PID Control DC motor

ABSTRACT

Implemented PID control on Arduino with an L298N motor driver for precise angular position control of a JGA25-370 Geared Motor. Tested using step, sine, and potentiometer inputs, achieving minimal error oscillations. Addressed orientation change errors with friction compensation for improved stability. The PID controller exhibits high precision, suitable for applications like robotic arm position control in industrial settings.

KEY WORDS

DC-Gear motor, L298 Motor Driver, PID, PWM control.

Contents

List of Figures:

1 INTRODUCTION

DC motors find widespread use in defense, industries, and robotics due to their simplicity, reliability, and cost-effectiveness. While speed control is commonly achieved by varying terminal voltage, precise shaft position control is essential for applications requiring accuracy. Microcontrollers offer an efficient solution for DC motor control, especially in position control systems.

This paper focuses on a JGA25-370 Geared Motor with an integrated quadrature encoder, providing pulse counts per revolution of the gearbox's output shaft. Geared DC motors, with their ability to provide high torque, are utilized for high torque position control and low-speed applications. An L298 H-Bridge module, driven by an Arduino microcontroller, enables bi-directional motor control through pulse-width modulation (PWM). PID tuning algorithms calculate the PWM control signal, allowing the PID controller to correct errors in the motor's measured position and adjust accordingly.

The study explores real-time DC motor position control with various input signals, including step and sine functions and a variable resistor. To enhance performance, friction compensation is integrated based on the velocity-dependent Coulomb's friction force. The results demonstrate effective correction of errors, showcasing the PID-controlled DC motor's ability to achieve and maintain desired positions or speeds in real-time applications.

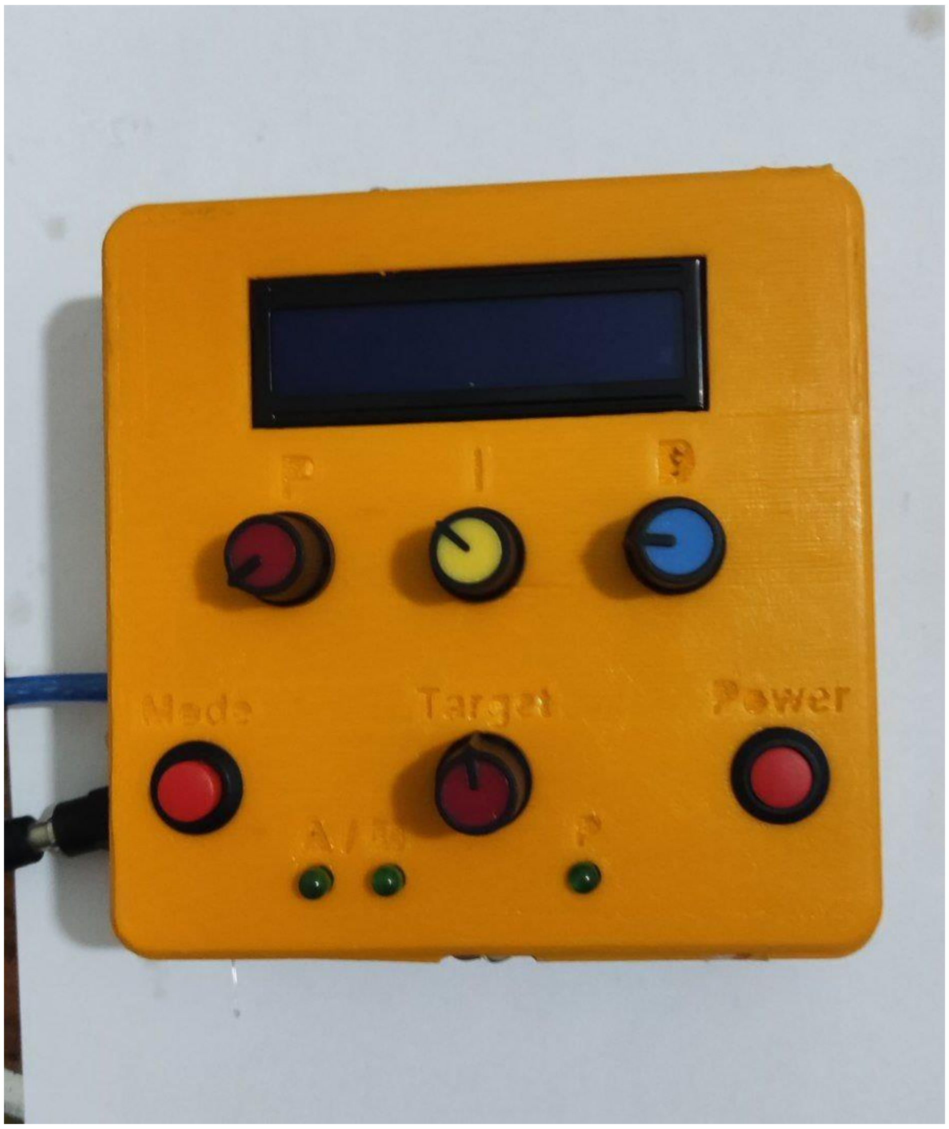


Figure 1 Implemented Model

2 SYSTEM BLOCK DESCRIPTION

The block diagram for overall system description is shown in Fig.1. In this block, three main hardware components are implemented for this research. Arduino Uno microcontroller is to control the position of DC motor by controlling the input voltage to the motor.

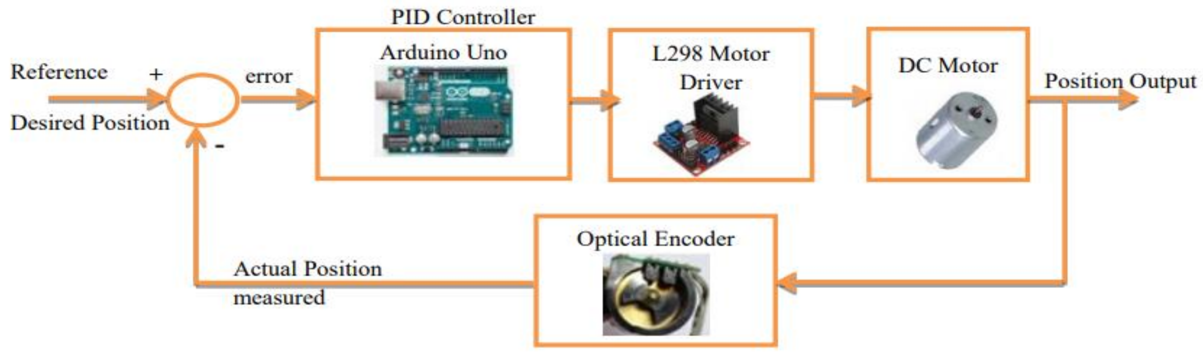


Figure 2Block Diagram for DC Motor Position Control

PID tuning algorithms is implemented in microcontroller to execute the PWM signal for DC motor drive. A JGA25-370 Geared Motor is a powerful motor to drive the position control system. It comes with the photoelectric encoder output, planetary gear reducer and 80:1 gear ratio, which provide 120 rpm with 12VDC rated voltage. L298 dual H-Bridge motor driver which is allows controlling the direction and position of DC motor.

3 HARDWARE DEVICES

Arduino Uno Microcontroller Arduino Uno is a microcontroller board based on the ATmega328P shown in Fig. 2. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started [3]. In this paper, the Arduino microcontroller is very well-suited to drive the PWM signal for DC motor for the improvement of the output response for the DC motor position control system.

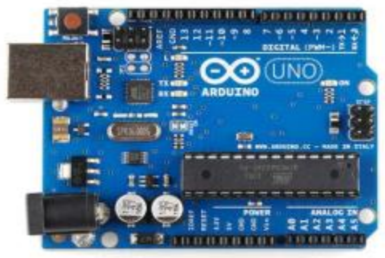
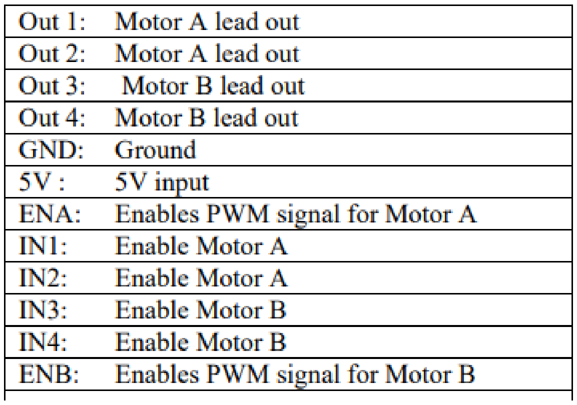


Figure 3Arduino

L298N Dual H-Bridge Controller The L298N H-bridge IC shows in Fig. 3 that can allow to control the speed and direction of two DC motors. This module can be used with motors that have a voltage of between 5 and 35V DC with a peak current up to 2A. The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5Vpin which can either be an input or output. Pin assignments for L298N dual H-Bridge Module is shown in table 1. The digital pin assign from HIGH to LOW or LOW to HIGH is used IN1and IN2 on the L298N board to control the direction. And the controller output PWM signal is send to ENA or ENB to control the position. The forward and reverse speed or position controlling for the motor has done by using PWM signal . Then using the analog Write() function and send the PWM signal to the Enable pin of the L298N board, which actually drives the motor.



