Final Lab Report

EE346

Mobile Robot Navigation and Control

Name: 肖锐卓 SID: 11911119

NAme: 徐嘉睿 SID: 11911116

1 Introduction

In this lab, the students will integrate on TurtleBot3 the functions that have been developed in all the previous labs throughout the semester.This lab will be run in the form of a competition among all the student groups.The groups can selectively attempt various tasks in order to maximize the total number of points collected.The final mark for this lab will depend on the relative rank of a group within the class.

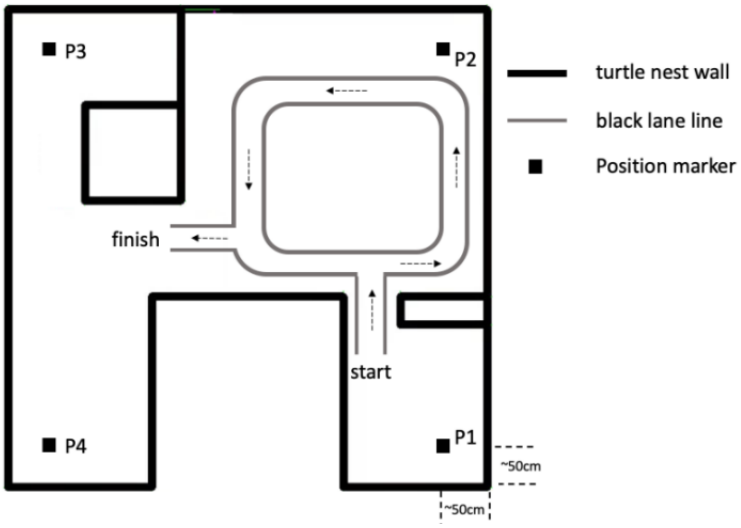

图 1: Robot environment with the racetrack to traverse and positions to visit

The rules of competition are as follows.

1. Your robot will begin its operation at P1
2. Each successful visit to a Pn is worth 10 points. If your robot fails to cover the location marker (numbered tape on the floor) by any part of its body, deduction of points will be made based on the nearest distance to the tape, at 1 point/2cm.
3. Success traversal of the racetrack is worth 20 points.There are a total of five corners, and your robot must not “cut the corners” .Each “cut corner” will result in a deduction of 1 point.
4. Your robot can choose to visit both P3 and P4 in any order, and just one of the two (P3 or P4) in each lap of the competition.
5. If you attempt the racetrack task, visit to P2 is optional.
6. If you do not attempt the racetrack task, you must visit two of the three locations, P2, P3 and P4, before returning to P1.
7. Your robot must return to P1 (worth 10 points), before it can attempt any of the other tasks in another lap of the competition.

8. If at any time, you need to lift your robot and re-place near where it is picked up, so that it can resume its operation, there is a penalty of 5 points.

2 Group Goals

1. The turtlebot starts at P1, passes through the corridor, and follows the arrows to complete the ”lane following” task.
2. When near P2, the turtlebot can come to P2 and then return to the lane to continue the ”lane following” task.
3. During the driving, the turtlebot can detect the AR code at any time, and make a sound broadcast after recognizing it.

3 Turtlebot description

ROS (Robot Operating System) is an open source operating system for robots that is more mainstream and used by more people in the world.

The computer operating system used in this project is Ubuntu18.04.

The turtlebot burger used in this project is a small, low-cost, fully programmable, open-source mobile platform developed on the basis of ROS, mainly for educational purposes, research, product prototyping and robotics enthusiast applications.

The operations that turtlebot can implement include forward, backward, left and right turn, SLAM based on Gmapping algorithm, autonomous navigation, etc.

