Project 2 Report: Adaptive Cruise Control and Autonomous Lane Keeping

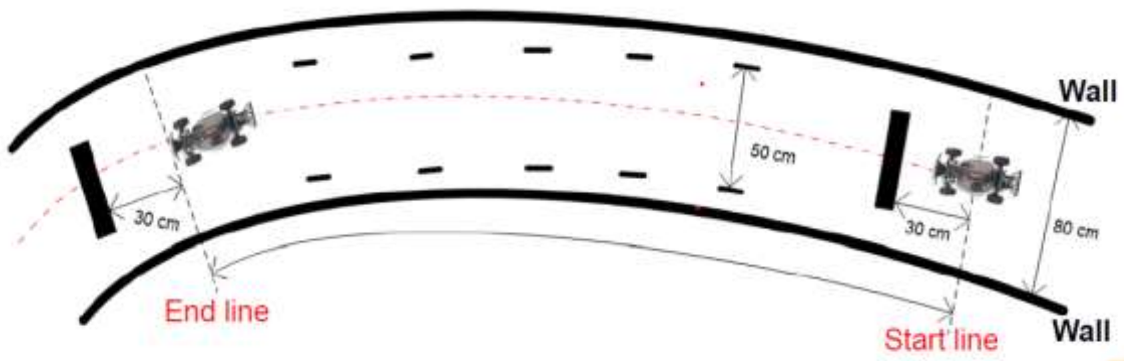
Introduction:

Nowadays, almost all the vehicles come with Adaptive Cruise Control and Autonomous Lane Keeping as a standard active safety feature. While Adaptive Cruise Control helps the driver keep safe distance between the vehicle in the front, the autonomous lane keeping aids in keeping the vehicle within the lanes and prevents it from veering off the road. Project 2 was focused on the simple implementation of these technique.

One of the most important topics covered during the semester was Kalman filter and signal processing. Another important topic was model-based and non-model-based controllers. While the Project 1 focused on the signal processing, the second project was more focused on the controllers.

1. Adaptive Cruise Control

1.1 Problem Statement



The objectives for the Adaptive cruise control can be broken down into the following parts:

1) The vehicle should stop at 30 cm away at the start line from the object in the front.
2) Once the object is removed, the vehicle should traverse the test track.
3) At the end, vehicle should be able to stop 30 cm away from the object placed at the finish line.

1.2 Technical Approach

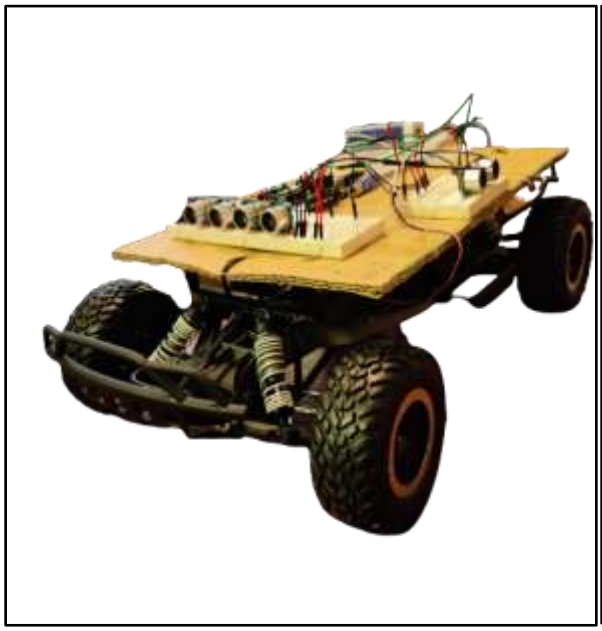
Simply put the approach to Adaptive cruise control was to measure the distance with ultrasonics sensor and use it to give the control input to the ESC.

