AIN SHAMS UNIVERISTY

FACULTY OF ENGINEERING

Senior-1 MECHATRONICS ENGINEERING

MCT211 : Automatic Control

TEAM (8)

DC motor position control

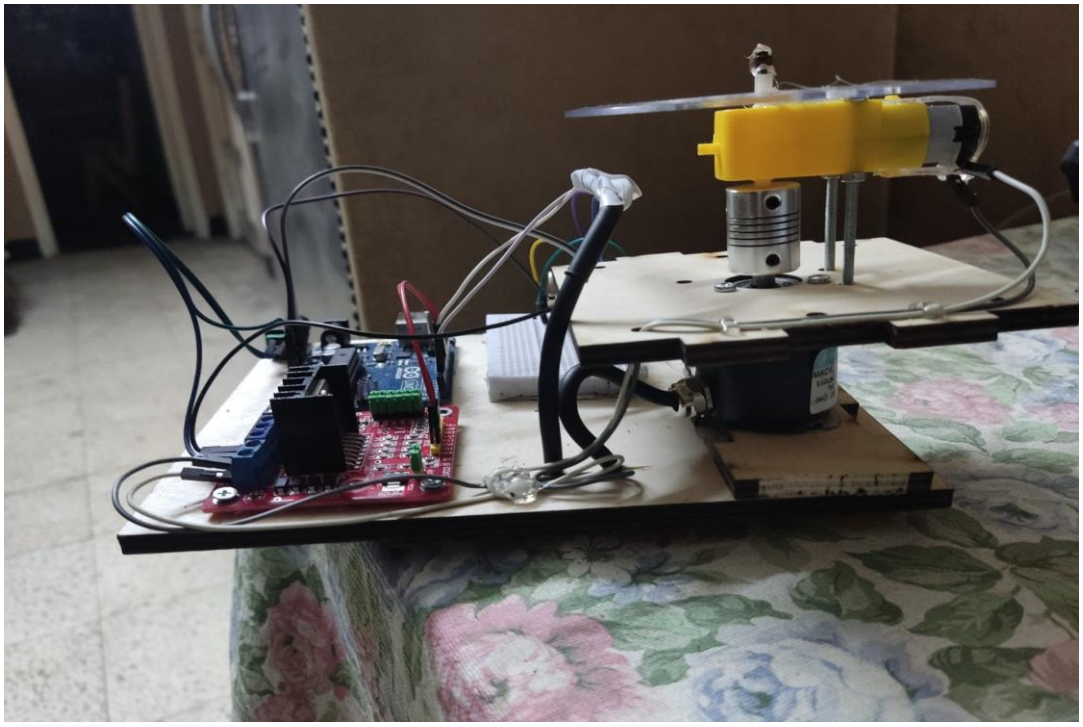
Name	ID	Section
El Sayed Ayman El Sayed Ali	1804765	1
Mahmoud Ayman Abdelaziz EL Said	1802936	1
Anas Ahmed Talaat Khaled	1806766	1
Ahmed Yasser Ahmed Abdelsalam	1805632	1
Abdelrahman Mahmoud Ehab Leithy	1809161	1
Ali Hany Ahmed Ali	1803896	1

