University of Pennsylvania
School of Engineering and Applied Science
Department of Electrical and Systems Engineering

AUTOPLUG -ADAPTIVE CRUISE CONTROL

Group 8

Kavita Kamesh
kkavi@seas.upenn.edu

Anand Madhusoodanan
anandmad@seas.upenn.edu

Aswath Gajendran
aswathg@seas.upenn.edu

INTRODUCTION

AutoPlug is an Automotive ECU test-bed to diagnose, test, update and verify controls software. It consists of multiple ECUs interconnected by a CAN bus, a driving simulator which behaves as the plant model and a dashboard/control monitor in MATLAB. AutoPlug was mainly developed to improve warranty and recall management of vehicles.

ADAPTIVE CRUISE CONTROL (ACC)

Adaptive Cruise Control system is an enhancement over the conventional cruise control system. Adaptive Cruise control uses forward sensing radar which is mounted behind the grill at the front of the vehicle to detect the speed and distance of the vehicle ahead of it and can automatically adjust the car's speed via throttle and braking control to maintain a safe following distance. If the lead vehicle slows down, or if another object is detected, the system sends a signal to the engine or braking system to decelerate. Then, when the road is clear, the system will re-accelerate the vehicle back to the set speed.

SYSTEM ARCHITECTURE

Figure 1 : Architecture diagram

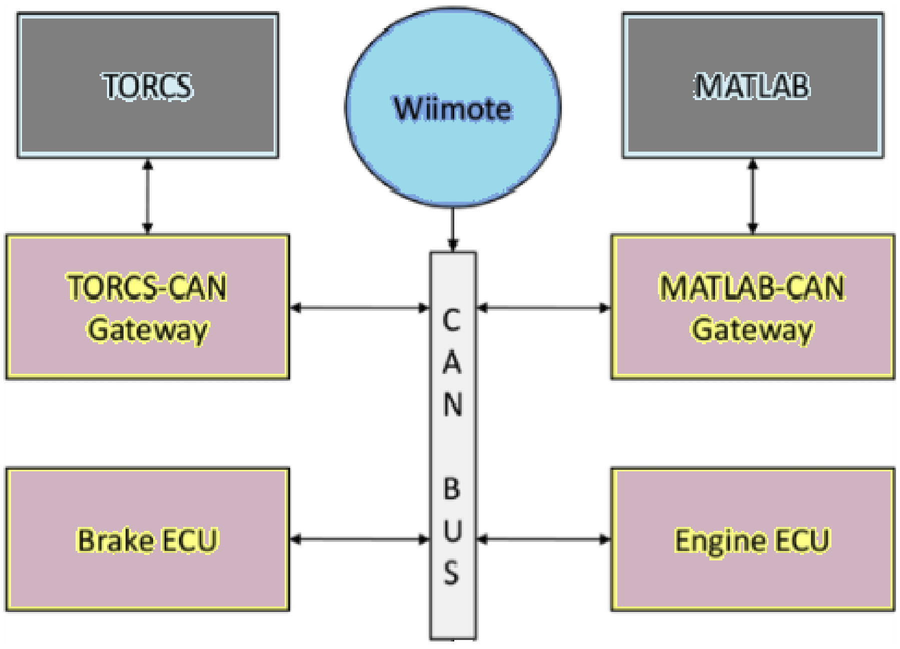


Figure 1 shows the Architectural diagram of AutoPlug. There are 4 Electronic Control Units each of which perform special functions which will be described later. These ECUs are interconnected by a CAN bus (500 Kbps).Instead of a real car we have used The Open-source Race Car Simulator (TORCS). TORCS is a driving simulator that provides physics-based feedback to the ECU network. There is a MATLAB dashboard at the front end to set and change critical parameters and analyse the output. The driver inputs like acceleration, brake and UI buttons for Stability control, ACC etc. are provided through a WiiMote Steering wheel. WiiMote also makes the system more interactive. CAN messages formed from the WiiMote are transmitted on to the CAN bus, from which, each of the ECU's read these messages and implement appropriate control algorithms.

Figure 2 : Test-Bed Setup

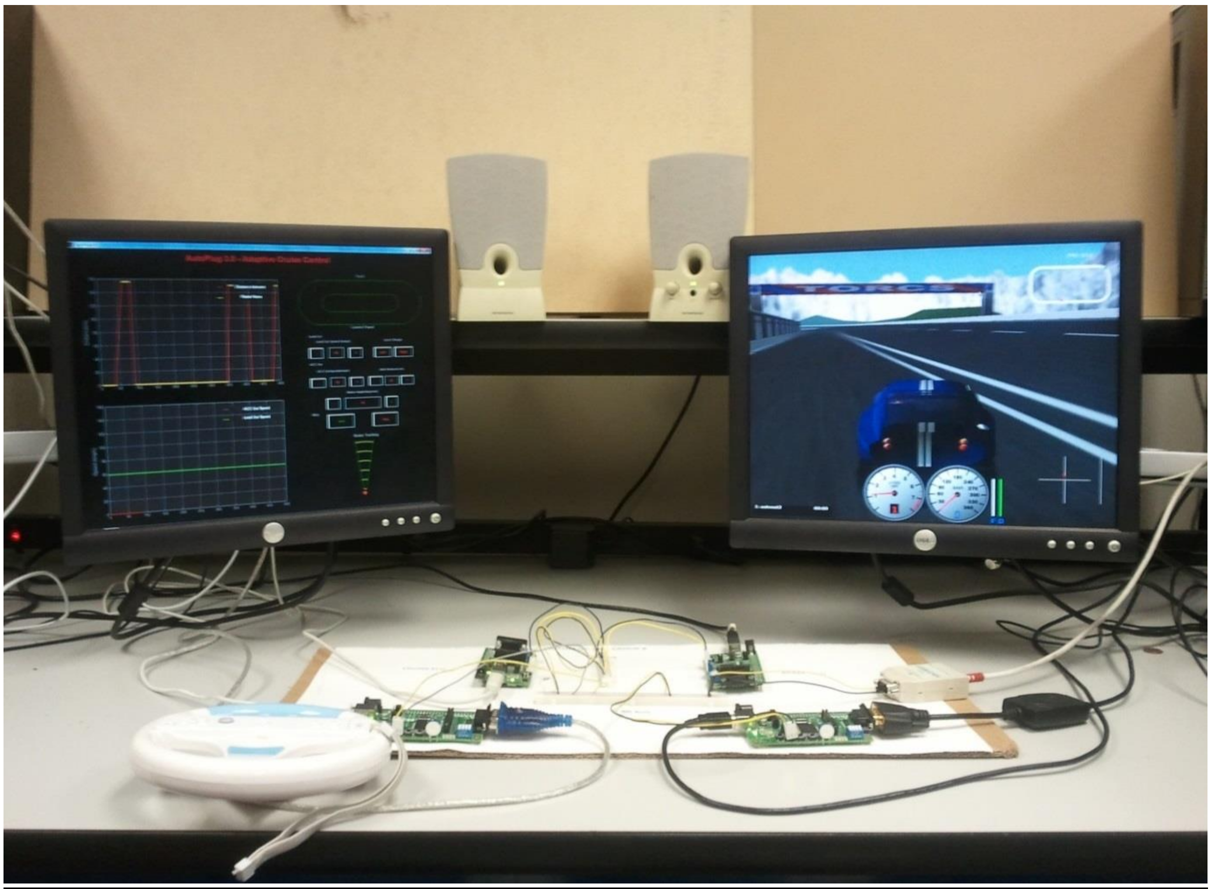


Figure 2 shows the complete test-bed setup.

