Team 7 Lab 2 Report

Reactive Autonomy

Hong Ray Teo, hteo6@gatech.edu, Fan Han Hoon, fhoon3@gatech.edu, Ethan Tse, etse8@gatech.edu

***Abstract*—This is our report for lab 2 of ECE 4804 Autonomous Mobile Robotics. In this lab, we primarily set out to create two different reactive autonomy packages for our car: automatic breaking, and follow the gap. These packages were written in ROS 2 and built of a template provided to us by the TA team. We were successfully able to complete these tasks both in simulation and in hardware.**

I. Introduction

REACTIVE autonomy is a simple yet key part of robotics. A robot that has reactive autonomy can more readily adapt to a dynamic environment than one that follows a strict set of guidelines. With such a key importance to the field of robotics, Team 7 worked to create two different reactive autonomy features: automatic breaking, and follow the gap. In addition, we also installed a new AMRDocker container on the car itself. This was an update provided by the TA team.

Automatic breaking allows the car to come to a stop when it thinks it is about to hit an obstacle. We used a LiDAR mounted on top of our car to detect obstacles. Follow the gap is a method for relatively simple self driving capabilities. In essence, the car scans its surroundings using the aforementioned LiDAR, marks out any obstacles in its way, and navigates to gaps in between those obstacles.

II. Background

Team 7 worked on the lab in Klaus 2446. The team was provided with AMR packages that were updated by TA Matthew and the starter codes for the reactive autonomy. The team was expected to develop a working algorithm that enables the robot to perform automatic braking and gap following without any human interference.

III. Procedure

Our team started out by working on both automatic braking and follow the gap in simulation, before transitioning towards testing on the real car.

The idea behind automatic braking is to stop the car if the time it takes to hit any obstacle is below a threshold distance. This is known as the Instantaneous Time to Collision, with its definition given by Equation 1.

$$ittc = \frac{r}{max(v_x cos(\theta), 0)} \tag{1}$$

where r is the range measurement obtained from the LiDAR, v_x is the forward car velocity obtained from the wheel encoders and θ is the angle of the obstacle with respect to the car. The max function ensures that we get an infinite ittc rather than a negative value whenever the car is moving away from an obstacle. This algorithm was implemented and tested on simulation successfully after a few attempts. The few points of contention we faced were determining the correct sign when calculating the range rate and finding a suitable threshold value for ittc.

In follow the gap, we seek to continuously steer the car towards the largest gap, or in other words, the widest open space without any obstacles. This is achieved using LiDAR data by identifying the greatest number of consecutive range measurements that all exceed a certain threshold value. To improve obstacle avoidance, a safety bubble around the nearest obstacle is also drawn that sets certain forbidden angles that the car can steer towards. Similar to automatic braking, we had little trouble running the algorithm in the simulator and only had to tune several parameters.

However, we faced a lot more challenges when moving from the simulation to the actual car. The first problem, which was expected, was discarding LiDAR readings affected by other components on the car physically blocking the LiDAR. This was accomplished by only keeping LiDAR data corresponding to measurement angles 180° in front of the car. After much debugging, we found that another problem was that our LiDAR was randomly returning chunks of 0s. In automatic braking, we removed these erroneous values from calculations to determine whether to stop. In follow the gap, we accounted for this by setting any such values to the nearest non-zero value before it. Since the robot still had trouble navigating tight turns, we also had to be less ambitious by lowering our speed and selecting the middle of the widest gap as the best goal point, rather than the furthest, for better stability.

We physically worked together for the entire lab, but the tasks could roughly be split as such:

Ethan handled the data extraction and sending of commands to the vehicle. In addition, he used his laptop to run the simulations as well as interface with the car.

Han was in charge of the hardware. Besides physically moving obstacles and the car around during testing, he debugged some problems that we were facing with our LiDAR and serial connections.

Ray was responsible for debugging and tweaking the algorithms. He also wrote code to perform data manipulation in numpy.

IV. Software

For both automatic braking and follow the gap, the program is event-driven: data is processed and commands are sent every time a new LiDAR scan is completed.

