EE346 - Mobile Robot Navigation and Control

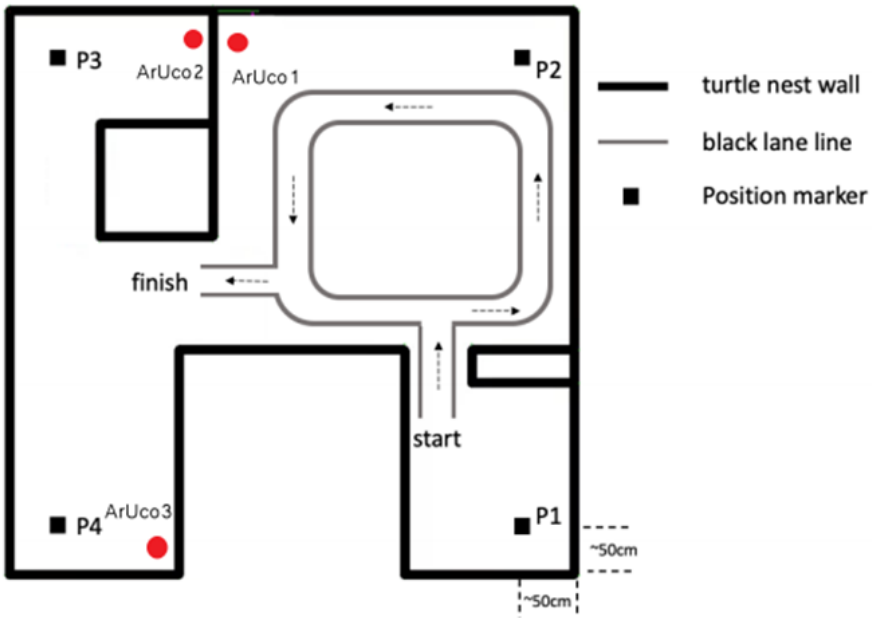
Final Lab Report

Haoteng Ye, Hongjing Tang

Background

Introduction

- In this competition, we aim to do as many tasks as possible in five minutes to score points. According to our strategy, we mainly completed the lane following, visited P2, and tested ArUco1. First, calibrate camera parameters to prepare for line patrol and ArUco tag detection. Then, the image is processed by perspective transformation, and the algorithm of line patrol and ArUco recognition are designed. Finally, design turtlebot to accurately visit P1 and P2.



Competition Rules

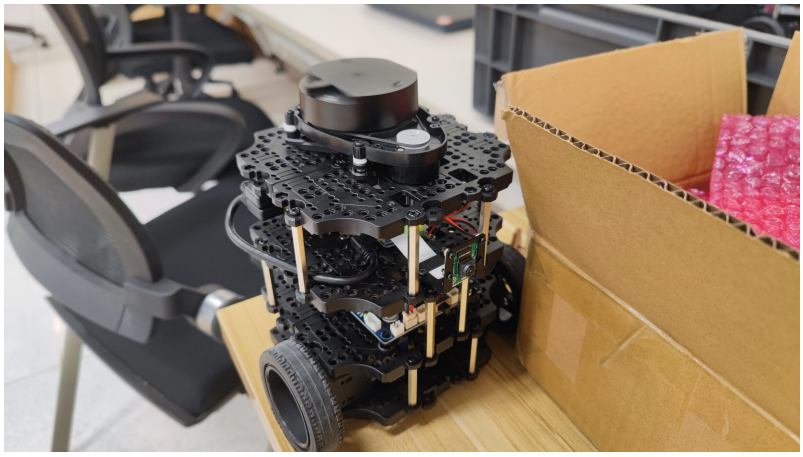
- In this race, we have three tasks: lane following, visiting 4 P points, and ArUco tag detection. Their corresponding points are 20 points, 40 points (10 points per point), and 15 points (5 points per point). Our starting point is set at P1, and we can only complete the task mentioned above once within each loop. If you want to complete a task more than once, you must go back to P1 to reclaim the task. In 5 minutes, we can do anything to get points. Of course, there are some deductions, such as the fact that

