Final Lab Report

11812031 Xinyi Zhou
11811116 Biao Wang

Project Report for EE346
Department of Electronic and Electrical Engineering
Southern University of Science and Technology

June 13, 2022

Contents

1 Introduction

EE346, the course of *Mobile Robot Navigation and Control* delivers to us both theoretical knowledge and practical experiences in autonomous driving. The final project takes the form of a competition where we have options to complete tasks including autonomous navigation, lane following, and target search (Aruco marker detection) within the environment shown in Figure 1. Each task earns some points for our team.
In this report, we will first identify the tasks we have completed, then describe the system we utilized for algorithm developing, after which provide step-by-step explanations for our algorithm, and finally conclude our performance in the competition.

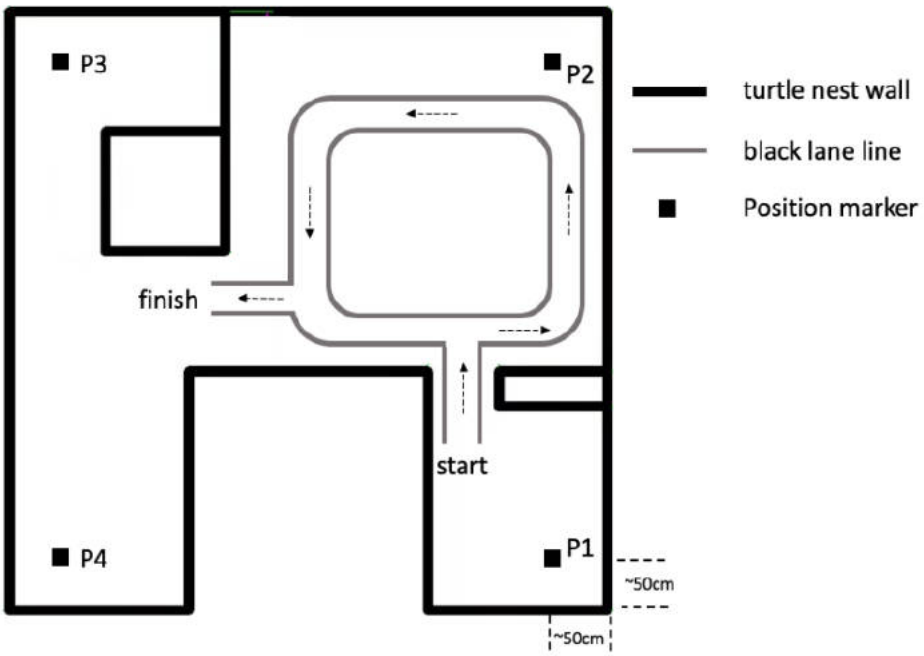


Figure 1: Robot environment with the racetrack to traverse and positions to visit

1.1 Accomplished Tasks

1.1.1 Task 1: Point Navigation

There are in total four points in the map. P_1 is the starting point and ending point of each lap while P_2, P_3, and P_4 are points each worthy of 10 points. During the task of lane following, we managed to navigate our

robot to each point to earn points.

