Design and Implementation of Autonomous Navigation System Based on Tracked Mobile Robot

Hui Li, Junhan Cui, Yifan Ma, Jiawei Tan, Xiaolei Cao, Chunlong Yin, and Zhihong Jiang(✉)

School of Mechatronical Engineering, Beijing Advanced Innovation Center for Intelligence Robots and System, Beijing Institute of Technology, Beijing 100081, China

jiangzhihong@bit.edu.cn

Abstract. In this paper, we introduce an autonomous exploration and rescue robot system based on a tracked mobile robot platform equipped with a 7 degree-of-freedom (DoF) manipulator, which realizes autonomous navigation in indoor environments and autonomous stair climbing for safe and efficient search and rescue tasks. In Sect. 2, the hardware design of the robot system is presented, which allows flexible movement and high passability to complete obstacle crossing and stair climbing. In Sects. 3 and 4, the indoor navigation algorithm and the stair detection algorithm of the robot system are presented, respectively. The ROS-based system uses the cartographer algorithm for map construction, ROS navigation for autonomous navigation and obstacle avoidance, and a depth camera for stair detection. The process of a four-flipper tracked mobile robot stair climbing is designed. The robot system is experimentally verified in Sect. 5.

Keywords: Tracked robot · SLAM · Path planning · Navigation · Autonomous stair climbing

1 Introduction

Mobile rescue robots play an extremely important role in search and rescue efforts for building fires and other situations. When building structures and fire conditions are not clear, firefighters who rashly enter a building on fire for rescue may face huge unknown risks; In response to this situation, we have designed a robot that can autonomously search and rescue in an indoor environment. The robot can perform SLAM mapping, autonomous navigation and movement in such indoor environment. In addition, it has the capability of autonomously climbing stairs to achieve autonomous exploration and rescue between multiple floors.

Supported by the National Natural Science Foundation of China (61733001, U22B2079, 61873039, 62103054, U1913211, U2013602, 62273049).

F. Sun et al. (Eds.): ICCSIP 2022, CCIS 1787, pp. 329–350, 2023.
https://doi.org/10.1007/978-981-99-0617-8_23

At present, Indoor navigation methods of mobile robots are mostly based on wheeled robots [2], but wheeled robots have many limitations, such as being unable to complete some obstacle-crossing operations in the indoor environment, and being unable to complete climbing stairs to explore multiple floors. much research on stair-climbing robots does not involve the navigation method of indoor mobile robots as well. For example, the article [1,5] proposes an autonomous stair climbing method based on tracked mobile robots. In addition, some methods for stair detection have not been applied to mobile robots, such as the stair detection algorithm proposed in the article [9,10] for disabled people carrying wearable devices.

In view of the existing problems, we use a tracked mobile robot with four flippers to carry out indoor rescue exploration tasks. the tracked robot with high-power motor could move quickly in an indoor environment, and due to its differential configuration, it can turn around and change direction in a narrow space, so it has high dexterity and can complete narrow obstacle avoidance tasks in space. At the same time, because the tracked robot has flippers at the front and rear, it also has high passability under the promise of satisfying dexterity and can complete tasks such as climbing stairs and crossing various obstacles.

The four-flipper tracked mobile robot has excellent performance, this paper proposes a method of autonomous mobile navigation and autonomous stair climbing based on it. The proposed method realizes the robot's autonomous navigation, obstacle avoidance, and autonomous stair climbing in the indoor environment so that the robot can complete detection and rescue tasks in various complex environments.

This paper assumes that the robot is located in a multi-story indoor environment, and the task objectives are as follows: (i) Detecting the floor and generating a floor map. (ii) Rescue would be carried out while the situation is found during the detection process, otherwise, entering the stairwell and carrying out detection on the next floor. (iii) Detecting stairs, completing the task of autonomous stair climbing, and entering the next floor for exploration. In the second section, the hardware components of the whole robot system and the experiment environment are mainly described. In the third and fourth sections, the autonomous navigation algorithm and the autonomous stair-climbing method of the robot are introduced. In the fifth section, the process and results of the whole experiment are presented.

2 Robot System Design

The four-flipper tracked mobile robot designed in this paper is used for the rescue of simulated fire buildings. The robot system consists of a four-flipper tracked robot platform, a 7-DoF redundant manipulator and sensors. The overall structure of the robot is shown in Fig. 1.

The robot system is driven by a vehicle-mounted 48 V lithium battery, with a weight of 180 kg. It adopts a four-flipper tracked robot platform and a humanoid manipulator configuration scheme. A four-flipper tracked robot platform is designed in this paper, two main tracks provide power, and four flippers

Fig. 1. The overall appearance of the robot.

are used as an auxiliary, which can make the tracked platform have high passability and stability. The tracked platform is 986 mm long, 731 mm wide, 306 mm high, and 623 mm in length of the flipper. The left and right tracks are driven by two 2000 W tracked motors, and the flipper are driven by two 400 W motors.

The industrial computer of the robot system designed in this paper is the Nvidia Jetson AGX Xavier computer, and the operating environment is Ubuntu18.04, ROS melodic. The entire robot system is developed based on the ROS system to implement algorithms such as SLAM mapping, autonomous navigation, and stair detection. The hardware framework and communication method of the robot system designed in this paper are shown in Fig. 2.

In the robot system designed in this paper, the sensors used in the navigation and detection algorithms are lidar, IMU, and depth camera. The lidar is Velodyne VLP-16, the IMU is Xsens MTI-630, and the depth camera is Microsoft Kinect v2. Lidar and IMU are used for SLAM mapping of robots, mobile navigation, and obstacle avoidance, IMU is also used for state feedback when robots climb stairs, and depth cameras are used for the detection and recognition of stairs.