Out 1:	Motor A lead out
Out 2:	Motor A lead out
Out 3:	Motor B lead out
Out 4:	Motor B lead out
GND:	Ground
5V :	5V input
ENA:	Enables PWM signal for Motor A
IN1:	Enable Motor A
IN2:	Enable Motor A
IN3:	Enable Motor B
IN4:	Enable Motor B
ENB:	Enables PWM signal for Motor B

Figure 4Pns assignment for L298

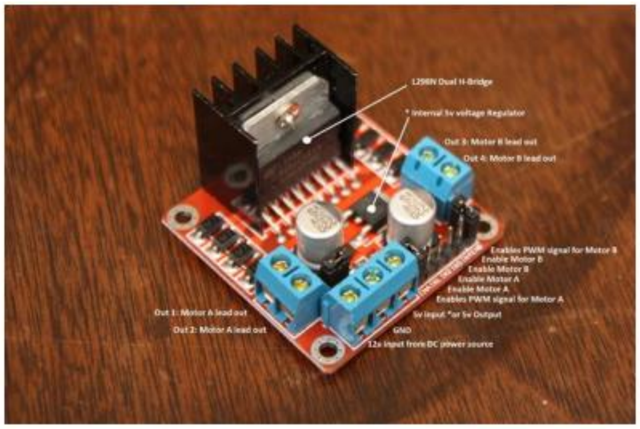
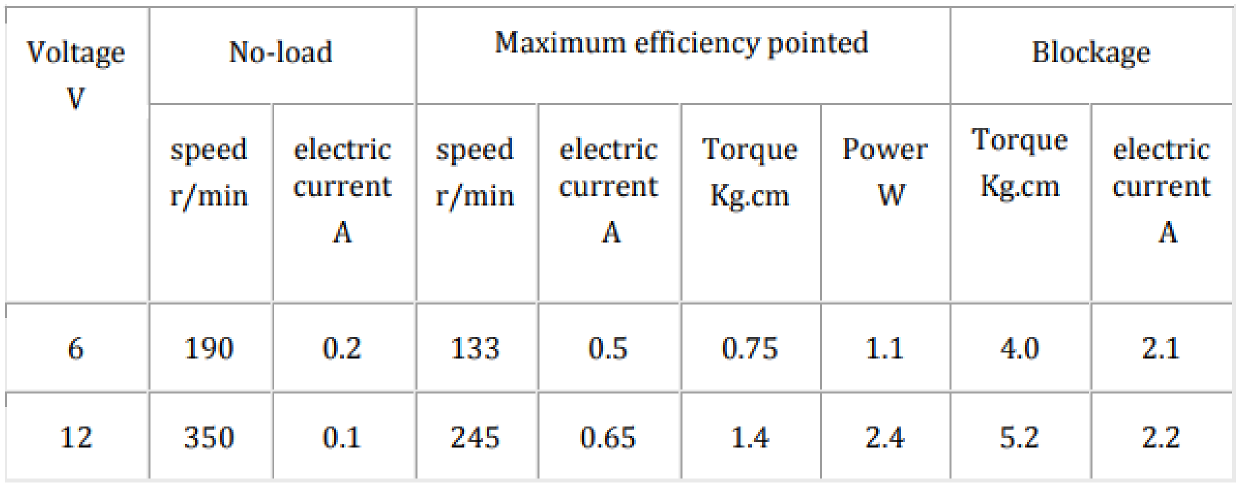


Figure 5Motor Driver

DC Geared Motor

The 12V DC gear-motor shown in Fig. 4 is a powerful motor to drive the position control system. It comes with the photoelectric encoder output and planetary gear ratio reduced by 80:1 gear. It can provide 120 rpm with the rated voltage of 12VDC . To read the count values from the encoder, the user would check the condition of channel A and B rotation applying by experiment. For the rotor shaft count per revolution values, it is very important to multiply the gear ratio by count values. The specification of the motor is shown in Table 1



Voltage V	No-load		Maximum efficiency pointed				Blockage	
	speed r/min	electric current A	speed r/min	electric current A	Torque Kg.cm	Power W	Torque Kg.cm	electric current A
6	190	0.2	133	0.5	0.75	1.1	4.0	2.1
12	350	0.1	245	0.65	1.4	2.4	5.2	2.2

Table 1 Specification of JGA25-370 Geared Motor

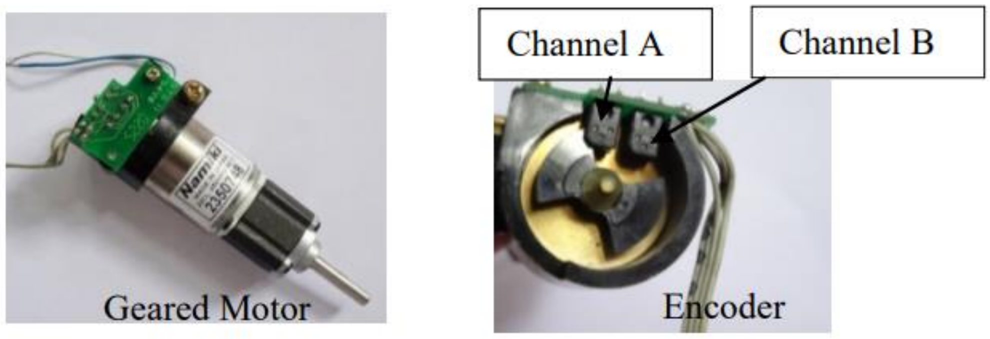


Figure 6DC motor with Encoder

Fig. 5 shows the encoder circuit for JGA25-370 Geared Motor. It is needed to connect two 10k Ohms to channel A and channel B supplied by 5V. One of the encoder pins is connected with 120 Ohms resistor and another pin ground.

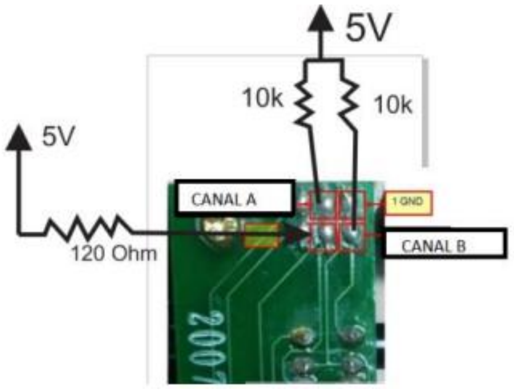


Figure 7Encoder Circuit

Reading the Encoder Pulse In this research work, the encoder comes with the photoelectric encoder output (channel A and channel B). It is used to sense the rotation of a magnetic disk on a rear protrusion of the motor shaft forward or backward direction according to the pulse count per revolution. From Fig. 6, it is the clockwise direction from channel A leading to channel B because the A interrupt seeing

attached to B whenever it changes state LOW to HIGH or HIGH to LOW condition. The another direction is counterclockwise caused by channel B leading to channel A because the B interrupt seeing attached to A whenever it changes state LOW to HIGH or HIGH to LOW condition.

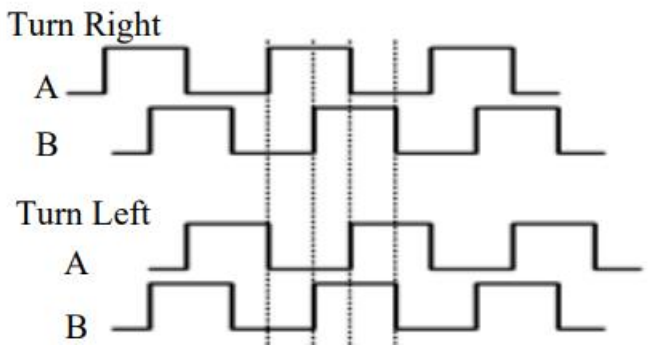


Figure 8Endoder Channel output

PID Tuning Algorithms A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism widely used in

industrial control systems. A PID controller calculates an "error" value as the difference between a measured process variable and a desired set point. The controller attempts to minimize the error by adjusting the process control inputs . Fig.7 mentions implementation program flow chart for this setup.

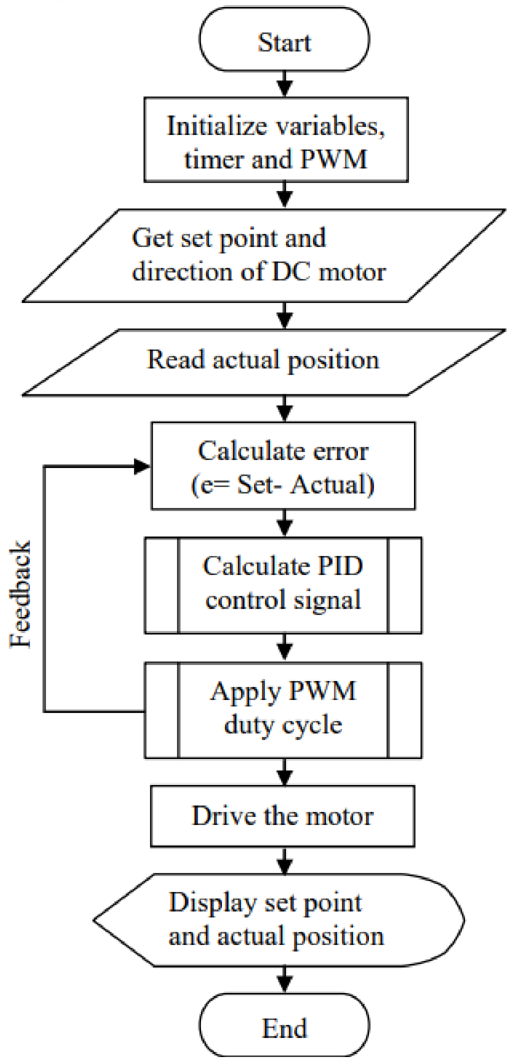

Start
Initialize variables, timer and PWM
Get set point and direction of DC motor
Read actual position
Calculate error (e= Set- Actual)
Calculate PID control signal
Apply PWM duty cycle
Drive the motor
Display set point and actual position
End
Feedback

Figure 9Flowchart

Fig:7, Flowchart for System Description

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

Fig.8 shows the PID controller design for this control system. PID equation is calculated to get the control signal for DC motor. From eq (1), e(t) is position error value difference between set angle and output measured angle(actual angle), u(t) is PWM signal for DC motor and y(t) is the actual angle. Kp, Ki and Kd were respectively with the values of proportion, integral, and the differential coefficient.