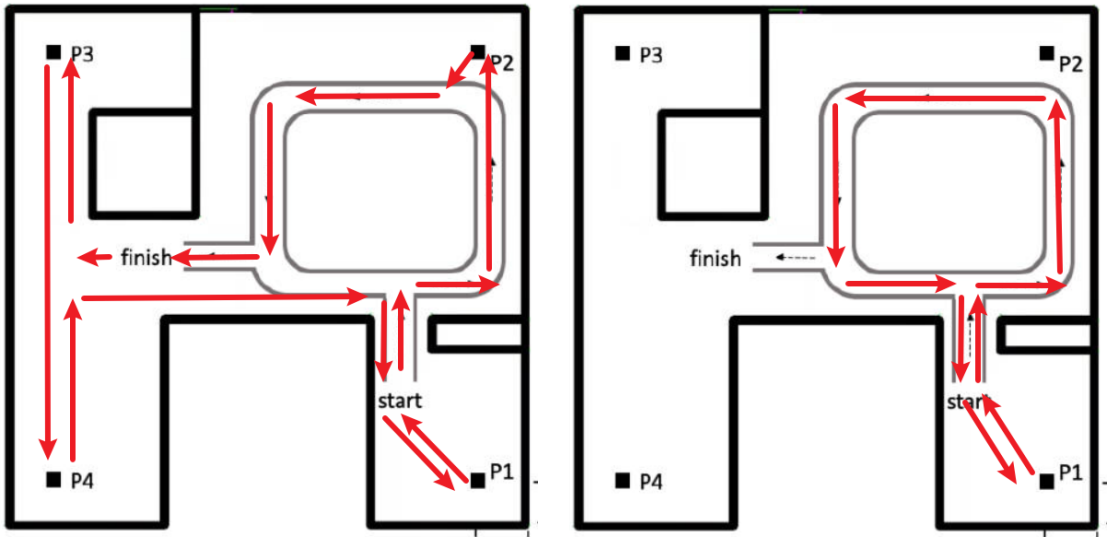
1.1.2 Task 2: Lane Following

Shown in Figure 1, our robot is asked to follow the black lane line to complete the lane following task after launching itself from P_1. Finishing this task can give us 20 points with a 1-point penalty for cutting each corner. We designed an algorithm based on images received by the robot to complete the task.

1.2 Round Route

In round 1, we did lane following and navigation. Each lap follows this order: $P_1 \Rightarrow Lane\ Following \Rightarrow P_2 \Rightarrow Lane\ Following \Rightarrow P_3 \Rightarrow P_4 \Rightarrow P_1$.

In round 2, we only did lane following. The whole traverse follows this order: $P_1 \Rightarrow Lane\ Following \Rightarrow P_1$.



(a) Round 1 (b) Round 2

Figure 2: Route for each round

2 System Description

2.1 Hardware System

In hardware, the whole system consists of a turtlebot3 burger robot and a linux-based PC, as shown in the Figure 3. These two parts communicate with each other via wifi. A 360° LiDAR and a camera are mounted to the robot. The LiDAR is used for SLAM and navigation, while the camera for RGB image capture.

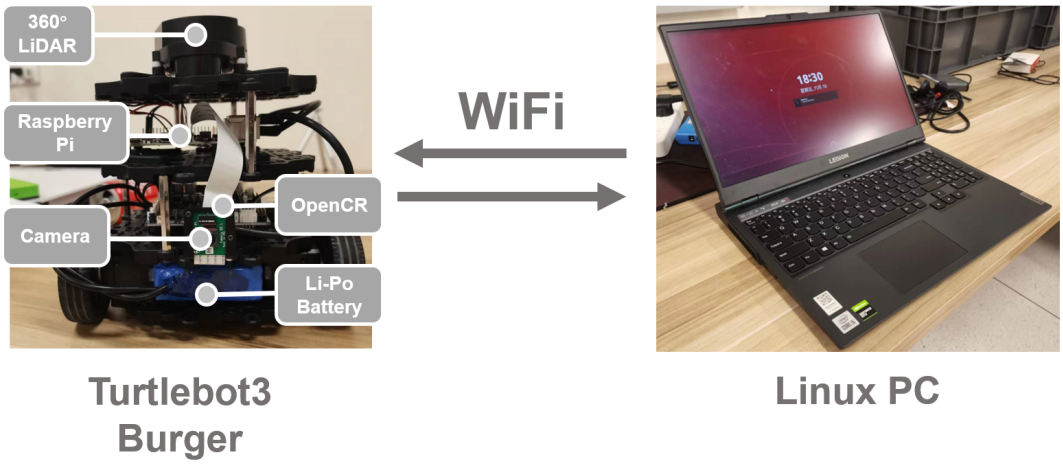


Figure 3: Hardware System

2.2 Software System

In Software, the system consists of a turtlebot robot as the server and a PC as the host. The PC will host the master to monitor the whole system, including varies topics and actions, as shown in the Figure 4. During operation, the robot will continuously send data captured by LiDAR and camera to the corresponding topics. And after processing and reasoning by the PC, commands will be sent to the topics, which are subscribed by the bringup node running in the robot. The robot will move according to these commands.