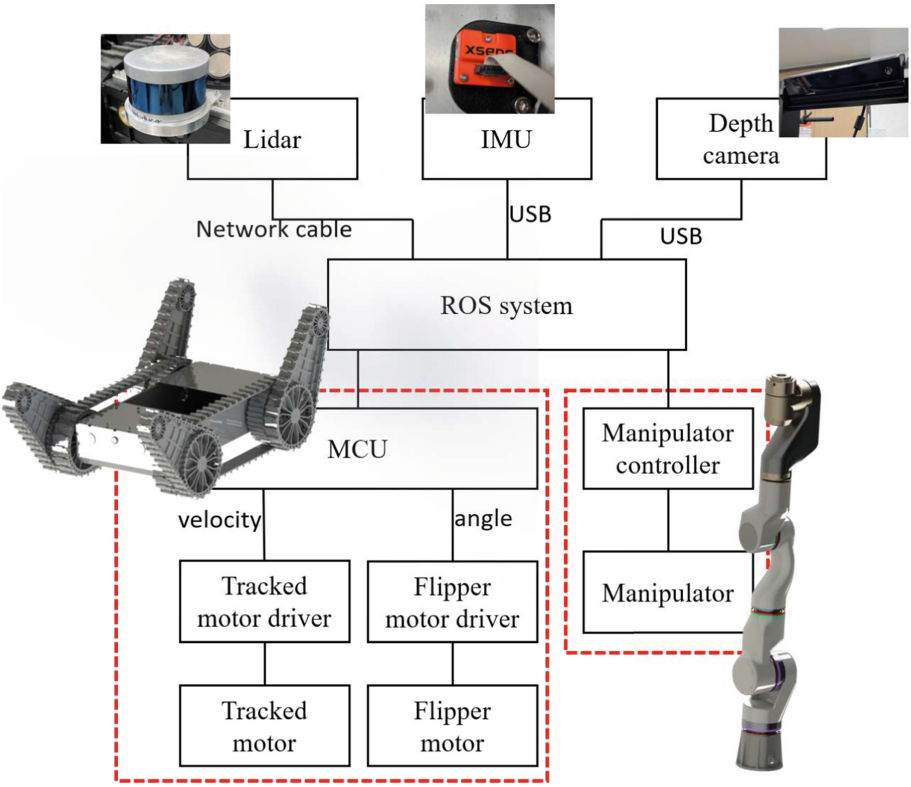


Fig. 2. The hardware framework and communication method of the robot system.

3 Mapping and Navigation Algorithm

In this paper, the cartographer algorithm developed by Google was adopted to complete the SLAM mapping of the robot in the indoor environment. Based on the ROS system, the localization of the cartographer and the move_base developed by [2] were used to realize the mobile navigation of the robot in the indoor environment.

3.1 Cartographer Algorithm

Google Cartographer algorithm adopts the theoretical framework of SLAM based on graph optimization. It is mainly composed of Local SLAM and Global SLAM. The Local SLAM is responsible for scan-to-submap and submap insertion, the Global SLAM is responsible for optimizing pose estimation, and the branch and bound method is used to complete the global loop closures detection. The mathematical model of the Cartographer algorithm is expressed as:

$$x_k = f(x_{k-1}, u_k) + w_k \tag{1}$$

$$z_{k,j} = h(y_j, x_k) + v_{k,j} \tag{2}$$

Equation (1) is called the motion equation, and the current position x_k is calculated by the position x_{k-1} at the previous moment and the sensor data u_k. Equation (2) is called the observation equation, which represents the observation data $z_{k,j}$ generated when the landmark y_j is observed at the position x_k.

The cartographer algorithm introduces the concept of a submap in the Local SLAM part. When the Local SLAM module receives a new frame of lidar data, it uses odometry and IMU data to calculate the trajectory to obtain the robot pose estimate. Then the robot pose estimate is used as the initial value to match with the newly established submap, to complete the update of the robot pose estimate. The filtered frame data is inserted into the best position of the submap to form a new submap.

When a new laser data frame arrives, the coordinate transformation needs to be done which transforms the laser data coordinate to the submap coordinate before inserting the new laser data frame into submap.

$$T_\xi p = \begin{Bmatrix} \cos\xi_\theta & -\sin\xi_\theta \\ \sin\xi_\theta & \cos\xi_\theta \end{Bmatrix} p + \begin{Bmatrix} \xi_x \\ \xi_y \end{Bmatrix} \tag{3}$$

The front end of the Cartographer algorithm, scan matching is based on the scan-to-map correlation method [8], modeled by the Maximum Likelihood Estimation:

$$x_k^* = \arg\max_{x_k} \{p(z_k|x_k, y_k)p(x_k|x_{k-1}, u_k)\} \tag{4}$$

Since the laser data of each frame is independent to each other, it can be considered to be uncorrelated, then:

$$p(z_k|x_k, y_k) = \prod_i p(z_k^i|x_k, y_k) \propto \sum_i \log p(z_k^i|x_k, y_k) \tag{5}$$

Equation (5) splits the probability of the entire observation data of one frame into the probability of each point of the current frame data. Before calculating this probability, the data frame and map need to be discretized. The process for updating an occupancy grid map is as follows:

$$odds(x) = \frac{p(x)}{1 - p(x)} \tag{6}$$

$$M_{new}(x) = clamp[odds^{-1}(odds(M_{old}(x)) \cdot odds(P_{hit}))] \tag{7}$$

When new data frames additions are finished, the submap is complete. But maps generated by Local SLAM module alone have a large cumulative error over time. The Global SLAM module is responsible for optimizing the pose estimation [7], and the branch and bound method is used to complete the global loop closure detection [6].

3.2 Trajectory Planning Algorithm

The whole path planning process is devided into two parts. Firstly, a general path is planned by the global path planner, and then the local path planner divides it into several small segments and performs local path planning. The advantage is that the obstacles saved on the map can be avoided during global planning, and new obstacles (including dynamic obstacles) can be avoided during local path planning. Global path planning and local path planning cooperate to complete the navigation together.

The Dijkstra algorithm and the A* algorithm are used for global path planning, and Dynamic Window Approach(DWA) is used for local path planning [4].

Global Path Planner. The global path planner mainly receives cartographer positioning information, global map information, and navigation end-point information. The Dijkstra algorithm and A* algorithm are mainly used in the shortest path planning in 2D grid maps, and the global path planning algorithm used in this paper is the A* algorithm.

Dijkstra algorithm is a typical breadth-first-based shortest path search algorithm, which is used to calculate the shortest path from a specific vertex to all other vertices in a graph or network. The main feature of this algorithm is that it starts from one vertex and expands layer by layer until the expansion covers all vertices. Dijkstra algorithm is essentially a greedy algorithm, and the key is to obtain a local optimal solution in each step in order to expect a global optimal solution.

Dijkstra algorithm will calculate the shortest path from the starting vertex to every vertices in the graph. Such a search is blind and consumes a lot of computing power and time.

In order to solve the blindness of the search direction in the Dijkstra algorithm, the A* algorithm is improved by a heuristic function. The A* algorithm calculates the priority of each vertex by Eq. (8).

$$f(n) = g(n) + h(n) \tag{8}$$

In the heuristic function of the A* algorithm, $f(n)$ is the overall priority of node n. When selecting the next node to traverse, the node with the highest overall priority (least cost) is selected. $g(n)$ is the cost of the node from the starting point and $h(n)$ is the estimated cost of node n to the end point.

