



Elevator system Control

Report submitted to Dr. Abdelaziz Morgan,

Eng. Norhan Ghoneim,

Eng. Amr Mohammed

Prepared by:

Arwa Ashraf Mahmoud Farag 1805479

Mohamed Eyad Sayed Abdallah 18P8431

Mohamed Alaa Eldin Mohamed 18P1851

Mohammed Ashraf Saeed Eladwy 18P5496

Mahmoud Magdy Mahmoud Elasmr 18P8983

TABLE OF CONTENTS

1. ABSTRACT	4
2. INTRODUCTION.....	4
2.1. Automatic Control.....	4
2.2. Different Types Of Controllers	5
2.2.1. Bang Bang controller	5
2.2.2. PID controller	5
3. ELEVATOR SYSTEM CONTROL	6
3.1. System Analysis.....	6
3.1.1. System identification	6
3.1.2. System Block diagram	7
3.2. System Mechanical design - CAD	7
3.3. Open Loop System Analysis	7
3.3.1. Open loop system model using Simscape.....	7
3.3.2. System response to unit step input.....	8
3.4. Closed Loop System Analysis	9
3.5. PID Tuning	10
3.6. System frequency response	14
3.7. System Flowcharts	15
3.8. Code	17
4. IMPLEMENTATION.....	21
4.1. Practical system response at different sets of parameters	21
4.2. Practical system response with the chosen parameters	23
5. WORKING VIDEO	23
6. PRESENTATION.....	23

TABLE OF FIGURES

Figure 1 Block diagram of a closed loop system.....	4
Figure 2 PID Control Block Diagram	6
Figure 3: Elevator System Block Diagram	6
Figure 4: Elevator System Cad	7
Figure 5: Open Loop physical model	7
Figure 6 open loop system response to unit step input	8
Figure 7: open loop system response to sin input	8
Figure 8 : response to ramp input.....	9
Figure 9: Closed loop System with PID control.....	9
Figure 10 $K_p=1, K_i=1, K_d =0$	10
Figure 11: System response – MATLAB tuned PID parameters	11
Figure 12 $K_p=0.125, K_i=12.753, K_d =0$	12
Figure 13 System response - $K_p=0.125, K_i=12.753, K_d=0$	12
Figure 14 $K_p=0.01, K_i=1.3, K_d=0$	13
Figure 15 System response - $K_p=0.01, K_i=1.3, K_d=0$	13
Figure 16 Bode plot	14
Figure 17 Nyquist Diagram	14
Figure 18 :PID control code Flowchart	15
Figure 19: Code flowchart	16
Figure 20 Setpoint 160 RPM	21
Figure 21 Measured RPM - Setpoint 120 RPM	22
Figure 22 Measured RPM - Setpoint 100	22
Figure 23 Measured RPM - Setpoint 200	23

1. ABSTRACT

Mechatronics systems can be described using block diagrams after determining the plant, feedback, and the controller. Our system is an elevator, and the goal of this project is to control the speed of the elevator using PID controller. In this report, PID controller will be illustrated. The steps of analyzing the system which starts with the block diagram till making the physical model using MATLAB are discussed. Also, the process of controlling the speed with all the trials and resulted responses are shown.

2. INTRODUCTION

2.1. Automatic Control

A control system manages, commands, directs, or regulates the behavior of other devices or systems using control loops. It can range from a single home heating controller using a thermostat controlling a domestic boiler to large industrial control systems which are used for controlling processes or machines.

For continuously modulated control, a feedback controller is used to automatically control a process or operation. The control system compares the value or status of the process variable (PV) being controlled with the desired value or setpoint (SP) and applies the difference as a control signal to bring the process variable output of the plant to the same value as the setpoint. Automatic control is the application of control systems without human intervention.

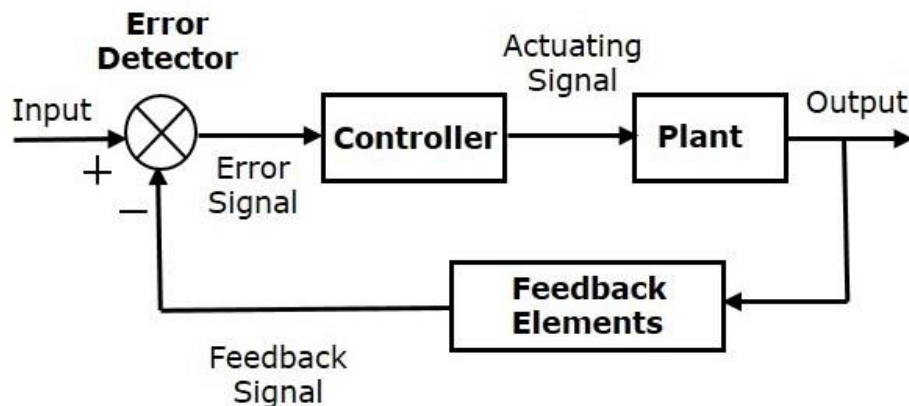


Figure 1|Block diagram of a closed loop system.

2.2. Different Types Of Controllers

2.2.1. Bang Bang controller

In control theory, a bang–bang controller (2 step or on–off controller), is a feedback controller that switches abruptly between two states. These controllers may be realized in terms of any element that provides hysteresis. They are often used to control a plant that accepts a binary input, for example a furnace that is either completely on or completely off.

Bang Bang controller is very simple mathematically, however; the physical realization of bang-bang control systems gives rise to several complications. First of those complications is due to the hysteresis gap. The width of the hysteresis and the inertia of the system will result in an output that oscillates around the desired setpoint. Second complication is the on/off action given to the system might result in metal fatigue or wear to the system elements. Thus, most of the times it's better to use a continuous control system, as PID.

2.2.2. PID controller

A proportional–integral–derivative controller (PID controller or three-term controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value $e(t)$ as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively), hence the name.

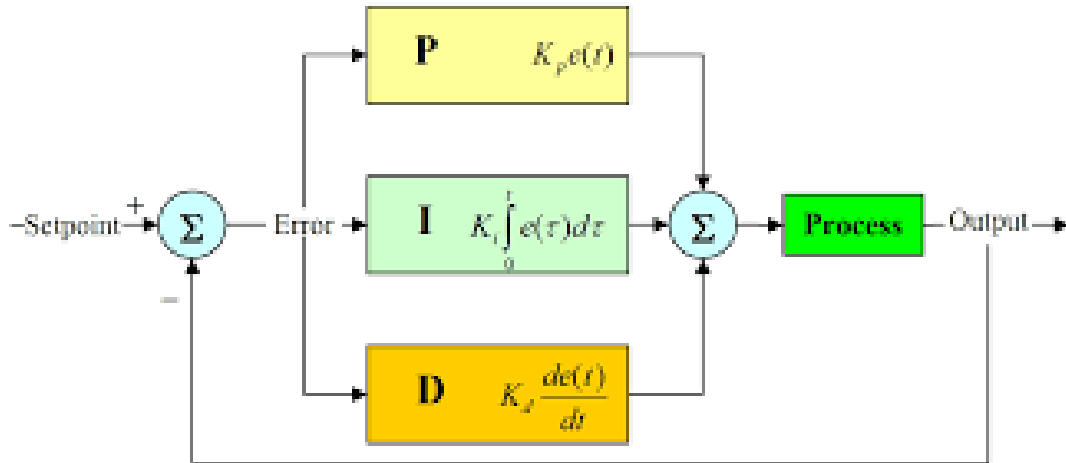


Figure 2 | PID Control Block Diagram

3. ELEVATOR SYSTEM CONTROL

3.1. System Analysis

3.1.1. System identification

- Controlled variable: Elevator speed
- The plant: Elevator cabinet
- Control element: Microcontroller
- Final control elements: Motor driver, motor, and pulley
- Feedback sensor: Encoder.

