mathematics

Article

Autonomous Navigation System of Indoor Mobile Robots Using 2D Lidar

Jian Sun [1], Jie Zhao [2], Xiaoyang Hu [3,*], Hongwei Gao [2,4,*] and Jiahui Yu [5,*]

1 School of Graduate, Shenyang Ligong University, Shenyang 110158, China
2 School of Automation and Electrical Engineering, Shenyang Ligong University, Shenyang 110158, China
3 School of Equipment Engineering, Shenyang Ligong University, Shenyang 110158, China
4 China State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110017, China
5 Department of Biomedical Engineering, Zhejiang University, Hangzhou 310058, China
* Correspondence: xiaoyang_hu@163.com (X.H.); ghw1978@sohu.com (H.G.); jiahui.yu@zju.edu.cn (J.Y.)

Abstract: Significant developments have been made in the navigation of autonomous mobile robots within indoor environments; however, there still remain challenges in the face of poor map construction accuracy and suboptimal path planning, which limit the practical applications of such robots. To solve these challenges, an enhanced Rao Blackwell Particle Filter (RBPF-SLAM) algorithm, called Lidar-based RBPF-SLAM (LRBPF-SLAM), is proposed. In LRBPF, the adjacent bit poses difference data from the 2D Lidar sensor which is used to replace the odometer data in the proposed distribution function, overcoming the vulnerability of the proposed distribution function to environmental disturbances, and thus enabling more accurate pose estimation of the robot. Additionally, a probabilistic guided search-based path planning algorithm, gravitation bidirectional rapidly exploring random tree (GBI-RRT), is also proposed, which incorporates a target bias sampling to efficiently guide nodes toward the goal and reduce ineffective searches. Finally, to further improve the efficiency of navigation, a path reorganization strategy aiming at eliminating low-quality nodes and improving the path curvature of the path is proposed. To validate the effectiveness of the proposed method, the improved algorithm is integrated into a mobile robot based on a ROS system and evaluated in simulations and field experiments. The results show that LRBPF-SLAM and GBI-RRT perform superior to the existing algorithms in various indoor environments.

Keywords: mobile robots; path planning; RBPF-SLAM; Lidar sensor; ROS system

MSC: 70B15

check for updates

Citation: Sun, J.; Zhao, J.; Hu, X.; Gao, H.; Yu, J. Autonomous Navigation System of Indoor Mobile Robots Using 2D Lidar. *Mathematics* **2023**, *11*, 1455. https://doi.org/10.3390/math11061455

Academic Editor: Daniel-Ioan Curiac

Received: 19 February 2023
Revised: 10 March 2023
Accepted: 11 March 2023
Published: 17 March 2023

1. Introduction

In recent years, the widespread use of mobile robots for a variety of applications, such as rescue operations [1], household cleaning [2], and food service [3], has been facilitated by their high stability and affordability. To meet the needs of these applications, mobile robots require acquiring poses from Lidar sensors and building maps for environmental awareness, and then using path planning algorithms to determine travel trajectories. Mobile robots typically have three main functions: map building, positional estimation, and path planning. The main task of SLAM (Simultaneous Localization and Mapping) is to obtain real-time data from the robot's sensors in an unknown environment and construct a map, while also completing autonomous localization [4]. Moreover, after the localization and map building is completed, it is not feasible to manually set the walking path, which limits the robot's autonomy. Thus, we use SLAM technology to provide environmental information for path planning, helping mobile robots autonomously perform complex navigation tasks.

SLAM plays a crucial role in the field of mobile robotics, serving as a key precondition for the autonomous behavior and intelligence of mobile robots. The solution to SLAM can be mainly divided into two categories: the graph optimization [5–7] and the probabilistic estimation method [8]. The classical graph optimization algorithm is Karto SLAM [9], which solves the optimization problem through graph representation. Karto consists of three parts: front-end graph matching, back-end graph optimization, and loop closure detection. The loop closure detection reduces the drift of the map and ensures global consistency by recognizing loops and accordingly optimizing. Graph Optimization SLAM has the advantage of slow error accumulation and high robustness, but its disadvantages include a slow loop closure detection speed and the possibility of false loop closures. In addition to graph optimization, probabilistic estimation methods are also utilized to solve SLAM problems. Extended Kalman Filters (EKFs) are commonly applied by linearizing the system through a first-order Taylor expansion to address weakly nonlinear conditions [10]. However, EKFs can result in an erroneous pose and map estimates, especially under conditions of linearization error accumulation. On the other hand, Particle Filters (PF) can effectively handle nonlinear non-Gaussian probability estimation [11], but their complexity significantly increases as the spatial dimensionality increases. The RBPF SLAM [12] is a particle filter-based solution to SLAM problems that improves runtime by utilizing an accurate proposal distribution and selective resampling strategy [13], reducing the number of required particles. GMapping [14] is a probabilistic estimation algorithm that inputs odometry information and Lidar sensor measurements, producing the robot's pose and occupancy grid maps. The prediction of the proposed distribution function in RBPF-SLAM is based on odometry data, making it difficult to incorporate additional information in the Monte Carlo localization framework. Furthermore, the instability of the proposed distribution function, based on odometry, makes it challenging to eliminate motion uncertainty in large environments and long-term tasks. To address these challenges, some studies have proposed FastSlam [15], a combination of RBPF and EKF, to improve particle distribution.

Path planning is a critical component of mobile robot navigation [16], and its goal is to determine a feasible and optimal path for the robot to travel from a starting position to a goal position while avoiding obstacles in its environment. Path planning algorithms are mainly divided into graph-based search algorithms and sampling-based algorithms. Graph-based search algorithms use a graph representation of the search space to plan paths for mobile robots. These algorithms perform a search of the graph to find the optimal path from the starting position of the robot to the goal position while avoiding obstacles. The most common graph-based search algorithms are A* [17], Dijkstra [18], and D* [19]. The A* algorithm is a heuristic search algorithm that finds the shortest path from the starting position to the goal position by using a heuristic function to evaluate the next state. However, the A* algorithm requires additional storage space to maintain a set of open points, which can result in memory overhead. The Dijkstra algorithm is a classic shortest-path algorithm that finds the shortest path between any two points in a graph. The algorithm works by gradually relaxing the edge weights and updating the distance estimates of vertices. However, the time complexity of the Dijkstra algorithm is $O(n^2)$, where n is the number of vertices in the graph, and when the graph is large, the efficiency of the algorithm can be severely affected. The D* algorithm combines the advantages of the A* algorithm and the Dijkstra algorithm. The algorithm is capable of re-planning in real time according to the changing environment, which makes it well suited for dynamic and uncertain environments.

The sampling-based algorithm is an algorithm that finds the optimal path by random sampling method. This algorithm finds the optimal path by randomly selecting a point in space as the starting or ending point, and then continues expanding the nodes in space when the expansion reaches the target point. The rapidly exploring random tree (RRT) [20] algorithm is a popular and efficient algorithm in the field of sampling-based path planning. The RRT algorithm uses a random sampling method to explore the search space, so it can effectively avoid local optimum problems [21]. However, RRT requires sampling and searching the entire graph, and many redundant random nodes are generated near each node, increasing the corresponding search time and leading to slower convergence. One of the main advantages of the bidirectional rapidly exploring random tree (Bi-RRT) [22,23] algorithm is its efficiency compared to RRT algorithms. Since the trees are simultaneously expanded in both directions, the search space can be reduced by half, which can significantly reduce the search time. However, the Bi-RRT algorithm may not be able to find the optimal solution in complex environments with high-dimensional state spaces. This is because the algorithm relies on the random sampling method, which may not effectively cover all parts of the state space and may not promptly find the optimal solution. Many scholars have proposed improved methods based on the Bi-RRT algorithm; Xu et al. presented a post-processing fusion algorithm [24], which combines PRM and P-Bi-RRT algorithms. Compared to RRT, Bi-RRT, and P-Bi-RRT algorithms, this algorithm has shown improved results in terms of planning time, path length, and the number of path nodes. Yi et al. proposed the 1-0Bg-RRT algorithm [25], which uses a biased probability of 1 and 0 changes to construct a tree, resulting in shorter computation time and paths compared to traditional RRT algorithms. Jiankun Wang et al. presented a kino dynamically constrained Bi-RRT with efficient branch pruning algorithm [26]. This algorithm extends the Bi-RRT method by incorporating kino dynamic constraints, leading to improved performance. Grothe et al. presented the Space-Time RRT (ST-RRT*) algorithm [27]; ST-RRT* can effectively handle unbounded time-space and optimize arrival time in environments with moving obstacles on known trajectories. Huanjie Zhao. et al. proposed an Improved Bi-RRT algorithm based on Gaussian sampling [28]. This algorithm introduces heuristic search ideas based on bidirectional search, sample points with a Gaussian distribution constrained with a certain probability near the start, and goal points to reduce the blind search and improve search efficiency. Guojun Ma et al. presented a new algorithm for path planning named Probabilistic Smoothing Bi-RRT (PSBi-RRT) [29]. The proposed algorithm utilizes a θ-cut mechanism to optimize the path toward the global optimal solution, reducing the possibility of getting stuck in local optima. In comparison to the traditional Bi-RRT algorithm, PSBi-RRT exhibits a significant reduction in runtime with improved performance.