Figure 3 : ECU Network

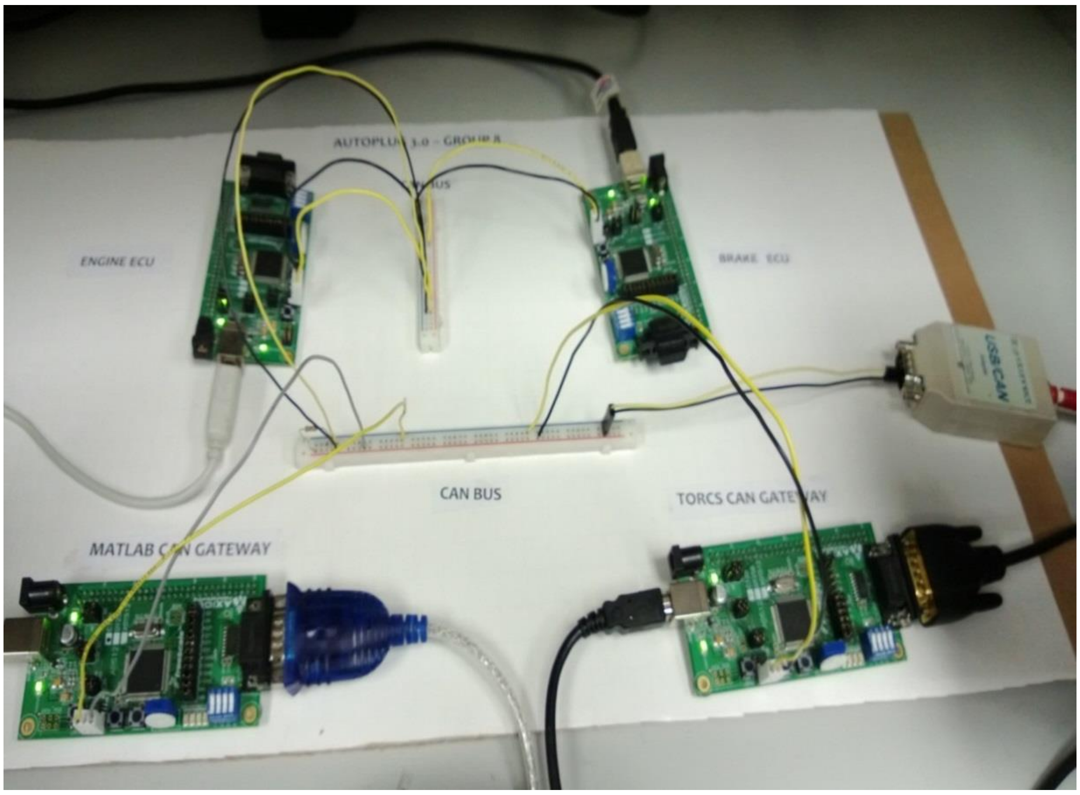


Figure 3 shows the ECU network which consists of 4 ECUs (Freescale HCS12 microcontrollers) each of which performs a specific function. These microcontrollers are networked together using the Industry standard CAN automotive protocol. The four ECUs perform the following functions.

1) **<u>EngineECU</u>**-Controls features like Throttle Control, Auto Transmission, Cruise Control and Adaptive Cruise Control

2) **<u>Brake ECU</u>**- Controls features like ABS, Traction Control, and Stability Control.

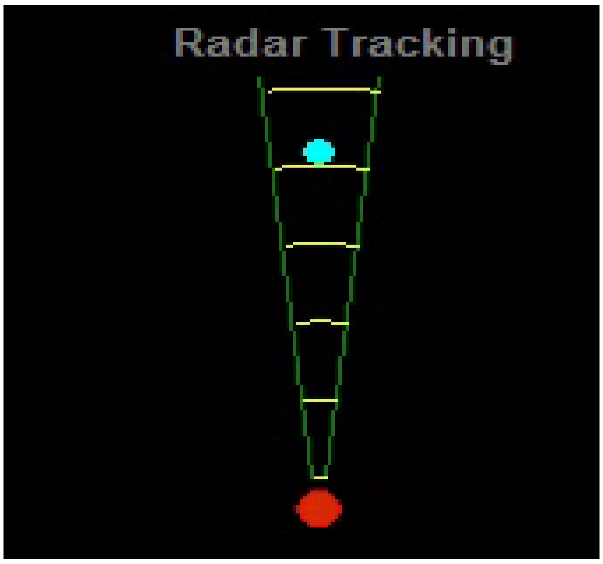
3) **<u>TORCS-CAN Gateway</u>** – Acts as the interface/gateway between the TORCS simulator and the CAN bus. It deals with Serial conversion and filtering of CAN messages required for the TORCS simulator and driver input.

4) **<u>Matlab - CAN Gateway</u>** - Acts as the interface/gateway between the CAN bus and the MATLAB dashboard. It deals with serial conversion and filtering of CAN messages required for graphing in a Matlab environment.

VIRTUAL SENSOR

We have implemented a Virtual sensor for the ACC car in order to sense the distance of the ACC car from the lead car. If the lead car is in the range (required distance and angle of view) of the ACC car, then the distance between cars is obtained in real time and is plotted on a graph on the MATLAB dashboard. If however, the lead car is not in the range of view of the ACC car, the distance between them is interpreted as infinity and one can observe this as radar noise (a yellow dotted line) on the real time plot for distance. Figure 4 shows how the ACC car tracks the lead car using the virtual sensor.

Figure 4: Virtual Sensor tracking



The blue dot indicates the lead car and the red dot indicates the radar sensor mounted on the ACC car. The blue dot indicates that the lead car is in the range of view of the ACC car. If the lead car is not in range of the ACC car (on curves or during lane changes), the blue dot disappears.

ACC SCENARIOS

1. LANE CHANGE

   We can change the lane of the lead car to left or right from the MATLAB dashboard. When both cars are in the same lane and the lead car is in the range of view of the ACC car, the ACC car tries to follow up with the speed of the lead car. When the lead car changes lane, it is no more in the range of view of the ACC car; so the ACC car gradually speeds up to its pre-defined set speed.

2. LEAD CAR SPEEDING UP

   If the lead car starts speeding up and the distance between the cars increases, the ACC car also gradually starts increasing its speed until it catches up with the new speed of the lead car and thus can maintain the safe distance from the lead car.

