EECS 461 Final Project Report

1. Introduction

Automatic driving plays an important role in modern automotive systems. In this project, we implement and simulate an automatic driving system using **Simulink**, incorporating a dynamic vehicle model and three distinct control modes (as shown in Figure 1). In **manual mode**, the vehicle is controlled using a haptic steering wheel (with a rotary encoder) for steering and a potentiometer for throttle input.

When the **adaptive cruise control (ACC) mode** is enabled, a *pick lead logic* first determines the closest car ahead by analyzing the positions of other cars via CAN communication. The ACC operates in two sub-modes: **velocity control**, where the vehicle drives at a user-defined speed, and **position control**, which activates when a leading vehicle is within a critical distance—causing the car to maintain a safe following distance. The controller automatically switches between position control mode and velocity control mode. If the car gets too close to a leading car, it will switch to a position control mode.

The third mode is **automatic steering control**, enabled via a separate dip switch input signal. In this mode, a PID controller uses road geometry (via (s, n) coordinates) to minimize the lateral deviation (n) and steer the vehicle toward the center of the road. This allows the car to autonomously follow curved and straight sections of the path. The system integrates throttle, steering torque, and visual output through serial communication to create a cohesive real-time simulation.

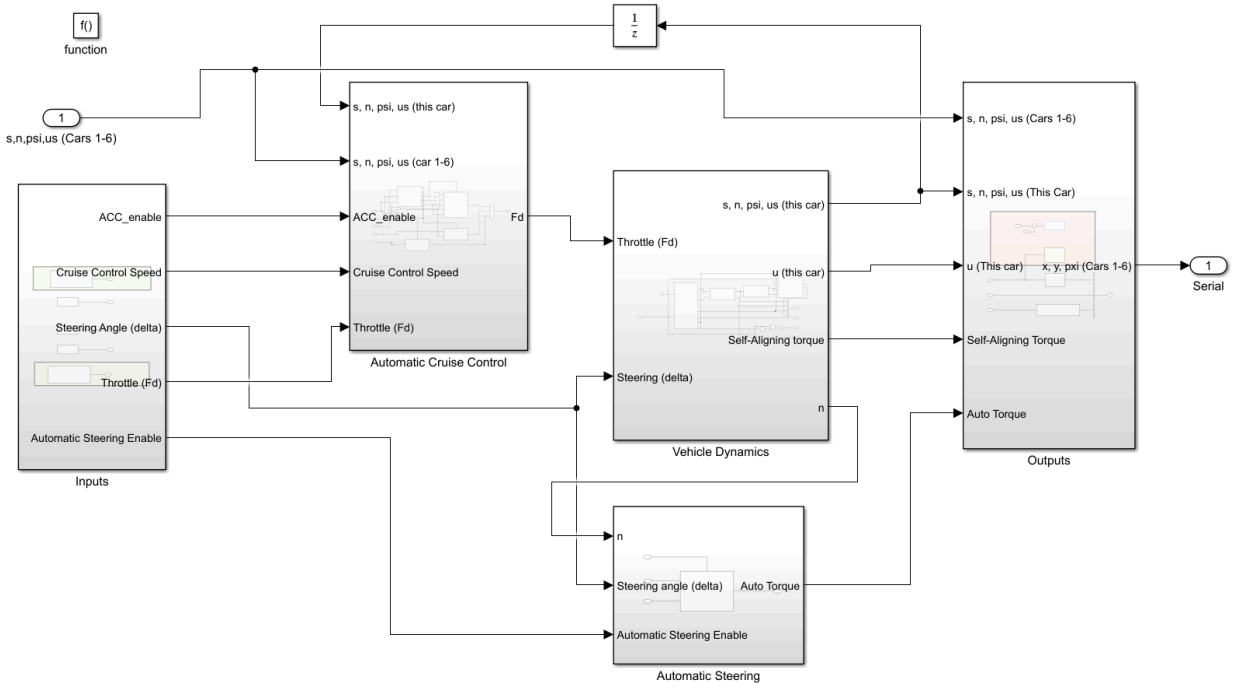


Figure 1: High level block diagram the system

2. Pick Lead Logic

We used an S-function builder block in SIMULINK to implement the pick lead logic. The block takes the positions of all 7 cars (s) and their velocity (us) as input, and the statistics of my car are stored in the first place of these 0-based arrays. The system finds the closest car that is ahead of my car and outputs two variables “lead_car_distance” and “lead_car_velocity”.

At the beginning, the value of "lead_distance" is set as a large number (1e9 in our case) and the initial values of variables "lead_car_distance" and "lead_car_velocity" are set as my position and velocity (s[0] and us[0] respectively) so the variables will stay the same if my car is the lead. It iterates through other 6 cars (number 1 to 6) and checks (1) if it is ahead of my car (s[i] > s[0]) and (2) if it is closer than another car ahead (s[i] < lead_distance). Then in the following iterations if both conditions are met, we update the position of the current closest car ahead as well as the variable "lead_car_velocity". After the program iterates through all six other cars, we will send two outputs "lead_car_distance" and "lead_car_velocity" to the "Position Control" block.

3. Automatic Steering Controllers

The whole block takes two inputs (current n and Steering angle delta) and outputs Auto Torque (as shown in Figure 2). It is composed of two PD controllers, one is Vehicle PD controller (outer loop) and the other is Haptic wheel PD controller (inner loop). For the outer loop the input is (n_desired+n) and the output is Target steering angle. On the other hand, the Haptic wheel controller takes (delta+target steering angle) as input and outputs auto torque. Both controllers follow a similar structure: Discrete PD controller so let us elaborate the first one as an instance.

