ECE 2010 | Control Systems

Review III

Motor speed regulation

Allen Ben Philipose | 18BIS0043
Shayon Gupta | 18BIS0154

Electric car with adjustable speed

Acknowledgement -

I would like to thank Prof. Bagubali A, School of Electronics Engineering Department Faculty at the Vellore Institute of Technology, Vellore, for his guidance, patience and time. I hope the given project will be a starting point for us students to learn more about different methods of controlling a system and understanding its stability.

Objective -

The system is designed keeping in mind its practicality and versatility. It should be able to stabilize itself to a set value within a short period. It should be able to do so with varying load. It should be able to achieve all of this without exceeding the budget and the output must be repeatable.

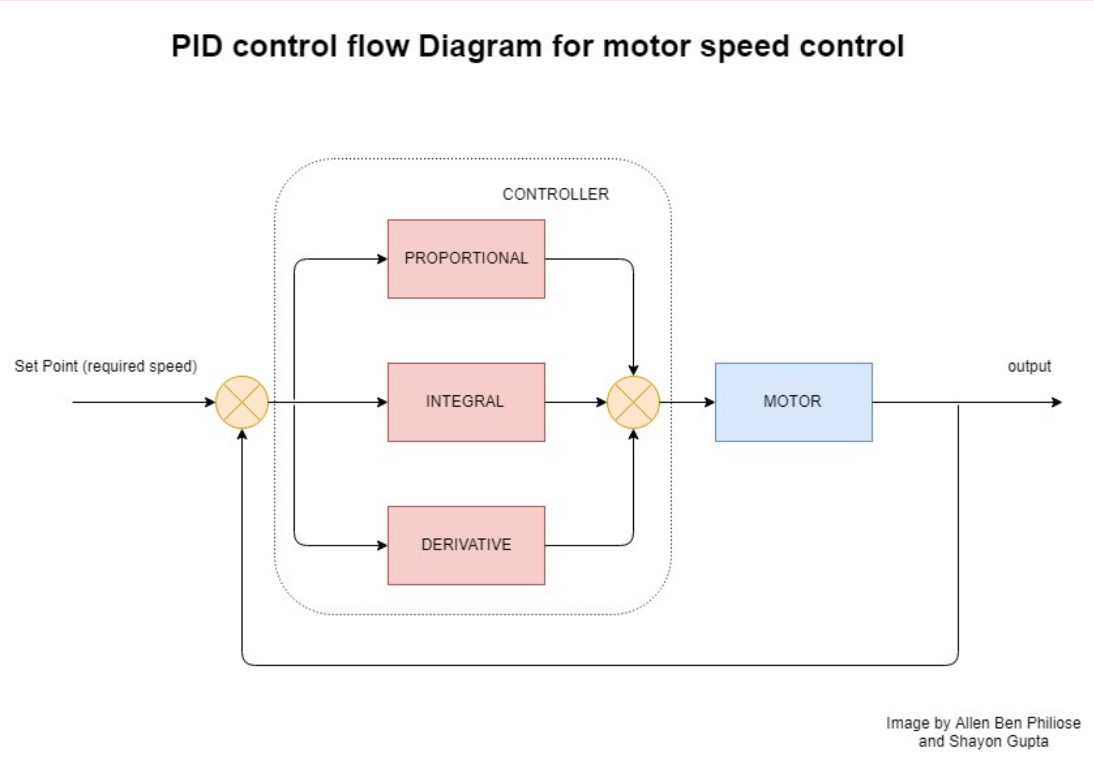
Methodology -

We first implemented the algorithm to a MATLAB program that will help visualize the behavior of the system by graphing it out. Once the algorithm was finalized, we applied it to hardware.