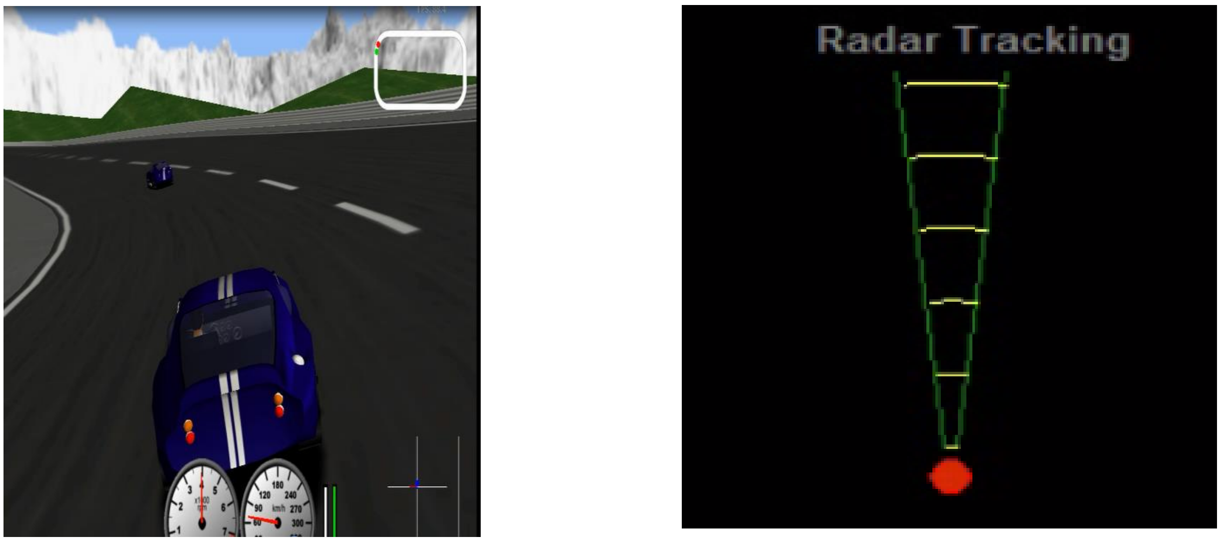
3. VIRTUAL SENSOR ON STRAIGHT LANES

   The Virtual sensor is able to track the lead car on both straight lanes and on curves. On a straight lane, the lead car appears in and out of the radar sensor's range of view as it changes lanes.

4. VIRTUAL SENSOR ON CURVES

   When the lead car is moving on a curve, the radar sensor cannot detect the distance between the 2 cars and the lead car is no more in the range of view of the ACC car. This can be observed on the GUI as shown in Figure 5.

Figure 5: Virtual Sensor on Curves

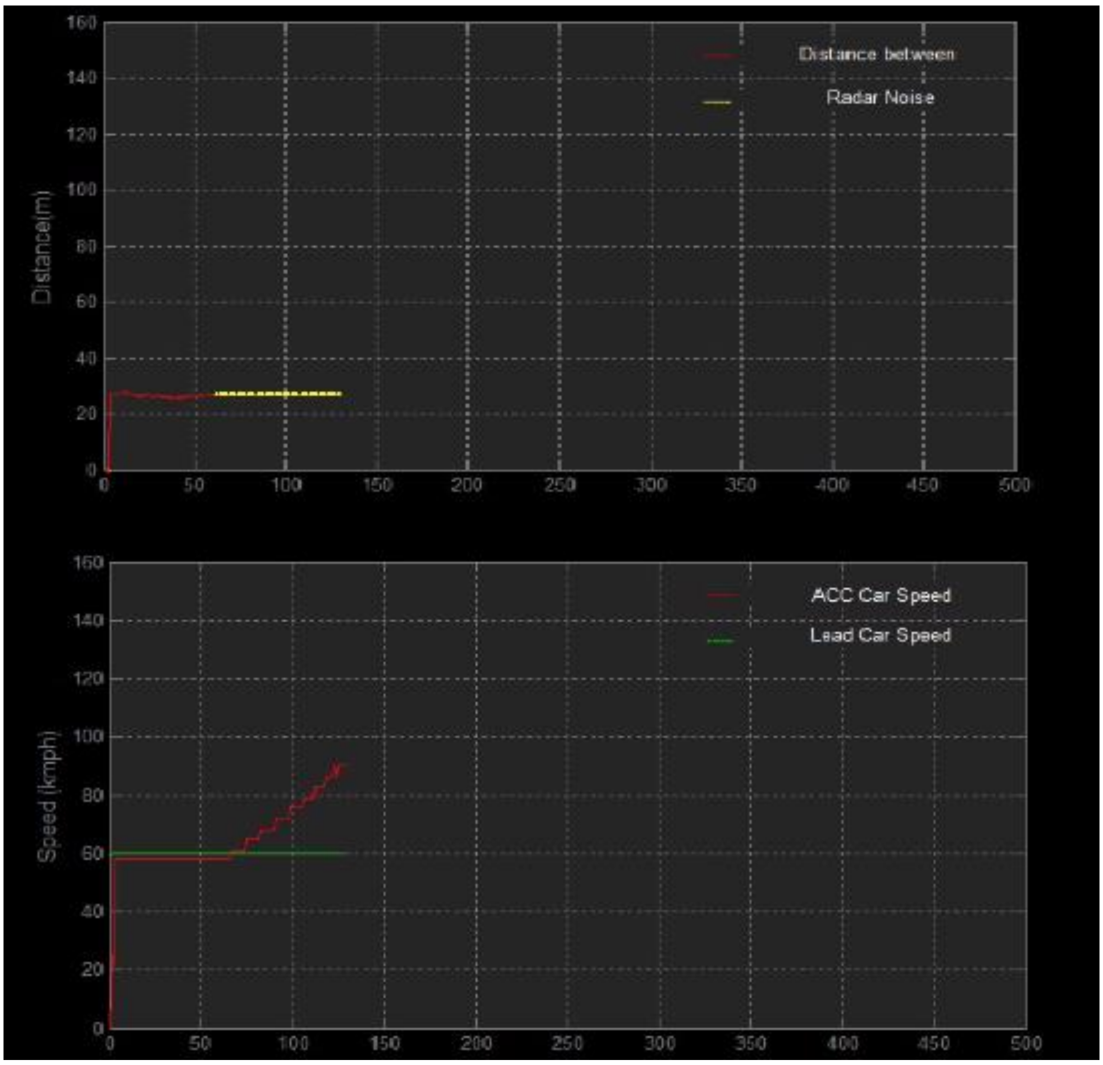


From Figure 5 we can observe that as the lead car turns on a curve, the blue dot which represents the lead car disappears from the radar tracking zone.