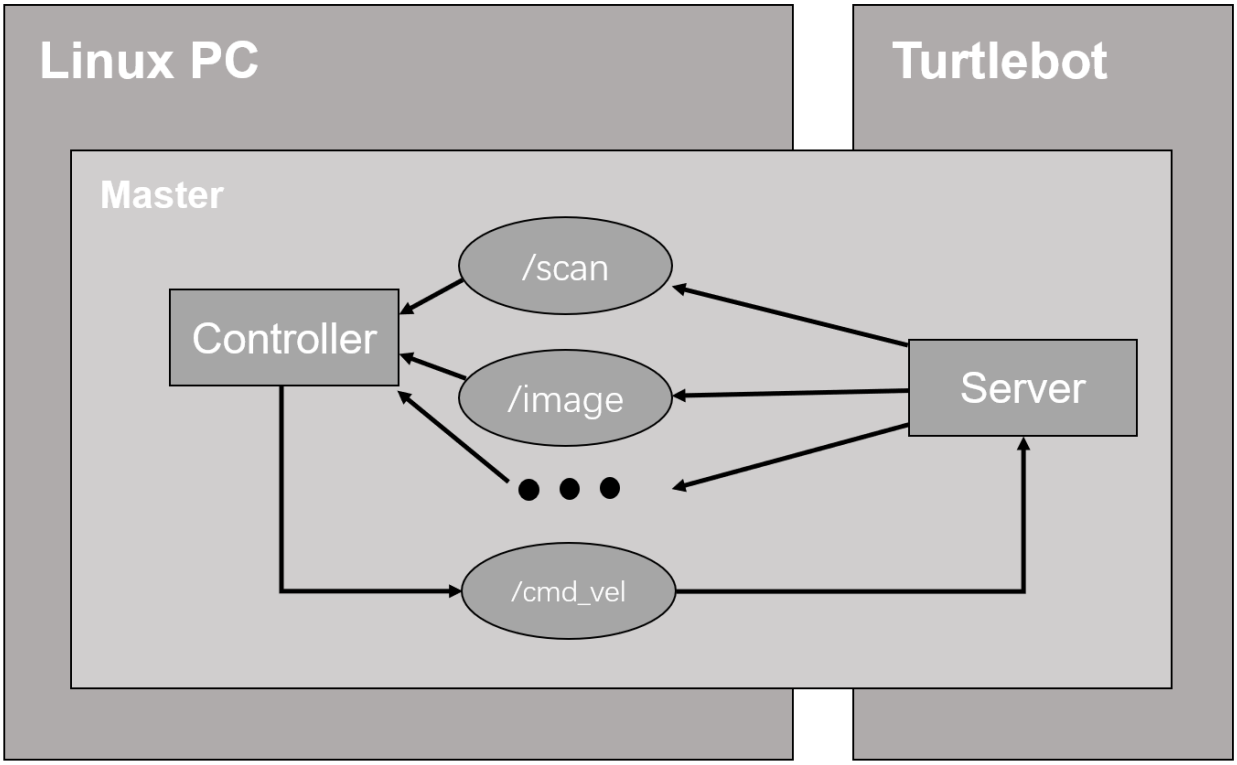


Figure 4: Software System

3 Methodology

Our complete algorithm can be shown in the form of an rqt graph as Figure 5.

3.1 Mapping

To do the navigation, we first need to generate a map for the environment. We used the *turtlebot3_slam* package, the *turtlebot3_teleop* package, and the visualization tool RViz.