At first, we decided to approach the problem with model-based controller, especially the full state feedback controller. However, while making the assumption of considering the throttle/control input as acceleration was wrong. Therefore, even after calculating the K matrix with different values of R and Q, we were unable to control the throttle output as needed. For the R and Q value of 1 and 1 respectively, we got the K matrix to be [1, 1,732]. For 4 and 1 value of

R and Q respectively, we derived the k matrix to be [0.5 1]. For the calculation of the K matrix LQR method was used.
Eventually, we shifted to PID controller in particular the discretized PID for Adaptive Cruise Control.

1.3 Hardware and Software Implementation

When it comes to Hardware, we were given a TRAXXAS buggy/RC car skeleton. The Arduino kit from the Project 1 was utilized in this project as well. The Arduino Kit included An Arduino Uno board, multiple HR-SC04 ultrasonic sensors among other things. The following image shows the hardware setup of our vehicle. We mounted two ultrasonic sensors in the front and one on each side of the vehicle. Schematic diagram of the connection is shown in the image next to the RC vehicle.



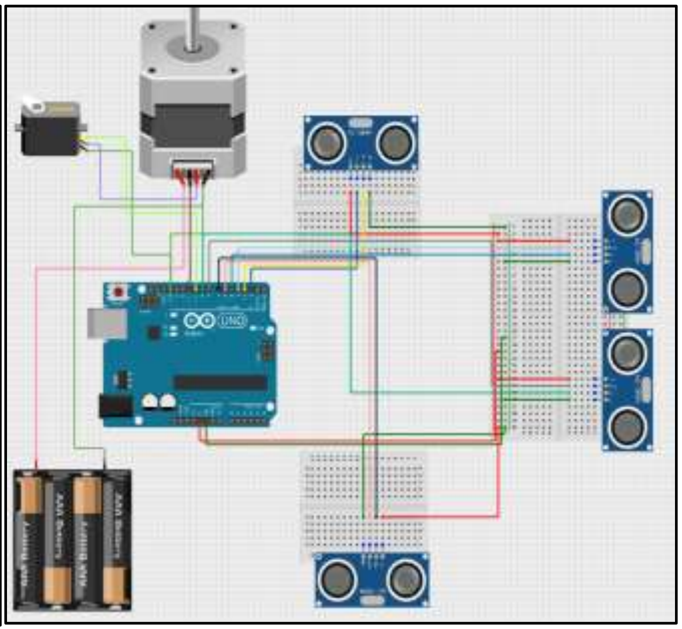


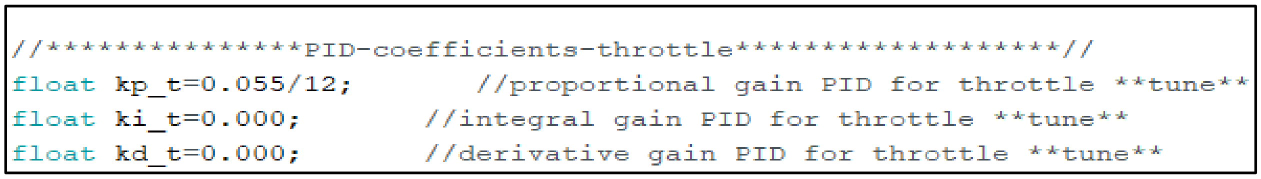
fig. Hardware Setup and Schematic

Software Implementation-

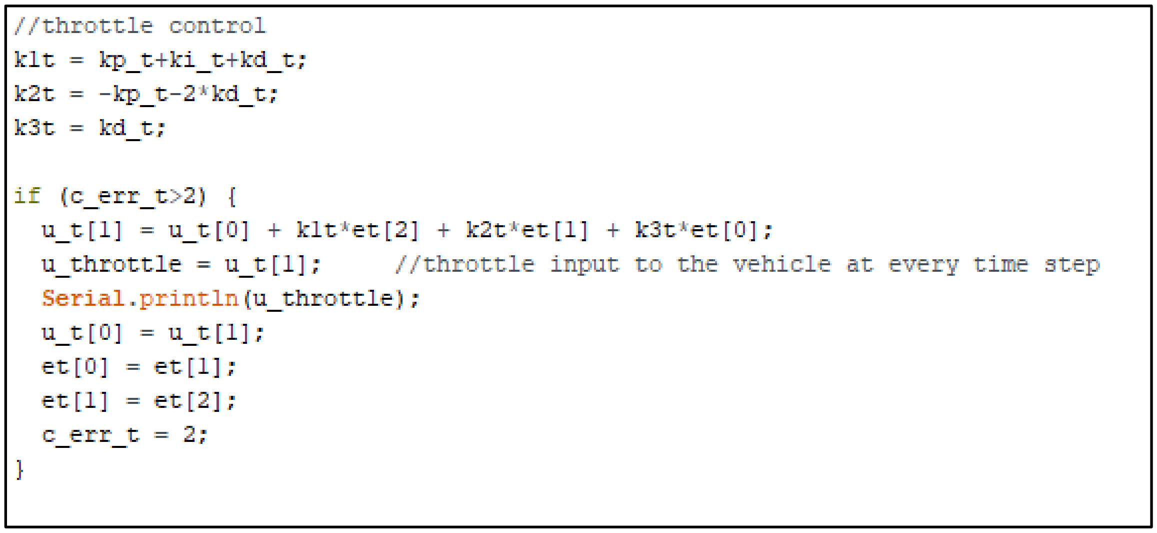
In this section, I will briefly explain some of the functions we used. One of the functions that was provided to us was the “setVehicle” function. This function aimed at the limiting the input given to the ESC and the Steering Servo, to safeguard the delicate internal circuitry. The Vehicle