SE498 Lab 6 (Mini Project) Report

Tianqi Liu(tliu51) Zhenghe Shangguan(zhenghe3)

Introduction

This project aims at designing a complete controller, a obstacle avoidance algorithm as well as the wall following of Jackal UGV robot with a single VLP-1 LiDAR as its extra sensor, the IMU as well as the tyre odometry in Jackal itself are also used for designing the whole framework. The final goal is letting Jackal take turns and bypass some randomly placed obstacles in collision-free mode in the given Gazebo ECEB map.

How to run:

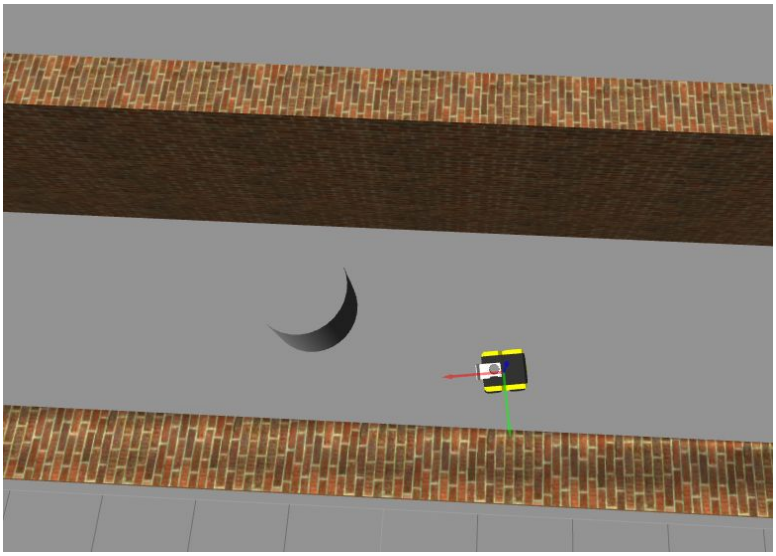
The same as last Jackal lab of passing btw two pillars.

Terminal 1:
use jackal_simulator in Lab 5 (which is not included in the src.tar.gz). After “catkin_make” and “source devel/setup.bash”, run “roslaunch jackal_drive jackal_drive.launch”

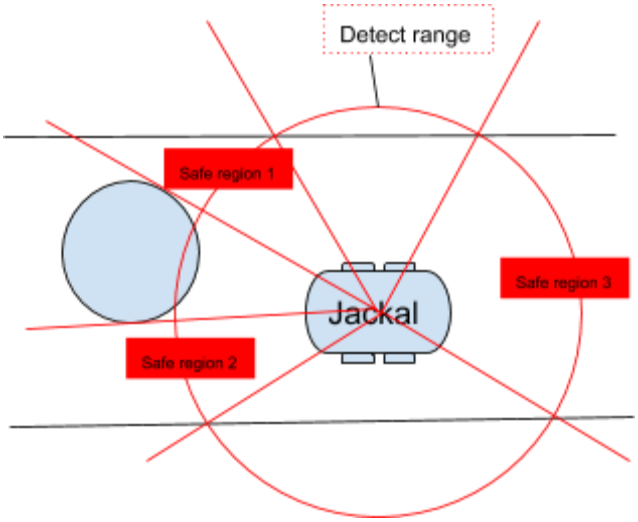
Terminal 2:
After “catkin_make” and “source devel/setup.bash”, run “rosrun jackal_drive jackal_drive_node”

(1) Obstacle avoidance using LiDAR only. 10pt

We use VLP-1 LiDAR on Jackal in ECEB map. For the obstacle avoidance, we used LiDar information. We consistently scan the surroundings of the robot and detect the obstacles within 2 meters of the robot. We store the transition angle between “obstacle region” and “safe region”. if there are multiple “safe” regions, we will choose the one closer to the heading of the robot. Then we calculate the middle point of the “safe” region and try to navigate the robot to it.



