

Final Competition Report

11812904 张靖明

11812112 吴迅

Introduction

The final experiment took the form of a competition. We need to go to the designated parking spot within the fixed map for parking or go to the designated location for AR Tag identification. The map of the competition is shown below:

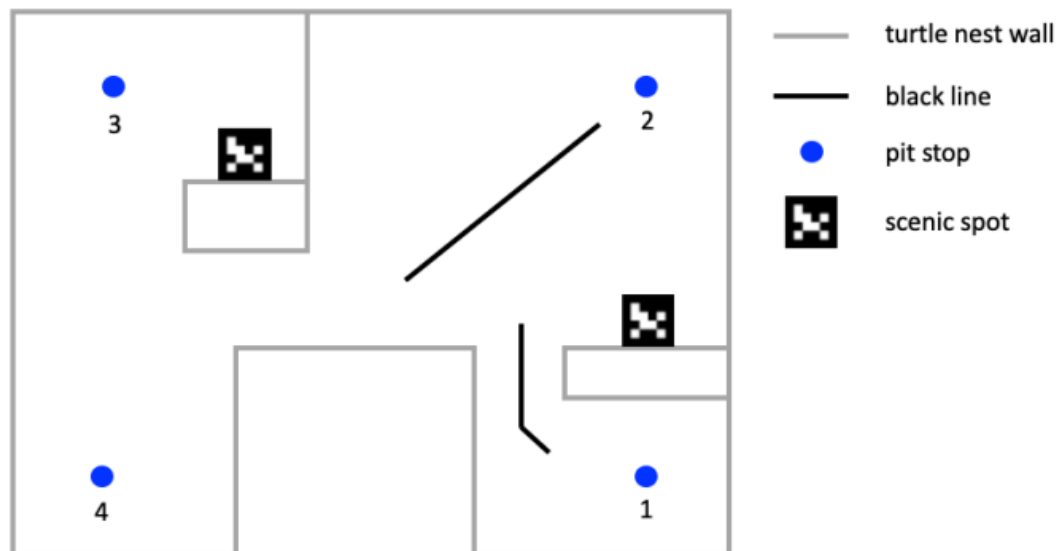


Fig 1 planar graph of competition site

In task 1, the robot starts from position 1, goes to each parking point in the order of 1234, and completes the identification of two AR tags in the process. There will be penalties if the robot does not touch the sign while parked, or if it touches a wall while driving. The black line in the picture is the sign of the assistant robot. Task 2 is similar to Task 1 in that you have to complete different tasks in a given amount of time. The more tasks you complete, the higher the score.

In this experiment, we mainly used GMapping to construct the map and then arrived at the destination through navigation. ROS provided us with almost all the toolkits we needed, and our task was just to adjust the parameters so that the system could achieve better results. AR Tag recognition is carried out by using ROS toolkit after camera calibration, and a sound program is written by ourselves.

We used a turtlebot3-burger to complete the race. Turtlebot3 is a programmable robot based on ROS, including camera, laser radar, gyroscope, speaker and other modules, which can realize SLAM, navigation and other functions. As shown in Figure 2:

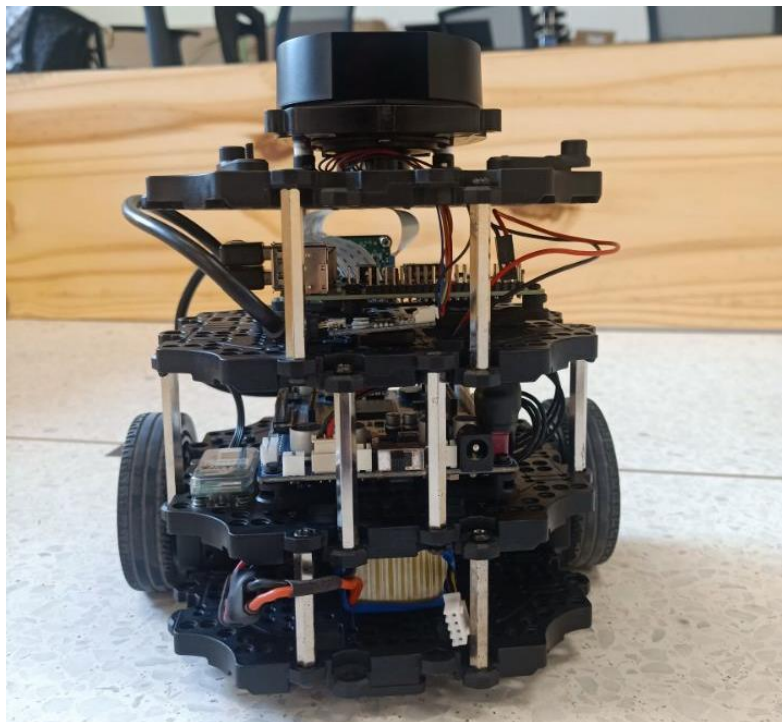


Fig 2 Turtlebot3-burger

Approach

Mapping

In this lab, Gmapping is used to create map and to do localization. Gmapping package provides laser-based SLAM(Simultaneous Localization and Mapping). It creates a ROS node called slam_gampping.