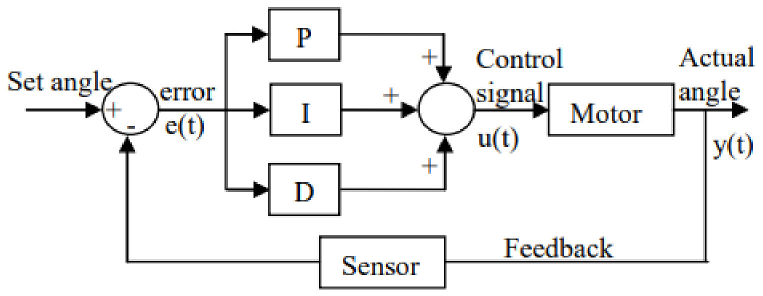


Figure 10PID Control

Fig:8, Closed Loop Position Control of the DC Motor using PID Controller

The programming code for PWM signal execution is calculated by the following procedure. The instruction code is implemented by the Arduino microcontroller to execute the control signal.

```
error           = output - input;
Integral        = xT * error + prv_ui;
Diff            = (error - prv_error) / xT
PWM             = kp * error + ki * Integral +   kd *    Diff;
Usignal         = abs (PWM);
```

From above code, the proportional gain (Kp) examines the magnitude of the error and it responds proportionally. Although it has a large error, the magnitude of the motor position will receive a large response. The integral gain (Ki) efforts to reduce the steady state error. And the derivative gain (Kd) attempts to look at the rate of change of the error signal. Derivative control will decrease the overshoot to become a greater system response curve of the motor position to a rapid rate of change. Finally, the controller is calculated the PWM signal depends on the signal variation of DC motor by using PID gain values to improve the system performance.

Pulse Width Modulation PWM, or pulse width modulation is a technique which allows adjusting the average value of the voltage that's going to the electronic device by turning on and off the power at a fast rate. The average voltage depends on the duty cycle, or the amount of time the signal is ON versus the amount of time the signal is OFF in a single period of time. It is depending on the size of the motor, the user can simply connect an Arduino PWM output to the base of transistor or the gate of a MOSFET and then control the speed or position of the motor by controlling the PWM output . PWM and duty cycle relation diagram is shown in Fig. 9

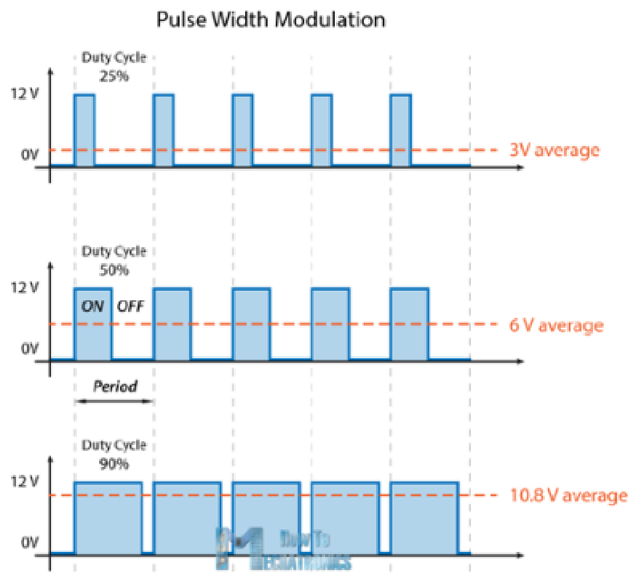


Fig:9, PWM and Duty Cycle Diagram

Figure 11PWM Diagram

In this paper, the value of PWM signal is from 0 to 255 or that's 0 to 100% duty cycle of the PWM signal. The duty cycle diagram is as shown in Fig.10. The PWM value is executed by PID controller to reduce the error difference between the desired angle and the actual angle.

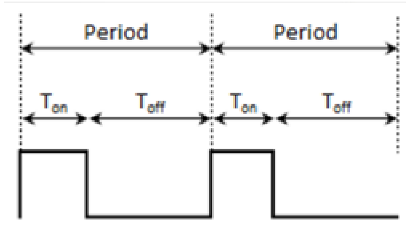


Fig:10, Duty Cycle Diagram

Figure 12Duty Cyle

The duty cycle of PWM signal is calculated by the following equation.

$$\text{Period} = 1/\text{Frequency} \tag{2}$$
$$\text{Period} = T_{on} + T_{off} \tag{3}$$
$$\text{Duty Cycle} = T_{on} / (T_{on} + T_{off}) * 100 \text{ (percentage)} \tag{4}$$

4 System Wiring

Firstly, the Dc motor was connected to A and B connections on the L298N module. In addition, power supply was connected with the positive power supply sign on the module and negative/GND. In this system the power supply as up to 12V, so the 12V jumper was left and 5V was available on the module . Also, Arduino GND is connected to pin 5 on the module as well to complete the circuit. Then six digital output pins on the Arduino were connected to the L298N Dual H Bridge DC Motor Driver, two of which needed to be PWM (pulse-width modulation) pins. PWM pins were denoted by the tilde ("~") next to the pin number as shown in Fig. 11.

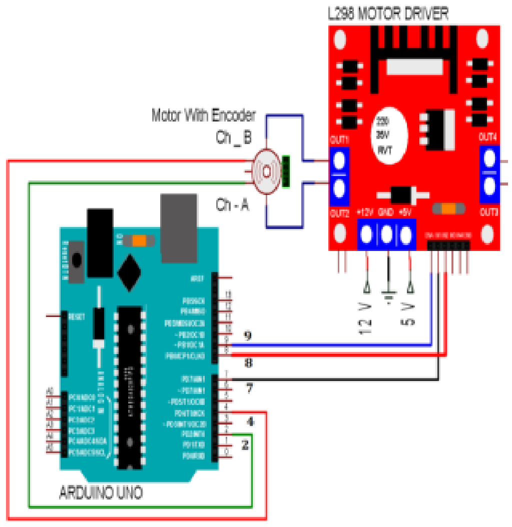


Figure 13wiring D=diagram

Finally, the Arduino digital output pins were connected to the driver module. Digital pins D7, D8 and D9 were connected to pins IN1, IN2 respectively. Then, the En A jumper is removed and D9 is connected to it. The Arduino digital input pins was connected to the Incremental Encoder, channel A with the digital pins D2 and channel B with digital pins D4. The wring diagram of the Arduino and the L298N Dual H-Bridge DC Motor Driver and Incremental Encoder

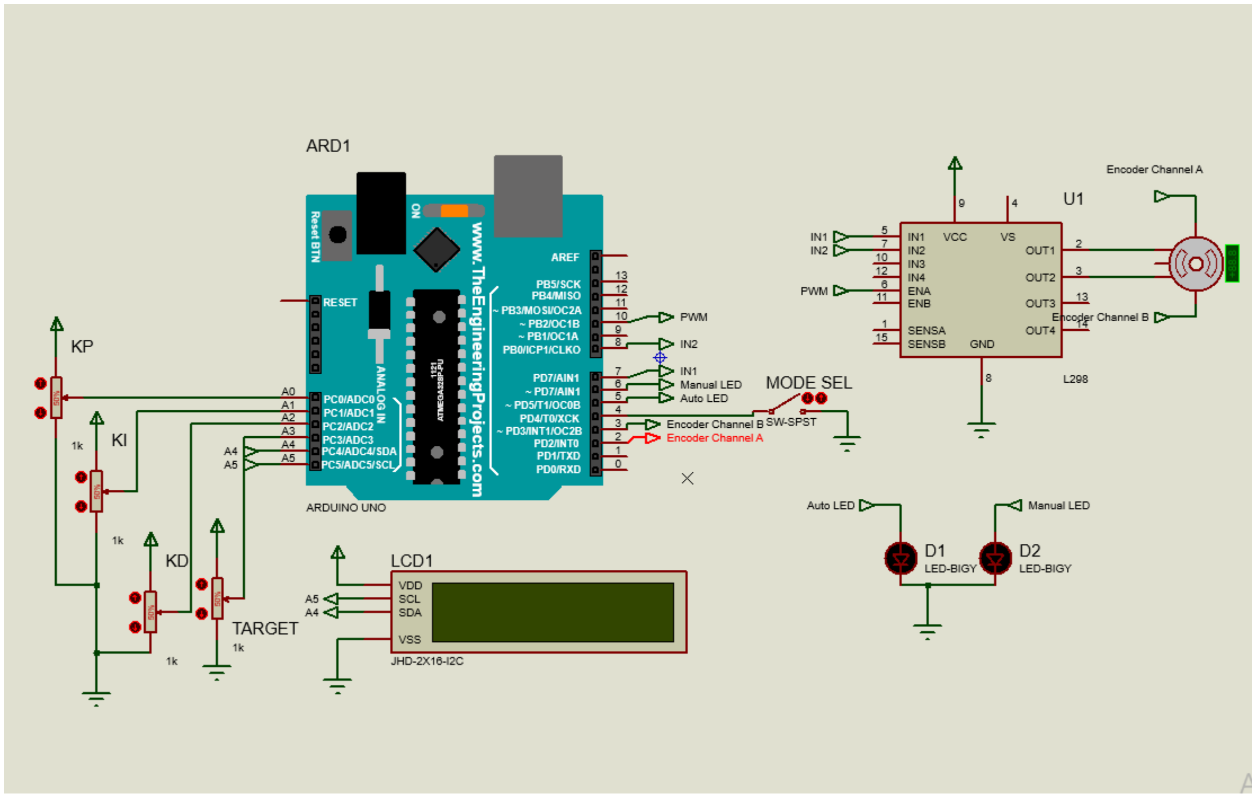


Figure 14 Proteus Schematic

5 MATHEMATICAL MODEL OF DC MOTOR

In armature voltage control scheme for separately excited dc motors, voltage applied to armature is varied without varying the voltage applied to the field. Equivalent model of dc motor is shown in following figure. 12