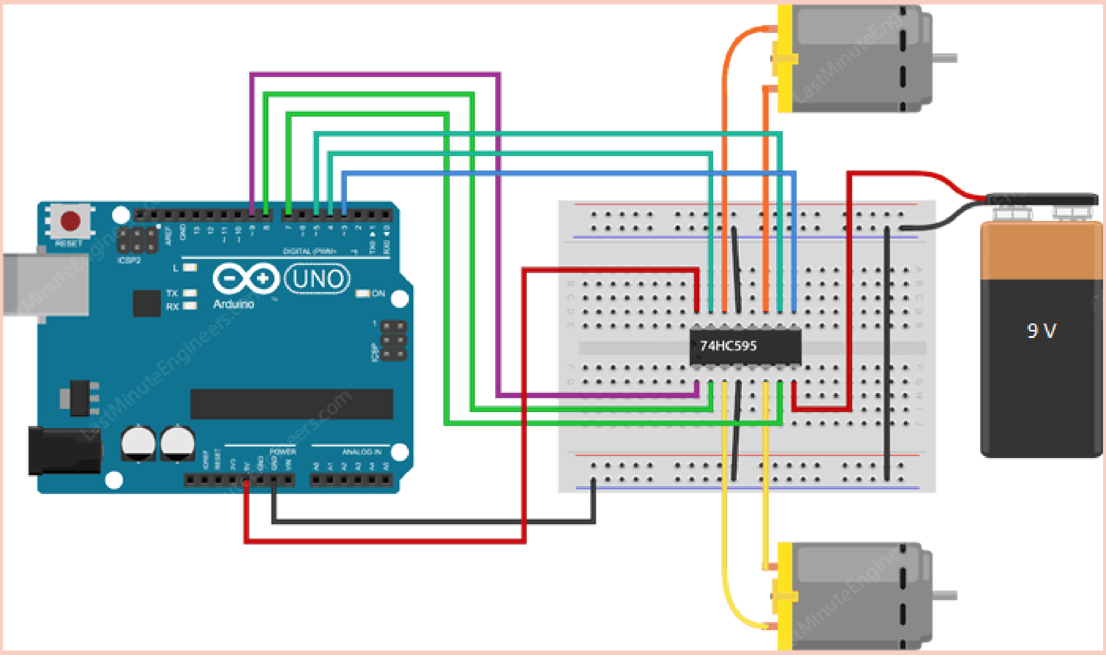
We can achieve this with the help of using a microcontroller, in this case Arduino, and implementing PI, PD and PID algorithms to test the outcomes. But we face certain technical difficulties with the motor, hence moved to the concept of Pulse width modulation to adjust the speed of the motor.

Allen Ben Philipose | 18BIS0043
Shayon Gupta | 18BIS0154

Block diagram -



Architecture diagram -



Abstract -

Vehicles are essential in the modern day. It enables people to go from one end of the city to the other and almost all sectors of work benefit from the convenience of this. Vehicles although extremely useful, must be driven with caution as miscalculations can lead to fatal crashes.

It is thus very important to be able to control the speed of the vehicle precisely at a steady rate. Here a closed loop control structure is necessary to make the required adjustments. This system can be readily applied to conventional gas engines and electric motors, especially since electric dc motors have become very sturdy over the years and responsive to quick changes.

Such characteristics are visible not only in the top end but also in the more affordable ranges in the market. It is yet to become common for cheap control setup to accurately have the dc motor to spin at a given speed quickly. Manual acceleration control is still almost used everywhere.

Requirements -

Hardware:

- A microcontroller (Preferably Arduino)
- DC motor (with speed encoder built in)
- Small model car chassis
- 6 V batteries
- L293D shield for Arduino

Software:

- Arduino IDE

MATLAB demonstration -

Code and Output for P:

