded in the global layer $\mathcal{L}_{global}$ and local $\mathcal{L}_{local}$. Although an incremental method of updating the pointcloud is proposed, in the case of complex terrain and high-

resolution grids, the computational resources used to update the pointcloud are still very high. Therefore, a general sparsification module denoted as $\mathcal{F}$ is used to create the holed-structure local grid and incrementally update the pointcloud.

The dilated convolutional module [49] is usually used in neural networks to enlarge the receptive field in the picture, and its whole structure gives another way to deal with the pointcloud in the local grid: as shown in Figure 3, the pointcloud will be stored in the complementary hole grid.

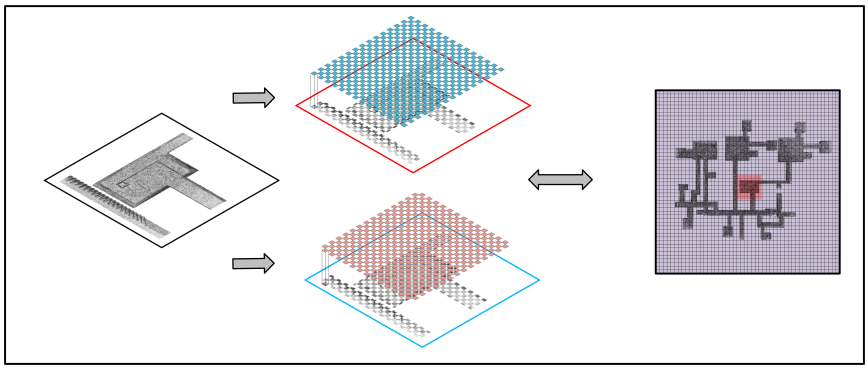


**Figure 3.** A schematic diagram of the holed grid structure used for the local update. The pointcloud is updated through the complementary holed grids between every two data frames and merged into the global layer.

The holed-structure local grid is defined as $Grid_d$, and the $\mathcal{F}$ contains $Sub \subset Grid_d$; when obstacles are detected by the LiDAR, the sensor data $\mathcal{S}$ is transferred to $Sub$, all $Sub$ forms the $Grid_d$. We denote the voxel size as $V_S$, and this value will affect the density of pointcloud. In this paper, the value of $V_S$ is set to 0.15 m. When the $Grid_d$ is formed, a PCL filter with a kernel size of ($V_S$, $V_S$, $V_S$) will be applied to reduce the size of the pointcloud. $\mathcal{S}^{'}$ is the remaining pointcloud in the $Grid_d$. After the local pointcloud is formed, the $\mathcal{S}^{'}$ are classified as obstacles or free, and we denote the classified $\mathcal{S}^{'}$ as $\left\{P^k_{cloud}\right\}$. At this step, the fully classified $Grid_d$ is denoted as $\mathcal{L}_{local}$, then integrates $\mathcal{L}_{local}$ to $\mathcal{L}_{global}$. The complementary hole grids will be generated respectively in different data frames, so that the final $\mathcal{L}_{global}$ still contains all the information of the pointcloud. The module $\mathcal{F}$ is shown in Algorithm 1.

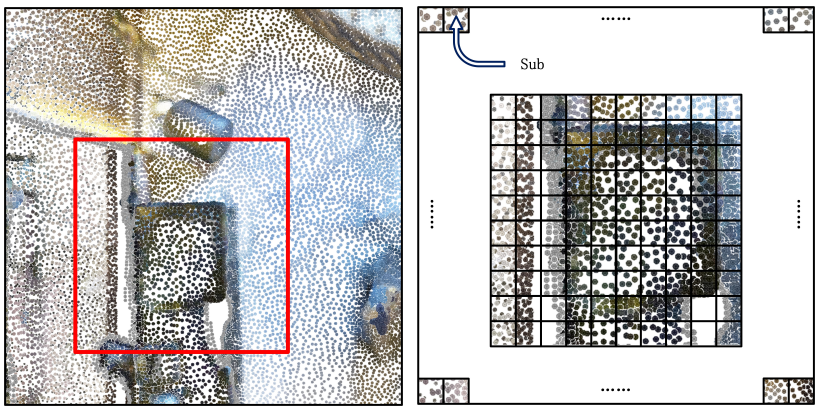
**Algorithm 1** Module $\mathcal{F}$.

**Input:** Sensor data $\mathcal{S}$
**Output:** $\mathcal{S}^{'}$

```
input S
for every data frame do
    Generate Grid_d
    for each Sub ⊂ Grid_d do
        store pointcloud ∈ S
    end for
end for
Apply PCL filter to point ∈ Grid_d
classify point ∈ Grid_d
S' = remain pointcloud in Grid_d
Update Grid_d to L_local
```

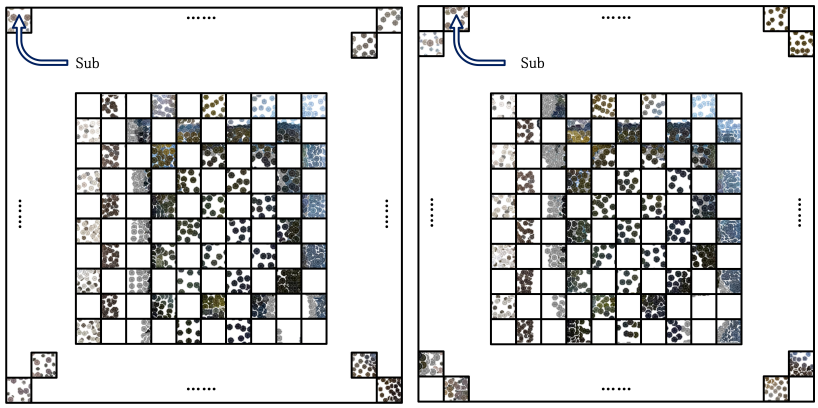
As shown in Figure 4, the $Sub$ is a cell in the grid, and it stores a part of the pointcloud information in the 3D space. The standard practice is to form a grid from all cells, but in this paper, a grid with holed structure, as shown in Figure 5a,b, is used to let only part of the cells participate in the calculation. In fact, for a grid of a certain size, the number of

cells depends on its resolution. As shown in Figure 6, the higher the resolution, the more cells there are, and the denser the grid, the better its mapping effect. However, the amount of computation increases dramatically. Therefore, this kind of holed-structure grid can save the calculation cost very well because it mainly requires half cells to participate in the calculation in each data frame.



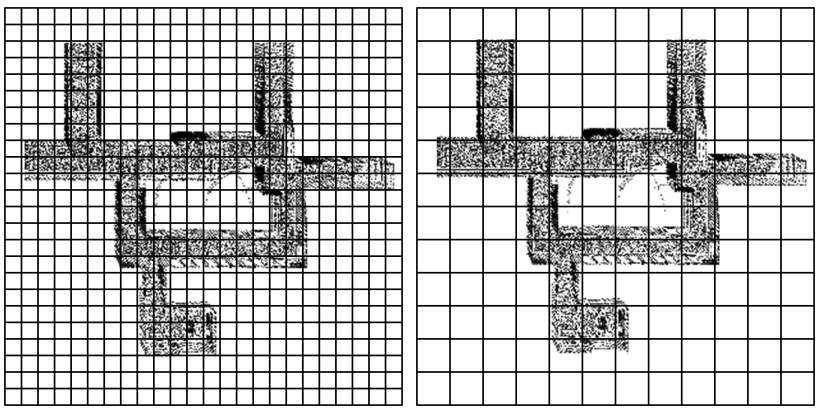

(**a**) Point cloud (**b**) Local grid

**Figure 4.** (**a**) is the spatial 3D pointcloud; (**b**) is the mapping of the pointcloud information in the local grid.




(**a**) Holed structure-based grid1 (**b**) Holed structure-based grid2

**Figure 5.** The complementary grids with holed structure.



(**a**) Grid with higher resolution (**b**) Grid with lower resolution

**Figure 6.** Grid with (**a**) higher resolution, and (**b**) lower resolution.

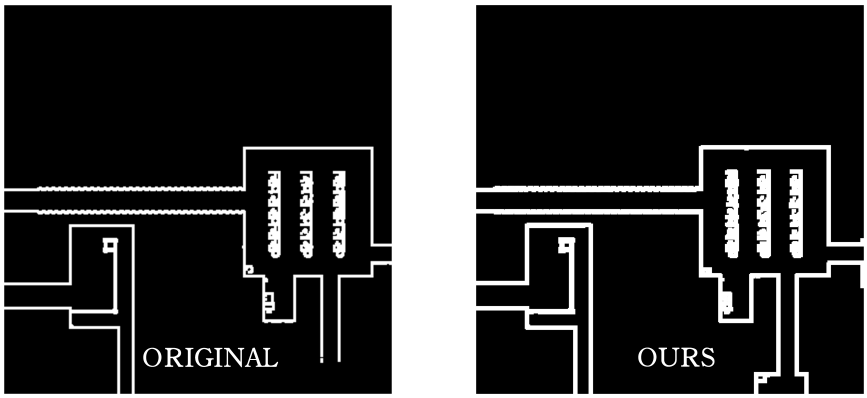
For the obstacle extraction process and obstacle vertices reconstruction process in the v-graph, the sensor information $\mathcal{S}$ is gridded and stored by the module $\mathcal{F}$ to obtain $\mathcal{S}^{'}$. After that, the $\mathcal{S}^{'}$ will be converted to a binary image $\mathcal{I}$. To enhance robustness, the image $\mathcal{I}$ will be blurred, then obstacle vertices will be extracted through image processing [50] to generate polygons $\left\{ P^{k}_{contour} \subset \mathcal{Q} \,|k \in Z^{+} \right\}$. The polygon extraction algorithm is shown in Algorithm 2.

To define the kernel size of the box filter in Algorithm 2, the equation is as follows, where $R_W$ and $R_L$ are the width and length of the robot, respectively, and $V_S$ is the voxel size. In this paper, $V_S$ is set to 0.15 m:

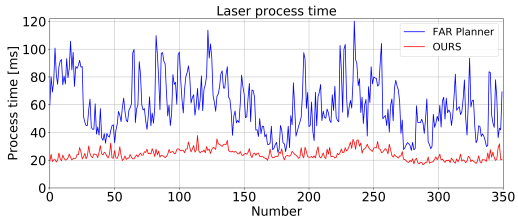
$$(kernel\ width,\ kernal\ height) = max(\lfloor \frac{\frac{max(R_W,\, R_L)}{2} + V_S}{V_S} \rceil,\ 5) \tag{1}$$

Figure 7 demonstrates the blurred picture of LiDAR-mapped obstacle geometry and the time consumption of the laser process. As can be seen from Figure 7a, the hollow structure (using module $\mathcal{F}$) does not affect the image after blurring. Compared with the original (without using a hollow structure), it can be found that our generated contour approximates the original image. The projected outline of the obstacle is thicker because of the blurred image, and the details inside the outline are lost.

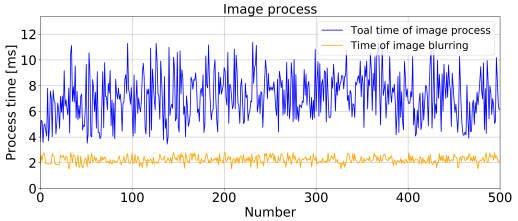
The time required for the laser process, according to Figure 7b, includes gathering the raw pointcloud and downsampling. For the same area, the time consumed by using the holed structure is more gentle, while the processing process without the holed structure is steeper and its curve fluctuates greatly. The time it takes to process an image is depicted in Figure 7c. The total time of the image process includes mapping the pointcloud in $\mathcal{L}_{local}$ into the image $\mathcal{I}$, blurring the image, and initially extracting the obstacle contour points. About 20% of the total image processing time is spent on blurring the outline of obstacles.




(**a**) Blurred image comparison




(**b**) Laser process consumption




(**c**) Image process

**Figure 7.** (**a**) shows blurred images, the vertices of the obstacle will be extracted through the blurred image. The FAR Planner generates images without holed structure, and ours generates pictures through the holed structure. (**b**) shows the consumption of the laser process and (**c**) shows the consumption of the image process.

**Algorithm 2** Polygon Extraction.

**Input:** $\mathcal{S}^{'} \in \mathcal{L}_{local}$
**Output:** $Polygons : \left\{ P^{k}_{contour} \right\}$

1: $input\ \mathcal{S}^{'}$
2: $Create\ binary\ image\ \mathcal{I}\ from\ points\ in\ \mathcal{S}^{'}$
3: $Apply\ box\ filter\ with\ kernal\ size\ of\ (kernel\ width,\ kernal\ height)\ \ to\ blur\ image\ \mathcal{I}$
4: $Extract\ polygons\ \left\{ P^{k}_{contour} \right\}\ based\ on$ [50]
5: **for** $each\ P^{k}_{contour}$ **do**
6: $\quad Downsample\ vertices\ in\ P^{k}_{contour}\ based\ on$ [51]
7: **end for**

### *3.2. Simplified Complex Contours Algorithm*

In the previous Section 3.1, the $\mathcal{L}$ was constructed to obtain the pointcloud from LiDAR. In this section, the polygons will be extracted and simplified based on $\mathcal{L}$.

For the graph update method of the two-layer architecture, shown in Figure 8, we define $\mathcal{G}_{local}$ and $\mathcal{G}_{global}$. Between them, $\mathcal{G}_{local}$ is the local layer around the robot, and $\mathcal{G}_{global}$ is the layer set of the entire observation environment. $\mathcal{G}_{local}$ will be generated by the sensor information $\mathcal{S}$ for each data frame and then merged into $\mathcal{G}_{global}$ For each data frame, the sensor information $\mathcal{S}$ will generate $\mathcal{G}_{local}$ and then be merged with $\mathcal{G}_{global}$, noting that since the module $\mathcal{F}$ is used to store the sensor information $\mathcal{S}$, now $\mathcal{S}^{'} \in \mathcal{L}_{local}$ can be used to merge $\mathcal{G}_{global}$ at a lower cost.

