ECE 5532 - Autonomous Vehicle Systems I

Adaptive Cruise Control with Audibot

Laura Schwabel, Maddy Hicks, Nick Durham

Abstract

Our goal for this project was to set up a control system for adaptive cruise control within existing Audibot simulations. The project leverages two instances of existing Audibot simulations to create a lead car and following car. The challenge of the project was to control the following car to a speed based on the distance from the lead car.

This project focuses on three levels of automation as described below.

Level 1: Simulation uses GPS position of the two cars to determine following distance. This is used as the input to control the speed of the following car so that it maintains a consistent distance to the lead vehicle.

Level 2: Add 2D LIDAR to the simulation to detect distance between lead car and following car through a LIDAR scan.

Level 3: Use the following car's built-in camera to compute the distance between the lead car and following car.

Main Algorithm Overview:

The main controls of the adaptive cruise control algorithm are defined in one .cpp file. This node is the "DriveFP.cpp". It contains code that controls the following car's speed and the requested follow distance. This .cpp file subscribes to the following distance signal. The node uses that distance to determine the desired speed of the following car. We determined the following car could not change velocity too quickly or it would not be able to lane follow. The drive code has one publisher node that outputs the steering and velocity command to the following car. This is because we needed to pass the steering angle from the lane following code to continue to lane follow. In debug, there was code to stop the car if the A1 car gets too far from A2 because A1 lost the road and started driving away from the other car. For tracking purposes, we write out every 0.1 second the vehicle speed, distance, error, and command angle. This is run on a different timer to allow the data to be read before the next data point comes in.

We tested multiple different control algorithms and the vehicle would drive off the road. The most stable was an else/if statement that changed the vehicle speed at different rates depending on the distance the vehicle was away from the initial vehicle. This algorithm has a slow start up acceleration but will stay on the road more often.

Level 1: GPS Based ACC

The “GpsCode.cpp” contains all the code for obtaining the GPS position of each car. This .cpp file includes subscribers that read the GPS information from each vehicle as well as a publisher node that outputs the distance between the cars based on GPS location. This had to be updated by subtracting 5 m from the distance because of the fact the GPS sensor is located in the middle of the vehicle.

GPS Functionality:

GPS uses the vehicles’ coordinates and the reference coordinates to determine the UTM of the vehicles. Once we have the UTM from the coordinates, we can determine the distance between the two vehicles. To determine the distance between the two points, we used the square root formula, Pictured below, of the relative positions of the vehicles to the reference coordinates.

Distance between A1 and A2=$\sqrt{(A1.x - A2.x)^2 + (A1.y - A2.y)^2}$

Dynamic Reconfigure server

With any adaptive cruise system, the user will be able to have various thresholds that they can set for following distance. Traditionally in vehicle systems this is a time amount in seconds between the lead and following vehicle. For our project we set this threshold using a dynamic reconfigure server. The dynamic reconfigure uses the basic techniques discussed in class to create the functionality. One precaution that had to be taken was setting the limits on the minimum follow distance such that the simulation would continue to run. In a practical application this distance would be defined by laws, or by the dynamic limits of the vehicle. Similarly, our minimum distance was set to 27 and the maximum distance was set to 80 which was completed in the Final_Project.cfg file.

Level 2: LIDAR ACC

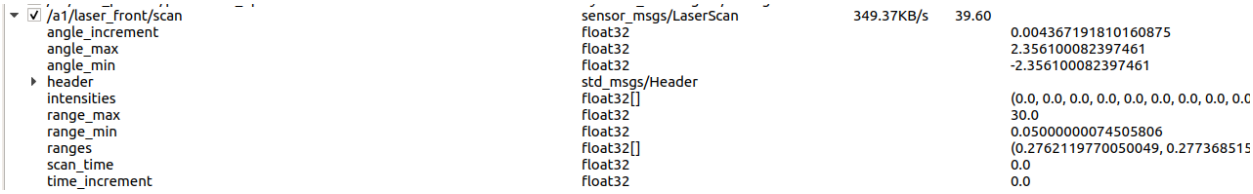
Urdf

We added the hokuyo lidar package to the audibot urdf files found in the melotic subfolder. We had to move the new file into the final project package because the melodic subfolder was read only on the computer.

