Simulation Research on Automatic Navigation of Indoor Wheelchair

Peng Shi[1,2], **Yizhun Peng**[1,2,*]

[1] *College of Electronic Information and Automation, Tianjin University of Science and Technology, Tianjin, 300222, China,*

[2] *Advanced Structural Integrity International Joint Research Centre, Tianjin University of Science and Technology, Tianjin, 300222, China*

E-mail: pengyizhun@tust.edu.cn*

www.tust.edu.cn

Abstract

A new auto-navigation wheelchair based on ROS system is proposed to deal with the global aging and the low behavioral ability of the elderly. Lidar is used to locate and map the active area using gmapping algorithm. Real-time map information is transmitted to the processor by the camera and lidar working together. Automatic navigation is completed by A* algorithm calculation. Important information points are marked by QR code and precisely positioned by camera recognition, which enables wheelchair to have automatic navigation function. It can help older people move safer and more freely at home; It can also be applied to nursing homes to reduce the pressure of nurses and centralize management of the elder.

Keywords: wheelchair, ROS, Gmapping, A algorithm*

1. Introduction

According to the World Bank, by the end of 2019, nearly 654.6 million elderly people were aged worldwide, accounting for 9% of the total population. With the increase of the total number of the elderly, it has become more important to pay attention to the quality of life of the elderly, among which, the travel convenience of the elderly is the premise of their high quality of life. As a frequently used tool for the elderly, its convenience is particularly important. Long-term use of traditional manual wheelchairs will have a huge burden on people's wrists and arms; electric wheelchairs need operators to pay attention to the surrounding environment for a long time, pay attention to their own control, easy to produce fatigue and then cause accidents.

For the above problems, we will design an automatic navigation wheelchair robot built around the ROS[1] platform. In the indoor environment, the surrounding environment can be perceived through lidar and multiple sensors. Establish a model in gazebo for simulation, and use the Gmapping[2] algorithm in the SLAM[3] algorithm to create a graph.[4] Then display in rviz simulation. Navigation using the A* algorithm[5] allows the wheelchair for stable autonomous driving in a simulated indoor environment.[6]

2. Model Building

For the authenticity of the simulation experiments, wheelchair models need to be imported into the Gazebo simulation environment. The SolidWorks software provides plugins to automatically export URDF files with physical shape, collision parameters, and inertia parameters, which can be used directly by the Gazebo simulation environment. Therefore, SolidWorks software was selected to build the physical model of the wheelchair. To enable the wheelchair to complete SLAM and navigation, add lidar and camera to its pedal position and upper backrest. The model established by SolidWorks is shown in Fig. 1.

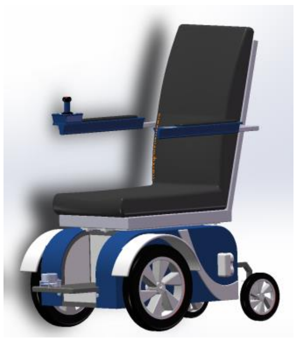


Fig. 1. The model established by SolidWorks

In SolidWorks only has physical models, and in order to complete simulating the real simulation environment in Gazebo, it is also necessary to add wheel drive plugin, differential drive control plugin, and sensor plugin to the established physical model. The simulations of the full model in Gazebo are shown in Fig. 2.

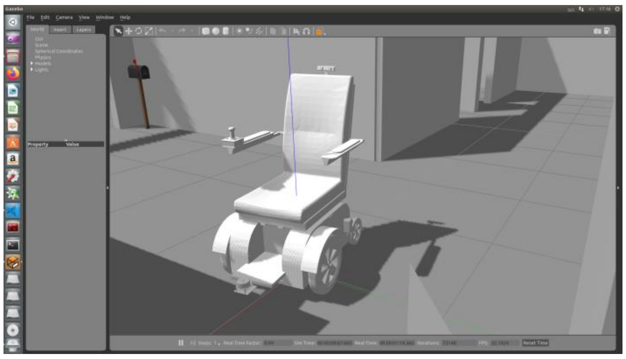


Fig. 2. The simulations of the full model in Gazebo

2.1. *Gazebo simulation environment construction*

The robot model is shown in the Gazebo. But currently by default, the robot model in Gazebo is in empty world, and there are no emulations similar to rooms, furniture, roads, trees.... There are three ways to create a simulation implementation in Gazebo:

- Add built-in components directly to create a simulation environment,
- Manual drawing of the simulation environment,
- Download using the official or third-party improved simulation environment plugins directly.

