Course No: EEE 318

Project Name:

Controller Design of Levitated Copter-Arm

Submitted to:

Dr. Mohammad Ariful Haque
Professor
Department of EEE
BUET

Towsif Taher
Lecturer
Department of EEE
BUET

Prepared by:

Mahmudul Islam **1506155**
Md. Suhanur Rahman **1506156**
Tanzir Ahmed Sami **1506157**

Level: 3 Term: 2

Section: C (1)

Group 09

Department of Electrical and Electronic Engineering

Date of Submission: 03-02-2018

FORWARDING LETTER

2nd February, 2019

To,
Dr. Mohammad Ariful Haque
Professor,
Towsif Taher
Lecturer,

Department of EEE,

BUET

SUBJECT: Report on 'Controller design of levitated copter-arm'

Dear Teachers,

We are so enthusiastic to present our report on the topic 'Controller design of levitated arm' on this occasion we would like to express our gratitude for your kind encouragement and guideline.

We apologize for any kind of mistake we have made in this report despite our best effort. We have tried our best to shade some light upon the project. If this effort helps in any way to understand a controller design for levitated arm, we will feel honored & delighted.

Yours sincerely,

Mahmudul Islam 1506155
Md. Suhanur Rahman 1506156
Tanzir Ahmed Sami 1506157

Abstract

Intelligent controllers are used everywhere, from a cruise control on a car to an air conditioning system. They are primarily used to provide corrective actions to maintain the stability of the system and produce an output close to the reference point selected by the user. There are many challenges related to the development of a controller. The type of the controller needed, the hardware components that will be used, the system identification technique that will be used to identify the mathematical model of the process and the tuning method are some of the considerations that should be made. All these considerations are part of this project. The aim of this project is to design and implement an intelligent, efficient and accurate P, PD, PID (Proportional, Integral, Derivative) controller of levitated arm. The purpose of this report is to describe the P, PD PID controller's. It provides an insight into the background information related to P, PD,PID controller.

TABLE OF CONTENTS

Introduction:

Control systems date back to ancient times. The first feedback control systems are believed to be "water clocks" used to keep time over 1000 years ago by regulating the liquid flow into or out of a vessel. Control systems are used to manipulate the state of the controlled process without the need for constant input or corrections by the user. Major progress in the development of feedback control systems appeared in the first decades of the last century, following the discovery of the Laplace and Fourier transforms.

Characteristics of a good control system are:
- Stability, the system should not oscillate.
- Fast response/rise time, the system should reach the reference point as fast as possible.
- Zero steady-state error, when the system is stable the difference between the output of the system and the reference output should be close to zero.
- Tracking of the reference point, the output should be as close to the reference point as possible.

Aim and Objectives

The main aim of this project is to design and develop a copter-levitated with a dc motor and an intelligent controller based on python and arduino platform that can be used to control the angle of the levitated arm

Objectives of the project include:

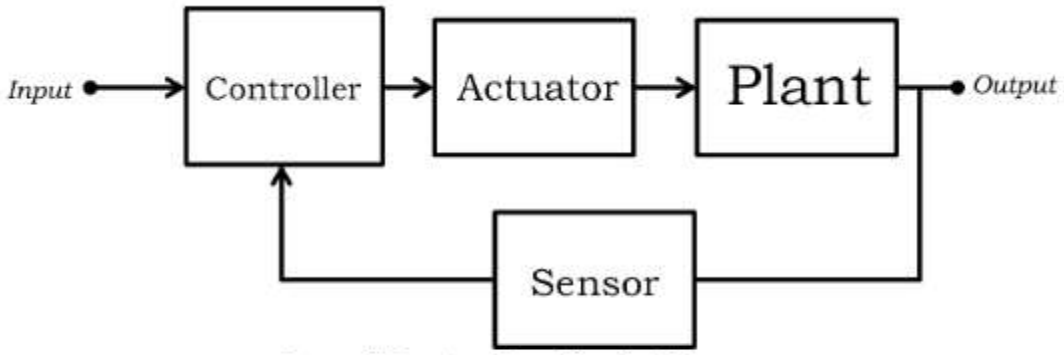
- Assemble the required hardware components including the arduino uno, motor driver, dc motor.
- Develop a mathematical model for the P, PD, and PID controller.
- Get good insight of the controller (P, PD, PID controller)
- Evaluation of the effectiveness of the assembled system.

Background

This chapter presents the principles used to design and develop the PID controller. The first section contains information about control theory, the fundamentals behind the design of every controller. Subjects such as open and closed loop control systems, components of the controlled system in this chapter. The second section contains information about P, PD and PID controllers as well as PID tuning. The third and last section describes the arduino platform and the other hardware components that are used in the project. The programming language used in this project is also described.

