Project 2 Report: Adaptive Cruise Control and Autonomous Lane Keeping

1. Adaptive Cruise Control

1.1 Problem Statement

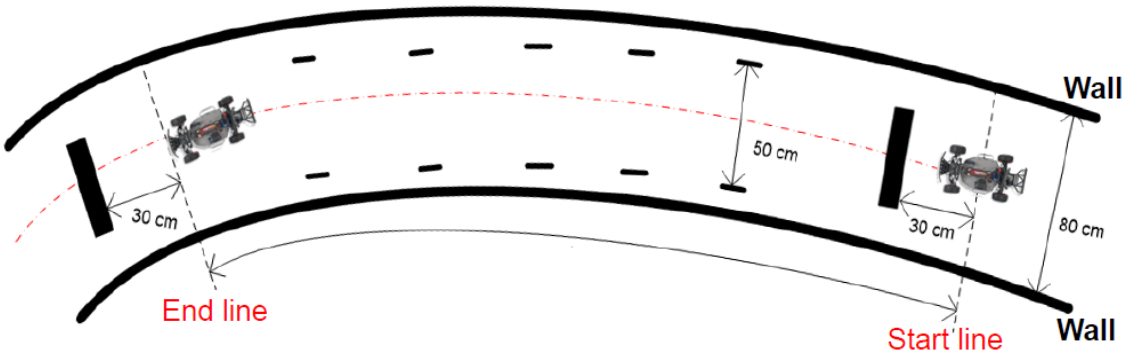

Fig1: Depiction of the problem statement

Design and implement an embedded controller based on ultrasonic sensors to make the RC car run down the ramp as fast as possible.

The problem statement for Adaptive Cruise Control are as follows –

- Keep the vehicle 30 cm away from obstacles ahead.
- If obstacles ahead are stationary, vehicle should stop at 30 cm away from the obstacles.

1.2 Technical Approach

- The ultrasonic sensor at the front will measure the distance of the front obstacle.
- Using a closed-loop (PID) controller the speed of the vehicle can be manipulated to keep the vehicle 30 cm away from the obstacle.
- Tuning the parameters of the PID controller to gain accuracy.
- Designing a Throttle cutoff function to decelerate the car on time.
- Defining appropriate delays in the code to cut-off the throttle continuously, so that the car doesn't over speed.

Sensor Selection

Three ultrasonic sensors for this test were selected based on the accuracy and variance of their measured test data.

Sensor Placement

Three ultrasonic sensors were placed on the RC car at the following locations –

- At the front of the car: This sensor will measure the distance of the front obstacle.
- At the front left side of the car: It measures the distance of the car from the left wall.
- At the front right side of the car: It measures the distance of the car from the right wall.