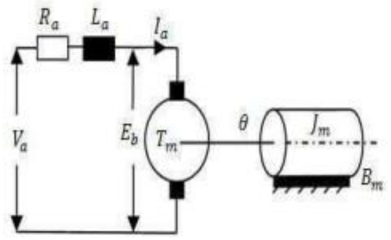


Figure 15 DC motor Model

$$v_a = R_a . i_a(t) + L_a . \frac{di_a(t)}{dt} + e_b(t) \quad (1)$$

$$e_b(t) = K_b . w(t) \quad (2)$$

$$T_m(t) = K_t . i_a(t) \quad (3)$$

$$T_m(t) = J_m . \frac{dw(t)}{dt} + B_m . w(t) \quad (4)$$

Where,

Va = armature voltage (V)

Ra = armature resistance(Ω)

La = armature inductance (H)

Ia = armature current (A)

Eb = back emf (V)

W = angular speed (rad/s)

Tm= motor torque (Nm)

θ = angular position of rotor shaft (rad)

Jm = inertia of rotor (Kg-m2)

Bm = viscous friction coefficient (Nms/rad)

KT = torque constant (N-m/A)

Kb = back emf constant (V/rad)

Let us combine the upper equations together:

$$v_a = R_a . i_a(t) + L_a . \frac{di_a(t)}{dt} + K_b . w(t) \quad (5)$$

$$K_t . i_a(t) = J_m . \frac{dw(t)}{dt} + B_m . w(t) \quad (6)$$

Laplace transforms of (5) and (6) are

$$V_a(s) = R_a(s) . I_a(s) + L_a . I_a(s) . s + K_b . W(s) \quad (7)$$

$$K_t . I_a(s) = J_m . W(s) . s + B_m . W(s) \quad (8)$$

If current is obtained from (8) and substituted in (7) we have

$$V_a(s) = W(s) . \frac{1}{K_t} . [L_a . J_m . s^2 + (R_a . J_m + L_a B_m) . s + (R_a . B_m + K_b . K_t)]$$

(9)

Then the transfer function which relates rotor speed and applied armature voltage is given as:

$$\frac{W(s)}{V_a(s)} = \frac{K_t}{[L_a . J_m . s^2 + (R_a . J_m + L_a . B_m) . s + (R_a . B_m + K_b . K_t)]} \quad (10)$$

Then the transfer function between shaft position and armature voltage at no-load is:

$$\frac{\theta(s)}{V_a(s)} = \frac{K_t}{[L_a . J_m . s^3 + (R_a . J_m + L_a B_m) . s^2 + (R_a . B_m + K_b . K_t) . s]}$$

6 Block Diagram For DC Motor

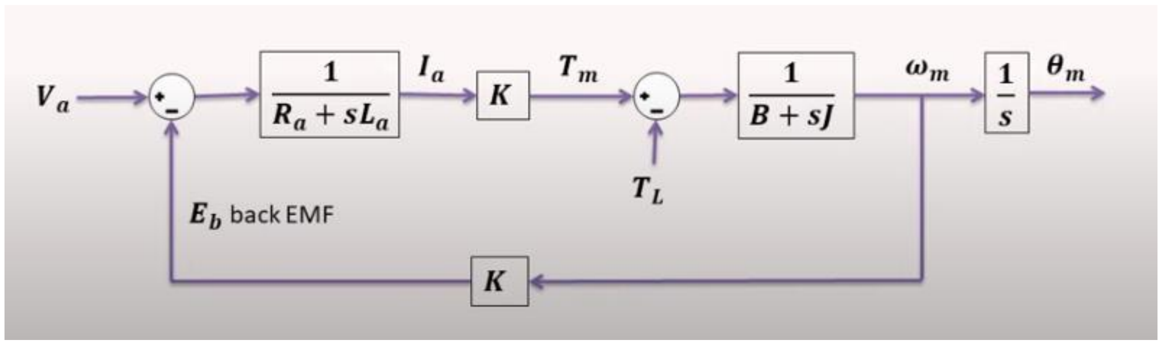


Figure 16DC motor Block diagram

Va = 12 ; % armature input voltage in volts
Ra = 0.4 ;   % armature resistance in ohms
La = 0.1 ;   % armature inductance in henry
B = 0.02;   % damping coefficient
J   = 7   ;   % motor body inertia
T1 = 15 ;   % load torque in Nm
K = 1.4 ; % Back EMF constant

7 Simulink Model

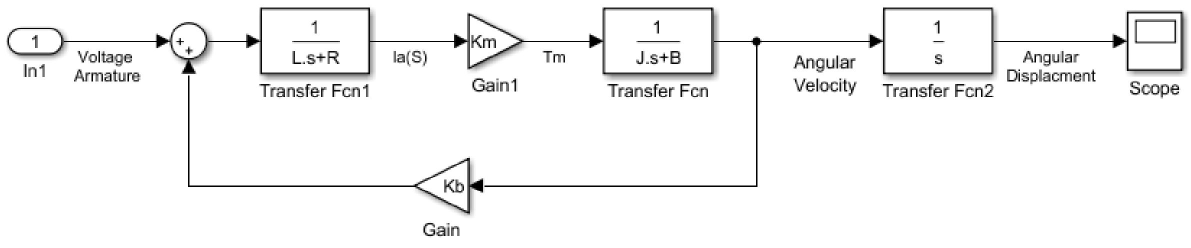


Figure 17 DC motor Modeling

Here is a matlab script to change Block diagram parameters automatically:

```
BL1 = '[0.1 0.4]';

BL2 = '[7 0.02]';

G1 = '1.4';

G2 = G1;

set_param('DcModeling/BL1','Denominator',BL1);

set_param('DcModeling/BL1','Denominator',BL1);

set_param('DcModeling/BL2','Denominator',BL2);

set_param('DcModeling/G1','Gain',G1);

set_param('DcModeling/G2','Gain',G2);
```