This node subscribes Scan (sensor_msgs/LaserScan) message and tf message. Laser scanning creates a map from it. Relevant framework required conversion laser, the reference and ranging information in tf message..

This node Published map_metadata, map and entopy. Map_metadata.

Do Gmapping, the command is:

```
roslaunch _slam turtlebot3_slam.launch
```

First, do mapping using the initial settings in Gmapping, the map result is as Fig shows.

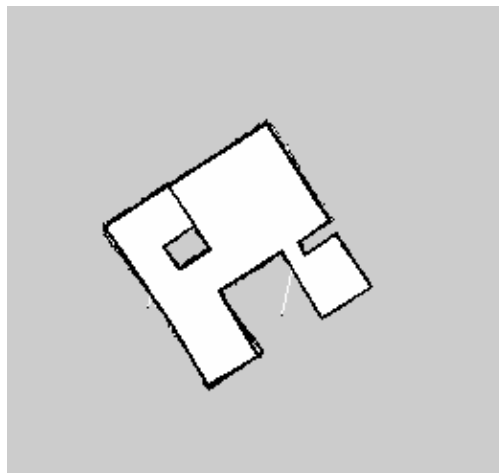


Fig 3. Map with initial setting

When do gmapping, parameters are useful to create an accurate map. Find and open the parameter setting of Gmapping as Fig shows.

Xmin, ymin, xmax and ymax is map size settings. Delta is map resolution and particles is number of particles in filter. These parameters affect the map quality and speed of creating a map. Set particles bigger takes more time to estimate the localization of the robot but more concrete. Setting delta from 0.05 to 0.01 makes the points in green smaller, then the obstacle can be draw more clearly.

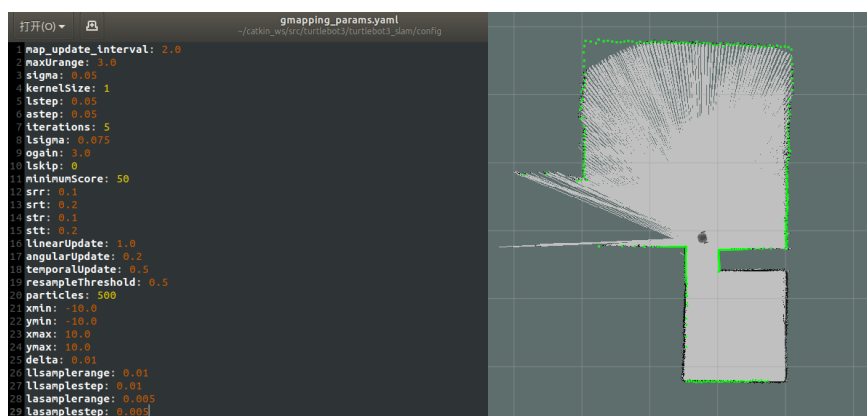


Fig 4. Gmapping Parameters and Gmapping process

Using these new setting, several tries have been done before we finally create a perfect map. It implies some problems. When particles number is larger, robot can't move fast as before during mapping process. If it moves too fast, the Gmapping process has fewer time to estimate the scan data. The boundary will get errors. At the same time, small delta(resolution) means precise boundary which needs more data. With small resolution, the boundary near robot can be estimated successfully, while the far boundary has information loss as Fig shows. There are empty spaces in far boundary

while near boundary is continuous. So, with small resolution, robot needs to move closer to the boundary to get more data.

The final map is as following Fig, with resolution of 0.01 and particle number 500. The boundary become more precise compared to original one. This map takes more effect to create, too.