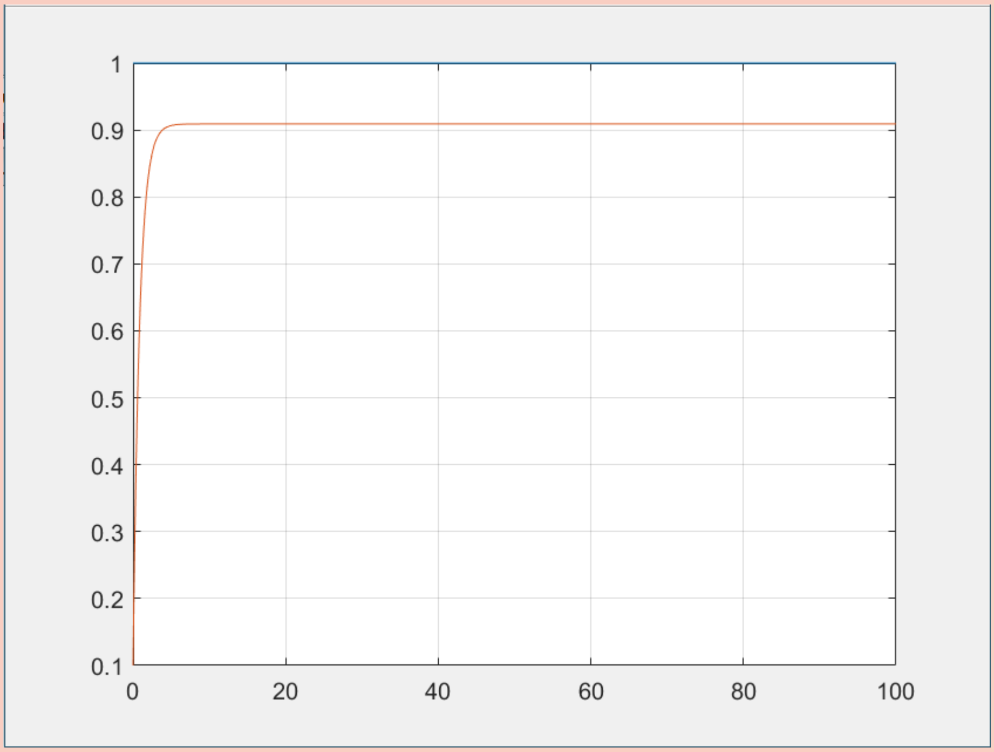
Editor - C:\Users\allen\OneDrive\Documents\MATLAB\P.m

P.m

```matlab
clear all
clc
Kp = .1; %Propotional constant
t = 0:0.1:100;
error = zeros(1,length(t));
setpoint = input('Enter setpoint = ');
processValue = 0; % we're starting from speed
pvArray = zeros(1,length(t)); % to store processValue at every instant
for i = 1:length(t)
    error(i) = setpoint - processValue;
    %calculate the difference of current value with required result
    output = Kp*error(i); %output for that exact moment (iteration)
    frictionLoss = .01*processValue;
    %speed loss due to friction (given as 1% of currentspeed/processValue)
    processValue = processValue + output - frictionLoss;
    %output keeps on getting added to our processValue...
    pvArray(i) = processValue; %stores the processValue at every instant...
end
plot(t,ones(size(t))*setpoint); %plotting the setpoint
hold on
plot(t,pvArray); %plotting the processValue
grid on;
```

Command Window

```
Enter setpoint = 1
>>
```



Code and Output for PI:

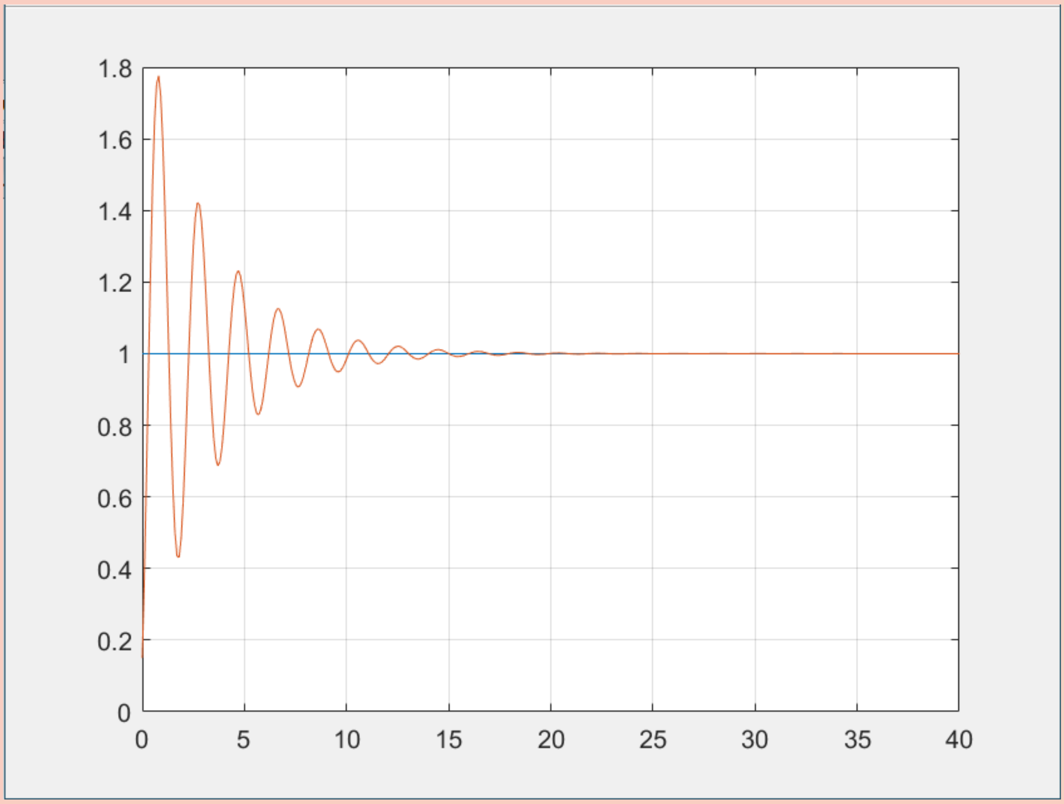
Editor - C:\Users\allen\OneDrive\Documents\MATLAB\PI.m