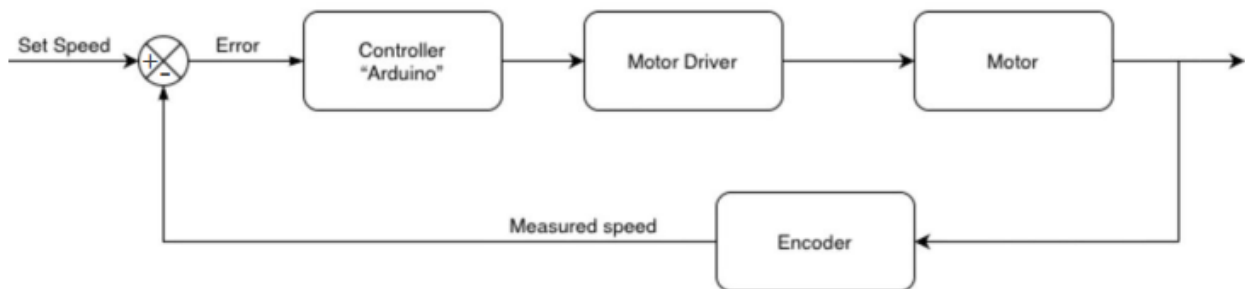


Figure 3: Elevator System Block Diagram

3.1.2. System Block diagram

3.2. System Mechanical design - CAD

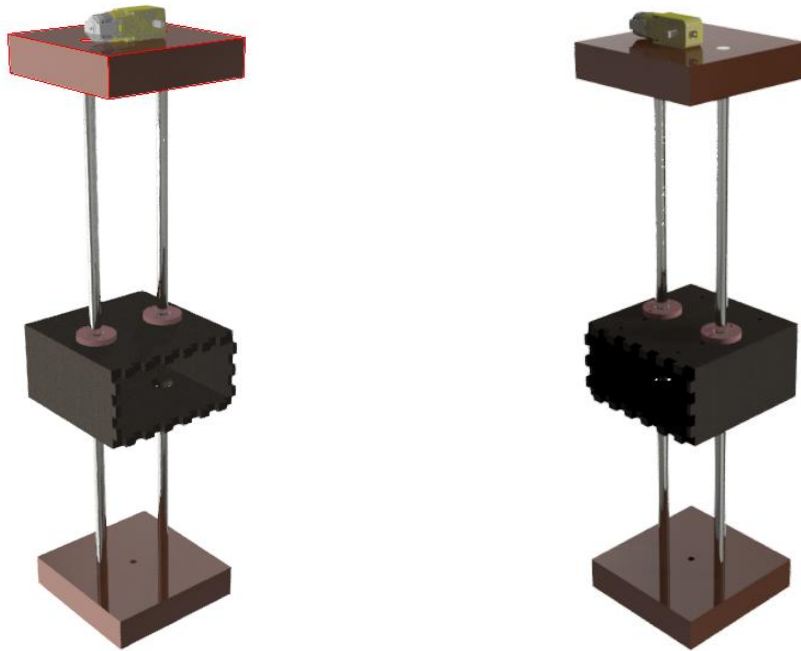


Figure 4: Elevator System Cad

3.3. Open Loop System Analysis

3.3.1. Open loop system model using Simscape

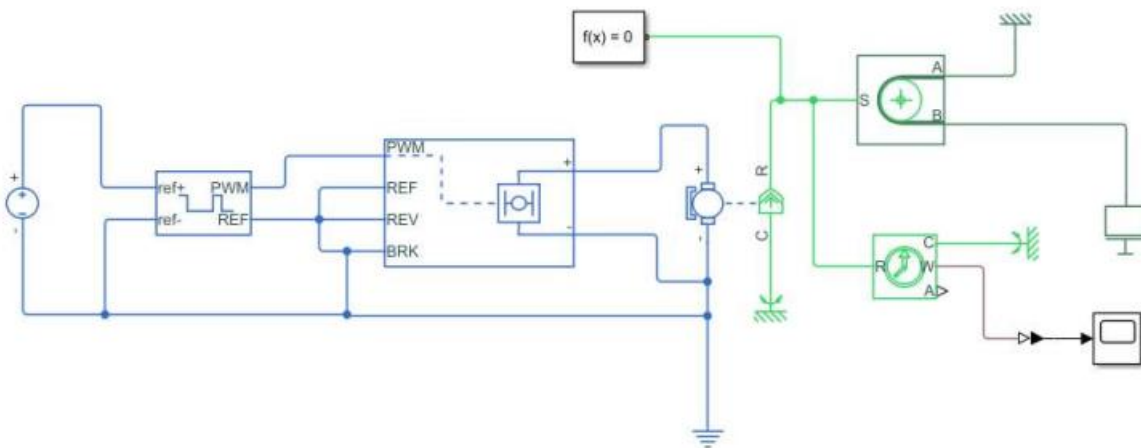


Figure 5: Open Loop physical model

3.3.2. System response to unit step input

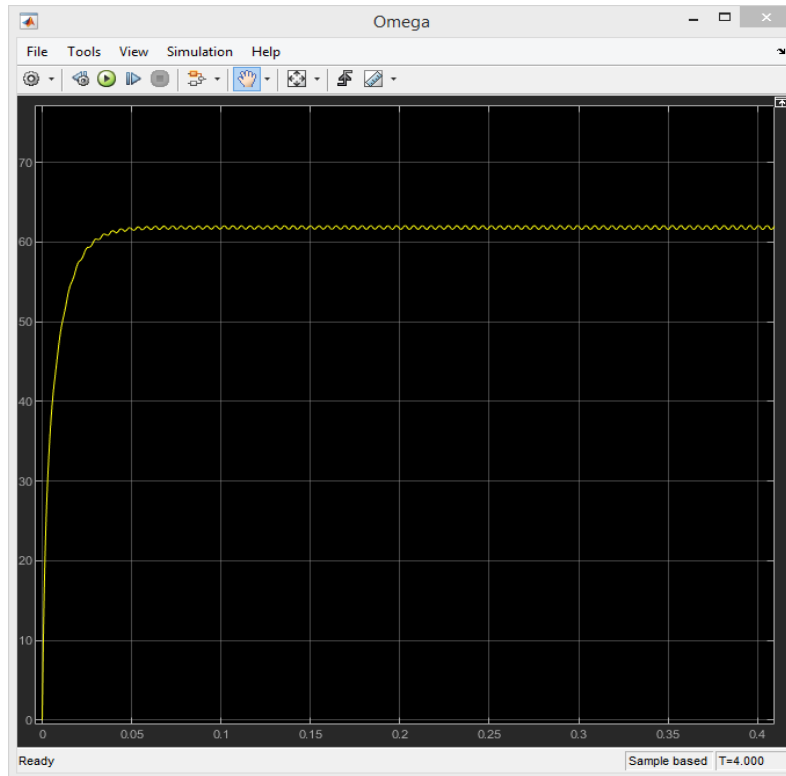


Figure 6| open loop system response to unit step input

3.3.3. System response to sinusoidal input

The phase shift is obvious in the response of the system and shows how quickly the system responds to changes.

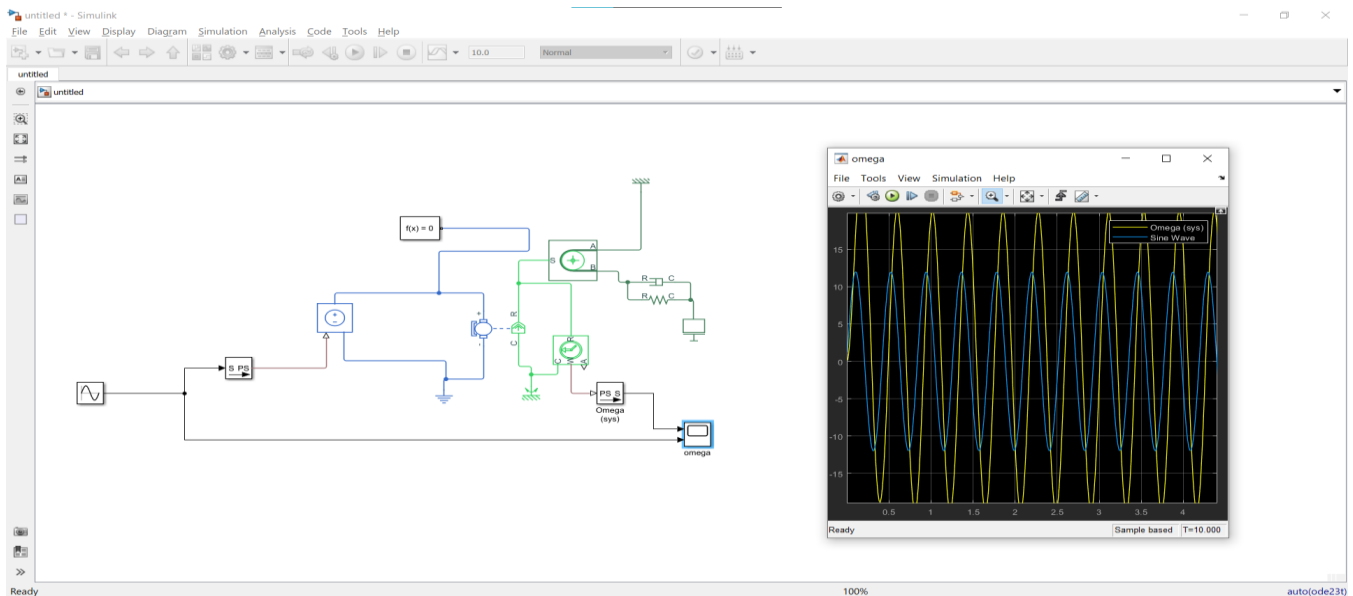


Figure 7: open loop system response to sin input

3.3.4. Ramp input

There is an error when at first and keeps growing as there is no feedback.