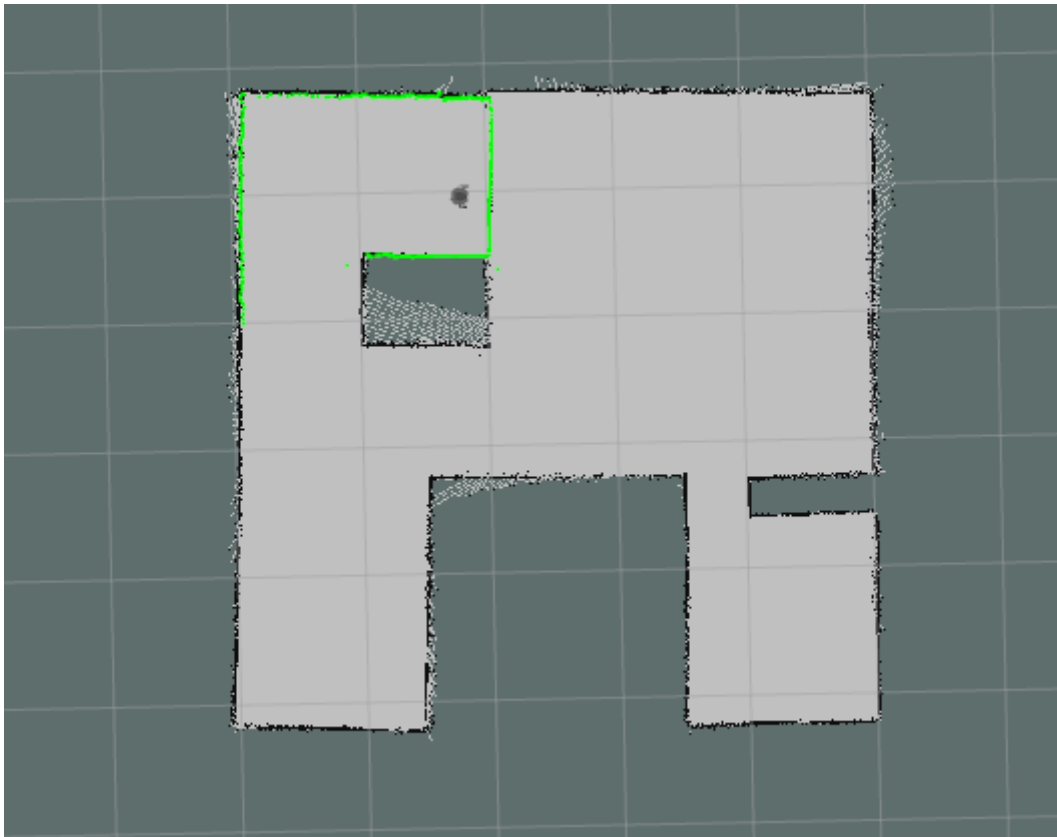


Fig 5. Map created by Gmapping process

This is part of our robot `rqt_graph`. The node and message information are just as described earlier. Gmapping node subscribes scan message and tf message. Gmapping publishes message to rviz and tf node. So, Gmapping does localization and mapping at the same time.

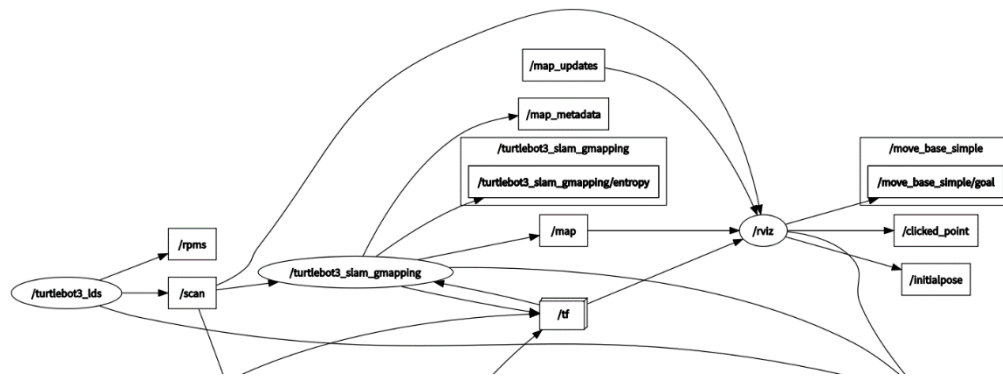


Fig 6. Rqt_Graph of gmapping

Navigation

After creating map of the environment, put robot into it. Now, start to navigate robot to specific destination.

First, do it in rviz by hand. The command is:

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
map_file:=$HOME/map.yaml
```

map_file is direction to the map that already created in previous step Gmapping.

Run the command, the following rviz window come out.

First, pose estimation needs to done to guarantee the robot's location and direction is same as the original pose when creating the map.

This Fig shows robot pose after pose estimation.

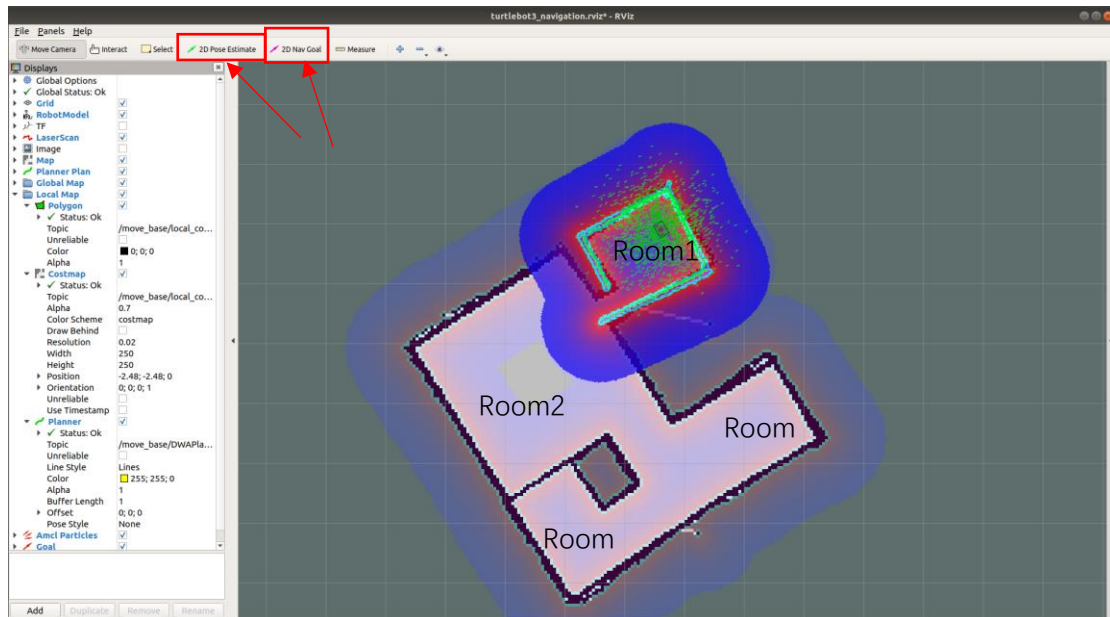
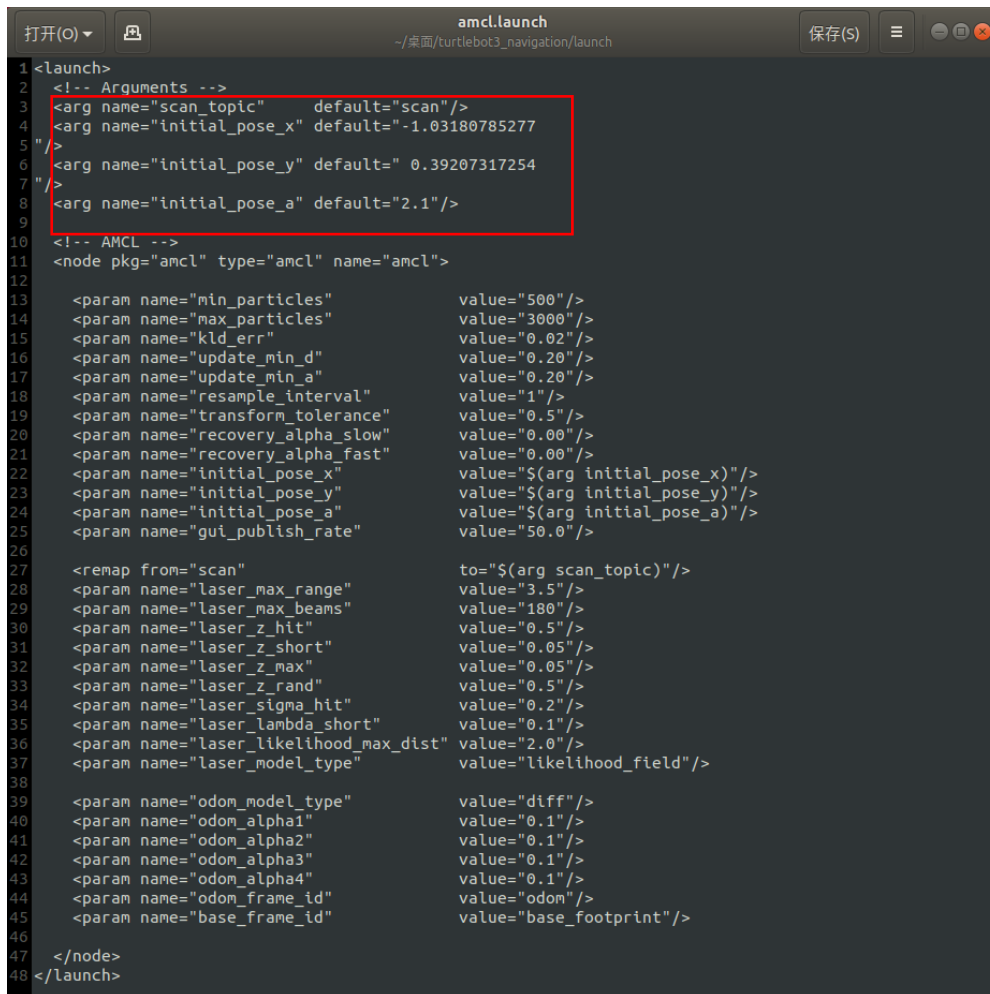


Fig 7. RVIZ working window

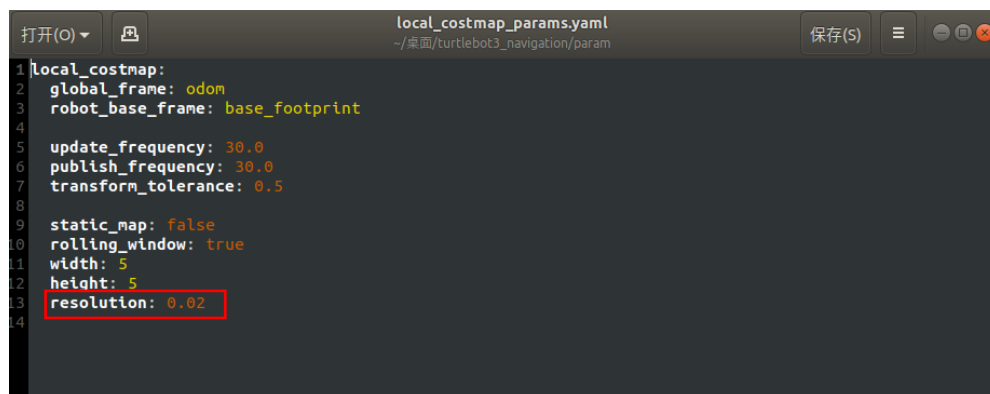
In fact, we can change initial parameters in `amcl.launch` file. There are three parameters: `initial_pose_x`, `initial_pose_y`, `initial_a`. Change these values to your initial pose parameters then every time start from the same location and direction.