P.m | PI.m | +

```
clear all
clc
Kp = .05; %Propotional constant
Ki = .1; %Integral constant
t = 0:0.1:100;
error = zeros(1,length(t));
setpoint = input('Enter setpoint = ');
processValue = 0;% we're starting from speed
reset = 0;% variable for storing integration value
pvArray = zeros(1,length(t));% to store processValue at every instant
for i = 1:length(t)
    error(i) = setpoint - processValue;
    reset = reset + Ki*error(i);% the error keeps on gettting summated
    output = Kp*error(i) + reset; %output for that exact moment (iteration)
    frictionLoss = .01*processValue;
    processValue = processValue + output - frictionLoss;
    pvArray(i) = processValue; %stores the processValue at every instant...
end
plot(t,ones(size(t))*setpoint); %plotting the setpoint
hold on
plot(t,pvArray); %plotting the processValue
grid on;
xlim([0 40]);
```

Command Window

```
Enter setpoint = 1
fx >>
```



Code and Output for PID:

Editor - C:\Users\allen\OneDrive\Documents\MATLAB\PID.m

P.m | PI.m | PID.m

```
%PID controller
clear all
clc

Kp = .1; %Propotional constant
Ki = .1; %Integral constant
Kd = .1; %Derrivative constant

t = 0:0.1:100;
error = zeros(1,length(t));
delE = zeros(1,length(t));

setpoint = input('Enter setpoint = ');
processValue = 0; % we're starting from speed
reset = 0; % variable for storing integration value
pvArray = zeros(1,length(t)); % to store processValue at every instant

for i = 1:length(t)
    error(i) = setpoint - processValue;
    reset = reset + Ki*error(i); % the error keeps on gettting summated
    if i>1
        delE(i) = error(i)-error(i-1);
    end