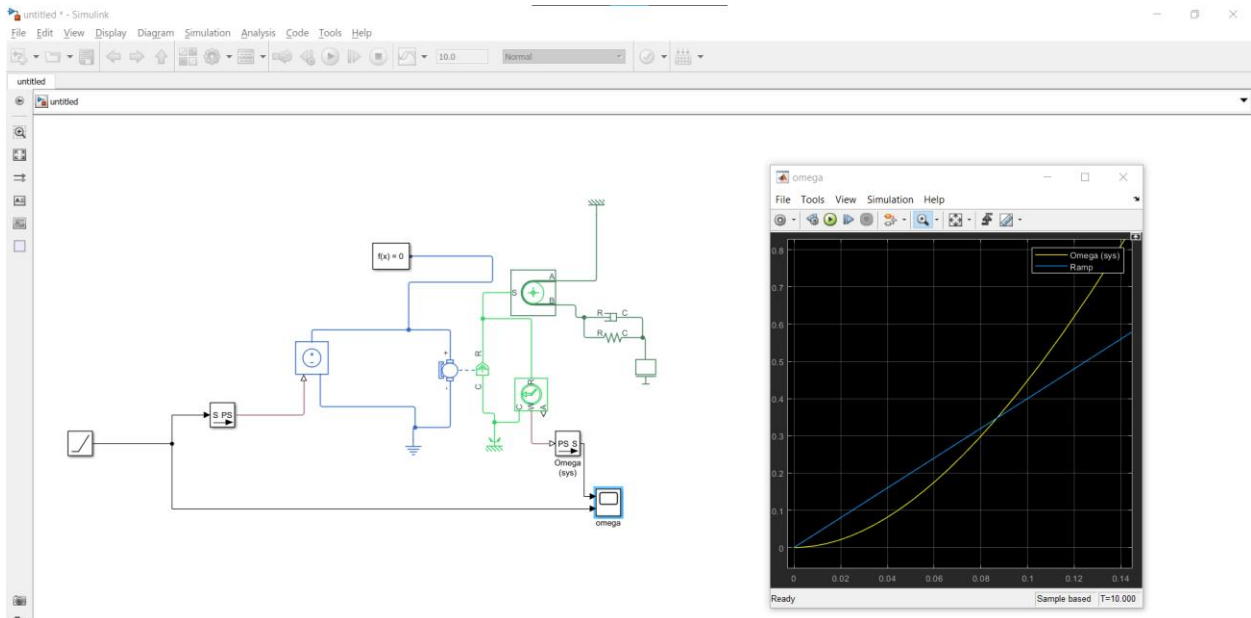


Figure 8 : response to ramp input

3.4. Closed Loop System Analysis

3.4.1. Closed loop system model using Simscape

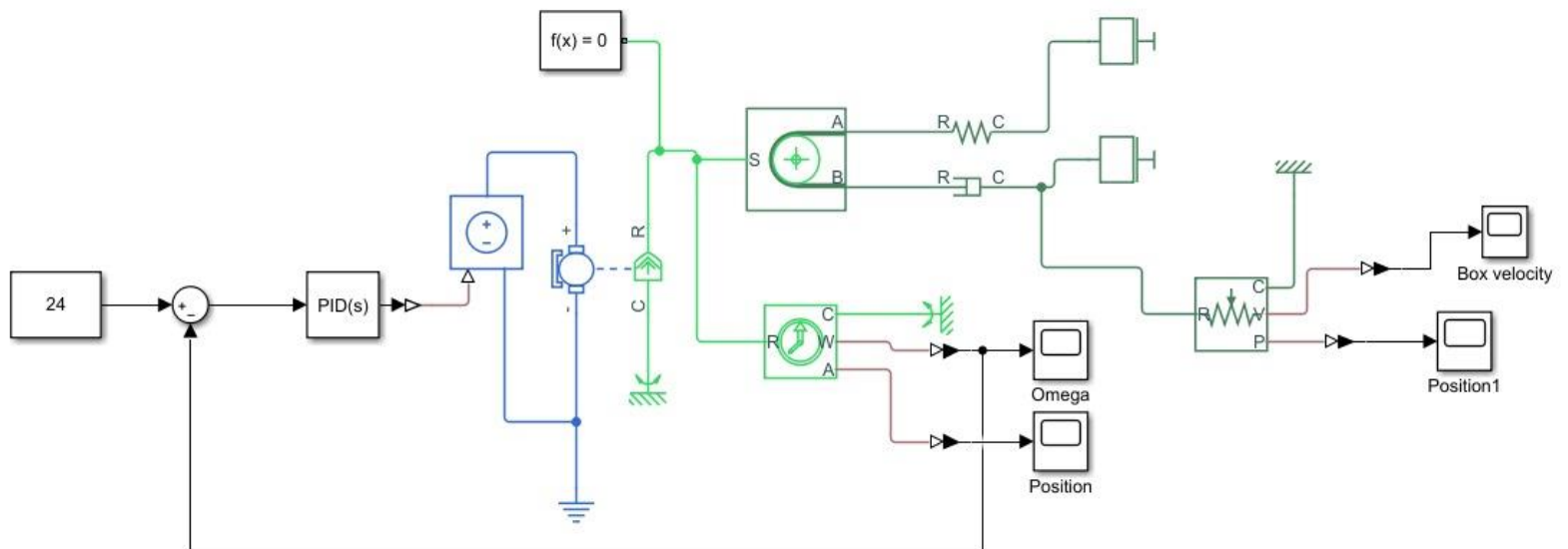


Figure 9: Closed loop System with PID control

3.5. PID Tuning

After adding the PID block in Simscape, the system response was checked without changing the original parameter which were as follows:

$$K_p=1, K_i=1, K_d=0$$

The motor velocity was oscillating, and the settling time was high. Thus, the 'tune' option of PID block is used to get a better response.

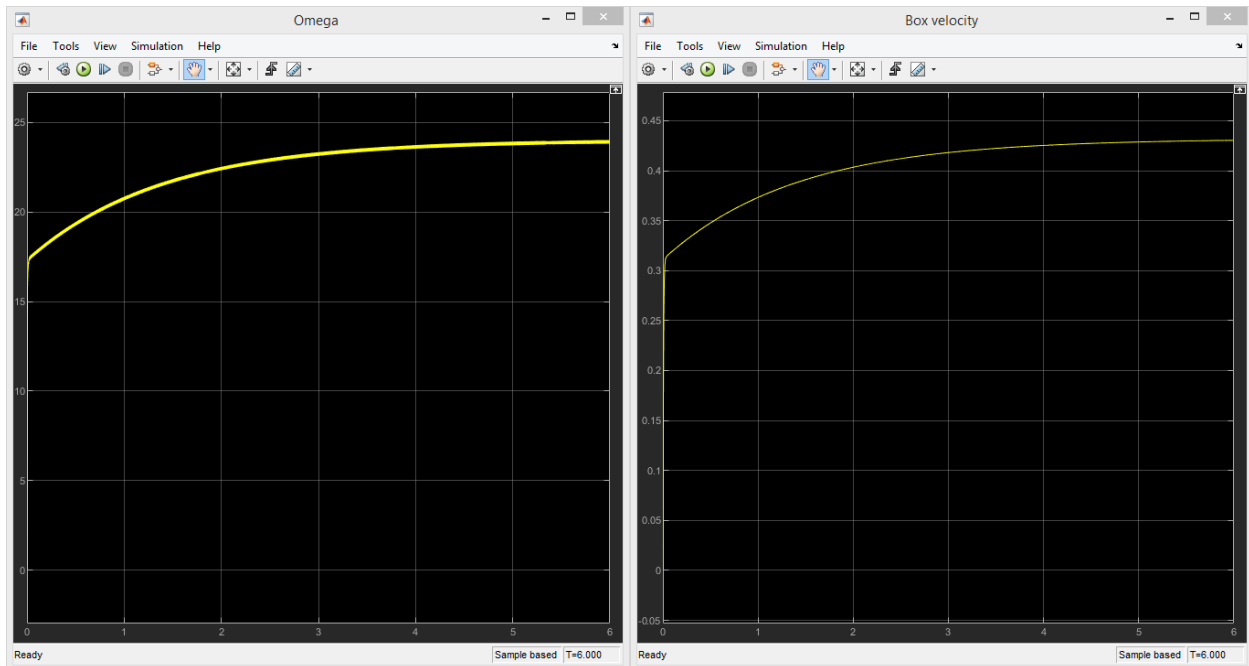


Figure 10| $K_p=1, K_i=1, K_d=0$

3.5.1. System response - Matlab PID tuned parameters

The response of the system had an overshoot at first and continues to oscillate about the setpoint. Thus, with the knowledge of the effect of each parameter (K_p , K_i , K_d), different combinations of parameters were tried, and the response was checked to remove the overshoot. As, an elevator can not have an overshoot while moving.