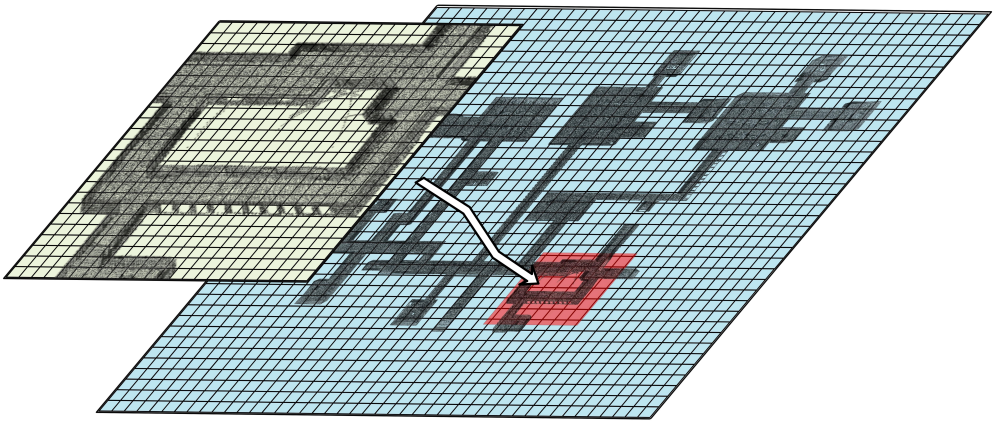


**Figure 8.** A two-layer update structure, where the blue grid is the global map, and the turquoise grid is the local map. The local map is located in the red boxed area in the global map.

It is known that the computational complexity involved in constructing a v-graph is $O(n^2 \log n)$ [52], where $n$ is the number of vertices in the graph. In normal case, the cost of building a local graph in the environment is small enough so that computational resources can be allocated to each data frame in an incremental update manner. However, redundant nodes will also be generated during each v-graph update if the environment has numerous complex obstacles, leading to a significant increase in the number of edges connecting the nodes. To ensure the effectiveness of the v-graph update, a method for further sparse operation on complex contours is required.

Constructing local layers: The $\mathcal{S}^{'} \in \mathcal{L}_{local}$ will be converted into local polygons $\left\{ P^{k}_{contour} \right\}$, and use $\left\{ P^{k}_{contour} \right\}$ to construct a local visualization graph $\mathcal{G}_{local}$. Note that for complex polygons, as shown in Figure 9, the polygon contains many vertices composed of short edges. Adding redundant vertices will construct more useless edges; thus, a lot of computing resources are wasted on unnecessary vertices and edges in the process of path search in complex terrain.

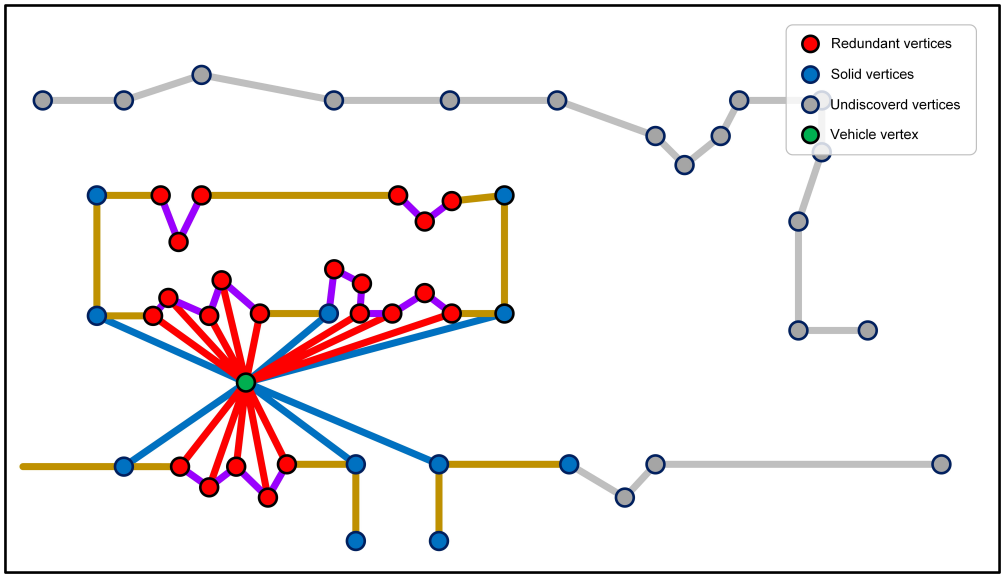
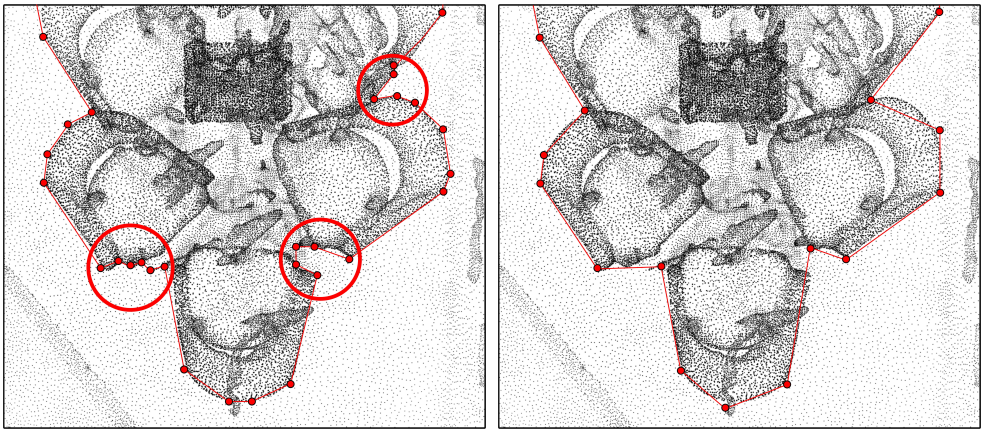



**Figure 9.** The red solid lines are redundant edges which connect with the robot, and the purple lines are the redundant edges from obstacles themselves.

A threshold $\eta$ is set to control the number of vertices for complex large local polygons. When the number of vertices of the polygon $\left\{P^k_{contour}\right\}$ is greater than $\eta$, the vertices will be reduced, which not only optimizes the geometric outline of large and complex obstacles, but also retains the geometric characteristics of small obstacles. As shown in Figure 10, the continuous vertices inside those red circles in Figure 10a should be eliminated, but the current method does not eliminate them well, resulting in more vertices and edges in the v-graph. Compared to Figure 10a, the optimized version in Figure 10b has fewer vertices.



(**a**) Extract vertices (**b**) Optimization after extracting vertices

**Figure 10.** (**a**) shows the obstacle vertices extracted after pointcloud mapping. (**b**) shows the remaining obstacle vertices after optimizing the (**a**).

When the number of vertices of an obstacle is greater than $\eta$ in the local layer, the algorithm preferentially records the distance between the two longest vertices in the obstacle. For example, the distance between the longest two vertices is $dist_{max}$. The algorithm traverses the three consecutive vertices $vertex_{i-1}$, $vertex_i$, $vertex_{i+1}$ in the obstacle and calculates the length between the two vertices, respectively. The distance between them is denoted as $dist_{(i,i-1)}$ and $dist_{(i,i+1)}$. If both $dist_{(i,i-1)}$ and $dist_{(i,i+1)}$ are less than $0.1\times\ dist_{max}$, it means that $vertex_i$ is an invalid vertex (excess vertex), in which case we delete $vertex_i$ and destroy its connection edges $edge_{(i-1,i)}$ and $edge_{(i,i+1)}$.

Since the simplified complex contours algorithm only works on the $\mathcal{G}_{local}$, the v-graph update process will not be slowed down by the accumulation of the number of nodes in the $\mathcal{G}_{global}$.

Update the global layer: After $\mathcal{G}_{local}$ is constructed, the $\mathcal{G}_{local}$ and the $\mathcal{G}_{global}$ are fused. The strategy is: take out the overlapping parts of $\mathcal{G}_{local}$ in $\mathcal{G}_{global}$, and associate the vertex position in the $\mathcal{G}_{local}$ to the $\mathcal{G}_{global}$. The Euclidean distance is used to associate vertices in two layers, and the associated vertices are recorded. The entire graph updating algorithm is as follows in Algorithm 3, and the final obstacle contours and edges are shown in Figure 11.

For the given two points a$(a_x, a_y, a_z)$ and b$(b_x, b_y, b_z)$, the $distance(a, b)$ in Algorithm 3 is defined as followed:

$$distance(a, b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2} \tag{2}$$

**Algorithm 3** Visibility Graph Update.

**Input:** $\mathcal{S}^{'} \in \mathcal{L}_{local}$, *graph* $\mathcal{G}$
**Output:** *Update graph* $\mathcal{G}$

$\left\{P^k_{contour}\right\} \leftarrow$ *Polygon Extraction*$(\mathcal{S}^{'})$; // *from Algorithm*2
**for** *each* $P^k_{contour}$ **do**
  **if** *the number of vertices* $> \eta$ **then**
    **for** *each vertex in contour* **do**
      $dist_{max} = max(distance(vertex_i, vertex_{i+1}))$
    **end for**
    **while** *true* **do**
      **for** *each vertex in vertices* **do**
        $dist_{(i,i-1)} = distance(vertex_{i-1}, vertex_i)$
        $dist_{(i,i+1)} = distance(vertex_i, vertex_{i+1})$
        **if** $dist_{(i,i\pm1)} < 0.1 \times dist_{max}$ **then**
          *Eliminate* $vertex_i$
          *Eliminate unnecessary* $edge_{(i-1,i)}$ *and* $edge_{(i,i+1)}$
        **end if**
      **end for**
      **if** *all* $dist_{(i,i\pm1)} \geq 0.1 \times dist_{max}$ **then**
        *break*
      **end if**
    **end while**
  **else**
    *continue*
  **end if**
**end for**
*Associate vertices* $P^k_{contour}$ *in the* $\mathcal{G}_{global}$
*Upate to visibility graph* $\mathcal{G}$

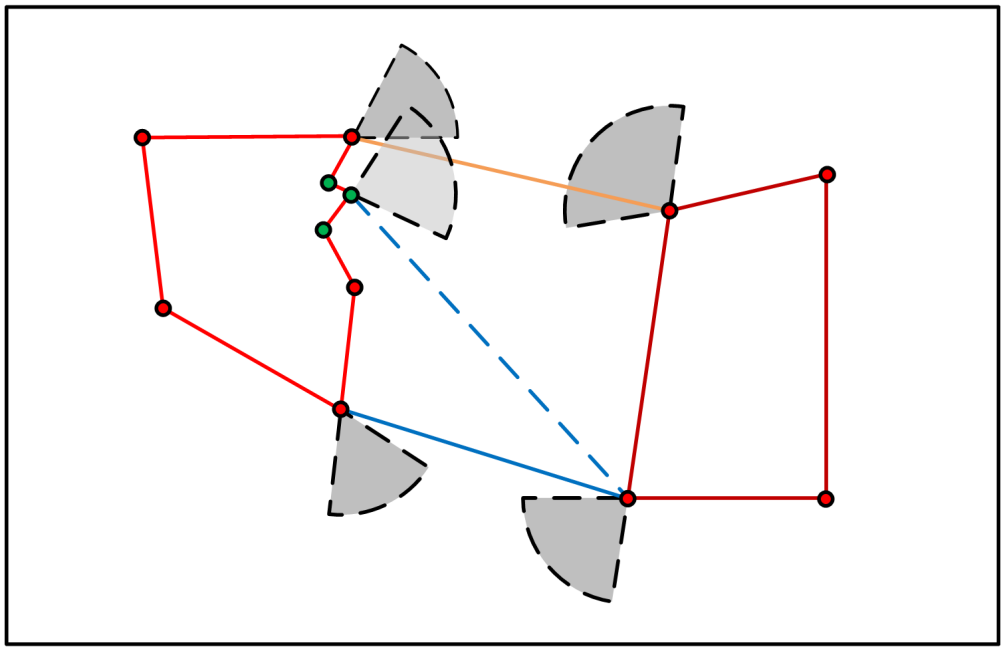


**Figure 11.** An illustration of sparse v-graph. The edge (orange) that head into at least one polygon from the shaded angle are eliminated, and the blue one will be kept. After eliminating those green vertices, the dot blue edge will be removed from the $\mathcal{G}_{local}$.

### *3.3. Path-Planning Based on Bidirectional Breadth-First Search*

In FAR Planner, the goal point $P_{goal}$ is used as a vertex, and the Euclidean distance is used as the score to update the parent node of $P_{goal}$. Although the path can be found in an unknown environment, its spatial search range is obviously wasted. The strategy adopted by the robot in many cases is to explore many unnecessary spaces until it finally reaches the goal. As shown in Figure 12, the robot travels from position 0 (start) to position 1, resulting in unnecessary exploration space.

A bidirectional breadth-first search (bidirectional BFS) structure is combined with the v-graph to search for a path, selecting a vertex of a connecting edge of the robot in the forward search while simultaneously beginning a backward search from the $P_{goal}$ to find the path to the robot's current position. This minimizes the amount of unnecessary exploration space.

In the planning, assume that there are no obstacles in the unknown area where the $P_{goal}$ is located. The $P_{goal}$ uses geometric collision checking to establish edges with existing vertices $\left\{ P_{node} \mid P_{node} \subset \mathcal{G}_{global} \right\}$ in the v-graph, and then the $P_{goal}$ will be connected to the vertices of the discovered obstacles in the v-graph as shown in Figure 13a. The one-way BFS usually wastes some search space in unknown or partially known environments. This is because the one-way BFS starts from the nodes connected to the robot, calculates the target point according to the cost, and then iterates to the robot position according to the parent node of the target point.

As shown in Figure 13b,c, the one-way BFS enters a fork in the planning of the global path from the starting point to the ending point, resulting in an increase in the search space. The result is shown in Figure 13d, from the red point to the green point, the one-way search wastes a huge amount of space. Therefore, this paper embeds the goal in the v-graph and associates it with the existing vertices in the graph, and adopts a bidirectional breadth-first algorithm for path search.