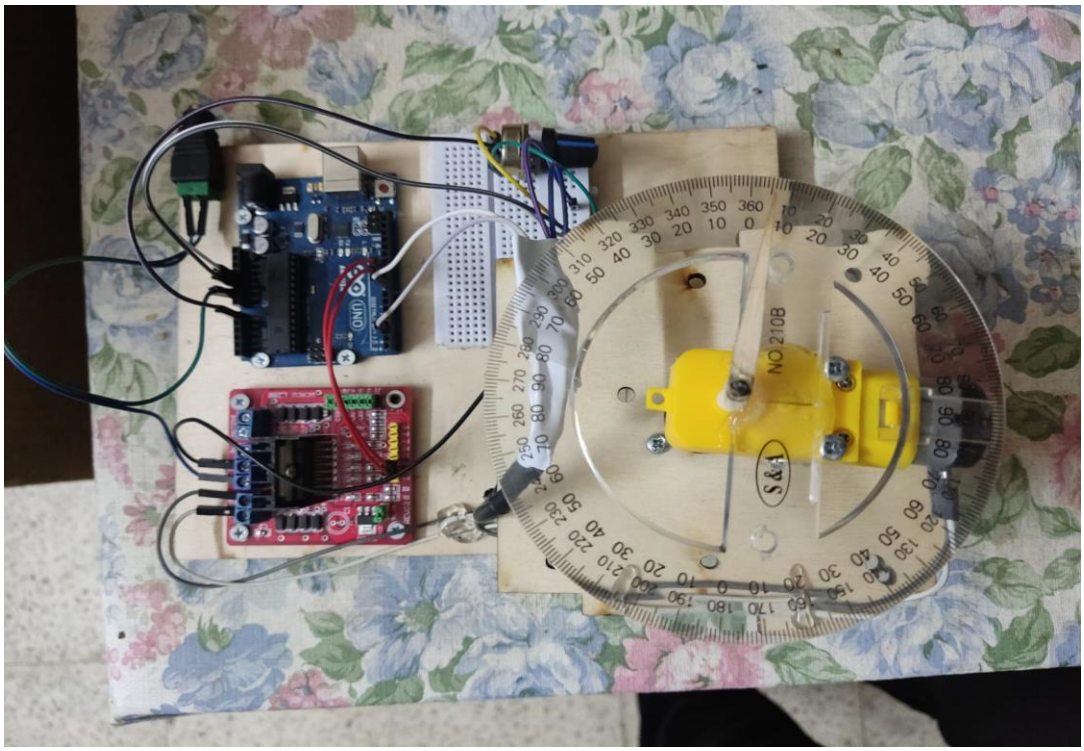
Table of Contents:

Project 1 Report……1

1. The Real Hardware Component's SETUP image……..3
2. Main components……...4
   - 2.1 Arduino Controller chip……4
   - 2.2 Motor Driver……..4
   - 2.3 DC Motor With Encoder……....4
   - 2.4 The protractor……4
   - 2.5 The protractor……4
3. Objective of the Project……5
4. Procedure……5
5. Block diagram……6
6. Circuit diagram……6
7. PID controller for DC motor position control……7
8. Tuning Trials of the DC motor position……..8
   - 8.1 First Trial……8
   - 8.2 Second Trial……...9
   - 8.3 Third Trial……10
   - 8.4 Fourth Trial (The Best One)……11
9. Final PID parameters after Tuning……12
10. Project Code……..13
11. Captured video's Link……19

➢ The Real Hardware Component's SETUP image ::



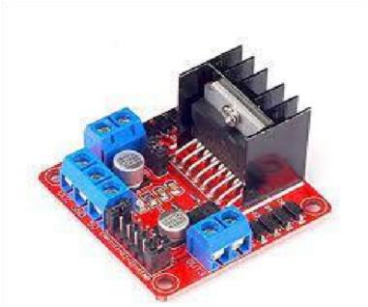


➢ Main components:

- **Arduino Controller chip:**

Function: Controlling the system.

- **Motor Driver:**



Function: Control the speed and the direction of the motor and provide the suitable current to it .

- **DC Motor With Encoder:**

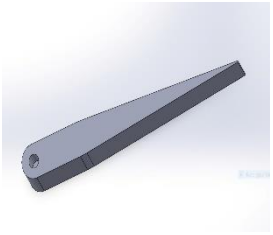
Function: Receive signals from motor driver to Rotate the Pointer.

I.E: The encoder is a position sensor.

- **The protractor:**

Function: it is used for measuring the angles cut by the pointer

- **The pointer:**
  - **Function:** act as an indicator of the cut angle by the motor on the protractor



➢ Objective of the Project:

1. Construct a closed loop feedback control system for position control systems.
2. The effect of the PID controller parameters on the system performance
3. The tuning of the controller parameters.
    - Getting the precision angular position from the motor suing the PID Controller

➢ Procedure:

The angular position of the motor shaft is controlled by a Code builded on the **Arduino UNO chip**

The **motor drive** supplied with power form an external battery and takes the signal from the **Arduino chip** to the **motor requires** more power than the out of Arduino chip passing by **the encoder** that acts a position sensor (encoder's accuracy is 2 degrees)

