Project 2 Report: Adaptive Cruise Control and Autonomous Lane Keeping

(Run Down the Hill)

1. Adaptive Cruise Control

1.1 Problem Statement:

To design an Autonomous RC vehicle by using ultrasound sensors for adaptive cruise control and keep the vehicle 30 cm away from obstacles ahead.

1.2 Technical Approach

1) **Data sampling** – Implementing one ultrasound sensor to measure the front distance from the RC vehicle. Sampling to be done as quickly as possible to detect the objects nearby quickly and allow autonomous vehicle to react accordingly.

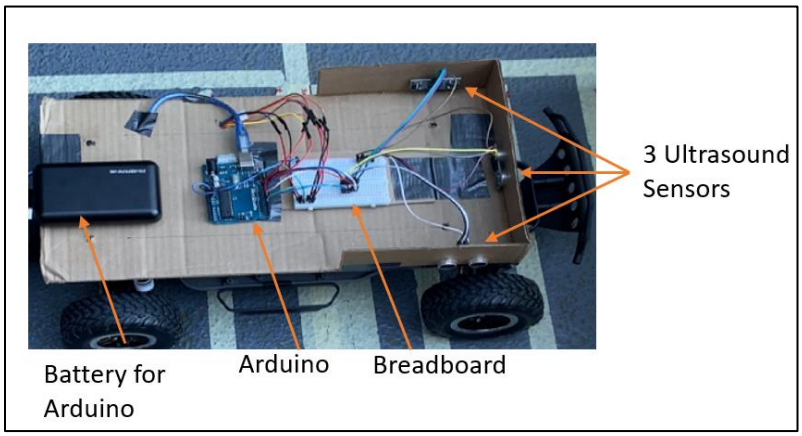
2) **Setting limits** – Setting the desired distances from the front wall by using if and else loop to control RC vehicle throttle and steering. (Keeping RC 30 cm from the frontal wall or obstacle)

3) **Pulse width modulation** – Setting positive and negative values of throttle for accelerating and braking whenever required.

4) **PID controller** – Using PID controller to tune and reduce the error in the system. Iterating the Proportional gain, Integral gain and differential gain for both the throttle and steering to allow RC vehicle follow adaptive cruise control.

1.3 Hardware and Software Implementation

Hardware Implementation:



1) Arduino controls the steering servo motor and the electronic speed control (ESC) / motor based on the sensing by 3 ultrasound sensors.
2) Ultrasound sensors are placed facing left, right and front detecting the distances and sending signal to for further control.
3) The battery or power bank attached provides power to the Arduino.

Software Implementation:

1) Throttle sensor reading:
Collecting the data from front ultrasound sensor for throttle input

```
//////Throttle front sensor reading/////////////////
float dist() {
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(8);
  digitalWrite(trig, LOW);
  duration = pulseIn(echo, HIGH);
  distance = (duration * 0.03435) / 2;
  //  delay(10);
  return distance;
}
```

2) Implementing PID control for throttle:
PID values defined in the beginning and controlling the throttle with respect to the PID values.

```
////////////////////PID for throttle/////////////////
#define minthrottle 77
void PIDt() {

  float Errort = setP - dist();
  float Pvalue = Errort * kpt;
  float Ivalue = toE * kit;
  float Dvalue = (Errort - preE) * kdt;

  float PIDtvalue = Pvalue + Ivalue + Dvalue;
  preE = Errort;
  toE += Errort;
  //  Serial.println(PIDtvalue);
  int Trotvalue = (int)PIDtvalue;
  if (Trotvalue >= 0)
    thr = map(Trotvalue, 0, 10, 90, minthrottle);
  else thr = map(Trotvalue, 0, -200, 90, 95);

  thr = max(min(thr, 95), minthrottle);
```

```
#define kp 0.8
#define ki 0.0001
#define kd 2
```