was supposed to have the neutral for throttle and steering as 90 and 90 respectively. However, we found out, even after running the calibration code, that there are operational dead zones for both the ESC and SSM. It was found out that for the ESC, the values from 91 to 94 do not move the vehicle in forward direction, even when the wheel are not touching the ground. For the rearward motion, the dead zones were from 89 to 86. Hence, keeping this in mind we wrote the control output as 95+u for the forward motion and 86-u for the rearward motion.

Initially, we were planning to use Kalman filtering for fusing the data obtained by the Ultrasonic sensors mounted on the front. However, doing so increased the control frequency beyond control introduced more error than anticipated. Hence, we decided to fire each of the Ultrasonic sensors alternatively.

The snippet below shows the values of proportional, integral, and derivative gain for the throttle control. From the image at the bottom, we used the discretized PID controller.



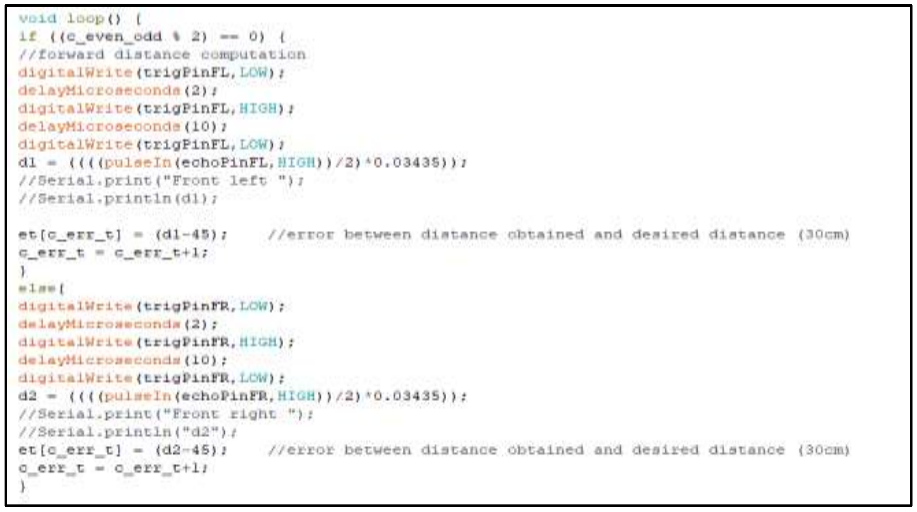
```
//***************PID-coefficients-throttle*******************//
float kp_t=0.055/12;       //proportional gain PID for throttle **tune**
float ki_t=0.000;       //integral gain PID for throttle **tune**
float kd_t=0.000;       //derivative gain PID for throttle **tune**
```



```
//throttle control
k1t = kp_t+ki_t+kd_t;
k2t = -kp_t-2*kd_t;
k3t = kd_t;

if (c_err_t>2) {
  u_t[1] = u_t[0] + k1t*et[2] + k2t*et[1] + k3t*et[0];
  u_throttle = u_t[1];     //throttle input to the vehicle at every time step
  Serial.println(u_throttle);
  u_t[0] = u_t[1];
  et[0] = et[1];
  et[1] = et[2];
  c_err_t = 2;
}
```