➢ Block diagram:

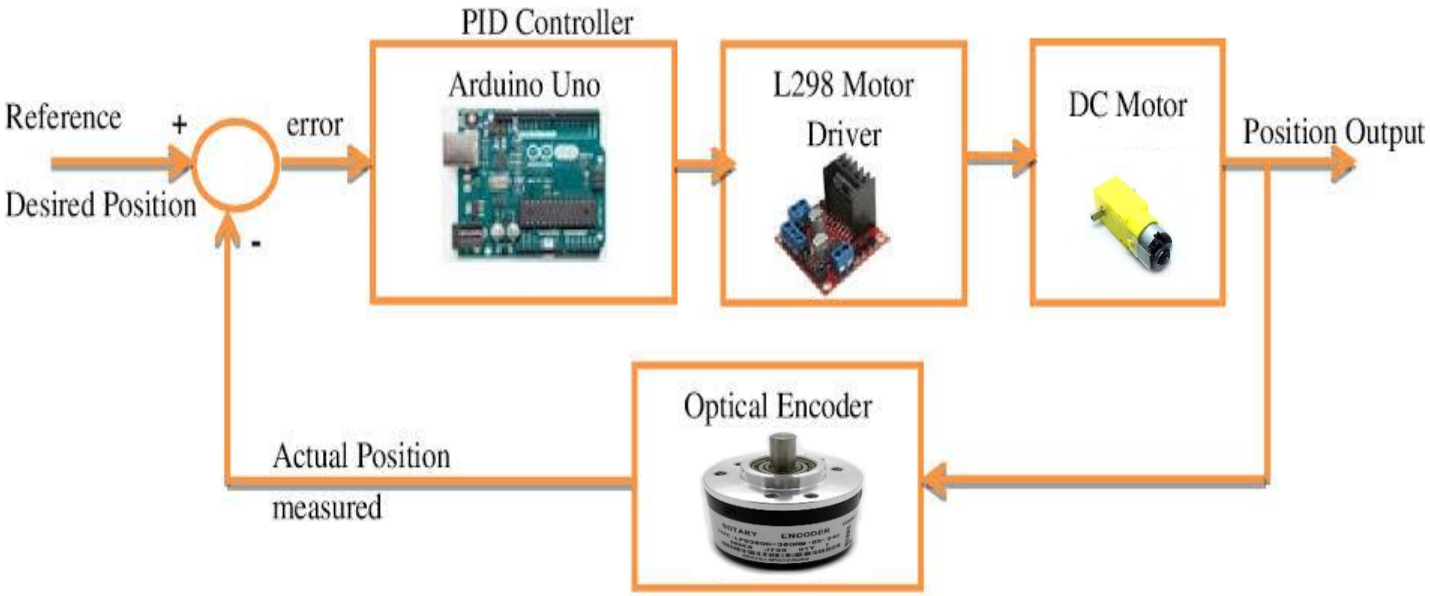
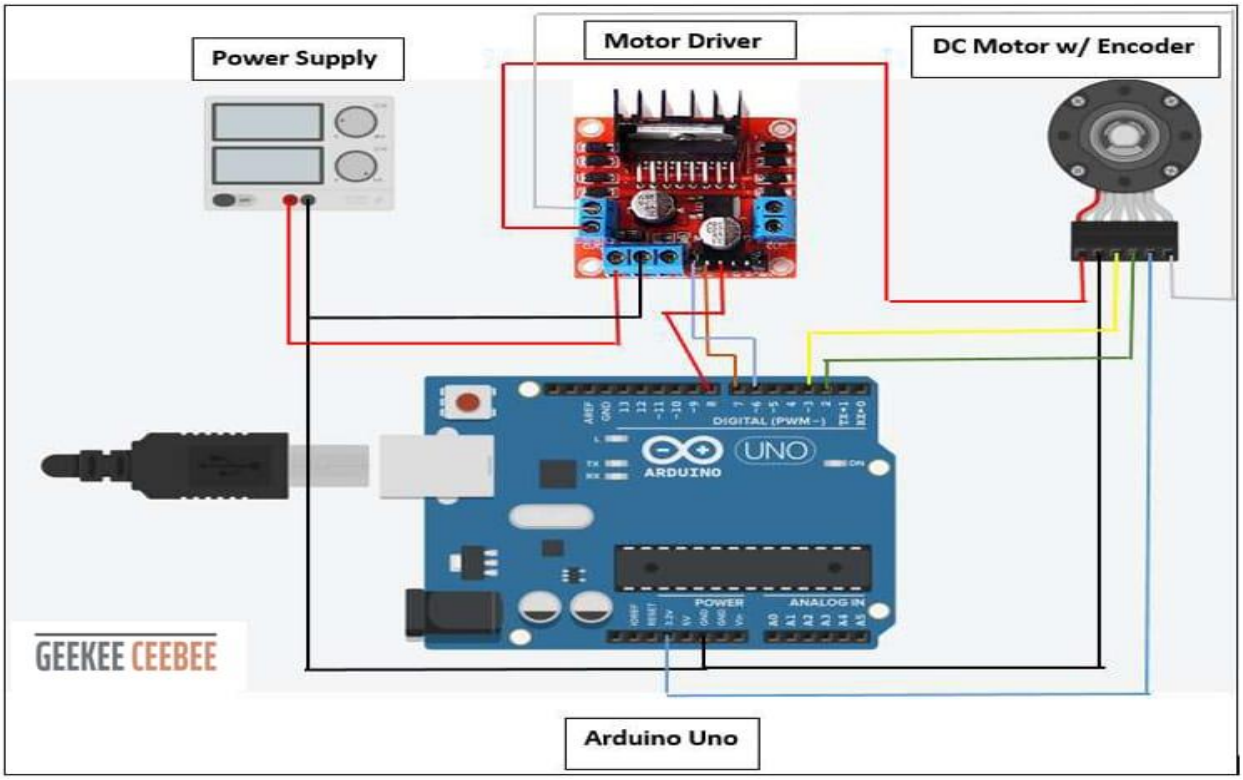