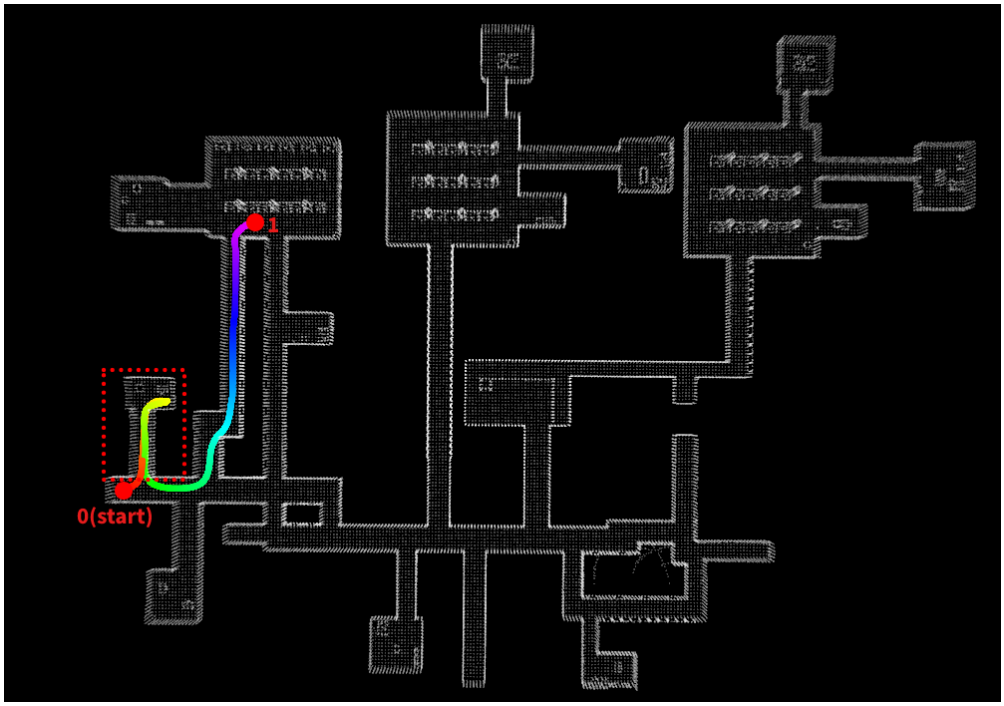



**Figure 12.** The robot travels from position 0 to position 1, and the red dotted box represents the wasted exploration space during navigation.

The bidirectional BFS structure shows in Algorithm 4, in which the two BFS are divided into forward and backward according to the direction of the search (forward searches from the robot position to the $P_{goal}$, and backward searches from the $P_{goal}$ to the robot position).

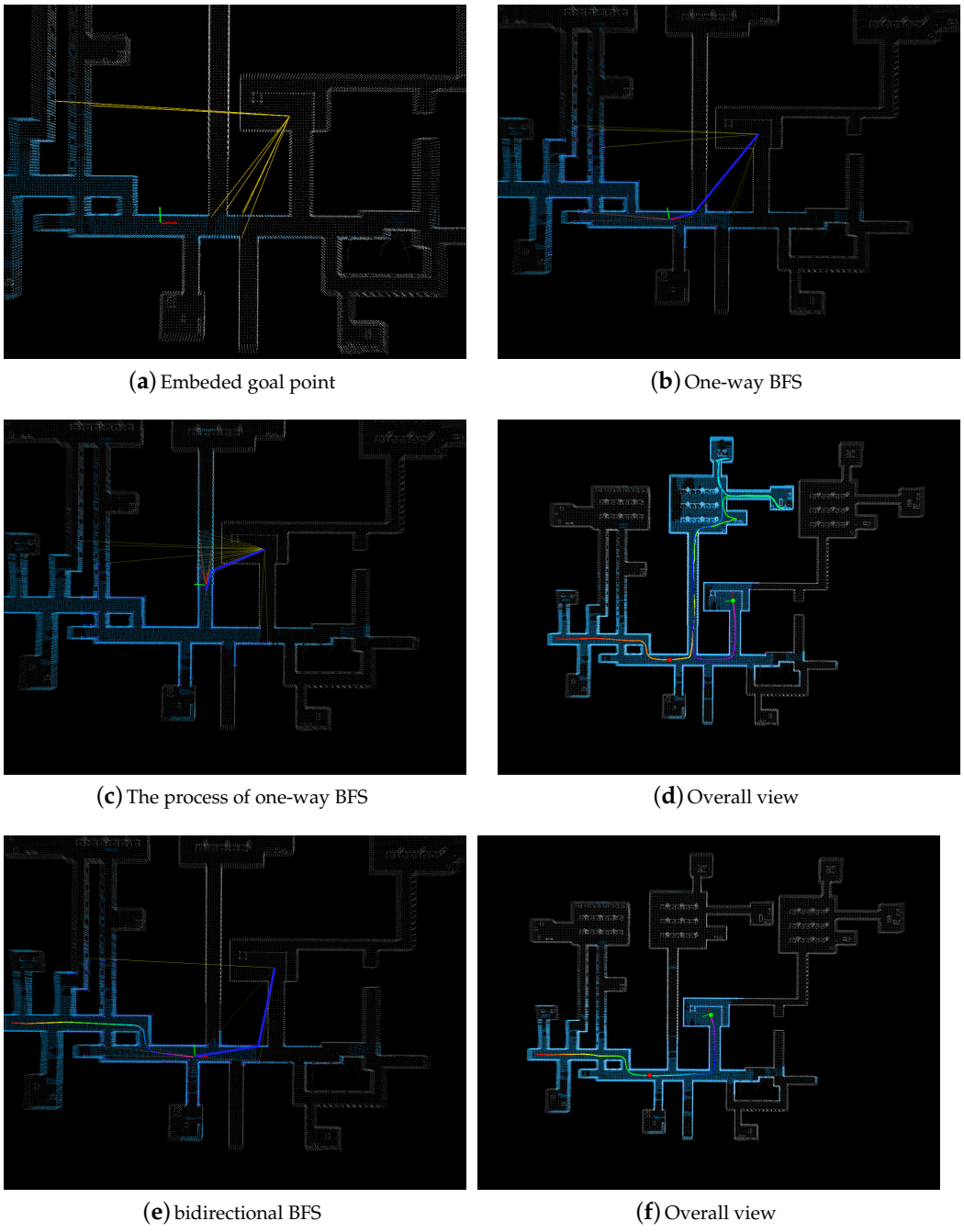
In the Algorithm 4, $parent_F(\cdot)$ and $parent_B(\cdot)$ are the functions returning the forward and backward parent of a node. $Q_F$ and $Q_B$ are the min-priority queues in forward and backward ones, respectively, and $Q_F$ is ordered by $g_F$, $Q_B$ is ordered by $g_B$. $\mu$ is the cost of the best path found so far (initially, $\mu$ is set to $\infty$). Whenever the robot reaches a node and expands in the other search, $\mu$ will be updated if a better path goes through the node. $g_F$ is the current distance from start and $g_B$ is the current distance from $P_{goal}$. $top_F$ and $top_B$ are the distances to the top nodes in the forward and backward queues, respectively. The STEP function in Algorithm 5 is responsible for advancing the search through the v-graph and updating $\mu$.

One of the benefits of bidirectional search is that it can, to some extent, avoid the wasted search space caused by entering invalid forks. As shown in Figure 13e, the globally planned path no longer passes through the fork, thus avoiding excessive searching. As a result, as shown in Figure 13f, compared to Figure 13d, the distance traveled by the robot is greatly reduced.

It is not possible to use a vertex that has just been extracted from the v-graph $\mathcal{G}$ as a point of navigation directly; instead, a transform is used to turn the vertex into a way-point. As Figure 14 shows, in the obstacle where the point is located, the vertices connected to the point at the polygon will be extracted to calculate the direction vector of the point (in Algorithm 6, they are $\overrightarrow{dir}_{front}$ and $\overrightarrow{dir}_{back}$, respectively, and the direction vector of the point is the $\overrightarrow{surf}_{dir}$).

A detailed description is shown in Algorithm 6. In Algorithm 6, the parameter of $search_{dist}$ is set to constrain the searching area, and the $near_{dist}$ is a step parameter which extends the way-point from $\overrightarrow{surf}_{dir}$ direction, and $R_W$ and $R_L$ are the width and length of the robot, respectively. When the way-point extends, the $max_{extend}$ is set to constrain the length of the extension. The $NearOBS(\cdot)$ function is to obtain a range of obscloud from $\mathcal{L}_{global}$, with $P_{way-point}$ as center and $search_{dist}$ as the radius. *Check_Collision* is used to

detect if the expanded $P_{way-point}$ collides with surrounding obstacles, and the detail of the *Check_Collision* function is shown in Algorithm 7.



(**a**) Embeded goal point (**b**) One-way BFS

(**c**) The process of one-way BFS (**d**) Overall view

(**e**) bidirectional BFS (**f**) Overall view

**Figure 13.** The blue area of the map represents the part that has been explored by LiDAR and is considered known. (**a**) shows the connection between the endpoint and the existing node, represented by a solid yellow line. (**b**,**c**) show the path planning using one-way BFS. (**d**) is the result of one-way BFS. The red dot is the starting point, and the green dot is the endpoint. (**e**,**f**), respectively, show the path planning and the final result using bidirectional BFS. The globally planned path appears on the map as a thick blue line.

**Algorithm 4** Bidirectional BFS.

**Input:** $P_{start}$, $P_{goal}$, *Visibility Graph*: $\mathcal{G}$

**Output:** $path : \left\{ P_{path} \right\}$

$Q_F, Q_B \leftarrow make\ min-priority\ queues\ for\ the\ nodes\ initially\ containing\ only\ P_{start}\ and\ P_{goal}$
$expanded_F, expanded_B \leftarrow make\ forward\ and\ backward\ lookup\ tables$
$search_F \leftarrow (Q_F, expanded_F, parent_F)$
$search_B \leftarrow (Q_B, expanded_B, parent_B)$
$\mu \leftarrow \infty$
$initialize\ \mathcal{G}, associate\ P_{goal}\ in\ \mathcal{G}$
$path \leftarrow none$
**while** $top_F + top_B < \mu$ **do**
    **if** *at least one queue is non − empty* **then**
        *choose the search to advance*
    **else**
        **return** *path*
    **end if**
    **if** $forward\ search\ was\ chosen$ **then**
        $(path, \mu) \leftarrow$ **STEP**$(search_F, search_R, path, \mu)$
    **else**
        $(path, \mu) \leftarrow$ **STEP**$(search_R, search_F, path, \mu)$
    **end if**
**end while**
**return** *path*

**Algorithm 5** STEP $(search_1, search_2, solution, \mu)$.

// 1 denotes the chosen direction and 2 is other direction
// $c(\cdot)$ is the cost function, in this paper, manhattan distance is used
// $c(u, v) = |u_x - v_x| + |u_y - v_y| + |u_z - v_z|$
$\mu \leftarrow pop\ the\ min\ g_1\ node\ from\ Q_1$
**for** $v \in parent_1(u)$ **do**
    **if** $v \notin expanded_1 \cup Q_1\ or\ g_1(u) + c(u, v) < g_1(v)$ **then**
        $g_1(v) \leftarrow g_1(u) + c(u, v)$
        Add $v$ to $Q_1$
        **if** $v \in expanded_2\ and\ g_1(v) + g_2(v) < \mu$ **then**
            $path \leftarrow reconstruct\ the\ path\ through\ u\ and\ v$
            $\mu \leftarrow\ g_1(v) + g_2(v)$
        **end if**
    **end if**
    **return** $(path, \mu)$
**end for**

In Algorithm 6, the $normalize(P_a, P_b)$ and $normalize(\cdot)$ are defined as followed:

$$normalize(P_a, P_b) = \frac{P_b - P_a}{||P_b - P_a||} \tag{3}$$

$$normalize(\overrightarrow{V}) = \frac{\overrightarrow{V}}{||\overrightarrow{V}||} \tag{4}$$

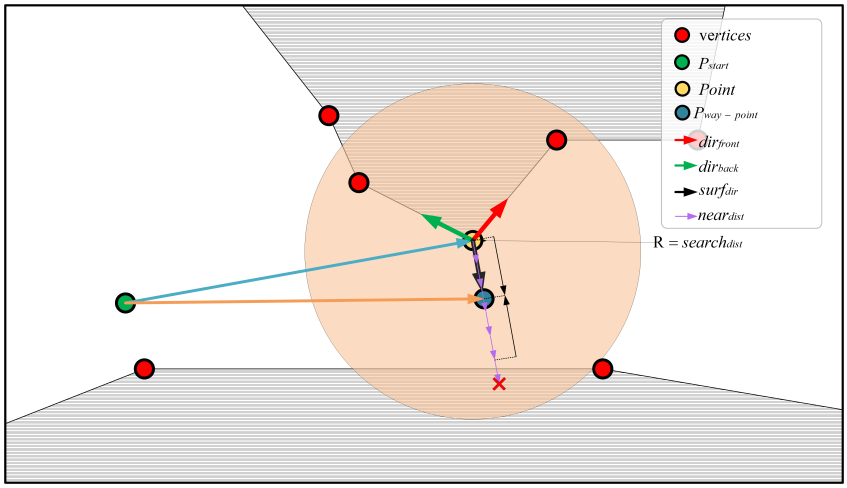



**Figure 14.** The schematic diagram of generating a way-point.

**Algorithm 6** Way-point generation.

**Input:** $P_{start}$, $Path$, $\mathcal{L}_{global}$
**Output:** $P_{way-point}$

$vertex = Path[0]$
$Path = Path[1:]$
$search_{dist} = max(2.5 \times max(R_L, R_W), 5)$
$near_{dist} = min(min(R_L, R_W), 0.5)$
$P_{way-point} = vertex$
$max_{extend} = min(search_{dist}, distance(P_{start}, P_{way-point}))$
$\overrightarrow{dir}_{front} = normalize(polygon_k(i-1), polygon_k(i))$
$\overrightarrow{dir}_{back} = normalize(polygon_k(i+1), polygon_k(i))$
**if** $P_{way-point}$ *is a convex point* **then**
    $\overrightarrow{surf}_{dir} = -normalize(\overrightarrow{dir}_{front} + \overrightarrow{dir}_{back})$
**else**
    $\overrightarrow{surf}_{dir} = normalize(\overrightarrow{dir}_{front} + \overrightarrow{dir}_{back})$
**end if**
$obscloud = NearOBS(\mathcal{L}_{global}, search_{dist}, P_{way-point})$
$obscloud = setInputCloud(obscloud)$ //setInputCloud is a pcl library function to build the KDTree of a set of pointcloud
$temp = P_{way-point} + \overrightarrow{surf}_{dir} \times near_{dist}$
$is\_collide = Check\_Collision(temp, obscloud)$ //from Algorithm 6
$extend_{dist} = near_{dist}$
**while** $is\_collide$ *is* $false$ *and* $extend_{dist} < search_{dist}$ **do**
    $temp += \overrightarrow{surf}_{dir} \times near_{dist}$
    $extend_{dist} += near_{dist}$
    $is\_collide = Check\_Collision(temp, obscloud)$
    **if** $extend_{dist} < max_{extend}$ **then**
        $P_{way-point} = temp$
    **end if**
**end while**
**if** $is\_collide$ *is true and* $extend_{dist} > max(R_W, R_L)$ **then**
    $P_{way-point} = \frac{P_{way-point} + vertex - \overrightarrow{surf}_{dir} \times near_{dist}}{2}$
    **return** $P_{way-point}$
**else**
    **return** *drop this vertex and re* $-$ *search path*
**end if**

