# ROS project

LU ZHICHEN
Master SAR
*Student number 21117174*

LUO JIAWEI
Master SAR
*Student number 21109757*

## I. INTRODUCTION

This project focuses on the control of a simulated and real Turtlebot 3 burger mobile robot, navigating through a realistic challenging environment. The robot is tasked with moving from a starting position to a goal position, solving different tasks along the path. To successfully navigate the environment, the robot must exploit images obtained from a simulated/real camera to detect and follow lines, a laser scan from a simulated/real LDS to detect and avoid obstacles, and ultimately combine both sensors to operate in a challenging environment where both are required. The project consists of three successive challenges that test the navigation capabilities of the Turtlebot:

(a) line following;
(b) corridor navigation;
(c) cluttered environment navigation.

## II. PRESENTATION OF THE ROS ARCHITECTURE

### A. A short overview

This section will provide the rqt diagram Figure 1 and rqt diagram Figure 2(All topics and nodes), as well as the nodes and topics used throughout the project.

Fig. 1. ROS_rqt_graph

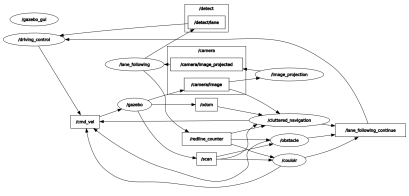

Fig. 2. ROS_rqt_graph_all

Nodes:

- Nodes list:
  - /cluttered_navigation: Cluttered navigation task execution.
  - /couloir: Couloir task execution.
  - /driving_control: PD control of the turtlebot3, and speed release after control.
  - /image_projection: Projection processing of the acquired camera image.
  - /lane_following: The path patrol further processes the received projection image to obtain the path trajectory, including BGR to HSV processing, mask processing and extraction of coloured lane lines, fitting the path with a polynomial fit method and plotting the processed path. There is also a count of the red task lines.
  - /obstacle: Construction task execution
  - /gazebo
  - /gazebo_gui
  - /rosout
- Topics list:
  - /camera/image: Image message returned by the camera.
  - /camera/image_projected: Projected image after processing.
  - /detect/image_output: Projected image of the turtlebot3's path of motion.
  - /cmd_vel: Velocity of turtlebot3.
  - /couloir_open: Message for controlling couloir task.
  - /detect/lane: Message for detecting lanes.
  - /detect/white_line_reliabilit: Detection of white line.
  - /detect/yellow_line_reliability: Detection of yellow line.
  - /lane_following_continue: Message for controlling lane following task.
  - /redline_counter: Storage of red task line detection counts
  - /redline_image: Image message of redlines
  - /scan: Distance message returned by laserscan
  - /rosout

### B. Algorithmic structure of the architecture

In this subsection, you must detail how all the nodes listed in your architecture work together for a given scenario. For instance:

- Node ***/lane_following*** performs projection processing on the image sent from node ***/gazebo***
- Node ***/lane_following*** will send the processed path information to node ***/driving_control***, it will send the message of counter of redline to nodes ***/couloir***, ***/obstacle*** and ***/cluttered_navigation***. Node ***/driving_control*** then will publish the velocity message based on the received path information.

- If counter of redlines == 0 or 1: It just runs the node ***/lane_following***.
- Elif counter of redlines == 2: ***/obstacle*** will receive this message, send the message to stop running the node ***/lane_following*** temporarily and then start to run itself to finish the obstacle task:
  - If laserscan detectes the first obstacle, the couter of obstacle will process the operation "turn left" and send the message to continue running the node ***/lane_following***.
  - Elif laserscan detectes the second obstacle, the couter of obstacle will process the operation "turn right" and send the message to continue running the node ***/lane_following***.
- Elif counter of redlines == 3: ***/couloir*** will receive this message, send the message to stop running the node ***/lane_following*** temporarily and then start to run itself to finish the obstacle task:
  - If there is one distance is less than 0.185 for a sector at an angle of 140 degrees ahead, this node will adjust the direction;
  - Elif all distance is more than 0.185 for a sector at an angle of 140 degrees ahead, this node will go straight.
- Elif counter of redlines == 4: ***/couloir*** stops and it will send the massage to continue running the node ***/lane_following***.
- Elif counter of redlines == 5: ***/cluttered_navigation*** will receive this message and send the message to stop running the node ***/lane_following***, then runs itself to finish the nagivation task:
  - If the robot is close to the exit, it switches to the 'exiting' state, this node publishs a velocity message to go straight;
  - Elif the robot is not close to the exit and the robot detects a nearby obstacle or boundary, it stays at rest and rotates to the next possible direction;
  - ELif there are no nearby obstacles or boundaries, the robot decides between 'searching' and 'navigating' states:
    - If the robot is in the 'searching' state (target not found), the node publish a velocity message to rotate the bot;
    - Elif the robot is in the 'navigating' state (target found), the node calculates the move_cmd.angular.z by the PID controller, the node publish a velocity message based on the PID control.

