Position Control of DC Motor

Using Arduino Uno and H-Bridge

Abstract:

The objective of this project was to implement a position control system for a DC motor using an Arduino Uno microcontroller and an H-bridge. The project involved designing a closed-loop PID controller to regulate the motor's position accurately. This report provides a detailed overview of the project, including the theory, methodology, implementation, and evaluation of the control system.

Table of Contents:

Implementation

4.1 Arduino Code

4.2 Explanation of Code

1 PID Controller Implementation

2 H-Bridge Control

Modelling

Conclusion

Introduction:

1.1 Background:

The control of DC motors is a crucial aspect of many industrial applications, such as robotics, automation, and precision positioning systems. By accurately controlling the motor's position, speed, and torque, various tasks can be accomplished effectively. This project focuses on implementing a position control system for a DC motor using an Arduino Uno microcontroller.

1.2 Objectives:

The main objectives of this project were as follows:

To design a closed-loop PID controller for position control.

To develop a hardware setup using Arduino Uno and H-bridge.

To implement motor encoder interfacing for position feedback.

To control the DC motor's position accurately.

1.3 Scope:

The scope of this project is limited to the position control of a DC motor using a closed-loop PID controller. The system utilizes an Arduino Uno microcontroller for signal processing and an H-bridge module for motor control. The project aims to achieve precise positioning within the specified operating voltage and RPM range.

Theory:

2.1 DC Motor Operation:

A DC motor converts electrical energy into mechanical rotational motion. It consists of a stator (field) and a rotor (armature). When a current is passed through the stator windings, a magnetic field is generated, which interacts with the armature's magnetic field, resulting in rotational movement. The motor's speed and torque are controlled by varying the applied voltage and current.

2.2 PID Control:

PID control is a widely used technique for feedback control systems. It stands for Proportional, Integral, and Derivative control. The PID controller continuously adjusts the control signal based on the error between the desired and actual system states.

Proportional (P) control provides a control signal proportional to the error. It helps in reducing steady-state errors but may result in overshoot and oscillations.

Integral (I) control integrates the error over time, compensating for steady-state errors. It eliminates offset but may introduce instability if not properly tuned.

Derivative (D) control calculates the rate of change of the error. It helps in reducing overshoot and stabilizing the system response.

2.3 H-Bridge Module:

An H-bridge module is a circuit that allows bidirectional control of DC motors. It consists of four switches that can be turned on or off to control the motor's direction and speed. By properly switching the switches, the motor can rotate in both clockwise and counterclockwise directions.

Methodology:

3.1 Hardware Setup:

The hardware setup for this project includes the following components:

1) Arduino Uno microcontroller
2) H-bridge module
3) DC motor (12V, 1000 RPM)
4) Motor encoder
5) Power supply(5V)

The H-bridge module is connected to the Arduino Uno to control the motor's direction and speed. The motor encoder is interfaced with the Arduino to provide position feedback. The power supply is connected to provide the necessary voltage for motor operation.

3.2 Software Development:

The software development for this project involves programming the Arduino Uno microcontroller using the Arduino IDE. The code includes the PID controller implementation, motor encoder interface, and H-bridge control logic. The PID controller calculates the control

signal based on the error between the desired and actual positions. The H-bridge control logic sets the appropriate switches to control the motor's direction and speed.

3.3 Closed-Loop Control Design:

The closed-loop control system design involves the following steps:

Reading the target position value from the user via serial communication.

Scaling the target position value to match the motor's operating range.