MATLAB DASHBOARD

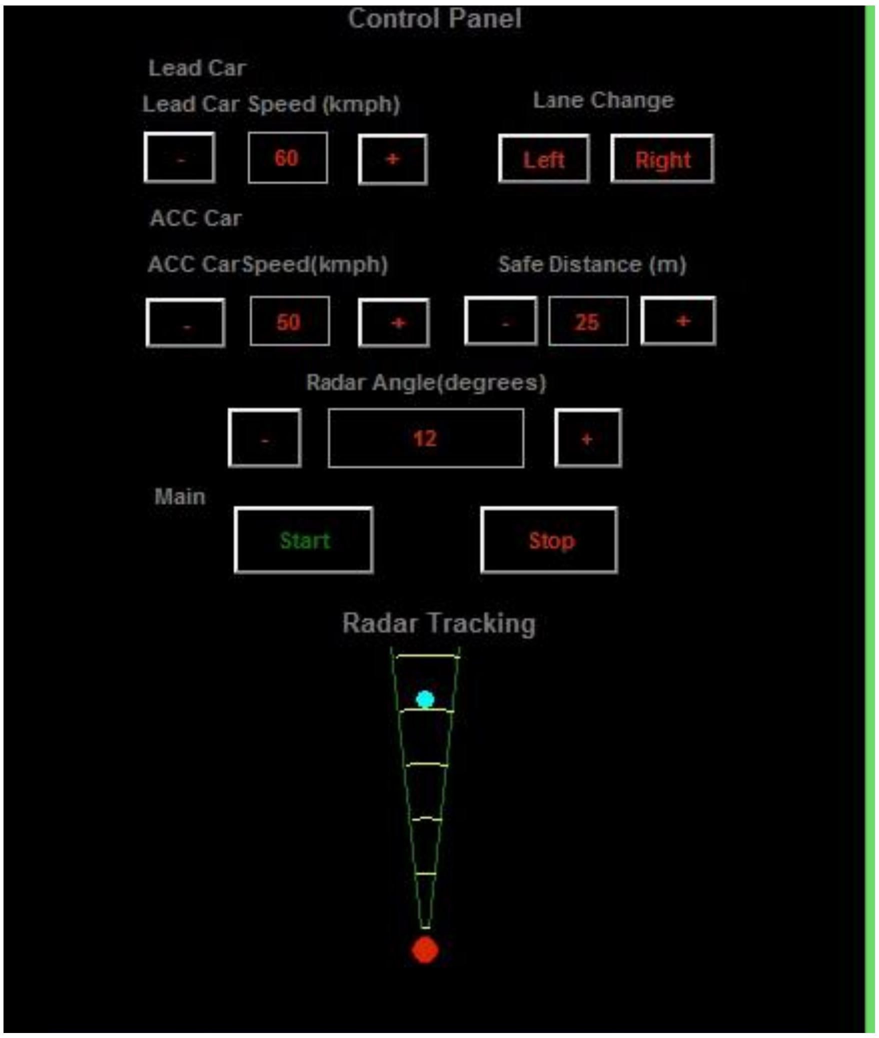
MATLAB is used in order to implement a dashboard that can be used to monitor in real time the speed of the lead car, speed of the ACC car and distance between the 2 cars as shown in Figure 6. From the dashboard, one can also control a number of parameters as shown in Figure 7.

Figure 6:Real time Plots



From Figure 6 we can observe that on the speed plot, red line indicates the ACC car speed, green line indicates the Lead car speed. On the distance plot, the red line indicates the distance between the 2 cars and yellow dotted line indicates the radar noise.

Figure 7: Dashboard Control Panel

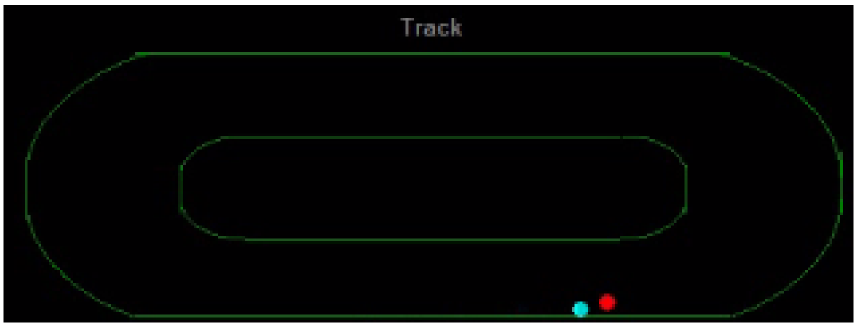


From the dashboard control panel as shown in Fig 7, one can control the following features.

a) Increase or decrease the radar sensor tracking angle to widen or narrow the field of view.
b) Increase or decrease the speed of the ACC car.
c) Increase or decrease the safe distance between the 2 cars.
d) Increase or decrease the speed of the lead car.
e) Change the lane of the lead car to either left or right.

One can also see the positions of the 2 cars on the track in real time on Matlab dashboard. This is as shown in Figure 8.

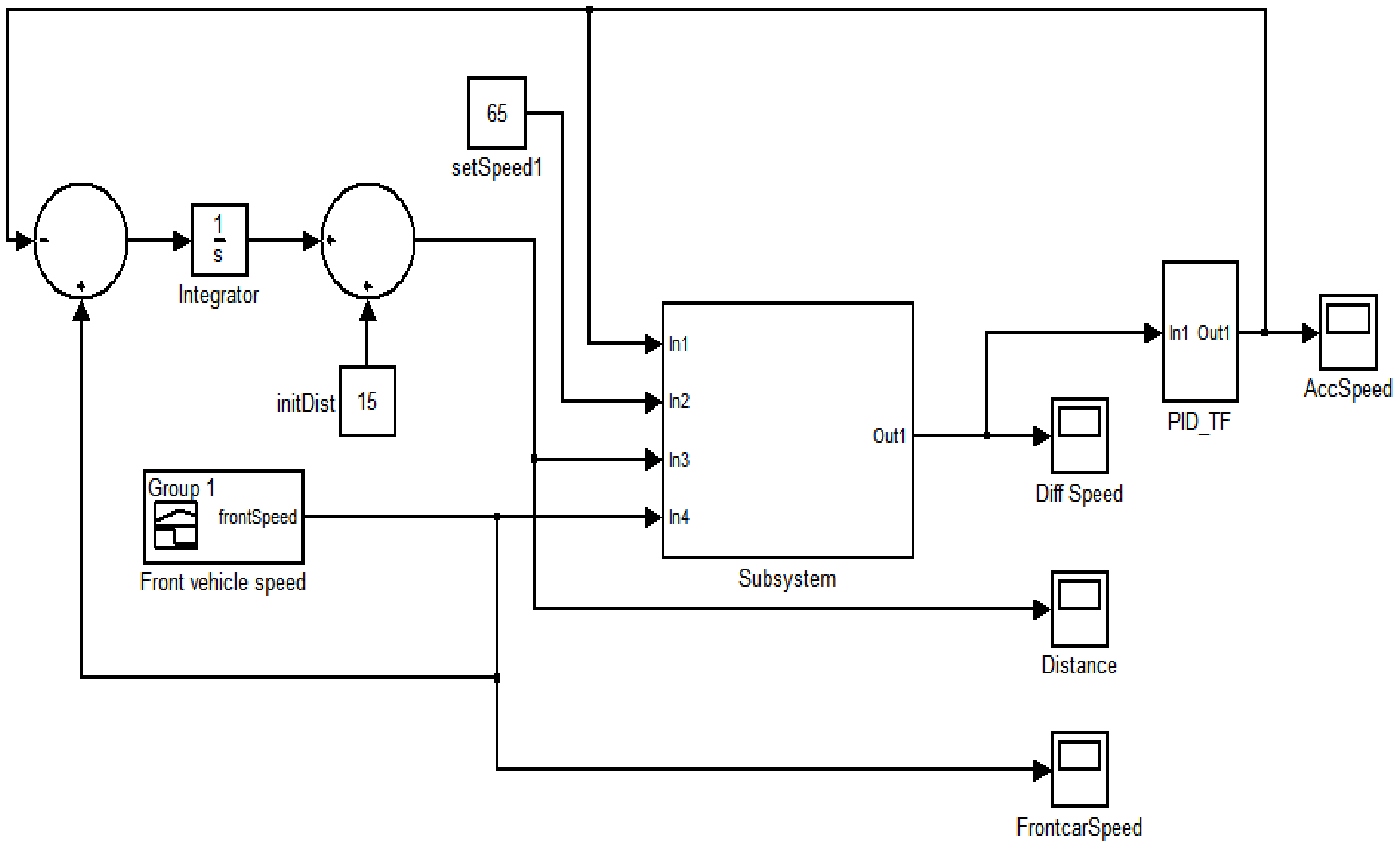
Figure 8: Positions of the 2 cars on the track