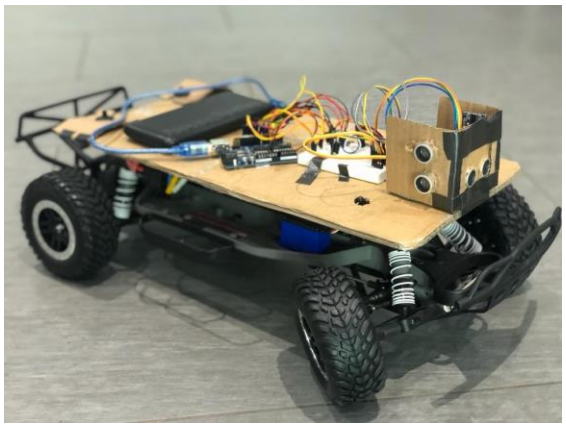

Fig2: RC car setup with three ultrasonic sensors

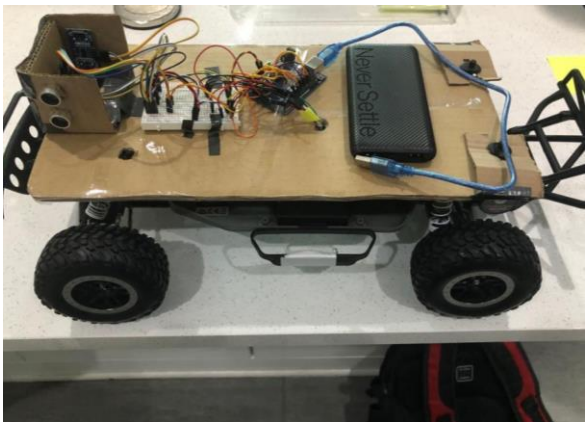

Fig3: RC car setup with three ultrasonic sensors

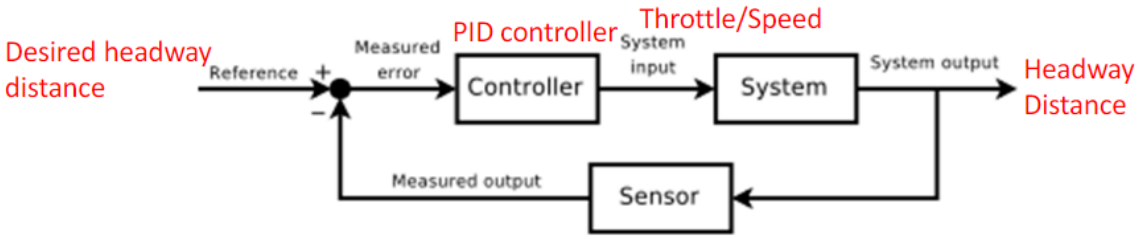

Fig4: Theory of PID controller to control throttle

A PID controlled was implemented to control the longitudinal movement of the vehicle. The input to the PID controller was the error between the measured distance and the set value. The output of the PID was then transitioned to control the throttle of the vehicle by using the 'map' function.

For this test, the throttle was limited to 85 to 95. Where, 85 is the maximum reverse throttle, 95 is the maximum forward throttle and 90 for idling.

```
//PID front parameters

#define kp_front 0.5975
#define ki_front 0.0001
#define kd_front 0.26
```

Fig5: PID constants for throttle

Braking

Since the ramp had a considerable steep decline, braking on the ramp was a huge challenge. To make the car stop as fast as possible, a braking function was implemented which was programmed to give reverse throttle when the car approaches an obstacle within the defined range.

If the distance of the obstacle in front of the vehicle is equal to 150/100/50. Then, the throttle is set to 85 with a delay of 8 milliseconds. After which, the vehicle stabilizes itself and then throttles forward at 95 with appropriate delays defined later in the program.

```
//////////Brake//////////

  if (E>50){
  E = distance3-50 ;
  setVehicleHR(steering, 90);
  setVehicleHR(steering, 85);
    delayMicroseconds(6000);
  setVehicleHR(steering, 90);
  }
  if (E>50){
  E = distance3-100 ;
  setVehicleHR(steering, 90);
  setVehicleHR(steering, 85);
    delayMicroseconds(8000);
  setVehicleHR(steering, 90);
  }
  if (E>50){
  E = distance3-150 ;
  setVehicleHR(steering, 90);
  setVehicleHR(steering, 85);
    delayMicroseconds(8000);
  setVehicleHR(steering, 90);
  }
```

Fig6: Braking function (here E is the difference between the measured distance and the set point)

Speed Control

Since, the downward slope of the ramp was adding to the velocity of the vehicle. To control the forward velocity of the car several delays in speed were given to make the car turn and stop efficiently.

```
setVehicleHR(steering, throttle);
steering = Fvalue;
throttle = throttle_new;
throttle = max(min(throttle, 95), 85);
diff = throttle - 90;
if (abs(diff) < 5) {
  if (diff < 0)
    setVehicleHR(Fvalue, 85);
  else if (diff > 0)
    setVehicleHR(Fvalue, 95);
  delay(int(abs(diff) * 3));
  setVehicleHR(Fvalue, 90);
}
```