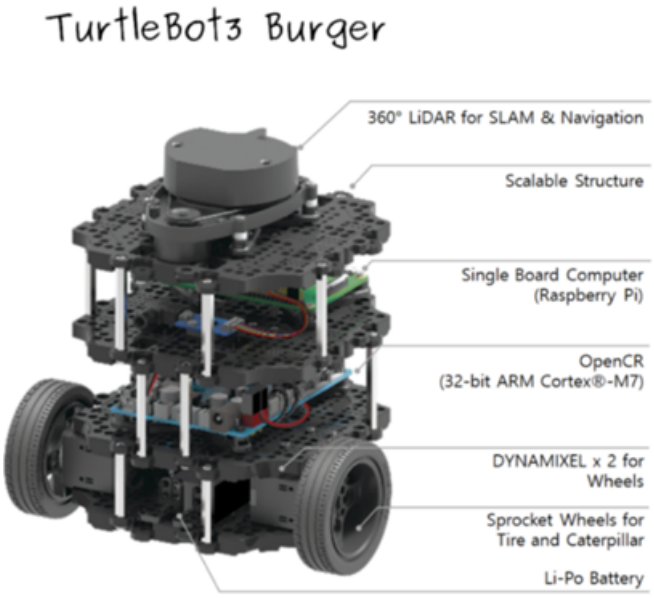
the Turtlebot does not accurately place a tag on the point P (1 point per 2cm) and that the Turtlebot cuts inside corners (1 point per corner) when turning.

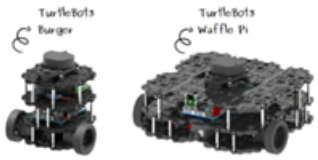
Experiment Platform

Turtlebot3

- In this project, we choose turtlebot3-burger as our robot. The details are here:
    - https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications





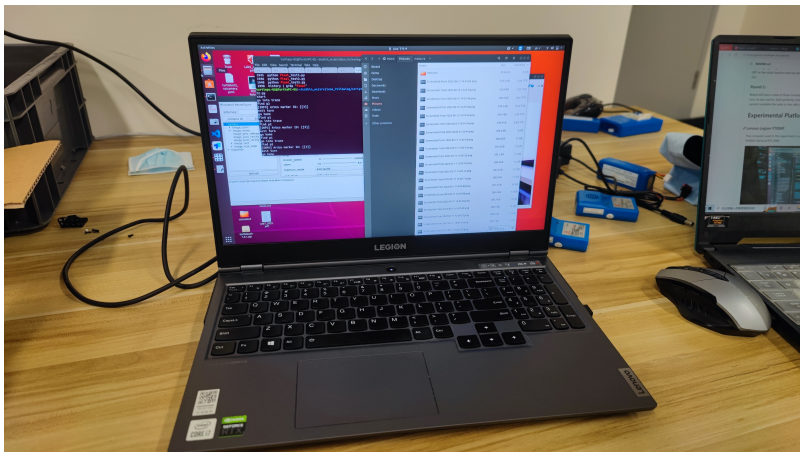


2. 1. 1. Hardware Specifications

Items	Burger	Waffle Pi
Maximum translational velocity	0.22 m/s	0.26 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)	1.82 rad/s (104.27 deg/s)
Maximum payload	15kg	30kg
Size (L x W x H)	138mm x 178mm x 192mm	281mm x 306mm x 141mm
Weight (+ SBC + Battery + Sensors)	1kg	1.8kg
Threshold of climbing	10 mm or lower	10 mm or lower
Expected operating time	2h 30m	2h
Expected charging time	2h 30m	2h 30m
SBC (Single Board Computer)	Raspberry Pi	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-	RC-100B + BT-410 Set (Bluetooth 4. BLE)
Actuator	XL430-W250	XM430-W210
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02	360 Laser Distance Sensor LDS-01 or LDS-02
Camera	-	Raspberry Pi Camera Module v2.1
IMU	Gyroscope 3 Axis Accelerometer 3 Axis	Gyroscope 3 Axis Accelerometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A	3.3V / 800mA 5V / 4A 12V / 1A

Lenovo Legion Y7000P

- The computer used in this experiment is Lenovo Legion Y7000P with 16GB RAM. CPU is Intel Core i7-10875H. GPU is NVIDIA GeForce RTX 2060.



ROS

- The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. It contains a wealth of tools, library code, and conventions designed to simplify the process of creating complex, robust robot behavior across robotics platforms. In addition, ROS provides standard operating system services, such as hardware abstraction, low-level device control, common function implementation, inter-process messaging, and packet management.

- In this project, we will use ros-melodic on Ubuntu to manage sensor messages and robot motion. We can check the rqt_graph to see the overall structure of our project, which includes all the nodes, topics, and messages. We also used the rqt tool to run all existing GUI tools as dockable windows.