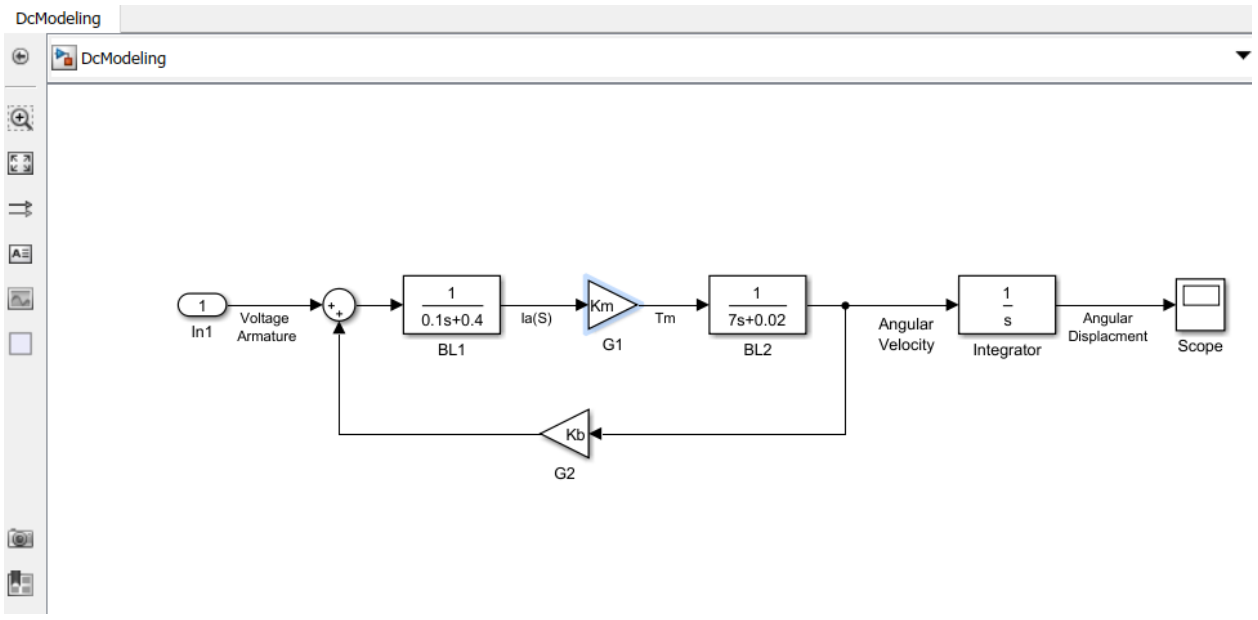


Figure 18Simulink Parameters changed by matlab script

7.1 Angular Diplacement Response

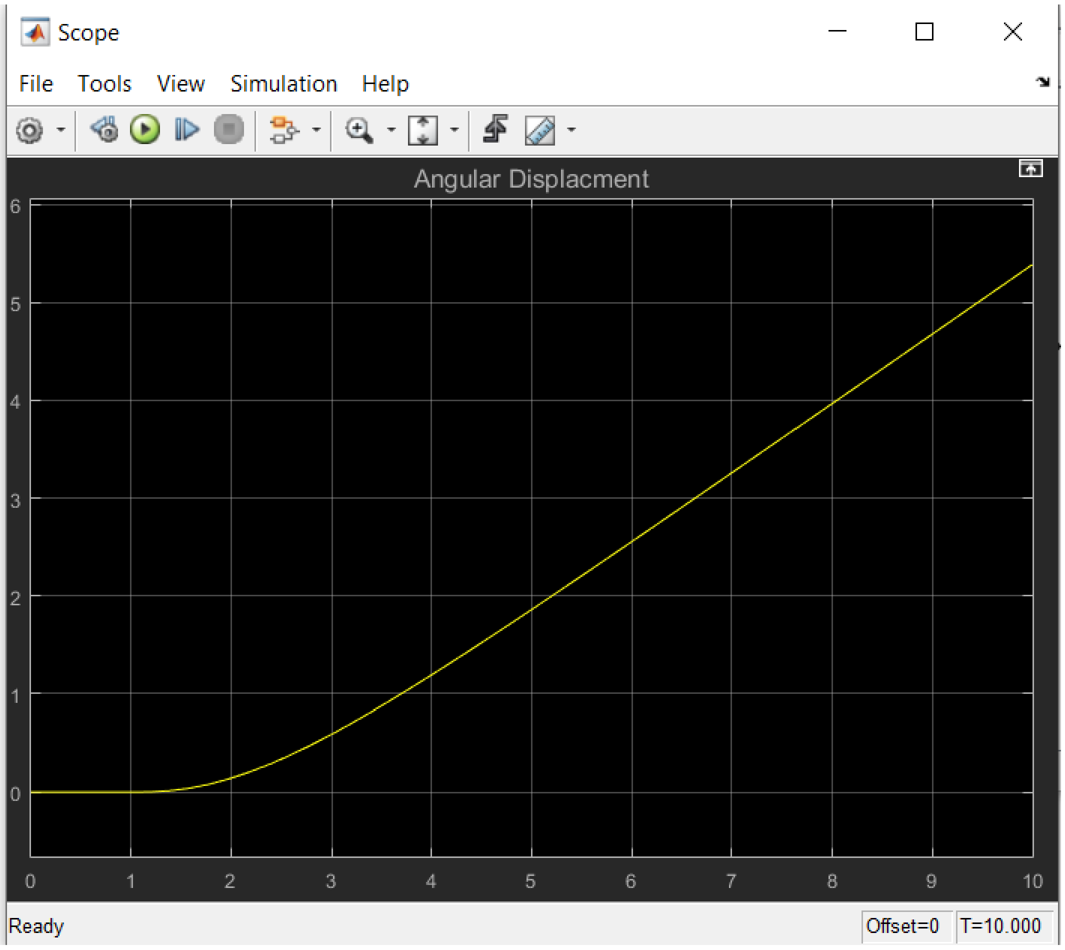


Figure 19Angular Displacment Plot

8 PID Tuning

For system stability, PID controller will be added as follows:

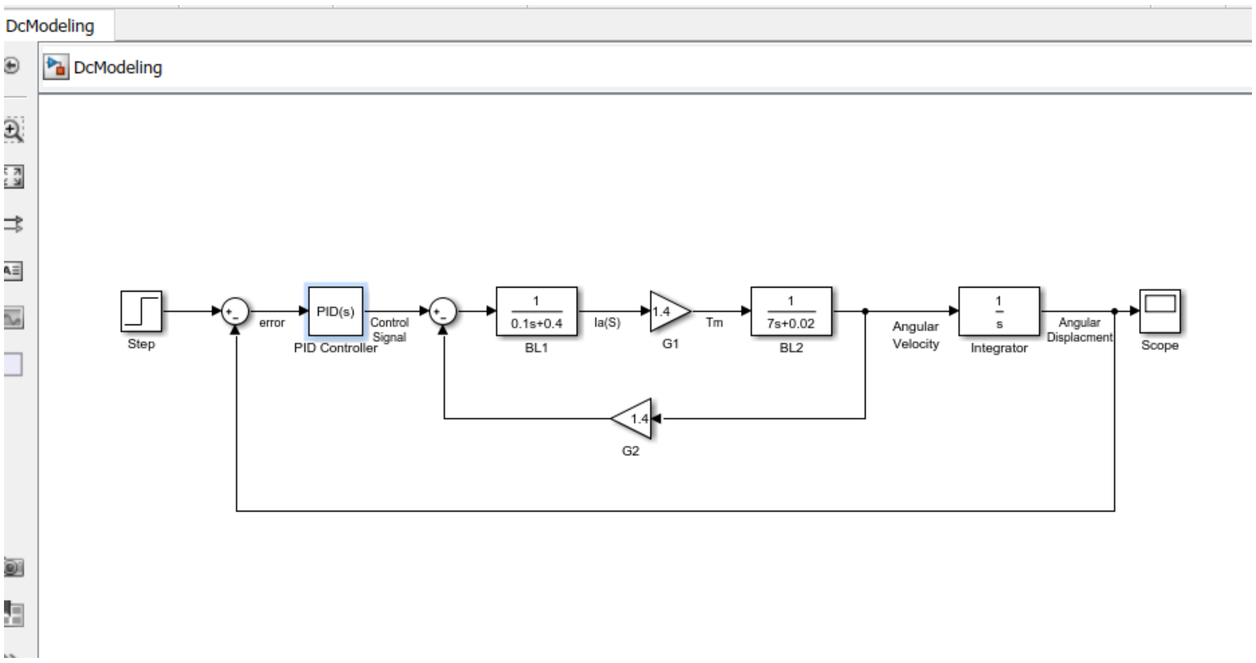


Figure 20PID Controller

8.1 System Respone at Unity parameters

If kp, Ki, kd are all unity

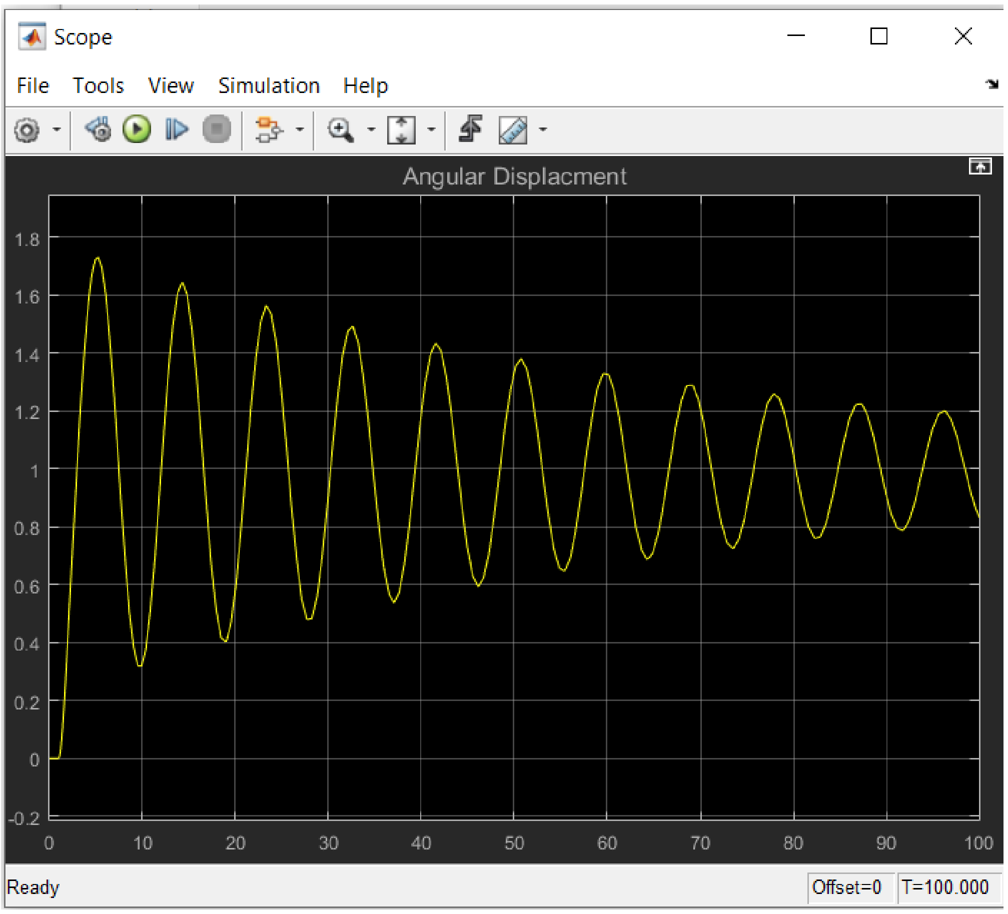


Figure 21 System response at unity Kp Kd Ki Parameters

8.2 tuning the system

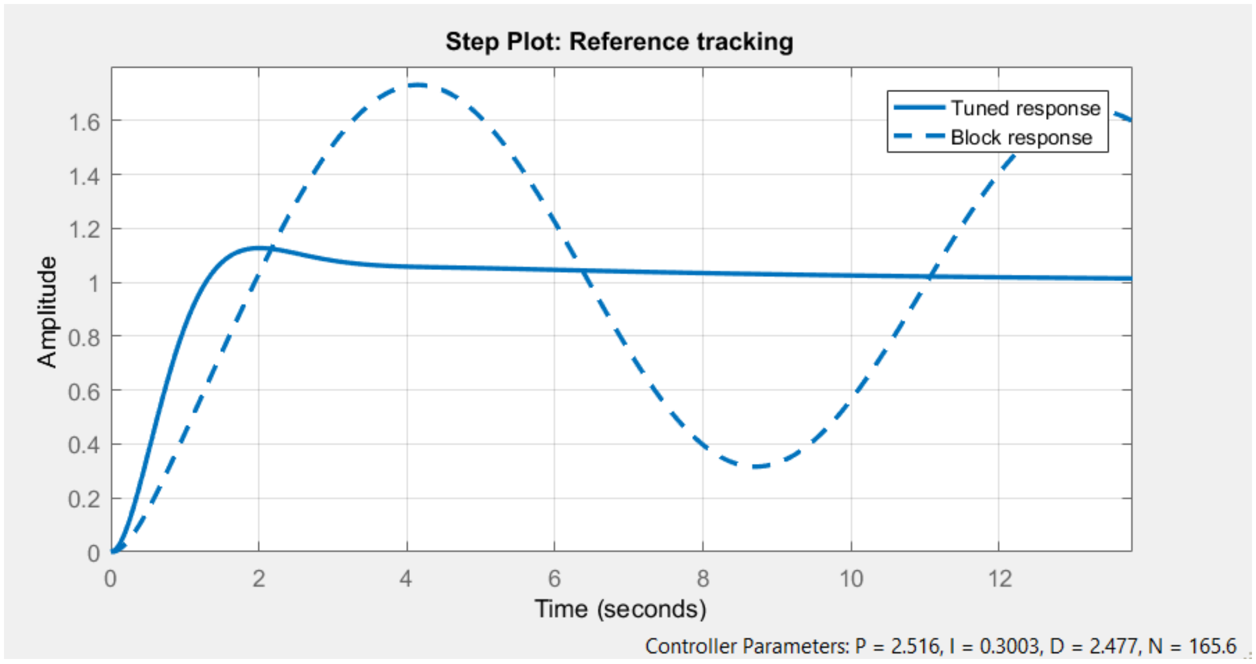
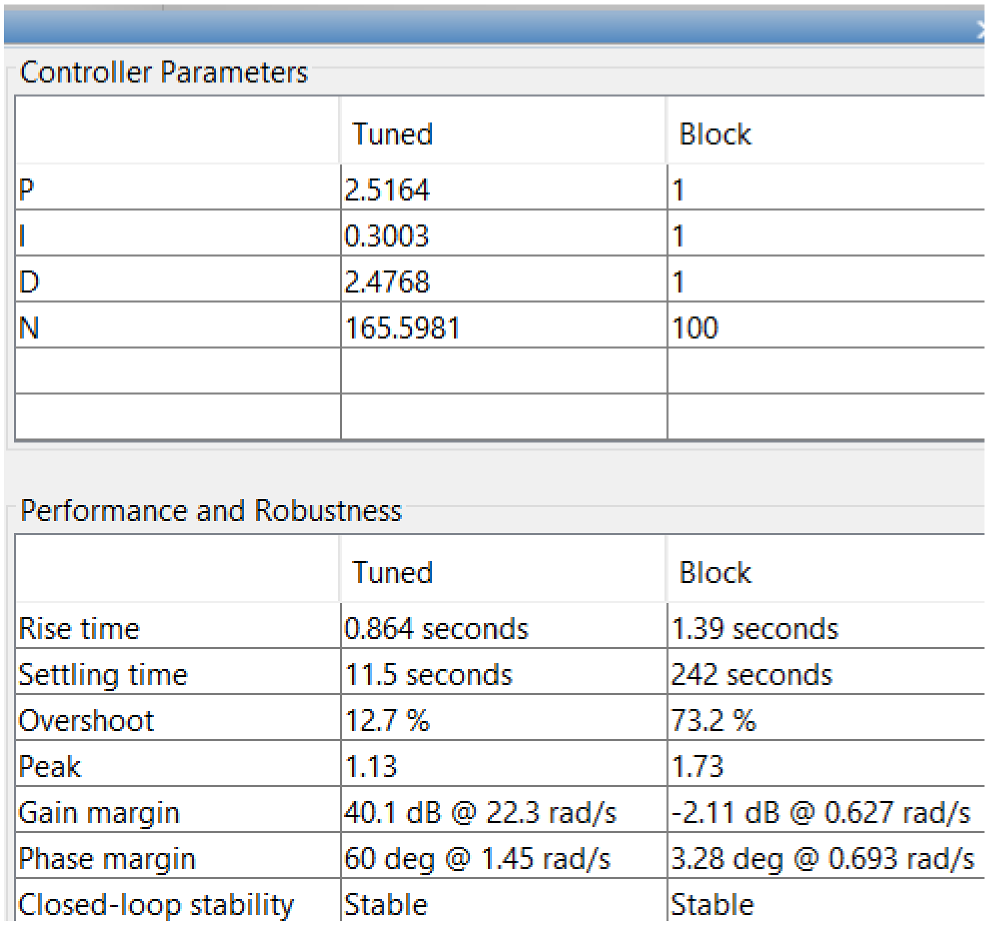


Figure 22PID auto tune

We found our kp, Kd, and Ki, we can now write our Code

8.3 Controller Parameters



Controller Parameters

	Tuned	Block
P	2.5164	1
I	0.3003	1
D	2.4768	1
N	165.5981	100

Performance and Robustness

	Tuned	Block
Rise time	0.864 seconds	1.39 seconds
Settling time	11.5 seconds	242 seconds
Overshoot	12.7 %	73.2 %
Peak	1.13	1.73
Gain margin	40.1 dB @ 22.3 rad/s	-2.11 dB @ 0.627 rad/s
Phase margin	60 deg @ 1.45 rad/s	3.28 deg @ 0.693 rad/s
Closed-loop stability	Stable	Stable

Figure 23Controller Parameters

9 Code

You can find the code here

➔ https://github.com/MuOssama/PID_Demo_Device

Or scan the QR code:

Figure 24 QR code for the Arduino code link

10 Model a discrete time system

G(s) =$\frac{(1.4)}{(0.1S+0.4)(7S+0.02)}$

G(s)=$\frac{G(s)}{1+G(s)H(s)}$ =$\frac{1}{\frac{1}{G(s)+H(s)}}$

G(s)=$\frac{1}{\frac{1}{G(s)+1.4}}$

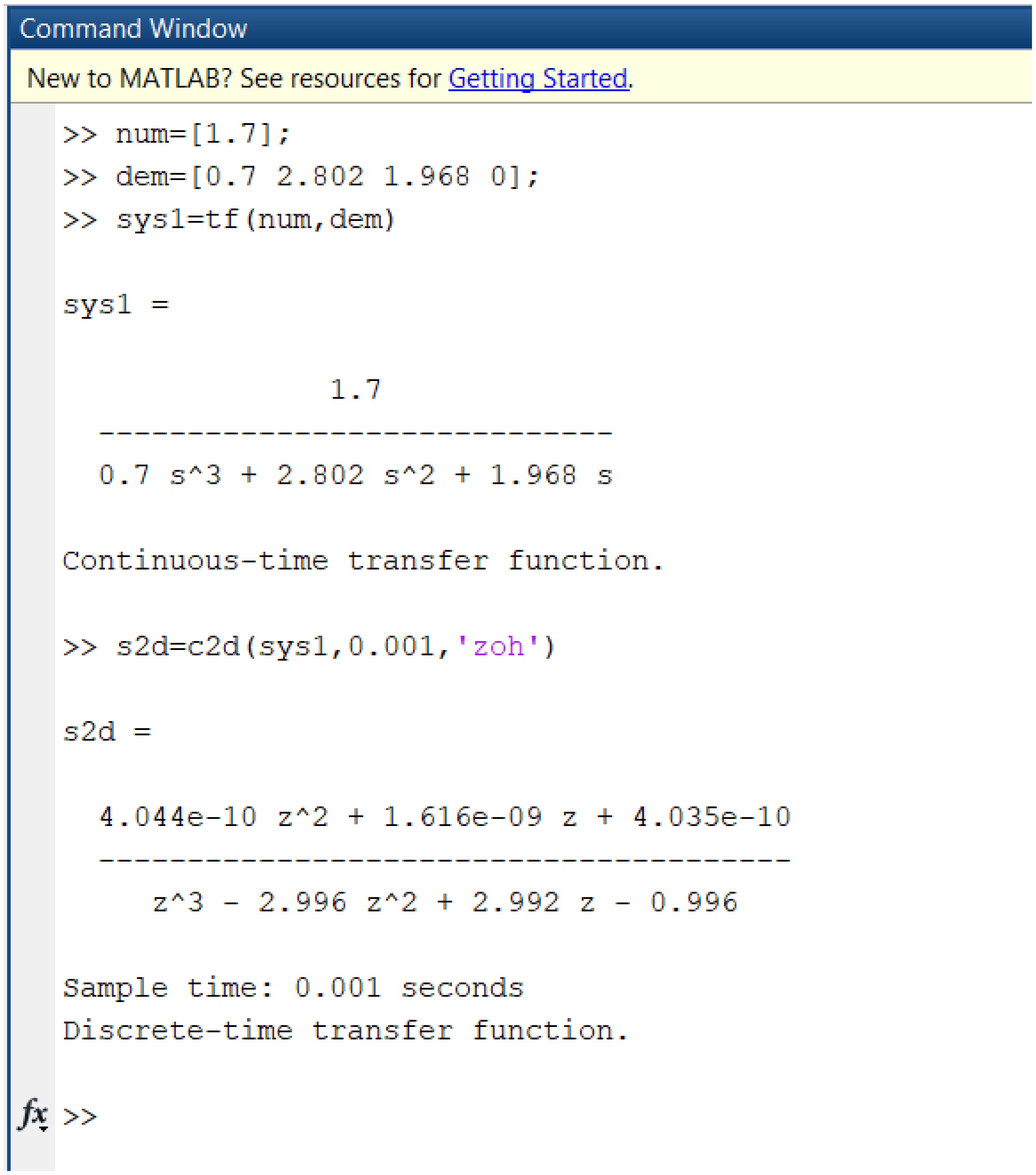
G(s) =$\frac{1.4}{(0.1s+0.4)(7s+0.02)+1.96}$