```

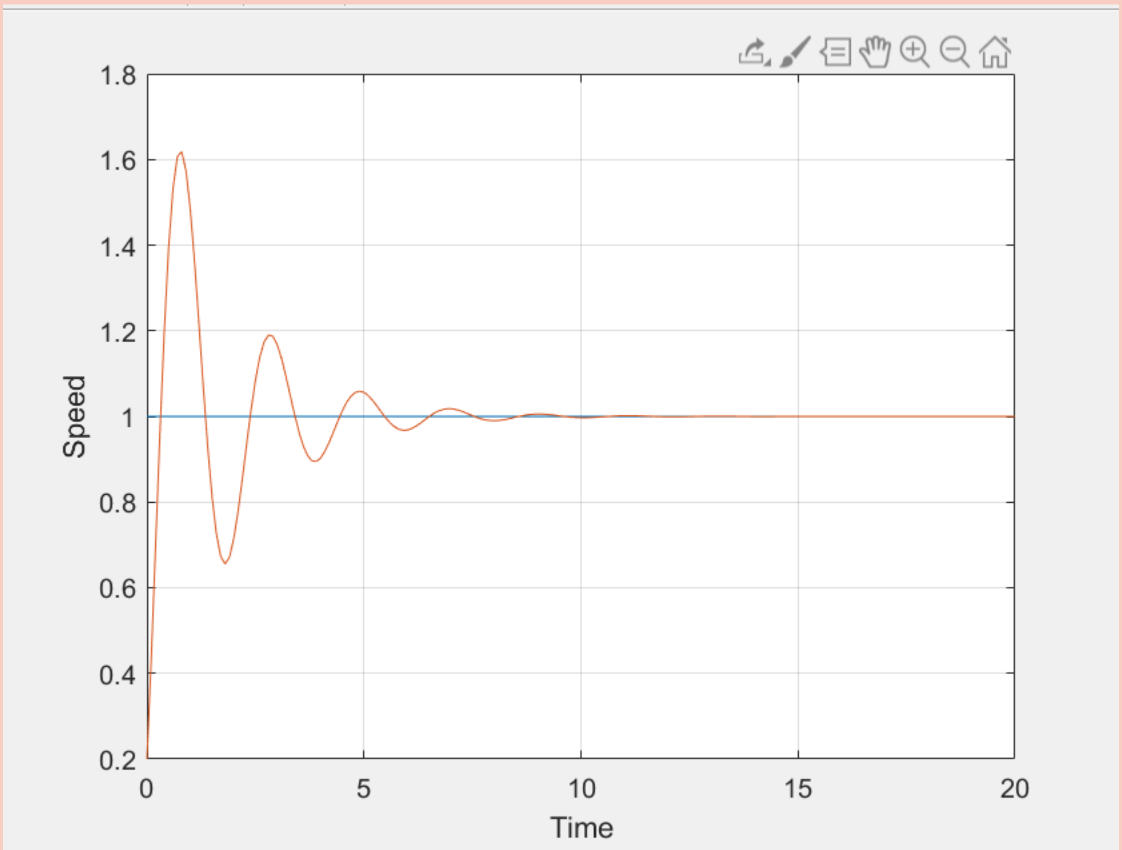
Editor - C:\Users\allen\OneDrive\Documents\MATLAB\PID.m*

P.m | PI.m | PID.m*

```

    output = Kp*error(i) + reset + Kd*delE(i);
    %output for that exact moment (iteration)
    frictionLoss = .01*processValue;
    %speed loss due to friction (given as 1% of currentspeed/processValue)

    processValue = processValue + output - frictionLoss;
    %output keeps on getting added to our processValue...
    pvArray(i) = processValue; %stores the processValue at every instant...
end

plot(t,ones(size(t))*setpoint); %plotting the setpoint
hold on

plot(t,pvArray); %plotting the processValue
xlabel("Time");
ylabel("Speed");

grid on;
xlim([0 20]);



```

Command Window

```
Enter setpoint = 1
fx >>
```



Arduino Code for Pulse Width Modulation

```
int EN1=12;
int EN2=13;
int spe;
char c,prev,s;

void setup() {
  // put your setup code here, to run once:
pinMode(EN1,OUTPUT);
pinMode(EN2,OUTPUT);
pinMode(5,OUTPUT); //left motor
pinMode(6,OUTPUT); //left motor
```