Based on the above analysis, we propose improvements to the simultaneous SLAM algorithm and the path planning algorithm. The distribution function in RBPF is susceptible to external factors such as robot tire skidding, resulting in suboptimal map construction. In contrast, Lidar navigation is highly stable because it is highly resistant to environmental noise. For this reason, we propose the LRBPF-SLAM algorithm, where the odometer data in the distribution function are replaced with the bit pose differences of adjacent moments from the 2D Lidar to improve the stability and accuracy of map construction. In addition, the Bi-RRT algorithm ignores the redundant computation due to the selection of random nodes. We improve the Bi-RRT algorithm by using target bias sampling to reduce invalid searches and combining the path reorganization strategy to minimize redundant path points and generate smooth trajectories. In summary, the contributions of this paper are as follows:

1. We embed the proposed algorithm into the ROS system [30] to verify the effectiveness of the algorithm;
2. In order to improve the stability and accuracy of the SLAM system, an algorithm called LRBPF-SLAM is proposed. In this algorithm, the odometer data in the distribution function is replaced by the 2D Lidar adjacent moment bit pose difference;
3. The GBI-RRT algorithm is proposed, which employs target bias sampling to reduce the negative impact of random sampling on path quality, and then optimizes the initial paths through a path reorganization strategy to eliminate redundant paths;
4. Extensive simulations were conducted to evaluate the improved algorithms, and the proposed algorithms were also ported to a mobile robot for real scenario experiments. The results of these experiments demonstrate that the proposed method exhibits

excellent performance compared to other algorithms in both simulation and real scenarios.

2. Robot Components and System Framework

The system we use is an advanced mobile robot navigation system equipped with sensors for environmental perception and data measurement. The main hardware used in this system is the NVIDIA Jetson Nano, which has sufficient processing capabilities to perform task planning. Additionally, it is equipped with an OpenCRP controller based on the STM32F4 core and an MPU6050 Inertial Measurement Unit (IMU) sensor that can be updated through ISP serial and implements closed-loop control for four DC motors. The robot is also equipped with the SICK A1 TK edition Lidar, with a range of 12 m and a measurement frequency of 8000 times per second, as well as an encoder that converts analog signals into electrical signals to obtain distance and angle data. The size of the mobile robot for navigation is 28 cm $\times$ 12 cm $\times$ 12 cm and weighs 2.3 kg, rated power is 60 W, and the linear velocity and acceleration are 1.2 m/s and 0.5 m/s, respectively. The physical structure of the robot is shown in Figure 1.

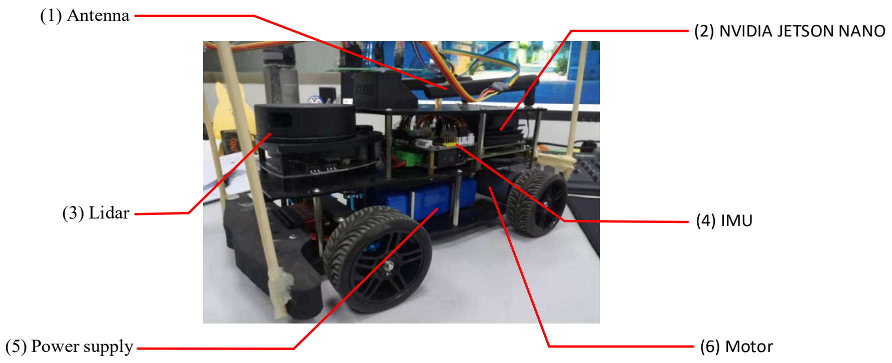


Figure 1. The physical structure of the robot. (1) The "Antenna" is utilized for transmission of communication protocols. (2) The "NVIDIA JETSON NANO" is utilized for receiving commands from the PC and running algorithms. (3) The "Lidar" is utilized for sensing the surrounding environment. (4) The "IMU" is utilized for acquiring the current attitude angles. (5) The "Power supply" sustains the operation of the moving robot by providing electrical energy. (6) The "Motor" is utilized for driving the movement of the robot.

The system control structure of the robot is shown in Figure 2.

1. The PC Module: The PC terminal uses a laptop and connects to the host computer on the same LAN via SSH (Secure Shell). Commands can be directly sent from the PC to the mobile robot host computer to achieve SLAM and navigation functions;
2. The Decision Module: The decision module is the host computer of the robot, namely NVIDIA Jetson Nano, which has an SSH tool installed with the ROS system to receive commands from the PC and run algorithms. It receives Lidar data through a USB interface, communicates I/O with the lower computer, and acquires sensor data connected to the lower computer;
3. The Execution Module: The execution module is a controller with STM32F4 as its core, which receives commands from the decision-making module, acquires data from the IMU and encoders, and controls motor drive operations;
4. The Sensor Module: The sensor module includes 2D Lidar for detecting the surrounding environment, IMU for estimating the motion posture of the robot, and encoders for estimating the robot's motion distance and rotation angle;
5. The Power Module: The power voltage is 12 V with a total capacity of 1200 mAh. The power expansion board can expand the 12 V power and 5 V output to facilitate the expansion of robot functions;
6. The Motor Drive Module: The motor drive module is responsible for controlling the movement of the robot, receiving control commands, and driving the motor

through current control. It includes the drive circuit, current sensor, and control circuit, ensuring the precise and stable movement of the robot.

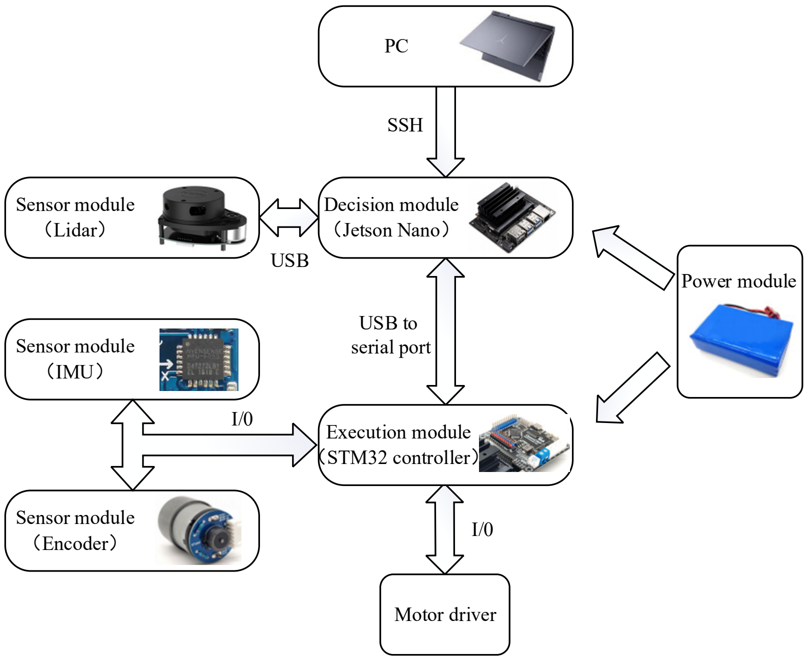


Figure 2. The system control structure of robot.

In the design of a robot navigation system, multiple critical steps are covered, including data conversion, SLAM mapping, and path planning. We designed a comprehensive robot navigation system framework to realize the navigation capability. This framework implements distributed communication through the ROS system, thereby enabling the collaboration between SLAM mapping and navigation path planning, and allowing for node publication and subscription, further improving the efficiency and reliability of the system. The flow of the robot navigation system framework is illustrated in Figure 3. The robot navigation can be divided into the following four steps:

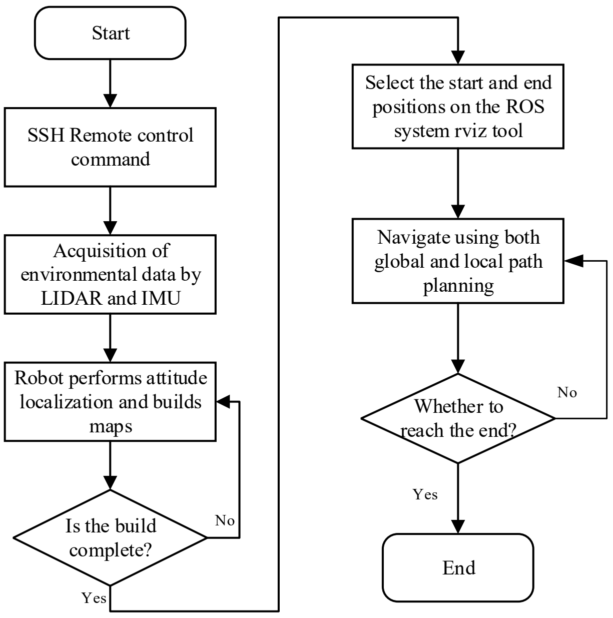

Start
SSH Remote control command
Acquisition of environmental data by LIDAR and IMU
Robot performs attitude localization and builds maps
Is the build complete?
No
Yes
Select the start and end positions on the ROS system rviz tool
Navigate using both global and local path planning
Whether to reach the end?
No
Yes
End

Figure 3. Framework flow of robot navigation system.

1. Install the Ubuntu operating system and ROS system on the robot and PC side, use the SSH remote control tool to realize the connection between the PC and the robot, and control the robot through PC input commands;
2. After receiving the PC command, the robot locates and builds a map using the data from the Lidar, and when the mapping is completed, the map is saved in the robot;
3. After starting the navigation command, the starting point and the end point are selected on the Rviz (a 3D visualization tool) visualization tool of ROS, and the robot autonomously plans the navigation path using the data from Lidar. The global path planning realizes safe and reliable path planning, and local path planning realizes real-time obstacle avoidance;
4. When the robot arrives at the destination, the navigation ends. If it does not reach the destination, it continues to navigate using the data from Lidar until it reaches the destination.

