Turtlebot Integrated Lab Reprot

Chenhao Ma 11911625 Zheng Gao 11912015 Shaohui Wei 11911625
Information Engineering

1 Introduction

In this lab, we will integrate on TurtleBot3 the functions that have been developed in all the previous labs throughout the semester. Our robot will work in the environment shown in Fig1. There are three separate tasks that our robot can attempt: autonomous navigation, lane following, and target search. For autonomous navigation, our robot will make use of the map that is built of the environment and visit the various locations of interest: P1, P2, P3, and P4. For lane following, our robot needs to move in the racetrack along the directions of the six arrows. For target search, our robot needs to look for an Aruco marker placed near one of the corners near P3, making a sound to describe the ID of the marker. The starting point of the competition is P1, and our robot will have a total of five minutes to collect points. Based on this, we designed the scheme to make the turtlebot complete the tasks in less time.

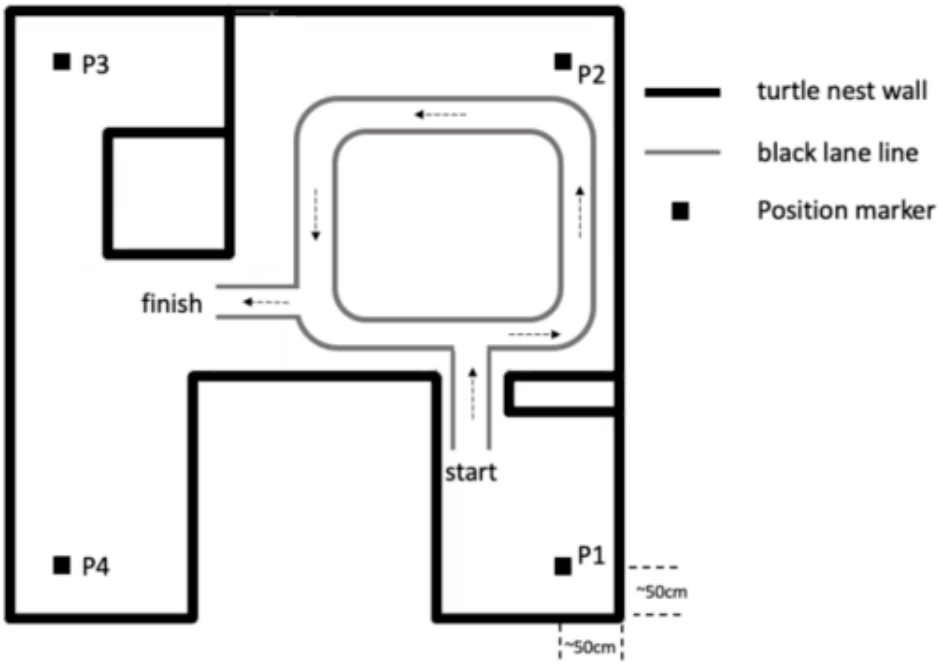

图 1: Figure1

2 Experiment Content

2.1 Navigation

2.1.1 Mapping

The first step should be build the map for the 2-D rnvironment.The LiDAR module of Turtlebot3 is used in this step to provide us with the depth information of the field, combining it with the odometry data from the robot will help estimate the location of robot and construct a 2-D occupancy grid map.

We used the official turtlebot3slam and turtlebot3teleop packages are used to initial mapping and controlling of Turtlebot3 through keyboard. Specifically, the following commands are used.