SIMULINK

We have designed a Simulink model for Adaptive Cruise Control is as shown in Figure9

Figure 9:Simulink Model for ACC



Here the blocks are the Subsystem (ACC subsystem) and PID_TF. We are plotting and observing on scopes - the distance between the 2 cars, speeds of the ACC and the lead car.

The 2 blocks ACC subsystem and PID_TF are described below

Figure 10 : ACC Subsystem

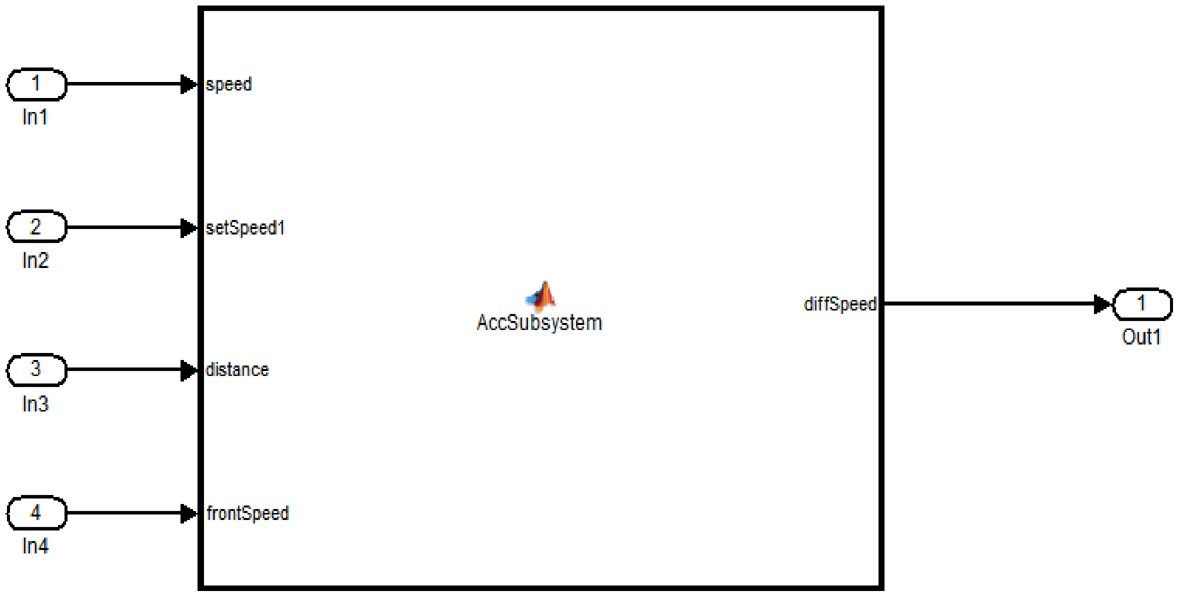


Figure 10 shows the ACC subsystem block. The inputs to the ACC subsystem are speed of the ACC car, speed at which the ACC button was pressed, distance between the 2 cars and the speed of the lead car which is taken from a signal builder so that the system can be tested for varying speeds of the lead car. The distance between 2 cars is computed as

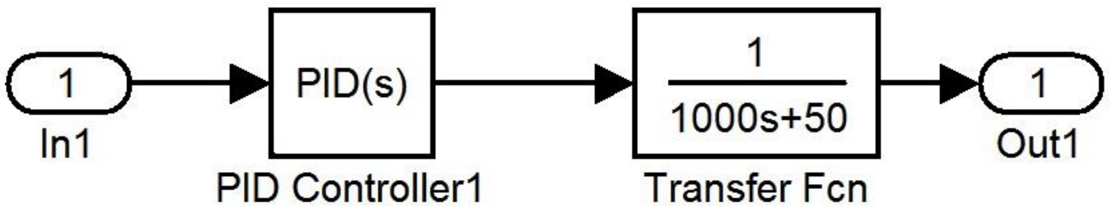
Distance r(t) = r(0) + $\int_0^t (v_f(t) - v(t))dt$

where r(0) is the initial distance between the cars = 15m (in this case)
$v_f(t)$ – velocity of the lead car
v(t) – velocity of the ACC car.

The output of the ACC subsystem is the difference between the desired speed of the ACC car and the measured speed.

This difference is fed to the PID controller block which is as shown in Figure 11

Figure 11: PID_TF block



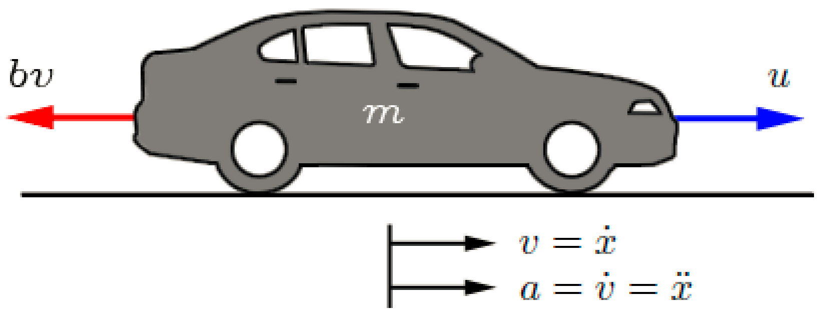
The values of Proportional and Integral gains are
Kp=2000 and
Ki=80 respectively

The transfer function is obtained as follows.

The factors that we have considered for Car dynamics are the weight of the car and the frictional force acting on it.