3. Algorithm Improvement

3.1. LRBPF-SLAM Algorithm

To better understand the proposed LRBPF-SLAM, we briefly review the basic principles of the RBPF. The RBPF-SLAM is an improved version of the particle filter. RBPF is a technique for reducing computational costs by lowering the dimensionality of the state space through the use of the chain rule. This is achieved by factoring the joint distribution of the variables into conditional distributions, which can be separately updated, resulting in improved computational efficiency. The problem of SLAM for RBPF involves the estimation of the posterior probability and posterior probability as shown in Equation (1).

$$p(m, x_{1:t}|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t-1})p(m|x_{1:t}, z_{1:t}) \tag{1}$$

Where $p(m, x_{1:t}|z_{1:t}, u_{1:t})$ is the posterior probability, the estimated joint posterior probability $p(x_{1:t}|z_{1:t}, u_{1:t})$ represents the distribution of the motion trajectory of a mobile robot, $p(m|x_{1:t}, z_{1:t})$ is the posterior map generated by particles using the occupancy grid mapping algorithm to create a two-dimensional planar map of the environment. m represents the grid map of the environment, $x_{1:t}$ denotes the motion trajectory of the mobile robot, $z_{1:t}$ is the sensor observations from 1 to t moments, $u_{1:t}$ is the odometry measurements in the odometer. The estimation of the robot's true pose can be achieved using the $z_{1:t}$ and $u_{1:t}$ parameters. The specific steps of RBPF are shown below:

1. Sampling: The particle at the previous moment $x^i_{1:t-1}$ is sampled from the distribution function to acquire new particles $x^i_{1:t}$. The distribution function obtained by the sensor is often termed the proposed distribution:

$$\pi(x^i_t \Big| x^i_{1:t-1}, z_{1:t}, u_{1:t-1}) \tag{2}$$

2. Importance weighting: Each particle x^i_t is assigned a weight ω^i_t, which is computed as the ratio of the posterior distribution to the proposal distribution (based on the probabilistic odometry motion model). The higher the weight, the more the particle's pose matches the true value. The importance weighting can be defined using Formula (3).

$$\omega^i_t = \frac{p(x^i_{1:t}|z_{1:t}, u_{1:t-1})}{\pi(x^i_{1:t}|z_{1:t}, u_{1:t-1})} \tag{3}$$

3. Resampling: Particles with smaller weights are discarded and replaced by resampled particles, but the total number of particles remains constant.
4. Map updating: Each particle's map is updated using the optimized pose represented by the particle and the current observations.

RBPF can effectively reduce the dimensionality of the state space and improve the particle quality. However, the proposed distribution based on odometry may suffer from increasing errors over time. As the Lidar signal has a single-peak characteristic and a small variance coefficient, it is more suitable to use it as the input to the proposed distribution function. To improve the accuracy of the proposed distribution, we augment the original odometry data by adding the position differences derived from the 2D Lidar data at adjacent time steps. The RBPF algorithm usually uses odometer data as the proposed distribution function:

$$\pi\left(x_t^i \middle| x_{1:t-1}^i, z_{1:t}, u_{1:t-1}\right) = p\left(x_t^i \middle| x_{t-1}^i, u_{t-1}\right) \tag{4}$$

IMU is a sensor used for attitude estimation. Typically consisting of an accelerometer, gyroscope, and magnetometer, it measures the acceleration, angular velocity, and magnetic field strength of an object in three axes. In attitude estimation, the IMU plays a key role by providing real-time attitude information that allows us to track the position, orientation, and motion of the object. However, the odometer data from the IMU can be affected by robot vibration, drift, and sliding. Lidar can provide higher spatial resolution and accuracy to ensure the accuracy of attitude estimation. LRBPF uses the Lidar positional difference as a distribution function, as shown in Equations (5) and (6).

$$p\left(x_t^i \middle| x_{t-1}^i, z_t\right) = x_{t-1}^i + \mathrm{h}_t(z_t, z_{t-1}) \tag{5}$$

$$\mathrm{h}_t = z_t - z_{t-1} \tag{6}$$

where h_t is the Lidar attitude difference between adjacent moments.

3.2. GBI-RRT Algorithm

To gain a better understanding of the proposed algorithm, it is necessary to first review the RRT and the Bi-RRT algorithms. Figure 4 shows the planning process of the RRT algorithm, where q_{init} and q_{goal} represent the start and target nodes of the random tree, q_{rand} is the random node generated by each sampling point, and q_{near} is the closest node to q_{rand} on the tree, q_{new} is a new node obtained after collision detection, which is obtained by growing from q_{near} to q_{rand} with step size ε. The RRT principle diagram is shown in Figure 4.

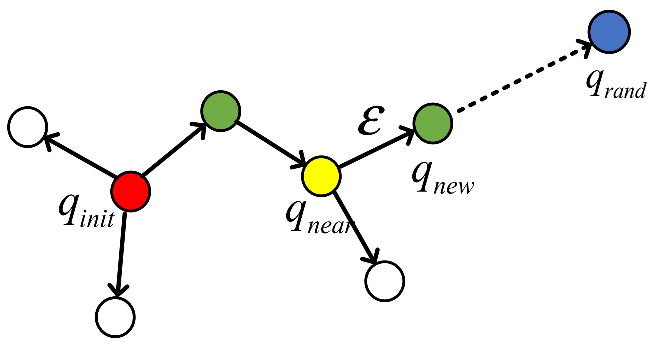


Figure 4. RRT principle diagram.

The RRT algorithm begins by selecting the q_{init} as the root node of the random tree growth. Next, q_{rand} is generated within the safe space. Then, the algorithm searches for the node q_{near} that is closest to q_{rand}, with q_{near} initially set to q_{init}. Starting from q_{near}, the random tree moves ε steps in the direction of q_{rand} to obtain a new node q_{new}. This process is repeated until the Euclidean distance between q_{init} and q_{goal} is less than a predetermined threshold, at which point the search is terminated. The resulting path is the extended tree path from the initial node q_{init} to the target node q_{goal}. The expansion rule for the new node in the RRT algorithm is expressed by Equation (7).

$$q_{new} = q_{near} + \varepsilon \frac{q_{rand} - q_{near}}{\|q_{rand} - q_{near}\|} \tag{7}$$

where $q_{rand} - q_{near}$ represents the normalization of two vectors, and $\|q_{rand} - q_{near}\|$ represents the Euclidean distance between two points. When the target node q_{goal} is added to the random tree or the number of iterations exceeds the specified threshold of iterations, the path planning will end with the corresponding result.

Although the RRT algorithm is better than the traditional algorithm in complex environments, its one-way search approach implies that it takes longer to reach the endpoint. To address this issue, the Bi-RRT algorithm was developed, which enables a two-way search. The Bi-RRT algorithm is shown in Figure 5:

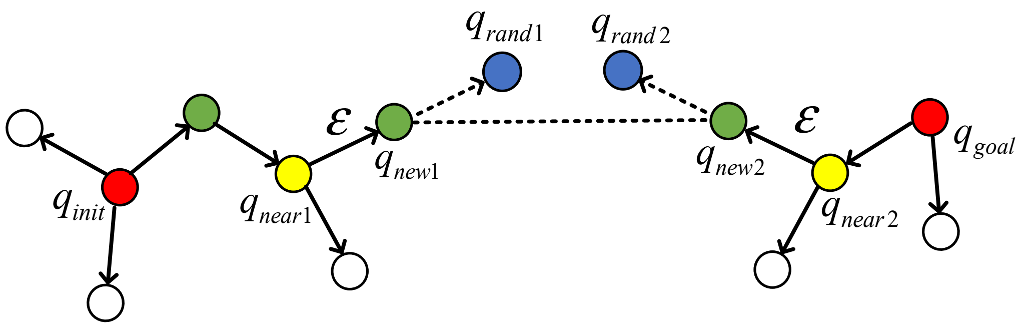


Figure 5. Bi-RRT principle diagram.

The Bi-RRT algorithm constructs two random trees T_1 and T_2 in the environment state space, using the same node generation method as the basic RRT algorithm. T_1 has the root node as the initial node, while T_2 has the target point as the initial node. The Bi-RRT algorithm process is shown in Algorithm 1.

Algorithm 1 presents the fundamental Bi-RRT algorithm. First, the algorithm initializes the random tree T_1 using q_{init} and then initializes the random tree T_2 using q_{goal}. To extend the random tree outward, the $Sample()$ function is designed to return a sample point q_{rand}. Then, the $Extend()$ function searches for the nearest node in the random tree and grows toward node q_{rand} in steps ε, generating a new node q_{new}. Subsequently, if q_{new} passes collision detection, it is added to the random tree T. If q_{new} is the same for both random trees, then the loop terminates.

Compared with the RRT algorithm, the Bi-RRT algorithm reduces the search time while retaining the advantages of the RRT algorithm. However, both algorithms have a common drawback: both randomly generate expansion points, resulting in poor search path quality [31]. Based on this, we propose an improved Bi-RRT algorithm to reduce the algorithm's blindness in the node expansion phase by introducing target bias sampling, generating random points with a higher probability towards the target point. Additionally, we propose a path reorganization strategy to address the low-quality generated paths by removing redundant nodes and optimizing the path state.

Algorithm 1: BI- RRT(q_{init}, q_{goal})

```
1 $T_1.add(q_{init}); T_2.add(q_{goal}); i = 0;$
2 while$(i < N)$
3    $q_{rand1}$ = Sample();
4    $q_{rand2}$ = Sample(); $i++$;
5    $q_{new1} = Extend(T_1, q_{rand1})$
6    $q_{new2} = Extend(T_2, q_{rand2})$
7    if $q_{new1} = q_{new2}$ then
8       return Path$(T_1, T_2)$
9    Swap$(T_1, T_2)$
```

1. Target bias sampling

The random sampling process of the Bi-RRT algorithm employs a global random search strategy, which generates a significant number of redundant random points and increases the length of the robot movement path. The path planning process can only be accelerated when the random tree grows toward the target point, so the target point can be considered as the sampling point. However, if the target point is selected as the only sampling point, the generated random tree may become trapped in a dead loop around the obstacles. To address this issue, we propose a target bias sampling that combines random search and target-oriented search. This strategy effectively guides the random tree to grow towards the target with a higher probability while avoiding interference from obstacles.