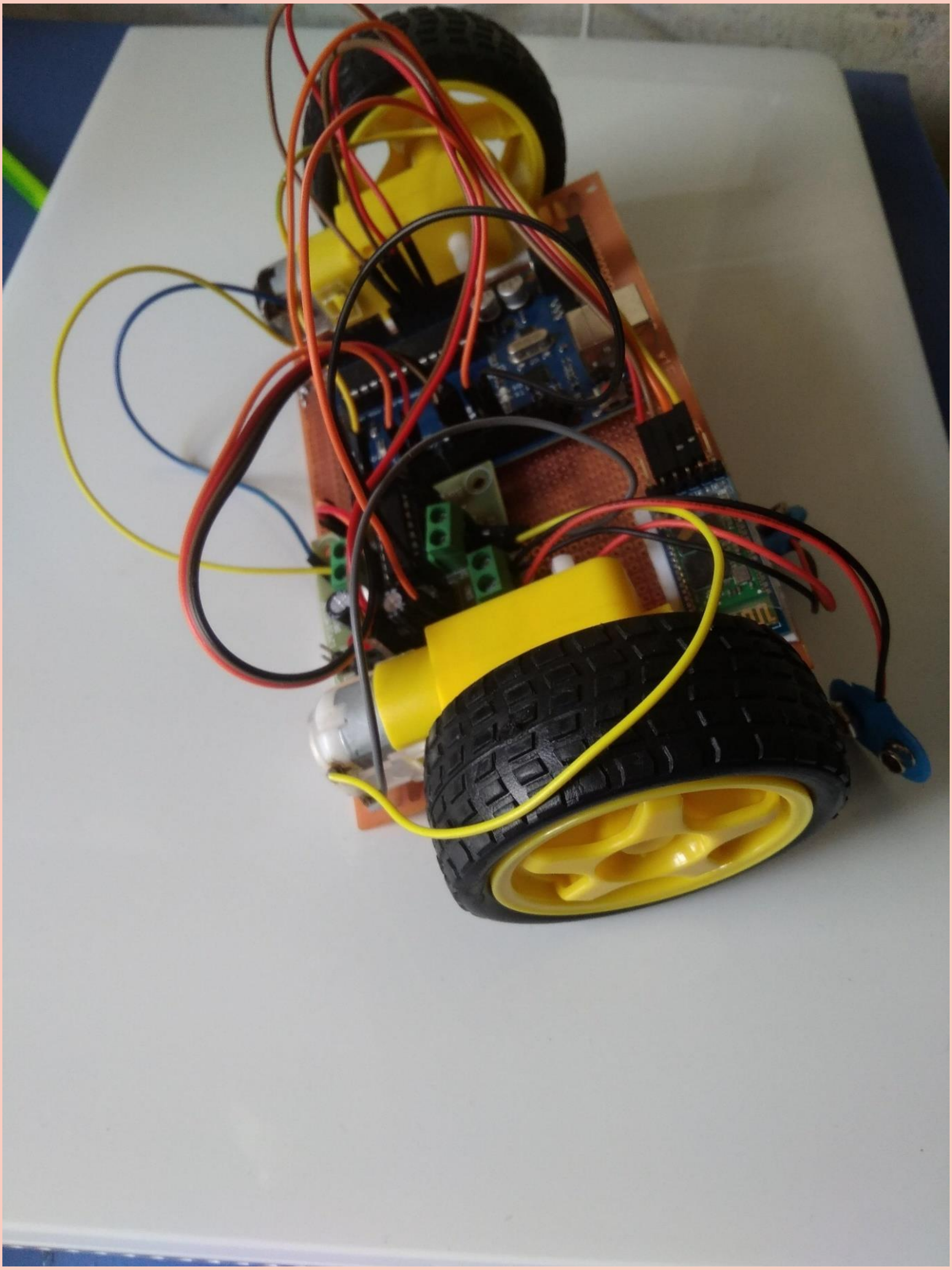
```
pinMode(9,OUTPUT); //right motor

pinMode(10,OUTPUT); //right motor

spe = 255;

c='H';

Serial.begin(9600);

Serial.println("Intitial speed is "+ String(spe));

Serial.println("Key : ");

Serial.println("F - Forward\nL - Left\nR - Right\nB - Reverse");

Serial.println("S - change Speed\nH - Halt");

}


void loop() {

  // put your main code here, to run repeatedly:

digitalWrite(EN1,HIGH);

digitalWrite(EN2,HIGH);

prev=c;

if(Serial.available()){

  c=Serial.read();

  Serial.println(c);

}

if(c=='F'){

  forward();

}

else if(c=='B'){

  reverse();

}

else if(c=='L'){

  left();

}
```