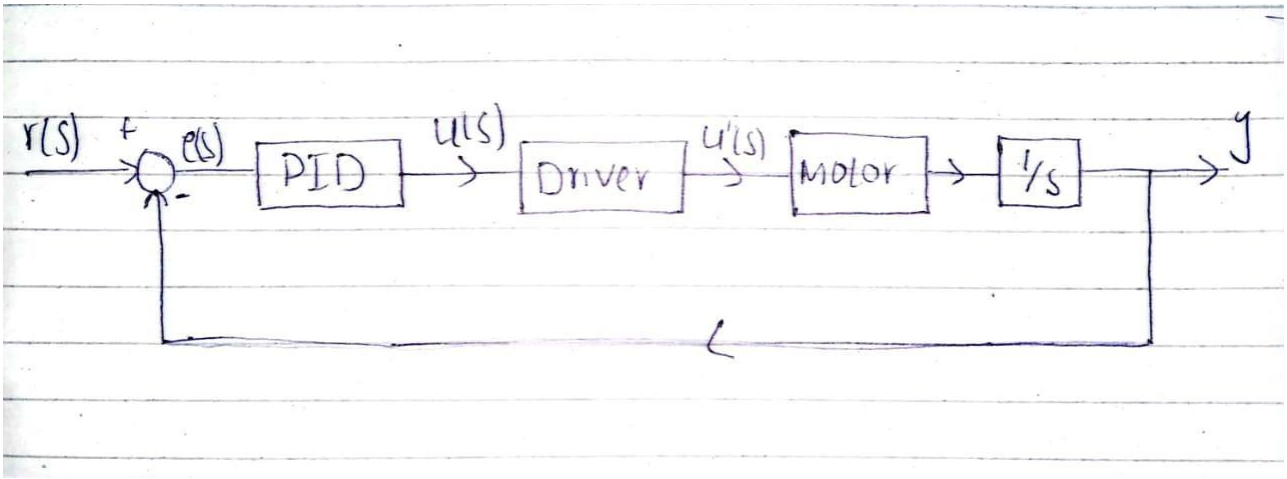
Calculating the error between the target position and the motor's current position using the encoder feedback.

Updating the control signal using the PID controller.

Mapping the control signal to the PWM range and applying it to the H-bridge module.

Monitoring the motor's position and adjusting the control signal accordingly.

Repeating the control loop to continuously update the motor's position.



Implementation:

4.1 Arduino Code:

The Arduino code provided below demonstrates the implementation of the position control system using the PID controller and H-bridge module.

```
//define pins for motor encoder
const int encoder1=2; const int
encoder2=4; int
encoder1_value=0;
//*********************************
                                                //define pins for H bridge
const byte PWMpin=10; const int PWMpin1=8; const int PWMpin2=7;
                                                //Initialization
int     PWM_value=0;     int     PWM_value1=0;     int     direc=0;
//-1=ccw, 1=cw

                                                  //target     values
float targetposition=0;                             //measured values
float motorposition=0; float newValue=0;                                  //
User entered Value


float controlsignal=0;
float proportional1=623.223;
float integral1=5692.35;            float
derivative1=8.277;

                                                //PID related definitions
float previoustime=0; float previouserror=0; float errorintegral=0; float
currenttime=0; float deltatime=0; float errorvalue=0;
float edot=0;
////////////////////////////////////////////
////////////////////////////////////////////

void setup()
{
  Serial.begin(115200);

pinMode(encoder1,INPUT);
```

```
pinMode(encoder2,INPUT);
pinMode(PWMpin,OUTPUT);
pinMode(PWMpin1,OUTPUT);
pinMode(PWMpin2,OUTPUT);


                                                                                //i
nterrupt function   attachInterrupt(digitalPinToInterrupt(encoder1),
checkencoder, RISING);
}
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

void loop() {    if
(Serial.available() > 0)
  {
    // Read the incoming value from the serial port
newValue = Serial.parseFloat();
        Serial.print("New target position set: ");
        Serial.println(newValue);

newValue=newValue/380;
if(newValue>0)
        {
        newValue=int(0.2006*pow(newValue,4)-
3.613*pow(newValue,3)+23.46*pow(newValue,2)+45.55*newValue-
2.299);        //linear equation developed usto match the error
}         else if(newValue <0)
        {
          newValue=int(-0.2006*pow(newValue,4)-
3.613*pow(newValue,3)23.46*pow(newValue,2)+45.55*newValue+2.299)-4;
        }      if
(newValue != 0.0)
    {       targetposition =
fabs(newValue);
    }
    else
    {
      Serial.println("Invalid input. Please enter a valid Position: ");
    }
  }    calculatePID();              //Function to calculate
PID   drivemotor();               //Function to control H-
bridge
}
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//I nterrupt function
 void
checkencoder()
```

```
{
encoder1_value=digitalRead(encoder1);
if(newValue>0)  //CW
  {
motorposition++;
  }    else
if(newValue<0)
  {     motorposition--;
//ccw
  }
}
////////////////////////////////////////////////////////////////////////////
/////
////////////////////////////////////////////////////////////////////////////
////

void calculatePID()
{
currenttime=micros();
 deltatime=(currenttime-previoustime)/1000000.00;
previoustime=currenttime;

if(newValue>0)
 {
   errorvalue=targetposition-motorposition;
 }    else
if(newValue<0)
 {    errorvalue=-motorposition-
targetposition;
 }
 edot=(errorvalue-previouserror)/deltatime;                    //derivative
 errorintegral=errorintegral+(errorvalue*deltatime);           //integral

 previouserror=errorvalue;
   controlsignal=(proportional1*errorvalue)+(derivative1*edot)+(integral1*erro
rintegral);

}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////


void drivemotor()                                              //Function
{
if(newValue>0)
  {     direc=1;   }
else if(newValue<0)
```

```
  {
direc=-1;
  }
else   {
direc=0;
  }

  PWM_value1=(int)fabs(controlsignal);                          //converting
control signal into integer
  PWM_value=map(PWM_value1, 0, 1023, 0,255);


  if(direc==-1 && errorvalue!=0)  //ccw
  {
    digitalWrite(PWMpin1,LOW);
digitalWrite(PWMpin2,HIGH);



  }    else if(direc==1 && errorvalue!=0)
//cw
  {
digitalWrite(PWMpin2,LOW);
digitalWrite(PWMpin1,HIGH);  }

  else  {
digitalWrite(PWMpin1,LOW);
digitalWrite(PWMpin2,LOW);
  }

analogWrite(PWMpin,PWM_value);



if(errorvalue==0)
 {
errorintegral=0;
targetposition=0;
motorposition=0;
previoustime=0;
previouserror=0;
currenttime=0;
deltatime=0;
errorvalue=0;
edot=0;
newValue=0;

 }

}
```