The details of how we implemented the algorithm into our controller can be find in section (3). And the basic thought looks like the following graph:



(2) Wall following/lane/hallway edge detection. 10pt

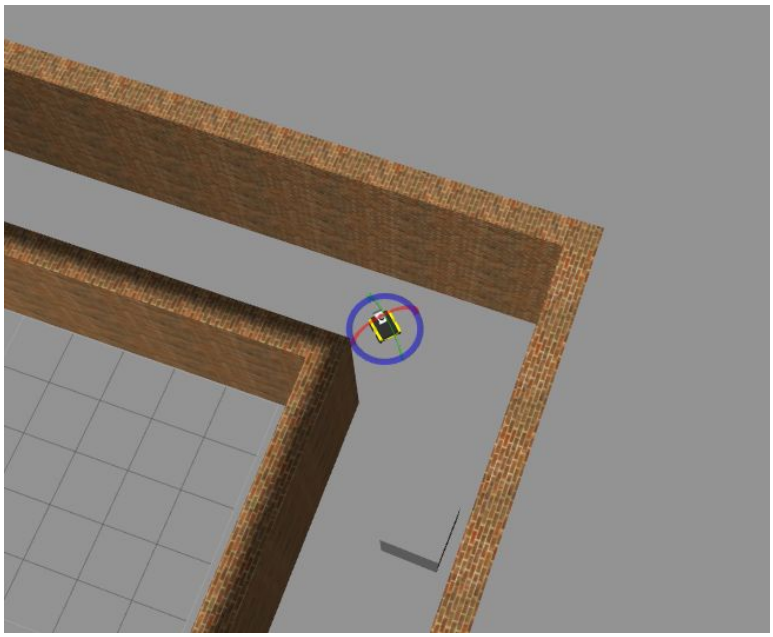
We implemented three different hallway following strategies using LiDAR information and the robot will try to move through the center of the hallway.

For first 20 meters, we try to keep a 1.25 meters distance from the right wall. We scan the -15 to 15 degrees for left and right direction, and calculate the average distance to the left and right wall. Then we will compare the current distance to the right wall and the desired distance. If the robot is too close to left wall, it will steer right. Otherwise, it will turn left.

Later, if the obstacle avoidance code is working properly, it should detect the hallway in the front and treat it as “safe” region. The robot will always steer towards the center of the hallway, thus keep the robot following the hallway.

If the obstacle avoidance code get into trouble and could not find the safe region, the robot will try to follow the left wall and keep a 1.25 meters distance from it. The implementation is similar to the first method. We scan the -15 to 15 degrees for left and right direction, and calculate the average distance to the left and right wall. If the robot is too close to left wall, it will steer towards right. Otherwise, it will turn left.

The combined algorithm works fine in our ECEB hallway environment. The second method might run into problem sometimes, especially at the corners. However, the third method can always save the situation when obstacle avoidance is not working well.



(3) Controller design and tuning. 5pt

Control I: for simple wall following

The overall intuition of our wall following controller is from Lab 3 with some modification. We still try to keep a constant linear velocity = 0.6 m /s, and put our controller on the angular velocity on z-axis for a wall following algorithm.

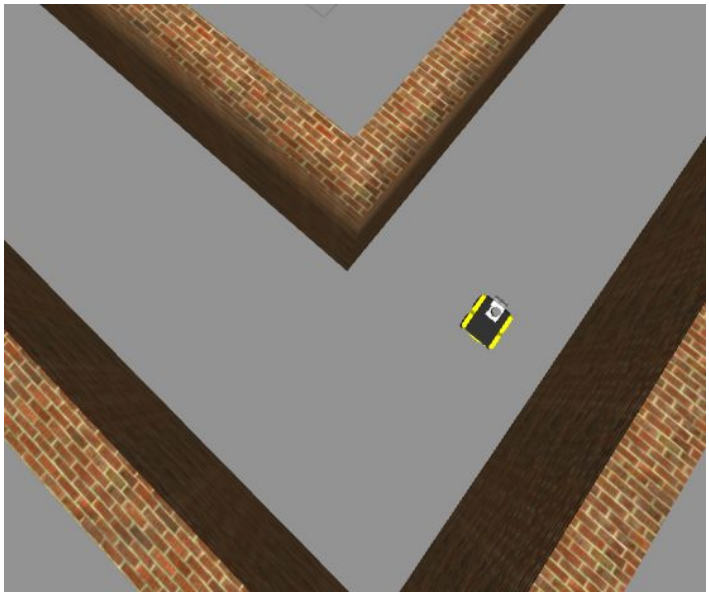
Firstly, we try to use a simple Proportional Controller:

angular.z = Kp * error_distance_to_target_wall

Moreover, since at the very beginning, we can only detect the parallel distance from the right side wall (the left side wall is more than 10 meters away from Jackal's left side at the initial position). Thus, we divide our control strategy into two parts. For the first 20 units of Jackal's tyre odometry distance, we follow the right wall. From then on, we will always follow the left wall for taking turns and going straight.