1. Run the *roscore* in the remote PC
2. Launch the turtlebot

roslaunch turtlebot3_bringup turtlebot3_robot.launch

3. Launch the SLAM package

roslaunch turtlebot3_slam turtlebot3_slam.launch

4. Run the RViz

$rosrun\ rviz\ rviz\ -d\ `rospack\ find\ turtlebot3\_slam`/rviz/turtlebot3_slam.rviz$

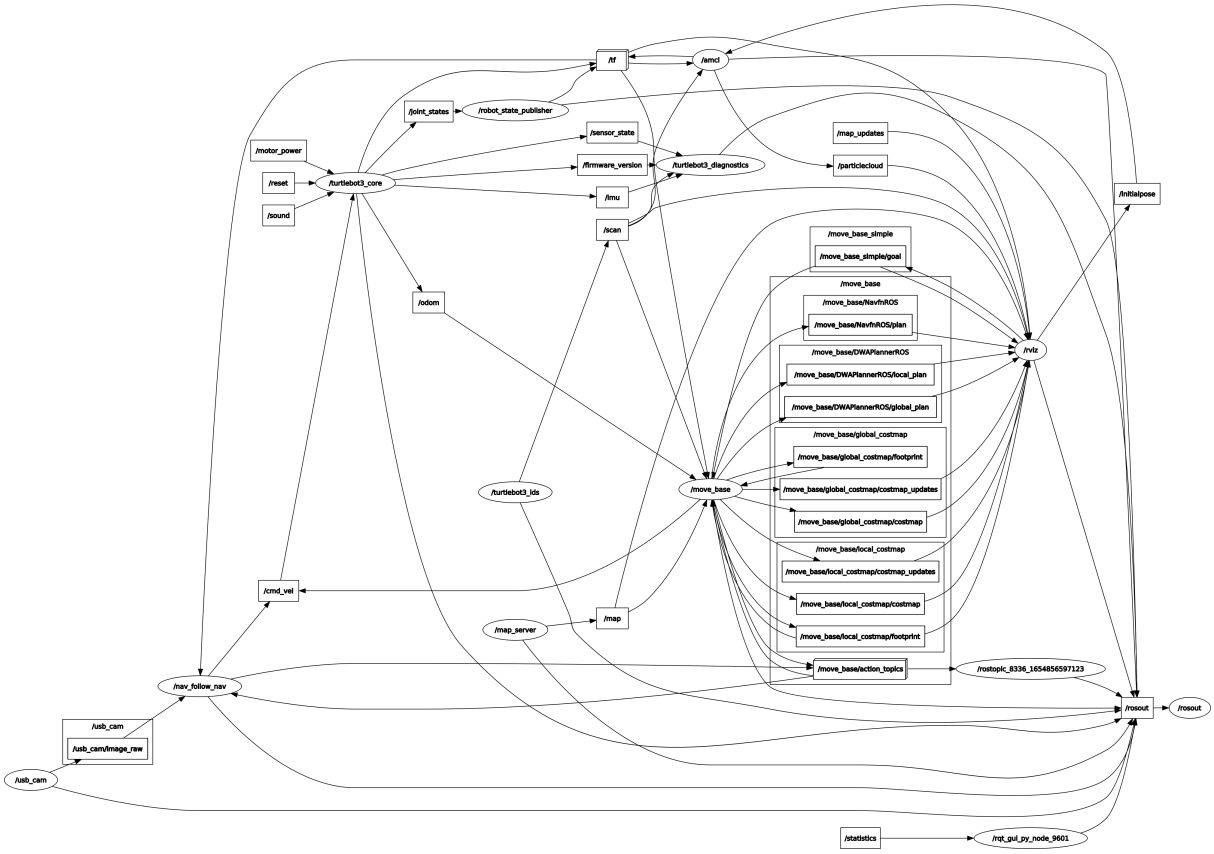


Figure 5: The rqt graph of our algorithm

5. Move the robot to scan through the whole map
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch 6. At last, we saved the map
rosrun map_server map_saver -f ./map