The heuristic function $h(n)$ affects the behavior of the A* algorithm. When $h(n) = 0$, the A* algorithm degenerates into Dijkstra algorithm; When $h(n)$ is much greater than $g(n)$, the A* algorithm becomes the best priority search algorithm, and the outputs cannot be guaranteed to be the shortest path; When $h(n)$ is always less than or equal to the cost of node n to the end point, the A* algorithm guarantees that the shortest path will be found. The navigation map used in this paper is a 2D grid map generated by the cartographer algorithm,

Algorithm 1: A* algorithms

Input: Start, End.
Output: Path.

```
initialization Start and End;
while True do
    if open list == NULL then
        Search failed, return;
    end
    Take the g(n) + h(n) smallest node in the open list Add the node to the
    closed list;
    if current node == End then
        Find the path, return;
    end
    Traverse current node neighbors that are not in the closed list;
    if node in open list then
        Update node of the g(n) ;
    else
        Calculate the g(n) of the node;
        add open list;
    end
end
```

therefore the Euclidean distance from the current point to the target point can be used as the heuristic function $h(n)$

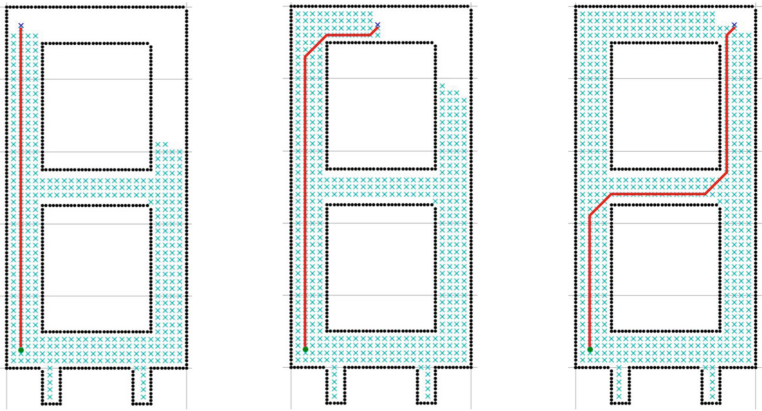
$$h(n) = \sqrt{(p_2.x - p_1.x)^2 + (p_2.y - p_1.y)^2} \tag{9}$$

Under the same grid map, the Dijkstra algorithm and the A* algorithm are used for path planning, as shown in Fig. 3 and Table 1.

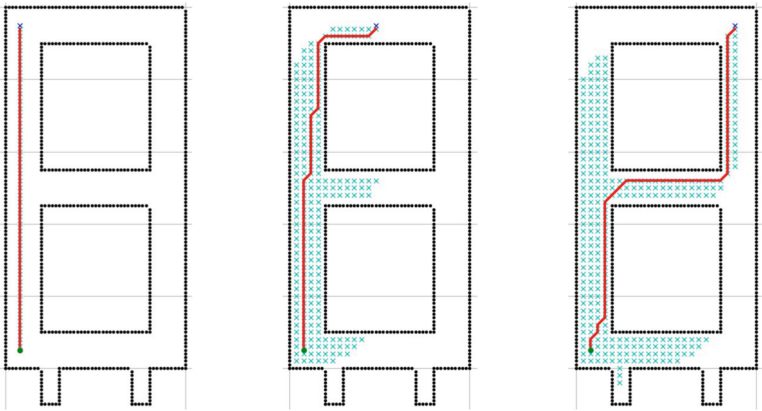
Table 1. *Dijkstra* and *A** algorithm comparison

Trajectory length	Point 1	Point 2	Point 3
Dijkstra algorithm	192.0	220.4	255.9
A∗ algorithm	192.0	220.4	255.9
Planning time	point 1	point 2	point 3
Dijkstra algorithm	22.309 s	30.545 s	33.493 s
A∗ algorithm	2.065 s	8.908 s	13.629 s

It can be seen from the simulation that comparing the two path planning algorithms, although the generated path is not completely consistent, the length of path is completely consistent, and both are the shortest path result. From the perspective of planning time, the planning time of the A* algorithm is much



(a) *Dijkstra* algorithm



(b) $A*$ algorithm

Fig. 3. *Dijkstra* algorithm and A^* algorithm comparison.

lower than that of the Dijkstra algorithm. Therefore, it can be concluded that the A* algorithm can greatly shorten the planning time without affecting the path planning performance.

Local Path Planner. The local path planner receives occupancy grid map generated by mapping algorithm and global path generated by the global path planner, and completes the local path planning based on the DWA, and then outputs the speed command of the tracked mobile platform movement.

DWA algorithm is a typical action sampling algorithm, which can avoid new obstacles and dynamic obstacles to solving the local obstacle avoidance problem of the robot. The DWA algorithm considers the motion constraints of the robot velocity and acceleration. It forms a velocity vector space composed of linear velocity and angular velocity, and samples the velocity space to simulate the

trajectory, and then scores the simulated trajectory through a predetermined cost function, and then selects the trajectory with the highest score as the output. Then the speed command is obtained for the difference of the trajectory. Choosing a velocity space considers the three main constraints:

- Both linear and angular velocity of robot motion are limited by the maximum and minimum velocity.
- The dynamic performance of the motor is limited, and the motor power that drives the robot is limited.
- Restrictions on braking safety distances.

The kinematic model of the tracked mobile platform in this paper is adopted a differential drive kinematic model, i.e., it can only move forward, backward and rotate. The kinematic model is shown in Eq. (10).

$$\begin{aligned} x(t_n) &= x(t_0) + \int_{t_o}^{t_n} v(t)\cos(\theta(t))dt \\ y(t_n) &= y(t_0) + \int_{t_o}^{t_n} v(t)\sin(\theta(t))dt \\ \theta(t_n) &= \theta(t_0) + \int_{t_o}^{t_n} \omega(t)dt \end{aligned} \tag{10}$$

Under the control in several discrete time Δt, the state of the tracked mobile platform changes as follows:

$$\begin{aligned} x(t_n) &= x(t_0) + \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} \left(v\left(t_i\right) + \dot{v}_i \Delta t\right) \cos\left(\theta\left(t_i\right) + \omega(t_i)\Delta t + \tfrac{1}{2}\dot{\omega}_i(\Delta t)^2\right) dt \\ y(t_n) &= y(t_0) + \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} \left(v\left(t_i\right) + \dot{v}_i \Delta t\right) \sin\left(\theta\left(t_i\right) + \omega(t_i)\Delta t + \tfrac{1}{2}\dot{\omega}_i(\Delta t)^2\right) dt \end{aligned} \tag{11}$$