However, we find the performance of this simple P-control did not work well under certain circumstances. The key reason for this is that we only use the -15~+15 degrees' corresponding distance data of Jackal's left & right sides, and we directly send this distance

error to produce the positive or negative angular distance. However, this relationship does not always hold since we are only use the relative distance error without considering Jackal's current pose, especially the heading direction. For example, in the following case, Jackal should keep its current heading and slightly turn left to go back to the central lane of the road, however, since it only considers about the distance error between the left wall and right wall, it will keep turning right and finally hit the right wall.



To rectify this kind of error and improve the performance, we then introduce a new item into the controller, our final controller for simple wall following looks like this:

```
vel_msg.angular.z = fmax(fmin((total_dist_l-desired_dist)*0.35+fmax(fmin(pose_angle*(-1.2),0.45),-0.45),2),-2);
```

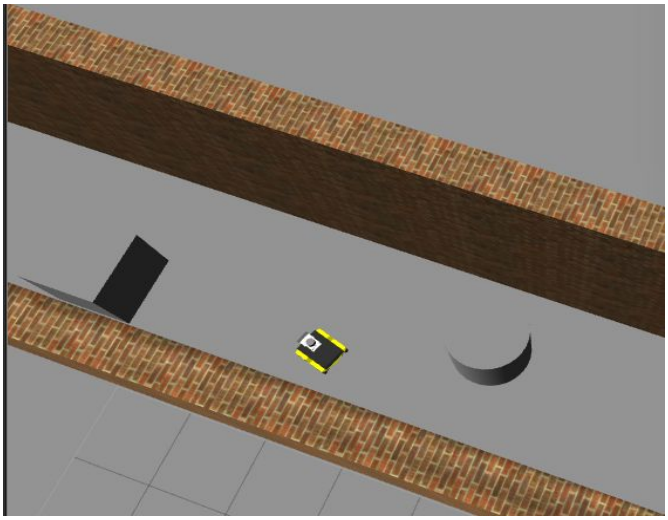
In this version, we introduce the current pose_angle, which is the real Jackal yaw angle by using the corresponding LiDAR angle in the minimum right wall distance. We treat the forward direction along the wall as yaw = 0, thus the pose_angle itself is just the yaw error, and we add a relatively large gain for this P-part control, also with some maximum and minimum value restrictions. This behavior of this setting will be: the Jackal controller will take its heading very carefully, and try to always keep the heading parallel to the wall direction unless the left wall and right wall difference is very large (which means Jackal is taking turns). When Jackal is running along a relatively long straight line, the first P-part in controller will gradually "drag" itself back to the middle of lane with keeping the heading almost always parallel to the wall.

Control II: for obstacle avoidance

When we implement the obstacle avoidance controller, we are trying to let Jackal detect the good regions around it, which means the region with relatively large free space to drive inside. In this eceb map, usually Jackal will have 1~3 good regions depending on its pose and place.

```
else if (good_regions.size() == 2) {
    vel_msg.linear.x = 0.3;
    float center1 = good_regions[0][0];
    float center2 = good_regions[1][0];
    float width1 = good_regions[0][1];
    float width2 = good_regions[1][1];
    float angle_err = 0;
    if (width2 > width1) {angle_err = center2-PI;}
    else {angle_err = center1-PI;}
    cout<<"Angle"<<angle_err<<endl;
    vel_msg.angular.z = fmax(fmin(angle_err * 0.35, 2), -2); //constraint
  }
```

For the controller, it will choose the largest good region in front of itself and pick up the middle angle of that good region as its desired heading direction, with a P-control. At the same time that it detects obstacles in front of itself, it will also slow down the linear speed to give the controller more time to adjust its pose and get through the obstacle, as you can see in the video.



(4) Team management 5 pt

Zhenghe and Tianqi finished all the requirements of this report together and shared the same contribution. Since we are the only group left, we did the whole project by our two people.