EECS 461 Final Project: Adaptive Cruise Control*

1 Overview

Many automobiles manufactured today include a cruise control feature that commands the car to travel at a desired speed set by the driver. In the future, automobiles may adopt a more dynamic form of cruise control that adapts to the traffic conditions. Adaptive Cruise Control (ACC) behaves like the typical cruise control when the road in front of the vehicle is free of traffic, but if a car comes within a certain distance ahead of the vehicle, the controller will slow down and maintain a safe gap. In maintaining the gap, the ACC will match the speed of the vehicle in front of it, but if the vehicle in front increases its speed past the ACC speed set point, it will resume speed control.

2 Your Task

You will create a Simulink model that models a simple vehicle and implements an ACC system. The haptic wheel will serve as the steering wheel to your simulated vehicle. The position of the wheel will determine the angle of the front wheels of your car and the steering forces will be reflected back to the wheel. The ACC system will be activated/deactivated by toggling one of the GPIO bits with the dip switch. When the ACC system is inactive, the vehicle speed will be manually controlled with the potentiometer connected to the QADC. When the ACC system is enabled, the potentiometer input is ignored and the speed set point in m/s will be given by the binary value of the lower eight GPIO bits divided by four (i.e. a max speed of $255/4$ m/s).

If traffic ahead of the vehicle comes within some critical distance, the ACC system will switch to a position control mode that maintains the gap to the vehicle in front. If the *lead vehicle* (the vehicle immediately ahead) increases its speed beyond the ACC speed set point, the ACC system resumes speed control. The road traffic will consist of up to five vehicles simulated by the other lab stations. Each lab station must broadcast its vehicle's position and speed on the CAN bus so that each vehicle is aware of all the other vehicles. The IDs and the format of the CAN messages are described in Section 7.

Some of the blocks needed to create the vehicle model and the ACC system have been provided for you (in the Simulink library browser), some of the models are described here and you must create them in Simulink and the remainder are left for you to design and implement. Blocks that provide access to the road geometry (Section 5), blocks that perform coordinate transforms (Section 6), and blocks for the position control and the speed control (Section 9) have been provided for you.

To model the vehicle motion, you will implement the differential equations describing the motion of the vehicle model as given in Section 4. Section 9 describes some simple logic that you can use to switch the ACC system between speed and position control modes. To implement this system, it will be necessary to determine which of the other five vehicles is immediately ahead. You will need to develop the logic required to determine the lead vehicle. You must also write code to interface with the hardware (GPIO, PWM, QADC, FQD, and CAN).

*Revised March 26, 2009.

In addition to implementing the vehicle model and the ACC system, we ask you to also implement an automatic steering controller using a PID implementation. When enabled, the controller should steer the car down the full length of the road by actuating the haptic wheel. Using the (S,N) road coordinates (see section 6) the error signal for the controller is fully contained in the value N. We can setup a controller to eliminate all or part of N; if we eliminate all the error, i.e. force N to zero, then our controller will guide our car to the center of the road. Alternatively, if we eliminate a fraction of the error, then the controller will guide our car to a position offset from the road center. Ideally, we seek a controller that operates with no overshoot, no steady state error, and remains stable for all throttle inputs over the full road length. As expected, this is not possible in practice. For our purposes, we seek a controller that operates stably over the full road length for as large a throttle input as possible while maintaining minimal overshoot and steady state error.

3 Files to Download

You will need to download the following files to build and run your model.

- *path_data.c*: initializes a data structure with the road geometry
- *path_data.h*: defines the roadway data structure
- *DrivingSim.zip*: contains the Windows graphics program
- *parameters.m*: MATLAB script to set model parameters
- *controllers.mdl*: contains a position and a speed controller

4 Vehicle Model

The vehicle model can be decomposed into a longitudinal and a lateral model. The longitudinal model applies Newton's second law to determine the velocity of the vehicle as the wheels apply driving forces and friction forces act to slow the vehicle. For simplicity, the vehicle will be assumed to have mass m, viscous damping coefficient b, but zero rotational inertia. The mass will be lumped at the front axle. The speed of the front wheels u will then be given by $\sum F = m\dot{u}$, where $\sum F = F_d - bu$, where F_d is the driving force. The front wheel speed u is then an input to the lateral vehicle model.