4.2 Explanation of Code:

The code begins by defining and initializing variables and pin modes. The encoder1 and encoder2 pins are specified for motor position feedback from the encoder. The PWMpin, PWMpin1, and PWMpin2 pins are used for controlling the H-bridge module. The proportional1, integral1, and derivative1 variables represent the PID controller's coefficients.

In the setup() function, the serial communication is initialized with a baud rate of 115200. The pin modes for encoder and H-bridge are set, and an interrupt is attached to the encoder1 pin to monitor the motor's position.

In the loop() function, the code checks if there is new data available from the serial communication using Serial.available(). If data is available, it reads the target position value entered by the user using Serial.parseFloat(). This value is then scaled and stored in the newValue variable.

The calculatePID() function is called to compute the control signal based on the error between the target position and the motor's current position. The error is calculated as targetposition - motorposition for a positive target position and -motorposition - targetposition for a negative target position. The time difference (deltatime) between the current and previous calculations is also determined. The proportional, integral, and derivative terms are multiplied by their corresponding coefficients and summed up to calculate the control signal.

The drivemotor() function is responsible for controlling the H-bridge and driving the motor. First, the direction (direc) is determined based on the newValue. If newValue is positive, the direction is set to clockwise (1), and if newValue is negative, the direction is set to counterclockwise (-1). Otherwise, if newValue is zero, the direction is set to stop (0).

The control signal (PWM_value1) is converted to an integer and mapped to the range 0-255 to correspond to the PWM range. The H-bridge switches are then set accordingly to control the motor's rotation. If the direction is counterclockwise (-1) and the error value is not zero, the appropriate switches are set to rotate the motor counterclockwise. Similarly, if the direction is clockwise (1) and the error value is not zero, the switches are set to rotate the motor clockwise. Otherwise, if the direction is stop (0), both switches are turned off to stop the motor. The control signal is applied to the H-bridge by setting the PWM value using analogWrite().

Finally, if the error value reaches zero, indicating that the motor has reached the target position, the variables and parameters are reset to their initial values, allowing the system to handle new target positions.

Modelling of Motor:

To design a controller for motor, it is required that we should know the transfer function of motor. So, for this purpose we didn't have any model of motor. Attempting to find the model of transfer function, we took a input (Voltage) VS output(frequency) data using oscilloscope, then by knowing the gear ratio of 110, we got Voltage vs Speed data.