## III. Performance characterization

***Refer to the annex for performance analysis 2.***

In this performance analysis, we evaluate the behaviour of the robots based on test data. Consistency in performance: The test data showed that the robot's performance showed variability even with the same parameter settings. This suggests that the robot's behaviour may be influenced by initial conditions and other unpredictable factors. After our analysis, we believe that there may be too much noise in the distance and image information received, which affects the conditional judgement of the running code, which in turn affects the different variations in turtlebot3 speed, ultimately leading to different performance of the results for the same initial conditions and the same parameters.

**Obstacle avoidance**: As the robot detects obstacles at different directional distances, it is possible to get too close, which can cause the camera to catch the exit behind the obstacle (through the obstacle) and subsequently cause the robot to execute a command to run in a straight line towards the exit and keep travelling towards the obstacle.

**Boundary compliance and target tracking**: Adjusting the parameters: area_threshold_boundary_min and area_threshold_boundary_max has little effect on the robot's ability to stay within the boundary or to improve target tracking accuracy. That is, the turtlebot is able to operate well within the specified boundaries.

**Influence of the control parameters**: Increasing the value of Kp causes the robot to move unsteadily and leads to excessive turning speeds, which can seriously affect the radar detection distance and the processing information of the images. It is therefore necessary to control the Kp value to a value that makes the turning speed as large and stable as possible.

**Green pixel threshold** : The choice of the green pixel threshold parameter greatly affects the behaviour of the robot as it approaches the exit. A low threshold may cause the robot to turn in other directions before reaching the end due to a late transition to the exit state. Careful selection of this parameter is essential to ensure optimal performance.

## IV. Conclusion and perspectives

In this project, we learned to become more proficient in applying ROS commands to write programs and understand the logical relationships between nodes, as well as gaining a deeper understanding of how to handle different tasks in an unmanned autonomous driving scenario. We also noticed how the possible external environment can have a huge impact on the performance of the program when calling the camera images and distance information from the radar, such as the lighting conditions of the turtlebot in a real environment, the slope of the ground, the friction, the size of the camera pixels and the detection effect of the radar. Also learned how to remove noise and compensate for image information, which was much better understood when applying openCV.

However, we have also encountered some difficult problems:

- **Removal of noise from images**: In a virtual environment, where the external environment is fixed and the scene is fixed, it is possible to obtain a clearer image by adjusting the mask, the hsv values for different colours and the image obtained by compensation. However, in a real environment, the intensity of light, the position of the

camera and the height of obstacles can affect the messages received by the hardware, so it is necessary to set a wider range of hsv values during debugging to reduce the effect of light, as well as to make the information obtained by the hardware more accurate by adjusting the position of the camera and the height of obstacles. But even then, we would like to design algorithms to accomplish the processing of complex external environmental factors, such as for illumination, by considering the following for each frame obtained:

  - Colour distribution calculation: The process of colour distribution calculation can be implemented using histogram statistics, dividing the colour space into small intervals and counting the number of pixel points in each interval to obtain the colour distribution.
  - Average colour calculation: The average colour calculation can be done by calculating the average of the hue, saturation and luminance channels separately to obtain the average colour of the whole image.
  - Colour similarity judgement: Colour similarity can be calculated using methods such as Euclidean distance, comparing other colours with the average colour, and if the distance between them is less than a certain threshold, the colour is judged to be similar.