Once the vehicle urdf was corrected, we had to correct all the launch files to call the new urdf file. All of the launch files were also read only, so we copied them into our package and modified them there.

Distance calculation

To determine the distance that the lidar was reading, we had to determine the angle of where the front car would be. To do this, we drove the car using the GPS code and printed the angles where the range was between 30 m and 1 m. this gave us that the location of the range data we wanted was listed in the array between points 450 and 560. These points convert to the angles 103° to 126°. Below is pictured the information shown in RQT for the Lidar sensor data.



By cutting the number of points the equation uses, the while loop is able to run as few points as possible while still being accurate. To calculate the distance, the loop takes the weighted average of the current range. A 0.75 weight was used to keep the current number changing at a slower rate. This weight was done to keep the car on the road and able to react to the lines on the road.

We compared the Lidar distance calculation to the GPS calculator before publishing the distance msg with the Lidar node. This allowed us to look at the lidar data at a steady distance rate and compare it for accuracy.

Level 3: Audibot Camera ACC

Current Progress

As a starting point for the camera implementation, we leveraged the opencv_package that was used in class. This package gave us a starting point to integrate camera features into our simulation package.

Audibot Camera Topic

When running the audibot simulation the camera images are published as a topic, a1/front_camera/image_color. This topic is mapped in our launch file to 'image_in' which we leverage in the RgbHough.cpp file to support the image processing features.

Image output

In the RbhHough.cpp file we post-process the image into several new images. The processing of these images was covered in class. In order to get our images to process correctly we did change parameters in the arguments of the threshold function.