```
1 <launch>
2 <!-- Arguments -->
3 <arg name="scan_topic" default="scan"/>
4 <arg name="initial_pose_x" default="-1.03180785277" />
5 </>
6 <arg name="initial_pose_y" default="0.39207317254" />
7 </>
8 <arg name="initial_pose_a" default="2.1"/>
9 </>
10 <!-- AMCL -->
11 <node pkg="amcl" type="amcl" name="amcl">
12
13 <param name="min_particles" value="500"/>
14 <param name="max_particles" value="3000"/>
15 <param name="kld_err" value="0.02"/>
16 <param name="update_min_d" value="0.20"/>
17 <param name="update_min_a" value="0.20"/>
18 <param name="resample_interval" value="1"/>
19 <param name="transform_tolerance" value="0.5"/>
20 <param name="recovery_alpha_slow" value="0.00"/>
21 <param name="recovery_alpha_fast" value="0.00"/>
22 <param name="initial_pose_x" value="$(arg initial_pose_x)"/>
23 <param name="initial_pose_y" value="$(arg initial_pose_y)"/>
24 <param name="initial_pose_a" value="$(arg initial_pose_a)"/>
25 <param name="gui_publish_rate" value="50.0"/>
26
27 <remap from="scan" to="$(arg scan_topic)"/>
28 <param name="laser_max_range" value="3.5"/>
29 <param name="laser_max_beams" value="180"/>
30 <param name="laser_z_hit" value="0.5"/>
31 <param name="laser_z_short" value="0.05"/>
32 <param name="laser_z_max" value="0.05"/>
33 <param name="laser_z_rand" value="0.5"/>
34 <param name="laser_sigma_hit" value="0.2"/>
35 <param name="laser_lambda_short" value="0.1"/>
36 <param name="laser_likelihood_max_dist" value="2.0"/>
37 <param name="laser_model_type" value="likelihood_field"/>
38
39 <param name="odom_model_type" value="diff"/>
40 <param name="odom_alpha1" value="0.1"/>
41 <param name="odom_alpha2" value="0.1"/>
42 <param name="odom_alpha3" value="0.1"/>
43 <param name="odom_alpha4" value="0.1"/>
44 <param name="odom_frame_id" value="odom"/>
45 <param name="base_frame_id" value="base_footprint"/>
46
47 </node>
48 </launch>
```

Fig 8. Amcl launch file in navigation

Local_costmap is used to calculate and navigate the path. First, we use the original setting of resolution equal to 0.05. The output is awful, it can hardly pass the first door to outer space (room 2). Then we try several resolution values, finally 0.02 gets best performance.