The car model is as shown in Figure 12

Figure 12:Car Model



Assumptions

m = 1000 kg (Mass of the car)

u = 500N (force generated between the road/tire interface)

b = 50 N.sec/m(Coefficient of friction)

Using Newton's 2nd law, the governing equation for this system becomes:

$$m\dot{v} = u - bv$$

Taking Laplace transform on both sides

ms.V(s) + b.V(s) = U(s)

- ⇨ V(s) [ms+b] = U(s)
- ⇨ Transfer function = V(s) / U(s) = 1/(ms+b) = 1/(1000s+50)

SIMULATION

After running simulations, we get the following graphs

Figure 13 : Speed of the ACC car

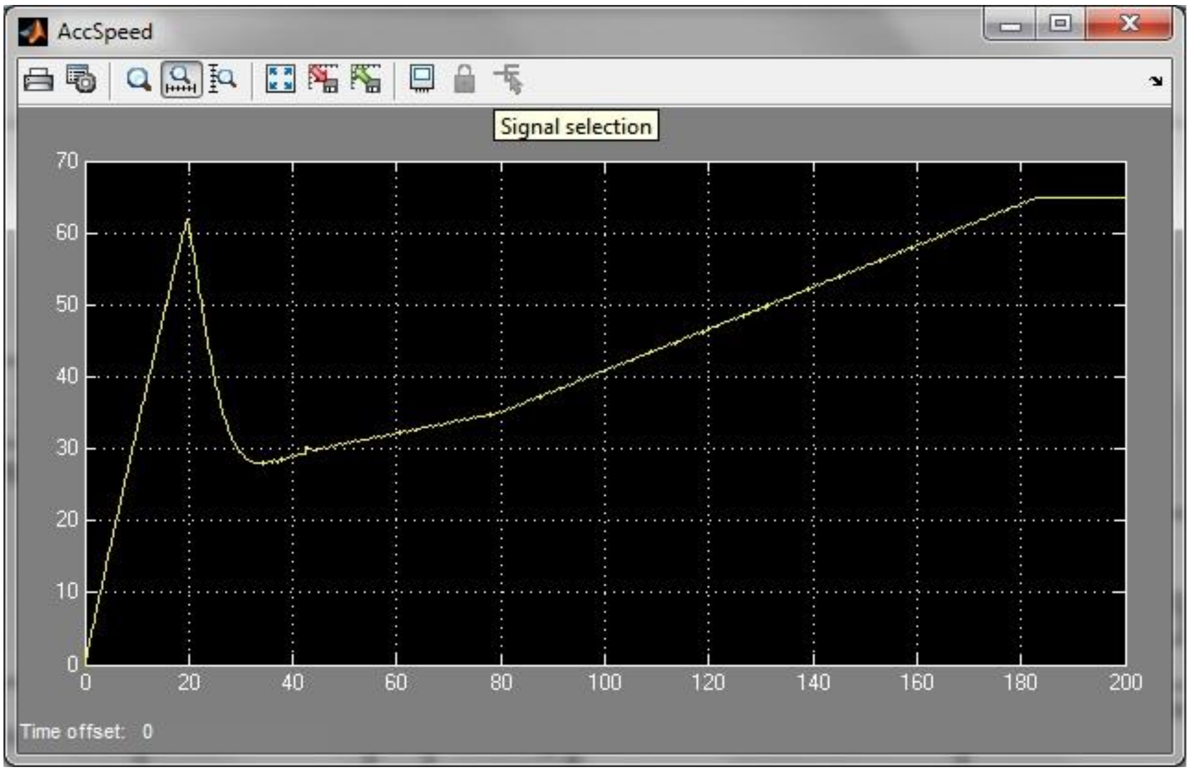


Figure 14: Speed of the lead car

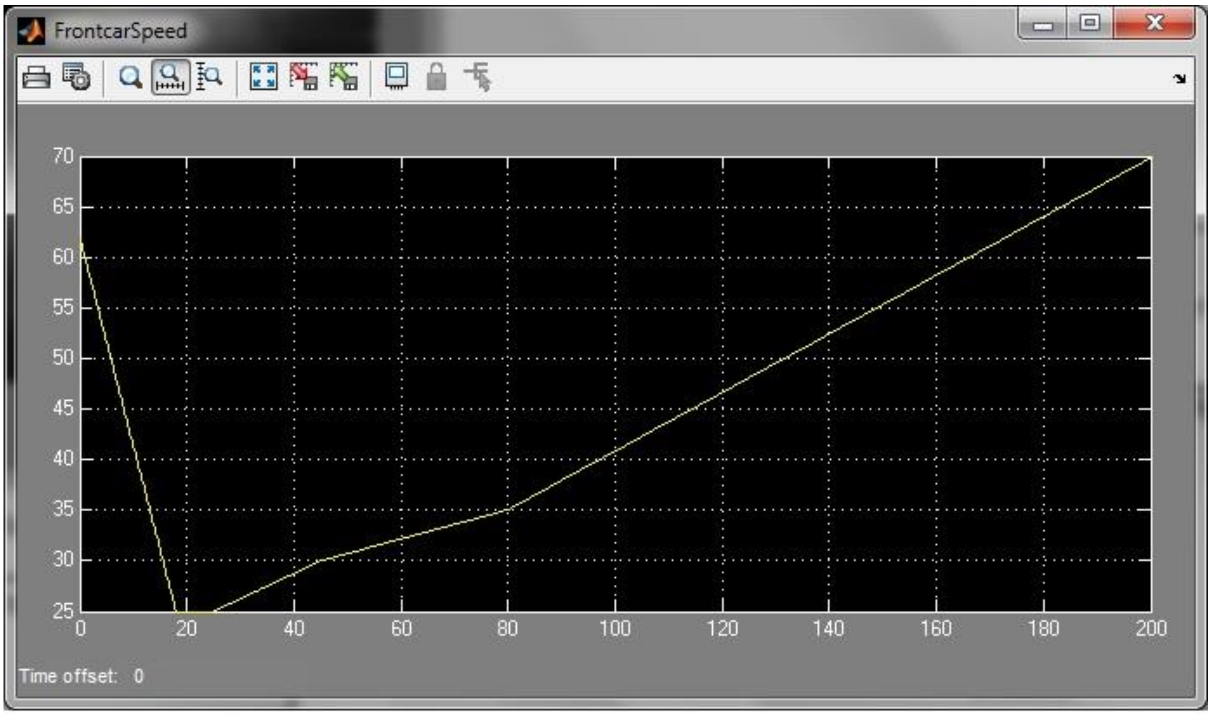
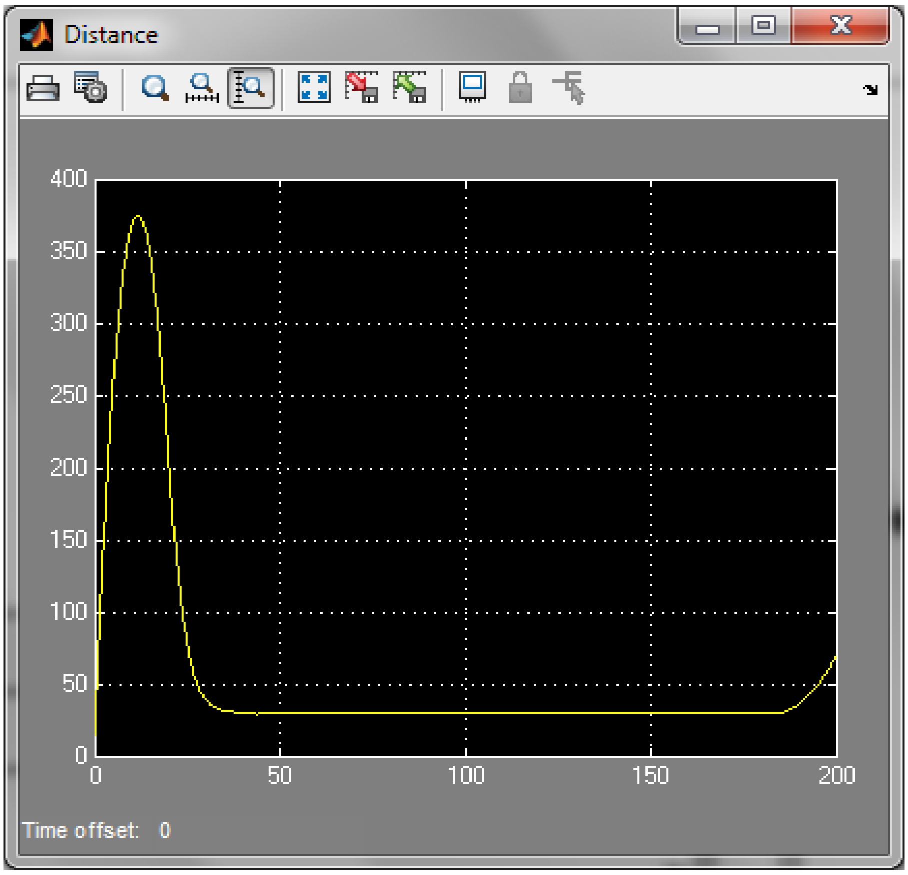