At each moment Δt, the linear and angular velocity are sampled to calculate the simulated trajectory, as shown in Fig. 4:

$$\begin{aligned} v(t_i) &\in \left\{ v_{\min}, v_{\min} + \Delta v, \cdots, v_{\max} \right\}, v_{\min,\max} = v(t_{i-1}) \mp \dot{v}_{\max}\Delta t \\ \omega(t_i) &\in \left\{ \omega_{\min}, \omega_{\min} + \Delta\omega, \cdots, \omega_{\max} \right\}, \omega_{\min,\max} = \omega(t_{i-1}) \mp \dot{\omega}_{\max}\Delta t \end{aligned} \tag{12}$$

Several trajectories generated are evaluated and scored, and the evaluation functions used are:

$$G(v,\omega) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega)) \tag{13}$$

In (13), $heading(v,\omega)$ is the azimuth evaluation term used to evaluate the gap between the end point of the trajectory and the current robot orientation. In general, choosing trajectory with smaller azimuth makes the robot walk as straight as possible. $dist(v,\omega)$ is the trajectory distance evaluation term that represents the distance between the trajectory and the nearest obstacle. A higher value indicates that the trajectory is safer $dist(v,\omega)$ is the speed evaluation term. α,

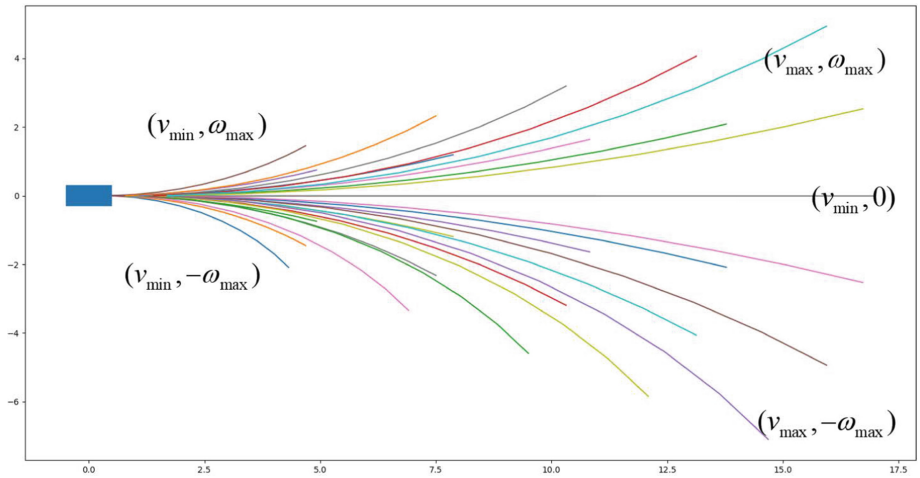


Fig. 4. Simulated trajectory after velocity and angular velocity sampling.

β, and γ are the weighting coefficients of the three evaluation term. The physical meaning is that it is expected that the mobile platform can avoid obstacles, face the target, and drive at a high speed.

DWA's algorithm flow can be summarized as: The simulation results are shown in Fig. 5.

Algorithm 2: DWA algorithms

Input: Start, End.
Output: Speed control.

```
Initialization (maximum and minimum velocity of mobile platform, evaluation
function weight, etc.);
while True do
    if Arrived then
        Reach the target point, return;
    end
    Calculate the speed range of the current sample (dynamic window);
    Traverse all v,ω and simulate trajectories;
    The evaluation function is scored;
    Select the optimal v, ω and send it to the mobile platform;
    move;
end
```

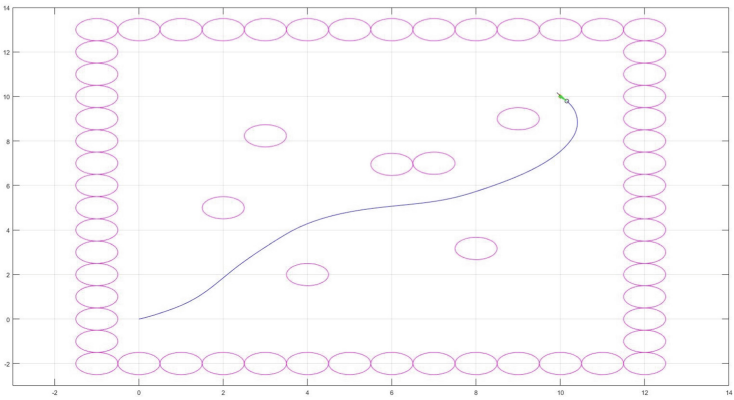


Fig. 5. DWA algorithm simulation.

4 Autonomous Stair Climbing Algorithm

In this paper, we assume a scenario that the robot needs to go to the next floor after exploring one floor. Arriving near the stairs, once the depth camera carried by the robot detects the nearby stairs, the robot judges the stairs data output from the stair detection algorithm. If the length, width, and height of the stairs meet the conditions for robot climbing, the robot is navigated directly in front of the first step of the stairs, completes the stairs climbing preparatory posture, and then begins to climb the stairs autonomously.

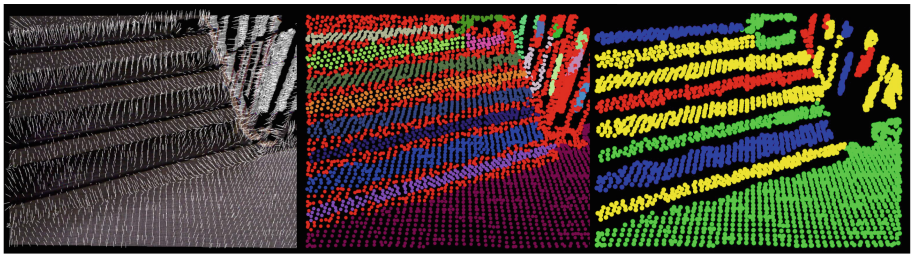
This section mainly introduces the stair detection algorithm and the robot climbing strategy.

4.1 Stair Detection Algorithm

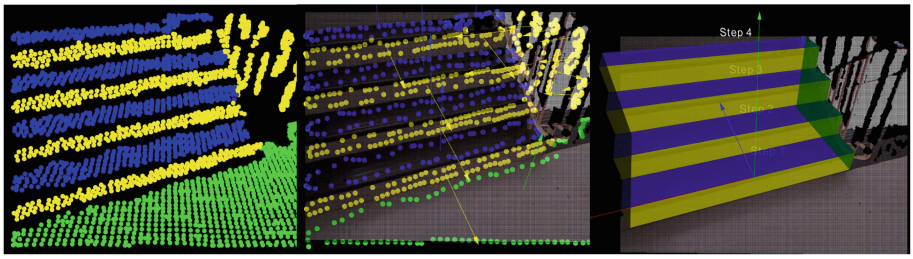
The stair detection algorithm bases on the point cloud photos taken by the Kinect depth camera, and still uses ROS as a framework to process the point cloud data by using the Point Cloud Library (PCL).