Fig7: Delay function to control speed of the vehicle

1.3 Hardware and Software Implementation

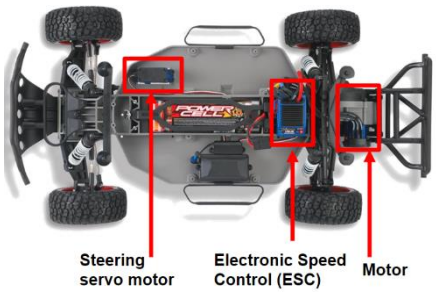

Fig8: The RC vehicle

Fig9: Battery pack/power bank

Fig10: Arduino UNO R3 controller

Fig11: Ultrasonic sensor

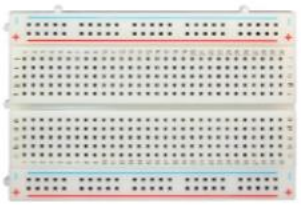

Fig12: Breadboard

Fig13: Wires for connection

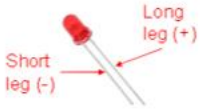

Fig14: Buzzer

Fig15: Resistor (1kohm)

Software Used

-Arduino IDE 1.8.16 was used to implement the code.
- Library <Servo.h>
- setvehicleHR function to set the steering output from the PID controller
- Implementation of PID controller to control the set parameters

-Functions in Excel and MATLAB – 'linear fit' and 'polyfit' were used to calibrate the data.

1.4 Experimental Results

The graph below depicts the results as the RC vehicle approached the wall, overshot the minimum 30 cm distance and then reversed back to the set desired distance of 30 cm.

The 'Blue' line depicts the output distance as sensed by the front sensor.

The 'Red' line depicts the throttle output as per the mapped values from the PID.

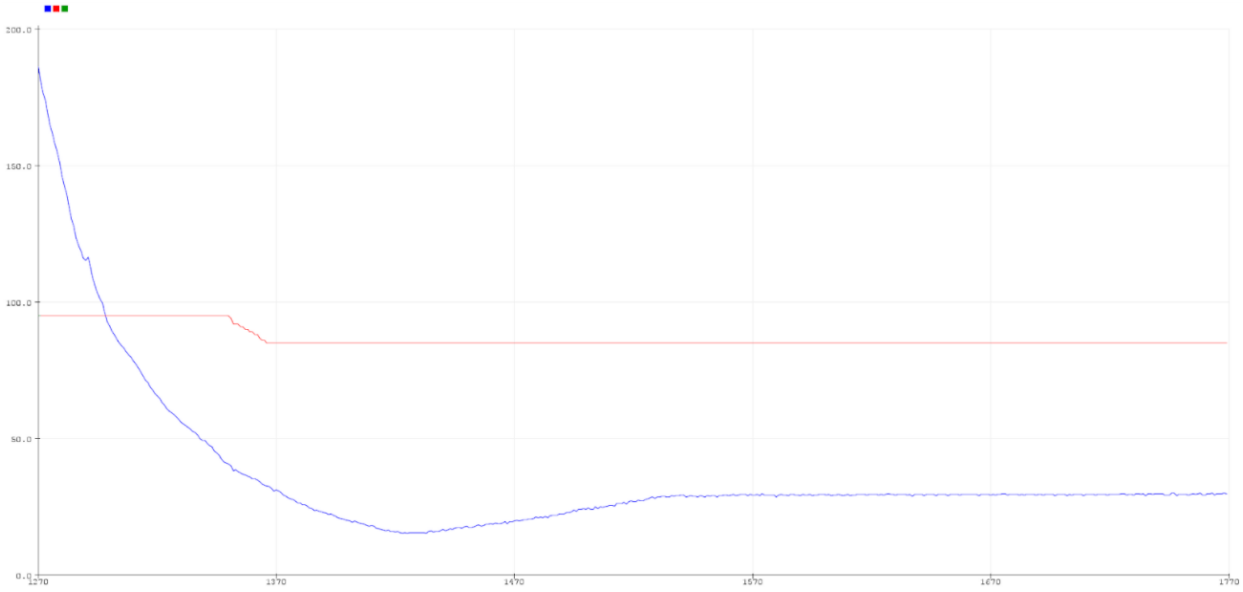

Fig16: Showing throttle output as the obstacle approaches the vehicle

2. Autonomous Lane Keeping

2.1 Problem Statement

- To keep the vehicle at the center of the defined lane using the ultrasonic sensors.
- To pass the vehicle from the five checkpoints of 50 cm width.