Figure 15: Distance between the 2 cars



From Figures 13, 14 and 15 we can observe the following.

This is one of the possible road scenarios.

Initially the lead car is at a high speed and the ACC car starts from a speed of zero. So the distance between the cars is initially large. As the lead car speeds down and the ACC car starts speeding up, the distance between cars reduces. In the zone around the safe distance, from approximately time t= 30 s to about t=182 s, the ACC car tries to catch up with the speed of the lead car. And in this time frame, the distance between cars is maintained at a safe distance of 30m. Once the lead car crosses a speed of 65 (speed at which ACC button was pressed), the ACC remains at a speed of 65 as seen from the graph and the distance starts increasing thereafter.

SOFTWARE (ALGORITHM FOR ACC)

The Algorithm that we have used to implement ACC is as follows.

1. We first compute the braking distance which is the distance at which the ACC car must start braking when the lead car is close. It is calculated as

   $\Delta x = ((\text{Front}_{\text{speed}})^2 - (\text{ACC\_Car}_{\text{speed}})^2) / 2a$ -(1)

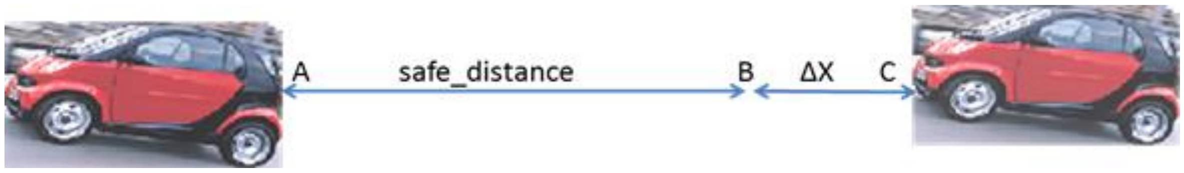
   This comes from the basic equation of Physics

   Displacement $s=(v^2 - u^2)/2a$ where

   v is the final velocity (In this case, final velocity is the speed of the front car $\text{Front}_{\text{speed}}$ which the ACC car intends to attain)
   u is the initial velocity that is the speed of the ACC car $\text{ACC\_Car}_{\text{speed}}$ and
   a is the acceleration or deceleration.

2. <u>There are 3 possible scenarios</u>

Figure 16: Road scenario



A) **(distance >safe_distance) && (distance < (Δx + safe_distance))**

This indicates that the lead car is at point A (from Figure 16) and the ACC car is in the zone between B and C. So, the speed that the ACC car must be set to is calculated as

setSpeed = $((\text{speed})^2 + (2*(-a)*S)^{1/2}$ - (2)

This equation is a variant of equation (1); but here we are finding the final velocity

speed –> current speed of the ACC car

Displacement S= ((Δx + safe_distance)-distance)

Desired distance =30m (safe_distance)

B) (distance <safe_distance)

This indicates that the lead car is at point A and the ACC car is in the zone between A and B. So, the speed that the ACC car must be set to is calculated as

setSpeed = $((\text{speed})^2 + (2*(-a)*S)^{1/2}$ - (3)

This equation is same as Equation 2 except for the displacement.
S=safe_distance-distance

C) **distance > (Δx+safe_distance)**

This indicates that the lead car is at point A and the ACC car is outside the zone AC. So, the ACC car must start accelerating gradually at a speed which is calculated as

setSpeed = speed + (a*time)-(4)

This equation comes from Physics
$v=u+at$

CODE GENERATION

We can also generate C code for a Simulink model or a particular subsystem using the Simulink Embedded coder. Here we have generated C code for the ACC Subsystem described earlier. This code is generated for a specific target processor in this case for Freescale HCS12.

Code Generation steps

1. Before generating code, the target hardware must be set. In order to do that, go to Simulation -> Configuration Parameters -> Hardware Implementation. Set the device vendor to Freescale and device type to HCS12.
2. Go to Simulation -> Configuration Parameters -> Code Generation.
   Set the System target file to ert.tlc.
3. Right click on the Subsystem, select Code Generation -> Build Subsystem and then click on Build.
4. After the build is through without any compilation errors , a code generation report in HTML format opens up, where you can see your generated code files – Subsystem.c, Subsystem.h, Subsystem_private.h, Subsystem_types.h.
5. You can also see all your generated files in the MATLAB build path that you have specified. The files are generated in a folder Subsystem_ert_rtw.
6. This can be integrated with an existing C code by making a function call to Subsystem.c , passing the required parameters and including all the required header files with the compilation environment.

HOW TO GET STARTED WITH THIS PROJECT

Instructions for Use

The test bed should be set-up exactly as explained and shown in the "**System Architecture**" section. Refer to the **test-bed setup (Figure 2)** and **ECU network (Figure 3)** diagrams to get a better understanding of the connections to be made.

