Report for EE346 – mobile robot navigation and control

11711902 陈昊皓　　11711915 潘知渊

1. Theory

1.1 ROS

ROS is the abbreviation of robot operating system. ROS is a highly flexible software architecture used to write robot software programs. It contains a large number of tools, library codes and protocols. It aims to simplify the process of creating complex and robust robot behaviors across robot platforms. ROS is based on a graphical architecture, so that processes of different nodes can accept, publish and aggregate all kinds of information (such as sensing, control, state, planning, etc.).

1.2 AR code detection

Camera calibration is the first step in AR code recognition. There are two main sources of common camera imaging error. The first is the sensor manufacturing error, such as sensor imaging unit is not square, sensor skew; The second is the error caused by lens manufacturing and installation. Generally, the lens has nonlinear radial distortion; The lens is not parallel to the sensor of the camera, which may cause tangential distortion. Camera calibration aims to correct the distortion of image.

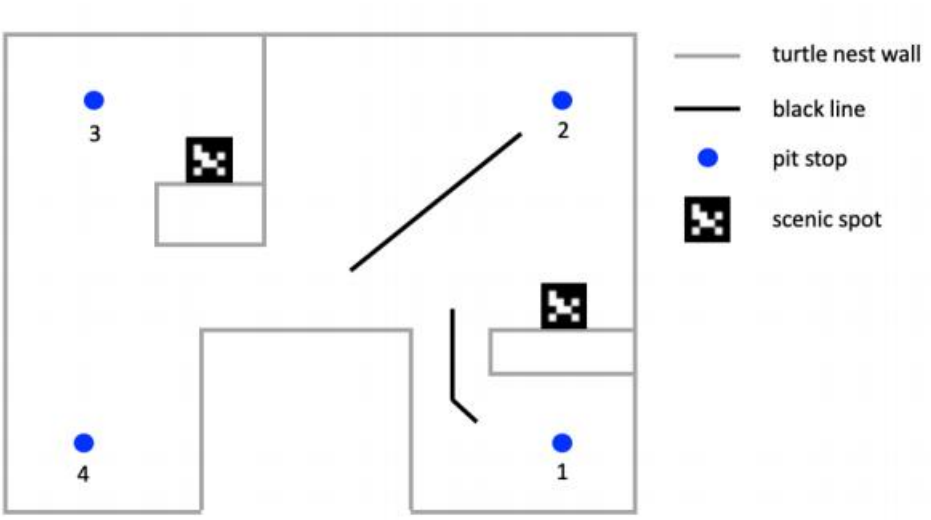
After calibration, with the aid of AR in ar_track_alvar package identifies ar code. ar_track_alvar will publish ar_pose_marker information, which is in AlvarMarkers format.

1.3 Gmapping and AMCL

AMCL (Adaptive Monte Carlo localization) is a probabilistic localization system for robot in the process of two-dimensional movement. Particle filter is used to track the robot's pose in known map, and it works well for large-scale local localization. The idea of particle filter is to assume that a handful of particles are evenly scattered in the map space at the beginning, and move the particles according to the motion of the robot. The position of each particle is simulated as a sensor information, which is compared with the observed sensor information (usually laser), thus giving each particle a probability. After that, the particles are regenerated according to the probability of generation. The higher the probability, the greater the probability of generation. After this iteration, all the particles will slowly converge together, and the exact position of the robot will be calculated.

2. Experiment

2.1 Rules of the competition



The experiment will be conducted in the walled environment shown above that has been set up in the lab. Four blue circles represent locations (pit stops or PS's) where the robot needs to make a stop. Each pit stop is exactly 50cm from the two adjacent walls. A successful stop requires your robot to touch the circle with any part of its body. There are also two black lines that lead your robot through (1) the narror channel from the room of PS1 to the room of PS2, and (2) the interior area of the largest room of the environment where LiDAR data may not be reliable for localization. These black lines are provided for the purpose of assisting your robot to move from one room to another, and you do not need to use them. Finally, there are AR code markers (scenic spots or SS's) placed somewhere on two walls and their numbers are not known a priori. The SS recogntion task requires robot to detect the AR code and make an audial signal that describes the AR code.