fig. PID controller for Throttle control

The code snipped below shows the how we calculated the error term to be used for PID controller. In the approach section, I have already mentioned, that we decided to trigger the front ultrasonic sensor alternately. Also, the vehicle was expected to stop 30 cm away from the obstacle. Considering the positioning of our sensors, we calculated error term as (distance - 45). That can also be seen in the image below.



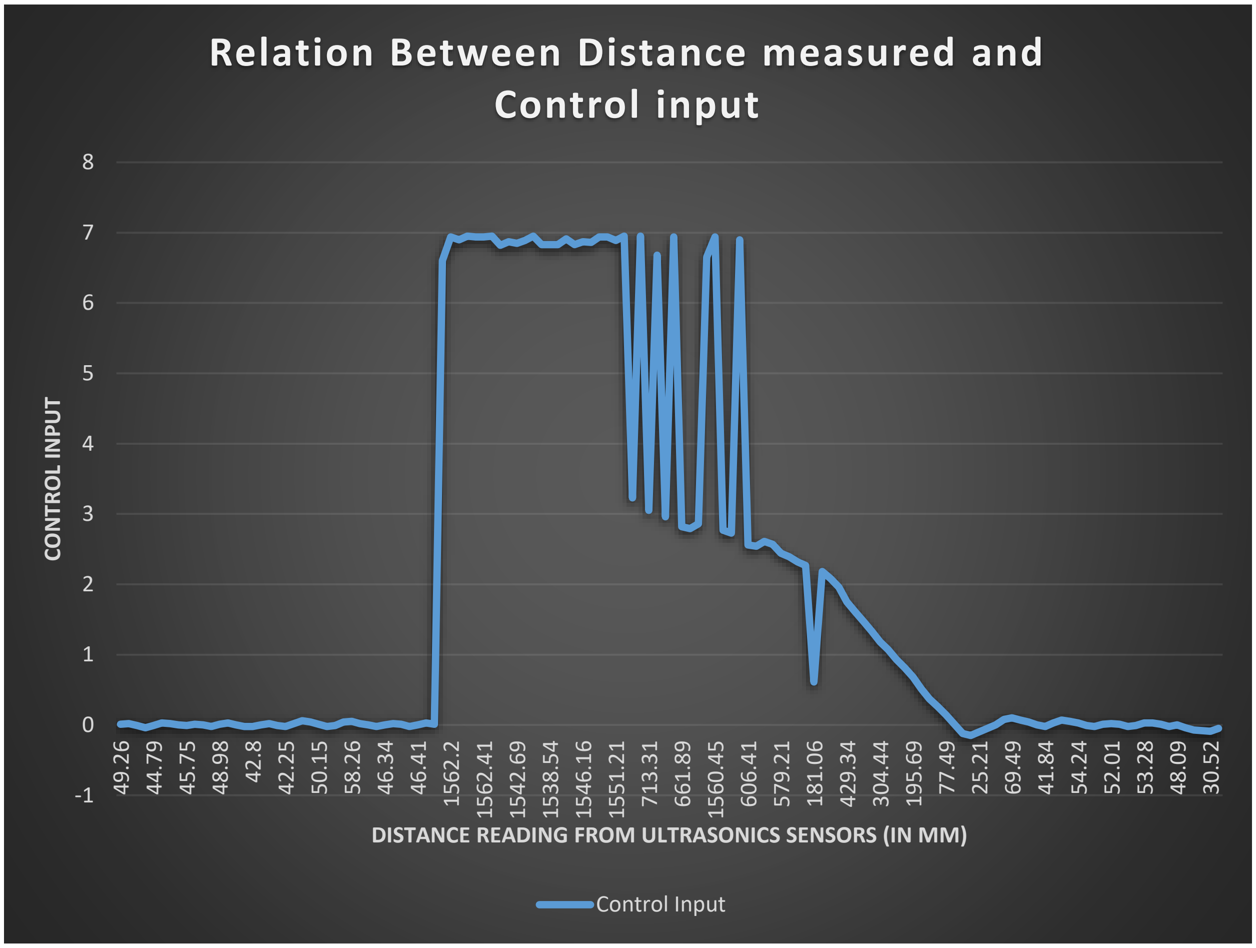
```
void loop() {
if ((c_even_odd % 2) == 0) {
//forward distance computation
digitalWrite(trigPinFL,LOW);
delayMicroseconds(2);
digitalWrite(trigPinFL,HIGH);
delayMicroseconds(10);
digitalWrite(trigPinFL,LOW);
d1 = ((((pulseIn(echoPinFL,HIGH))/2)*0.03435));
//Serial.print("Front left ");
//Serial.println(d1);

et[c_err_t] = (d1-45);    //error between distance obtained and desired distance (30cm)
c_err_t = c_err_t+1;
}
else{
digitalWrite(trigPinFR,LOW);
delayMicroseconds(2);
digitalWrite(trigPinFR,HIGH);
delayMicroseconds(10);
digitalWrite(trigPinFR,LOW);
d2 = ((((pulseIn(echoPinFR,HIGH))/2)*0.03435));
//Serial.print("Front right ");
//Serial.println("d2");
et[c_err_t] = (d2-45);    //error between distance obtained and desired distance (30cm)
c_err_t = c_err_t+1;
}
```