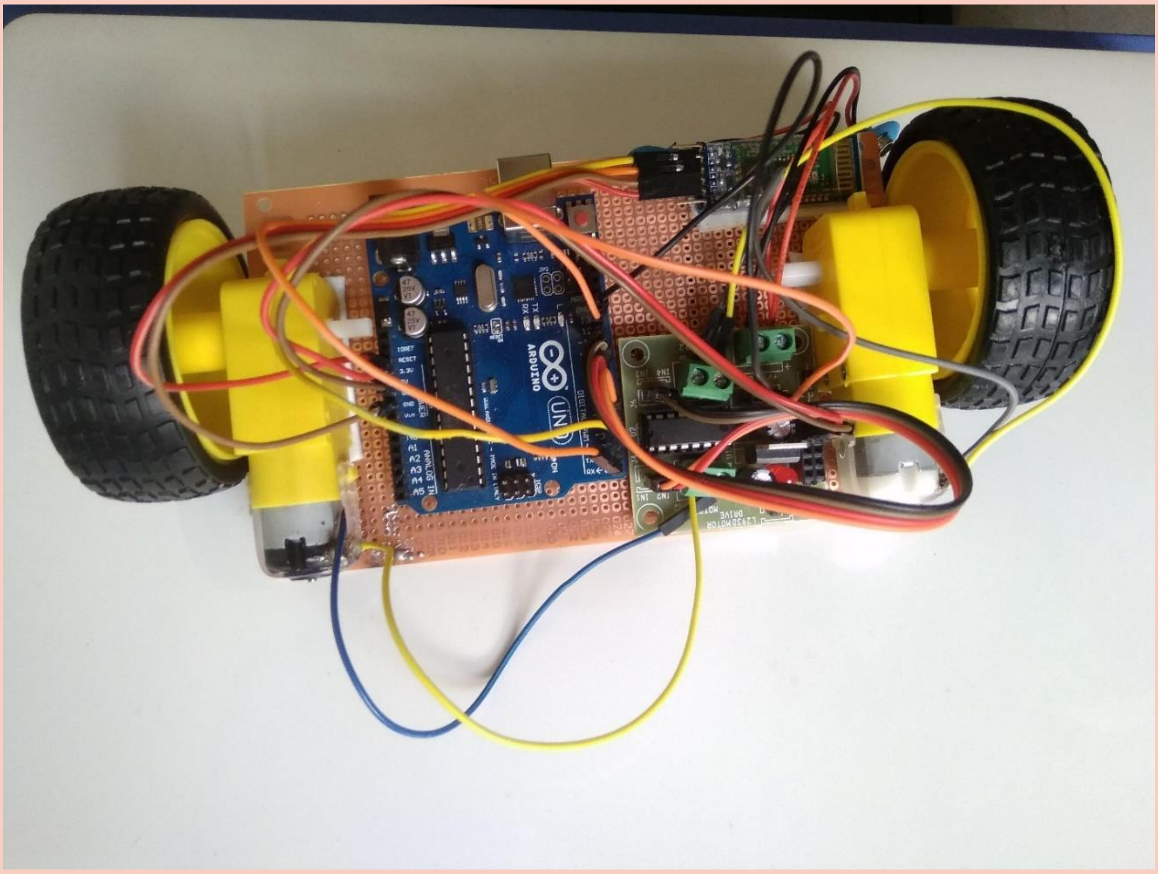
```
else if(c=='R'){
  right();
}
else if(c=='H'){
  halt();
}
else if(c=='S'){
  halt();
  spd();
}
else{
}
delay(100);
}

void forward()
{
  analogWrite(5,spe); // The speed value of the left motor//
  analogWrite(6,0);
  analogWrite(9,spe); //The speed value of the right motor//
  analogWrite(10,0);
}
void reverse()
{
  analogWrite(5,0); // The speed value of the left motor//
  analogWrite(6,spe);
  analogWrite(9,0); //The speed value of the right motor//
  analogWrite(10,spe);
}
```

```
void left()

{

  analogWrite(5,0); // The speed value of the left motor//

  analogWrite(6,0);


  analogWrite(9,spe); //The speed value of the right motor//

  analogWrite(10,0);

}

void right()

{

  analogWrite(5,spe); // The speed value of the left motor//

  analogWrite(6,0);


  analogWrite(9,0); //The speed value of the right motor//

  analogWrite(10,0);

}

void halt()

{

  analogWrite(5,0); // The speed value of the left motor//

  analogWrite(6,0);


  analogWrite(9,0); //The speed value of the right motor//

  analogWrite(10,0);

}

void spd()

{

  c=prev;

  Serial.println("Enter required speed: "); //for reading int values. default is for ascii and will cause
problems
```

```
  Serial.println("H - Highspeed\nM - Moderate speed\nL - low speed");
  while(!Serial.available()){
  }

  s=Serial.read();
  bool r=true;
  while(r){
    s=Serial.read();
    if(s=='H'){
      spe=255;
      Serial.println("Speed has been changed to"+String(spe));
      r=false;
    }
    else if(s=='M'){
      spe=220;
      Serial.println("Speed has been changed to"+String(spe));
      r=false;
    }
    else if(s=='L'){
      spe=195;
      Serial.println("Speed has been changed to"+String(spe));
      r=false;
    }
    else{
    }
    delay(100);
  }
}
```