```
1 local_costmap:
2   global_frame: odom
3   robot_base_frame: base_footprint
4
5   update_frequency: 30.0
6   publish_frequency: 30.0
7   transform_tolerance: 0.5
8
9   static_map: false
10  rolling_window: true
11  width: 5
12  height: 5
13  resolution: 0.02
14
```

Fig 9. Local_costmap_param

The navigation process will accumulate error in whole process. The longer distance it runs, the bigger errors would come up. This error can be reduced by pose estimation by hand during the navigation process.

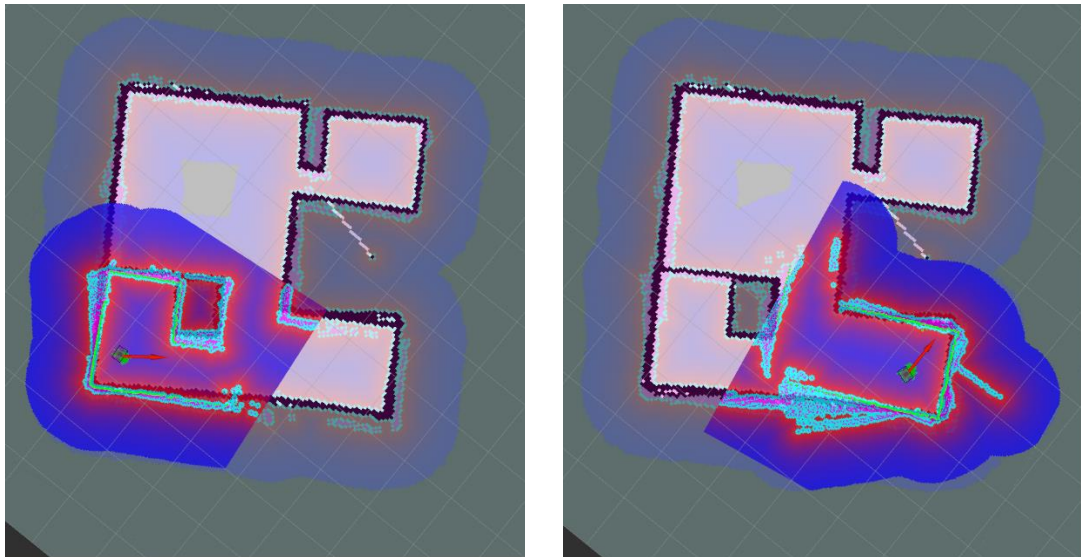


Fig 10. Error during the navigation process

As we can use rviz to navigate by hand now, a simple python script can achieve navigation in code.

The script is as Fig shows. Set seven destination points in the Map using pose information.

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Pose, Point, Quaternion
from move_base_msgs.msg import MoveBaseAction, MoveBaseActionGoal

def set_destination_points():
    # Set seven destination points in the Map using pose information
    locations = [
        Pose(Point(-2.211882886, 1.789118881, 0.000), Quaternion(0.000, 0.000, -0.9811981819, 0.1838772677)),
        Pose(Point(-2.297978829, 2.394232876, 0.000), Quaternion(0.000, 0.000, -0.9848232899, 0.1849222781)),
        Pose(Point(-4.812544721, 0.977921881, 0.000), Quaternion(0.000, 0.000, -0.9776288708, 0.2181113251)),
        Pose(Point(-4.378912781, 1.748429252, 0.000), Quaternion(0.000, 0.000, 0.2981951471, 0.9567716791)),
        Pose(Point(-0.318867629, 2.326484877, 0.000), Quaternion(0.000, 0.000, -0.9472212187, 0.3682899915)),
        Pose(Point(-1.786474839, 0.6428678161, 0.000), Quaternion(0.000, 0.000, -0.9795367018, 0.8771264813)),
        Pose(Point(-1.393365787, -1.484701132, 0.000), Quaternion(0.000, 0.000, -0.9716879788, 0.9311311323))
    ]

    # Initialize MoveBaseAction
    move_base = rospy.get_param('~move_base')
    move_base_goal = MoveBaseActionGoal()

    # Set the destination points
    move_base_goal.action.sequence = locations

    # Send the goal
    move_base_client.send_goal(move_base_goal)

    # Wait for the goal to be completed
    rospy.wait_for_message('~move_base', 'MoveBaseActionResult', 10)

    # Print the result
    print "Navigation completed successfully"

if __name__ == '__main__':
    rospy.init_node('set_destination_points')
    set_destination_points()
```