2.1 Control Theory

Control theory is a branch of mathematics and engineering with the purpose of modifying the behavior of a dynamic system with inputs (e.g. input from a temperature sensor) that changes over time. The main objective of control theory is stability. Usually (in closed loop systems) stability is achieved by continually taking measurements and making adjustments to the system, this procedure forms a loop. Usually a loop in control theory consists of: i) the referenced value or set point (the desired output, provided by the user or some other part of a larger system), ii) the controller, iii) the process (e.g. the heater) and iv) sensors. Systems are divided in two basic categories in terms of control structure, open-loop systems and closed-loop systems. In a closed-loop system, the output of the process is measured and subtracted from the reference value, producing the error value. The error value is then passed to the controller, which will then make various calculations and modify the process input accordingly, producing a new process output. The procedure is repeated in a continuous manner. In an open-loop system the output is not fed back to the controller, thus it cannot influence the input. Closed-loop is generally preferred as they react to disturbances better, can stabilize inherently unstable systems and track the reference value better. Figure 2.1 shows the structure of a closed-loop system.



Overall System Feedback Diagram::

2.2 Continuous time and discrete time in control theory

In the design of control systems both continuous and discrete time can be used. In discrete time, there are a finite number of points in a time interval. Each point is distinct and represents a uniform step in time from the previous point. In discrete time, each measurement is taken in a specific, distinct point in time and nothing can be assumed about the time interval between two distinct points. A common example of discrete time is a digital clock. In a digital clock, time is represented as hh:mm:ss and after a uniform step in time the representation of time changes. Discrete time is often used together with a digital device, such as a microcontroller. As a result during the implementation of a controller in a digital device, discrete time must be taken under consideration. In continuous time, there are infinite points between two different points in time. Changes in the environment are considered to happen continuously and not just in distinct points in time. Continuous time is usually employed when talking about the theory of a controller. For example, a transfer function in the s domain is expressed in continuous time. An experiment runs in continuous time but is usually sampled, i.e. taking measurements in specific points in time, so that it can be represented in a digital computer.

2.3 Sequences

Sequences can be considered to be a set of values indexed by an integer. Such sequences are useful for representing discrete-time data in a variety of applications, including digital audio, stock prices, bank balances, and control applications. For example, the Arduino-based controller for your copter-levitated arm has three sequences: two sequences of samples (the analog inputs from the propeller-motor and angle-sensor) and a sequence of outputs (the analog output to the motor driver transistor).

We will use, and translate between, three representations for sequences: *explicit enumeration*, *closed-form expression*, and implicit definition using *difference equations* and *initial conditions*. (Difference equations are a special case of the recurrences commonly studied in discrete mathematics.) As an example, let y be the sequence of powers of 1/2. The sequence can be represented by

$$y[n] = (1/2)^n, \; n>=0$$

2.4 Linear difference equation (LDE)

In mathematics and in particular dynamical systems, a linear difference equation or linear recurrence relation sets equal to 0 a polynomial that is linear in the various iterates of a variable—that is, in the values of the elements of a sequence. The polynomial's linearity means that each of its terms has degree 0 or 1. Usually the context is the evolution of some variable over time, with the current time period or discrete moment in time denoted as *t*, one period earlier denoted as *t*–1, one period later as *t*+1, etc.

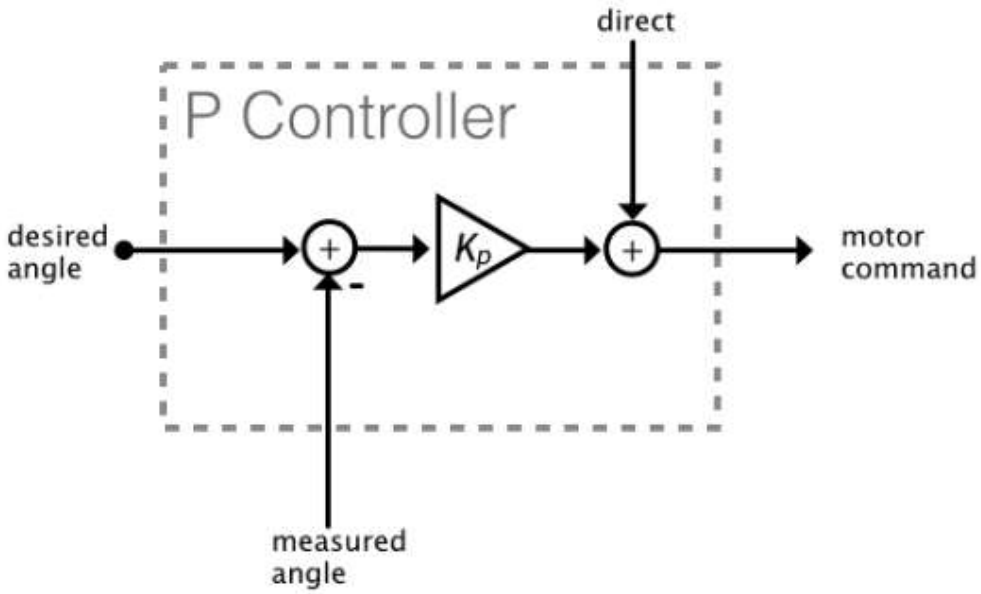
$$y[n] = \lambda\, y[n-1] \text{ or } y[n] - \lambda\, y[n-1] = 0$$