Working model Images -





Existing problems -

We are not able to simulate the disturbance that a car would normally face while it is travelling at a certain speed. We are taking the speed loss due to friction to be 1% of the car's speed which is far from what we get in the real world. We are still not able to produce accurate results with our software as a result of this.

Literature Cited -

~ https://www.ijser.org/researchpaper/Modeling-and-simulation-of-P-PI-and-PID-controller-for-speed-control-of-DC-Motor-Drive.pdf

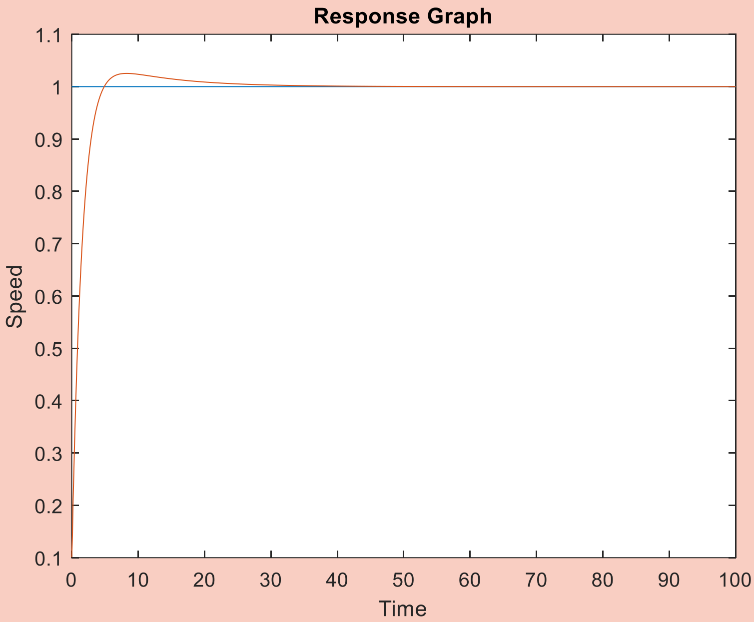
~ https://pdfs.semanticscholar.org/f1dd/aca7abcaaa74820bb63682b764cf82d9cab9.pdf

~ https://www.ijedr.org/papers/IJEDR1402230.pdf

Outcome -

Proper understanding of how different controllers work for acceleration of a car has been attained. All required components have been assembled and multiple outputs have been tested to find the best available option

It is observed that the P type is somehow not behaving as shown in some reference material although PI appears to be working as shown.



Different cases were simulated using MATLAB for P, PI and PID controllers. Standard retardation of 10% of the current speed was maintained.

The given graph was generated for kd = .5 and kp = .01. One observation was that high values of 'kp' increased overshoot.

Compiled and developed programs and arranged hardware four methods of achieving speed control, P, PI, PID and PWM, among which PID is theoretically the best method, but due to its requirement for motors with higher quality and cost, we decided to go for PWM which is the best considering its comparatively inexpensive implementation.

Result –

The model is successfully working using the PWM method and can be used to move in the required speed and direction given by the user.