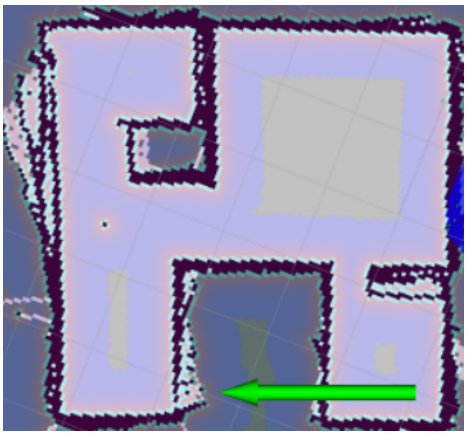
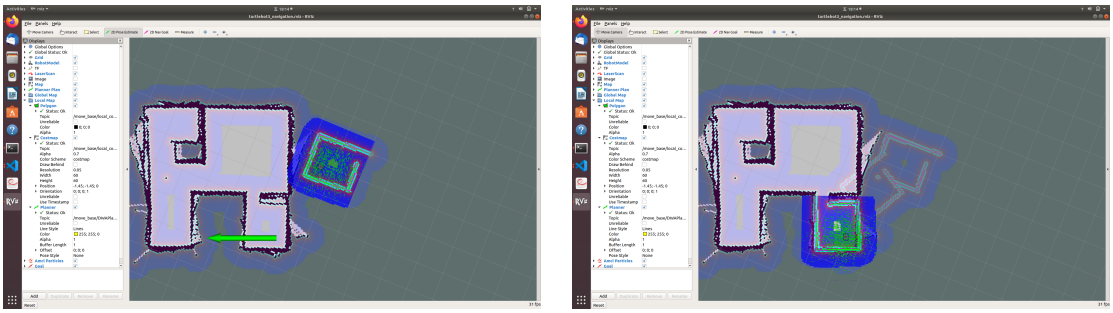


Figure 6: Mapping Result

3.2 Navigation

3.2.1 Pose Initialization

Launching the robot's LDS sensor and RViz, we would find the robot located outside the saved map. We first need to correctly locate the robot in the map so that its sensor's data can correctly overlap the saved map. Therefore, we followed the instructions in Initial Pose Estimation to initialize the pose of the robot.
First, click the 2D Pose Estimation button to manually locate the robot by draging the green arrow. Then, use the keyboard teleoperation node to finetune the location and pose of the robot in the map.



(a) Pose Initialization (b) Result

Figure 7: Pose Initialization Process

3.2.2 Navigation Goal Setting

By navigating the robot in the RViz, we found a topic named move_base/goal that sets the robot's navigation goal. Therefore, we wrote an algorithm to send target robot poses to this topic one by one so that the robot can reach each intermediate point and navigation point one after the other.
The python code uses *SimpleActionClient* to send *MoveBaseGoal* messages to make the robot move. The code can be found in the */scripts* file.

3.3 Lane Following

To do the lane following, we need to process the captured image to get the lane line information. Firstly, we need to binarize the RGB image based on HSV range to filter out the part of the lane line. Then, we need to get Bird Eye View(BEV) images via the homography. With the BEV image, the controller can be able to get to know the status of the robot, and therefore send corresponding commands. Specifically, we divide the BEV image into three parts evenly from above to below, and the boundaries are shown as red lines. We use the top two parts to compute moments respectively. The moment center of the first part is shown as a blue circle, while the center of the second part is shown as a red circle. The result is shown as the Figure 8. And when the "m00" part of the moment is under a threshold, we can know that there are no lines detected. At this time, a "NULL" will be shown instead a circle. Here, after analyzing the scenario, we define five different statuses to describe situations that robot will meet: Straight, Turn, Fork, Recovery, and Wall-avoiding. During the lane following, the robot will switch among different statuses.