λ, the natural frequency of a first-order LDE, or equivalently governs the *evolution* of its associated sequences. In particular,

- If **λ >1**, the sequence grows monotonically and without bound, and we say that the first-order LDE is ***unstable*.**
- If **0<λ<1**, the sequence decreases monotonically to zero, and we say that the first-order LDE is ***stable***.

2.5 Proportional (P) Controller

Proportional controller is mostly used in first order processes with single energy storage to stabilize the unstable process. The main usage of the P controller is to decrease the steady state error of the system. As the proportional gain factor Kp increases, the steady state error of the system decreases. However, despite the reduction, P control can never manage to eliminate the steady state error of the system. As we increase the proportional gain, it provides smaller amplitude and phase margin, faster dynamics satisfying wider frequency band and larger sensitivity to the noise. We can use this controller only when our system is tolerable to a constant steady state error. In addition, it can be easily concluded that applying P controller decreases the rise time and after a certain value of reduction on the steady state error, increasing K only leads to overshoot of the system response. P control also causes oscillation if sufficiently aggressive in the presence of lags and/or dead time. The more lags (higher order), the more problem it leads. Plus, it directly amplifies process noise.

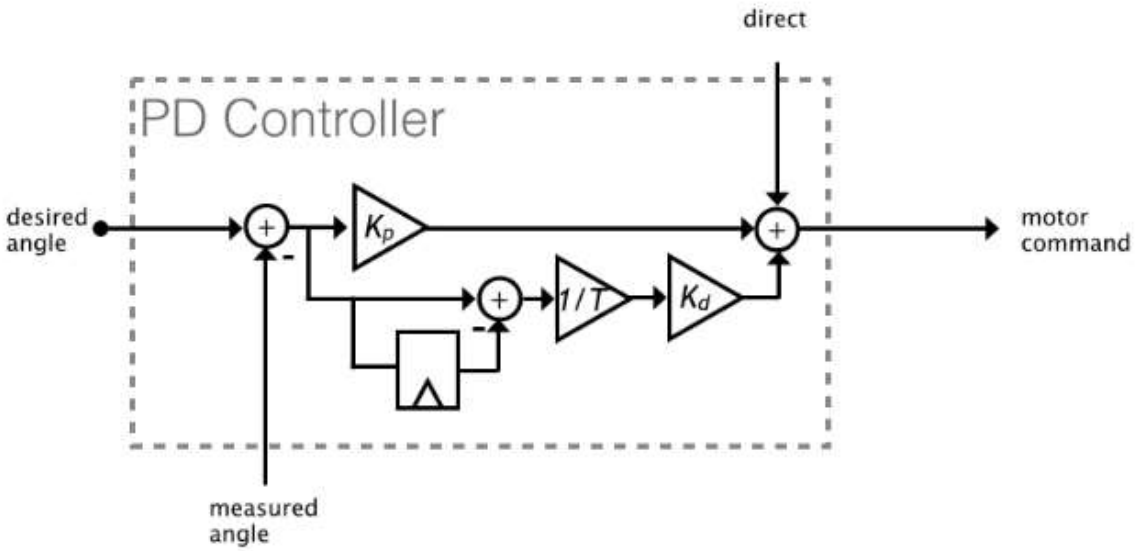


Overall System Feedback Diagram::

2.6 Proportional – Derivative (PD) Controller:

The control actions of the proportional is based on the current error or past errors. In derivative control the controller output is proportional to the *rate of change* of the error. The idea behind derivative control is that the controller should react immediately to a large change in the control error; in essence, predicting that the error will continue to increase (or decrease) and act accordingly. Although this quick reaction can result in fast response times, it can also result in undesirable overreaction, especially if the system output has significant stochastic.

The aim of using P-D controller is to increase the stability of the system by improving control since it has an ability to predict the future error of the system response. In order to avoid effects of the sudden change in the value of the error signal, the derivative is taken from the output response of the system variable instead of the error signal. Therefore, D mode is designed to be proportional to the change of the output variable to prevent the sudden changes occurring in the control output resulting from sudden changes in the error signal.