The simulation selection manually drew the simulation environment. A resthome environment with an area of 200m^2 was drawn with parts of furniture added to the environment to simulate the real nursing home environment. The established Gazebo simulation environment is shown in Fig. 3.

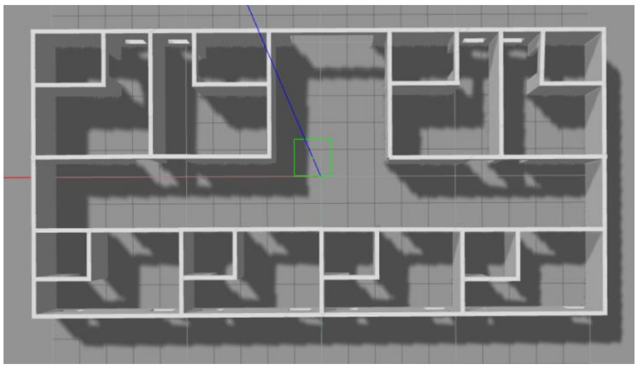


Fig. 3. The established Gazebo simulation environment

3. Navigation Preparation

3.1. *Slam*

SLAM, also known as CML, is Concurrent Mapping and Localization, or simultaneous localization and mapping. The question can be described as whether putting a robot in an unknown position in an unknown environment, is there a way to let the robot move and gradually draw a complete map of the environment. The so-called a complete map refers to every corner of the room without obstacles.

Gmapping is one of the more commonly used and relatively mature SLAM algorithms in the ROS open-source community. Gmapping can draw a two-dimensional grid map according to the mobile robot odometry data and laser data. Correspondingly, Gmapping also has certain requirements for hardware:

- The mobile robot can post an odometry message,
- Robots need to release radar messages.

Frist, write the launch file related to the Gmapping node and start the Gazebo simulation environment. Then start the mapping launch file and start the keyboard control node to control the robot movement. Add components to display the grid map in Rviz. Finally, the robot movement can be controlled in the Gazebo through the keyboard, and the grid map data released by the Gmapping can be displayed in Rviz. The display of the grid map data in Rviz is shown in Fig. 4.

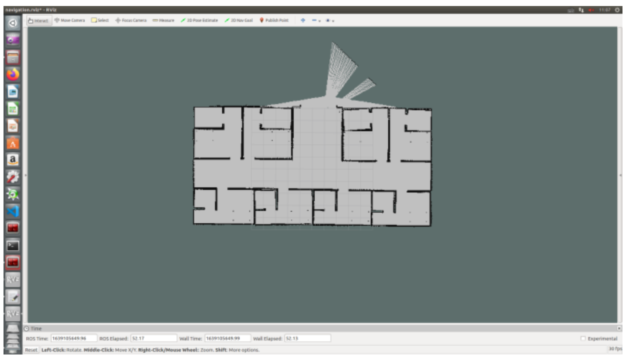


Fig. 4. The display of the grid map data in Rviz

3.2. *Robot localization*

localization is to calculate the location of the robot itself in the global map. Of course, SLAM also includes localization algorithm implementation, but SLAM localization is used to build a global map, belongs to the stage before the beginning of navigation, and the current localization is used for navigation. In the navigation, the robot needs to move according to the set route. Through localization, it can be judged whether the actual trajectory of the robot meets expectations. The AMCL feature package is provided in the ROS navigation feature package for enabling robot localization in navigation. AMCL is a probabilistic localization system for 2D mobile robots that implements an Adaptive Monte Carlo localization method to calculates the robot position using particle filters based on existing maps.

3.3. *Coordinate transformation*

The odometry itself can also assist the robot in localization, but there are odometry cumulative errors and some special cases. AMCL can improve thelocalization accuracy by estimating the robot's posture in the map coordinate system, and then combining with the odometry. The odometry itself can also assist the robot in localization, but the odometry has accumulation errors and localization errors may occur in some special circumstances. AMCL can improve the localization accuracy by estimating the robot's posture in the map coordinate system, combined with the odometry. Odometry localization and AMCL map localization is shown in Fig. 5. The simulation of the AMCL algorithm in Rviz is shown in Fig. 6.

- Odometry localization: Only coordinate transformation between /odom_frame and /base_frame via odometry data,
- AMCL map localization: You can provide coordinate transformations between /map_frame, /odom_frame, and /base_frame.