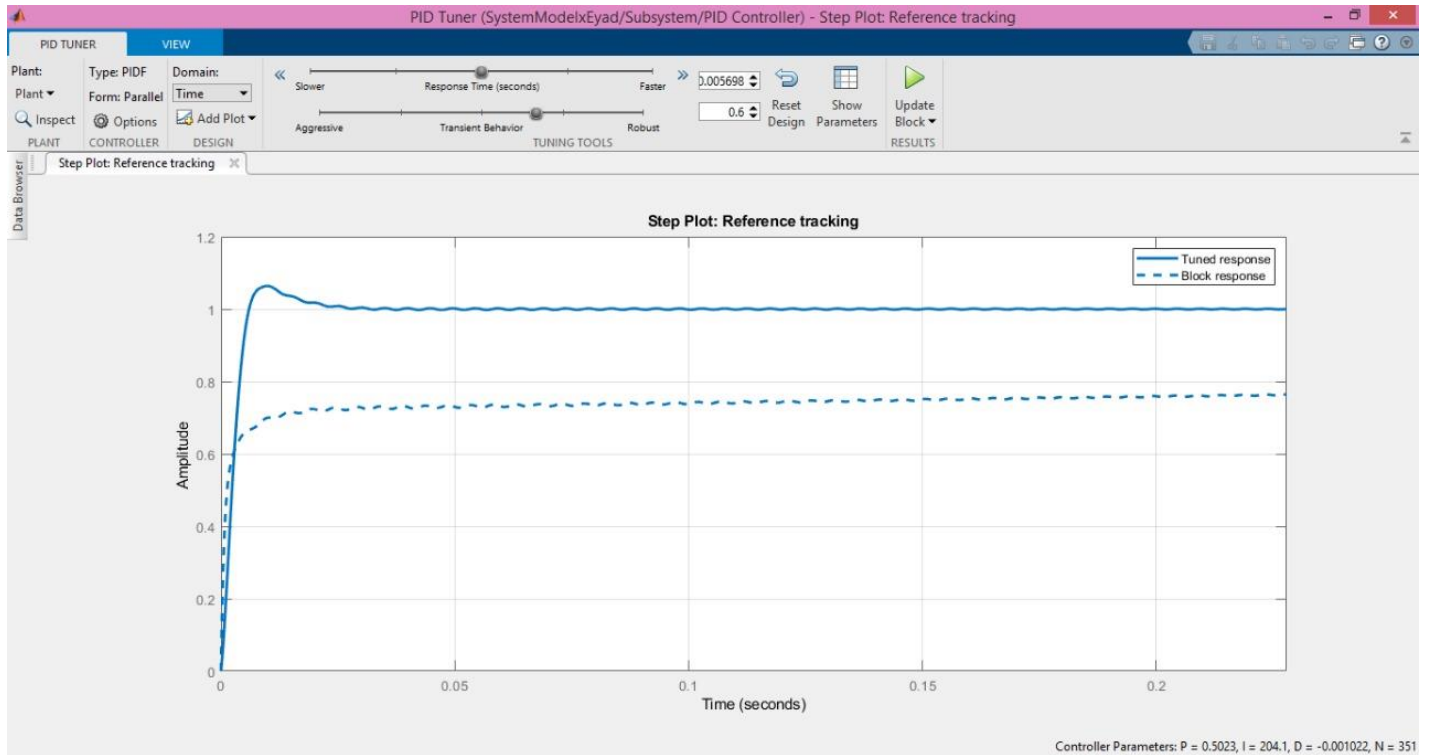


Figure 11: System response – MATLAB tuned PID parameters

3.5.2. System response - $K_p=0.125$, $K_i=12.753$, $K_d=0$

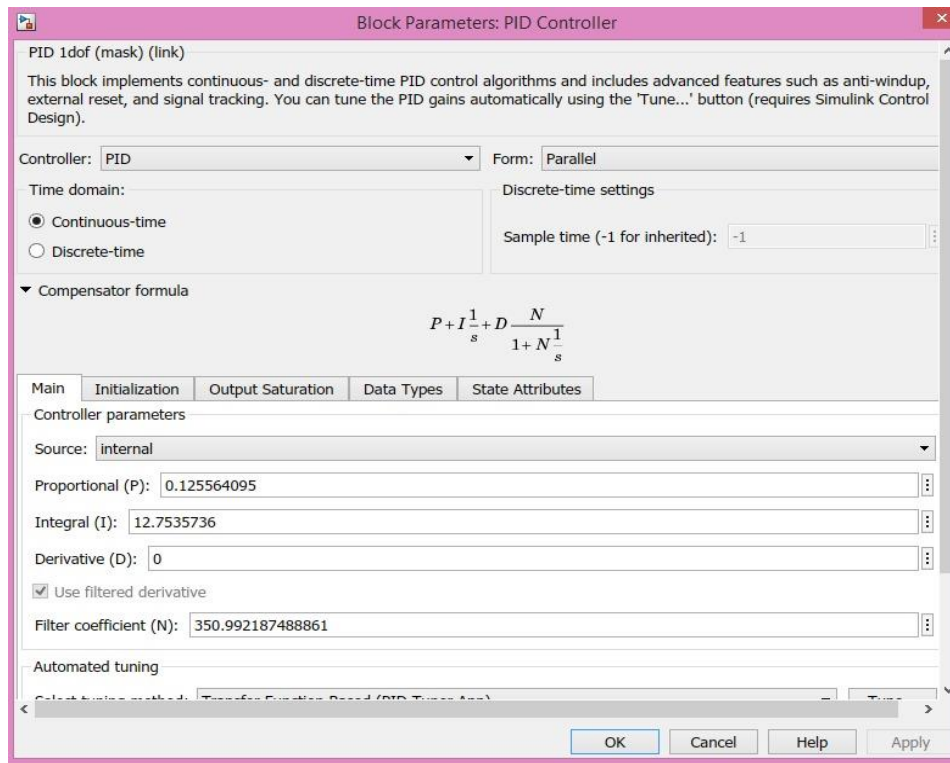


Figure 12 | $K_p=0.125$, $K_i=12.753$, $K_d=0$

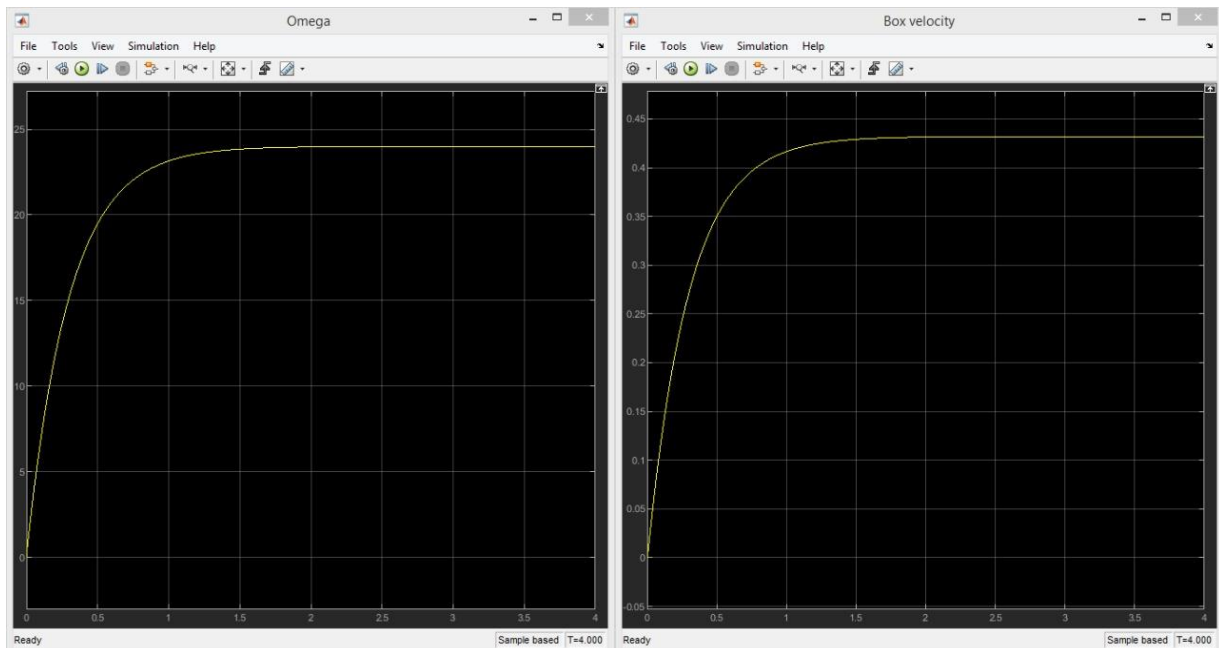


Figure 13 | System response - $K_p=0.125$, $K_i=12.753$, $K_d=0$.

3.5.3. System response - $K_p=0.01, K_i=1.3, K_d=0$

This combination of parameters gave the least settling time with no overshoot. Thus, those are the parameters used during implementation.