The point cloud data taken by the Kinect camera is too large, so in order to avoid affecting the calculation speed of the algorithm, it is necessary to carry out pre-processing first. The voxel filtering and statistical filtering are performed on the point cloud data to compress them, reduce computation load, and smooth the surface by eliminating the outlier points.

After preprocessing the point cloud data, the stair detection algorithm is courried out next. First, the point cloud photos are segmented to divide all the planar regions of the scene, and then these planar regions are classified and the planar regions identified as “steps” are modeled. Specifically, the algorithm is as follows (Fig. 6):



(a) Normal estimation (b) Region growth (c) Planar test



(d) Plane clustering (e) Plane classification (f) Stair modeling

Fig. 6. Point cloud processing methods.

Normal Estimation. The normal estimation [11] bases on local surface fitting method. In the point cloud, for any point p_i, its neighborhood $p_j \subset Nbhd(p_i)$ is obtained by the KnnSearch method (K is set to 16 in this paper), and the centroid of the neighborhood is calculated:

$$o_i = \frac{1}{k} \sum_{j=0}^{k} p_j \tag{14}$$

After obtaining the centroid of the neighborhood, the covariance matrix is constructed as follows:

$$\text{cov} = \frac{1}{k} \sum_{p_j \subset Nbhd(p_i)} (p_j - o_i) \cdot (p_j - o_i)^T \tag{15}$$

The SVD decomposition is performed on the covariance matrix, and the eigenvector corresponding to the obtained smallest eigenvalue is the normal n_i of the point.

Region Growth. The region is obtained by using the region growth method. The region growth method starts with a seed, which is the point with the minimum curvature, and then expands the region towards neighboring points with small angle between the normals and similar curvature values. The new seeds are updated as the neighboring points that meet the normal and curvature thresholds. And repeat this process until the region can no longer expand. Then, a new

initial seed is selected among the remaining points, and the process starts again until the regions are smaller than a certain threshold. Based on this method, the point cloud image is divided into several regions, which is convenient for subsequent detection and classification of each region.

Planar Test. The detected object are steps, floors and walls that all of which are flat, so the planar test is required for several areas divided by using the regional growth method. Due to the work way of the region growth algorithm, most regions are highly flat, but it cannot be ruled out that the regions are a surface with a small curvature. The RANSAC algorithm [3] is used to perform the plane test by finding the largest plane in each region, if the number of outer points is less than a threshold, it will be considered a plane and the fitted plane equation is obtained.

Plane Clustering. According to the position relationship of each region plane, the plane clustering is performed. If the difference in plane normal vectors between two regions is small, the vertical distance is small, and their boundaries are adjacent, it is considered as a planar region. At the same time, according to the direction of the normal vector of the plane, only the horizontal and perpendicular planes to the ground are retained, and other planes are regarded as obstacles without performing the subsequent plane classification.

Plane Classification. The existing regions are all horizontal or vertical planes, which are divided into different categories according to the normal vector and relative position of the planes. The planes that is close to 0° to the ground are preliminarily defined as "steps", and the planes that are close to 90° to the ground are preliminarily defined as "wall". And then these regions are further divided through their position relations.

Areas with a plane height of approximately zero are classified as floors, and areas with positive or negative heights are classified as steps. This paper sets the height range of the step to the minimum $H_{\min} = 13$ cm and maximum $H_{\max} = 18$ cm, with a allowable measurement error $H_{tol} = 3$ cm, and the plane height above the ground $H_{\min} - H_{tol} = 10$ cm is regarded as the candidate plane for the step.

After having a step candidate plane, it is necessary to establish a connection between each step, which can eliminate non-step planes and also allow the steps to form a hierarchy. The strategy of this paper is to analyze each step from the bottom up:

– The first is to screen out the first step and determine whether the plane height is between $H_{\min} - H_{tol} = 10\,\text{cm}$ and $H_{\max} + H_{tol} = 18$ cm, and meet this condition as the first step. If there are multiple planes that are satisfied, connectivity of that plane to the upper and lower layers is required.

– Once the first step is in place, the algorithm selects the remaining step candidates by height and begins testing connectivity and height conditions to determine whether they belong to the current layer or to the new layer.
– When all the candidates have been checked, if the number of floors is greater than 1, the algorithm starts modeling the stairs and outputs the stair parameters (length, width and height).

Stair Modeling. This paper refers to the method used in [10] to model the stair. The method makes full use of the geometric relationship of each plane to constrain the length, width and height of the stairs, uses the PCL library to draw and display the stair model, at the same time output the parameters of the stair.

4.2 Stair Climbing Strategy

Through the stair detection algorithm in the previous section, the length, width and height $(L_{stair}, W_{stair}, H_{stair})$ of the stairs and the coordinates of the center point of the first stair in the depth camera coordinate system (P_{stair}^{Camera}) can be obtained. After obtaining the stair information, If Eq. (16) is satisfied, robot can begin to climb the stairs (Fig. 7).

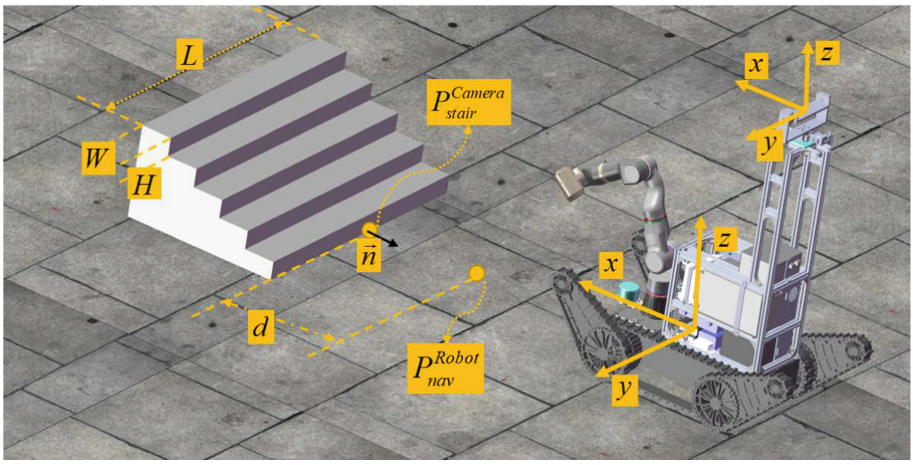


Fig. 7. Simulation scene graph.