Launch mapping program

```
roslaunch turtlebot3_slam turtlebot3_slam.launch
```

Use keyboard to control Turtlebot3

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

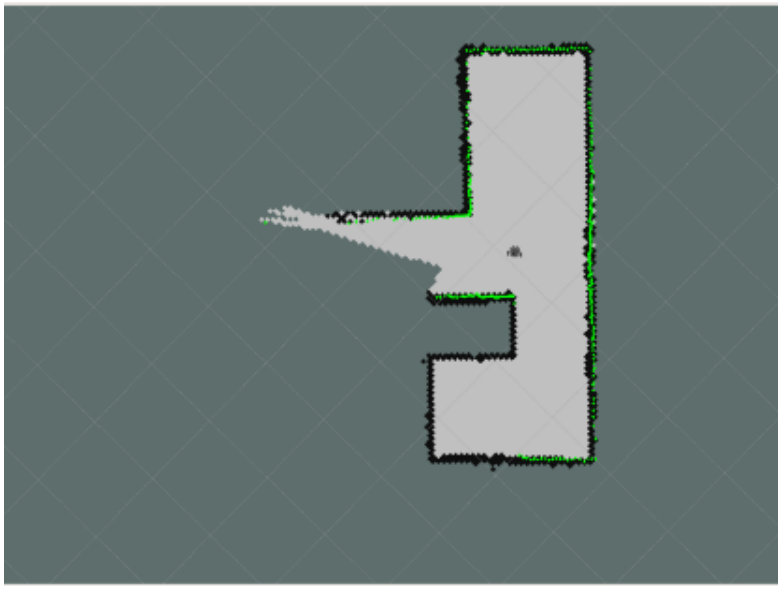

图 2: Figure2: GMapping with keyboard operation

Saving built map

```
rosrun map_server map_saver -f ~/EE346/map
```

The result is shown as

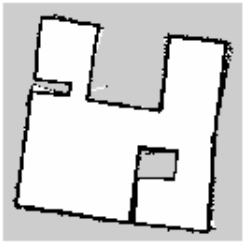

图 3: Figure3

2.1.2 Localization

Choose the real position and direction an the map, shown as Fig4

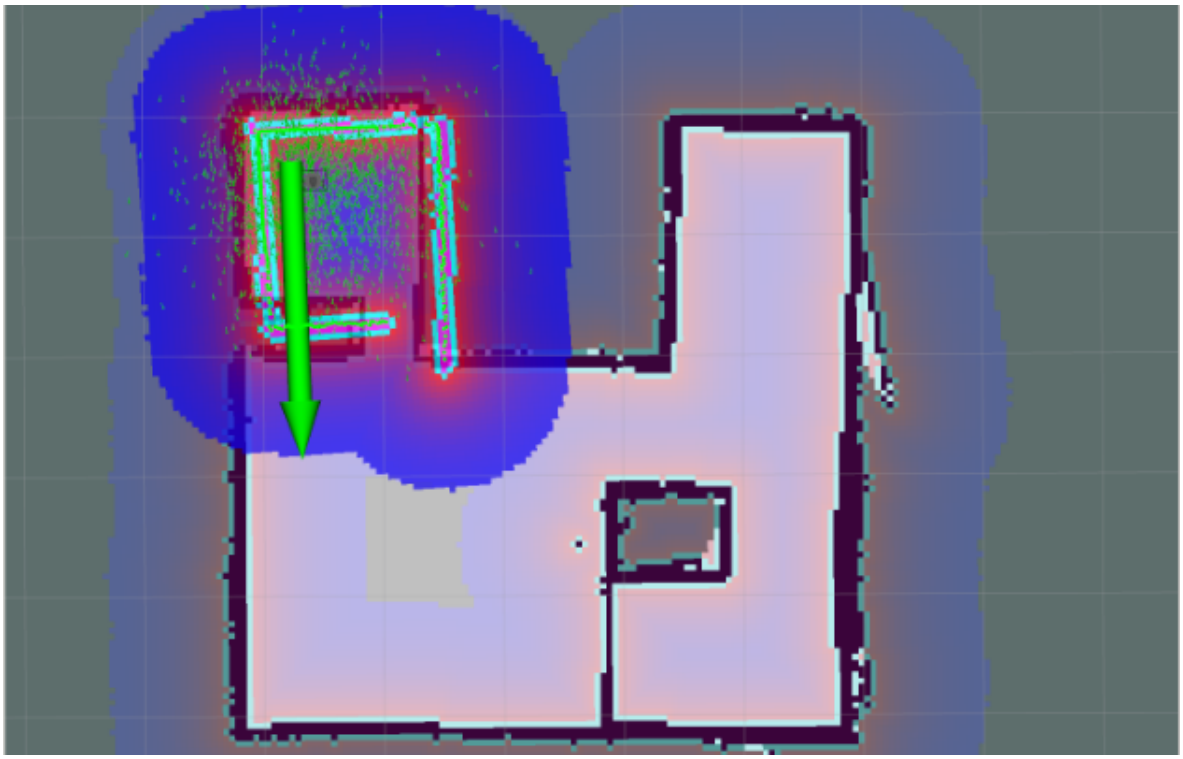

图 4: Figure4

2.1.3 4-points Navigation and Mark Detection

Our drone is asked to finish task PS1, PS2,, PS3, PS4 in order. In the first event, we found $cost_map$ is shown that turtlebot cannot go through narrow places. So we resuce the $inflation_radius$ to 0.1. As for each type of task, we used $move_base$ control robot to navigate to the point and stop at the point. After it reach each point it stop at each pit stop (while the accuracy is still not guaranteed). The ROS package ar_track_alvar is used to recognize ar code provided by the camera on Turtlebot3, and after each recognition,

it will publish information to topic ar_pose_marker, which is subscribed by another node "listener" designed by ourselves. This node provides the ability of playing sound using package $sound_play$.

2.1.4 complete commands and procedures of navigation

Starting the ROS service on PC

```
roscore
```

Connect Raspberrypi

```
ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Launch the Navigation

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.
    yaml
```

Start navigation (or task) script

```
rosrun linefollow navi.py
```

2.2 LineFollow

2.2.1 Image Processing