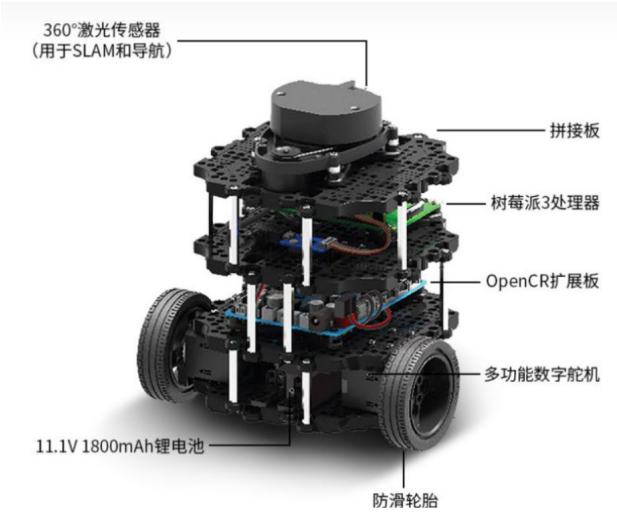

图 2: Turtlebot burger

The hardware models of the turtlebot burger are as follows.

硬件清单

BJROBOT

控制器	1x 控制器(ROS): OpenCR控制器
舵机	2x 智能马达: XL430
开发版	1x SBC 单板电脑: Raspberry Pi 3B+ 开发板
雷达	LDS（激光距离传感器）： 360激光距离传感器
内存	1x MicroSD 16G 记忆卡
电池	电池: Lithium polymer 11.1V 1,800mAh / 19.98Wh 5C

图 3: Hardwar of turtlebot burger

The basic performance parameters of the turtlebot are as follows.

Turtlebot3-Burger

标准版基础性能参数

BJROBOT

MAX平移速度:	0.22m/s
MAX旋转速度:	2.84 rad/s （162.7deg/s）
MAX有效载荷:	15kg
尺寸（长x宽x高）：	138×178x192(LxWxH, mm)
重量:	1 kg
爬高:	10mm 以下
预计运行时间:	2小时30分钟
预计充电时间:	2小时30分钟
MCU:	32-bit ARM Cortex M7 with FPU (216 MHz, 462 DMIPS)
IMU:	3 轴陀螺仪, 3 轴加速度计, 3 轴磁力计
外供电源:	3.3V/800mA; 5V/4A; 12V/1A
扩充脚位:	GPIO 18-pin; Arduino 32-pin
外围设备:	UARTx3,CANx1,SPIx1,I2Cx1,ADCx5, 5pin OLLO x4
舵机接口:	RS485x3,TTLx 3
音频:	可编程的蜂鸣声序列
可编程 LED:	LEDx4
状态LED:	电路板状态LEDx1；Arduino LEDx1；电源 LEDx1
按钮和开关:	按钮x2，复位按钮x1，Dip开关x2
PC 连接:	USB
固件升级:	透过 USB / 透过 JTAG
充电器:	输入100-240V, AC 50/60Hz, 1.5A max; 输出12V DC, 5A

图 4: Hardwar of turtlebot burger

4 Preparation

4.1 Camera Calibration

In the process of image measurement and machine vision applications, in order to determine 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, a geometric model of camera imaging must be established, and these geometric model parameters are camera parameters. Under most conditions, these parameters must be obtained through experiments and calculations. This process of solving parameters is called camera calibration.

The pinhole imaging model is simplified to a geometric expression as follows

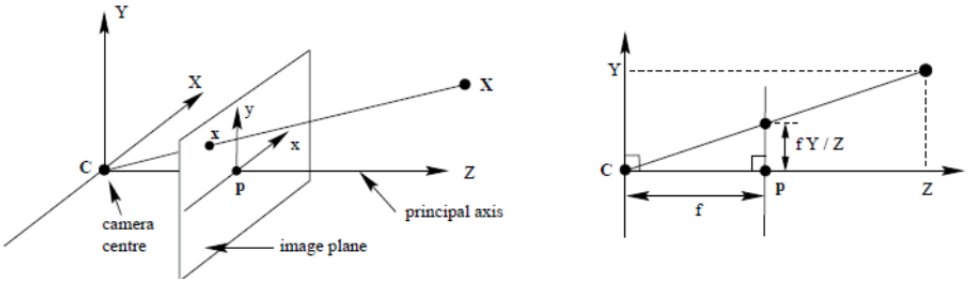

图 5: Pinhols imaging

Ideally, based on simple geometric knowledge of similar triangles, the relationship between the coordinates of the 3D target point in the camera coordinate system and the image pixel coordinates can be derived