**Algorithm 7** Check_Collision.

**Input:** *point P*, *obscloud*
**Output:** *false or true*

1: $radius = near_{dist}/2 + V_S$
2: $thre = near_{dist}/V_S$
3: $KDTree \rightarrow radiusSearch(P, radius, indices, dis)$ // search points with P as center and radius as the search radius, return the number of points in indices and the distance between each point and P.
4: **if** $indices > thre$ **then**
5: **return** *true*
6: **else**
7: **return** *false*
8: **end if**

## 4. Experiments and Results

The paper uses the same experimental parameters as FAR Planner [19], (uniform sensor parameters, the robot speed is set to 2 m/s). A highly complex channel network tunnel, a parking garage with multiple floors, and a forest of trees with many irregularly shaped trees are all included in the simulated experimental environment. The indoor is moderately complex but easy to detour. Additionally, Matterport3D [48] provides a simple environment 17DRP5sb8fy (denoted as 17DR), a slightly complex environment PX4nDJXEHrG (denoted as PX4n), and a large complex environment 2azQ1b91cZZ (denoted as 2azQ).

In the simulation environment, all methods run on a 2.6Ghz i7 laptop, and the v-graph-based methods use images at 0.2 m/pixel resolution to extract points to form polygons. The local layer on the v-graph is in a 40 m $\times$ 40 m area with the robot in the center. The threshold of the length of each visibility edge is set to 5 m. Finally, the simulated mobile robot is 0.6 m long, 0.5 m wide, and 0.6 m high.

In the physical environment, ours runs on an embedded device, and the robot speed is set to 0.7 m/s. To adapt to the real environment, the local layer of the v-graph is set to 20 m $\times$ 20 m. The length, width, and height of the mobile robot are set as 0.32 m, 0.25 m, and 0.31 m, respectively.

### *4.1. Simulation Experiment*

#### 4.1.1. Laser Process Simulation Experiment

In the laser process simulation experiment, seven different types of environments are used to compare the holed structure with the original one. The experimental results are shown in Figure 15. The robot moves according to a fixed route, and the experiment analyzes the time of the laser processing process. The laser processing process refers to the whole process of extracting and storing the pointcloud information of the local layer into the grid and classification (obstacle pointcloud or free pointcloud).

As shown in Figure 15, for example, in the indoor environment, the robot will start from 0 (start) and pass through the target points 1, 2, 3, 4, 5, and 6 in sequence. The initial state of the robot is set to be in an unknown environment, and the known information in the environment is continuously accumulated through exploration. The program records the time of receiving and processing the pointcloud data from LiDAR during the movement of the robot.

Figure 16a is a summary of the average time of laser processing in different environments. It can be clearly seen from Figure 16a that our method used less time in the processing of the pointcloud information in every data frame. Figure 16b–h show that using the grid with the holed-structure leads to the smooth processing of the laser pointcloud. Compared with the original grid, the use of the holed structure can improve the processing speed of the pointcloud by 30.5$\sim$44.5%.

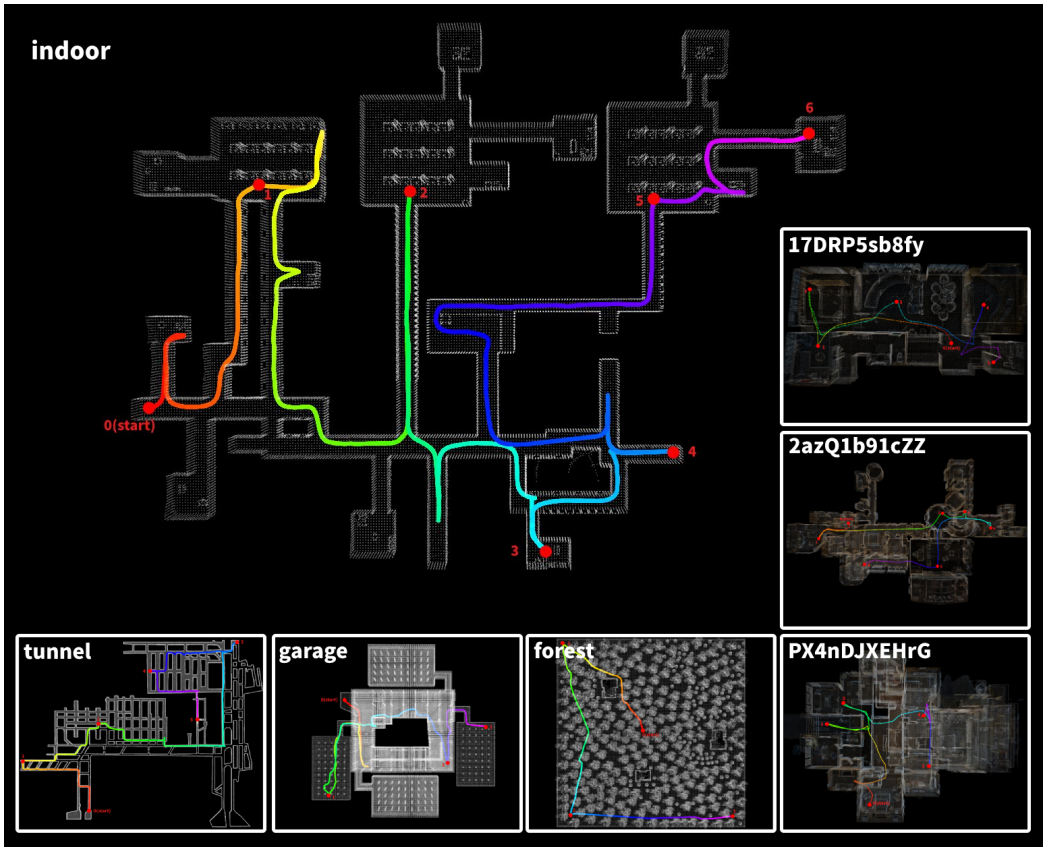
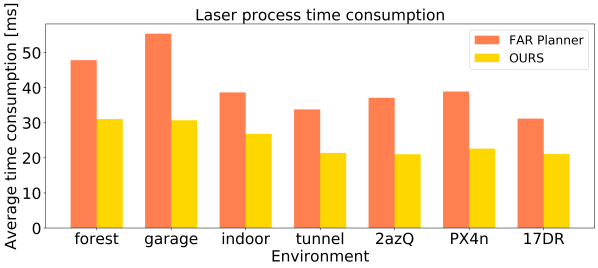
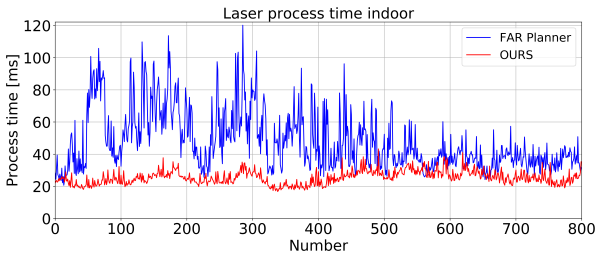


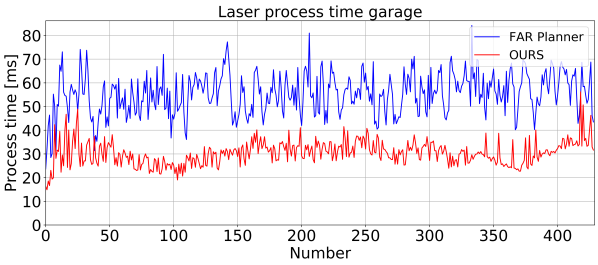

**Figure 15.** The overall view of seven different environments.



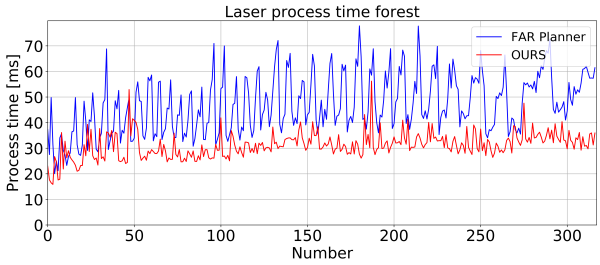

(**a**) Overall average process consumption




(**b**) Laser process(indoor)

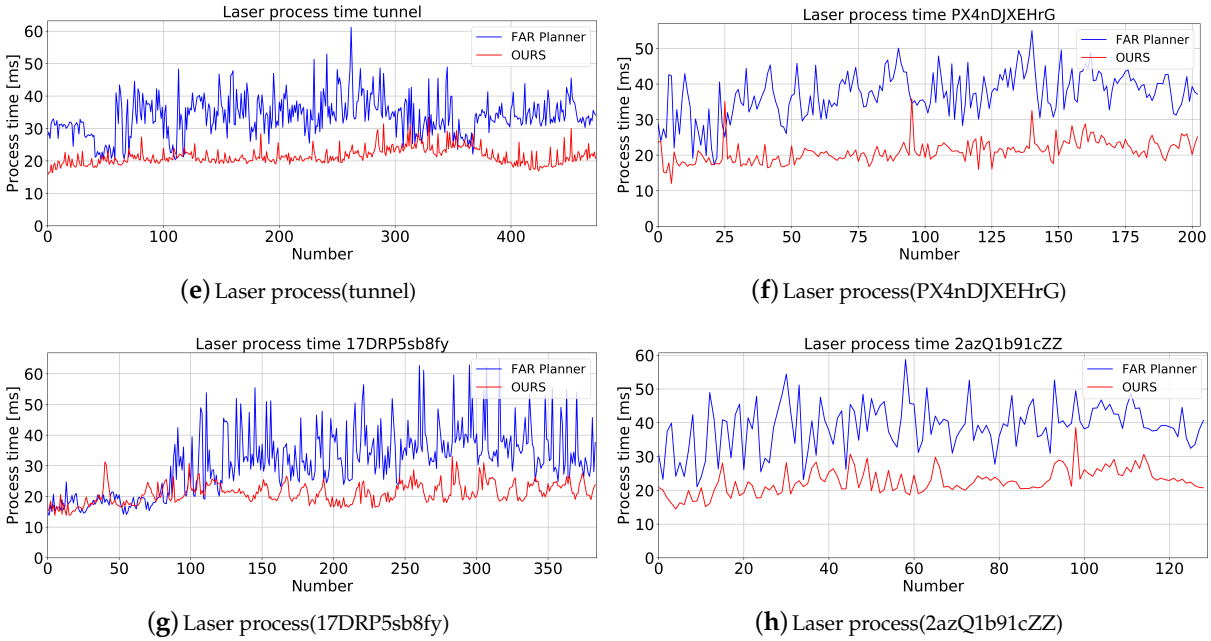



(**c**) Laser process(garage)




(**d**) Laser process(forest)

**Figure 16.** *Cont.*




(**e**) Laser process(tunnel)

(**f**) Laser process(PX4nDJXEHrG)

(**g**) Laser process(17DRP5sb8fy)

(**h**) Laser process(2azQ1b91cZZ)

**Figure 16.** (**a**) is the summary of the average time of laser processing in different environments. (**b**–**h**) show the process time of pointcloud in each data frame.

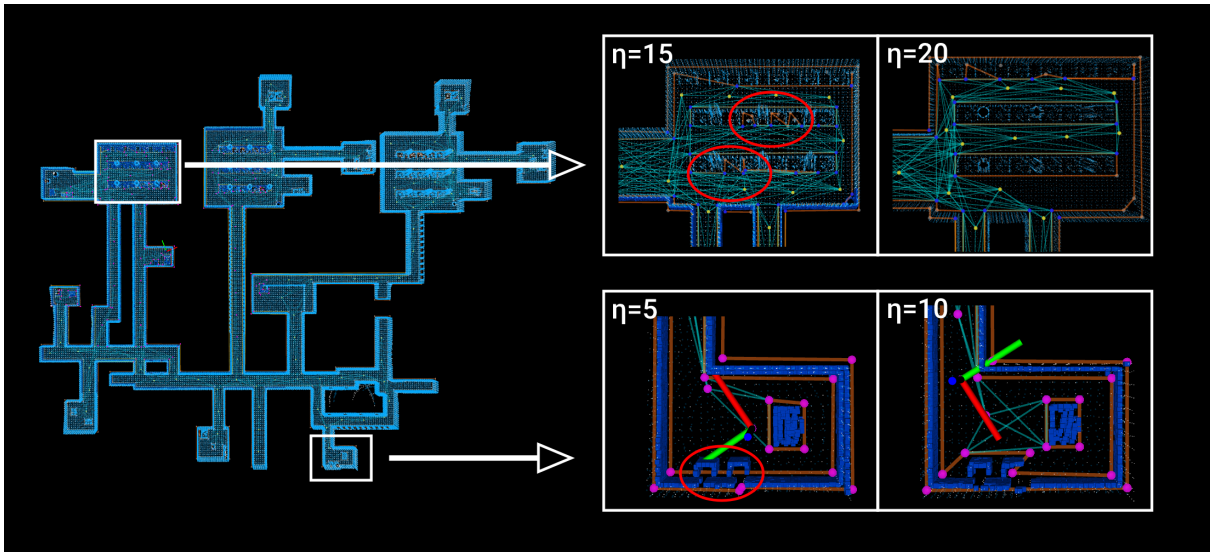
#### 4.1.2. Visibility Graph Update Simulation Experiment

Different values of $\eta$, ranging from 5 to 25, are set in representative indoor and outdoor environments in order to select an appropriate value. The robot in the simulation experiment travels throughout the entire environment to count all of the vertices in the v-graph and logs the average update speed. As the Table 1 shows, it can be seen that the number of vertices in the v-graph and the update speed of the v-graph are positively correlated with the value of $\eta$, but $\eta$ is not as small as possible.