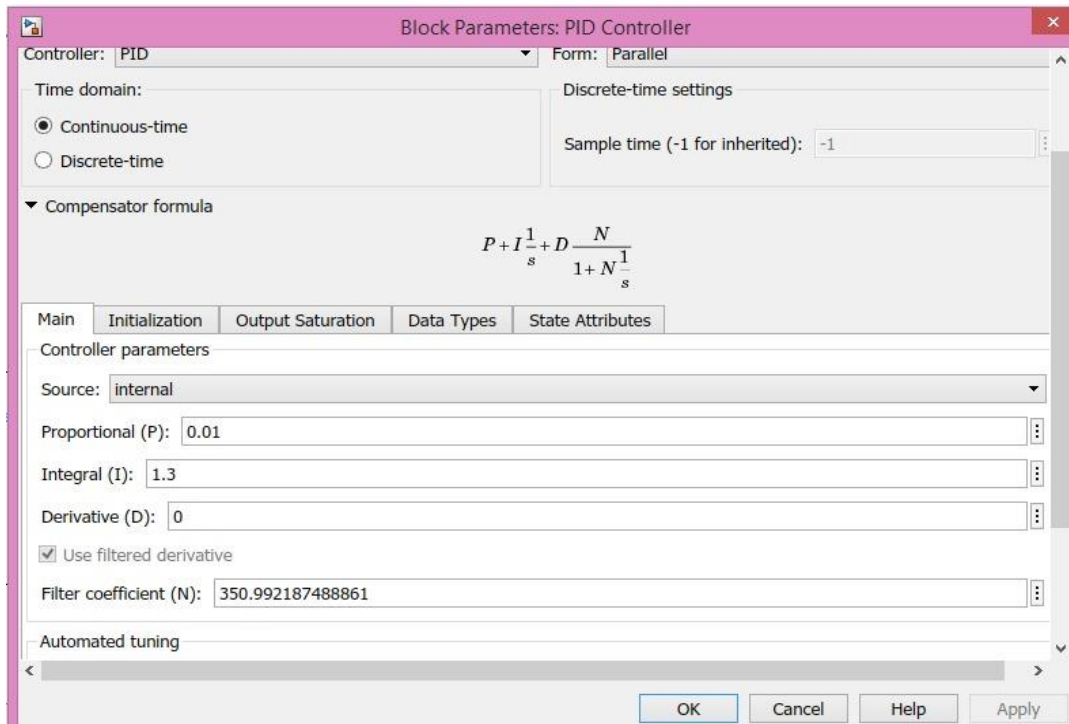


Figure 14 | $K_p=0.01, K_i=1.3, K_d=0$

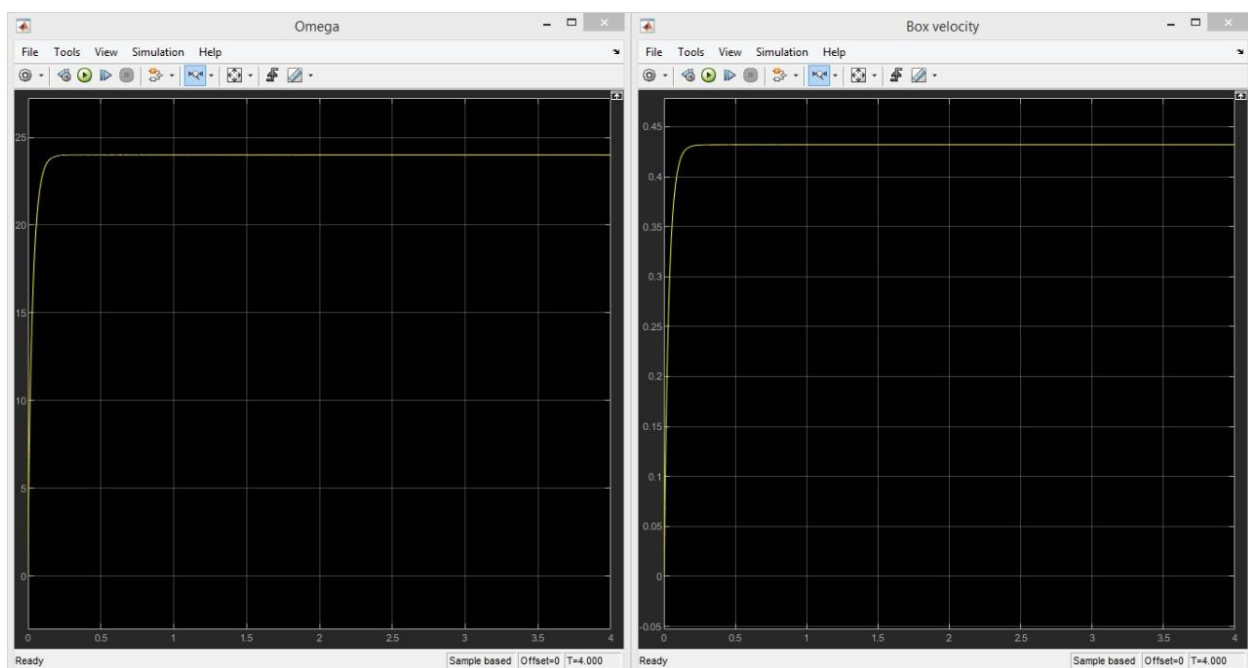


Figure 15 | System response - $K_p=0.01, K_i=1.3, K_d=0$.

3.6. System frequency response

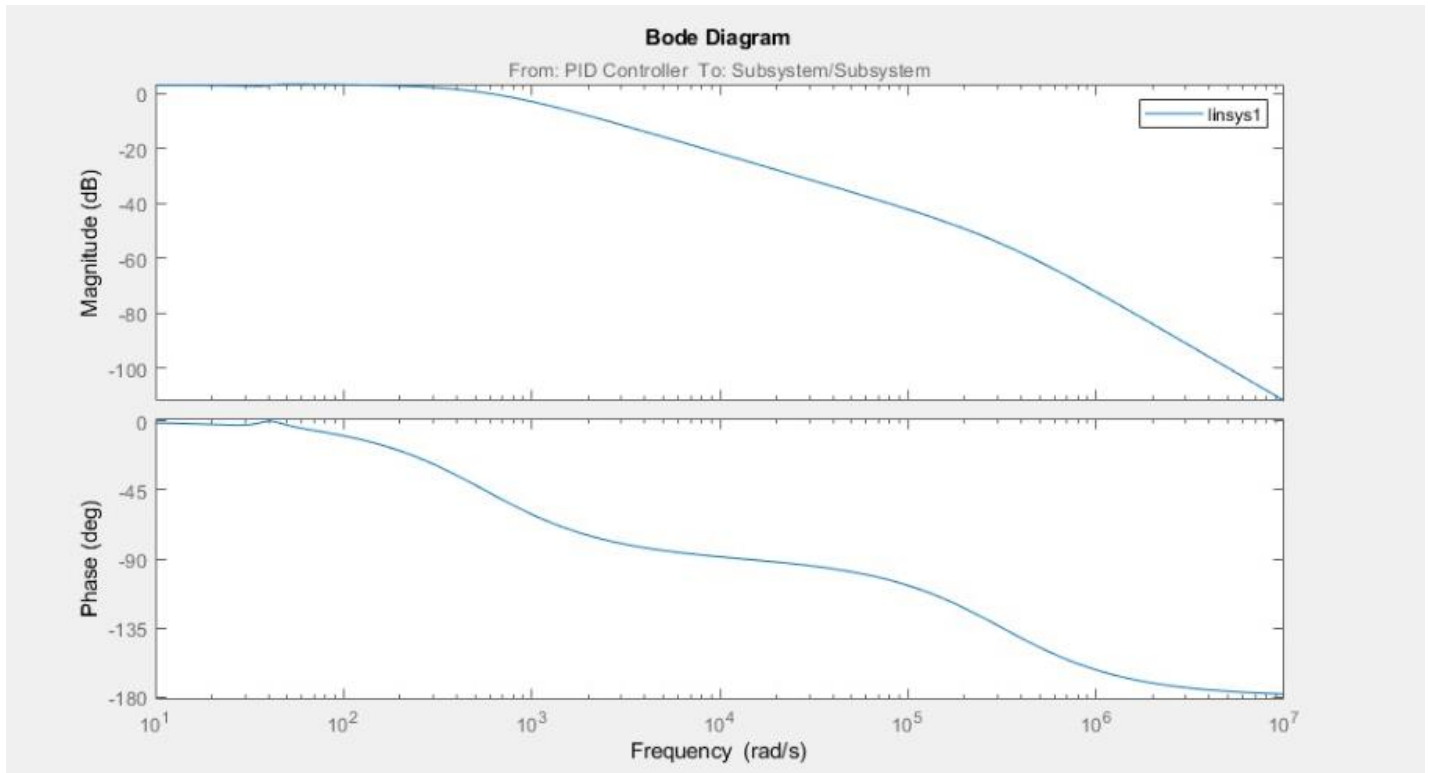


Figure 16| Bode plot

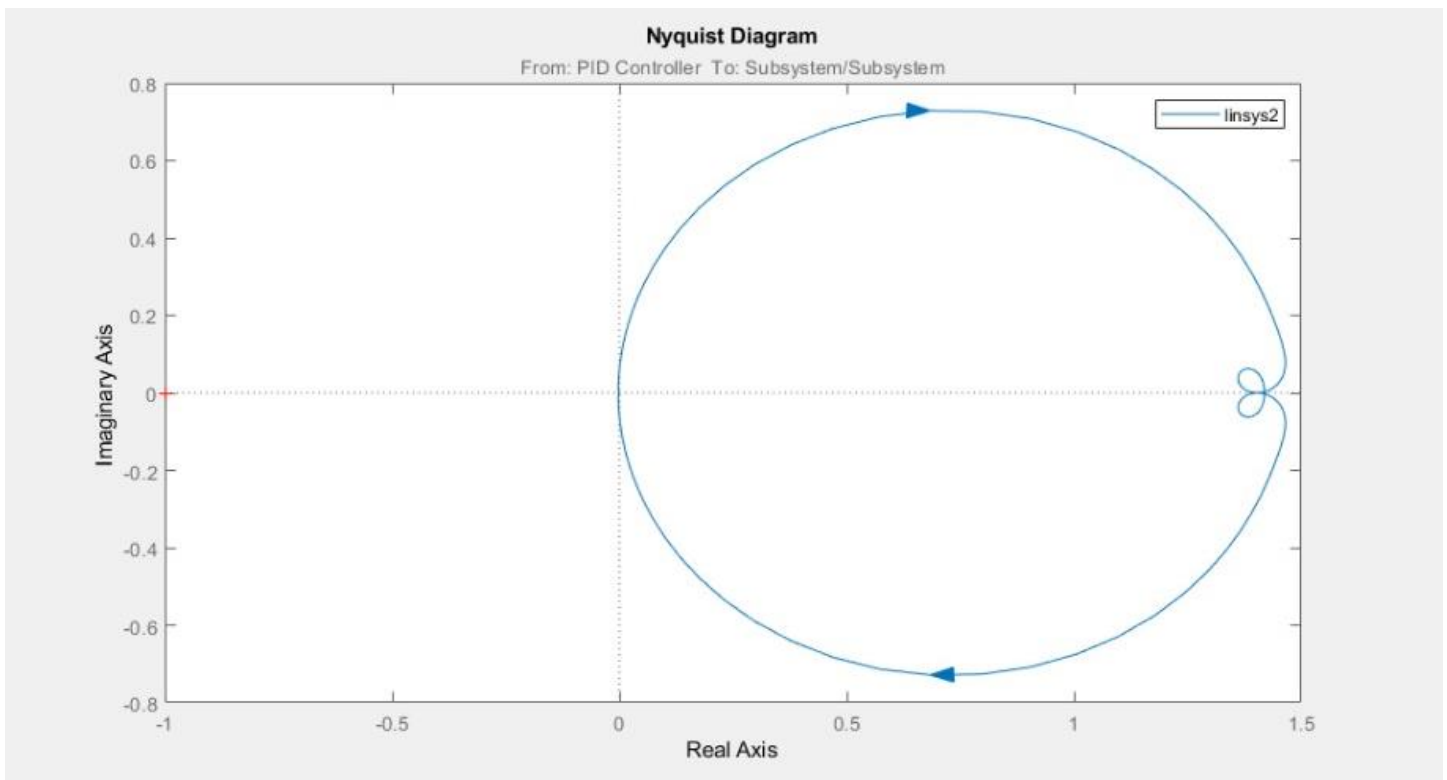


Figure 17| Nyquist Diagram

3.7. System Flowcharts

So, after modeling our system and got the predicted system outputs we started assembling the elevator and choose the controller we are going to use (**Arduino uno**). We First made an open loop system where we only took readings from our encoder -which was a wheel and an opto-coupler- to check the readings of our sensor. We then created a flow chart for the Code shown below.