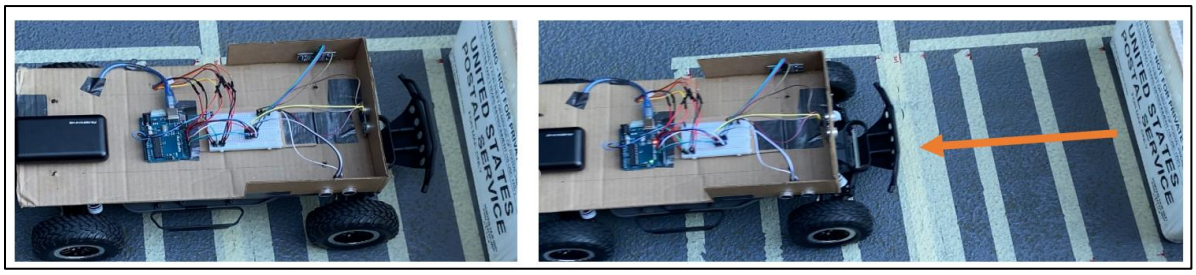
Throttle PID

3) Arduino code for hard braking with delay once the 30 cm distance is approaching:

```
 if (thr > 90) {
   setVehicle(Steervalue, 95);
   delay(50);
   setVehicle(Steervalue, 90);
   delayMicroseconds(10);
 }
 else if (thr == 90) {
   setVehicle(Steervalue, 90);
 }
 else setVehicle(Steervalue, 75);
 // delay(max(abs(thr - 90) * 10, 40));
 delay(60);
 setVehicle(Steervalue, 90);
 delayMicroseconds(10);
 Serial.print(Steervalue);
 Serial.print("\t");
```

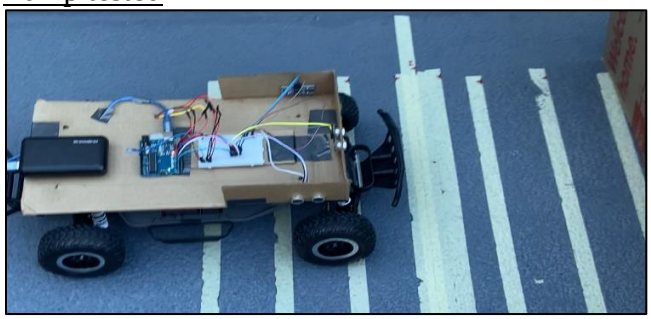
1.4 Experimental Results

1) The RC successfully vehicle adjusted its position to be 30 cm away from the box placed near than 30 cm.



2) The RC vehicle stopped at 30 cm before the box/wall kept at front.

Ramp tested



Data recorded individually as tested in the shown ramp:

The vehicle was successfully able to stop 30 cm before the frontal wall for adaptive cruise control

*The data points recorded by me are mentioned below in the **appendix at the end** as number of data points recorded were more.*

2. Autonomous Lane Keeping

2.1 Problem Statement

To control an Autonomous RC vehicle by using two ultrasound sensors for autonomous lane keeping. Allow the vehicle to be driven in the center of the ramp (80 cm wide).

2.2 Technical Approach

1) **Data sampling** – Implementing 2 ultrasound sensors to measure the distance from left and right side from the RC vehicle. Sampling to be done as quickly as possible to detect the objects nearby quickly and allow autonomous vehicle to react accordingly.

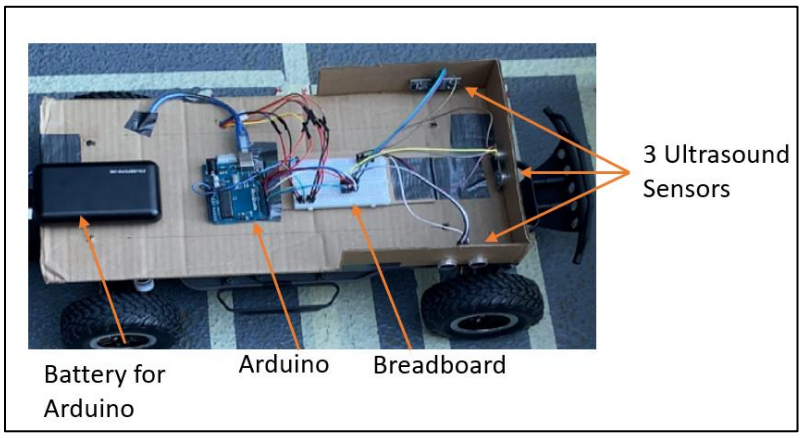
2) **Setting limits** – Setting the desired distances from left and right walls by using if and else loop to control RC vehicle throttle and steering. (Keeping RC car at the center of the track)

3) **Pulse width modulation** – Setting positive and negative values of throttle for accelerating and braking whenever needed.

4) **PID controller** – Using PID controller to tune and reduce the error in the system. Iterating the Proportional gain, Integral gain and differential gain for both the throttle and steering to allow RC vehicle follow autonomous lane keeping.