$$\begin{cases} \theta_{stair} = {}^{H_{stair}}/_{W_{stair}} < \theta_{\max} = 40^{\circ} \\ L_{stair} > 1.5 W_{robot} \\ H_{stair} < H_{\max} = 50\,\text{cm} \end{cases} \tag{16}$$

If the detected stair meets the climbable criteria, a navigation point P_{nav}^{Robot} is published, the robot navigates to the target point and enters the state of readiness to climb the stairs. The navigation point is determined by the following equation:

$$P_{nav}^{Robot} = T_{Camera}^{Robot} P_{nav}^{Camera} = P_{stair}^{Camera} + d\vec{n} \tag{17}$$

where P_{nav}^{Robot} is the coordinate of the navigation point in the robot's coordinate system; P_{nav}^{Camera} is the coordinate of the navigation point in the camera coordinate system; P_{stair}^{Camera} is the coordinates of the first stair in the camera coordinate system; $\vec{n}$ is the unit normal vector of the vertical plane of the first stair in the camera coordinate system; T_{Camera}^{Robot} represents the transformation matrix from the camera coordinate system to the robot coordinate system; d represents the distance directly in front of the stairs where the navigation point is located.

After the robot enters the preparatory climbing point, it begins to climb the stairs, the process is shown in Fig. 8.

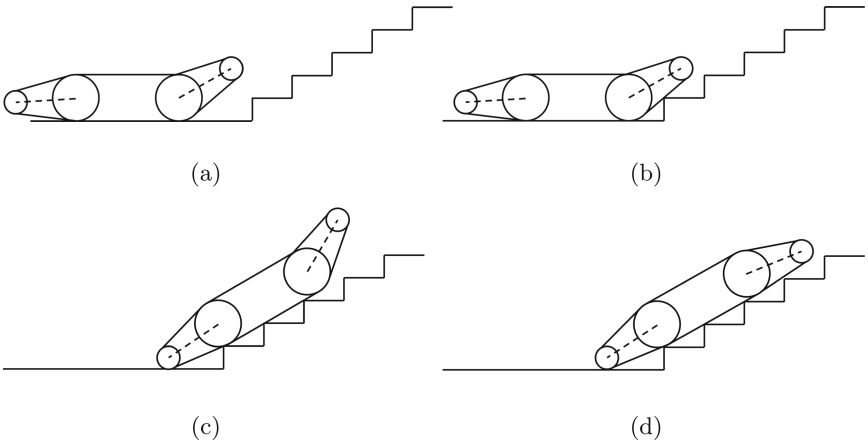


Fig. 8. Preparation and ascent phases.

As shown in Fig. 8(a), the robot navigates to the front of the stairs and adjusts the inclination angle of the flippers to the inclination angle of the stairs; The robot advances, reaches the state of Fig. 8(c), the whole robot cling to the steps, at this time the robot inclination angle is the angle of the stairs, exceeding the threshold angle, and the front flipper returns to the horizontal position; As shown in Fig. 8(d), the robot's flipper are placed horizontally, increasing the length of the robot on the stairs to improve the stability of the robot's climbing.

During the landing state, the control of the front flipper let the robot to land smoothly after crossing the last step. The strategy adopted in this paper is using current loop to control the front flipper motor during the landing stage, and the

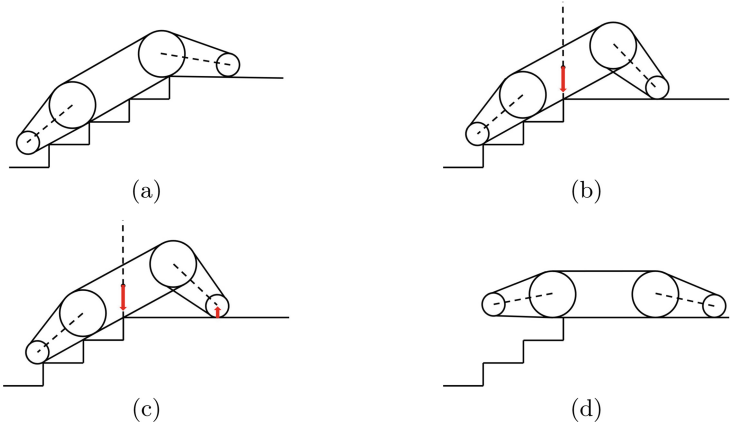


Fig. 9. Landing phase.

advantage is the front flipper can naturally sag to make the robot's center of gravity as far forward as possible before the robot lands; At the moment of landing, the front flipper is close to the ground to avoid impact on the robot, and at the same time, it can be slowly rotated to make the robot land smoothly.

As shown in Fig. 9(a), when the stair detection algorithm cannot detect the stairs, it means that the robot is about to enter the landing state, and the front flipper enters the damping mode and begins to hold the ground. Equation (18) gives the expected current of the front flipper motor, so that the front flipper can just overcome the friction of the reducer and rotate to ensure that the front flipper can always be close to the ground. After crossing the equilibrium point, the damping of the front flipper can effectively reduce the robot's falling speed.

$$I_{ref} = \frac{(T_f + \Delta T) - D\dot{q}}{K_T N} \tag{18}$$

where, K_T is the coefficient of electric current torque, N is the deceleration ratio of the reducer, T_f is the friction torque of the flipper, ΔT is the margin after the offset torque overcomes the friction torque, $\dot{q}$ is the angular velocity of the flipper, and D is the damping coefficient.

5 Experiment

The experimental part of this paper mainly simulates the detection process of the robot on one floor (SLAM mapping and navigation) and the process of climbing the stairs to the next floor after discovering stairs.

5.1 SLAM Mapping and Navigation Experiment

The experimental scene is in a circular corridor outside the laboratory, where the robot will perform 2D SLAM mapping based on cartographer. The experimental results are shown in Fig. 10:

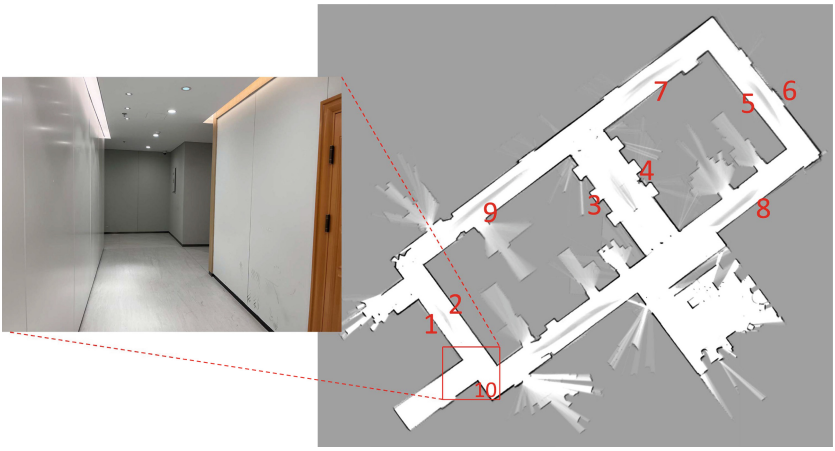


Fig. 10. *Cartographer* map and actual scene.