Figure 6 illustrates the GBI-RRT algorithm, which begins by selecting an initial point q_{init}. During each iteration, the system generates a random number p_{rand}. If p_{rand} is less than the given threshold p_{bias}, the algorithm generates a random point within the safe space $SampleFree()$. Otherwise, the random point is set to the target point coordinates. To implement the target bias sampling, we use Equation (8), which effectively guides the random tree to grow towards the target with a higher probability while avoiding obstacles.

$$q_{rand} = \begin{cases} q_{goal}, \ if \ p_{rand} > p_{bias} \\ q_{SampleFree()} \ , \text{ else} \end{cases} \tag{8}$$

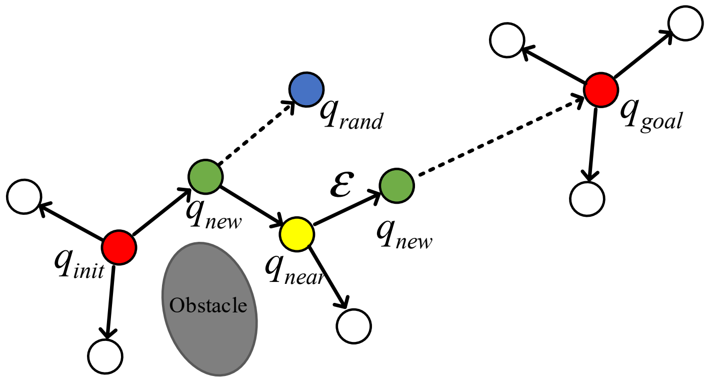


Figure 6. Bi-RRT random tree constructed by adding target bias sampling. The direction of the dashed line represents the random tree growth direction.

In the above Equation (8), P_{bias} represents the target bias threshold, P_{rand} represents that the random sampling probability range is (0, 1), and $q_{SampleFree()}$ represents the random point generated by the safe space.

Once the random node q_{rand} is obtained, we use a target bias sampling to guide the extension of the random tree towards the target point with a growth step of ε. This strategy promotes an explicit expansion direction for the random tree, which preserves the global expansion property of the RRT algorithm and allows the node expansions to spread across the state space. Moreover, the target bias sampling enables the preservation of local node properties on top of the global expansion properties, increasing the likelihood that the random tree will expand towards the target point. However, choosing an appropriate value for the threshold P_{bias} is crucial. A value that is too large can result in an expansion probability towards the target point that is too small to have a significant effect on the expansion speed, while a value that is too small can result in an overly large expansion probability towards the target point that is prone to local minima in environments with many obstacles. After experimental analysis, we set P_{bias} to 0.5.

The random growth function for the random tree to expand towards the target direction is shown in Equation (9).

$$X(n) = \varepsilon \frac{q_{near} - q_{goal}}{\|q_{near} - q_{goal}\|} \tag{9}$$

where ε denotes the step size when expanding towards the target point and $\left\|q_{near} - q_{goal}\right\|$ denotes the Euclidean distance between q_{near} and q_{goal}.

In addition, the random growth function $Y(n)$ for the random tree to randomly expand and avoid obstacles in the safe space is given by:

$$Y(n) = \varepsilon \frac{q_{SampleFree()} - q_{near}}{\|q_{SampleFree()} - q_{near}\|} \tag{10}$$

Therefore, by combining Equations (8)–(10), we can obtain the equation for generating a new node using the target bias sampling as follows:

$$q_{new} = \begin{cases} q_{near} + X(n), \ if \ p_{rand} > p_{bias} \\ q_{near} + Y(n), \ \text{else} \end{cases} \tag{11}$$

At this point, the calculation of the new node q_{new} not only takes into account the influence of the random sampling node q_{rand}, but also the gravitation of the target point q_{goal}. The threshold value P_{bias} plays a crucial role in determining the expansion direction. When the generated random sampling point is close to an obstacle, it may cause the newly generated node to collide with the obstacle, leading to expansion failure and getting stuck in a dead loop. If P_{rand} is larger than P_{bias}, the selected random point P_{rand} satisfies the requirement of expanding towards the target point and enables the system to approach the target point more quickly. On the other hand, when P_{rand} is smaller than P_{bias}, the selected random point q_{rand} no longer satisfies the requirement of expanding directly towards the target point, and random sampling points will be generated for expansion. By doing so, the expanded tree can bypass obstacles and reach the end point more efficiently.

2. Path reorganization strategy

In the Bi-RRT algorithm, the nearest tree node is determined by calculating the Euclidean distance from a random point to a tree node. However, this approach may result in zigzag node paths for the concatenated tree nodes, and such unsmooth paths are not optimal for mobile robot travel because they increase unnecessary steering time [32]. Even with a target bias sampling incorporated, the paths generated by the Bi-RRT algorithm may still contain many redundant nodes. Therefore, path reorganization strategies are needed to optimize the generated paths and obtain higher quality paths.

As shown in Figure 7, in path planning with multiple nodes, the distance through path $q_{init} \to b$ is less than the distance through $q_{init} \to a \to b$; the distance through $b \to d$ is less than the distance through $b \to c \to d$; and the distance through $d \to q_{goal}$ is less than the distance through $d \to e \to f \to q_{goal}$. Therefore, in the final path planning process, the redundant nodes a, c, e, and f can be removed. The node $q_{init} \to b \to d \to q_{goal}$ forms the optimal path, which only has a few key points, and thus improves the smoothness of the path and shortens the travel time of the mobile robot.

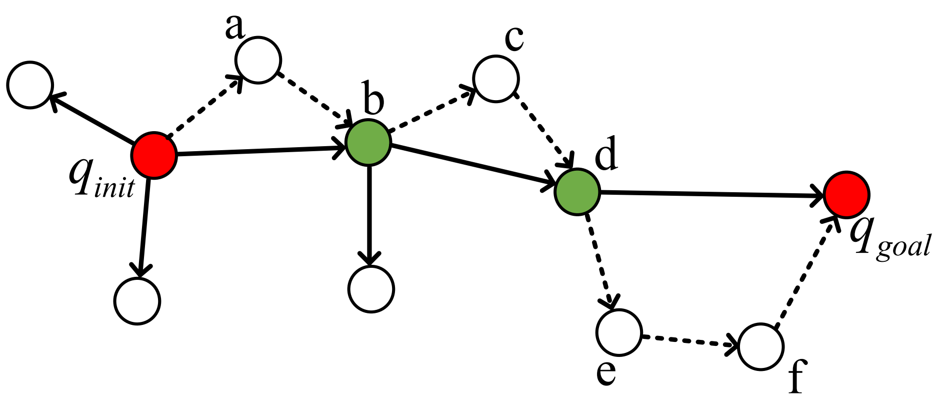


Figure 7. Path reorganization strategy.

The process of the path reorganization strategy is shown in Algorithm 2, where *keypoints* represents the set of key points. Starting from the initial node q_{init}, we traverse its children nodes for collision detection. Only the node q_{temp} closest to the end point q_{goal} is kept and added to the *keypoints*. Then, q_{temp} is used as the initial node for the next traversal.

Algorithm 2: *GetKeyPoints(path)*

```
1  $q_{temp} = q_{init};$
2  $while(q_{temp} != q_{goal})$
3    $for(x = q_{temp}; q != q_{init}; x = q_{temp}.child)$
4      $if\ CheckLine(x, q_{temp})$
5        $q_{temp} = x;$
6        $keypoints.add(q_{temp});$
```

4. Simulation Experiments of Robots

In this section, we compare and analyze two common SLAM algorithms and two path planning algorithms in a simulated environment. To visualize the performance of the algorithms, we construct maps using Rviz.

4.1. Simulation Platform

To evaluate the effectiveness of the LRBPF-SLAM algorithm in terms of mapping accuracy, we conducted simulation experiments on the Gazebo platform [33] using Ubuntu 18.04 and ROS systems. The study focused on three simulated indoor environments and used the TurtleBot3 Burger virtual robot model. The simulated sensor data included Lidar, odometer, and IMU data. The simulation was carried out on a laptop computer equipped with an Intel i7-11800H processor and 16 GB DDR4 3200 MHz memory. The simulation environment was designed to replicate realistic physical characteristics, making it a reliable reference for real-world application environments. Environment modeling of the Gazebo simulation platform is shown in Figure 8.

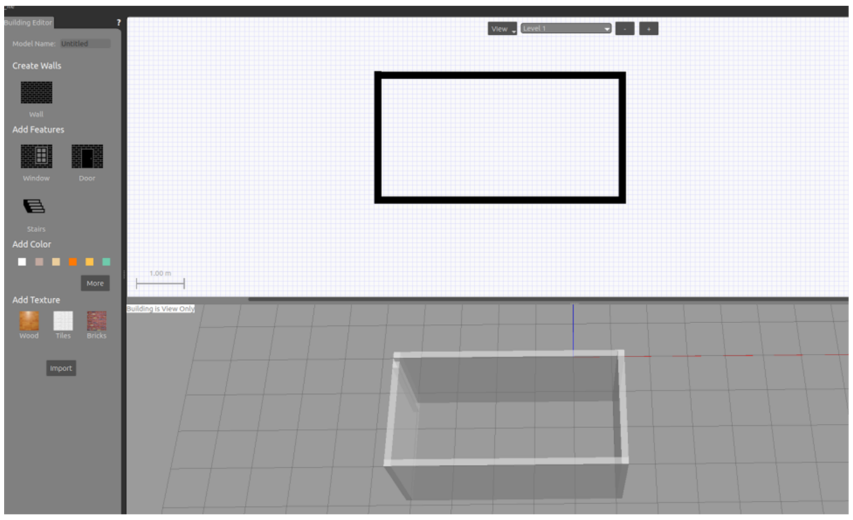


Figure 8. Environment modeling of Gazebo simulation platform.