Figure 1: Block Diagram for DC Motor Position Control

➢ Circuit diagram



➢ PID controller for DC motor position control:

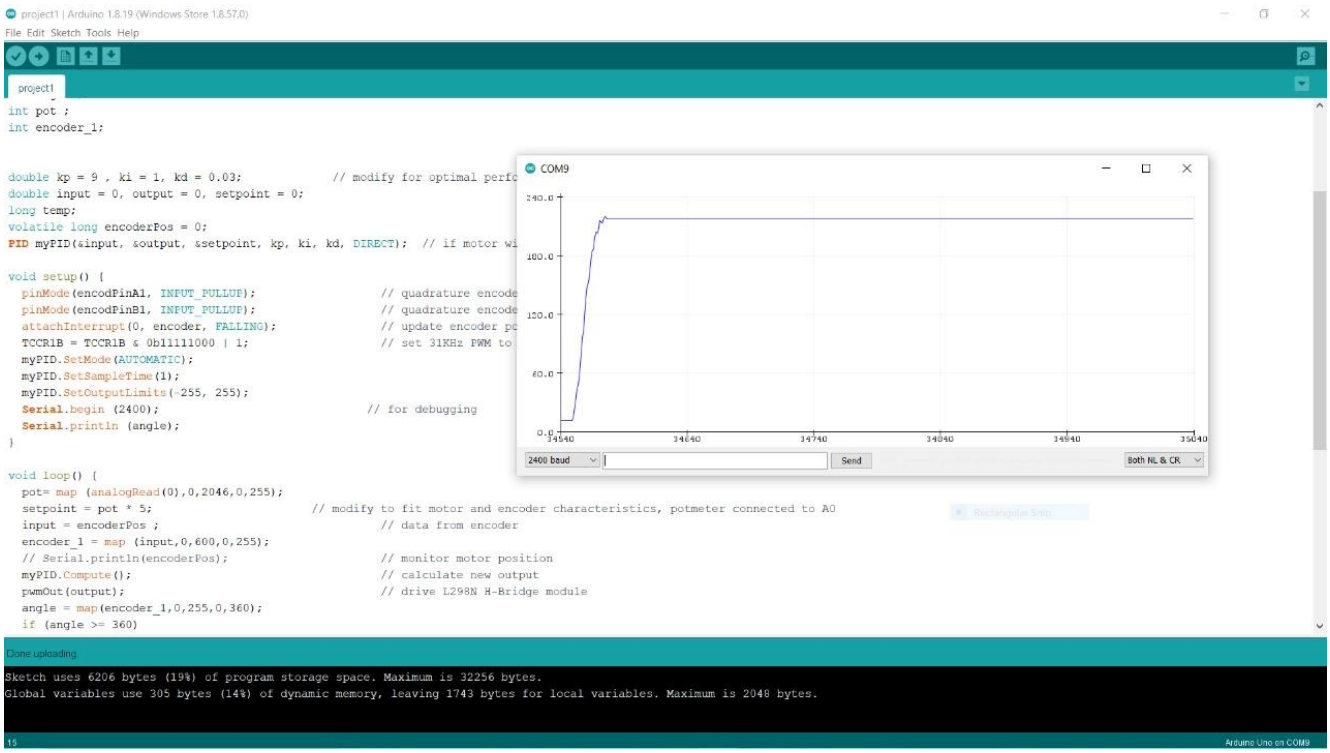
Control action:	it is the variation od the speed of the motor.
Final control element: "Actuator"	The Motor is the actuator
The Sensor:	The encoder is the sensor
Controlled variable:	The motor shaft's angle