Table 2. *Cartographer* measurement data

Index	Figure measured value/m	The measured value/m	Absolute error/m	Relative error/%
1	6.01	6.18	0.17	2.75
2	9.98	10.15	0.17	1.67
3	10.04	10.15	0.11	1.08
4	10.10	10.15	0.05	0.49
5	10.00	10.15	0.15	1.48
6	13.80	13.93	0.13	0.93
7	10.38	10.63	0.25	2.35
8	10.38	10.50	0.12	1.14
9	14.65	14.86	0.21	1.41
10	2.00	1.95	0.05	2.56

From Table 2, the average relative error of 2D map of the robot system in indoor environment is 1.57% and the maximum relative error is 2.75%, and wall boundary and obstacle boundary are sharply demarcated.

After completing the SLAM mapping, the unmanned navigation operation is carried out indoors, and the experimental content is to complete the fixed-point

navigation and obstacle avoidance experiments: In the robot navigation and obstacle avoidance experiment, the robot completed the obstacle avoidance of the original obstacles and new obstacles on the map, smoothly reached the designated target point, and completed the robot's navigation and obstacle avoidance task in the indoor environment (Fig. 11).

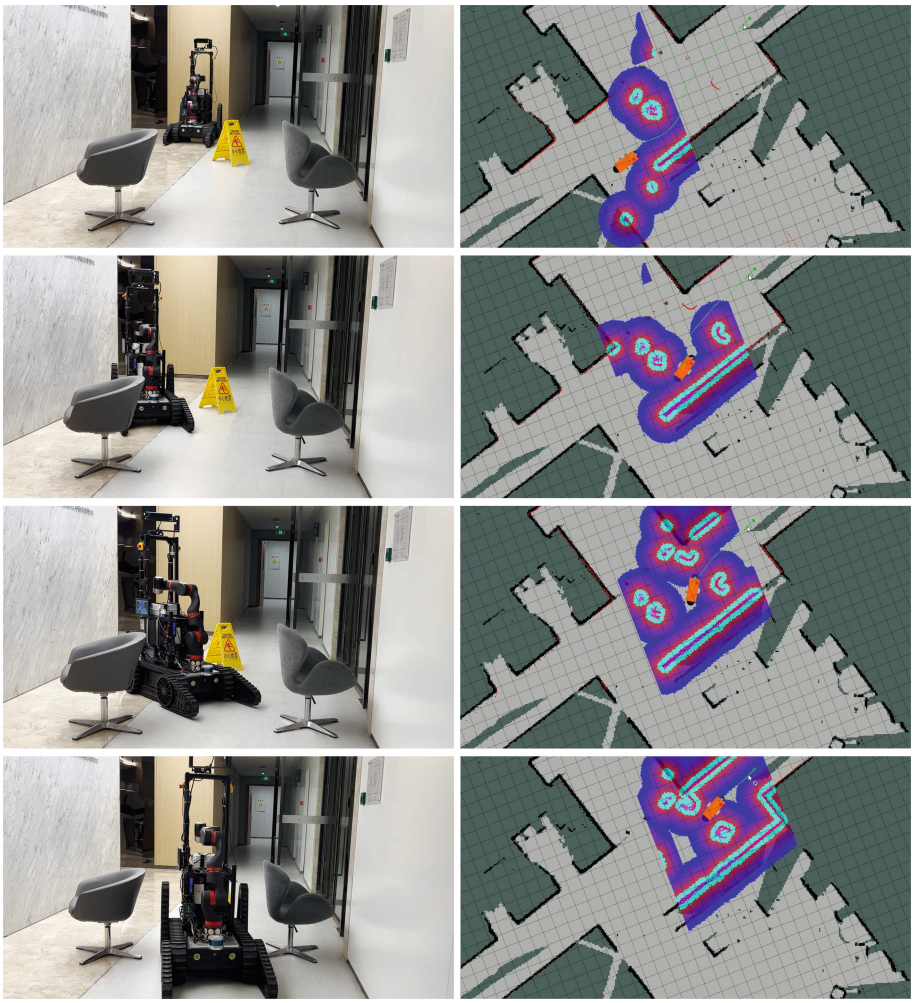


Fig. 11. Navigation and obstacle avoidance experiment diagram.

In this experiment, the robot completes the SLAM mapping of the corridor in the indoor environment, and can complete the trajectory planning and autonomously navigate to the target point, which verifies the robot's ability to unmanned autonomous exploration in the indoor environment.

5.2 Stair Detection Experiment

After the robot completes the exploration of this floor, it goes to the staircase, and in the process of approaching the staircase, the stair detection algorithm detects the staircase and calculates the coordinates of the length, width and height of stairs and the center point of the stairs in the coordinate system of the camera (Fig. 12).

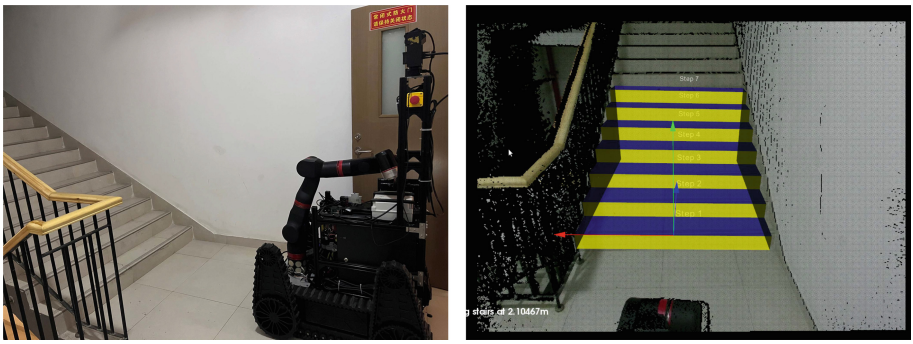


Fig. 12. Stair detection experiment.

Table 3. Stair detection data.