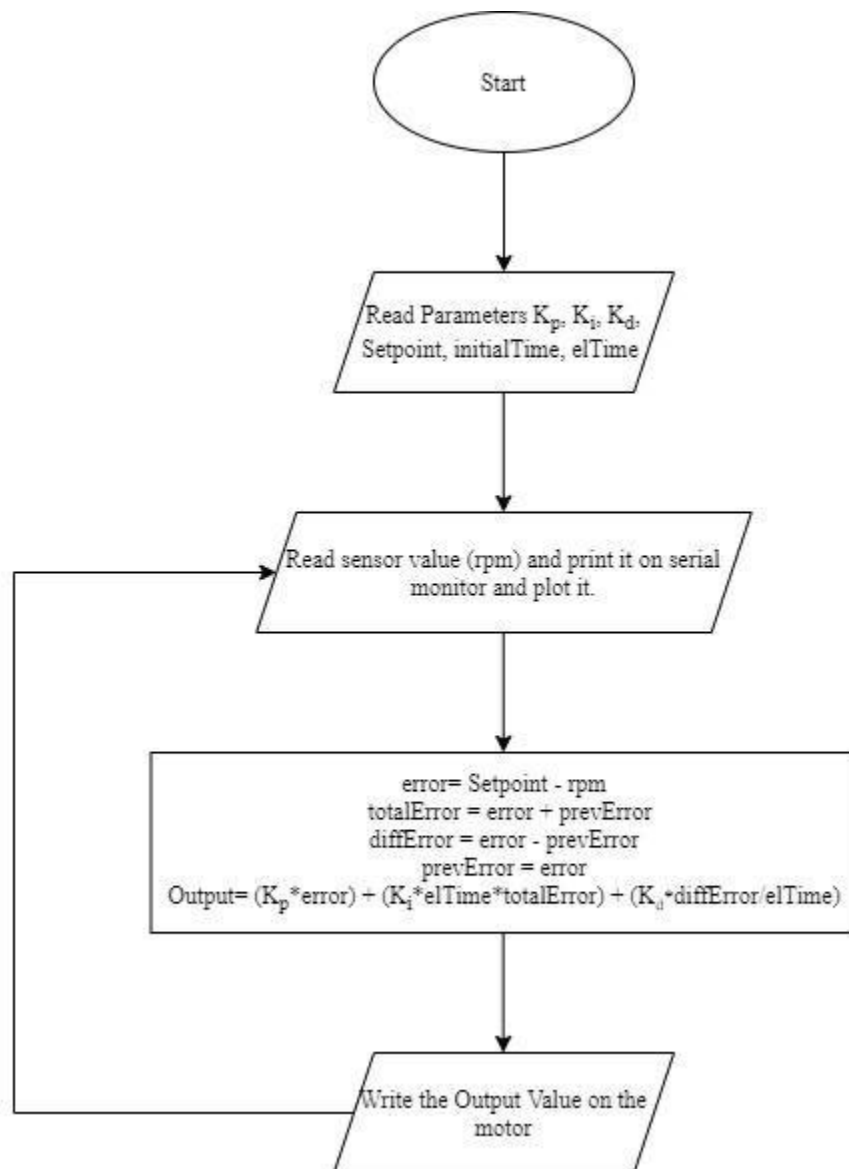


Figure 18 :PID control code Flowchart

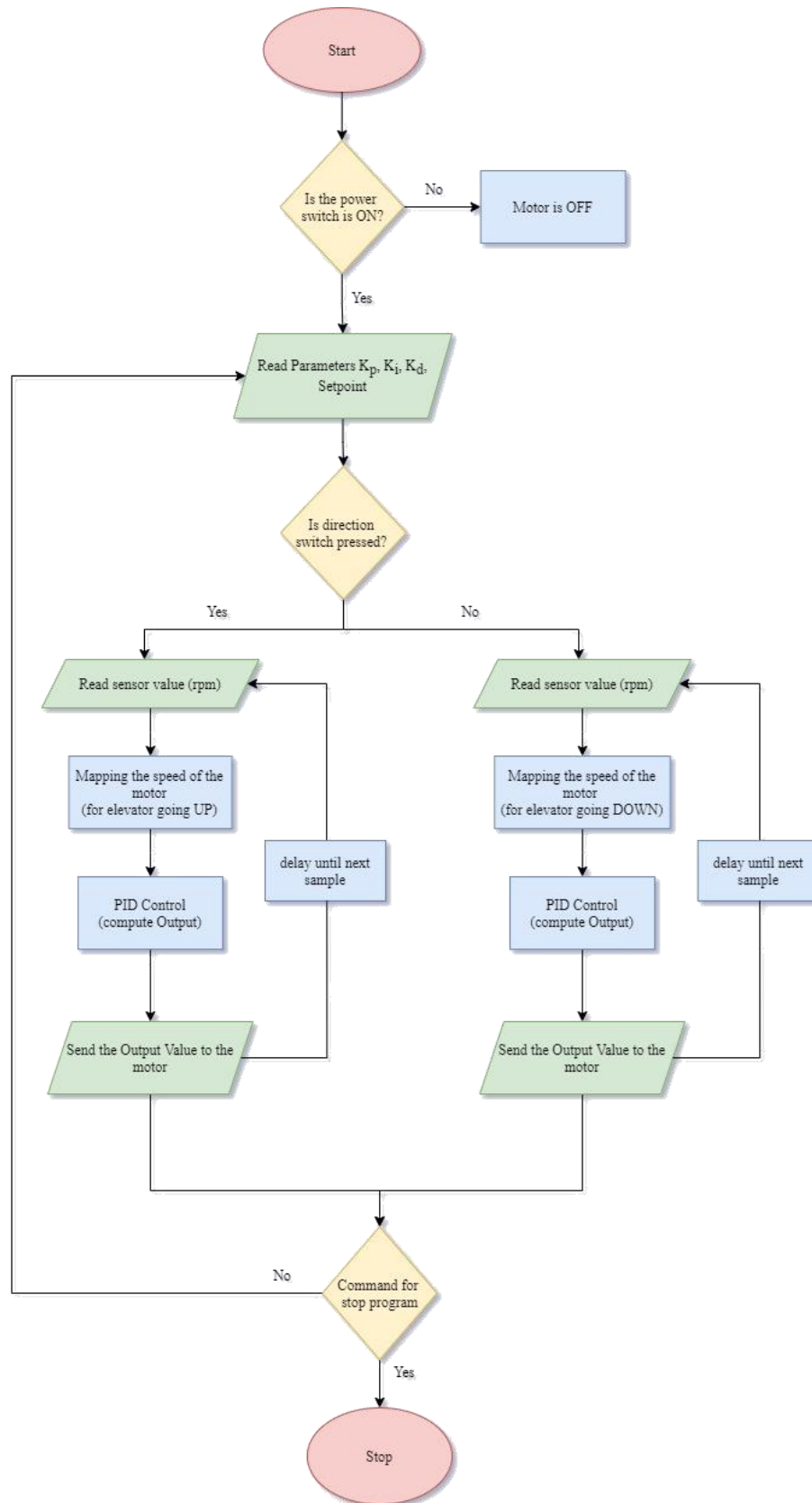


Figure 19: Code flowchart

3.8. Code

```
#include <PID_v1.h>
#include<TimerOne.h>
int opt = 2;
int m1 = 7;
int m2 = 6;
int ena = 3;
int power=8;
int direct_ion =9;
int initialTime = 0;
int elTime = 0;
volatile double count;
double limit;
//int period = 100;
double rpm = 0;
double Setpoint, Output;
double Kp =0.01, Ki=1.3, Kd =0.0;
double error, totalError, diffError, pevError;
PID myPID(&rpm, &Output, &Setpoint, Kp, Ki, Kd, P_ON_M, DIRECT);

void setup() {

myPID.SetMode(AUTOMATIC);
Timer1.initialize(1000000);
Timer1.attachInterrupt(measureRPM);
attachInterrupt(digitalPinToInterrupt(opt), COUNT, CHANGE);
pinMode(opt, INPUT);
pinMode(m1, OUTPUT);
pinMode(m2, OUTPUT);
pinMode(ena, OUTPUT);
pinMode(direct_ion,INPUT_PULLUP);
pinMode(power,INPUT_PULLUP);
```

```

/*if (limit<=245&&limit>10){
myPID.SetOutputLimits((limit-10),(limit+10));
}else if(limit>245){
myPID.SetOutputLimits((limit-20),255);
}else{
myPID.SetOutputLimits((limit),(limit+30));
}*/
//Serial.print(limit);

//analogWrite(ena,255);

//analogWrite(ena,100);
Serial.begin(9600);
}

void COUNT() {
count++;
//Serial.println(count);
}
void measureRPM() {
detachInterrupt(digitalPinToInterrupt(opt));
//Serial.println(count);
rpm = (count / 24.0) * 60;
Serial.print("RPM:");
//Serial.print(0);
Serial.println(rpm);
//Serial.print(200);
//Go_up_or_down();
pidControl();
count = 0;