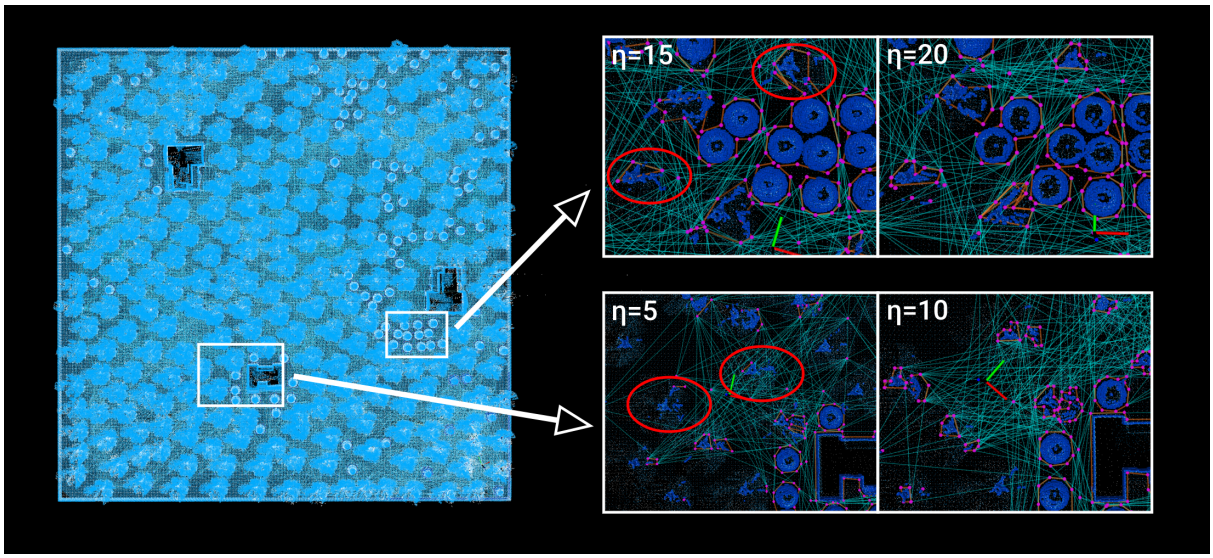
**Table 1.** Relevances among $\eta$, total vertices besides the speed of v-graph update.

| Test | Without $\eta$ | $\eta$ = 25 | $\eta$ = 20 | $\eta$ = 15 | $\eta$ = 10 | $\eta$ = 5 |
|---|---|---|---|---|---|---|
| total vertices (Indoor) | 904 | 873 | 725 | 714 | 695 | 630 |
| v-graph update (Indoor) | 21.48 ms | 20.64 ms | 14.13 ms | 13.88 ms | 13.82 ms | 12.81 ms |
| total vertices (Forest) | 3976 | 3589 | 2893 | 2838 | 2743 | 2334 |
| v-graph update (Forest) | 71.25 ms | 54.23 ms | 39.12 ms | 38.97 ms | 36.82 ms | 35.64 ms |

Theoretically, a smaller $\eta$ value should lead to fewer vertices in the v-graph. However, a value of $\eta$ that is too small will have some drawbacks. The function of collision detection may be affected when some minor obstacles are ignored, as shown in Figure 17a, and this phenomenon also occurs in forest environments. When $\eta$ is equal to 5, there are two small trees missing from the v-graph in Figure 17b. When $\eta$ is greater than or equal to 15 and less than or equal to 20, with the increase of $\eta$, the outline of the obstacle is well guaranteed, and the update speed of the v-graph is also relatively fast. When $\eta$ is greater than 25, the number of vertices in the v-graph gradually increases toward the direction of none $\eta$. After several rounds of testing, a preliminary conclusion can be drawn that when the value of $\eta$ is between 15 and 20, the algorithm is most suitable for generating obstacle vertices.



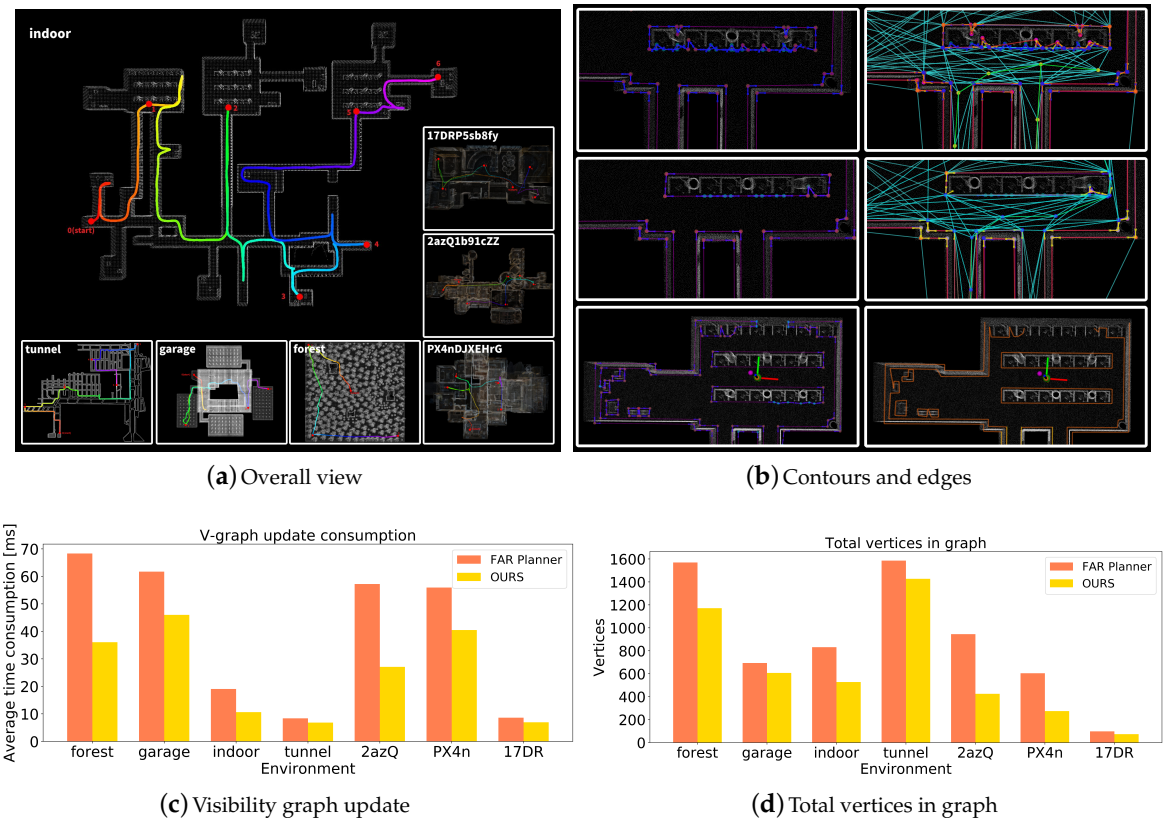

(**a**) Indoor environment




(**b**) Forest environment

**Figure 17.** Different $\eta$ in the indoor and forest environment.

In the v-graph update simulation experiment, seven different environments are used to compare our v-graph update method with FAR Planner's, and the parameter of $\eta$ is set to 20. In different environments, a series of target points are established for the mobile robot to travel. These points are fixed, and both methods let the mobile robot pass through them in sequence. For example, in the indoor environment, the robot will pass through the six target points 1, 2, 3, 4, 5, and 6 in sequence. During the driving process of the robot, the update speed of the v-graph will be recorded, and similarly, all the vertices in the v-graph will be recorded. The experimental results are shown in Figure 18.




(**a**) Overall view (**b**) Contours and edges

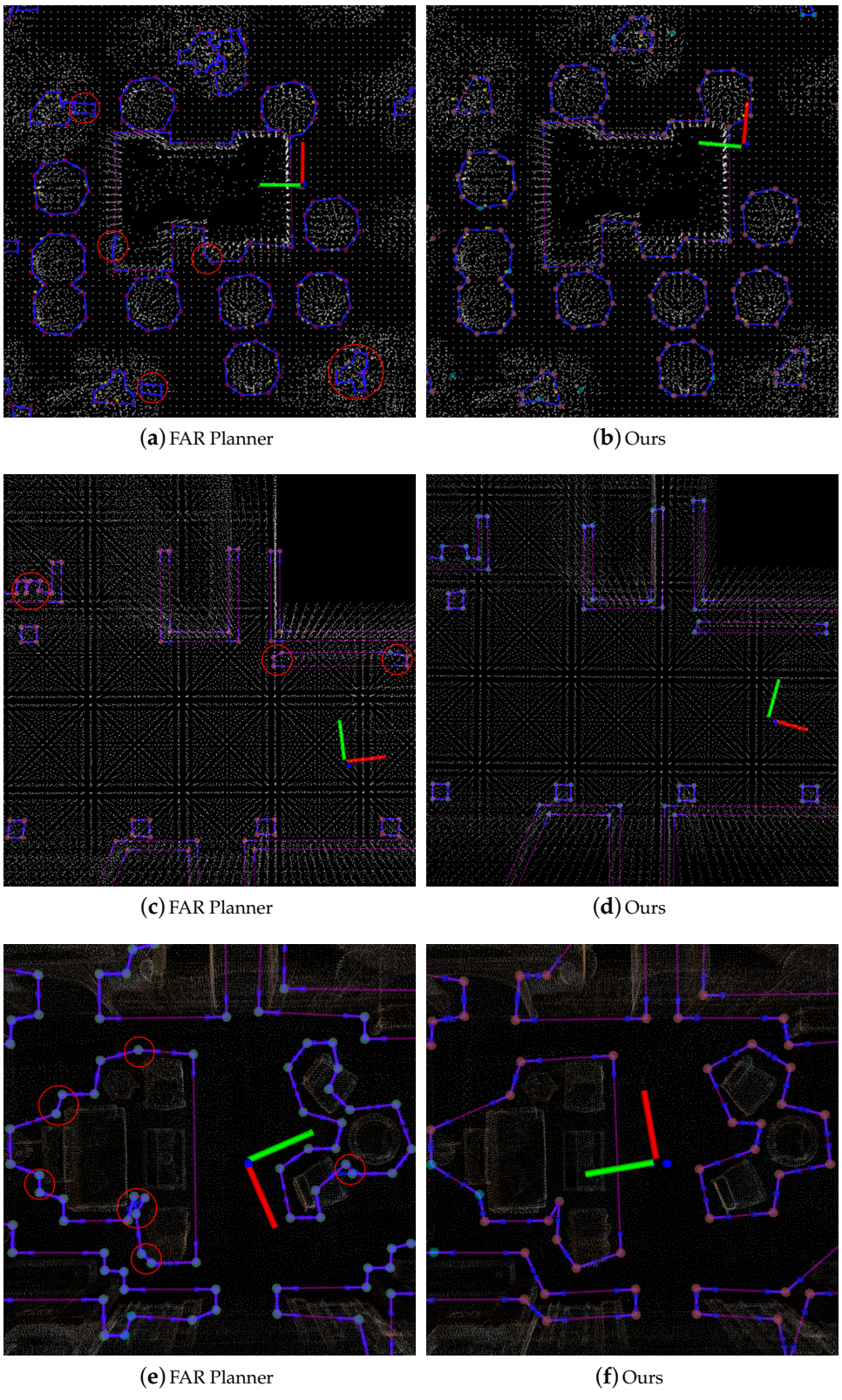
(**c**) Visibility graph update (**d**) Total vertices in graph

**Figure 18.** (**a**) shows that the robot runs in different environments and passes through a series of target points. (**b**) shows the geometric outline of obstacles and the connection of edges, red points are valid vertices, and cyan lines represent effective edges. The last one in (**b**) is the optimized global map. (**c**) shows the update speed of our method and the original method in different environments. (**d**) shows the number of vertices in the v-graph after running the same trajectory.

In each environment in Figure 18a, the robot passes through a series of target points, and the known environment information is reset after reaching each target point. Figure 18b shows the optimized nodes and edges for complex irregular objects. For complex obstacles, ours simplifies the vertex information of the obstacle. When the redundant vertices are reduced, the redundant edges will also be correspondingly reduced. As shown in Figure 18b, the cyan and blue edges and red nodes in the optimized v-graph are significantly reduced, respectively.

Figure 18c shows the average speed of the v-graph update, and Figure 18d shows the total number of vertices in the v-graph. For simple terrain and most of the obstacles with simple shapes and corners, such as the tunnel and 17DRP5sb8fy, our method obtains similar results to FAR Planner, and the improvement is only 10∼20%; for large-scale maps containing complex obstacles, such as the forest, 2azQ1b91cZZ, PX4nDJXEHrG, indoor, etc., our method significantly improves the efficiency by 40∼60% compared to FAR Planner.

It is evident from Figure 19 that for environments with complex terrain, our method greatly reduces the number of vertices for obstacles in the v-graph. In Figure 19e,f, the original method has a total of 73 vertices while our method has only 49 vertices.