  However, this algorithm requires that the ambient light shift is not too drastic, otherwise it will lead to distortion and a loss of accuracy. And if there are different objects in the environment, each of which may be a different colour, this will also require different colours to be judged depending on the object. In addition to this, the setting of thresholds needs to be considered.
- **The value of the distance information from the radar fluctuates too much**: Even in a virtual environment, the distance values returned by the radar fluctuate considerably even when the turtlebot is not moving. In order to minimise the impact of fluctuating values, we consider refining the algorithm: when the trolley is running at a relatively smooth speed (i.e. when the difference between the angular and linear acceleration changes over a period of time is small), the distance information of the trolley is collected over this period of time The average of these distances is used as a reference value, and the error value is set. In other words, when the difference between the detected value and the reference value in the sector in front of the trolley is less than the error value, the turtlebot will not make any directional adjustment, but will continue to drive at the original speed, which greatly reduces the number of directional operations and the impact of frequent changes in speed.

In addition to analyzing the issues that need to be addressed, there are also areas where we can improve:

- We did not apply service and msg in the initial design because we considered using node publication to switch between different tasks, instead of using services for node switching. This was partly due to our lack of familiarity with the use of services and incomplete understanding of the convenience that services provide for node switching. Furthermore, the most important factor is the ability to achieve bidirectional transmission, instead of having to write a publisher and a subscriber separately in two nodes.
- In the initial design of the software package, we planned to display all the information of the running nodes and the corresponding key information published in the log to complete the detection of task implementation. We could also further process the returned data for performance testing and analysis. Unfortunately, we lacked experience in using log files. After trying for a while, we chose to give up using logs and relied on qualitative analysis instead of quantitative analysis for performance analysis. Additionally, due to virtual machine issues, the performance testing package for the line-following code and the couloir code was damaged. We didn't have time to rewrite the code, so we could only submit the performance testing results for part three (which is also partially incomplete).

In conclusion, throughout the challenge, to optimize the system, we can envision a more complex world, such as one with moving obstacles. In order to enable the vehicle to navigate better, we need to add more conditional judgments in the navigating function, such as detecting whether the distance measured by the LDS is decreasing and making the vehicle respond accordingly.

| Parameter | | | | | Performance | | |
|---|---|---|---|---|---|---|---|
| area_threshold_ boundary_min | area_threshold_ boundary_max | Kp | min_ distance | green_pixel_ threshold | Whether it can reach the destination | The path of the robot when it reaches | The obstacle avoidance effect |
| 20 | 1300 | 0.01 | 0.18 | 4500 | o | 3.67m | very good |
| 20 | 1300 | 0.01 | 0.18 | 4500 | o | 2.78m | very good |
| 20 | 1300 | 0.01 | 0.18 | 4500 | x | - | - |
| 100 | 1300 | 0.01 | 0.18 | 4500 | x | - | - |
| 200 | 1300 | 0.01 | 0.18 | 4500 | o | 2.79m | very good |
| 200 | 1300 | 0.01 | 0.18 | 4500 | o | 3.0m | hit 1 time |
| 200 | 1300 | 0.01 | 0.18 | 4500 | o | 3.0m | hit 1 time |
| 250 | 1300 | 0.01 | 0.18 | 4500 | x | - | - |
| 220 | 1300 | 0.01 | 0.18 | 4500 | x | - | - |
| 220 | 1300 | 0.01 | 0.2 | 4500 | x | - | - |
| 220 | 1300 | 0.01 | 0.2 | 4500 | o | 3.86m | very good |
| 220 | 1300 | 0.01 | 0.2 | 4500 | x | - | - |
| 220 | 1300 | 0.001 | 0.2 | 4500 | x | - | - |
| 220 | 1300 | 0.05 | 0.2 | 4500 | o | 2.78m | very good |
| 220 | 1300 | 0.05 | 0.2 | 4500 | x | - | - |
| 220 | 1500 | 0.01 | 0.2 | 4500 | o | 2.63m | hit 1 time |
| 220 | 1500 | 0.01 | 0.2 | 3000 | o | 2.63m | hit 1 time |
| 220 | 1500 | 0.01 | 0.2 | 7000 | x | - | - |

TABLE II
PARAMETER AND PERFORMANCE TABLE