EECS461 – Fall 2016 — Final Project

Mason Sutorius and Miriam Figueroa-Santos

Introduction

With the push for autonomous cars, many vehicles on the road today are starting to incorporate Adaptive Cruise Control (ACC) to enable the vehicle's cruise control to adapt to traffic conditions. ACC enables a car to maintain a safe distance through the use of position control when a vehicle is within a set threshold distance of the front of the car and the vehicle's speed is less than or equal to that of the car. If there is no vehicle within a set threshold distance in front of the car, or if the vehicle in front of the car increases its speed beyond the set ACC speed, non-adaptive cruise control will be enabled. Our task in this project was to create a simple model that models a simple vehicle and implements an ACC system. A haptic wheel served as the steering wheel to our simulated vehicle, and a VR program enabled the vehicle to be driven on a simulated road which included other vehicles through the use of a CAN bus.

Pick Lead Logic

The pick lead logic works as follows. For each car $i \in [1, ..., 6]$, where i is one of the other cars, the code checks to see if car i's position s_i is greater than that of our car, and that the position of the 5 other cars is also greater than that of car i. If so, car i is the lead car. If no car $i \in [1, ..., 6]$ is the lead car, then the lead car is our car.

Automatic Steering Controllers

Let P denote a point on the center of the road, and define (s, n) coordinates, where s is the distance traveled down the center of the road to P and n is the lateral displacement to the right of P. Thus, to enable steering control, the lateral displacement n of the car must be measured and corrected. To correct the error in lateral displacement, the steering wheel angle delta must be altered to turn the car. However, the steering angle delta cannot be controlled without applying a torque to move the wheel. Thus, it is clear that two controllers are needed: one that takes as input the error in lateral displacement n and outputs the desired steering angle to correct for the error in lateral displacement, and another controller that takes as input the error between the actual and desired steering angle and outputs a corrective torque to move the wheel to the desired position and thus steer down the road.

The methods we used for tuning our controllers was to start at the midpoint of the range of some gains suggested by the TA's. After applying these gains to our system, we noticed that the system was overshooting and oscillating (unstable). Then we decided to pass the output of the delta PD controller (desired delta) to the input of the vehicle model and turn off the motor so that no torque was generated at the wheel. This enabled us to tune the delta PD gains without the effect of the torque PID controller (motor). Once these gains were properly tuned, we were able to re-incorporate the torque PID controller into the system and tune the PID gains of the torque controller.For the PD controller, we noticed that we needed to decrease P in order to obtain a stable response and for the PID controller we noticed we needed to increase D. After this tuning, we obtained a stable response with the car driving at 35 mph.response. Our selected gains are given as follows: $K_{P,n} = 0.2$, $K_{D,n} = 0.05$, $K_{P,\tau} = 500$, $K_{D,\tau} = 400$ $K_{I,\tau} = 0.005$.

Encountered Problems

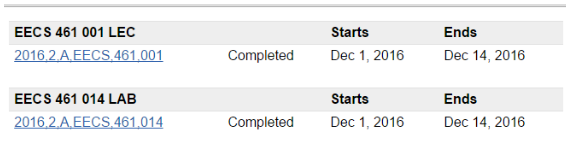
One significant problem that we encountered was tuning the delta PD and torque PID controller gains. To solve this problem, we passed the output of the delta PD controller (desired delta) to the input of the vehicle model and turned off the motor so that no torque was generated at the wheel. This enabled us to tune the delta PD gains without the effect of the torque PID controller (motor). Once these gains were properly tuned, we were able to re-incorporate the torque PID controller into the system and tune the PID gains of the torque controller. Another significant problem we encountered was building our simulink model with models of the same name located within higher levels of the file path. To resolve this issue, we deleted other copies of our build to ensure that files of the same name did not exist at multiple locations within the filepath hierarchy. The last problem we encountered on the project was the use of bits instead of bytes for the message length on the transferring of the data in the CAN bus. We were using a value of 256 (bits) instead of 8 (bytes). We solved this problem by changing 256 to 8.

Work-load Distribution

Work was completed together with both teammates present throughout the duration of the project. Mason drove the computer, while both partners contributed to the intellectual understanding of the project, implementation, debugging, and project execution. Both partners shared work on the final report, with Miriam completing the 'Introduction' and 'Automatic Steering Controllers' section, and Mason completing the 'Pick Lead Logic,' 'Encountered Problems,' and 'Work-Load Distribution' section.

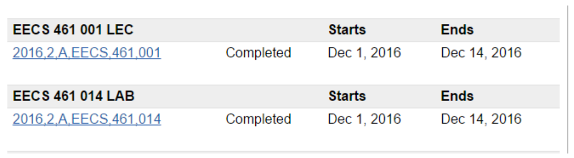
Course Evaluation Receipt

See Figure 1 and Figure 2 for the course evaluation receipt.



EECS 461 001 LEC		Starts	Ends
2016,2,A,EECS,461,001	Completed	Dec 1, 2016	Dec 14, 2016
EECS 461 014 LAB		**Starts**	**Ends**
2016,2,A,EECS,461,014	Completed	Dec 1, 2016	Dec 14, 2016

Figure 1: Miriam's Course Evaluation Receipt.



EECS 461 001 LEC		Starts	Ends
2016,2,A,EECS,461,001	Completed	Dec 1, 2016	Dec 14, 2016
EECS 461 014 LAB		**Starts**	**Ends**
2016,2,A,EECS,461,014	Completed	Dec 1, 2016	Dec 14, 2016

Figure 2: Mason's Course Evaluation Receipt.