fig. error term for the PID

One of the problems we faced was that our vehicle was carrying its momentum down the slope or the test rack. Hence, it was colliding with wall/ stop wall at the end of the track. In order to stop this, we created a nested if conditions, to control the length of the pulse. As the vehicle got closer to the end/ or wall, the "ON" duty cycle was reduced more and more.

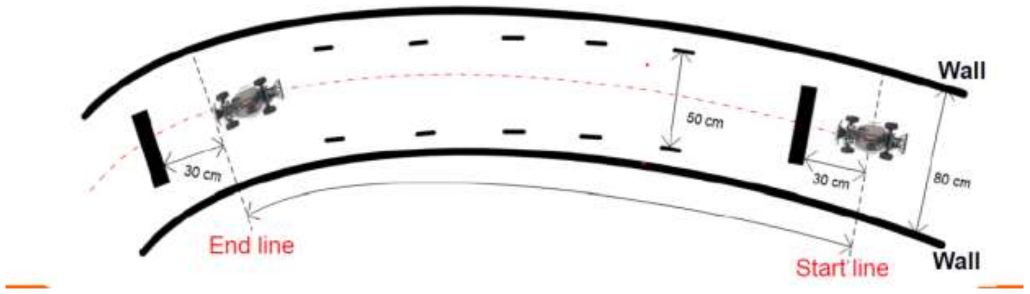
1.4 Experimental Results

The chart below shows the relation between the distance measured by the Ultrasonics sensors and the control input given to the ESC. It can be seen that as the value of the distance readings increase, the control input for the ESC is also increased. Since we used the PID controller for this task, when there is significant change in the controller input, it can be seen to overshoot before stabilizing. This can be due to high value of Kp as compared to the Ki and Kd.



2. Autonomous Lane Keeping

2.1 Problem Statement



The objectives for the Autonomous Lane Keeping was simply to keep the vehicle in the middle of the track. One of the main challenges was that vehicle must remain in the center while going through the checkpoints placed.