➢ Tuning Trials of the DC motor position:

✓ First Trial

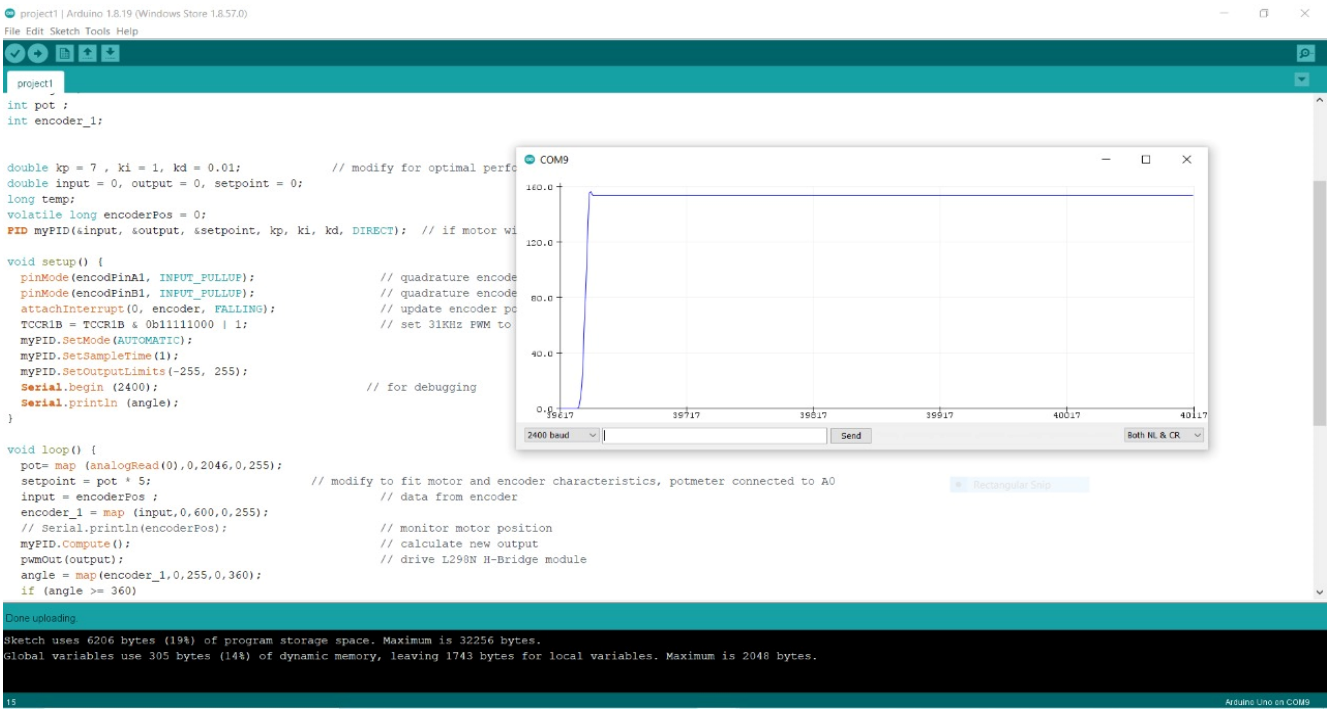
K_p	**10**
K_i	2
K_d	0.1
Set Point	40
Steady State error	8 degrees
Rise Time	66 ms
Overshoot	8 degrees
Settle Time	95 ms