Fig 11. Simple Navigation script

Then use send_goal function to send location information to move_base. In this function, we initialize a MoveBaseGoal() object, and then assign values to the parameters of the target_pose member function of

the object, where pose is the coordinate point of the navigation. The name of the global coordinate system and the current timestamp are set in the header. Finally, use the `send_goal` function of `move_base` to send the parameters set in the object to `global_planner` to start path planning.

```
def send_goal(self, locate):  
    self.goal = MoveBaseGoal()  
    self.goal.target_pose.pose = self.locations[locate]  
    self.goal.target_pose.header.frame_id = 'map'  
    self.goal.target_pose.header.stamp = rospy.Time.now()  
    self.move_base.send_goal(self.goal) #send goal to move_base
```

Fig 12. Send_goal function

Now run these commands in sequence:

At raspberry:

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

At PC:

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
```

```
map_file:=$HOME/map.yaml
```

```
roslaunch simple_navigation_goals simple_navigation_goals.py
```

Robot start to navigates to seven destination points automatically.

Navigation videos was uploaded to bilibili, where you can see the GitHub README for link.

AR Tag Recognition and Vocalization

AR Tag recognition is based on monocular Raspberry Pi camera.

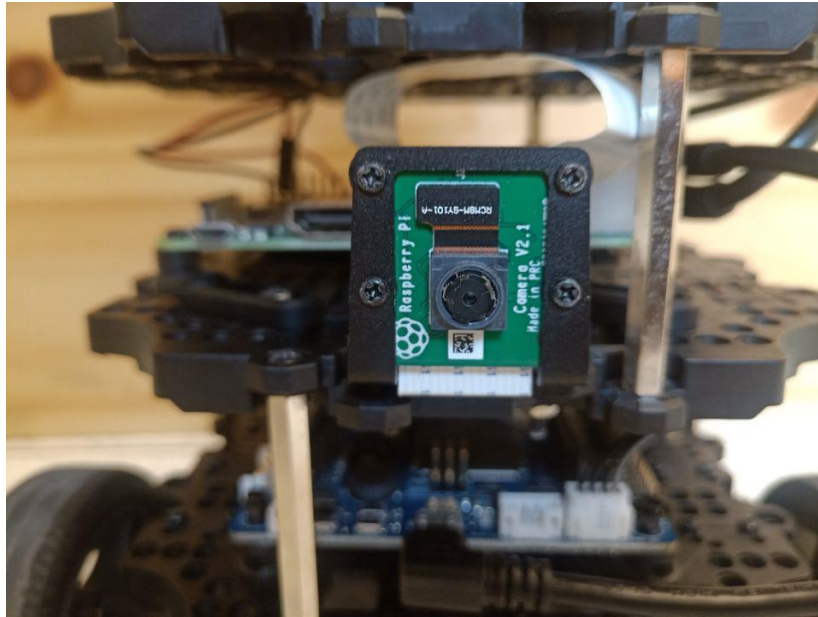


Fig 12. Raspberry Pi camera

We first calibrate the camera. The relationship between the three-dimensional geometric position of a point on the surface of a space object and its corresponding point in the image is determined by the geometric model of camera imaging, and these geometric model parameters are camera parameters. In most cases, these parameters can only be obtained by experiment and calculation. Therefore, good camera calibration is the premise of good follow-up work.(The toolkit of ROS was used to collect sample data and calculate correction parameters).

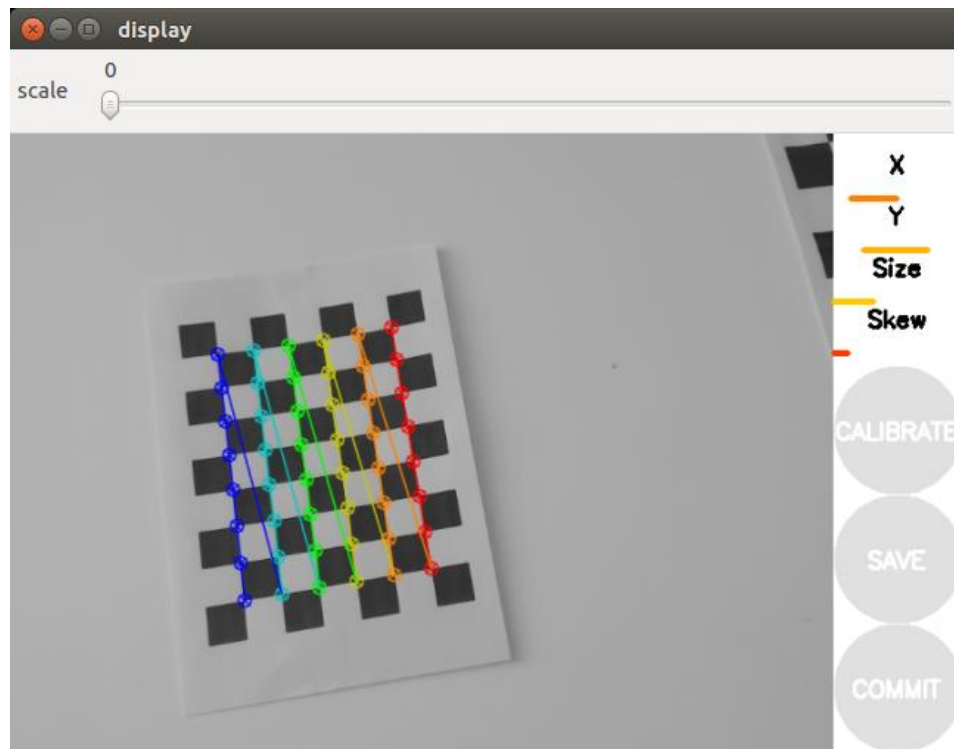


Fig 12. Camera calibration process

Then the AR Tag is identified with the calibrated camera. The following figure is the parameters calibrated by the camera:

```
[image]

width
640

height
480

[narrow_stereo]

camera matrix
563.314313 0.000000 309.506805
0.000000 564.005549 241.352215
0.000000 0.000000 1.000000

distortion
-0.056512 0.053653 0.011955 -0.007954 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
559.857788 0.000000 304.336871 0.000000
0.000000 561.949219 246.279239 0.000000
0.000000 0.000000 1.000000 0.000000
```

Fig 13. Camera calibration param

The `camera_matrix` specifies the internal parameter matrix of the camera.

The `distortion_coefficients` specify the coefficients of the distortion model.

`Rectification_matrix` is a rectifying matrix and is generally an identity Matrix.

`Projection_matrix` is the projection matrix of the external world coordinates to the image plane.

A common AR Tag shape is shown below. The pattern represents a number. The figure below is the pattern produced after encoding the number 1:

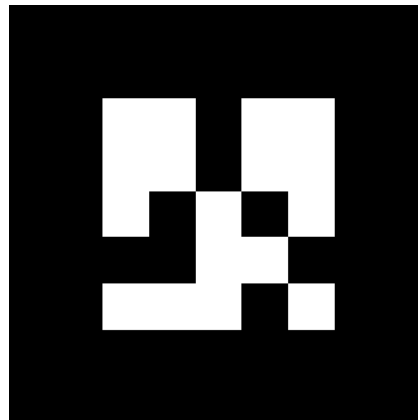


Fig 14. AR Tag

It is a black and white rasterized image, the smallest unit is a small square, the entire AR Tag is 10*10 grid, the edge is composed of 2 units thick black square, the center is 6*6 valid data area, black 0, white 1, which represents a 36-bit binary number. Of these 36 bits, 10 represent valid data and the other 26 bits are used for fault-tolerant coding. In this experiment,

we directly call the ROS toolkit for identification.

The `ar_pose_marker` node gets data from the AR Tag and is displayed by `data.markers[0].id`, which is the node that identifies the AR Tag. Then we call the vocalizing node, and set the number of vocalizing cycles according to the result of recognition. The main program and callback functions are as follows:

```
52 def callback(data):
53     id = data.markers[0].id
54     distance = data.markers[0].pose.pose.position.z
55     rospy.loginfo('I heard id=%d and distance=%f', id, distance)
56
57     soundhandle = SoundClient()
58     rospy.sleep(1)
59
60     stop_pub = rospy.Publisher('chatter', String, queue_size=1)
61     rate = rospy.Rate(60)
62     stop_msg = "run"
63     rospy.loginfo('Playing sound %d.', id)
64     for k in range(id):
65         soundhandle.play(1, 1)
66         rospy.sleep(0.5)
67
68     rospy.loginfo(stop_msg)
69     stop_pub.publish(stop_msg)
70     rospy.sleep(1)
71
72 def listener():
73
74     rospy.init_node('listener', anonymous=True)
75
76     rospy.wait_for_message('ar_pose_marker', AlvarMarkers)
77
78     rospy.Subscriber('ar_pose_marker', AlvarMarkers, callback, queue_size=1)
79
80     rospy.spin()
81
82 if __name__ == '__main__':
83     listener()
```

Fig 15. Code of vocalizing node

Result and Conclusion

In this competition, we used GMapping for map construction, Navigation and AR Tag identification, and used Turtlebot3-Burger robot to successfully complete the tasks of fixed sequence fixed-point parking and

Tag identification. Two people in our team were promoting the project at the same time, and each contributed almost 50%. In this experiment, we still did a lot of things not well enough. For example, our robot always adjusted its posture repeatedly when parking. Even if the adjustment time was greatly reduced after optimization, it still wasted a lot of time. In the process of Tag recognition, the same Tag will be repeated to sound, which is not so intelligent. We will continue to discuss the problems related to system optimization, and the process of discussion is also the process of summarizing the project and making continuous progress.