The competition will consist of two rounds. In the first round, robot should start from PS1 and visit SP2, SP3, and SP4 in turn, as well as

searching for the two SS's and reporting/signaling their numbers when they are found. Recognition of the SS's must be performed without external assistance (e.g., placing a black background behind the marker).

In the second round, robot will have a total of three minutes to compete. Also starting at PS1, it can attempt any of the SP's and/or SS's in turn, to earn points. Robot can attempt SP's only or SS's only, but it must alternate tasks between the left half and the right half of the turtle nest, e.g., SP2 followed by SP3 or SS1 followed by SS2. As well, robot can attempt a SS followed by any SP, and vise versa.

2.2 Overall process

(desktop) : roscore

This step is used to start the ROS framework to facilitate the use of related ROS programs later.

(pi) : roslaunch raspicam_node cameravl_1280x720.launch

This step is to activate the camera on the raspberry pie. Camera calibration has been done in previous experiments, and we modified the size of the camera to 320x180 and the frame number to 30.

(pi) : roslaunch turtlebot3_bringup turtlebot3_robot.launch

This step is to start the initialization of the turnlebot3 model on the car. The model used in the experiment is burger.

(desktop) : rosrun sound_play soundplay_node.py

This step is to complete the sound_ Initialization of the play package. Only after the initialization is completed can the command that sounds be executed.

(desktop) : rosrun ar_track_alvar pr2_indiv_no_kinect.launch

This step is to start the AR code recognition function of ROS.ar_track_alvar will publish ar_pose_marker information, which is in AlvarMarkers format.

(desktop) : rosrun sound_play playbuiltin.py

Playbuilder.py is a file we write, and its main function is to live from ar_pose_marker node receives the data identified by AR code, and uses the computer to make sound based on the AR code number.

(desktop) : roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml

This step uses our pre drawn map to initialize the car position. map_ file is where the map is saved. We modify the launch file of turbobot3_navigation, so that when running this command, the car can automatically modify its initial position.

(desktop) : roslaunch my_navigation patrol_nav.launch

This step is the core of this experiment. This step will use patrol_nav.py in the scripts of my_navigation. Based on the map information and our preset target location, this file will let the car perform the behavior of going to the target location in turn.

*(desktop) : rosluanch turtlebot3_teleop turtlebot3_teleop_key.launch

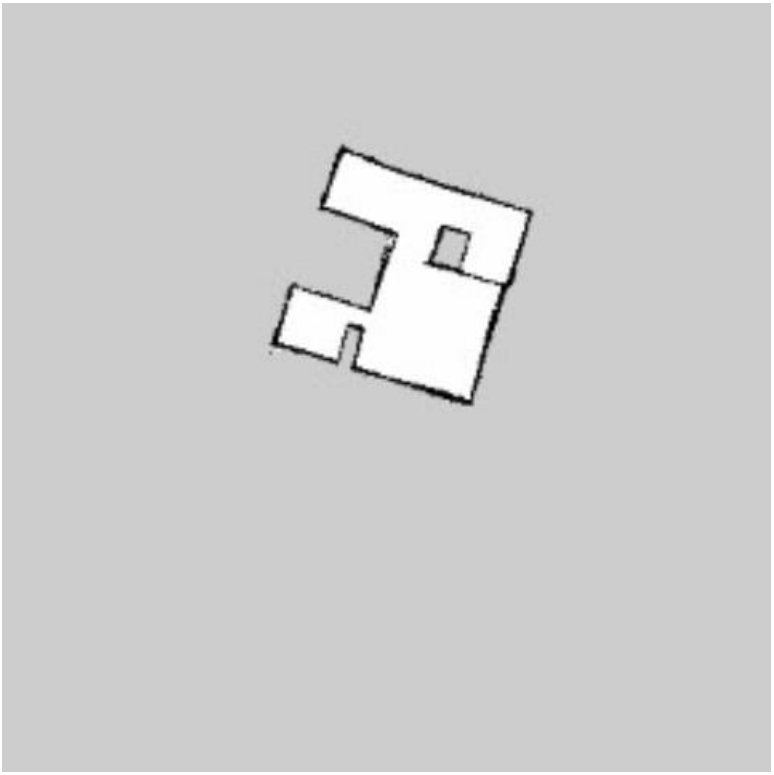
This step is through the computer keyboard control car action, in the establishment of the map.