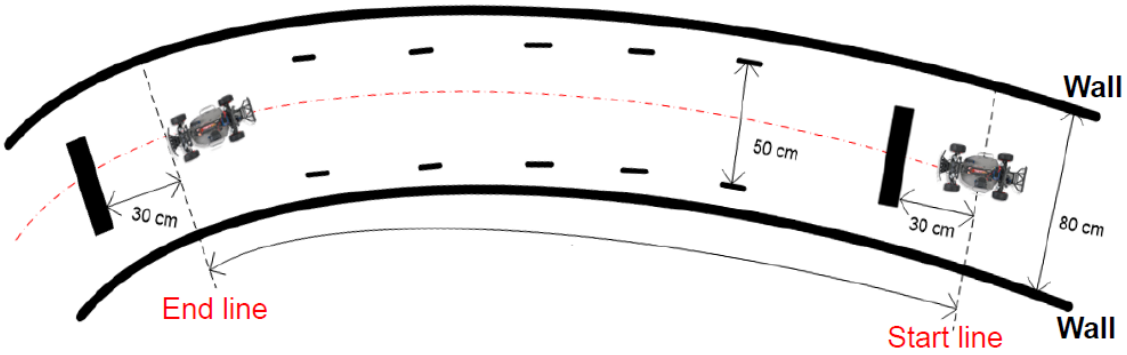

Fig17: Depiction of the problem statement

2.2 Technical Approach

- PID was used to control the lateral movement (steering) of the vehicle.
- Two inputs from the left and right sensors are converted to one attribute to feed into the PID controller.
- Based on the conversion of two inputs to one attribute, the control signal output from the PID is made to control the steering left and right respectively
- A minimum threshold on sensing is set to be make the steering more responsive.
- Steering or lane keeping while reversing is addressed by reversing the steering control while reversing.
- To control the steering entirely, we choose set the steering control throughout the loop with the controller output

Reverse Steering

To make the car align properly while reversing near the walls on the side. A function for reverse steering was implemented. The goal of this function was to reverse the direction of steering if the car reverses back near the walls. This mimics how we reverse actual vehicles in real scenario. Hence, it keels the car aligned and ready to go.

```
//////////Reverse Steering//////////

int FR_multiplier = 1;
if (throttle_new < 90) {
  FR_multiplier = -1;
}
else if (throttle_new > 90) {
  FR_multiplier = 1;
```

Fig18: Reverse Steering function

2.3 Hardware and Software Implementation

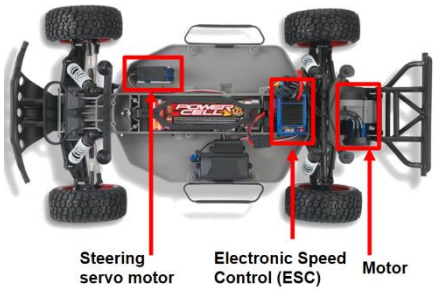

Fig19: The RC vehicle

Fig20: Battery pack/power bank

Fig21: Arduino UNO R3 controller

Fig22: Ultrasonic sensor

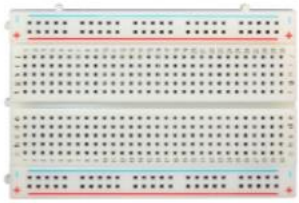

Fig23: Breadboard

Fig24: Wires for connection

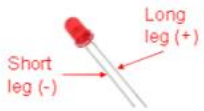

Fig25: Buzzer

Fig26: Resistor (1kohm)

Software Used

-Arduino IDE 1.8.16 was used to implement the code.
- Library <Servo.h>
- setvehicleHR function to set the steering output from the PID controller
- Implementation of PID controller to control the set parameters

-Functions in Excel and MATLAB – 'linear fit' and 'polyfit' were used to calibrate the data.