4.2. Simulation Experiment of SLAM Algorithm

The simulation experiments of SLAM were constructed on Gazebo with three environments of different complexity for map building simulation. The different environment experiments could more accurately reflect the building effect and generalization ability of the proposed algorithm. Simulation environment 1 had a length and width of 11.25 m $\times$ 6.75 m, with regular surroundings and geometrical wall obstacles inside, to test the algorithm's building effect on geometrically shaped objects. Simulation environment 2 was 13.5 m $\times$ 8.5 m in length and width, and there were right-angle wall obstacles inside, which were used to test the algorithm's effect on building the details of corner-shaped objects. The overall simulation environment 3 was 11.85 m × 9.75 m, surrounded by irregular walls, and the internal obstacle objects were also irregular, testing the algorithm for the irregular walls and the building effect of the objects that account for the object. We compared the proposed algorithm with the Gmapping and Karto algorithms and visualized the map building results using the Rviz tool. The results of the SLAM simulation experiments for building maps are shown in Figure 9.

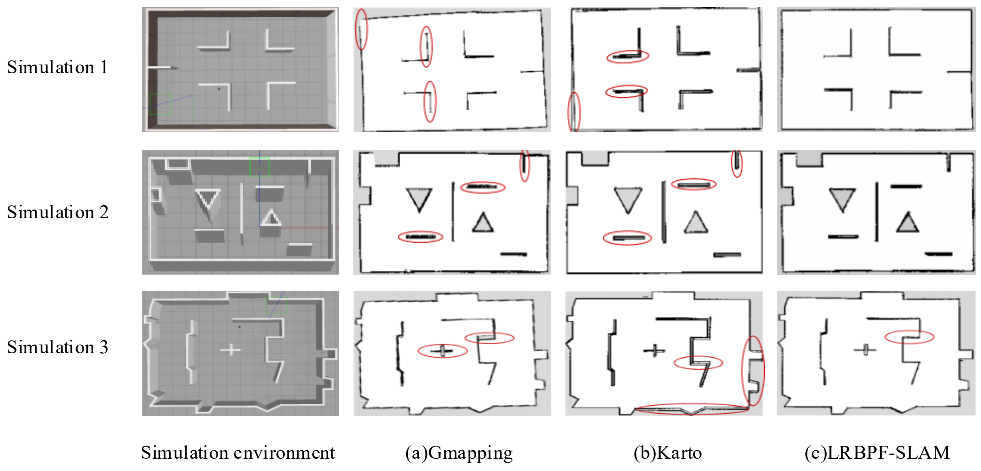


Figure 9. SLAM simulation results of three algorithms.

It can be seen from the three groups of simulation experiments that (a) the Gmapping algorithm distorts and makes a lot of noise in the wall and vertical obstacle construction, which is mainly caused by using single odometer data as the input of the distribution function. The (b) Karto algorithm is relatively good in the overall drawing effect, but in some details, the problem of wall overlap will appear. This is because the Karto algorithm is a graph optimization algorithm, which requires multiple loopback detection to optimize the result of graph construction. The (c) LRBPF-SLAM algorithm achieves satisfactory performance in the overall mapping effect and details, which benefits from using the Lidar data bit pose difference as the input of the distribution function, thus improving the mapping accuracy.

In addition to the subjective evaluation, we selected several feature points of the simulation environment for dimensional measurements and then compared the errors. The error results of the SLAM simulation experiments are analyzed in Table 1.

Based on the comparison of the feature locations between the actual and measured values by the three algorithms, we obtained the error of each feature location, as shown in Table 1. From the table, we can see that the average errors of simulation 1, simulation 2, and simulation 3 of the Gmapping algorithm are 10.4 cm, 6.4 cm, and 17.59 cm, respectively, which are relatively large and become larger as the length of the measured object increases. Simulation 1, simulation 2, and simulation 3 of the Karto algorithm have average errors of 9.34 cm, 7.64 cm, and 19.45 cm, respectively, the error of the Karto algorithm in measuring the feature size is larger. The average errors of simulation 1, simulation 2, and simulation 3 of the improved algorithm are 6.9 cm, 2.85 cm, and 11.27 cm, respectively. It can be seen that the improved algorithm always maintains smaller errors in terms of error control and has higher accuracy than the other algorithms.

Table 1. SLAM simulation experiment error results analysis.

Simulation			Gmapping		Karto		LRBPF-SLAM	
	Feature Point	Actual Value/cm	Measured Value/cm	Absolute Error/cm	Measured Value/cm	Absolute Error/cm	Measured Value/cm	Absolute Error/cm
1	1	100.00	97.99	2.01	98.48	1.52	100.86	0.86
	2	175.00	188.15	13.15	185.60	10.60	182.33	7.33
	3	325.00	341.03	16.03	340.91	15.91	337.50	12.50
	Mean	-	-	10.40	-	9.34	-	6.90
2	1	200.00	208.29	8.29	208.57	8.57	201.72	1.72
	2	225.00	216.00	9.00	231.32	6.32	228.88	3.88
	3	200.00	196.71	3.29	204.78	4.78	197.84	2.16
	4	125.00	131.14	6.14	136.52	11.52	128.02	3.02
	5	175.00	169.71	5.29	182.02	7.02	178.45	3.45
	Mean	-	-	6.40	-	7.64	-	2.85
3	1	300.00	316.00	16.00	315.74	15.74	312.05	12.05
	2	225.00	252.80	27.80	249.47	24.47	2370	12.00
	3	350.00	367.35	17.35	378.11	28.11	363.40	13.40
	4	300.00	316.00	16.00	319.64	19.64	308.10	8.10
	5	400.00	410.8	10.80	409.29	9.29	410.80	10.80
	Mean	-	-	17.59	-	19.45	-	11.27

4.3. Simulation Experiment of GBI-RRT Algorithm

In order to verify the effectiveness and search efficiency of the proposed GBI-RRT algorithm, we conducted simulations in three different environments using MATLAB2019. The simulated maps are represented with black for obstacles and white for safe space. We compared the performance of the RRT algorithm, the Bi-RRT algorithm, and the GBI-RRT algorithm by simulating each algorithm 30 times with the same parameters, including a fixed step size of 14 and identical start and end point locations. The simulated map had a horizontal coordinate range of (0, 500) and a vertical coordinate range of (0, 500).

Figure 10 presents the path planning results obtained from the (a) RRT, (b) Bi-RRT, and (c) GBI-RRT algorithms in the three simulated environments. Figure 10 indicates that the RRT and Bi-RRT algorithms produce a large number of unnecessary nodes scattered throughout the simulated map, resulting in discontinuous path curvature. However, the GBI-RRT algorithm generates a smoother planning path with fewer turning points.

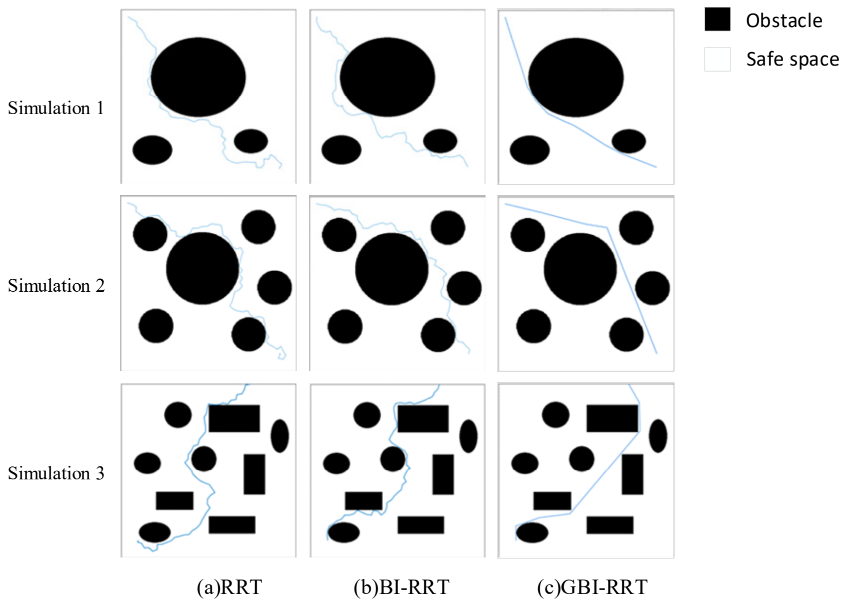


Figure 10. Results of path planning simulation experiment.

Table 2 shows the path planning times and lengths obtained using the RRT, Bi-RRT, and GBI-RRT algorithms. The results indicate that the RRT algorithm requires a much longer time to plan the path in all three simulation environments than the other two methods, especially in the complex obstacle simulated environment 3 where the longest planning time reaches 110.5 s. This is mainly due to the blindness of the expansion of the RRT algorithm. In contrast, Bi-RRT uses bi-directional search for speed optimization, which reduces the planning time to some extent. However, using the same random expansion strategy as RRT does not significantly improve the final path length, with only about a 6 m improvement in simulated environment 1.

Table 2. Comparison of the results of 30 experiments averaged over three path planning algorithms. The bold font indicates the optimal value.

Simulation	Algorithm	Time/s	Length/m
1	RRT	58.79	860.24
	Bi-RRT	16.18	854.30
	GBI-RRT	**5.08**	**674.45**
2	RRT	98.88	880.39
	Bi-RRT	13.19	859.37
	GBI-RRT	**5.46**	**679.21**
3	RRT	110.50	803.90
	Bi-RRT	5.28	777.35
	GBI-RRT	**4.84**	**594.14**

It is worth noting that the GBI-RRT algorithm probabilistically grows towards the target point with the help of the proposed target bias sampling, resulting in a significant reduction in planning time compared to the previous two. It performs well in all three environments with an average planning time of about 5 s. The path length is further optimized by using a path reorganization strategy for the already planned paths, with an average reduction of about 181 m compared to the previous two.