Voltage	Speed
1	31.8
1.5	75
2	121.2
2.5	165
3	210
3.5	260.4
4	309.6
4.5	360
5	402
5.5	468
6	513.6
6.5	552
7	595.2
7.5	645
8	684
8.5	735
9	783
9.5	831
10	877.2
10.5	924
11	974.4
11.5	1011
12	1056

By getting this data we tried MATLAB command to estimate the transfer function, but there was another additional requirement of sampling rate of input and out data. So, to find the sampling rate, we modelled the above data in a curve fitting tool and got a polynomial equation. We implemented that polynomial equation in Simulink. Our input was Ramp at that case, and extract again input vs output data at the sampling rate of 0.2s.

Voltage	Speed
0.8	10.72
1	29.54
1.2	48.37
1.4	67.19
1.6	86.01
1.8	104.8
2	123.7
2.2	142.5
2.4	161.3
2.6	180.1
2.8	198.9
3	217.8
3.2	236.6
3.4	255.4
3.6	274.2
3.8	293.1
4	311.9
4.2	330.7
4.4	349.5
4.6	368.3
4.8	387.2
5	406
5.2	424.8
5.4	443.6
5.6	462.4
5.8	481.3
6	500.1
6.2	518.9
6.4	537.7
6.6	556.6
6.8	575.4
7	594.2
7.2	613
7.4	631.8
7.6	650.4
7.8	669.5
8	688.3
8.2	707.1
8.4	726
8.6	744.8
8.8	763.6
9	782.4
9.2	801.2
9.4	820.1
9.6	838.9
9.8	857.7
10	876.5
10.2	895.4
10.4	914.2
10.6	933

We used the given data to find the estimate transfer function, using this matlab code

```
clc;
clear all;
close all;
v=[0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3 3.2 3.4 3.6 3.8 4 4.2 4.4 4.6 4.8 5 5.2 5.4 5.6 5.8 6 6.2 6.4 6.6 6.8 7 7.2 7.4 7.6 7.8 8 8.2 8
v=v';

speed=[10.72 29.54 48.37 67.19 86.01 104.8 123.7 142.5 161.3 180.1 198.9 217.8 236.6 255.4 274.2 293.1 311.9 330.7 349.5 368.3 387.2 406 424
speed=speed';

z = iddata(speed,v,0.2);
sys = tfest(z,1)
```

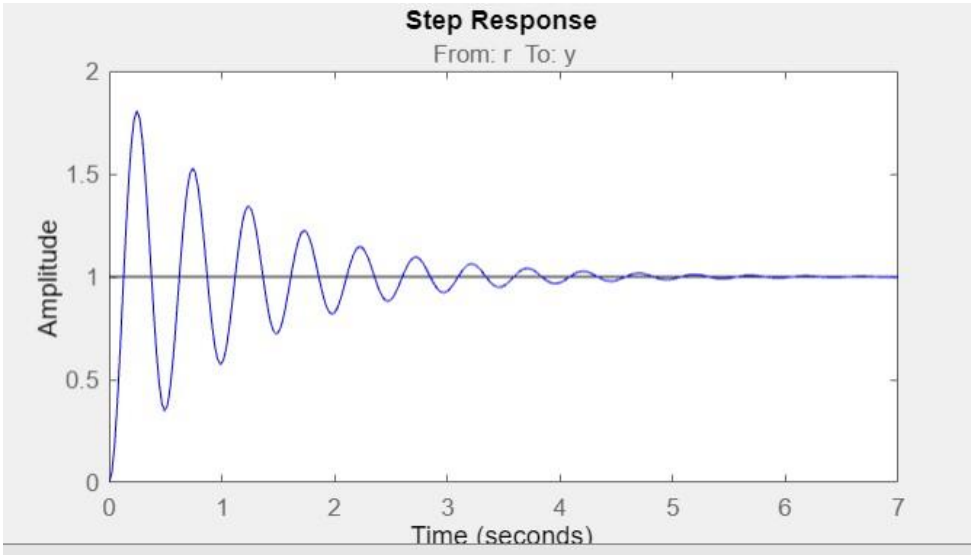
So, at the end this was the estimated transfer function of our model:

```
sys =
  From input "u1" to output "y1":
    162.2
  ---------
  s + 1.723
```

The above transfer function give us the relation between voltage and speed of motor.