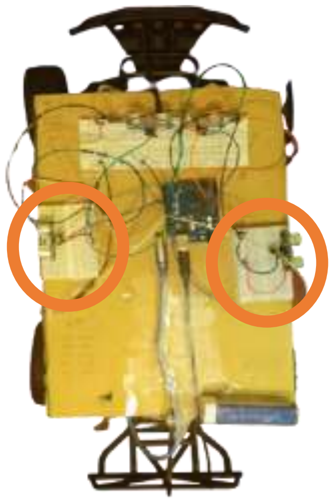
2.2 Technical Approach

The problem statement above underestimates the challenge posed here. We had decided to use the discretized PID controller for this task from the beginning.

One of the challenges we faced in this task was figuring out the neutral steering value. By default, the neutral value (value/input to the SSM which does not steer the vehicle) of RC car was supposed to be 90. However, due the high negative camber on the left wheel, the vehicle would take a sharp left turn even for neutral steering input. This also meant that there was an operational dead zone for right turn. We found that for our vehicle, 100 was the value for neutral steering. However, this meant that the values between 90 and 100 had no effect on the steering.

2.3 Hardware and Software Implementation

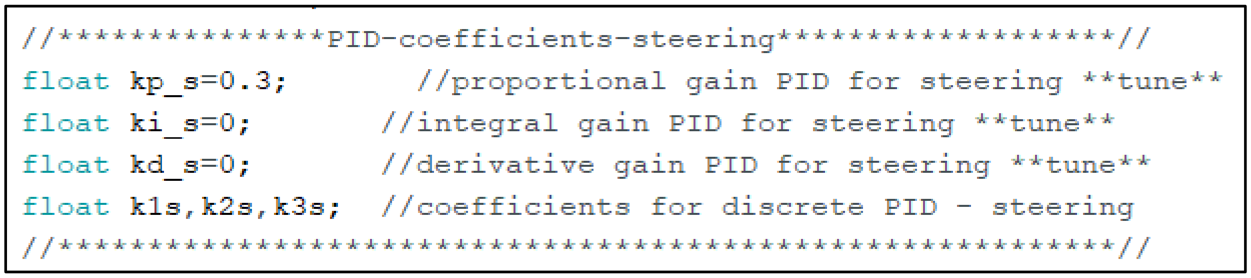
The Hardware setup remained pretty much the same for this part of the project. The only difference in the setup is the sensors used and their location with respect to the vehicle. As described in the earlier section of this report, we mounted two ultrasonic sensors on the either side of the vehicle. Apart from this, there were no changes in the hardware setup.



Software Implementation

This part of the project was straightforward once we figured out the dead zones mentioned in the technical approach section. I have already mentioned earlier in the report that we used the discretized PID for this. In this section, I will briefly explain the key components of software implementation.

The following snippet shows the controller gains we used.



```
//***************PID-coefficients-steering*******************//
float kp_s=0.3;       //proportional gain PID for steering **tune**
float ki_s=0;       //integral gain PID for steering **tune**
float kd_s=0;       //derivative gain PID for steering **tune**
float k1s,k2s,k3s;  //coefficients for discrete PID - steering
//***********************************************************//
```