Figures 11 and 12 show the line graphs depicting the planning time and planning paths obtained by the GBI-RRT algorithm in 30 experiments across three environments. From the plots, it can be observed that the planning time is generally stable within a certain range, while the planning path length fluctuates within a certain range, indicating good performance.

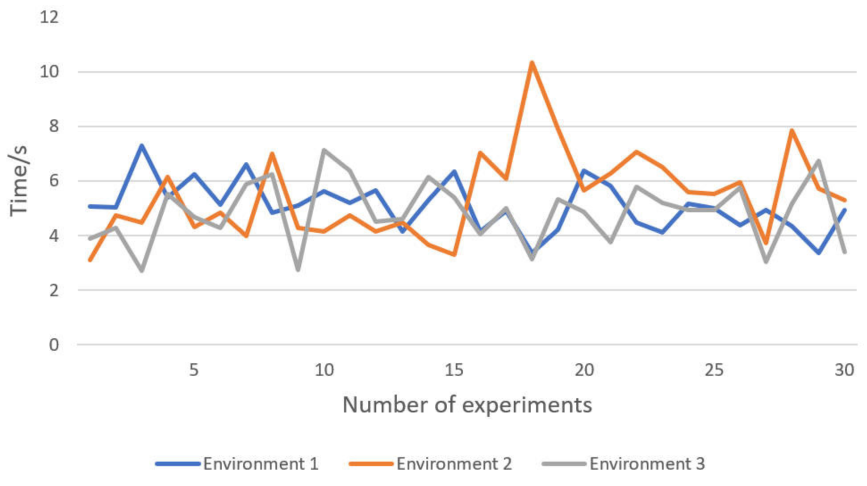


Figure 11. Planning time of 30 times GBI-RRT algorithm in three environments.

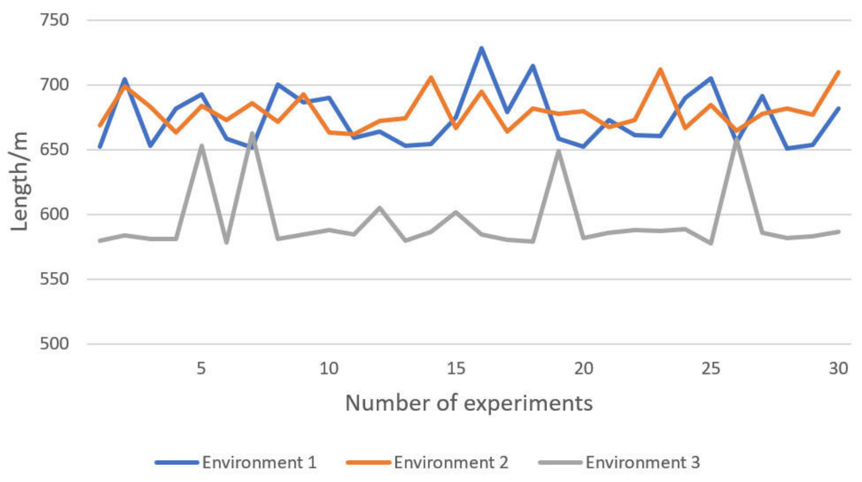


Figure 12. Planning path length of 30 times GBI-RRT algorithm in three environments.

5. Real Scenario Experiments for Robots

5.1. Real Scenario Experiment Setup

We use the distributed framework of the ROS platform to perform robotic tasks. The framework enables communication between nodes through a loosely coupled approach and is able to run on different computers. The robot and the computer must be on the same LAN to enable remote control of the robot via SSH commands. In addition, we provide a visual interface to make the control of the robot more intuitive by operating it from the computer terminal. This configuration greatly improves the flexibility and operability of the robot tasks.

Our specific configuration is as follows:

1. Host controller Jetson Nano and laptop are connected to the same network. A hotspot network on the phone is used to cover the robot's movement area.
2. The "ifconfig" command is used to check the IP addresses of the Jetson Nano and the laptop.
3. In the Ubuntu system of the laptop, the environment variables "ROS_MASTER_URI" and "ROS_HOSTNAME_URI" are added to the "bashrc" file. "ROS_MASTER_URI" points to the IP address of the Jetson Nano, while "ROS_HOSTNAME_URI" points to the IP address of the Ubuntu system on the laptop.
4. Finally, the robot is remotely accessed using SSH commands in the Ubuntu system terminal for visual remote control. This remote access method makes the robot more visible and makes it easier for the operator to control. These configuration measures greatly improved the efficiency and flexibility of the robot's tasks.

5.2. Experiment of SLAM Algorithm

In our practical experimental study, we conducted SLAM experiments in three real scenarios. Environment 1 is an indoor bedroom measuring 4.5 $\times$ 4.5 m, featuring obstacles such as cabinets, refrigerators, and tables. Environment 2 is a corner corridor with a total length of 15 m and a width of 2.5 m, containing obstacles such as regular wooden doors and irregular walls. Environment 3 is a conference room with a space of 4.5 $\times$ 6 m, featuring obstacles such as tables, chairs, uneven walls, and monitor stands. This scene is characterized by a high obstacle density. By performing experiments in these diverse real scenarios, we can more effectively evaluate the proposed method's effectiveness.

According to the experimental results in Figure 13, it can be seen that the (a) Gmapping algorithm underperforms in all three scenes with low building accuracy, blurred obstacle contours, the ghosting phenomenon in local details, incomplete wall building, and an inability to identify support legs of many chairs. In comparison, the (b) Karto algorithm can build complete maps in all three scenes, but with average reconstruction of local details. However, the (c) LRBPF-SLAM algorithm outperforms both algorithms with the best overall map-building effect in all three scenes without the ghosting phenomenon. In the complex conference room environment, the algorithm can fully scan wall contours, recognize chair support legs with high accuracy, and build highly precise detailed maps.

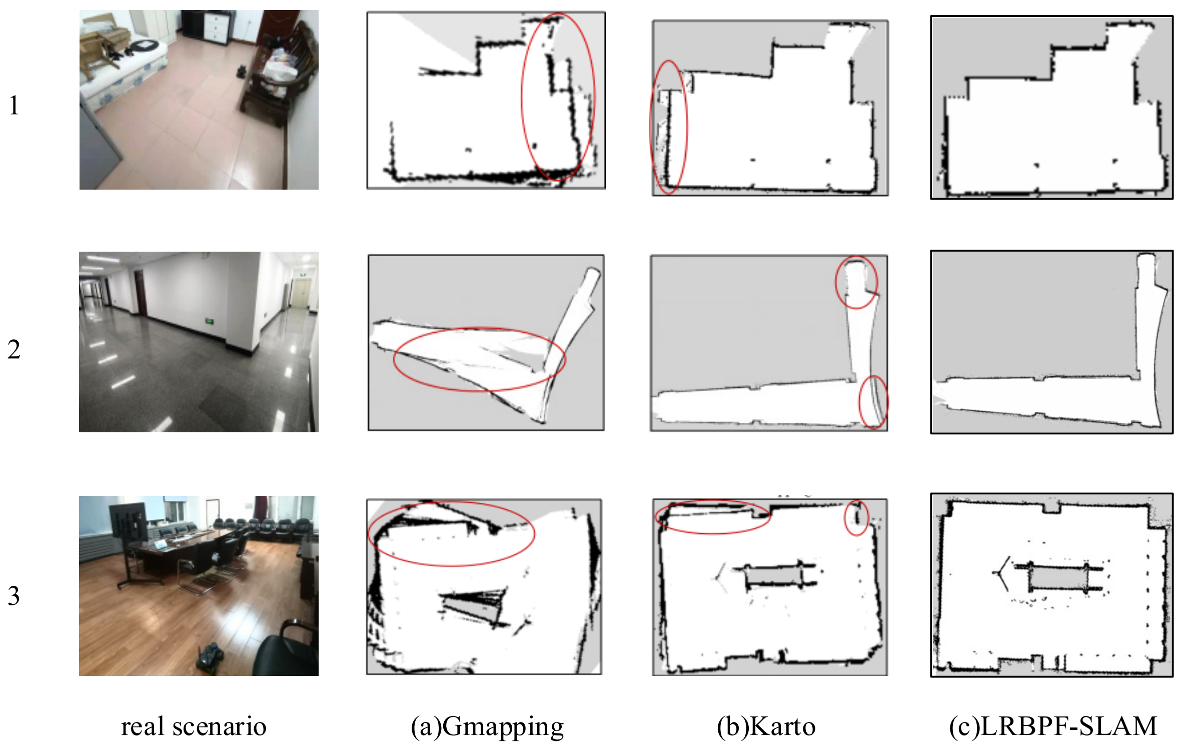


Figure 13. SLAM results of three algorithms in real scenarios.

Furthermore, we selected several typical feature locations in the real scenarios and compared their real values with the measured values, producing error results analysis tables.

According to the data in Table 3, it can be found that the average error of the Gmapping algorithm in the three different environments is 3.44 cm, 8.95 cm, and 6.74 cm, respectively. It is worth noting that the maximum error of the algorithm in feature location 3 of experiment 3 reaches 12.9 cm; in comparison, the average error of the Karto algorithm in these three environments is 3.83 cm, 6.04 cm, and 5.86 cm. The LRBPF-SLAM algorithm, on the other hand, exhibits the best accuracy, with average errors of 2.63 cm, 4.33 cm, and 2.74 cm in the three environments, and the maximum error is only 7.5 cm in feature 1 of experiment 2. The algorithm is also able to accurately reconstruct the details of the environment. The experimental results show that the proposed LRBPF-SLAM algorithm has a small overall error and high accuracy in map building, and can effectively reconstruct the overall state of the environment. From these data, it can be concluded that the LRBPF-SLAM algorithm has significant advantages, especially in complex environments that show better performance.