Figure 1 shows a top view of a vehicle whose configuration in the X-Y plane may be specified using the coordinates (x, y) of the vehicle center-point C and the angle ψ in radians between the vehicle centerline and the X-axis. The vehicle has two rear wheels on a common axle fixed perpendicular to the centerline and two front wheels on a common axle that may be oriented through a steering linkage. The common heading angle of the front wheels relative to the centerline of the vehicle is δ. **Note all angles in this project are in radians.**

Assuming no slip between the tires and the road, the kinematics of the vehicle are described by the bicycle model. Given δ and the speed of the front axle u as inputs, the motion of the vehicle is governed by the following set of differential equations.

$$\dot{x} = \left(-\frac{l_2}{l_1}\sin\delta\sin\psi + \cos\delta\cos\psi\right)u \tag{1}$$

$$\dot{y} = \left(\frac{l_2}{l_1}\sin\delta\cos\psi + \cos\delta\sin\psi\right)u \tag{2}$$

$$\dot{\psi} = \left(\frac{1}{l_1}\sin\delta\right)u \tag{3}$$

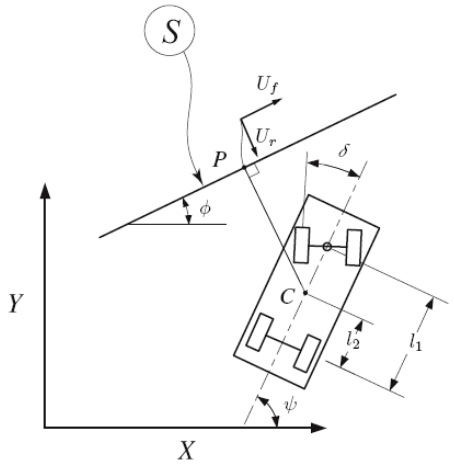

Figure 1: A bicycle model vehicle with heading ψ and front wheel heading $\psi + \delta$ is shown in the global coordinate frame X-Y and in relation to a straight section of roadway S.

Due to interaction between the front wheels and the road, a torque σ_{sa}, called the self-aligning torque, acts on the steering linkage and tends to drive the steering angle δ to zero:

$$\sigma_{sa} \quad = \quad -A_k u \delta. \tag{4}$$

Although the proportionality constant A_k is also a function of vehicle parameters for a full tire/vehicle model, we will simplify the model by using a constant A_k. The self-aligning torque should be reflected back to the haptic wheel so you can feel the steering forces. Remember that there is a gear ratio R that scales torques between the steering wheel and the front wheels.

For the ACC system, we will need to calculate the component of the vehicle velocity in the forward direction of road, which we define as u_s. To do this, we define unit vectors U_f and U_r that are tangent and normal to the road centerline S at P. The positive sense of U_f and U_r corresponds to a forward and a right direction, respectively. The velocity of the vehicle projected onto U_f gives u_s:

$$u_s = u \left\langle \cos(\psi + \delta), \sin(\psi + \delta) \right\rangle \cdot U_f \tag{5}$$

5 Road Geometry

The road consists of 16 straight segments and 15 left and right curving segments of constant curvature $\kappa = 0.025$ m^{-1} and varying length. Thirty solid orange cylinders of diameter 1.4 m are located at various points in the center of the road to act as obstacles. The separation between cylinders varies according to a uniform random distribution between 20 and 80 meters. Figure 2 shows a top view of the roadway and the obstacles. The roadway length is 1993 m.
The road geometry is compiled into a target ELF file by placing path_data.c and path_data.h in the directory next to the model. During the build process, the roadway coordinate transform blocks instruct Real Time Workshop to include these two files in the compilation.

Special Simulink s-function blocks provide access to the road geometry. Specifically, given an s-coordinate, which is the path length along the center of the road, there is a block that returns the (x, y) coordinates on the centerline of the road, a block that returns the curvature, and two other blocks that return unit vectors in the forward and the right directions, U_f and U_r. These blocks can be found in "U of M blockset for MPC5500/Vehicle Simulator."

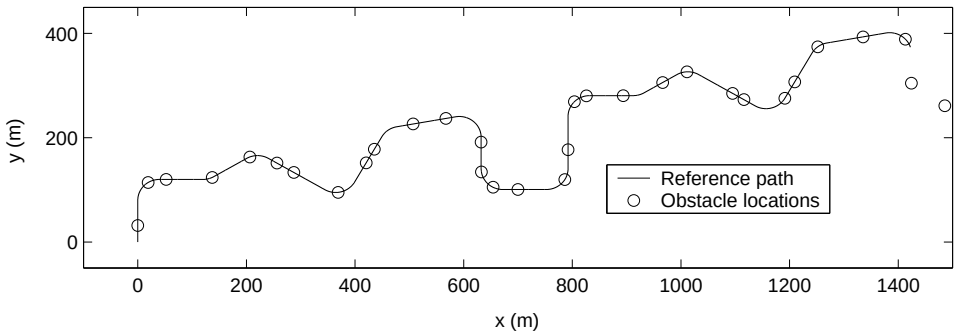

Figure 2: A top-down view of the driving course.