G(s)=$\frac{1.4}{0.7S^2+0.28S+0.002S+0.008+1.96}$

G(s)=$\frac{1.4}{0.7S^2+0.282S+1.968}$

G(s) =$\frac{1.4}{0.7S^3+0.282S^2+1.968S}$

10.1 By transferring this equation in matlab



```
>> num=[1.7];
>> dem=[0.7 2.802 1.968 0];
>> sys1=tf(num,dem)

sys1 =

                1.7
  -----------------------------
  0.7 s^3 + 2.802 s^2 + 1.968 s

Continuous-time transfer function.

>> s2d=c2d(sys1,0.001,'zoh')

s2d =

  4.044e-10 z^2 + 1.616e-09 z + 4.035e-10
  ---------------------------------------
     z^3 - 2.996 z^2 + 2.992 z - 0.996

Sample time: 0.001 seconds
Discrete-time transfer function.

>>
```

Figure 25 ZOH of S-domain transfer function

10.2 Simulink model

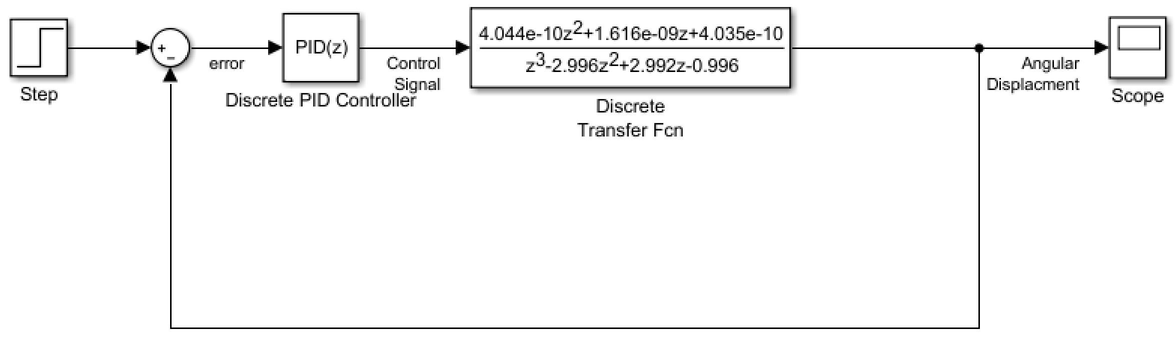


Figure 26 Simulation diagram for a step input, discrete transfer function, PID controller and scope

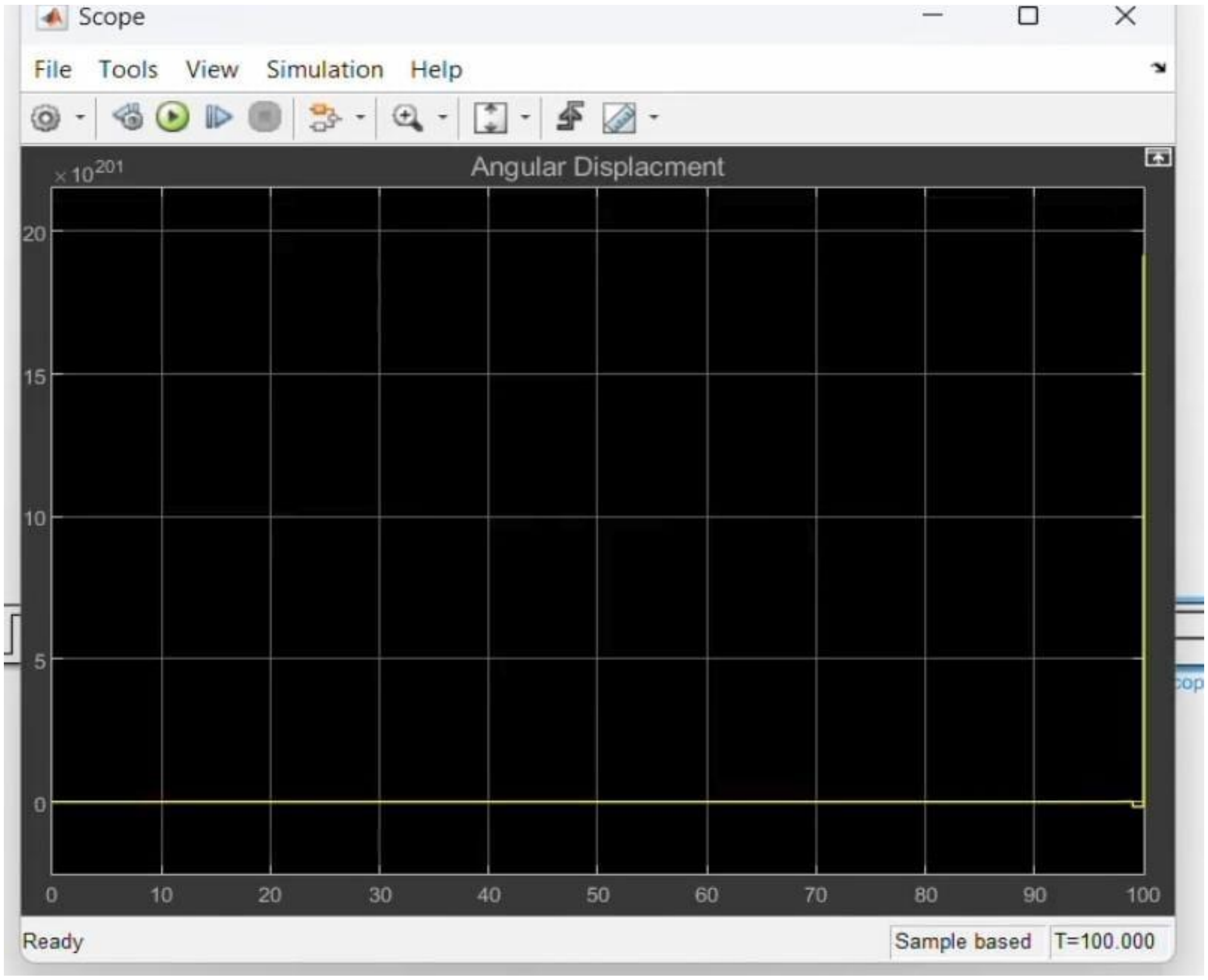


Figure23 step response

10.4 tuning the system

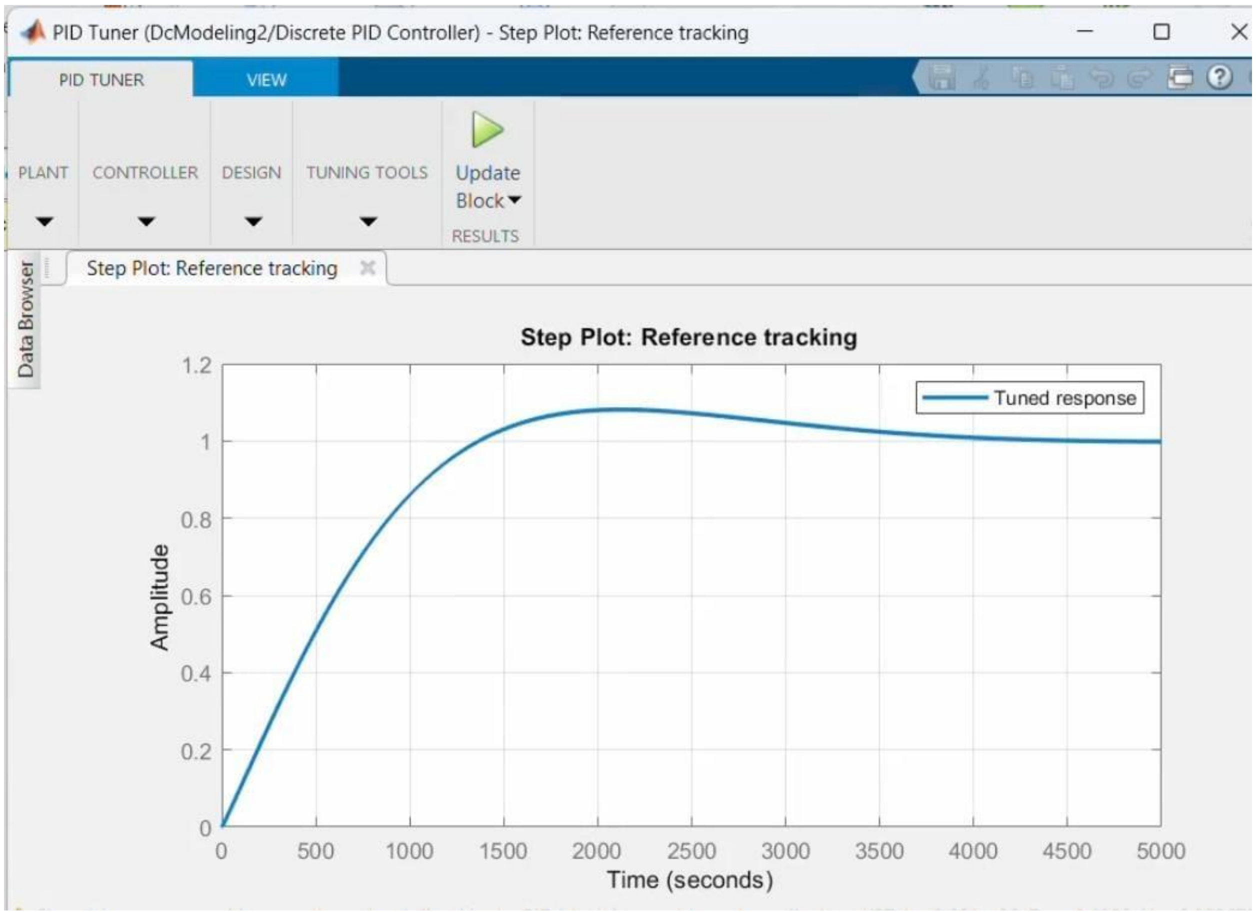

PID Tuner (DcModeling2/Discrete PID Controller) - Step Plot: Reference tracking
PID TUNER
VIEW
PLANT
CONTROLLER
DESIGN
TUNING TOOLS
Update Block
RESULTS
Data Browser
Step Plot: Reference tracking
Step Plot: Reference tracking
Tuned response
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