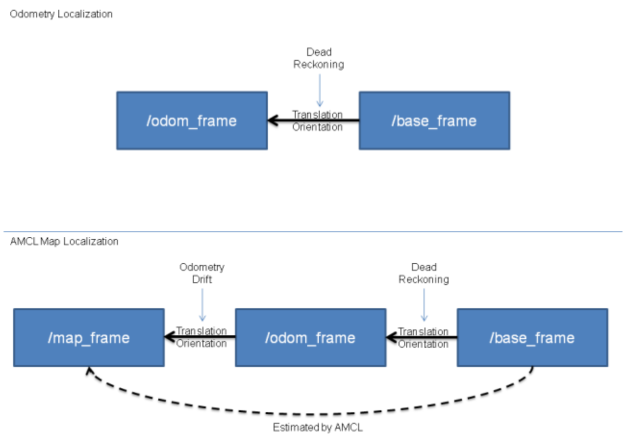


Fig. 5. Odometry localization and AMCL map localization

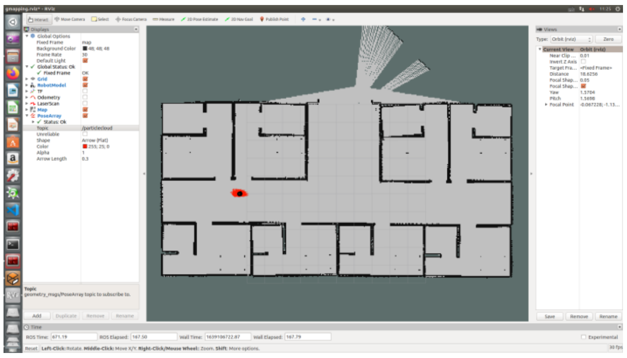


Fig. 6. The simulation of the AMCL algorithm in Rviz

4. Navigation Path Planning

4.1. *Introduction to the move_base*

Undoubtedly, path planning is one of the core functions in navigation. The move_base function package is provided in the navigation function package set of ROS to realize this function. The move_base function package provides an action-based path planning implementation.

move_base can control the robot chassis to move to the target position according to the given target point, and continuously feedback the robot's own posture and the status information of the target point during the movement. move_base is mainly composed of global planner[7]and local planner.[8]

4.2. *Costmap*

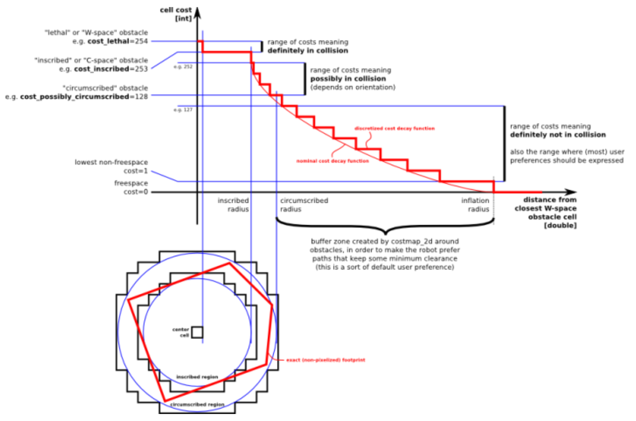
Robot navigation relies on a map. The map in ROS is actually a picture. This picture has metadata such as width, height, and resolution. The gray value is used in the picture to indicate the probability of obstacles. However, the map constructed by SLAM cannot be used directly in navigation, because the map constructed by SLAM is a static map. In the navigation process, the obstacle information is changeable. The obstacle may be removed or new obstacles may be added. Obtain the obstacle information from time to time during navigation. It is best to set a warning on the edge of the obstacle on the map. In the area, try to prohibit robots from entering. Therefore, static maps cannot be directly applied to navigation. On top of it, some auxiliary information needs to be added to the map, such as obstacle data obtained from time to time, and data such as inflation layer added based on static maps.

There are two cost maps: global_costmap and local_costmap. The former is used for global planner, and the latter is used for local planner. Both cost maps can be stacked in multiple layers, and generally have the following layer:

- Static map layer: Static map built by the SLAM,
- Obstacle map layer: The obstacle layer tracks the obstacles as read by the sensor data,
- Inflation layer: Inflate on the above two layers to avoid the robot from hitting obstacles,
- Other layers: Other layers can be implemented and used in the costmap via pluginlib.