6 Coordinate Transforms

It will be necessary to describe the position and velocity of the vehicle in two different coordinate systems, and thus it is necessary to be able to transform between these systems. The coordinates (x, y) are with respect to the global coordinate system X-Y. Integration of the vehicle position is performed in global coordinates. It is more convenient to order the vehicles based on the distance they have travelled down the center of the road. Let P denote a point on the center of the road, and define (s, n) coordinates, where s is the distance travelled down the center of the road to P and n is the lateral displacement to the right of P. Then the s coordinate can be used to determine the distance between vehicles measured along the centerline.

The vehicle center C can be transformed from (s, n) to (x, y) coordinates easily with the road geometry blocks described in Section 5. Point P and U_r can be found from s using the blocks, and then C is $P + nU_r$. The other transformation, from (x, y) to (s, n), is challenging. Rather than using a closed form solution, a feedback-stabilized closest point algorithm is used to find s such that C is to the right or left of P without being ahead or behind. Then n is the distance in the U_r direction from C to P.

The two coordinate transformations have been provided as blocks to you. Be aware that the (x, y) to (s, n) transformation is not exact and relies on an initial estimate of s that must be close to the actual s coordinate. Figure 3 shows the feedback-stabilized algorithm described and points out the integrator with the initial estimate of s.

7 CAN Communication

In an actual ACC system, radar would be used to measure relative location and velocity of traffic in front of the vehicle. In this lab, we will replace radar information with communication between vehicle models by using the CAN network. Each computer on the CAN network will periodically transmit 4 values (32-bits each) onto the network. The ACC systems need the speed along the centerline u_s (see Section 4) and the s-coordinate of all the other vehicles. The graphics software needs x, y and ψ for each vehicle, where x and y can be found from s and n. To provide these values s, n, u_s, and ψ, each lab station must transmit 2 64-bit Messages (with two 32-bit values per message). Also, the format of these messages must be standard so that everyone can understand each others messages, so the following format *must* be used.

- Message 1, Bytes 1-4: s as a float
- Message 1, Bytes 5-8: n as a float
- Message 2, Bytes 1-4: u_s as a float
- Message 2, Bytes 5-8: ψ as a float in radians

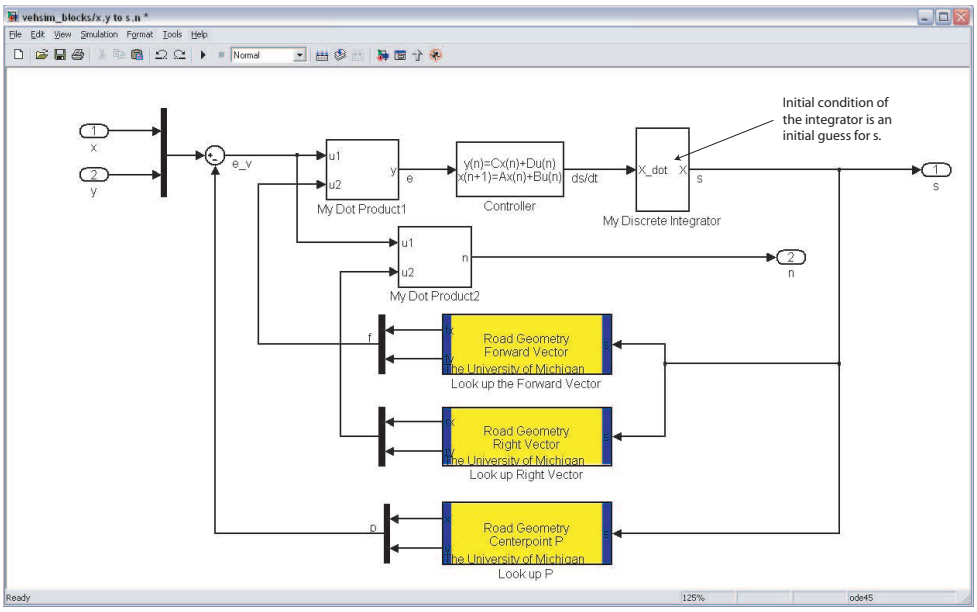

Figure 3: The diagram shown converts (x, y) coordinates to (s, n) coordinates.