The algorithm for automatic braking is as follows:

1) Only keep LiDAR data corresponding to measurement angles 180° in front of the car

2) Calculate ittc for each angle in the LiDAR measurement
3) If any of them are less than a threshold:
   Send a pulse of drive commands to stop the car

The algorithm for follow the gap is as follows:

1) Preprocess LiDAR data
   - Set any 0s in the data to the nearest non-zero value before it
   - Only keep data corresponding to measurement angles 180° in front of the car
   - Average data points over a window to smooth out noise
   - Set a maximum cap on data values
2) Find the closest point among the processed data points
3) Draw a safety bubble around that point to avoid traveling in a direction with obstacles nearby
4) Find the largest gap among the remaining data points
5) Find the best point (the midpoint) to travel to within that gap
6) Set the car's steering angle to the point identified

V. Results

Team 7 was able to achieve the goals of automatic braking and gap-following by using the AMR package provided by ECE 4804 TA Matthew and the algorithms developed by the team. The preliminary testing was conducted in the F1Tenth simulator where the team figured out the impact of different parameters such as the convolution size of the LIDAR average points, threshold distance to reject the LIDAR points, safety bubble size, and threshold size of a gap for the robot to follow.

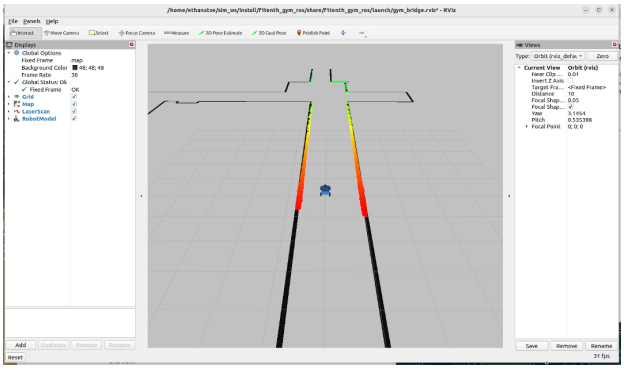

Fig. 1. Photo of the F1Tenth simulation running in RViz.

According to Figure 1, the results in the F1Tenth simulator show promising results as it was able to automatically break when it was in proximity to a wall and able to avoid collision with the walls in the simulator. The optimal parameters for the simulator are observed in Table 1.

Parameter	Optimal Value
Convolution size	10
Threshold distance	5.0
Safety bubble size	300
Threshold size	0.7

TABLE I
PARAMETERS OF ALGORITHM TESTING IN F1TENTH SIMULATOR.

The team also observed that a higher convolution size would result in a less sensitive algorithm as too many Lidar points were accounted for in the calculations. On the other hand, the speed of the robot will also affect other parameters such as threshold distance to reject the LIDAR points, safety bubble size, and threshold size of a gap for the robot to follow, and the tuning of these parameters relies on trial and error.

After the success of the F1Tenth Simulator, the team uploaded the algorithm script to the physical robot and did multiple test runs on the designated track shown in Figure 2. However, the initial results were disappointing as further calibrations on the parameters were required. The robot had encountered issues such as the Lidar's sensitivity being too low for the robot to react and other groups also commented on the steering issues where the steering packages provided had bugs on the steering angles. For instance, any high steering angles calculated in the algorithm might not be reflected in the steering speed as the robot speed of 1 would cause the steering speed to drop 50% of the original value and the results are seen in Figure 3.

Fig. 2. Photo of the designated track for the test.

Fig. 3. Photo of faulty steering angle issue.

After multiple experiments on the parameter tuning, the team adopted a new set of parameters as shown in Table 2:

Parameter	Optimal Value
Convolution size	10
Threshold distance	3
Safety bubble size	125
Threshold size	0.5

TABLE II
PARAMETERS OF ALGORITHM TESTING IN THE DESIGNATED TRACK

The new set of parameters enabled the robot to avoid collision with the walls and the obstacles of the designated track but it will occasionally encounter issues such as failing to detect the better gaps in the track and resulting in a u-turn motion in the track. The bug in the package provided by Yahboom also suggested that further investigation was required as it affected the performance and tuning of the robot for the current lab and the upcoming localization lab.

VI. Conclusion

The goal of this lab was to develop two different reactive autonomy packages in ROS 2: automatic breaking, and follow the gap. We were successfully able to demonstrate both this capabilities. The biggest challenges we ran into were primarily when transitioning from simulation into the real world. Most of our issues stemmed from erroneous measurements coming from our LiDAR, but we also had some more minor issues with the steering commands sent to our car. We have been told that that the firmware for steering will be updated before the next lab and are excited to hit the ground running on that. Ultimately, we achieved our goal of designing reactive autonomy for our car, and enjoyed the development process overall.

Acknowledgment

The authors would like to thank Dr. Michael West, Matthew Woodward, and Sidhant Gulati.