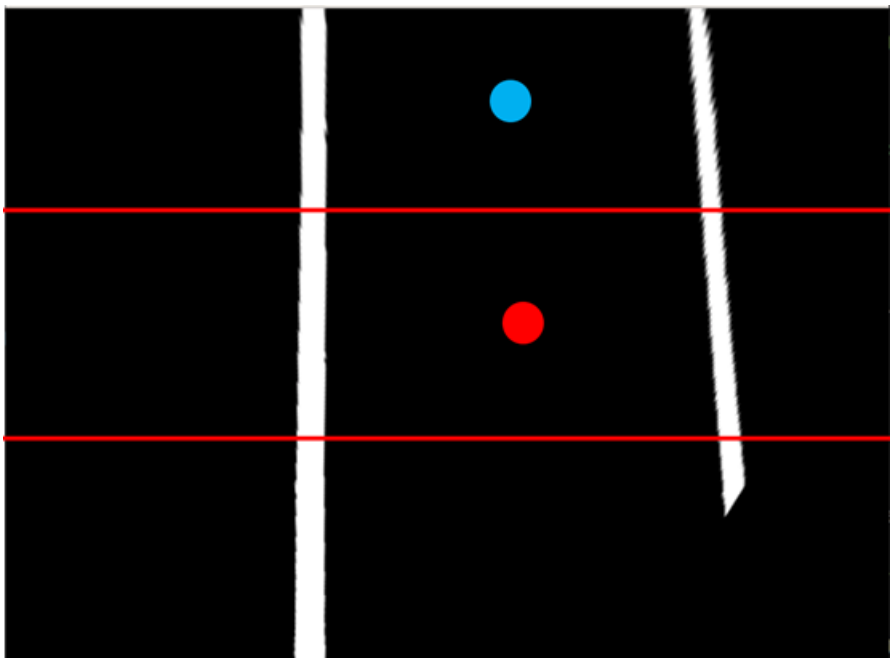


Figure 8: Status of Going Straight

3.3.1 Going Straight

Most of the running time, the robot would be at the "Straight" status, when there are blue circle and red circle at the same time, as shown in the Figure 8. This status describes the situation that the robot needs to go straight along the lane. The robot might deviate from the lane slightly, but never significantly. At this status, the extent of deviation of the red circle is used for the PD control of both the angular and the linear velocity of the robot.

3.3.2 Turning

When the robot meets the corner of the lane, it would be at the "Turn" status, when there is only red circle, as shown in the Figure 9. This status describes the situation that the robot needs a sharp turn. At this status, the extent of deviation of the red circle is used for the PD control of both the angular velocity of the robot, while the linear velocity will be fixed to a small value, 0.15, specifically.