(**a**) FAR Planner (**b**) Ours

(**c**) FAR Planner (**d**) Ours

(**e**) FAR Planner (**f**) Ours

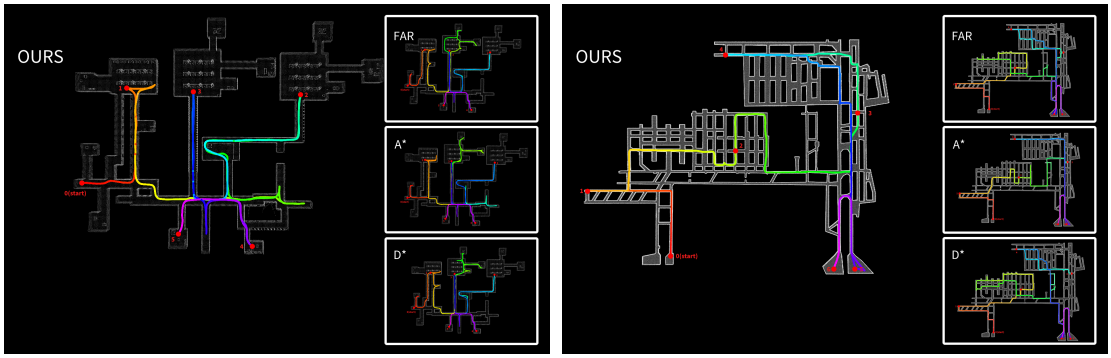
**Figure 19.** From (**a**–**f**) is the comparison result of the extracted obstacle vertices.

#### 4.1.3. Path Planning Simulation Experiment

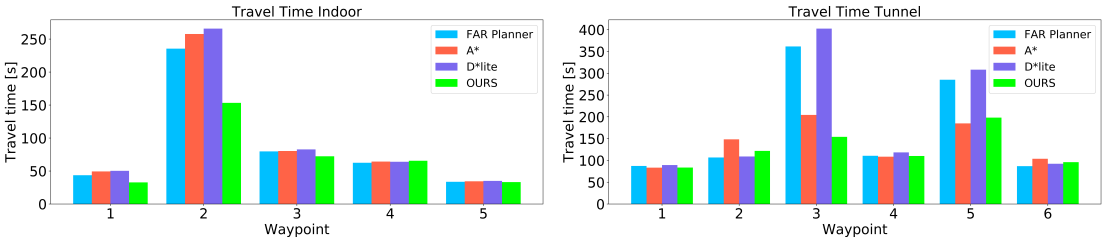
For the path exploration in the unknown environment, we compared the slightly complex indoor environment and the tunnel environment with complex network structures, respectively. Similar to the graph update simulation experiment, a series of waypoints are

set up in each environment, allowing the robot to pass one by one. Define the robot to accumulate environmental information while exploring the unknown environment.

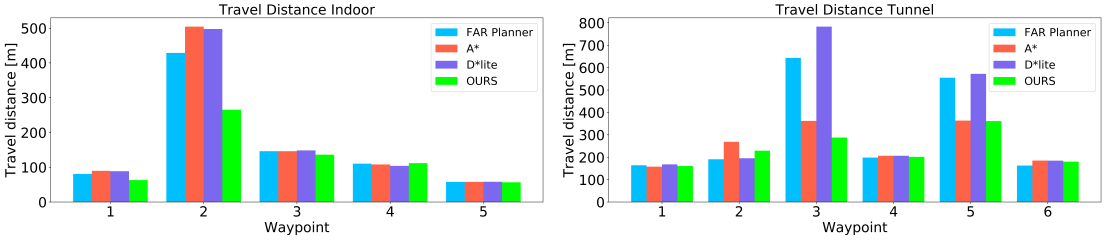
FAR Planner (v-graph-based), A*, and D* Lite (occupancy grid-based) are all added for comparison with our algorithm. Figure 20 shows the trajectory paths generated by our algorithm and other algorithms in navigation. In the case of reaching the same navigation point, our algorithm can avoid unnecessary exploration space to a greater extent than the FAR Planner and A*, D* Lite, so as to achieve a shorter distance and less time.



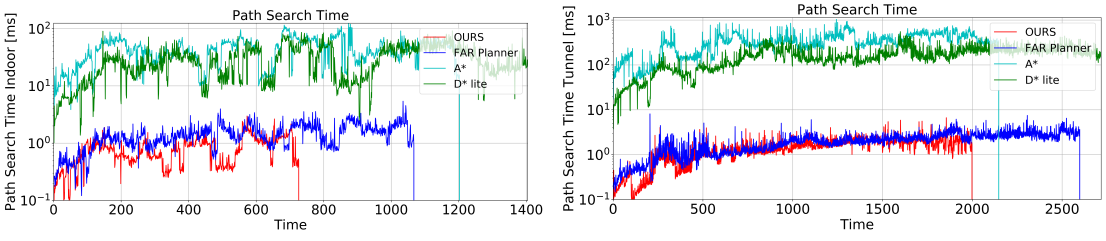

(**a**) Indoor (**b**) Tunnel




(**c**) Travel Time (Indoor) (**d**) Travel Time (Tunnel)




(**e**) Travel Distance (Indoor) (**f**) Travel Distance (Tunnel)




(**g**) Path Search Time (Indoor) (**h**) Path Search Time (Tunnel)

**Figure 20.** (**a**,**b**) show that the trajectory of the robot using different algorithms to navigate in indoor and tunnel simulation environments respectively and passes through a series of target points. (**c**,**d**) show the time consumption for each target point. (**e**,**f**) show the distance robot takes to travel to each target point. (**g**,**h**) show different path-searching algorithms' time consumption.

Figure 20c–f show the time and distance it takes for the robot to travel from one navigation point to another, and the robot accumulates map information during driving.

Tables 2 and 3 (FAR Planner denoted as FAR) gives a summary of the overall travel time, distance, and search time used for each map robot navigation.

As can be seen from Figure 20a for the indoor environment, the search space is wasted to varying degrees when using FAR Planner, A*, and D* Lite to navigate from point 1 to point 2. They are more inclined to explore the place where point 3 is located and then turn back after finding that there is no passage leading to point 2. The possible reason for this is that most of their cost functions only refer to the cost of the current node itself and the cost of the Euclidean distance between the endpoint and the current node. This causes the robot to tend to drive towards the node with the lowest total cost in a single direction, even though that node may not be able to reach the goal.

For adjacent navigation points with relatively short distances, such as from navigation point 4 to navigation point 5 in an indoor environment, the time and distance consumed by all algorithms are not much different.

The A* and D* Lite are known for their search integrity in finding the optimal path. However, those methods are difficult to scale as the computational cost increases significantly when environments are large and complex [19]. For the tunnel environment, although the environment contains a series of complex #- and *T*-shaped structures, there are almost no dead ends, that is, the goal can be reached in any direction, so the robot hardly needs to be turned back during the running process. For traditional A* and D* Lite, due to the increase in scene scale, the number of grids that need to be calculated increases sharply, and the cost of algorithm operation increases significantly in large-scale scenes. However, for us, the use of bidirectional BFS allows us to avoid some bifurcations and travel a shorter distance to the destination point.

**Table 2.** The overall time spent by the robot using different algorithms in [s] in Indoor environment.

| Test | Overall Time [s] | Overall Distance [m] | Average Search Time [ms] |
|---|---|---|---|
| FAR(baseline) | 455.31 | 824.1 | 1.43 |
| A* | 486.4 | 906.3 | 44.21 |
| Compare to FAR | +6.7% | +9% | +2991% |
| D* Lite | 498.1 | 896.6 | 29.6 |
| Compare to FAR | +9.4% | +8.8% | +1967% |
| OURS | 357.2 | 633.1 | 0.81 |
| Compare to FAR | −21.5% | −19.5% | −43.3% |

**Table 3.** The overall time spent by the robot using different algorithms in [s] in Tunnel environment.

| Test | Overall Time [s] | Overall Distance [m] | Average Search Time [ms] |
|---|---|---|---|
| FAR(baseline) | 1038.2 | 1914.4 | 2.05 |
| A* | 833.6 | 1543.8 | 332.95 |
| Compare to FAR | −19.7% | −19.3% | +16,141% |
| D* Lite | 1119.8 | 2109.8 | 162.3 |
| Compare to FAR | +7.8% | +10.2% | +7817% |
| OURS | 763.8 | 1420.4 | 1.44 |
| Compare to FAR | −26.4% | −25.8% | −29.7% |

As shown in Figure 20a,b, our planner is able to search for shorter paths and generate effective trajectories. Table 2 shows that in the indoor case, our method reduces travel time by 23% compared to A*, 28% compared to D* Lite, and 21% compared to FAR Planner's original algorithm. In terms of increased time, FAR Planner has the most time wasted due to ineffective exploration, while A* and D* Lite are time-consuming when the robot is

constantly swinging back and forth in a certain position, but what all three have in common is wasted search space between some navigation points. Table 3 shows that in the tunnel case, our method produces the shortest distance, which is 8% shorter than A*, 32% shorter than D* Lite, and 25% shorter than FAR Planner.

Tables 2 and 3 show that our planning algorithm can run faster because of the use of a hole-structured mesh to update the graph and vertex optimization for complex obstacles. Compared to FAR Planner, our search algorithm update rate is 43% faster.

### *4.2. Physical Experiment*

The physical experiment uses the mobile robot platform in Figure 21 with the speed set to 0.7 m/s. The mobile robot is equipped with a Velodyne-16 LiDAR and an Inertial Measurement Unit (IMU) 9250. The entire autonomous system is built on Robot Operating System [53], and the Raspberry Pi 4B is the master and the laptop is the slave. In the autonomous system, the master is used to transmit LiDAR data, run the CMU autonomous exploration interface [47], and drive the stm32. The navigation algorithm runs on the slave. The camera at 640 × 480 resolution in the mobile robot is only used to obtain pictures of the environment. The LiDAR and IMU are coupled to generate the state estimation of the robot through Lego-LOAM [4]. The main system structure of the mobile robot is shown in Figure 22, the autonomy system incorporates navigation modules from CMU autonomous development interface, e.g., terrain analysis and way-point following as fundamental navigation modules.

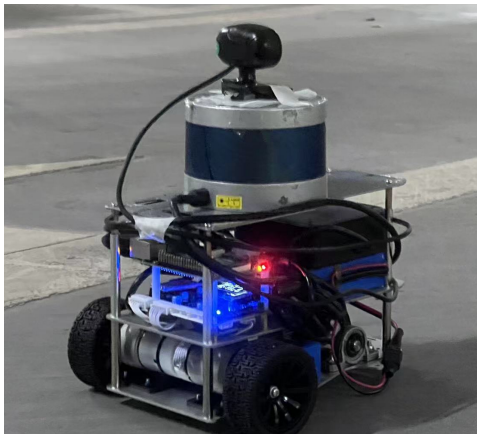


**Figure 21.** The mobile robot.

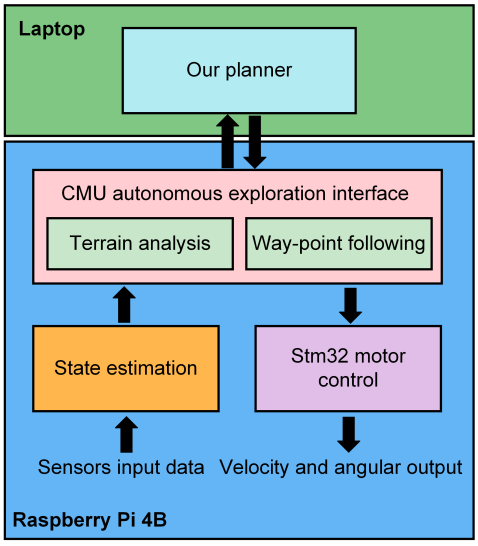
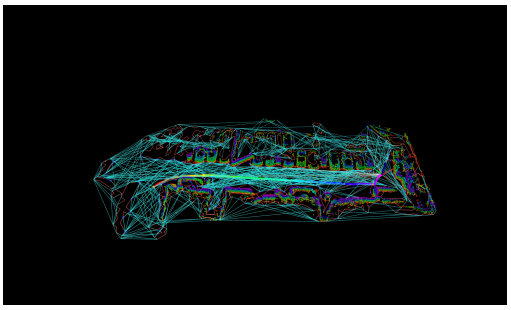


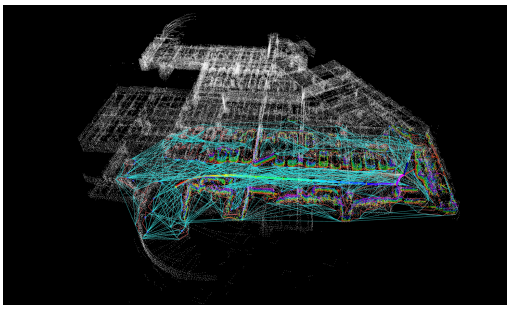

**Figure 22.** The main structure of autonomous system.

As shown in Figure 23a,b, obstacles are mapped as polygons in exploration, and solid edges are formed from each relevant vertex. In Figure 23c, colored pointclouds of obstacles are displayed to better show the details; orange lines denote the obstacles' outlines, while cyan lines denote the relationships between the edges of various obstacles.

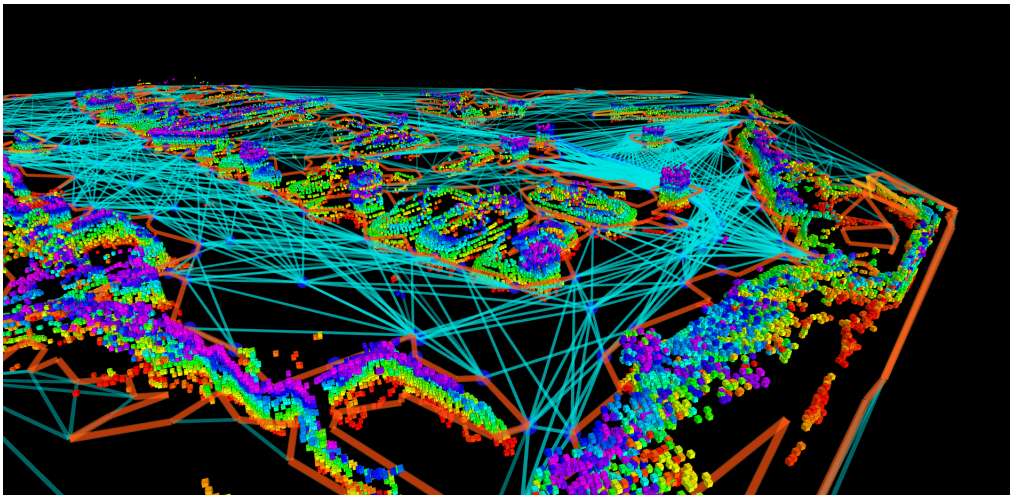
In the navigation, as shown on the left side of Figure 23d, the mobile robot started from point 1 and arrived at points 2, 3, and 4 in sequence and on the right side of the Figure 23d shows pictures when the mobile robot navigates to the corresponding position.