2.3 Hardware and Software Implementation

Hardware Implementation:



1) Arduino controls the steering servo motor and the electronic speed control (ESC) / motor based on the sensing by 3 ultrasound sensors.
2) Ultrasound sensors are placed facing left, right and front detecting the distances and sending signal to for further control.
3) The battery or power bank attached provides power to the Arduino.

Software Implementation:

1) Steering sensor reading:
   Collecting the data from left and right ultrasound sensors for steering input.

```
//////Steering side sensor reading/////////////////
float d_left() {
  digitalWrite(trig1, LOW);
  //digitalWrite(light, LOW);
  delayMicroseconds(2);
  digitalWrite(trig1, HIGH);
  delayMicroseconds(8);
  digitalWrite(trig1, LOW);
  duration1 = pulseIn(echo1, HIGH, 10000);
  if (duration1 != 0)
    distance1 = ((0.176291554 * duration1) - 12.52197797) / 10;
  else return 1000;
  return distance1;
}
float d_right() {
  digitalWrite(trig2, LOW);
  //digitalWrite(light, LOW);
  delayMicroseconds(2);
  digitalWrite(trig2, HIGH);
  delayMicroseconds(8);
  digitalWrite(trig2, LOW);
  duration2 = pulseIn(echo2, HIGH, 10000);
  if (duration2 != 0)
    distance2 = ((0.176291554 * duration2) - 12.52197797) / 10;
  else return 1000;
  return distance2;
}
```

2) <u>Implementing PID control for steering:</u>

PID values as defined to control steering servo motor for autonomous lane keeping

```
//////////////////PID for Steering////////////
void PID() {
  //  float error = StrD - (dis() * side());
  float error = dis();
  float Pvalue = error * kp;
  float Ivalue = toError * ki;
  float Dvalue = (error - priError) * kd;

  float PIDvalue = Pvalue + Ivalue + Dvalue;
  toError = priError;
  priError = error;
  Steervalue = max(min(PIDvalue * 5, 250), -250);

  Steervalue = map(Steervalue, -250, 250, 50, 130);
```

```
#define kpt 1.2
#define kit 0
#define kdt 6
```

<u>Steering PID</u>

2.4 Experimental Results

The RC vehicle follows autonomous lane keeping by sensing the distance to the side walls by using 2 ultrasound sensors on either side.

After sensing it successfully keeps the RC vehicle in the center and allows for autonomous lane keeping. PID controller is used to set the steering inputs. (Refer below image for RC vehicle following the autonomous lane keeping)

*The data points recorded by me are mentioned below in the **appendix at the end** as number of data points recorded were more.*

3. Conclusions and Discussions

3.1 Conclusions (a summary the results of different approaches)

1) Autonomous lane keeping and adaptive cruise control: RC vehicle successfully follows autonomous lane keeping and adaptive cruise control (Stopping 30 cm before the wall/obstacle) with 3 ultrasound sensors, PID controller and pulse width modulation.

2) Pulse width modulation: RC vehicle desired speed is achieved by adjusting the negative and positive throttle inputs for braking and accelerating.

3) PID implementation for throttle and steering control: Iterating Kp, Ki & Kd values gives you different response on steering and throttle and tuning of these gains is necessary to achieve better autonomous lane keeping and adaptive cruise control.

Other approaches:

1) Using Kalman Filter to filter the sample distance received from the ultrasound sensor.
2) Implement model base controller design to make the autonomous RC vehicle work in asymptotically stable method.
3) Controlling the throttle inputs based on the slope detected by the gyroscopic sensor.

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

1) Using Kalman filter:

1) Ultrasound sensor detect the distances from the wall but are not accurate.

2) To improve accuracy of the distance data, Kalman filter can be used in the Arduino code to measure predict exact distance allowing the vehicle to be stable and perform better for autonomous lane keeping and adaptive cruise control.

3) High power Arduino may be required to implement the Kalman filter

2) Implementing model-based controller design:

As a group, we tried to implement the PID controller for adaptive cruise control first and we executed and performed well in the task.

For the further opportunity, implementing model-based controller design for adaptive cruise controller would give better RC vehicle control.

3) Integrating gyroscopic sensor for angular data detection.

A potential future work to control an autonomous RC vehicle would be to implement the gyroscopic sensor to detect the angle for the slope it is travelling on. After detection of the angle, the throttle input will be adjusted accordingly to maintain same speed at all angles.