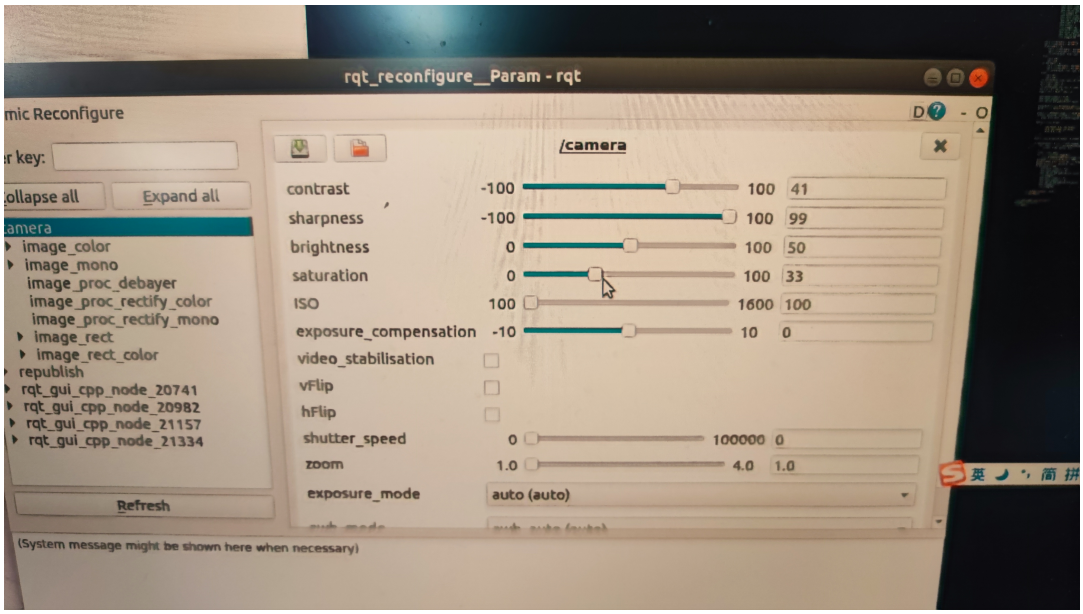
OpenCV

- OpenCV (Open Source Computer Vision Library) is a cross-platform Open Source software Library for Computer Vision processing that can be used for free in business and research. OpenCV can be used to develop real-time image processing, computer vision, and pattern recognition programs. For example, we use OpenCV for binarization, perspective changes, and ArUco detection of images.

Experiment Method

Lane Following

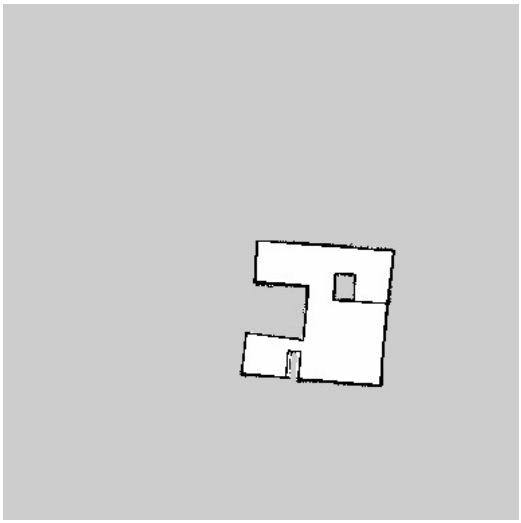
- The lane following algorithm will be similar to the lane following strategy we used before. We first extracted the black lines of the field. We can adjust the camera parameters using `rosrun rqt_reconfigure rqt_reconfigure` command. The interface is shown below.



- We tried to adjust the camera so that the track lanes can be detected clearly. After that, we used HSV color channel to extract black colors. Then, we made projection to the image so that we can see a bird-view image of the track.
- We calculated the moment of projection image to get the coordinates of the center of the track, and use the horizontal offset comparing with the image center to change the turning angular speed of the robot. With the algorithm, the car will dynamically keep itself at the center of the track lane.

Point Navigation

- We used Lidar to try to scan the whole area of the robot field. The scanned map is shown here.



- With the map we can set goals in Rviz to control the car to the specific point, and use the output topic to read the exact position of the points.
- However, sometimes with the delay of the Internet and the misjudge of the Lidar, the position of the wall and specific points will be not so accurate. Thus, navigation with Lidar will take some risks to realize.

AR Code Detection

- We have tried to detect AR code before. The main theory is the same as before. The robot compare the image with the code library to find a right one. After that, we used a python package called `pyttsx3` to read the result of the code. The program will speak out the corresponding number of the code.