4.3. *Collision algorithm*

Inflation is the process of propagating cost values out from occupied cells that decrease with distance. For this purpose, we define 5 specific symbols for costmap values as they relate to a robot. Collision algorithm is show Fig. 7.



cell cost [int]
"lethal" or "W-space" obstacle e.g. cost_lethal=254
"inscribed" or "C-space" obstacle e.g. cost_inscribed=253
"circumscribed" obstacle e.g. cost_possibly_circumscribed=128
e.g. 252
e.g. 127
lowest non-freespace cost=1
freespace cost=0
range of costs meaning definitely in collision
range of costs meaning possibly in collision (depends on orientation)
range of costs meaning definitely not in collision
also the range where (most) user preferences should be expressed
discretized cost decay function
nominal cost decay function
inscribed radius
circumscribed radius
inflation radius
distance from closest W-space obstacle cell [double]
buffer zone created by costmap_2d around obstacles, in order to make the robot prefer paths that keep some minimum clearance (this is a sort of default user preference)
center cell
inscribed region
circumscribed region
exact (non-pixelized) footprint

Fig. 7. Collision algorithm

- "Lethal" cost means that there is an actual (workspace) obstacle in a cell. So if the robot's center were in that cell, the robot would obviously be in collision,
- "Inscribed" cost means that a cell is less than the robot's inscribed radius away from an actual obstacle. So the robot is certainly in collision with some obstacle if the robot center is in a cell that is at or above the inscribed cost,
- "Possibly circumscribed" cost is similar to "inscribed", but using the robot's circumscribed radius as cutoff distance. Thus, if the robot center lies in a cell at or above this value, then it depends on the orientation of the robot whether it collides with an obstacle or not. We use the term "possibly" because it might be that it is not really an obstacle cell, but some user-preference, that put that particular cost value into the map. For example, if a user wants to express that a robot should attempt to avoid a particular area of a building, they may inset their own costs into the costmap for that region independent of any obstacles. Note, that although the value is 128 is used as an example in the diagram above, the true value is influenced by both the inscribed_radius and inflation_radius parameters as defined in the code,
- "Freespace" cost is assumed to be zero, and it means that there is nothing that should keep the robot from going there,
- "Unknown" cost means there is no information about a given cell. The user of the costmap can interpret this as they see fit,
- All other costs are assigned a value between "Freespace" and "Possibly circumscribed" depending on their distance from a "Lethal" cell and the decay function provided by the user.

4.4. *parameter setting*

In the simulation, it may happen that the robot enters the expansion area when the local planner does not conform to the global planner and appears to "feign death." In order to avoid this situation as much as possible. Through the physical model of the robot itself, the parameters set by the global planner and the local planner can be changed. In this way, in the global planner, the planner will be as far away from the obstacles as possible, and in the local planner, even if the robot deviates from the global planner, it will retain more free space between the obstacles, thereby avoiding the "feign death" situation.

Set the maximum speed in the x direction in the basic local planner parameters to 0.3 m/s, configure the expansion radius of the global costmap to 0.3m, and the expansion radius of the local costmap to 0.1m. Set the size of the local cost map to 3m * 3m * 3m. Set the obstacle perception range to 2m, and eliminate the obstacle range after it is greater than 2.5m.

5. Conclusion

First start the Gazebo simulation environment; load the launch file related to the startup navigation; load the Rviz component with the added configuration data. Navigate by setting the destination through the 2D Nav Goal on the Rviz toolbar. The action trajectory is shown in Fig. 8. The green line represents the global planner, and the red line represents the local planner. For the obstacles that appear on the global planner route, the simulation will avoid the obstacles through local planner and guide the robot to reach the target point. Obstacle placement is shown in Fig. 9. Local planner avoid obstacles is shown Fig. 10. Reaching the final target point is shown Fig. 11.