2.4 Experimental Results

The graph below shows the results from the experiment.

A video of one of the best run for this test: *Click Here!*

From left to right:

Orange line = Throttle
Pink line = Steering output
Red line = Output from the right ultrasonic sensor
Blue line = Output from the left ultrasonic sensor
Green line = Minimum distance from left and right ultrasonic sensor

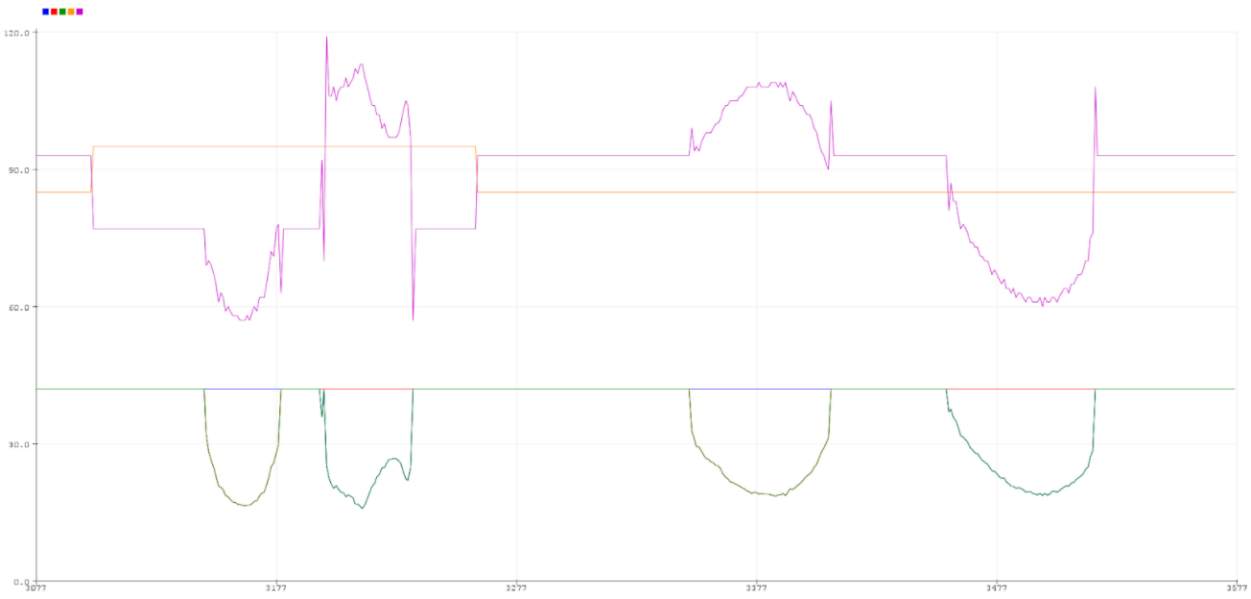

Fig27: Showing steering output as the obstacle on sides approaches the vehicle

3. Conclusions and Discussions

3.1 Conclusions (a summary the results of different approaches)

The problem in this project was addressed by implementing a non-model based closed loop controller. Although, PID is not as accurate as model-based controller, it was preferred because of the ease of implementation.

In the autonomous lane keeping, the signals from both the sensors were challenging to implement. Therefore, the least of the two readings was taken into consideration and was used to calculate the error for the PID to estimate the output parameter by controlling the steering. Also, it was quite hard to control the speed of the RC vehicle because of the descend on the ramp. To counter this – delays, braking function, and velocity cut-off functions were used.

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

In this problem, the approach was to first measure the distances on the sides and front of the vehicle. Considering the results from the 1st project the best sensors were placed on the RC car.

The next approach to this problem could be to rectify the error of signals from the ultrasonic sensors. Also, Kalman filter could be used to increase the accuracy of every reading from the sensors. Not to forget, since this is a dynamic system implementing Kalman filter and error rejection function could make the system less responsive.

Notice: The data in an individual report should be recorded individually. Students in one group can use the same hardware and software but cannot use the same data in the report.