Strategy and Optimization

Contest Strategy

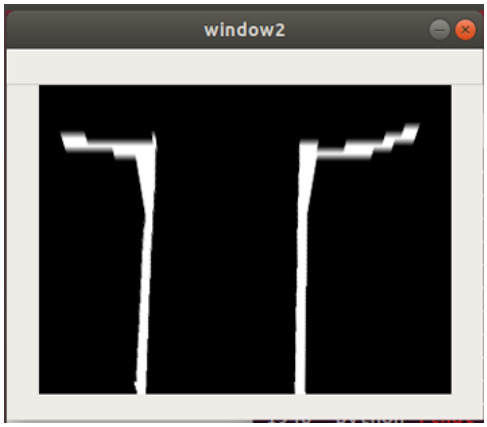
- In practice, we tried many strategies of finishing the 5 minutes, including pure lane following, pure point navigation and mixing them.
- In the end, we used the strategy that we thought will be best. The strategy figure is shown here.

- The robot will start from P1 to travel around the lane, in the meantime go to the P2 point and scan the ARUCO in that area. Then, the car will go back immediately to P1 and start a new lap.
- This is a strategy which will not complete all the tasks in the area in a single lap. However, with a fast speed and short lap time, we can repeat the laps for around 4

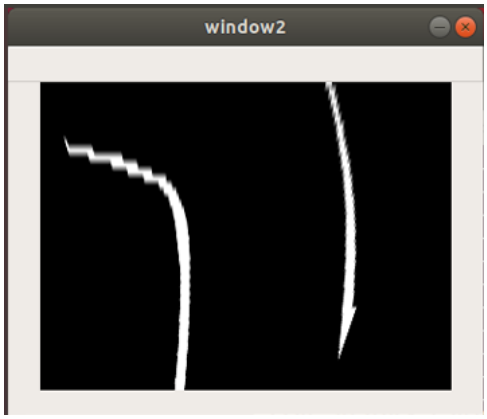
times to obtain as many points as possible.

Important Points in Our Algorithm

- The program we wrote is a mixture of lane following and fixed program. We wrote some fixed process to help the car to go into or out some important stages. Thus, one of the important problem is to decide where the car will stop lane following and use a fixed program.
- We define a status identifier variable in the initialization of the class called `step`. Every time the car entered a stage, the `step` will add 1, so that the car will go into a new detection. And also, for some points that the car have a high possibility to misjudge, we used a timer to catch the passing time after former stage, if the current time is not longer than the expectation, the car will not detect the condition we set.
- Here are some of the fixed programs we wrote, and the changing position for them.
    - **Start the car and go into the lane.** When the race start, the car will follow a fixed program to the beginning position of the lane, so that the car can detect the lane and adjust its pose.
    - **Go out the room, turn right and go into the straight road to P2.** Before entering the turn, the car will see the empty part of the cross road, in this time, the upper part of the image projection will be blank. Thus, the robot will detect the existence of the three road junction by seeing if the upper part has pixels. If not, a fixed program will lead the car go straight to the intersection, turn right, and then pass the coming let turn into the straight road.



    - **Find P2.** In the straight line, if the robot see a left turn, it will follow a fixed program to P2, cover it, and then turn back to the road. We detected the left turn by splitting the projection image into four parts. If the top left part has no pixels while the top right one has pixels, the left turn is detected. After that, the car will spin itself to see the lane and follow it again.



- **Pass the junction and go back to room.** After returning to the lane from P2, the car will follow the lane for some time, until it detected the first junction of three roads. We detected it by seeing if the top left part has pixels and the top right has no pixels. When seeing it, the robot will follow a fixed program to pass the junction, go into left turn, and then go straight to the entrance of the P1 room.

- **Find P1 and start a new lap.** After stopping at the entrance of the room, the car will follow the lane to the end of the lanes, and then go into a program. It will lead the car to find P1, stop for half a second, and then go back to the beginning of the lane to start a new lap.

- One thing that has to be mentioned is that although we set a lot of conditions for the robot, and some of them might be misjudged by the stain on the ground or other noise pixels, we have set some timers to help the robot decide if the situation need to be detect or not. If the time is lower than the value we expected, the robot will directly skip the condition and continue the current status.

Problems We Have Met

- Before deciding the strategy, we have met a lot of problems.

Navigation

- At first, our preferred algorithm is to mix lane following and point navigation, so that the robot can switch between each of the two tasks to accomplish the contest.
- However, we met problems in the choices of the switching point and the method of algorithm switching. Sometimes, after switching the algorithm, the car will be in a bad

position that it might take a long time to find itself on the road to the target. And also, if the internet got stuck for a short time, the robot will be confused by both of the algorithms.