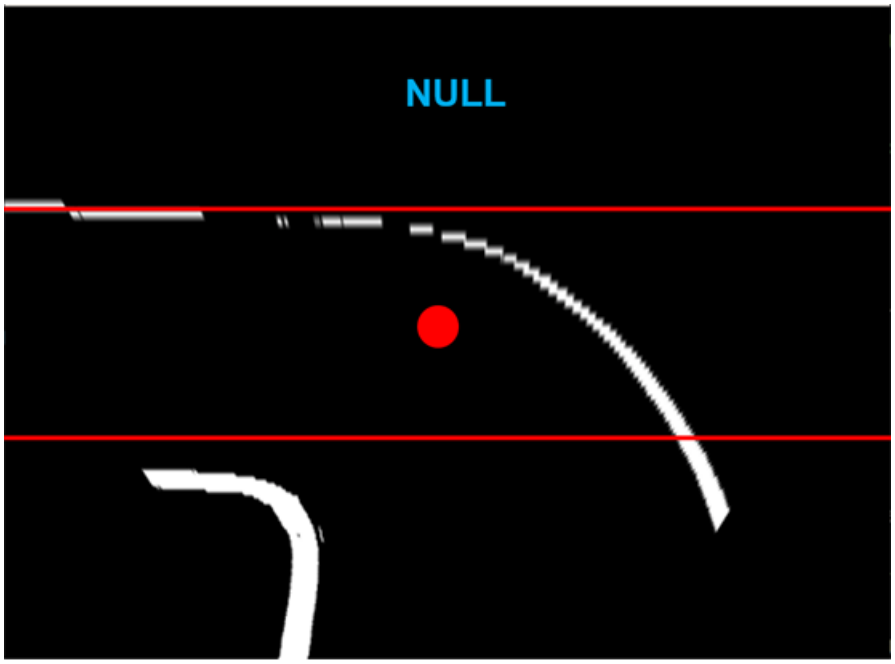


Figure 9: Status of Turning

3.3.3 Fork Road

When the robot meets the fork road, it would be at the "Fork" status, when there are horizontal lines detected, as shown in the Figure **??**. This status describes the situation that the robot needs a sharp turn. At this status, the robot would be set to turn 90° right or left, according to the predefined path.

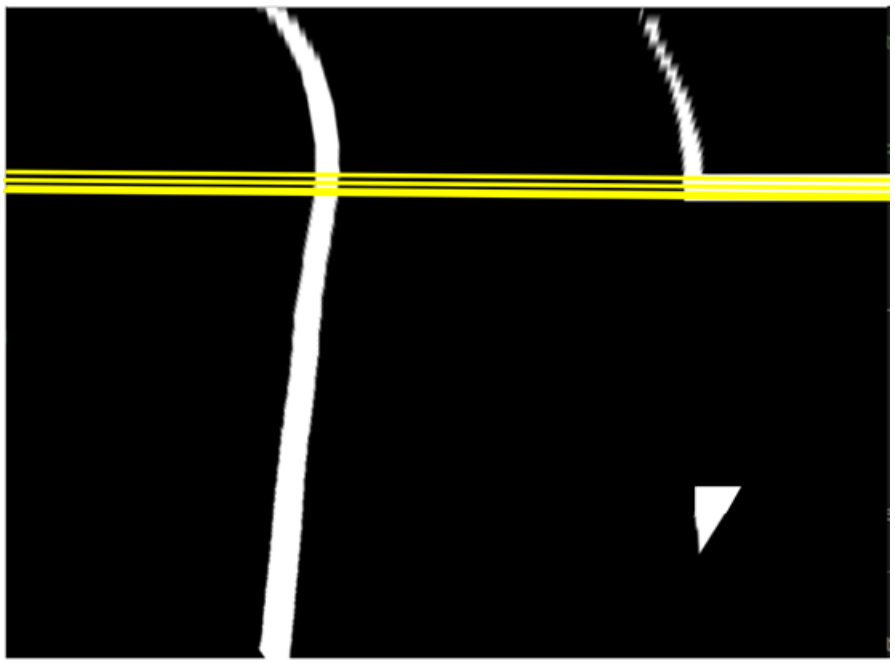


Figure 10: Status of Fork Road Turning

3.3.4 Recovery

When the robot unfortunately deviate the lane significantly and could not see any lines, it would be at the "Recovery" status, when there are no circles, as shown in the Figure 11. This status describes the situation that the robot needs to recover to see the lane. At this status, the linear velocity of the robot would be fixed to be 0, while the angular velocity would be set to be last value.

3.3.5 Wall-avoiding

We also detect the wall continuously based on the HSV field. When the aera of the wall over the threshold, we can state that there is a wall in the

Figure 11: Status of Recovery

front of the robot, and the robot needs a sharp turn to avoid the wall.

4 Conclusion

We first cooperated in finishing the tasks: mapping(Xinyi Zhou and Biao Wang), navigation(Xinyi Zhou), and lane following(Biao Wang). After developing the codes, we helped each other in debugging and tuning the parameters.

We achieved over 90 points in the first round and only over 30 points in the second round. Watching other teams' performance, we realized that we might have sped up our robot and added the Aruco marker detection to achieve better performance.

All in all, we really appreciate the teachers and this course for providing us with a valuable experience in autonomous driving, which will sure benefit us in our future learning process.