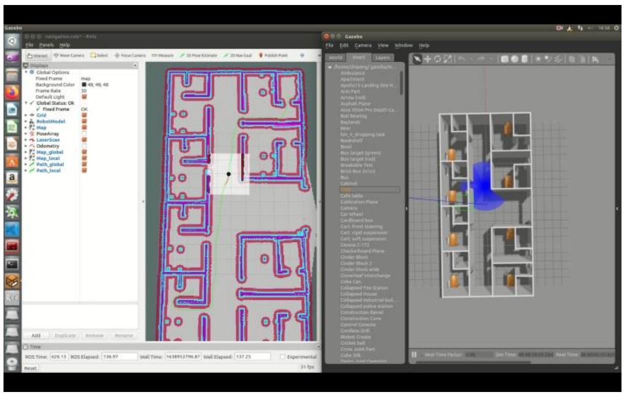


Fig. 8. The action trajectory

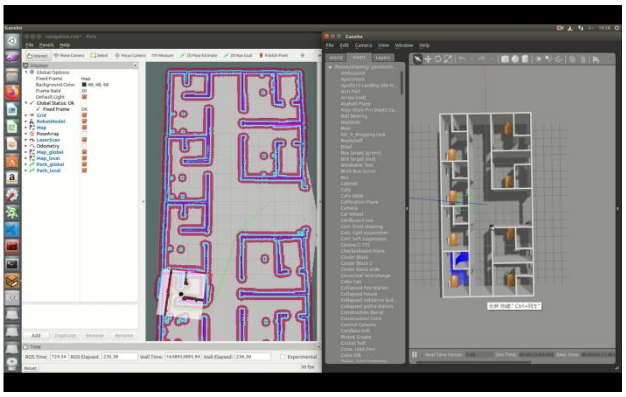


Fig. 9. Obstacle placement

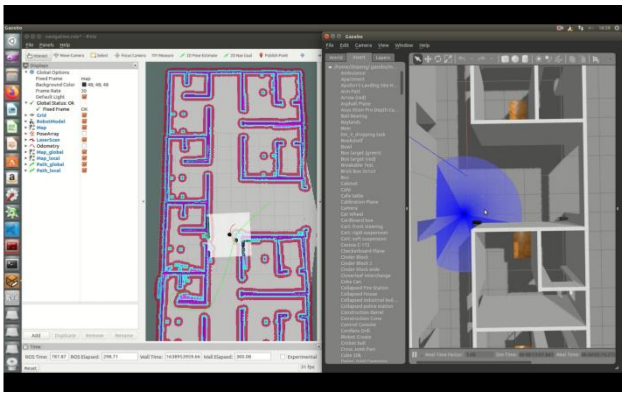


Fig. 10. Local planner avoid obstacles

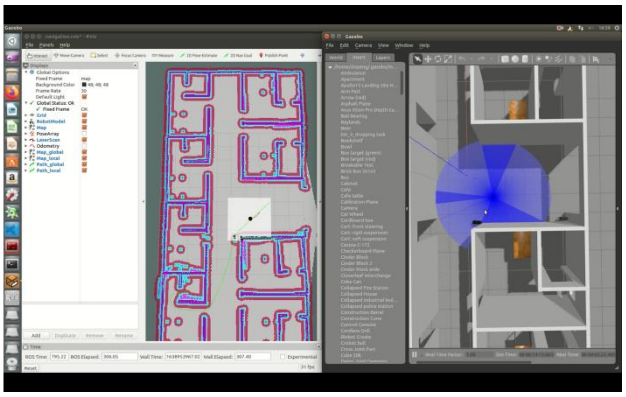


Fig. 11. Reaching the final target point

References

1. Koubâa, Anis, ed. *Robot Operating System (ROS)*. Vol. 1. Cham: Springer, 2017.

2. G. Grisetti, C. Stachniss and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2432-2
3. Y. Abdelrasoul, A. B. S. H. Saman and P. Sebastian, "A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM," *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, 2016, pp. 1-6
4. W. Xiaoyu, L. Caihong, S. Li, et al. "On Adaptive Monte Carlo Localization Algorithm for the Mobile Robot Based on ROS," *2018 37th Chinese Control Conference (CCC), 2018*, pp. 5207-5212.
5. J. Röwekämper, C. Sprunk, G. D. Tipaldi, et al.On the position accuracy of mobile robot localization based on particle filters combined with scan matching, *Intelligent robots and systems*,2012:3158-3164.
6. Gilli é ron, Pierre-Yves, et al. "Indoor navigation performance analysis." *Proceedings of the 8th European Navigation Conference GNSS*. No. CONF. 2004.
7. Huijuan Wang, Yuan Yu and Quanbo Yuan, "Application of Dijkstra algorithm in robot path-planning," *2011 Second International Conference on Mechanic Automation and Control Engineering, 2011*, pp. 1067-1069.
8. Eduardo J. Molinos, Ángel Llamazares, Manuel Ocaña, Dynamic window based approaches for avoiding obstacles in moving, Robotics and Autonomous Systems, Volume 118, 2019, Pages 112-130, ISSN 0921-8890.

Authors Introduction

Mr. Peng Shi

He received a bachelor's degree in automation from Tianjin University of Science and Technology in 2019. He currently holds a master's degree in electronic information from Tianjin University of Science and Technology.

Dr. Yizhun Peng

He is an Associate Professor in Tianjin University of Science & Technology. He received a doctor's degree in control theory and control engineering from the Institute of Automation,Chinese Academy of Sciences, in 2006. His research field is intelligent robot and intelligent control.