The Code required for getting the project up and running is divided into three sections in the **"Final_Submission"** Folder on Github at the following path.
https://github.com/ESE519/AutoPlug-3.0

1) HCS12
2) Matlab Dashboard
3) Torcs

1) HCS12

This folder contains all the code needed for flashing onto the appropriate ECU's. It also contains the code generated and merged using the Simulink Model in the folder named "Simulink_Versions".
The code to flash for each ECU is given below. The IDE to be used is Freescale CodeWarrior.

Engine ECU

Final_Submission - > HCS12 -> Final_Code - > ESE_519-Dist_speed -> Firmware -> CruiseControl -> CruiseControl.mcp

Matlab-CAN Gateway

Final_Submission - > HCS12 -> Final_Code - > ESE_519-Dist_speed -> Firmware -> Serial-CAN-> Serial-CAN.mcp

TORCS-CAN Gateway

Final_Submission - > HCS12 -> Final_Code - > Auto2012 -> Firmware -> CAN-Serial->

CAN-Serial.mcp

Brake ECU

Final_Submission - > HCS12 -> Final_Code - > Auto2012 -> Firmware -> ABS-new -> ABS-new.mcp

Steps for using Simulink generated Code

1. In a PC/laptop with Matlab and Simulink installed, first install the Simulink Embedded coder. This is a licenced software and cannot be downloaded for free. You have to take the licenced version from a TA or Professor or have to buy the version online from the Mathworks website.
2. Open the Simulink model ACC_Model_frontspeedout1.mdl. This can be found at https://github.com/ESE519/AutoPlug-3.0/tree/master/Final_Submission/HCS12/Simulink_Version/Simulink_Model
3. Press the play button to run simulations and verify the results and graphs as indicated in the 'Simulations' section.
4. To generate code for the ACC subsystem, follow all the instructions in the 'Code Generation' section of the report.
5. Now, all the code generated files and also additional files from the MATLAB installation folder are added in the Codewarrior project which you can find at https://github.com/ESE519/AutoPlug-3.0/tree/master/Final_Submission/HCS12/Simulink_Version/ESE_519%20-%20Dist_speed-2/Firmware/CruiseControl/CruiseControl.mcp
6. The manual changes that are needed to correct all the linking errors. Also from the main.c file, a function call is made to the file Subsystem.c which has all the ACC logic. Look at all the files in the project mentioned in the previous step to Check for these changes.
7. Compile and flash the project in Codewarrior.
8. You are now ready to test your system.

2) Matlab Dashboard

An important part of the project is the Matlab Dashboard which has been designed to monitor and control several ACC parameters. The files necessary for running the Matlab Dashboard can be found in the folder named **“Matlab-dashboard”** .

Steps to get the dashboard up and running:

i) Copy the files present inside the Matlab-dashboard folder to a machine having Matlab installed.

ii) Open Matlab and change the directory to the one where the files were copied in Step 1.

iii) Open the AutoPlugGUI.m file and locate the following line and change the COM port number to the appropriate one for the machine.

```
“s = serial('COM4'); %assigns the object s to serial port”
```

iv) Run the code and you can see the Dashboard appear.

v) Make sure that the TORCS race is started and that the correct version of code is flashed for each of the ECU’s as explained in the previous section.

vi) Click the “Start” button in the dashboard to start plotting the graphs that appear on the left side.

vii) The Control panel on the right side allows changing different parameters and visualizing change in the graphs and in the game.

viii) Press “Stop” to finish the graphing. Close and Open the dashboard in Matlab to run the operation again.

3) TORCS

We use the Open Source Racing Simulator – TORCS for the project. All the files needed to get Torcs up and running can be located inside the Torcs folder within the “Final_Submssion “folder.

The version of Torcs used in this project is Torcs-1.3.2-test1. Torcs can be installed on a machine running the latest version of Ubuntu.

Steps for getting Torcs up and running:

1) Install torcs-1.3.2-test1 by following the instructions from the Torcs online installation tutorial.
2) Copy the autonet folder located in torcs-1.3.2-test1 -> src ->drivers into the Torcs installation folder ->src->drivers.
3) Execute the following commands from the Torcs installation folder and make sure to correct all errors that appear. Most errors generally tend to be missing library files which can be installed using “Synaptic Package Manger” for Ubuntu. The Torcs webpage offers a nice FAQ section on common errors and solutions to correct them.

   Commands to be run

   ```
   make clean
   make
   make install
   make datainstall
   ```

   Note: Make sure that the ‘LD_LIBRARY_PATH’, ‘MAKE_DEFAULT’ and ‘TORCS_HOME’ path files are set correctly for the make command above to proceed without error.

4) Proceed to the autonet folder which was copied over in Step 2 and run the following commands.

   ```
   make clean
   make
   make install
   ```

5) Type “torcs” from the terminal and configure a practice race by choosing autonet and autonet2 from the options for cars on the right hand side. “autonet” is the car controlled from the WiiMote and “autonet2” is the car

which is programmed to run automatically around the track. Additionally it can also be controlled from the Matlab dashboard.

6) As stated above, the "autonet" is completely controlled using the Wiimote controller.

Steps for configuring and setting up the Wiimote controller

Pre-requisites:

1) The signals from the WiiMote are transmitted over bluetooth to a bluetooth receiver connected to the system running TORCS. Therefore the system will need to have bluetooth library files installed
2) The bluetooth signals are collected and framed into a CAN message by the USB-PCAN adapter connected to the machine running Torcs. The USB-PCAN adapter forms the CAN messages and transmits then onto the CAN bus as shown in the Architecture section. Therefore it is necessary to have the latest "pcan-usb" drivers installed.
3) The WiiMote controller requires the latest wiiuse drivers to be installed to detect the different movements of the WiiMote. Download the latest wiiuse drivers and follow the instructions and install them.

After all the above 3 pre-requisites are met, proceed to the following steps.

i) Copy the "WiiMote_PCAN_orig" over to the system running Torcs.
ii) Navigate to the copied folder and type the following commands

```
make clean
make
make install
```

Note: The above steps will succeed only if you have successfully installed the bluetooth , wiiuse and pcan-usb drivers as stated in the Pre-requisites section.

iii) After successful completion of Step ii), type the following command.

```
./driver_input
```

At the same time, press the 1 & 2 buttons of the WiiMote to synchronise the controller with the machine. If everything is successful, the WiiMote controller will be detected and the battery level will be shown as a confirmation. Press the "Home" button to enable the WiiMote.

Now everything has been set-up to test run the project. Start the race in TORCS and press the Start button in the Matlab dashboard to visualize the parameters.

ACKNOWLEDGEMENT

We would like to sincerely thank Prof.Rahul Mangharam for giving us a wonderful opportunity to do this interesting and challenging project. Also, special thanks to our mentors - Yash Pant, Nicola Bezzo, Truong X. Nghiem and Miroslav Pajic and also the TAs - Madhur Behl and Abhijeet Mulay for constantly guiding and supporting us throughout the project.