```

```

attachInterrupt(digitalPinToInterrupt(opt), COUNT, CHANGE);
}
void pidControl() {
myPID.Compute();
//Serial.println(Output);
analogWrite(ena, Output);
//error= Setpoint - rpm;
//totalError = error + prevError;
//diffError = error - prevError;
//prevError = error;
//if (totalError>=350){
//totalError=350;}
//if (totalError<=-350){
//totalError=-350;
//}
//Output = (Kp*error + Ki*totalError + (Kd)*diffError);
//if (Output <=255 && Output >0){
//analogWrite(ena,Output); //set motor speed
//}
//else{
//if (Output>255){
//analogWrite(ena,255);
//}
//else{
//analogWrite(ena,45);
//}}
}

/*void Go_up_or_down() {

//if going down

```

```

}  */

void loop(){

  if (digitalRead(power)==0){
    digitalWrite(m1, LOW);
    digitalWrite(m2, LOW);
  }
  if (digitalRead(power)==1 && digitalRead(direct_ion)==1){
    digitalWrite(m1, HIGH);
    digitalWrite(m2, LOW);
    Setpoint = 100;
    limit=map(Setpoint,0,430,0,255);
    myPID.SetOutputLimits(limit-7,limit+8);

    // if going up
  }
  if (digitalRead(power)==1&&digitalRead(direct_ion)==0){
    digitalWrite(m2, HIGH);
    digitalWrite(m1, LOW);
    Setpoint = 100;
    limit=map(Setpoint,0,880,0,255);
    myPID.SetOutputLimits(limit-8,limit+8);

  }
}
//}

```

4. IMPLEMENTATION

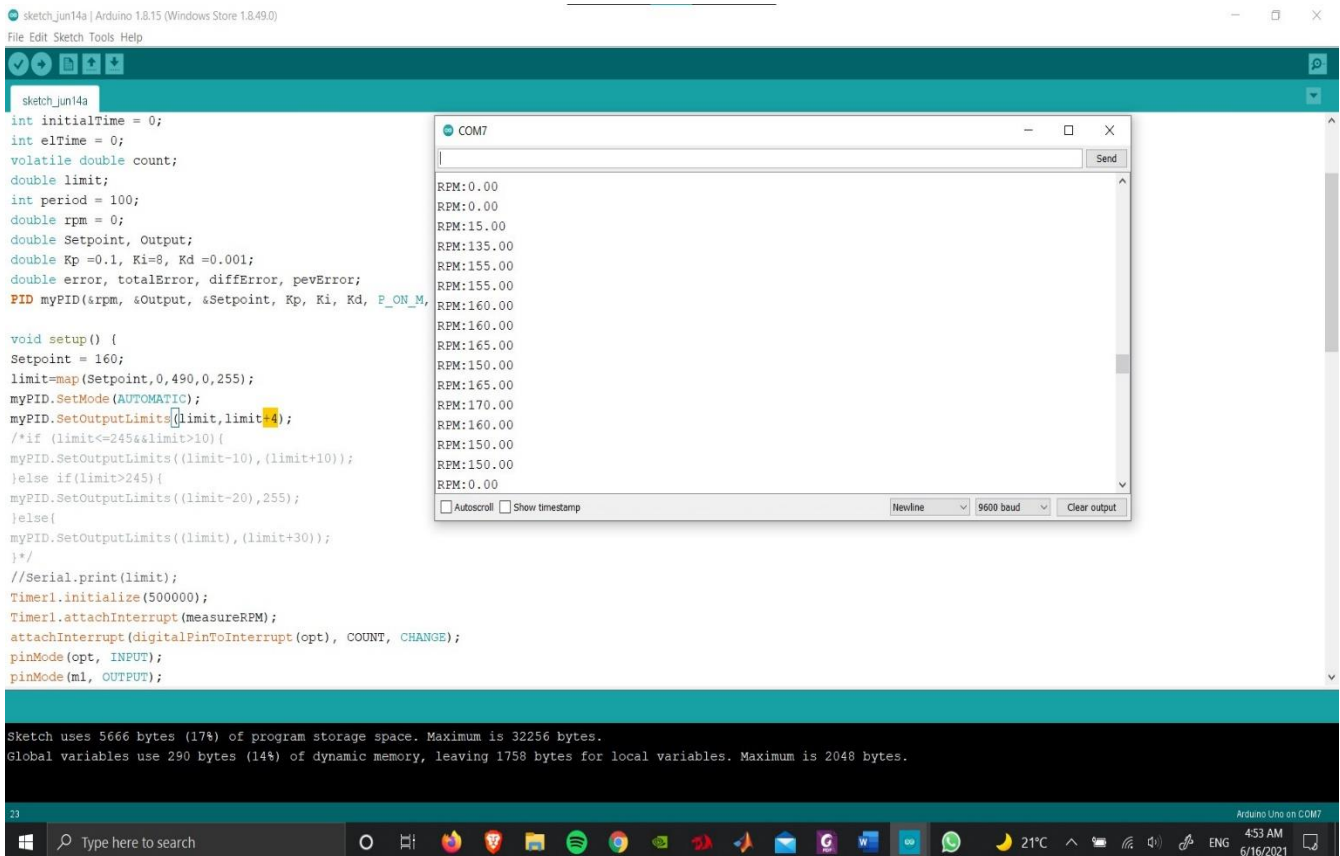
We did not get the same response when we used the **PID** tuning parameters that MATLAB generated. Due to the error in defining the motor characteristics and because the motor we used did not have a proper Data sheet, we started trial and error method in calculating the tuning parameters (K_i , K_p , K_d). At the end of this process, we achieved a set of parameters that gave us the best response among other many tries. The values of parameters were:

$K_p = 0.01$, $K_i = 1.3$, $K_d = 0$

4.1. Practical system response at different sets of parameters

- $K_p = 0.1$ $K_i = 8$ $K_d = 0$

This set of values was not chosen.



```
sketch_jun14a | Arduino 1.8.15 (Windows Store 1.8.49.0)
File Edit Sketch Tools Help

sketch_jun14a
int initialTime = 0;
int elTime = 0;
volatile double count;
double limit;
int period = 100;
double rpm = 0;
double Setpoint, output;
double Kp = 0.1, Ki=8, Kd =0.001;
double error, totalError, diffError, pevError;
PID myPID(&rpm, &output, &Setpoint, Kp, Ki, Kd, P_ON_M,

void setup() {
  Setpoint = 160;
  limit = map(Setpoint, 0, 490, 0, 255);
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(0, limit);
  /*if (limit<=245&&limit>10){
  myPID.SetOutputLimits((limit-10), (limit+10));
  }else if(limit>245){
  myPID.SetOutputLimits((limit-20), 255);
  }else{
  myPID.SetOutputLimits((limit), (limit+30));
  }*/
  //Serial.print(limit);
  Timer1.initialize(500000);
  Timer1.attachInterrupt(measureRPM);
  attachInterrupt(digitalPinToInterrupt(opt), COUNT, CHANGE);
  pinMode(opt, INPUT);
  pinMode(m1, OUTPUT);

Sketch uses 5666 bytes (17%) of program storage space. Maximum is 32256 bytes.
Global variables use 290 bytes (14%) of dynamic memory, leaving 1758 bytes for local variables. Maximum is 2048 bytes.

COM7
RPM:0.00
RPM:0.00
RPM:15.00
RPM:135.00
RPM:155.00
RPM:155.00
RPM:160.00
RPM:160.00
RPM:165.00
RPM:150.00
RPM:165.00
RPM:170.00
RPM:160.00
RPM:150.00
RPM:150.00
RPM:0.00
```

Figure 20| Setpoint 160 RPM

- $K_p = 0.1$ $K_i = 5$ $K_d = 0$

This set of values was not chosen.