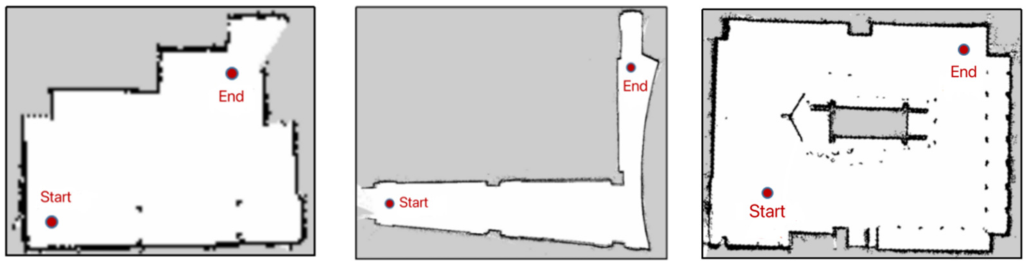
Table 3. SLAM experimental error results analysis in real scenes. The bold font indicates the optimal value.

Experiment			Gmapping		Karto		LRBPF-SLAM	
	Feature Point	Actual Value/cm	Measured Value/cm	Absolute Error/cm	Measured Value/cm	Absolute Error/cm	Measured Value/cm	Absolute Error/cm
1	1	42.00	45.30	**3.30**	45.48	3.48	46.92	4.92
	2	41.00	47.11	6.11	50.94	9.94	46.92	4.92
	3	50.00	48.92	1.08	49.12	0.88	50.55	0.55
	4	112.00	108.72	3.28	110.98	1.02	111.89	0.11
	Mean	-	-	3.44	-	3.83	-	2.63
2	1	139.00	151.00	12.00	149.60	8.30	146.50	7.50
	2	115.00	126.40	11.40	124.35	7.35	121.00	6.00
	3	84.00	92.60	5.60	93.00	3.20	89.40	2.40
	4	104.00	114.80	10.80	112.90	5.70	111.20	5.10
	5	115.00	131.30	7.80	127.70	8.50	118.60	3.60
	6	57.00	63.10	6.10	60.20	3.20	58.40	1.40
	Mean	-	-	8.95	-	6.04	-	4.33
3	1	57.00	64.30	5.30	53.60	3.40	59.10	2.10
	2	41.00	47.80	6.80	45.60	7.60	42.50	1.50
	3	370.00	382.90	12.90	379.80	9.80	378.70	5.20
	4	43.00	46.50	3.50	39.90	3.10	44.80	1.80
	5	39.00	42.20	5.20	43.50	4.50	42.10	3.10
	Mean	-	-	6.74	-	5.68	-	2.74

5.3. Experiment of Path Planning Algorithm

We compare the path planning results of the RRT, Bi-RRT and GBI-RRT algorithms in three different real scenarios.

As Figure 14 shows, the map of the three experimental sites obtained from the experiments in the previous section, the starting and ending points of the mobile robot are set. Table 4 shows the experimental data of path planning for the three algorithms RRT, Bi-RRT, and GBI-RRT. To minimize the error, the experimental data represent the average value of 20 experiments.



(**a**) Environment 1 (**b**) Environment 2 (**c**) Environment 3

Figure 14. Three experimental maps.

As shown in Table 4, the 20 experiments were conducted for three real scenarios, and then their averages were taken for path planning quality analysis.

In the simple Environment 1, the RRT and Bi-RRT algorithms require an average of 2.65 and 2.5 turns, respectively, while GBI-RRT requires only 0.45 turns on average, and the other two metrics (time and length) differ less among the three algorithms. In Environment 2, the number of turns increases for all three algorithms. Nevertheless, GBI-RRT outperforms the other two algorithms in terms of path planning time and length, with 5.1 and 3.75 s less time than RRT and Bi-RRT, respectively, and less difference in planning length between the three algorithms. In Environment 3, compared with Bi-RRT, the path planning time of GBI-RRT is reduced by 3.15 s, the path planning length is reduced by 2.35 m, and the number of turns is reduced by 2.2 turns. These results show that the GBI-RRT algorithm can quickly generate a smooth and optimal path from the origin to the destination.

Table 4. Quality analysis of three algorithms for path planning in three different environments. The bold font indicates the optimal value.

Environment	Algorithm	Time/s	Length/m	No. of Turns
1	RRT	6.95	5.97	2.65
	Bi-RRT	6.90	5.96	2.50
	GBI-RRT	**4.55**	**5.24**	**0.45**
2	RRT	31.25	18.14	7.55
	Bi-RRT	29.90	17.31	7.20
	GBI-RRT	**26.15**	**16.42**	**4.75**
3	RRT	17.95	11.29	4.85
	Bi-RRT	17.70	11.05	4.70
	GBI-RRT	**14.55**	**8.70**	**2.50**

5.4. Robot Navigation Process

Figure 15 depicts the autonomous navigation process of the robot, which is conducted within a known map constructed by SLAM. Connect to the computer through the ssh command to control the robot, run the navigation command and select the map path, and then start the visualization tool Rviz.

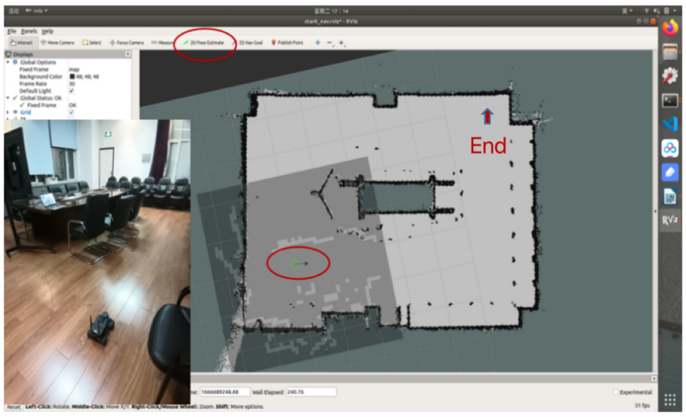
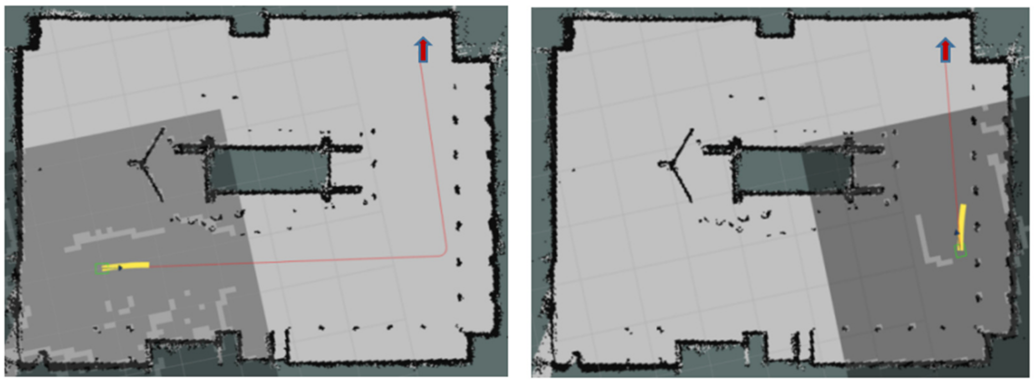


Figure 15. Initial position and pose of navigation robot.

As shown in Figure 15, the lower left corner depicts the pose of the robot in the real environment. The red circle located on the top menu bar is the 2D Pose Estimate that is utilized to determine the robot's initial pose, with the red circle marked on the map representing the determined initial pose. The shaded square surrounding the robot denotes the local cost map, which represents the area for local path planning. By selecting the navigation endpoint in the upper right corner of the map, the robot can execute autonomous navigation operations.

Figure 16 is the robot's initial pose and planning information during the movement process. The yellow line segment situated in front of the robot represents the local path planning Dynamic Window Approach (DWA) algorithm [34]. Whenever the robot approaches an obstacle, the DWA algorithm executes obstacle avoidance processing by selecting a safe path around the obstacle. Meanwhile, the long red line segment indicates the path planned by the global path planning GBI-RRT. Finally, in Figure 17, the robot arrives at its destination, concluding the navigation.



(**a**) Initial planning information. (**b**) Motion process planning information.

Figure 16. Mobile Robot Status Information.

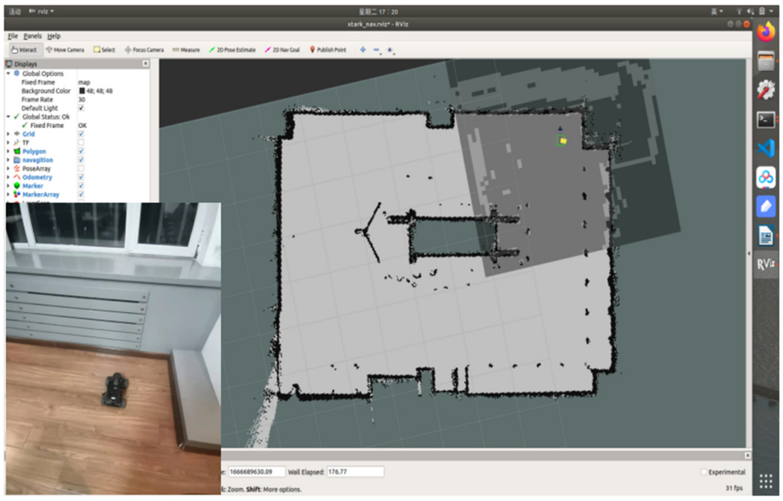


Figure 17. End point posture.

6. Conclusions

This study proposes an enhanced LRBPF-SLAM and GBI-RRT path planning algorithm to improve the navigation of autonomous mobile robots in indoor environments. LRBPF-SLAM overcomes the limitations of traditional distribution functions by utilizing Lidar data, resulting in more accurate pose estimation of the robot. GBI-RRT incorporates target bias sampling to efficiently guide nodes towards the goal, reducing ineffective searches. The path reorganization strategy further improves navigation efficiency by eliminating low-quality nodes and improving path curvature. The proposed method is evaluated in simulations and field experiments, and the results demonstrate superior performance compared to existing algorithms. Future research could focus on applying the currently proposed methods to more complex environments to better address the challenges of the real world. Researchers can also consider how to improve model speed and accuracy more effectively, and apply these algorithms to other fields.