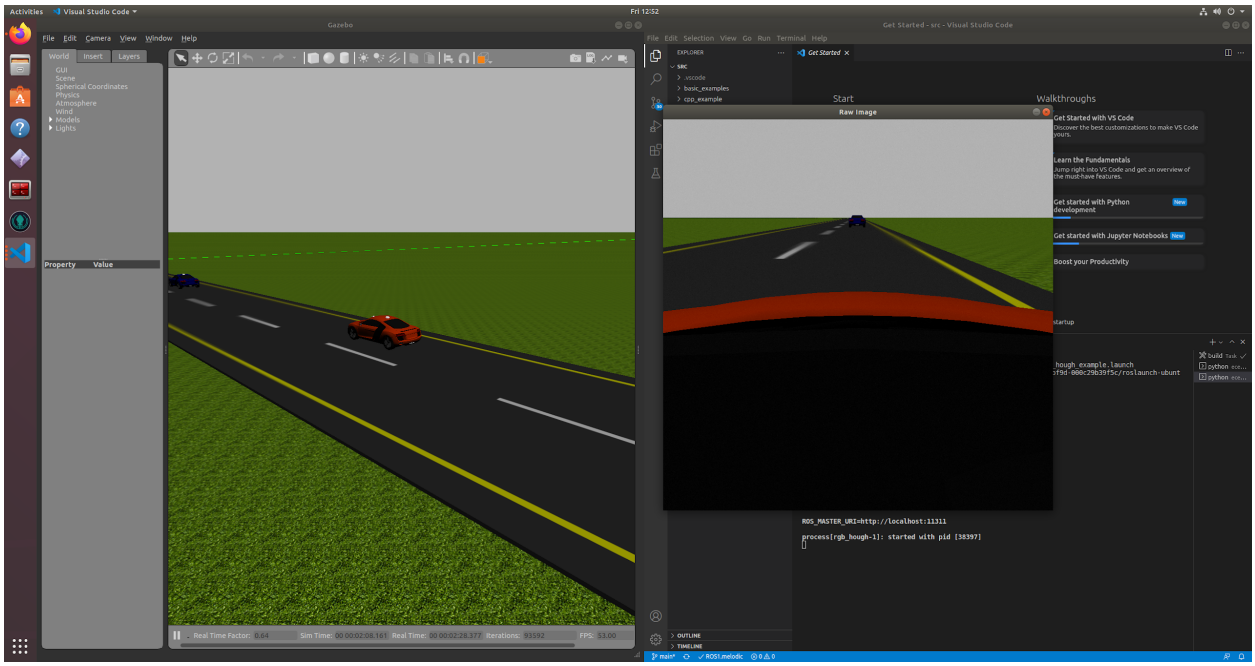
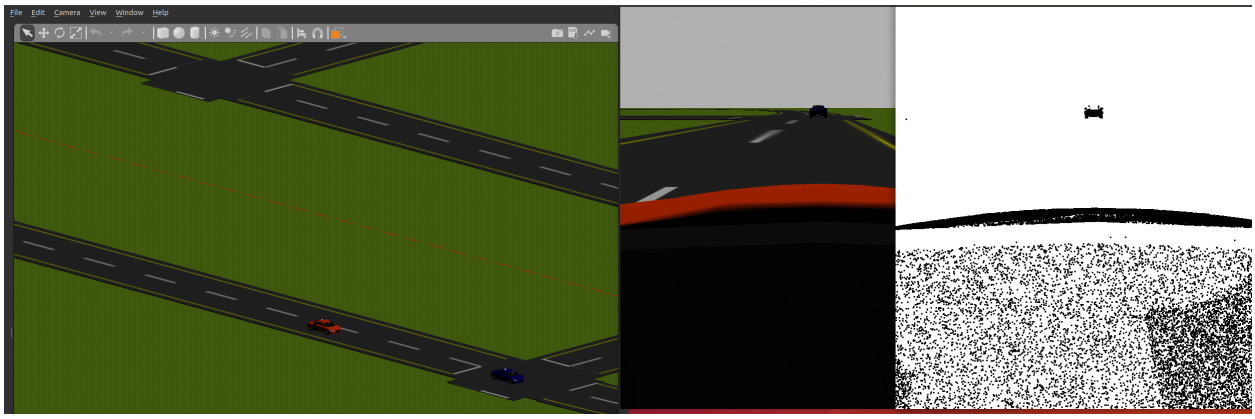


Image Processing

The image below shows the comparison between the erode image, and the current vehicle positions relative to each other. The lead vehicle is shown as a small cluster of black pixels in the erode image.



Determining Distance from Camera Image

Our goal was to take the camera images we were getting from the front camera of the following car and find the distance to the lead car. We used the raw image for this part of the code. In order to get a distance we first found the size of the image. We also found max and min distance values in the image. Then we attempted to find the distance of a single pixel of the image. This is where we began to have trouble. The distance output from the pixels in the image were extremely small and did not translate to real world distance values.

Next Steps for Level 3

We simply ran out of time to continue implementing the camera portion of the cruise control algorithm. In order to get this algorithm fully implemented, these are the next steps.

First, determine the area of the camera in which you detect the lead vehicle. This could be achieved by setting a constant search area for the vehicle or even doing a scan of object distances and by taking the average distance over an area, determining where the vehicle is based on the distance concentrations.

After determining where the vehicle is, you would need to find the distance from the lead vehicle to the following vehicle. Then this distance could be input to the algorithm for controlling the vehicle speed.

The main algorithm for controlling vehicle speed could then intake gps distance, lidar distance and camera distance and control based on the three values.

Another update needed to improve the project is to stabilize the lane keeping with the added automated cruise control functionality. We had many issues with this as if the vehicle changed speeds two quickly, the vehicle would run off the road. The code works 90% of the time now but every once in a while the car still drives off the road.

Resources:

http://wiki.ros.org/Documentation

https://moodle.oakland.edu/course/view.php?id=262895

https://answers.ros.org/question/141741/calculate-depth-from-point-cloud-xyz/?answer=381085#post-id-381085

GitHub Link

https://github.com/lschwabel/Final_Project_ECE5532_ACC.git