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1}$$

Image sensors may not be square in their original size during manufacturing and may be skewed, so these factors need to be considered. Without considering the distortion, considering the principal point offset and the characteristics of the image sensor, the 3D target point imaging mathematical model can be fully expressed by the following formula.

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & \mathrm{s} & \mathrm{O}_x & 0 \\ 0 & \eta f & \mathrm{O}_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} K & 0_3 \end{bmatrix} P \tag{2}$$

K is called the internal parameter matrix. The camera internal parameter calibration is mainly to calibrate the internal parameters of the camera such as focal length, principal point, skew, etc.

In the lab, the process of camera calibration is as follows.

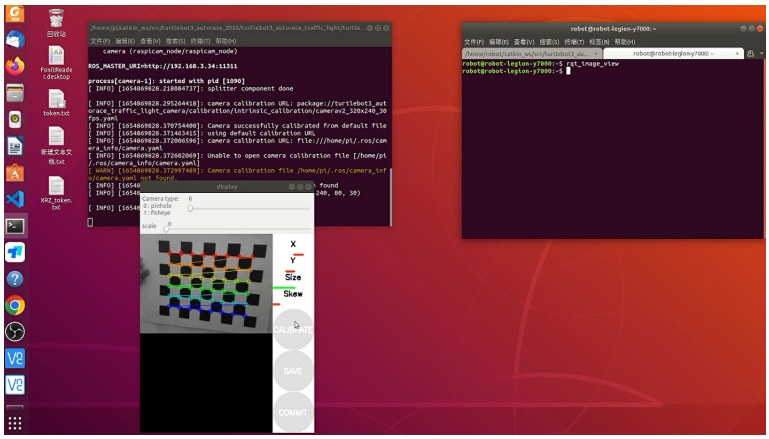

图 6: Intrinsic cameara calibration

The parameters of instrinsic calibration are stored in a ”yaml” file.

```yaml
image_width: 320
image_height: 240
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [ 249.50447,    0.     ,  150.23186,
            0.     ,  248.88341,  124.02887,
            0.     ,    0.     ,    1.     ]
camera_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.150148, -0.266848, 0.010839, -0.021014, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [ 1.,  0.,  0.,
          0.,  1.,  0.,
          0.,  0.,  1.]
projection_matrix:
  rows: 3
  cols: 4
  data: [ 251.56349,    0.     ,  142.60785,    0.     ,
            0.     ,  257.29599,  126.37844,    0.     ,
            0.     ,    0.     ,    1.     ,    0.     ]
```

图 7: Intrinsic calibration

4.2 SLAM

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 $tfmessage$. Laser scanning creates a map from it.Relevant framework required conversion laser, the reference and ranging information in tf message.

The map result are shown below.

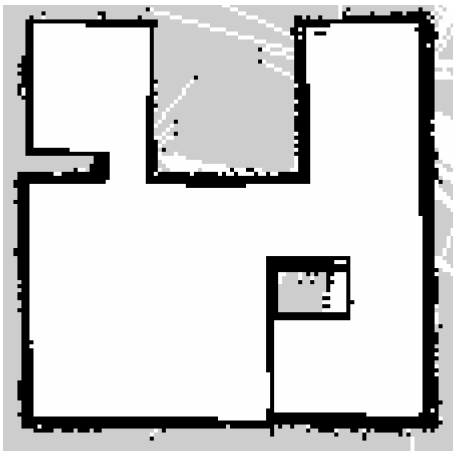

图 8: Map

The information about the map are stored in a ”yaml” file.

```yaml
image: rosData/GoodMap3.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

图 9: Map yaml

After completing the map construction, we can measure the coordinates of different points on the map and record them. In automatic navigation, we will publish the coordinates of these points to the turtlebot as the destination, and let the turtlebot navigate to these points.

For example, we measured the coordinates of some points.