To form the CAN messages, use the "Single-to-Bytes" block to create two 4-byte arrays and then mux them into the port data of the CAN transmit block. When receiving CAN messages, they can be unpacked by an inverse operation. The data can be demuxed into an array of 8-bytes and then the upper and lower bytes are muxed into 4-byte arrays and passed to the "Bytes-to-Single" block to reconstruct the floating-point value.

8 Manual Control

If the ACC system is disabled, the vehicle responds to "pedal" inputs from the potentiometer. The potentiometer determines the driving force F_d. You will need to determine an appropriate scaling and offset for the QADC output (which is in the range $[0, 2^{14} - 4]$) to the range of forces you want. Consider allowing negative forces since you do not have a brake.

9 Adaptive Cruise Control Design

There are three modes of operation for the ACC system. If the system is deactivated, then the vehicle is under manual control, with the potentiometer functioning as a "gas" pedal. When ACC is turned on, there are two modes of operation: there is a "speed control" mode that maintains the vehicle at the desired velocity unless another vehicle is too close, in which case a "position control" mode is implemented that keeps the vehicle a fixed distance from the car immediately in front.

The ACC system is actually two controllers with some logic that selects one of the two controllers or selects manual control if the system is deactivated. The switching logic needs to know the s coordinate of the lead vehicle, so you must create a subsystem that takes the s coordinates from all the other vehicles, compares them against your own s-coordinate, and selects the lead vehicle if one exists.

If there is no lead vehicle, then the switching logic activates the speed control block. Let subscript i denote the lead vehicle. Then given a lead vehicle s-coordinate s_i and your own s-coordinate s and the speeds u_{si} and u_s, a simple switching logic can be defined by

$$\text{Mode} = \begin{cases} \text{Position Control} & \text{for } u_{si} \leq u_s \text{ AND } s_i - s \leq H \\ \text{Velocity Control} & \text{for } u_{si} > u_s \text{ OR } s_i - s > H \end{cases} \tag{6}$$

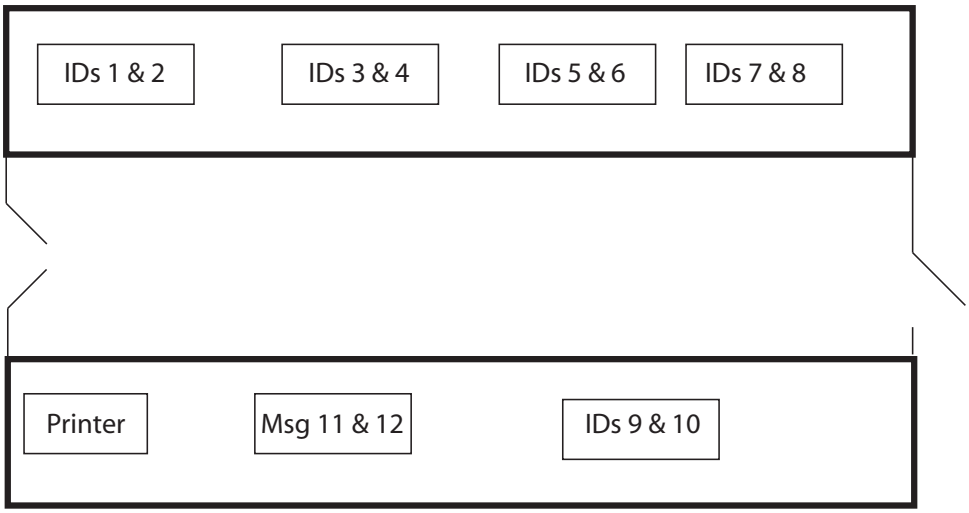

Figure 4: Each lab station should transmit CAN messages with the IDs as given above.

where H is the safe gap distance. This switching logic may tend to introduce oscillations between modes, so it may be useful to use additional logic to prevent switching out of position control mode until the value of $(u_{si} - u_s)$ exceeds a small positive amount (consider why).