	Length/m	Width/m	Height/m	Angle/°
Algorithm detection value	1.645	0.287	0.161	29.29
The measured value	1.600	0.270	0.150	29.05
Absolute error	0.045	0.017	0.011	0.24
Relative error	2.81	6.29	7.33	0.83

It can be seen from Table 3 that the absolute error between the detected stair parameters and the actual stair parameters is within 5 cm, the relative error is within 8%, and the calculated stair inclination error is 0.24°, and the relative error is only 0.83%, indicating that the stair detection algorithm in this paper has high accuracy.

5.3 Stair Climbing Experiment

The stair parameters obtained by the stair detection algorithm meet the condition of Formula (16), that is, the condition of climbing the stairs, and the robot can climb the stairs autonomously. The robot navigates to the preparation point of climbing the stairs and starts to climb the stairs. The experimental process is shown in Fig. 13, 14:

Experiments show that the autonomous stair climbing algorithm based on four-flipper tracked mobile robot proposed in this paper can effectively complete the function of autonomous stair climbing (Fig. 15).

Fig. 13. Preparation and ascent phases.

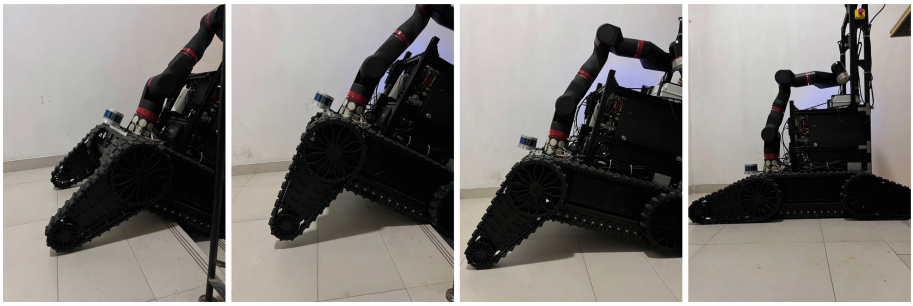


Fig. 14. Landing phase.

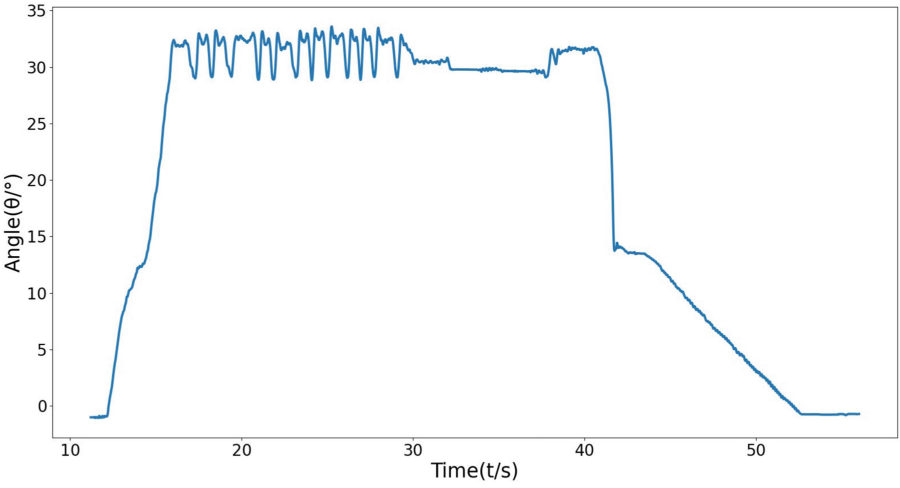


Fig. 15. Angle of robot.

6 Summarize

In this paper, a tracked rescue robot system is designed for the scene of building rescue, which can be targeted to complete the unmanned autonomous detection and rescue in the indoor environment, and can detect the stairs to complete the task of autonomous stair climbing. Based on the cartographer algorithm, the SAM mapping in indoor environment is realized by using lidar and IMU, and the autonomous navigation function of tracked mobile robot is realized based on ROS system. The stair detection algorithm in literature [10] is improved and applied to the detection of stairs by mobile robots during driving, which has higher accuracy and stability. Finally, as the robot fit with the stair detection algorithm and the robot navigation algorithm, a strategy for climbing stairs on the four-flipper tracked robot platform is proposed, which realizes the function of the tracked mobile robot to climb the stairs autonomously. The robot system designed in this paper provides a new tool for the search and rescue work in case of building fire, and plays a role in protecting people's lives and property.

References

1. Chen, B., Wu, J., Wang, F., Yang, D., Zhang, W.: Motion planning for autonomous climbing stairs for flipper robot. In: 2020 IEEE International Conference on Real-time Computing and Robotics (RCAR), pp. 531–538 (2020). https://doi.org/10.1109/RCAR49640.2020.9303039
2. Deng, Y., Shan, Y., Gong, Z., Chen, L.: Large-scale navigation method for autonomous mobile robot based on fusion of GPS and lidar SLAM. In: 2018 Chinese Automation Congress (CAC), pp. 3145–3148 (2018). https://doi.org/10.1109/CAC.2018.8623646
3. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
4. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robot. Autom. Mag. **4**(1), 23–33 (1997)
5. Guo, J., Shi, J., Zhu, W., Wang, J.: Approach to autonomous stair climbing for tracked robot. In: 2017 IEEE International Conference on Unmanned Systems (ICUS), pp. 182–186 (2017). https://doi.org/10.1109/ICUS.2017.8278337
6. Hess, W., Kohler, D., Rapp, H., Andor, D.: Real-time loop closure in 2D lidar SLAM. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1271–1278 (2016). https://doi.org/10.1109/ICRA.2016.7487258
7. Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., Vincent, R.: Efficient sparse pose adjustment for 2D mapping. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 22–29 (2010). https://doi.org/10.1109/IROS.2010.5649043
8. Olson, E.B.: Real-time correlative scan matching. In: 2009 IEEE International Conference on Robotics and Automation, pp. 4387–4393 (2009). https://doi.org/10.1109/ROBOT.2009.5152375
9. Perez-Yus, A., Gutiérrez-Gómez, D., Lopez-Nicolas, G., Guerrero, J.: Stairs detection with odometry-aided traversal from a wearable RGB-D camera. Comput. Vis. Image Underst. **154**, 192–205 (2017)

10. Pérez-Yus, A., López-Nicolás, G., Guerrero, J.J.: Detection and modelling of staircases using a wearable depth sensor. In: Agapito, L., Bronstein, M.M., Rother, C. (eds.) ECCV 2014. LNCS, vol. 8927, pp. 449–463. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16199-0_32
11. Qin, X.-J., Hu, Z.-T., Zheng, H.-B., Zhang, M.-Y.: Surface reconstruction from unorganized point clouds based on edge growing. Adv. Manuf. **7**(3), 343–352 (2019). https://doi.org/10.1007/s40436-019-00262-5