✓ Second Trial



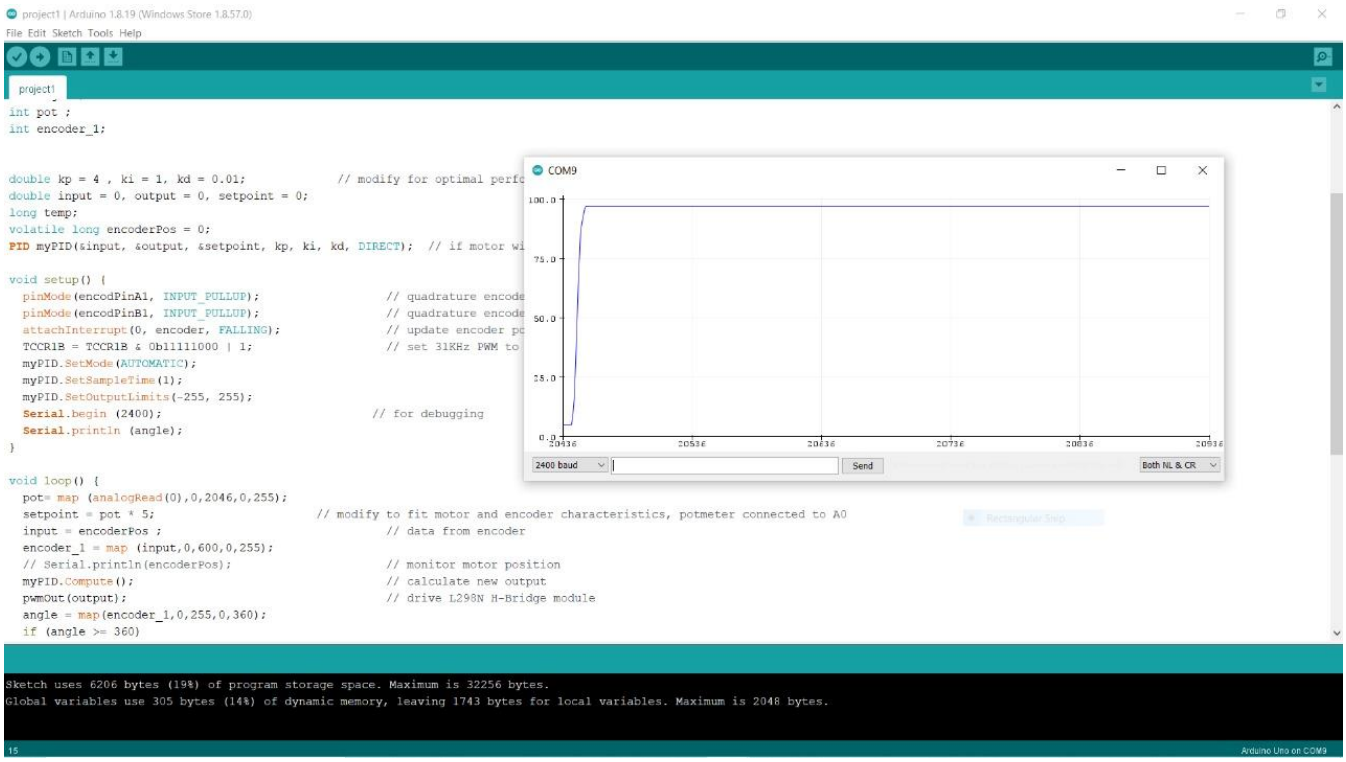
K_p	**9**
K_i	1
K_d	0.03
Set Point	218
Steady State error	0.5 degrees
Rise Time	63 ms
Overshoot	1 degree
Settle Time	65 ms