*(desktop) : roslaunch turtlebot3_slam turtlebot3_slam.launch

This step is to build a map based on the radar information of the car, which will be built in rviz.

*(desktop) : rosrun map_server map_saver -f ~/map

After the last command is executed, the car draws a complete picture of the map. This command is used to save the map.



our map

2.3 code analysize

playbuiltin.py

The goal of this file is to receive ar node, real-time access to the car

camera to see the AR digital information, and read the number through the computer.

We start a node called "play" and wait for the node " ar_pose_marke "to publish information. The node of "play" receives the information and reads the ID information of the first marker in the information. If the digital information is obtained successfully, the soundhandle model will be called for digital voice.

The recall-back function as follows :

```
def make_sound(msg):
    try:
        marker = msg.markers[0]
        rospy.loginfo('find the marker')
    except:
        rospy.loginfo('i cannot find the marker')
        return

    myid = marker.id
    rospy.loginfo('the number is : '+str(myid))
    soundhandle.say(str(myid), 'voice_kal_diphone', 5)
    r.sleep()
```

patrol_nav.py

This document controls the car to plan for several points of automatic route. Before the competition, we first measured the global position of PS and AR points based on the map drawn. The post information of PS point is that the car steps on the PS point accurately, and at the same time, it can see the position and angle of AR code as much as possible. Ar point is that the car can read ar code information accurately and stably. By recording the global position, the car can plan the route before the competition and complete the sequential progress of multiple points.

```
self.locations['PS1'] = Pose(Point(1.3167, -0.1066, 0.000), Quaternion(0.000, 0.000, 0.987815, 0.155627))
self.locations['AR1'] = Pose(Point(-0.3476, 0.2827, 0.000), Quaternion(0.000, 0.000, 0.991664, 0.128843))
self.locations['PS2'] = Pose(Point(2.6036, 3.5758, 0.000), Quaternion(0.000, 0.000,-0.760863, 0.648911))
self.locations['AR2'] = Pose(Point( 2.3106, 2.7577, 0.000), Quaternion(0.000, 0.000, 0.987931, 0.154893))
self.locations['PS3'] = Pose(Point(-1.2825, 4.7572, 0.000), Quaternion(0.000, 0.000, 0.975706, 0.219082))
```

This file will be combined with this node "move_base" to control the speed and direction of the car.

```
self.move_base = actionlib.SimpleActionClient("move_base",
```

```
MoveBaseAction)
self.move_base.wait_for_server(rospy.Duration(30))
```

The following function is for a given position, car automatic path planning, and according to the planned route to move, and finally move to the target point.

```
def send_goal(self, locate):
        # Set up the next goal location
        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
```

3. summary

In this experiment, the final result of our group is that we got 48 points from 50 points in the first round. Because we didn't step on PS3 accurately, we deducted points. In the second round, he scored 58 points in three minutes, and the specific ranking was lower than average.

The points that our group can improve are: 1) the map drawing is not accurate enough, which leads to the car hitting the wall in the first attempt of the first round. However, in the perception of the car, there is still a certain distance from the wall, which makes the car based on AMCL

algorithm unable to judge where it is. 2) In the second round, due to the desire to achieve faster completion, the rotation angle of PS3 position is too large, so the car's local map thinks that there is a wall between PS3 point and other points, which leads to the low total score.

In this experiment, we applied the knowledge learned in the course to the experiment, and completed the basic requirements of the competition. In the process of preparing for the competition, we found a lot of resources on the Internet for learning and research, and made our own modifications based on some ROS built-in toolkits, so as to complete the final project. In this process, we feel the benefits of learning ROS framework. There are a lot of encapsulated algorithms and tools in the ROS framework. As developers, we only need to call these existing nodes, and with our own understanding and improvement, we can complete a project with a large amount of Engineering in a short time. This is the advantage of the ROS framework.

Finally, we would like to express our thanks to Mr. Zhang, the teaching assistants and the students who helped us in the course.

4. Contribution

Our group is made up of two senior students. To a large extent, because most of the time two people are discussing in the laboratory at the same time, there is no strict division of labor, which is basically completed by

the cooperation of two people. Pan Zhiyuan focuses on the code debugging of the car's multi-point automatic navigation, while Chen Haohao focuses on the drawing of the map and the function of recognizing the AR code.