Figure24

11 User manual

After wiring and flashing the code you face an automatic mode and manual mode

11.1 Automatic mode

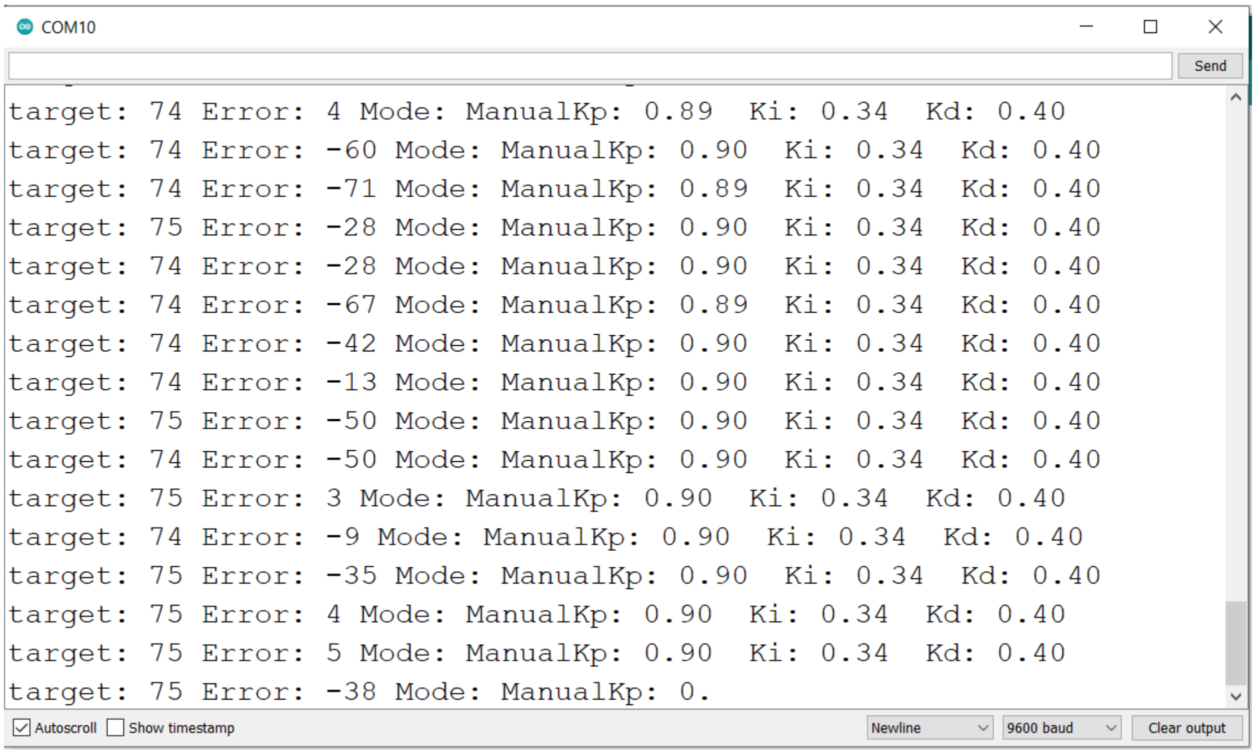
A predefined kp, ki, and kd are given, determined with the best suited, vales and the Simulink auto-tune

11.2 Manual Mode

You can change the kp and ki and kp from the 3 knobs given

11.3 Serial monitor

The serial monitor introduces the following



```
target: 74 Error: 4 Mode: ManualKp: 0.89  Ki: 0.34  Kd: 0.40
target: 74 Error: -60 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 74 Error: -71 Mode: ManualKp: 0.89  Ki: 0.34  Kd: 0.40
target: 75 Error: -28 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 74 Error: -28 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 74 Error: -67 Mode: ManualKp: 0.89  Ki: 0.34  Kd: 0.40
target: 74 Error: -42 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 74 Error: -13 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 75 Error: -50 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 74 Error: -50 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 75 Error: 3 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 74 Error: -9 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 75 Error: -35 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 75 Error: 4 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 75 Error: 5 Mode: ManualKp: 0.90  Ki: 0.34  Kd: 0.40
target: 75 Error: -38 Mode: ManualKp: 0.
```

11.4 Serial Plot

You can see the response of the system from serial plot in Arduino ide.

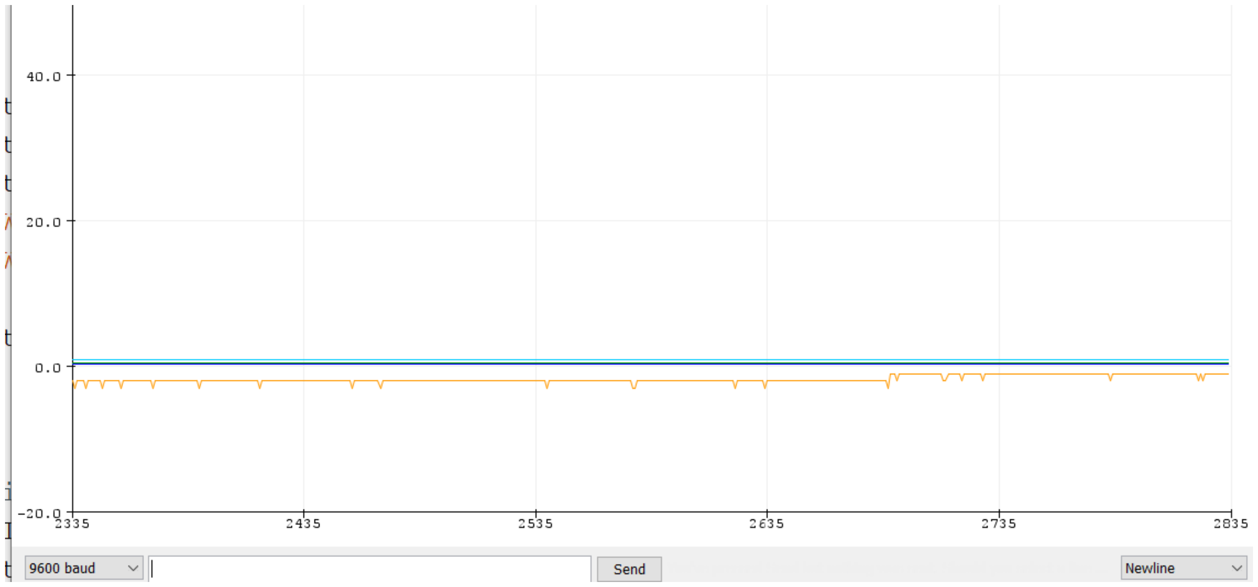


Figure 27 Steady state response

when the target is changed the system oscillates and get back stable again

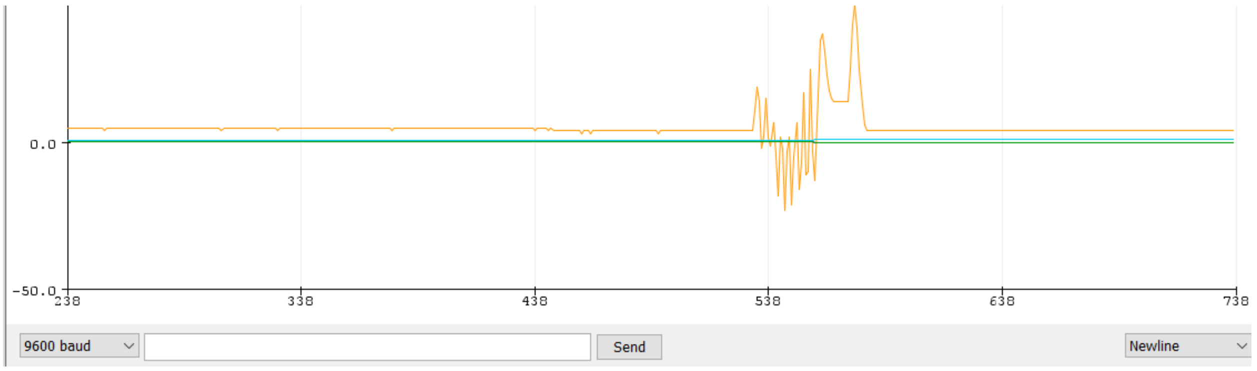


Figure 28 Altering the position.

REFERENCES

[1] Maung, M. M., Latt, M. M., & Nwe, C. M. (2018). DC motor angular position control using PID controller with friction compensation. *International journal of scientific and research publications*, *8*(11), 149.

[2] Khalifa, M., Amhedb, A. H., & Al Sharqawi, M. (2021). Real time DC motor position control using PID controller in LabVIEW. *Journal of Robotics and Control (JRC)*, *2*(5), 342-348.

[3] Dubey, S., & Srivastava, S. K. (2013). A PID controlled real time analysis of DC motor. *International Journal of Innovative Research in Computer and Communication Engineering*, *1*(8), 1965-1973.