(**a**) Path planning and exploration



(**b**) Garage overview

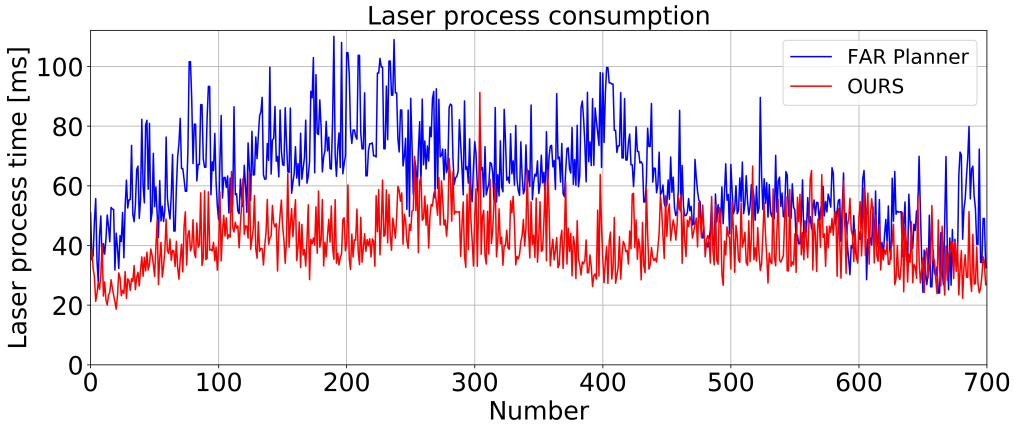


(**c**) Details of v-graph

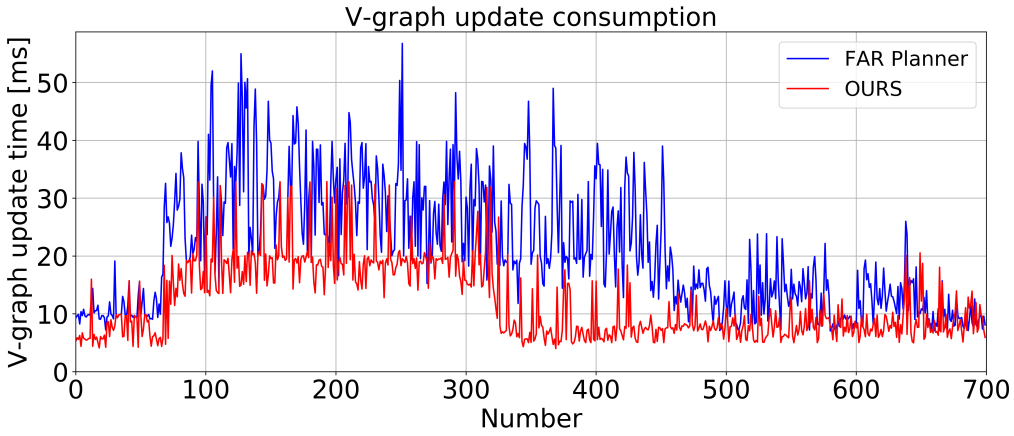

(**d**) Navigate in the garage

**Figure 23.** (**a**,**b**) show the path planning and exploration of the garage. (**c**) shows the details of exploration, and (**d**) shows the entire navigation in the garage.

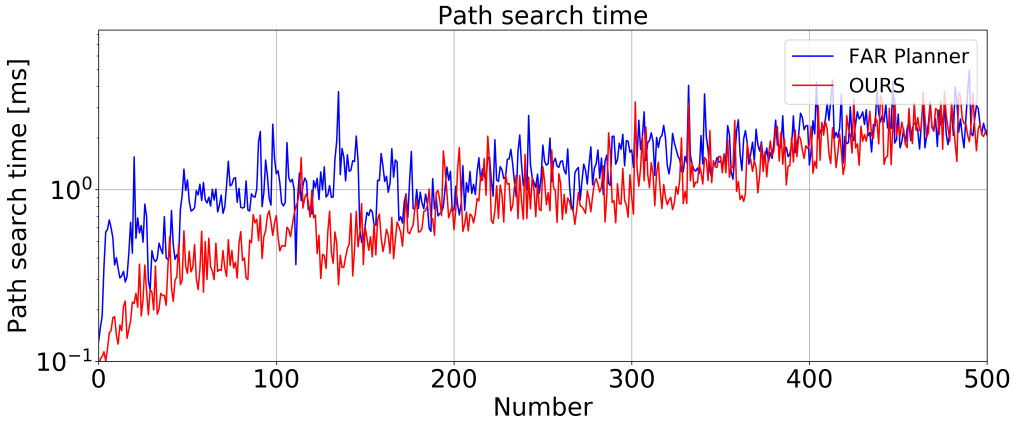
As shown in Figure 24, our algorithm is compared with FAR Planner in the physical experiment. The processing speed of the laser data, the update speed of the v-graph, and the operation speed of the search algorithm are recorded every 0.5 seconds.




(**a**) Laser process time in the physical experiment




(**b**) V-graph update time in the physical experiment




(**c**) Path search time in the physical experiment

**Figure 24.** Data recorded when the path planning algorithm runs.

In Figure 24a, The mean and standard deviation for the laser processing of FAR Planner are 60.95 ms and 16.59 ms, respectively, while our method's mean and standard deviation of the laser processing are 41.59 ms and 9.93 ms, respectively. Not only is ours 31.76% faster on average than FAR Planner, but the time taken by the algorithm is also more stable. As shown in Figure 24b, both approaches will take longer to update the v-graph because of the unexpected rise in obstacles, but ours is on average 37.12% quicker than FAR Planner. In Figure 24c, ours is on average 18.06% faster than the FAR Planner on the search algorithm.

## 5. Discussion

### *5.1. Grid in Mapping*

In the process of robot simultaneous localization and mapping, grids can be used to store pointcloud data [12,54–58], and can also be used as occupancy grid maps for robot navigation [56,59–63]. As mentioned in Lau, B et al. [58], the processing speed of a computer is limited by the resolution of the grid. When the grid resolution is higher, the information represented by each cell is more accurate, but the calculation time is longer. In Homm et al. [61], they use a Graphics Processing Unit (GPU) to speed up the formation of fine grids, and in A. Birk et al. [56], they use multiple robots to jointly maintain grid maps, an approach that indirectly speeds up the construction of the grid map.

The aim of the work in this paper is to use a discrete hollow grid to convert pointcloud information into a binary image and extract obstacle vertices. Therefore, the focus of this paper is on how to quickly extract the grid to generate the vertices of obstacles. For a 30 m $\times$ 30 m local grid, the processing speed of a high-resolution grid, such as 0.5 m $\times$ 0.5 m per cell, is much slower than that of a low-resolution grid, such as 1 m $\times$ 1 m per cell. Therefore, a grid with spaced hollow structures is designed to speed up the processing of high-resolution grids. Since the use of hollow structures will affect subsequent images, such as the discontinuous edges of obstacles, it is necessary to blur the generated images. Blurring the image can cover empty spots caused by hollow structures. Additionally, a complementary hollow-structured grid is also considered, storing a set of complementary hollow-structured grids under adjacent data frames and integrated into the local map.

It is foreseeable that the grid application with this sparse structure can significantly reduce the amount of computation and shorten the computation time in three-dimensional space. In future work, the authors hope to use this sparse structure for 3D grids.

### *5.2. The Reduced Visibility Graph*

The v-graph is a topology map that is widely used for path planning since it is constructed using the vertices of obstacles. The difficulty of calculating and maintaining the v-graph mainly depends on the number of vertices in the graph; thus, many researchers focus on how to simplify the v-graph [19,64–70].

In Nguyet et al. [69], he proposed a method for clustering small obstacles according to their volume, which can well reduce the number of vertices in the v-graph, thereby improving the efficiency of the path search algorithm. However, this method needs to calculate the total area of the global map for each iteration, which wastes a lot of time for multiple small-volume obstacles. In Yang et al. [19], an angle $\zeta$ is set to limit the visibility of each obstacle vertex, which can well reduce the number of edges of the obstacle vertex. The method used in this paper combined Yang's method [19] and proposed a method of simplifying obstacle vertices that only act on the local map. Compared with [69], the speed of the algorithm proposed in this paper is not affected by the global map, and it effectively reduces invalid vertices and redundant edges.

### *5.3. Uncertainties in the Path Planning*

The path planning is based on the produced map (they can be occupancy maps, topological maps, or semantic maps), and the localization of the robot is very important for constructing the map. In the simulation experiment, we can easily obtain the relevant

data of the robot, such as the simulated IMU sensor information and the simulated LiDAR information, and this information is accurate and unbiased in the simulated environment to estimate the state of the robot. How well the robot is positioned determines whether the map used for navigation is available. However, in real life, we cannot obtain such unbiased data; therefore, we used Lego-LOAM for the state estimation of the robot. If the robot's state estimation data has a large error, the v-graph it builds will deviate from the real world.

As shown in Figure 25, in the real world, the mobile robot made an error in the state estimation, and the white pointcloud newly scanned by the LiDAR was obviously offset from the colored obstacle pointcloud. This will lead to the establishment of an unreliable v-graph, thus affecting the effect of path planning.

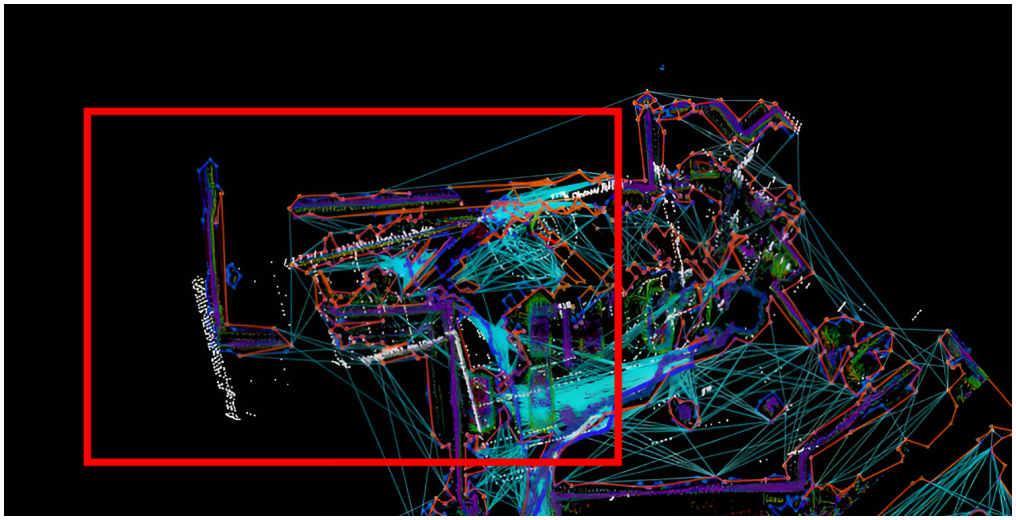


**Figure 25.** The unreliable v-graph.

## 6. Conclusions

This paper proposed a sparse visibility graph-based path planner based on the FAR Planner framework. Our method is far superior to the FAR Planner in terms of the efficiency of v-graph maintenance and generation. Our method can be used for navigation in known environments and exploration in unknown environments. Moreover, a complementary hollow grid is designed for local layer updates and merges the local layer into the global layer. For complex obstacles in the environment, a method was proposed to reduce the cost of maintaining the v-graph by simplifying the vertices and edges of polygons. For small obstacles, their information is still preserved in the graph. Moreover, the paper proposed a path planning method based on a bidirectional breadth-first search combined with the v-graph. By comparing the original algorithm of FAR Planner and the traditional search algorithms A* and D* Lite, ours achieves the optimal path planning in unknown environments, and the speed of the path search algorithm is faster than that of FAR Planner.

**Author Contributions:** Conceptualization, Q.L. and F.X.; methodology, Q.L., F.X., J.Z. and B.X.; software, Q.L., F.X. and J.Z.; validation, Q.L., F.X. and B.X.; formal analysis, F.X., J.Z.; investigation, J.Y.; resources, X.L.; data curation, H.S. and J.Z.; writing—original draft preparation, Q.L. and F.X.; writing—review and editing, Q.L., F.X. and B.X.; visualization, H.S. and J.Z.; supervision, J.Y.; project administration, F.X. and X.L.; funding acquisition, J.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 41974033 and 61803208), the Scientific and technological achievements transformation project of Jiangsu Province (BA2020004), 2020 Industrial Transformation and Upgrading Project of Industry and Information Technology Department of Jiangsu Province, Postgraduate Research & Practice Innovation Program of Jiangsu Province.

**Data Availability Statement:** The CMU simulation environment can be acquired through https://www.cmu-exploration.com. The Matterport3D simulation environment can be acquired through https://niessner.github.io/Matterport. The original framework used in the paper can be acquired through https://github.com/MichaelFYang/far_planner (all accessed on 14 June 2022).