```python
def send_goals_python():
    client = actionlib.SimpleActionClient('move_base',MoveBaseAction)
    client.wait_for_server()
    #定义四个发送目标点的对象
    goal00 = MoveBaseGoal()
    goal0 = MoveBaseGoal()
    goal1 = MoveBaseGoal()
    goal2 = MoveBaseGoal()
    goal3 = MoveBaseGoal()
    goal4 = MoveBaseGoal()
    goal5 = MoveBaseGoal()
    # 初始化四个目标点在 map 坐标系下的坐标,数据来源于《采集的目标点.docx》
    goal00.target_pose.pose.position.x =     0.77
    goal00.target_pose.pose.position.y =    -0.95
    goal00.target_pose.pose.orientation.z =  0.707106796641
    goal00.target_pose.pose.orientation.w =  -0.707106796641

    # p2
    goal0.target_pose.pose.position.x =    -0.0287735648453
    goal0.target_pose.pose.position.y =    -4.08753183365
    goal0.target_pose.pose.orientation.z =  0.707106796641
    goal0.target_pose.pose.orientation.w =  -0.707106796641

    # p3
    goal1.target_pose.pose.position.x = 3.95840157999
    goal1.target_pose.pose.position.y = -4.03311231613
    goal1.target_pose.pose.orientation.z = 0.707106796641
    goal1.target_pose.pose.orientation.w = 0.707106765732

    # p4
    goal2.target_pose.pose.position.x = 3.8586430645
    goal2.target_pose.pose.position.y = -0.0125219726562
    goal2.target_pose.pose.orientation.z =  0.707106796641
    goal2.target_pose.pose.orientation.w = - 0.707106796641

    goal3.target_pose.pose.position.x = 3.24684820175
    goal3.target_pose.pose.position.y =  -2.10940843582
    goal3.target_pose.pose.orientation.z = 1
    goal3.target_pose.pose.orientation.w = 0

    goal4.target_pose.pose.position.x = 0.753088831902
    goal4.target_pose.pose.position.y =  -1.91926503181
    goal4.target_pose.pose.orientation.z = 0.707106796641
    goal4.target_pose.pose.orientation.w = 0.707106796641

    # p1
    goal5.target_pose.pose.position.x = 0
    goal5.target_pose.pose.position.y =  0
    goal5.target_pose.pose.orientation.z = 0
    goal5.target_pose.pose.orientation.w = 1


    # goal_lists=[goal00, goal0, goal1, goal2, goal3, goal4, goal5]
```

图 10: coordinates of points

Next, we can publish these points as destinations for navigation。

```
goal_lists=[goal0, goal1, goal2, goal5]
total_goal = len(goal_lists)
print(total_goal)
goal_number = total_goal     # total is 6 goals
while(goal_number):
    if(total_goal - goal_number ==0):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
        rospy.loginfo(str_log)
    elif(total_goal - goal_number ==1):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
        rospy.loginfo(str_log)
    elif(total_goal - goal_number ==2):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
        rospy.loginfo(str_log)
    elif(total_goal - goal_number ==3):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
    elif(total_goal - goal_number ==4):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
    elif(total_goal - goal_number ==4):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
    elif(total_goal - goal_number ==5):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
    elif(total_goal - goal_number ==5):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
    elif(total_goal - goal_number ==6):
        goal_lists[total_goal-goal_number].target_pose.header.frame_id = "map"
        goal_lists[total_goal-goal_number].target_pose.header.stamp = rospy.Time.now()
        client.send_goal( goal_lists[total_goal-goal_number])
        str_log = "Send NO. %s Goal !!!" %str(total_goal-goal_number)
        rospy.loginfo(str_log)