✓ Third Trial



K_p	7
K_i	1
K_d	0.01
Set Point	150
Steady State error	5 degrees
Rise Time	20 ms
Overshoot	5 degrees
Settle Time	22 ms

✓ Fourth Trial (The Best One)



K_p	4
K_i	1
K_d	0.01
Set Point	98
Steady State error	0 degree
Rise Time	20 ms
Overshoot	0 degree
Settle Time	20 ms

➢ Final PID parameters after Tuning

From the point of view of the settle time and the steady state error we chose the most
suitable parameters to be the same of the 4th trial where:

$$K_p = 4, \qquad K_i = 1, \qquad K_d = 0.01$$

➢ Project Code

```
#include <PinChangeInt.h>
#include <PID_v1.h>
#define encodPinA1      2
#define encodPinB1      8
#define M1              9
#define M2              10
int angle ;
int pot ;
int encoder_1;
double kp = 5 , ki = 1, kd = 0.01;
double input = 0, output = 0, setpoint = 0;
volatile long encoderPos = 0;
PID myPID(&input, &output, &setpoint, kp, ki, kd,
DIRECT);
void setup() {
  pinMode(encodPinA1, INPUT_PULLUP);
  pinMode(encodPinB1, INPUT_PULLUP);
  attachInterrupt(0, encoder, FALLING);
```

```
  TCCR1B = TCCR1B & 0b11111000 | 1;
  myPID.SetMode(AUTOMATIC);
  myPID.SetSampleTime(1);
  myPID.SetOutputLimits(-255, 255);
  Serial.begin (2400);
  Serial.println (angle);
}
void loop() {
  pot= map (analogRead(0),0,2046,0,255);
  setpoint = pot * 5;
  input = encoderPos ;
  encoder_1 = map (input,0,600,0,255);
  myPID.Compute();
  pwmOut(output);
  angle = map(encoder_1,0,255,0,360);
  if (angle >= 360)
  {
    angle = 360;
    }
  Serial.println (angle);
```

```
}
void pwmOut(int out) {
  if (out > 0) {
    analogWrite(M1, out);
    analogWrite(M2, 0);
  }
  else {
    analogWrite(M1, 0);
    analogWrite(M2, abs(out));
  }
}
void encoder()  {
  if (PINB & 0b00000001)    encoderPos++;
  else                      encoderPos--;
}
```

Functions in (PinChangeInt.h) and (PID_v1.h)

```
//Constants used in some of the functions below
  #define AUTOMATIC    1
  #define MANUAL       0
  #define DIRECT       0
  #define REVERSE      1

/*Compute all the working error variables*/
double input = *myInput;
double error = *mySetpoint - input;
double dInput = (input - lastInput);
outputSum+= (ki * error);

PID (double*, double*, double*, double, double, double, int);
// * constructor.  links the PID to the Input, Output, and
Setpoint.  Initial tuning parameters are also set here

void SetMode(int Mode);
 // * sets PID to either Manual (0) or Auto (non-0)
```

bool Compute();

// * performs the PID calculation. it should be called every time loop() cycles. ON/OFF and calculation frequency can be set using SetMode SetSampleTime respectively

void SetOutputLimits(double, double);

// * clamps the output to a specific range. 0-255 by default, but it's likely the user will want to change this depending on the application available but not commonly used functions

void SetTunings(double, double, double);

// * While most users will set the tunings once in the constructor, this function gives the user the option of changing tunings during runtime for Adaptive control

void SetTunings(double, double, double, int);

// * overload for specifying proportional mode

```
void SetControllerDirection(int);
// * Sets the Direction, or "Action" of the controller. DIRECT
means the output will increase when error is positive.
REVERSE means the opposite.  it's very unlikely that this
will be needed once it is set in the constructor.


void SetSampleTime(int);
// * sets the frequency, in Milliseconds, with which the PID
calculation is performed.  default is 100


/*Remember some variables for next time*/
lastInput = input;
lastTime = now;
```

- Captured video's Link

https://drive.google.com/drive/folders/137M1QrBRsqJ2cvn5wlVcci19IapBKC_?usp=sharing