**Acknowledgments:** The authors would like to sincerely thank all the funders.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| FAR Planner | Fast, Attemptable Route Planner using Dynamic Visibility Update [19] |
| SLAM | Simultaneous Localization And Mapping |
| IMU | Inertial Measurement Unit |
| GPU | Graphics Processing Unit |

## References

1. Hess, W.; Kohler, D.; Rapp, H.; Andor, D. Real-time loop closure in 2D LIDAR SLAM. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 1271–1278. [CrossRef]
2. Grisetti, G.; Kümmerle, R.; Stachniss, C.; Burgard, W. A Tutorial on Graph-Based SLAM. *IEEE Intell. Transp. Syst. Mag.* **2010**, *2*, 31–43. [CrossRef]
3. Li, T.; Pei, L.; Xiang, Y.; Wu, Q.; Xia, S.; Tao, L.; Guan, X.; Yu, W. P3-LOAM: PPP/LiDAR Loosely Coupled SLAM With Accurate Covariance Estimation and Robust RAIM in Urban Canyon Environment. *IEEE Sens. J.* **2021**, *21*, 6660–6671. [CrossRef]
4. Shan, T.; Englot, B. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 4758–4765.
5. Li, J.; Pei, L.; Zou, D.; Xia, S.; Wu, Q.; Li, T.; Sun, Z.; Yu, W. Attention-SLAM: A Visual Monocular SLAM Learning From Human Gaze. *IEEE Sens. J.* **2021**, *21*, 6408–6420. [CrossRef]
6. Mur-Artal, R.; Montiel, J.M.M.; Tardós, J.D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Trans. Robot.* **2015**, *31*, 1147–1163. [CrossRef]
7. Mur-Artal, R.; Tardós, J.D. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Trans. Robot.* **2017**, *33*, 1255–1262. [CrossRef]
8. Qin, T.; Li, P.; Shen, S. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Trans. Robot.* **2018**, *34*, 1004–1020. [CrossRef]
9. Chan, S.H.; Wu, P.T.; Fu, L.C. Robust 2D Indoor Localization Through Laser SLAM and Visual SLAM Fusion. In Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 7–10 October 2018; pp. 1263–1268. [CrossRef]
10. Debeunne, C.; Vivet, D. A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping. *Sensors* **2020**, *20*, 2068. [CrossRef]
11. Nguyen, T.M.; Cao, M.; Yuan, S.; Lyu, Y.; Nguyen, T.H.; Xie, L. VIRAL-Fusion: A Visual-Inertial-Ranging-Lidar Sensor Fusion Approach. *IEEE Trans. Robot.* **2022**, *38*, 958–977. [CrossRef]
12. Thrun, S. Learning Occupancy Grid Maps with Forward Sensor Models. *Auton. Robot.* **2003**, *15*, 111–127. [CrossRef]
13. Lozano-Pérez, T.; Wesley, M.A. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM* **1979**, *22*, 560–570. [CrossRef]
14. Kitzinger, J.; Moret, B. The Visibility Graph among Polygonal Obstacles: A Comparison of Algorithms. Ph.D. Thesis, University of New Mexico, Albuquerque, NM, USA, 2003.
15. Alt, H.; Welzl, E. Visibility graphs and obstacle-avoiding shortest paths. *Z. Für Oper. Res.* **1988**, *32*, 145–164. [CrossRef]
16. Shen, X.; Edelsbrunner, H. A tight lower bound on the size of visibility graphs. *Inf. Process. Lett.* **1987**, *26*, 61–64. [CrossRef]
17. Sridharan, K.; Priya, T.K. An Efficient Algorithm to Construct Reduced Visibility Graph and Its FPGA Implementation. In Proceedings of the VLSI Design, International Conference, Mumbai, India, 9 January 2004; IEEE Computer Society: Los Alamitos, CA, USA, 2004; p. 1057. [CrossRef]
18. Wu, G.; Atilla, I.; Tahsin, T.; Terziev, M.; Wang, L. Long-voyage route planning method based on multi-scale visibility graph for autonomous ships. *Ocean Eng.* **2021**, *219*, 108242. [CrossRef]
19. Yang, F.; Cao, C.; Zhu, H.; Oh, J.; Zhang, J. FAR Planner: Fast, Attemptable Route Planner using Dynamic Visibility Update. *arXiv* **2021**, arXiv:2110.09460.

20. Pradhan, S.; Mandava, R.K.; Vundavilli, P.R. Development of path planning algorithm for biped robot using combined multi-point RRT and visibility graph. *Int. J. Inf. Technol.* **2021**, *13*, 1513–1519. [CrossRef]
21. D'Amato, E.; Nardi, V.A.; Notaro, I.; Scordamaglia, V. A Visibility Graph approach for path planning and real-time collision avoidance on maritime unmanned systems. In Proceedings of the 2021 International Workshop on Metrology for the Sea; Learning to Measure Sea Health Parameters (MetroSea), Reggio Calabria, Italy, 4–6 October 2021; pp. 400–405. [CrossRef]
22. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
23. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
24. Stentz, A. Optimal and efficient path planning for partially-known environments. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, USA, 8–13 May 1994; Volume 4, pp. 3310–3317. [CrossRef]
25. Koenig, S.; Likhachev, M. Fast replanning for navigation in unknown terrain. *IEEE Trans. Robot.* **2005**, *21*, 354–363. [CrossRef]
26. XiangRong, T.; Yukun, Z.; XinXin, J. Improved A-star algorithm for robot path planning in static environment. *J. Phys. Conf. Ser.* **2021**, *1792*, 012067. [CrossRef]
27. Zhang, L.; Li, Y. Mobile Robot Path Planning Algorithm Based on Improved A Star. *J. Phys. Conf. Ser.* **2021**, *1848*, 012013. [CrossRef]
28. LaValle, S.M.; Kuffner, J.J.; Donald, B. Rapidly-exploring random trees: Progress and prospects. *Algorithmic Comput. Robot. New Dir.* **2001**, *5*, 293–308.
29. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
30. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, 14–18 September 2014; pp. 2997–3004.
31. Kuffner, J.J.; LaValle, S.M. RRT-connect: An efficient approach to single-query path planning. In Proceedings of the 2000 ICRA, Millennium Conference, IEEE International Conference on Robotics and Automation, Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, USA, 24–28 April 2000; Volume 2, pp. 995–1001.
32. Tu, J.; Yang, S. Genetic algorithm based path planning for a mobile robot. In Proceedings of the 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), Taipei, Taiwan, 14–19 September 2003; Volume 1, pp. 1221–1226. [CrossRef]
33. Chen, J.; Ye, F.; Li, Y. Travelling salesman problem for UAV path planning with two parallel optimization algorithms. In Proceedings of the 2017 Progress in Electromagnetics Research Symposium—Fall (PIERS - FALL), Singapore, 19–22 November 2017; pp. 832–837. [CrossRef]
34. Pan, Y.; Yang, Y.; Li, W. A Deep Learning Trained by Genetic Algorithm to Improve the Efficiency of Path Planning for Data Collection With Multi-UAV. *IEEE Access* **2021**, *9*, 7994–8005. [CrossRef]
35. Hao, K.; Zhao, J.; Wang, B.; Liu, Y.; Wang, C. The application of an adaptive genetic algorithm based on collision detection in path planning of mobile robots. *Comput. Intell. Neurosci.* **2021**, *2021*, 5536574. [CrossRef] [PubMed]
36. Rahmaniar, W.; Rakhmania, A.E. Mobile Robot Path Planning in a Trajectory with Multiple Obstacles Using Genetic Algorithms. *J. Robot. Control (JRC)* **2022**, *3*, 1–7. [CrossRef]
37. Zhang, T.W.; Xu, G.H.; Zhan, X.S.; Han, T. A new hybrid algorithm for path planning of mobile robot. *J. Supercomput.* **2022**, *78*, 4158–4181. [CrossRef]
38. Richter, C.; Vega-Brown, W.; Roy, N., Bayesian Learning for Safe High-Speed Navigation in Unknown Environments. In *Robotics Research: Volume 2*; Bicchi, A., Burgard, W., Eds.; Springer: Cham, Switzerland, 2018; pp. 325–341. [CrossRef]
39. Zeng, J.; Ju, R.; Qin, L.; Hu, Y.; Yin, Q.; Hu, C. Navigation in unknown dynamic environments based on deep reinforcement learning. *Sensors* **2019**, *19*, 3837. [CrossRef]
40. Guo, X.; Fang, Y. Learning to Navigate in Unknown Environments Based on GMRP-N. In Proceedings of the 2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), Suzhou, China, 29 July–2 August 2019; pp. 1453–1458.
41. Lin, G.; Zhu, L.; Li, J.; Zou, X.; Tang, Y. Collision-free path planning for a guava-harvesting robot based on recurrent deep reinforcement learning. *Comput. Electron. Agric.* **2021**, *188*, 106350. [CrossRef]
42. Tutsoy, O.; Brown, M. Reinforcement learning analysis for a minimum time balance problem. *Trans. Inst. Meas. Control* **2016**, *38*, 1186–1200. [CrossRef]
43. Tutsoy, O.; Barkana, D.E.; Balikci, K. A novel exploration-exploitation-based adaptive law for intelligent model-free control approaches. *IEEE Trans. Cybern.* **2021**, 1–9. [CrossRef]
44. Oommen, B.; Iyengar, S.; Rao, N.; Kashyap, R. Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case. *IEEE J. Robot. Autom.* **1987**, *3*, 672–681. [CrossRef]
45. Wooden, D.; Egerstedt, M. Oriented visibility graphs: Low-complexity planning in real-time environments. In Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, FL, USA, 15–19 May 2006; pp. 2354–2359.
46. Rao, N.S. Robot navigation in unknown generalized polygonal terrains using vision sensors. *IEEE Trans. Syst. Man Cybern.* **1995**, *25*, 947–962. [CrossRef]

47. Cao, C.; Zhu, H.; Yang, F.; Xia, Y.; Choset, H.; Oh, J.; Zhang, J. Autonomous Exploration Development Environment and the Planning Algorithms. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022.
48. Chang, A.; Dai, A.; Funkhouser, T.; Halber, M.; Niessner, M.; Savva, M.; Song, S.; Zeng, A.; Zhang, Y. Matterport3D: Learning from RGB-D Data in Indoor Environments. *arXiv* **2017**, arXiv:1709.06158.
49. Wei, Y.; Xiao, H.; Shi, H.; Jie, Z.; Feng, J.; Huang, T.S. Revisiting dilated convolution: A simple approach for weakly-and semi-supervised semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7268–7277.
50. Suzuki, S. Topological structural analysis of digitized binary images by border following. *Comput. Vision Graph. Image Process.* **1985**, *30*, 32–46. [CrossRef]
51. Teh, C.H.; Chin, R.T. On the detection of dominant points on digital curves. *IEEE Trans. Pattern Anal. Mach. Intell.* **1989**, *11*, 859–872. [CrossRef]
52. Lee, D.T. Proximity and Reachability in the Plane. Ph.D. Thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1978.
53. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3, p. 5.
54. Will, P.; Pennington, K. Grid coding: A preprocessing technique for robot and machine vision. *Artif. Intell.* **1971**, *2*, 319–329. [CrossRef]
55. Agarwal, P.K.; Arge, L.; Danner, A., From Point Cloud to Grid DEM: A Scalable Approach. In *Progress in Spatial Data Handling: 12th International Symposium on Spatial Data Handling*; Riedl, A., Kainz, W., Elmes, G.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 771–788. [CrossRef]
56. Birk, A.; Carpin, S. Merging Occupancy Grid Maps From Multiple Robots. *Proc. IEEE* **2006**, *94*, 1384–1397. [CrossRef]
57. Meyer-Delius, D.; Beinhofer, M.; Burgard, W. Occupancy grid models for robot mapping in changing environments. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, ON, Canada, 22–26 July 2012.
58. Lau, B.; Sprunk, C.; Burgard, W. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robot. Auton. Syst.* **2013**, *61*, 1116–1130. [CrossRef]
59. Kim, B.; Kang, C.M.; Kim, J.; Lee, S.H.; Chung, C.C.; Choi, J.W. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 399–404. [CrossRef]
60. Elfes, A. Using occupancy grids for mobile robot perception and navigation. *Computer* **1989**, *22*, 46–57. [CrossRef]
61. Homm, F.; Kaempchen, N.; Ota, J.; Burschka, D. Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection. In Proceedings of the 2010 IEEE Intelligent Vehicles Symposium, La Jolla, CA, USA, 21–24 June 2010; pp. 1006–1013. [CrossRef]
62. Li, H.; Tsukada, M.; Nashashibi, F.; Parent, M. Multivehicle Cooperative Local Mapping: A Methodology Based on Occupancy Grid Map Merging. *IEEE Trans. Intell. Transp. Syst.* **2014**, *15*, 2089–2100. [CrossRef]
63. Li, Y.; Ruichek, Y. Occupancy Grid Mapping in Urban Environments from a Moving On-Board Stereo-Vision System. *Sensors* **2014**, *14*, 10454–10478. [CrossRef]
64. Kneidl, A.; Borrmann, A.; Hartmann, D. Generation and use of sparse navigation graphs for microscopic pedestrian simulation models. *Adv. Eng. Inform.* **2012**, *26*, 669–680. [CrossRef]
65. Welzl, E. Constructing the visibility graph for n-line segments in O(n2) time. *Inf. Process. Lett.* **1985**, *20*, 167–171. [CrossRef]
66. Majeed, A.; Lee, S. A Fast Global Flight Path Planning Algorithm Based on Space Circumscription and Sparse Visibility Graph for Unmanned Aerial Vehicle. *Electronics* **2018**, *7*, 375. [CrossRef]
67. Oleynikova, H.; Taylor, Z.; Siegwart, R.; Nieto, J. Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 1–9. [CrossRef]
68. Himmel, A.S.; Hoffmann, C.; Kunz, P.; Froese, V.; Sorge, M. Computational complexity aspects of point visibility graphs. *Discret. Appl. Math.* **2019**, *254*, 283–290. [CrossRef]
69. Nguyet, T.T.N.; Hoai, T.V.; Thi, N.A. Some Advanced Techniques in Reducing Time for Path Planning Based on Visibility Graph. In Proceedings of the 2011 Third International Conference on Knowledge and Systems Engineering, Hanoi, Vietnam, 14–17 October 2011; pp. 190–194. [CrossRef]
70. Ben-Moshe, B.; Hall-Holt, O.; Katz, M.J.; Mitchell, J.S.B. Computing the Visibility Graph of Points within a Polygon. In Proceedings of the Twentieth Annual Symposium on Computational Geometry, Brooklyn, NY, USA, 8–11 June 2004; Association for Computing Machinery: New York, NY, USA, 2004; p. 27–35. [CrossRef]