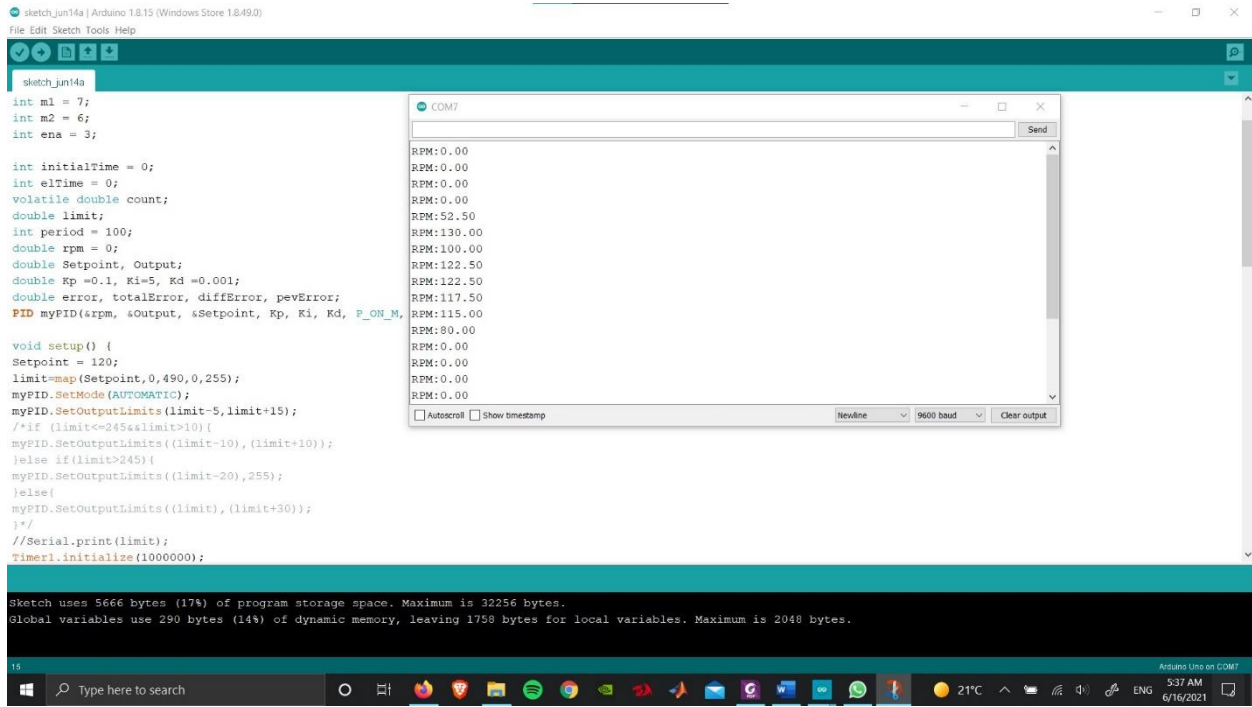


Figure 21| Measured RPM - Setpoint 120 RPM

- $K_p = 0$ $K_i = 5$ $K_d = 0$

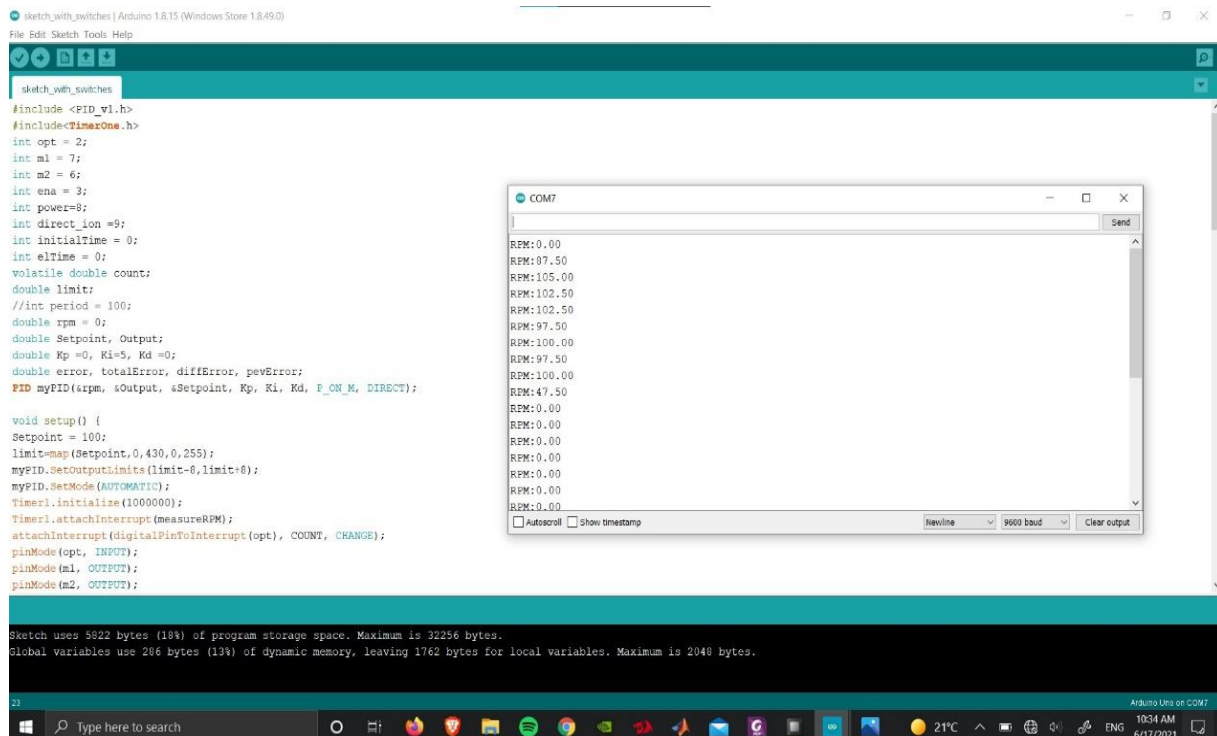
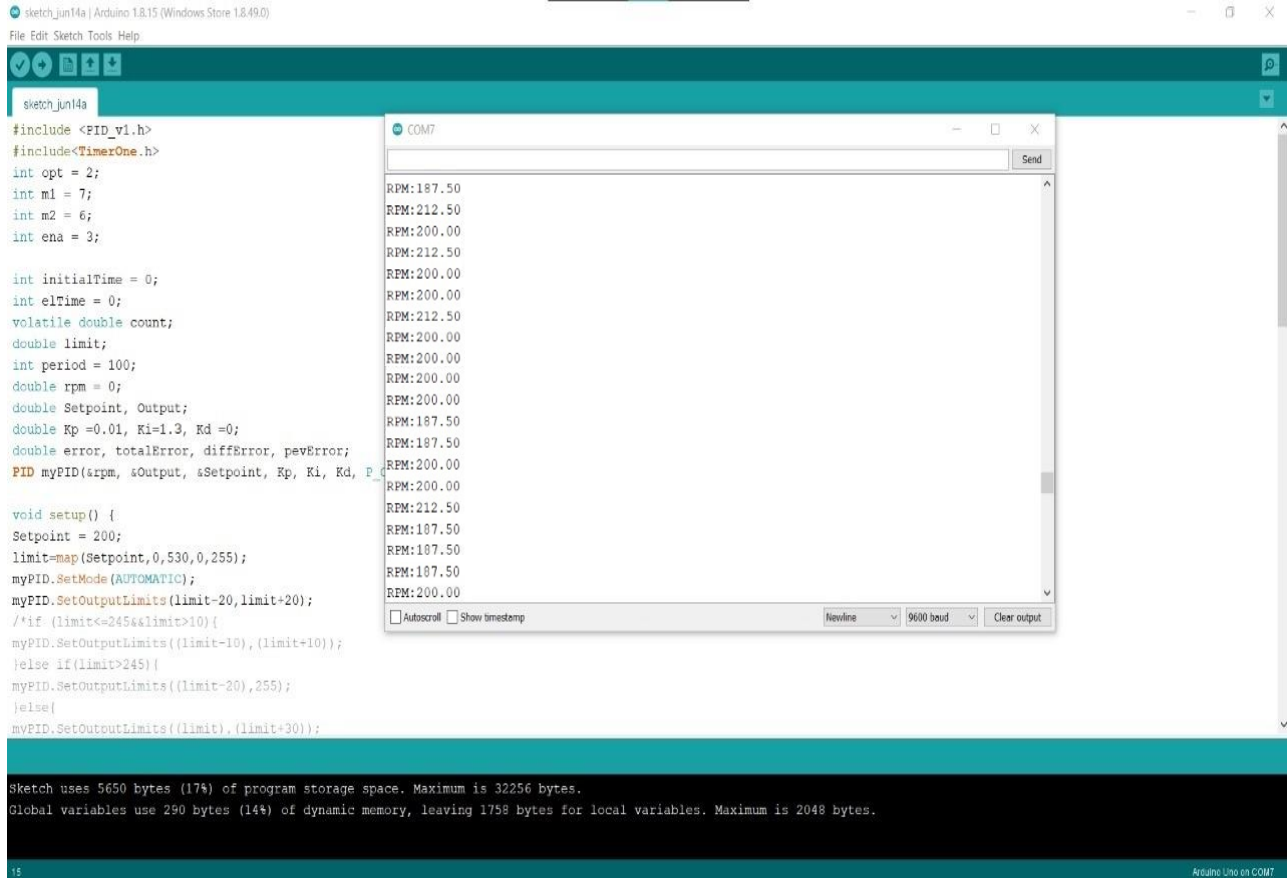


Figure 22| Measured RPM - Setpoint 100

4.2. Practical system response with the chosen parameters

- $K_p = 0.01$ $K_i = 1.3$ $K_d = 0$



```
sketch_jun14a | Arduino 1.8.15 (Windows Store 1.8.49.0)
File Edit Sketch Tools Help

sketch_jun14a
#include <PID_v1.h>
#include<TimerOne.h>
int opt = 2;
int m1 = 7;
int m2 = 6;
int ena = 3;

int initialTime = 0;
int elTime = 0;
volatile double count;
double limit;
int period = 100;
double rpm = 0;
double Setpoint, Output;
double Kp =0.01, Ki=1.3, Kd =0;
double error, totalError, diffError, pevError;
PID myPID(&rpm, &Output, &Setpoint, Kp, Ki, Kd, P, I, D);

void setup() {
  Setpoint = 200;
  limit=map(Setpoint,0,530,0,255);
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(limit-20, limit+20);
  /*if (limit<=245&&limit>10){
  myPID.SetOutputLimits((limit-10), (limit+10));
  }else if(limit>245){
  myPID.SetOutputLimits((limit-20),255);
  }else{
  myPID.SetOutputLimits((limit), (limit+30));
  }

Sketch uses 5650 bytes (17%) of program storage space. Maximum is 32256 bytes.
Global variables use 290 bytes (14%) of dynamic memory, leaving 1758 bytes for local variables. Maximum is 2048 bytes.
```

COM7

```
RPM:187.50
RPM:212.50
RPM:200.00
RPM:212.50
RPM:200.00
RPM:200.00
RPM:212.50
RPM:200.00
RPM:200.00
RPM:200.00
RPM:200.00
RPM:200.00
RPM:187.50
RPM:187.50
RPM:200.00
RPM:200.00
RPM:212.50
RPM:187.50
RPM:187.50
RPM:200.00
```

Autoscroll Show timestamp Newline 9600 baud Clear output

Figure 23 | Measured RPM - Setpoint 200

5. WORKING VIDEO

<https://drive.google.com/file/d/1VK-6wsE0aRFgVy0nBOUNfYwuoefZt40m/view?usp=sharing>

6. PRESENTATION

<https://drive.google.com/file/d/1bq1Srd-rgn8OYRDJLiP-nTb6rdPv0eU0/view?usp=sharing>