    wait = client.wait_for_result(rospy.Duration.from_sec(60.0))  # 发送完毕目标点之后，根据act
```

图 11: Publish the destination

4.3 Aruco tag detection

Aruco tag is a fiducial marking system that can be understood as a reference for other objects. It looks similar to a QR code, but its coding system is still very different from a QR code. It is mostly used in camera calibration, robot positioning, augmented reality (AR) and other applications. Its main function is to reflect the pose relationship between the camera and the label, and then reflect the reference relationship between the object and the camera in the scene.

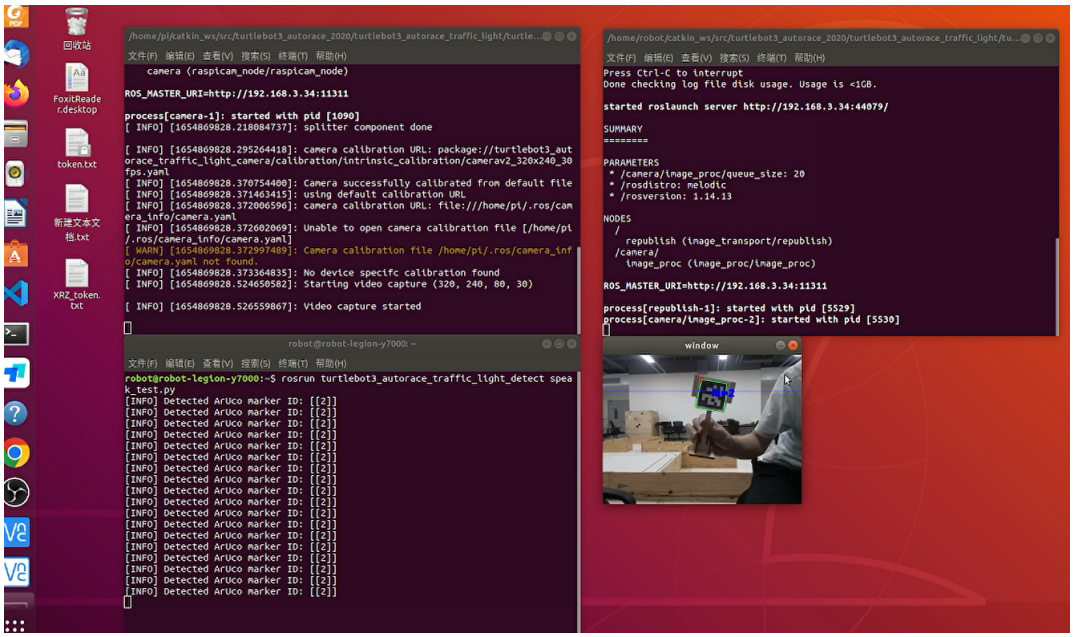

图 12: Subscribe the image

5 Progress and Result

In the task ”lane following”, we use a very simple algorithm to ensure that the car can travel in the two black lines. We put the camera of turtlebot on the second layer, which allows the camera to observe the lane more closely.

First, the camera publishes the image to the computer through the router. We subscribe to images on the computer, and the images at this time are BGR types. In order for OPENCV to process the image, we need to convert the image to HSV type.