We set n_desired as 0 as the car needs to follow the center line. For the differentiation term feedback, we utilize the "Unit delay" block and "1/dT", "S_Kd1"Gain blocks **to compensate for future error based on the rate of change of input**. As for the Proportional term, we use the "S_Kp1"Gain block to provide proportional feedback to the system.

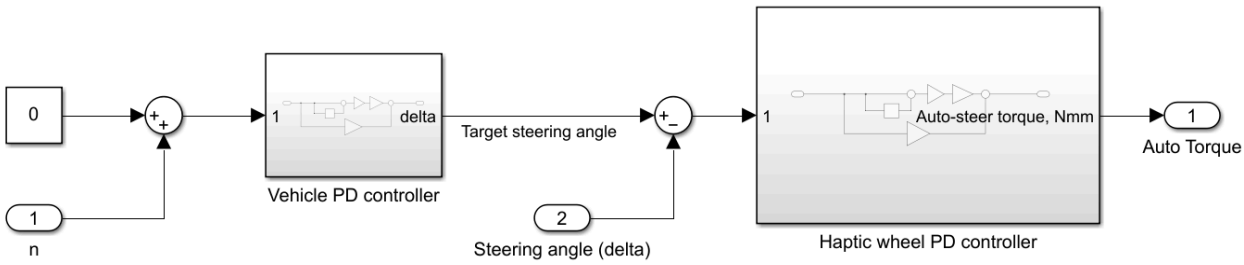


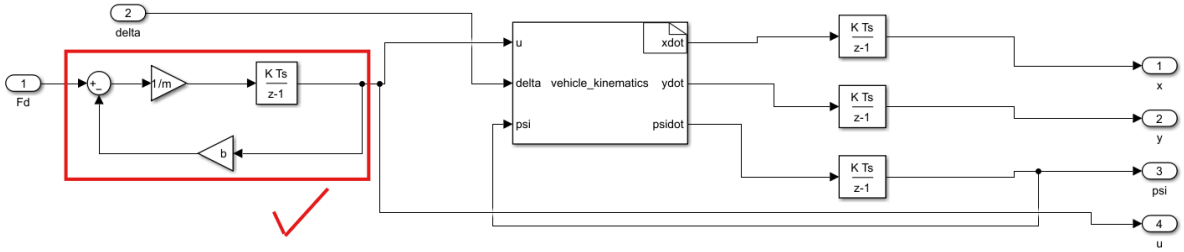
Figure 2: PD controller for auto-steering mode

To find the appropriate parameters for the PD controller, we first find the closed loop transfer function of the controller, which is a second order system. Then, we find the characteristic equation of the transfer function and set its natural frequency ω_n and damping ratio ζ to match our needs. The the proportional gain K_P and differential gain K_D are computed based on equating the coefficients of the characteristic equation with that of the standard equation.

The natural frequency controls how fast the system responds and the damping ratio determines the extent of overshoot. We find tunes these two parameters to make our controller responsive and robust and finally, we choose $\omega_n = 10,\ \zeta = 0.7$ for the outer loop (for the vehicle simulation) and $\omega_n = 30,\ \zeta = 0.7$ for inner loop (providing torque on the haptic wheel). The gains we selected are $K_P = 0.0358,\ K_D = 137.43$ for the outer loop and $K_P = 0.2560,\ K_D = 573.45$ for the inner loop.

4. Encountered Problems

We encountered a problem when it comes to the Vehicle Dynamics block. At first we tried to compute variable u using S-Function by taking throttle Fd as input. However, due to the inappropriate initialization in our "Vehicle_dynamics_Update_wrapper" function it will return an error showing that "wrong . operator in S-Function". In the end we figured out the mismatch between discrete states and continuous states here and utilized several gain blocks and discrete integrator instead.




```
void vehicle_dynamics_Update_wrapper(const real32_T *Fd,
                                     real32_T *u,
                                     real_T *xD)
{
/* Update_BEGIN */
//xD[1] = -100*xD[0]/2000+Fd[0]/2000;
  xD[0] = xD[0] + 0.01 * (-100 * xD[0] / 2000 + Fd[0] / 2000);
/* Update_END */
}
```

5. Work-load Distribution

Equal contribution. We two accomplished all the functions together during the lab time.

6. Course Evaluation Receipt

Your responses have been submitted successfully!

Guanyu Xu, thank you for completing the evaluation for EECS 461-018: Embedded Control.

You will receive an **email confirmation** that you completed this evaluation. You can also print or save this page as a PDF for confirmation that you submitted your evaluation.

Your responses have been submitted successfully!

Guanyu Xu, thank you for completing the evaluation for EECS 461-001: Embedded Control.

You will receive an **email confirmation** that you completed this evaluation. You can also print or save this page as a PDF for confirmation that you submitted your evaluation.

Office of the Registrar - Evaluations <ro.evaluations.comm@umich.edu>
to me

Dear Haobo,

Your feedback for EECS 461-018: Embedded Control was submitted.

Thank you,
Office of the Registrar Evaluations team

Office of the Registrar - Evaluations <ro.evaluations.comm@umich.edu>
to me

Dear Haobo,

Your feedback for EECS 461-001: Embedded Control was submitted.