First we found other objets will make noise in camera image so we do image processing.

first we use Binary processing and set the threshold value to 60.

```
ret,image = cv2.threshold(image,60,255,cv2.THRESH_BINARY)
```

We use Kernel Filtering to reduce the noise in camera image.

```
kernel1 = numpy.ones((9,9),numpy.uint8)
 mask=cv2.morphologyEx(mask,cv2.MORPH_CLOSE,kernel1)
```

Finally, we found the turtlbot would be interfered when it turned. So we cover outer line to reduce the noise of outer line when the turtlebot turn.

2.2.2 Turtlebot Control

In order to make turtlebot follow line more stable, we change the linear velocity. And we have found that $v = 0.2$ is the best value that turtlebot can be stable and efficient.

To control the turtlebot turn smoothly and increase system robustness. We check the $turnerror$ if it is larg enough, the linear velocity will be changed to 0.075.

```
  angz=(err*90.0/160)/13
  print(angz)
  if abs(angz) > 0.15:
    self.twist.linear.x = 0.075
  else:
    self.twist.linear.x = 0.2
    self.twist.angular.z = angz
```

2.2.3 complete commands

Starting the ROS service on PC

```
roscore
```

Connect Raspberrypi

```
ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Launch the Raspberrypi Camera

```
roslaunch turtlebot3_autorace_traffic_light_camera turtlebot3_autorace_camera_pi.
    launch
```

Run the script

```
rosrun linefollow line_following_real.py
```

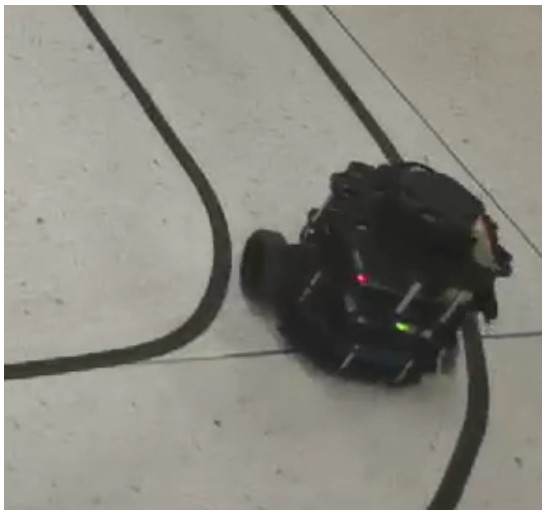

图 5: Figure5

3 Task Planning

Finally, after we complete all single tasks. We find the line following task takes much time and its robustness and stability are poor. So we decide to make turtlebot complete the navigetion and marker detection tasks. And the order of the points to navigate is $[2, 3, 4, 1]$.