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

Funding: This work was supported by the [Liaoning Province Higher Education Innovative Talents Program #1] under Grant [number LR2019058] and [Liaoning Province Joint Open Fund for Key Scientific and Technological Innovation Bases #2] under Grant [number 2021-KF-12-05].

Data Availability Statement: Data sharing not applicable, No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The authors would like to acknowledge support from the following projects: Liaoning Province Higher Education Innovative Talents Program Support Project (Grant No. LR2019058), Liaoning Province Joint Open Fund for Key Scientific and Technological Innovation Bases (Grant No.2021-KF-12-05) and Zhejiang Provincial Natural Science Foundation of China (LQ23F030001).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Py, F.; Robbiani, G.; Marafioti, G.; Ozawa, Y.; Watanabe, M.; Takahashi, K.; Tadokoro, S. SMURF software architecture for low power mobile robots: Experience in search and rescue operations. In Proceedings of the 2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Sevilla, Spain, 8–10 November 2022; pp. 264–269. [CrossRef]
2. Sui, L.; Lin, L. Design of Household Cleaning Robot Based on Low-cost 2D LIDAR SLAM. In Proceedings of the 2020 International Symposium on Autonomous Systems (ISAS), Guangzhou, China, 6–8 December 2020; pp. 223–227. [CrossRef]
3. Farooq, M.U.; Eizad, A.; Bae, H.-K. Power solutions for autonomous mobile robots: A survey. *Robot. Auton. Syst.* **2023**, *159*, 104285. [CrossRef]
4. Ismail, H.; Roy, R.; Sheu, L.-J.; Chieng, W.-H.; Tang, L.-C. Exploration-Based SLAM (e-SLAM) for the Indoor Mobile Robot Using Lidar. *Sensors* **2022**, *22*, 1689. [CrossRef]
5. Gao, L.; Dong, C.; Liu, X.; Ye, Q.; Zhang, K.; Chen, X. Improved 2D laser slam graph optimization based on Cholesky decomposition. In Proceedings of the 2022 8th International Conference on Control, Decision and Information Technologies (CoDIT), Istanbul, Turkey, 17–20 May 2022; Volume 1, pp. 659–662.
6. Hampton, B.; Al-Hourani, A.; Ristic, B.; Moran, B. RFS-SLAM robot: An experimental platform for RFS based occupancy-grid SLAM. In Proceedings of the 2017 20th International Conference on Information Fusion (Fusion), Xi'an, China, 10–13 July 2017.
7. Juric, A.; Kendes, F.; Markovic, I.; Petrovic, I. A Comparison of Graph Optimization Approaches for Pose Estimation in SLAM. In Proceedings of the 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 27 September–1 October 2021; pp. 1113–1118. [CrossRef]
8. Dhaoui, R.; Rahmouni, A. Mobile Robot Navigation in Indoor Environments: Comparison of Lidar-Based 2D SLAM Algorithms. In *Design Tools and Methods in Industrial Engineering II: Proceedings of the Second International Conference on Design Tools and Methods in Industrial Engineering, ADM 2021, Rome, Italy, 9–10 September 2021*; Springer International Publishing: Berlin/Heidelberg, Germany, 2021; pp. 569–580.
9. Konolige, K.; Grisetti, G.; Kümmerle, R.; Burgard, W.; Limketkai, B.; Vincent, R. Efficient sparse pose adjustment for 2D map-ping. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010; pp. 22–29.
10. Ribeiro, M.I. Kalman and extended kalman filters: Concept, derivation and properties. *Inst. Syst. Robot.* **2004**, *43*, 3736–3741.
11. Talwar, D.; Jung, S. Particle filter-based Localization of a mobile robot by using a single Lidar sensor under SLAM in ROS environment. In Proceedings of the 2019 19th International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 15–18 October 2019; Volume 43, pp. 3736–3741. [CrossRef]
12. Cai, Y.; Qin, T. Design of Multisensor Mobile Robot Vision Based on the RBPF-SLAM Algorithm. *Math. Probl. Eng.* **2022**, *2022*, 1518968. [CrossRef]
13. Dai, Y.; Zhao, M. Grey Wolf Resampling-Based Rao-Blackwellized Particle Filter for Mobile Robot Simultaneous Localization and Mapping. *J. Robot.* **2021**, *2021*, 4978984. [CrossRef]
14. Tee, Y.K.; Han, Y.C. Lidar-based 2D SLAM for mobile robot in an indoor environment: A review. In Proceedings of the 2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST), Miri, Malaysia, 7–9 July 2021; pp. 1–7. [CrossRef]
15. Maziarz, B.; Domański, P.D. Customized fastSLAM algorithm: Analysis and assessment on real mobile platform. *Nonlinear Dyn.* **2022**, *110*, 669–691. [CrossRef]
16. Muhammad, A.; Ali Mohammed, A.H.; Turaev, S.; Abdulghafor, R.; Shanono, I.H.; Alzaid, Z.; Alruban, A.; Alabdan, R.; Dutta, A.K.; Almotairi, S. A Generalized Laser Simulator Algorithm for Mobile Robot Path Planning with Obstacle Avoidance. *Sensors* **2022**, *22*, 8177. [CrossRef]
17. LaValle, S.M. Rapidly-exploring random trees: A new tool for path planning. *Annu. Res. Rep.* **1998**.
18. Dijkstra, E.W. A Note on Two Problems in Connexion with Graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*; ACM: New York, NY, USA, 2022; pp. 287–290. [CrossRef]
19. Jin, J.; Zhang, Y.; Zhou, Z.; Jin, M.; Yang, X.; Hu, F. Conflict-based search with D* lite algorithm for robot path planning in unknown dynamic environments. *Comput. Electr. Eng.* **2023**, *105*, 108473. [CrossRef]
20. Liu, B.; Liu, C. Path planning of mobile robots based on improved RRT algorithm. *J. Phys. Conf. Ser.* **2022**, *2216*, 012020. [CrossRef]
21. Pohl, I. *BI-Directional and Heuristic Search in Path Problems*; Stanford Linear Accelerator Center: Menlo Park, CA, USA, 1969.
22. Li, Z.; Li, L.; Zhang, W.; Wu, W.; Zhu, Z. Research on Unmanned Ship Path Planning based on RRT Algorithm. *J. Phys. Conf. Ser.* **2022**, *2281*, 012004. [CrossRef]

23. Zhang, X.; Zhu, T.; Du, L.; Hu, Y.; Liu, H. Local Path Planning of Autonomous Vehicle Based on an Improved Heuristic Bi-RRT Algorithm in Dynamic Obstacle Avoidance Environment. *Sensors* **2022**, *22*, 7968. [CrossRef] [PubMed]
24. Xu, J.; Tian, Z.; He, W.; Huang, Y. A fast path planning algorithm fusing PRM and P-BI-RRT. In Proceedings of the 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan), Jinan, China, 23–25 October; pp. 503–508.
25. Gan, Y.; Zhang, B.; Ke, C.; Zhu, X.F.; He, W.M.; Ihara, T. Research on Robot Motion Planning Based on RRT Algorithm with Nonholonomic Constraints. *Neural Process. Lett.* **2021**, *53*, 3011–3029. [CrossRef]
26. Wang, J.; Li, B.; Meng, M.Q.-H. Kinematic Constrained Bi-directional RRT with Efficient Branch Pruning for robot path planning. *Expert Syst. Appl.* **2020**, *170*, 114541. [CrossRef]
27. Grothe, F.; Hartmann, V.N.; Orthey, A.; Toussaint, M. ST-RRT*: Asymptotically-Optimal Bidirectional Motion Planning through Space-Time. *arXiv* **2022**, arXiv:2203.02176.
28. Zhao, H. Path Planning of Mobile Robots Based on Improved Bi-RRT Algorithm. In Proceedings of the 2022 IEEE 5th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), Shenyang, China, 18–20 November 2022; pp. 1043–1050. [CrossRef]
29. Ma, G.; Duan, Y.; Li, M.; Xie, Z.; Zhu, J. A probability smoothing Bi-RRT path planning algorithm for indoor robot. *Future Gener. Comput. Syst.* **2023**, *143*, 349–360. [CrossRef]
30. Choi, J.; Jeong, B.; Theotokatos, G.; Tezdogan, T. Approach an autonomous vessel as a single robot with Robot Operating System in virtual environment. *J. Int. Marit. Saf. Environ. Aff. Shipp.* **2022**, *6*, 50–66. [CrossRef]
31. Kang, J.-G.; Lim, D.-W.; Choi, Y.-S.; Jang, W.-J.; Jung, J.-W. Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning. *Sensors* **2021**, *21*, 333. [CrossRef] [PubMed]
32. Zhang, Y.; Wang, H.; Yin, M.; Wang, J.; Hua, C. Bi-AM-RRT*: A Fast and Efficient Sampling-Based Motion Planning Algorithm in Dynamic Environments. *arXiv* **2023**, arXiv:2301.11816.
33. Platt, J.; Ricks, K. Comparative Analysis of ROS-Unity3D and ROS-Gazebo for Mobile Ground Robot Simulation. *J. Intell. Robot. Syst.* **2022**, *106*, 80. [CrossRef]
34. Li, Y.; Li, J.; Zhou, W.; Yao, Q.; Nie, J.; Qi, X. Robot Path Planning Navigation for Dense Planting Red Jujube Orchards Based on the Joint Improved A* and DWA Algorithms under Laser SLAM. *Agriculture* **2022**, *12*, 1445. [CrossRef]