```
# 获取图像信息
if self.sub_image_type == "compressed":
    #converting compressed image to opencv image
    np_arr = np.fromstring(image_msg.data, np.uint8)
    image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
elif self.sub_image_type == "raw":
    image = self.cvBridge.imgmsg_to_cv2(image_msg, "bgr8")

# 将图像转换为HSV格式
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
h, w, d = image.shape
# print(h,w)
```

图 13: Subscribe the image

To detect lanes, we need to binarize the image. We set the black HSV pixel parameters, and then binarize the image to get an image that only contains black lanes.

The core of this algorithm is to mark the rectangular window at the appropriate position on the image. When a black lane appears inside the rectangular window, the turtlebot adjusts its direction and speed to avoid rushing out of the lane. For example, when a lane appears in the rectangular window on

the right, it means that the car is on the right. So the computer sends command to the turtlebot to let it obtain an angular velocity for a left turn.

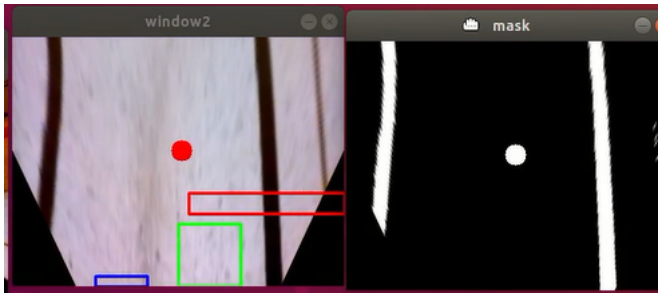

图 14: detection window in image

Under this algorithm, the time and position of the turtlebot for one lap will be very stable. In the process of driving, the turtlebot will record the driving time. After a certain time, the turtlebot will perform a specific action, so that the turtlebot can reach the point P2 and return to the lane.

```
# Go forward
twist.linear.x = 0.21
twist.linear.y = 0
twist.linear.z = 0
twist.angular.x = 0
twist.angular.y = 0
twist.angular.z = -0.6
self.cmd_vel_pub.publish(twist)
self.cmd_vel_pub.publish(twist)
time.sleep(3)

# Rotate leftforward
twist.linear.x = 0
twist.linear.y = 0
twist.linear.z = 0
twist.angular.x = 0
twist.angular.y = 0
twist.angular.z = 1.0
self.cmd_vel_pub.publish(twist)
time.sleep(2.4)

# Resume following
twist.linear.x = 0.21
twist.angular.z = 0
self.cmd_vel_pub.publish(twist)
time.sleep(1.5)

global t00
```

图 15: Go To P2

In the process of driving, in order not to affect the normal driving of the turtlebot, we started another node to identify the aruco tag.

```
        # 判断是否识别到aruco
        corners, markerID, rejected = aruco.detectMarkers(image, aruco_dict, parameters=param)
        if len(corners)>0:
            if time.time()-t0<15:
                pass
            else:
                rvec, tvec = aruco.estimatePoseSingleMarkers(corners, 0.05, matrix, dist)
                (rvec-tvec).any()
                for i in range(rvec.shape[0]):
                    aruco.drawDetectedMarkers(image, corners,markerID)
                    print("[INFO] Detected ArUco marker ID: {}".format(markerID))
                    engine.say('ID is'+str(markerID))
                    engine.runAndWait()
                    last_ID=markerID[0][0]
                    t0=time.time()
                    break
        # 窗口绘图
        cv2.imshow("window", image)
        cv2.waitKey(10)
```

图 16: Code of Aruco tag detection

After identifying the aruco tag, the computer will make a sound, indicating that the car has successfully identified the aruco tag.

5.1 Conclusion

In this project, based on turtlebot burger, we have realized camera calibration, aruco tag detection, automatic navigation, lane following and other functions. Two people in our team were promoting the project at the same time, and each contributed almost 50 percent.

There are still some shortcomings in our programming. For example, when the turtlebot rushes out of the lane, it cannot return to the lane by itself and needs human help to correct its attitude. In addition, whether the turtlebot can reach the points P1 and P2 accurately depends on whether the turtlebot can accurately reach the desired position. If we can improve the robustness of the program, the turtlebot can perform better tasks without human assistance