The ACC enables either position or speed control if activated, or manual control otherwise. In Simulink this is accomplished by using either a Stateflow diagram or an "enabled subsystem," which only runs if the enable signal is 1. The three possible subsystems all produce a force F_d to drive the vehicle's front wheel. Use a Simulink "merge" block to select the output of the one enabled subsystem.

Lastly, we want to consider the situation in which one of the car's system sensors fails. For a real car using an ACC controller this may be the infrared camera used to determine the following distance of the car in front of it, or the other sensors used to monitor wheel speed etc. For this task, we will introduce one sensor who's probability of failure is 1 percent, and second, adapt our ACC controller to handle the situation when the sensor fails. Specifically, when the sensor fails, we want our ACC controller to only select manual mode. Only when the sensor has been repaired, by toggling a dipswitch, should the ACC then be able to again select from all three operating modes.

10 Graphics Software

We are providing you with a program that displays a roadway from the view point of your simulated vehicle and will display the five other vehicles on the road. A screen shot is shown in Figure 5. The program can be run on the Windows lab stations and it communicates with your vehicle model through the serial port. It expects an array of eighteen float (32-bit floating point) numbers preceded by the character "s" and terminated with the character "e". The array of floating-point values, in order, are: x, y, ψ, x_1, y_1, ψ_1, x_2, y_2, ψ_2, x_3, y_3, ψ_3, x_4, y_4, ψ_4, x_5, y_5, and ψ_5.

Once you have connected the graphics program to the serial port, it will display the roadway from the location (x,y) looking in the direction ψ with the most recent values that it reads through the serial port. The program is set to read COM1 at a speed of 115200 baud. Initial conditions, $x_0 = 0$, $y_0 = 0$, and $\psi_0 = 0$ will start the vehicle simulation in the middle of the road and facing forward.

Because the serial port transfers data slowly in comparison to the rest of the model, sending of the data should be performed in a separate task. You should use the Resource Allocation blocks to transfer the 18 values from the main model task to the a task dedicated to transmitting those values through the serial port.

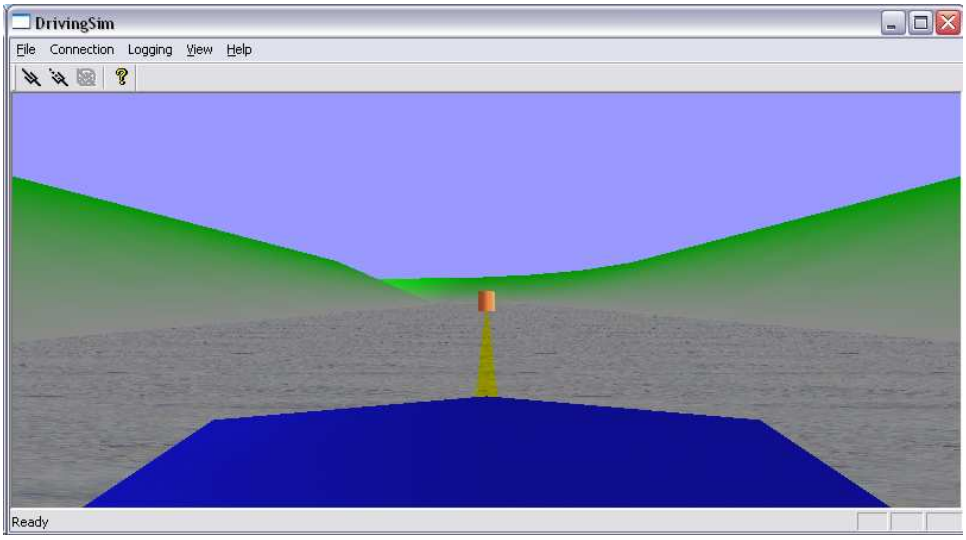

Figure 5: Screen shot of the graphics software. The first toolbar button connects the program to COM1. The second button forms a simulation connection and you can steer with the mouse while pressing the left mouse button. The third button disconnects from the active connection.

You can log all the data read from the serial port to a log file by selecting the logging menu option. A text file will be created, in which each line of the file contains a time stamp from the PC and the eighteen values read, in order. You can read this log file into Matlab for analysis and plotting.

11 Prelab

1. Before coming to lab, make a block diagram showing the inputs and outputs of both the ACC and car model. Be sure to include, as inputs, any information these systems need to make decisions. Show how the blocks connect.

2. Before coming to lab, make a state transition diagram of the ACC controller mode. Recall that the three modes are Position control, Velocity control, and Manual control.