Stability

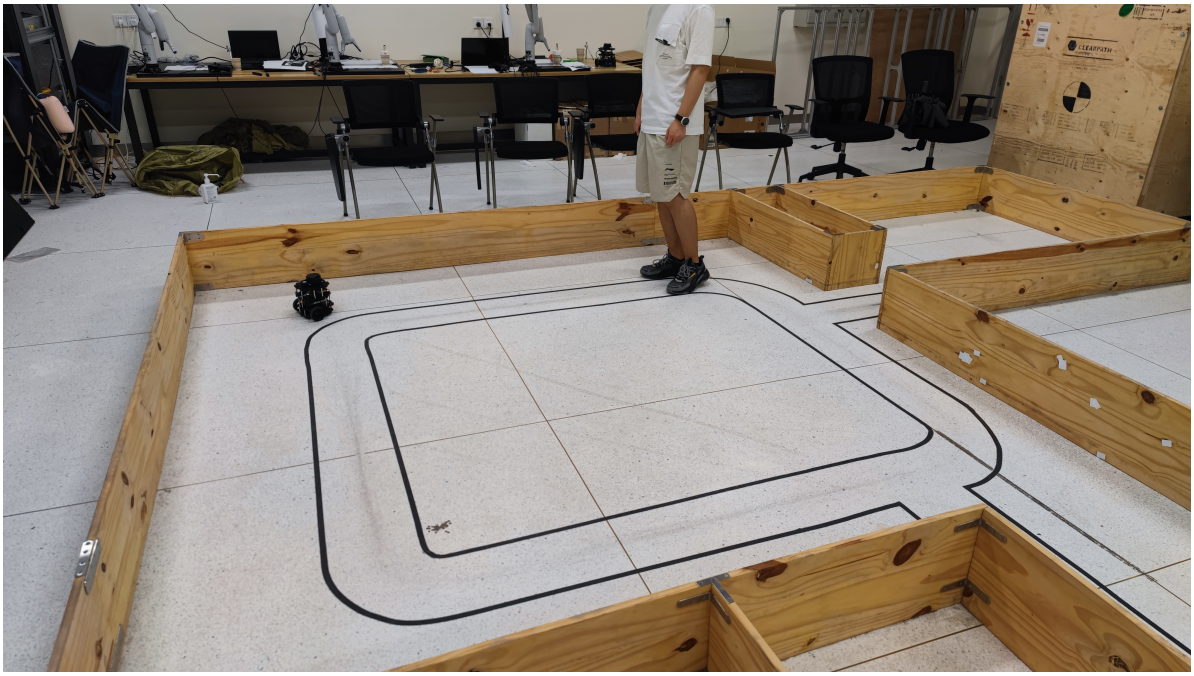
- We met some problem in the robot stability. The problem is not only caused by the sometimes-poor network connection, but also the algorithm parameters we used. The problem is rather serious after the car left Point P2 and before going back to the P1. In this sector, the robot will meet a turn and two junctions of three roads. With the sudden disconnection and reconnection of the network, and the effect of the fork in the road, the car direction will vary when the car passes the fork.

Optimization Solution

- With these confusions, we then decide to use a fixed program to lead the robot to specific points. When the car reached some set points, the car will stop lane following and use a fixed speed-time sequence to go to the target.
- This is a bad way in class learning, if have to say. However, with practice and comparison, it is a faster and stabler way to control the car.
- However, although we used a fixed program to lead the car pass the two forks and go back to P1, with this small offset, the car situation will be unstable. In the end, we still cannot fix the problem and decided to give it a hand when it stopped at the entrance of the P1 room.

Test Result

- In the real contest, we made a single trial of the robot. As a result, the robot had passed 4 laps, in each of which, it will go lane following, find P2 and scan an ARUCO in the meantime before going back to P1.



- After the first attempt, we thought it is already beyond our expectation, and both of us were sure that there will be nothing we can do to increase the score. Thus, we gave up our second chance.
- Some of the running video we recorded can be seen in the website below:
    - https://www.bilibili.com/video/BV1kF411F7De?share_source=copy_web

---

Conclusion

- We learned and used many robot control techniques in this course, such as GMapping, AMCL, Line Follow, and ArUco Detection. However, we did not use radar for navigation because our radar was not always accurate, and there was a risk of packet loss and delay. On the other hand, the influence of the network on lane following is small, so we studied how to make lane following have a good and stable effect. Although we used a few tricks to find spots and take some corners, it worked well. The only pity was that turtlebot often ran off course when returning to its starting point, and we had to correct its position manually. Nevertheless, in the end, we have achieved satisfactory results.
- Contribution:
    - Haoteng Ye: Program logic design; Main program writing. (50%)
    - Hongjing Tang: Program logic design; Parameter adjusting. (50%)