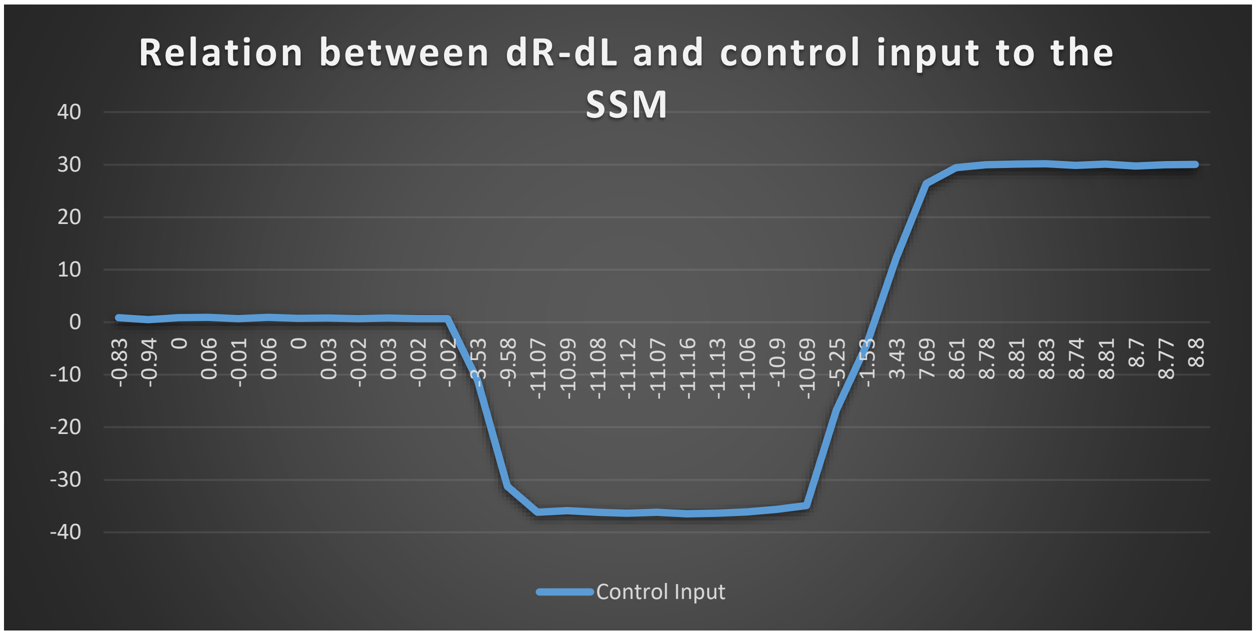
To resolve the errors in steering system caused by the operational dead zones, we designed our control input for the steering as below-

1) If dR-dL is positive, then the vehicle should turn right. Since the vehicle's neutral steering value was observed to be 100. The control input to turn right was chosen to be 100+u. The value of 'u' was calculated from the PID.
2) If dR-dL is negative, then the vehicle should turn left. The vehicle neutral was observed to be 100 but there was a operational dead zone from 100 to 90. Hence, the control input to turn the vehicle left was 88+u.

We also noticed that throttle input and steering input cannot be given simultaneously. This is because the ESC is powerful. This would just make the front wheel skid and not turn.

2.4 Experimental Results

The chart below shows the relation between the difference between the distances measured by Right and Left Ultrasonics sensors and the control input given to the SSM. If the value of dR-dL (where dR and dL are distances measured by the Ultrasonic sensors on the right side and left side respectively) goes below zero that means that the vehicle should turn left. If it goes above zero, the vehicle should turn right. As we can see in the plot below that control input changes the values according to the difference. However, this plot does not show the exact relationship between the positive control input to the SSM from the negative input.



3. Conclusions and Discussions

3.1 Conclusions (a summary the results of different approaches)

In Adaptive Cruise Control, we observed that the Battery SoC affected the performance of the ESC. Same can be said for the Steering Servo Motor. Too low of control frequency results in the vehicle not moving, whereas too high control frequency results in erratic movement of the vehicle.

In Autonomous Lane Keeping, the high negative camber induced in the vehicle resulted in many different problems. One of the main problems was the vehicle would to the left even after the vehicle was set to move forward with no steering.
A simple PID controller might not be as accurate as the model-based controller of any other type, however, for the given application it was quite reliable.

3.2 Discussions (a comparison of different approaches, and potential future work to further improve each approach)

One of the noteworthy things we observed that for a full state feedback controller or model-based controller to work, one need to model the equation for the system as accurate as possible. Inconsistencies in the model created may cause the model to not function at all.

One of the points that can improved significantly is improved filter that filters out stray values without increasing the time required for the execution. The vehicle had some inherent high negative camber. This affected the working of the controller tremendously. Vehicle Parameters should be kept neutral for initial setup and then once the working base model created such parameters could be changed.

The test case could be made better where in the middle of the track (at random place) a wall like structure could be placed. The vehicle should stop here until the object is lifted and continue till the exit. This would be more appropriate test of the Adaptive Cruise Control.

(Some of the images from both the reports were taken from the Group presentation slide.)