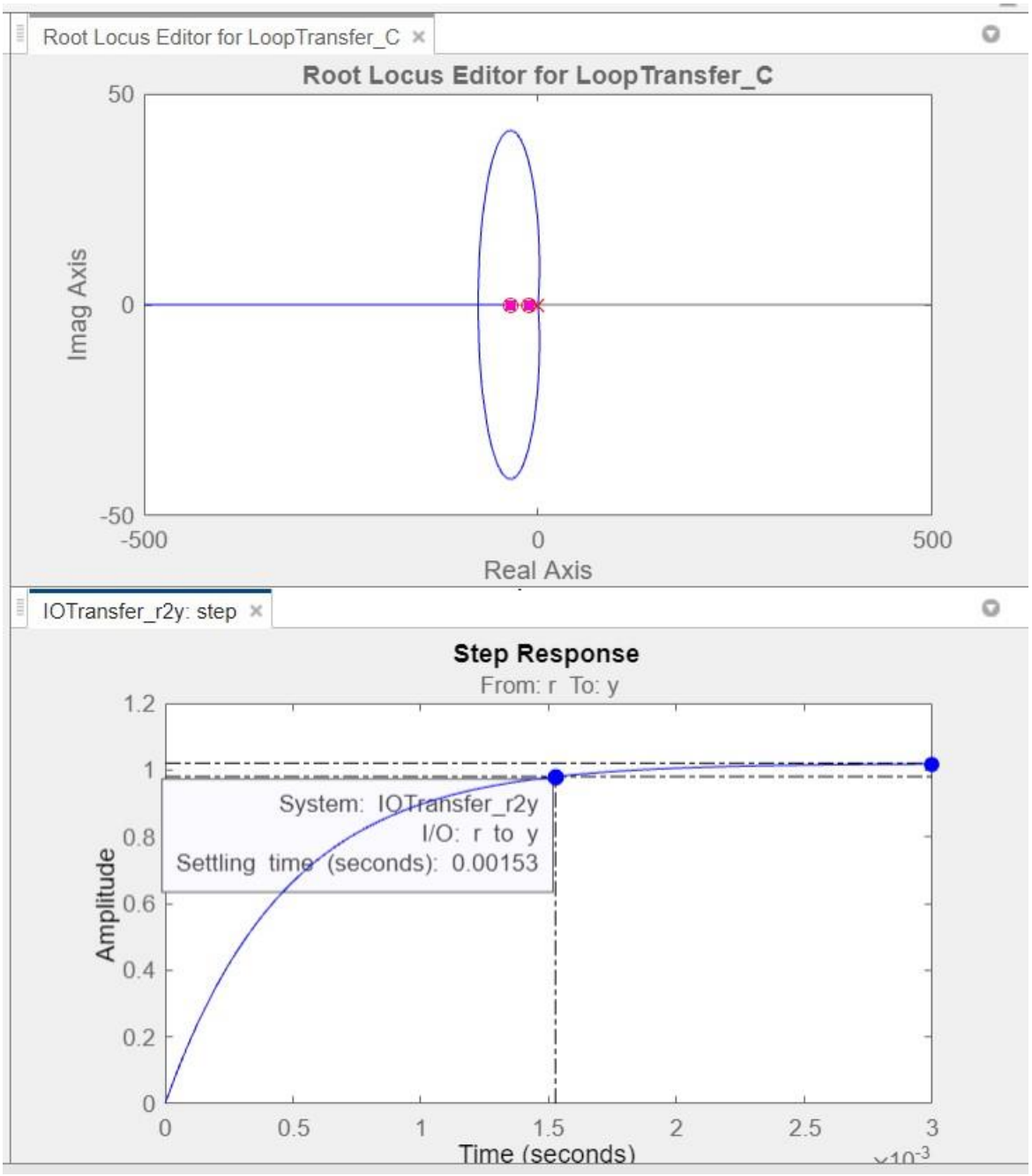
In our case output is position not a speed. So, we have to integrate the above data or multiplying by: $\frac{1}{s}$

$$T(S) = \frac{162.2}{s(s+1.723)}$$ (Relation of position vs Voltage)



So, this was the initial step response of our system:

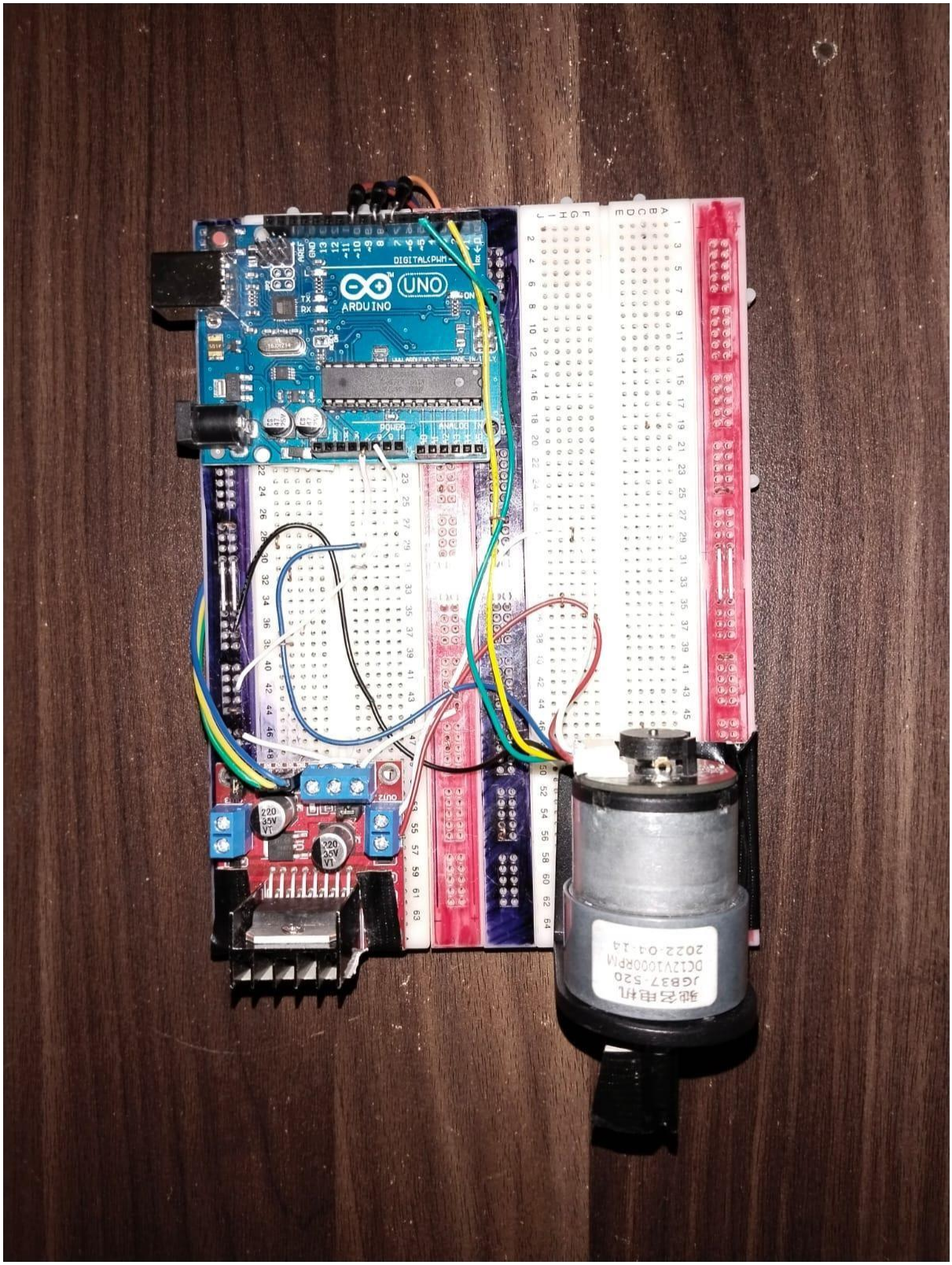
Then we added a PID controller in siso tool. So, after controller our response become:




```
Tunable Block
Name: C
Sample Time: 0
Value:
  101.14 (s+371.9) (s+285.8)
  --------------------------
              s
```

We made possible to remove the overshoot and settling time, lest graph shows the transfer function of PID controller.

HARDWARE PICTURE:



CONCLUSION:

The project successfully implemented a position control system for a DC motor using an Arduino Uno microcontroller and an H-bridge module. The closed-loop PID controller

effectively regulated the motor's position, demonstrating accurate and precise control. The system has potential applications in various fields, including robotics, automation, and mechatronics. Future improvements and enhancements can further enhance the system's performance and expand its capabilities.