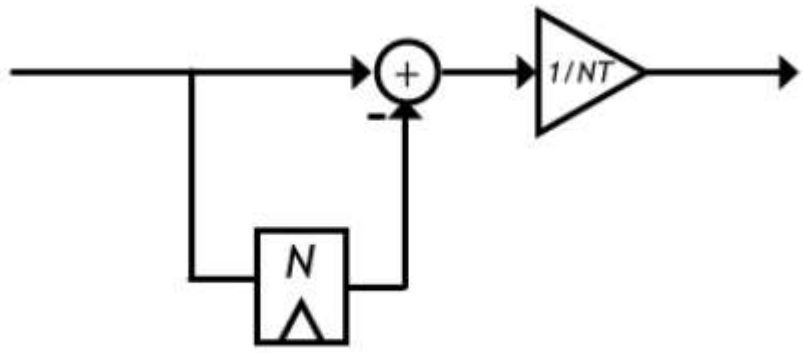
Proportional + Derivative Control::

That new mess of of boxes and arrows is a *discrete time differentiator*. Basically it finds the rate of change of the error (or what is often called the **delta or derivative**).

$$\text{Delta} = \frac{change\ in\ e}{change\ in\ time}$$

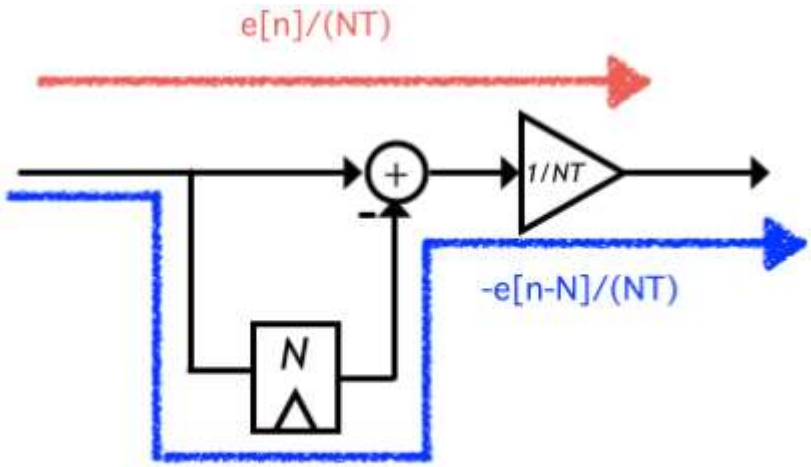
$$\text{Or, Delta} = \frac{e[n]-e[n-N]}{N*T}$$

This is implemented in code by remembering the error on the last programming loop and comparing it to the current error. In block diagram form it looks like this:



Derivative::

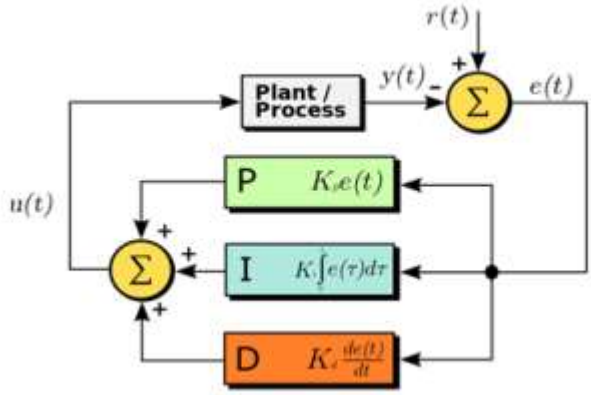
And the final form can be seen in the flow if signals below:



Signal Flow in a Differentiator::

2.7 Proportional-Integral-Derivative (PID) Controller:

The most used type of closed-loop controller architecture and the one that is employed in this project is PID. PID controllers, if tuned correctly, can increase the stability of the system, reduce the response time needed to reach the reference value and reduce the steady state error to zero. A PID controller, as its name suggest, can be tuned using 3 variables P, I and D. These 3 values are then added to provide the process input. Figure 2.2 shows the structure of a PID controller.
The P value compensates for the present error and is just a multiplication of the proportional gain K_p multiplied by *e(t)* (error on time t). The I value compensates for the errors in the past, is calculated by multiplying the internal gain K_i and the integration of *e(t)*. The D value is a prediction for errors in the future, based on the current rate of errors and is calculated by multiplying the derivative gain K_d and the differentiation of *e(t)*



Generally a large P value makes the system more responsive but it can also make it oscillate if its value is too large, the correct I value will help the system reach the reference value faster and the correct D value will make the system more stable. Depending on the requirements of a project, a controller can be designed to use only some of these values.

Hardware components:

Arduino Uno R3

Arduino Uno is a microcontroller board based on the ATmega328P. 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.

L293D Motor Driver module

L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.

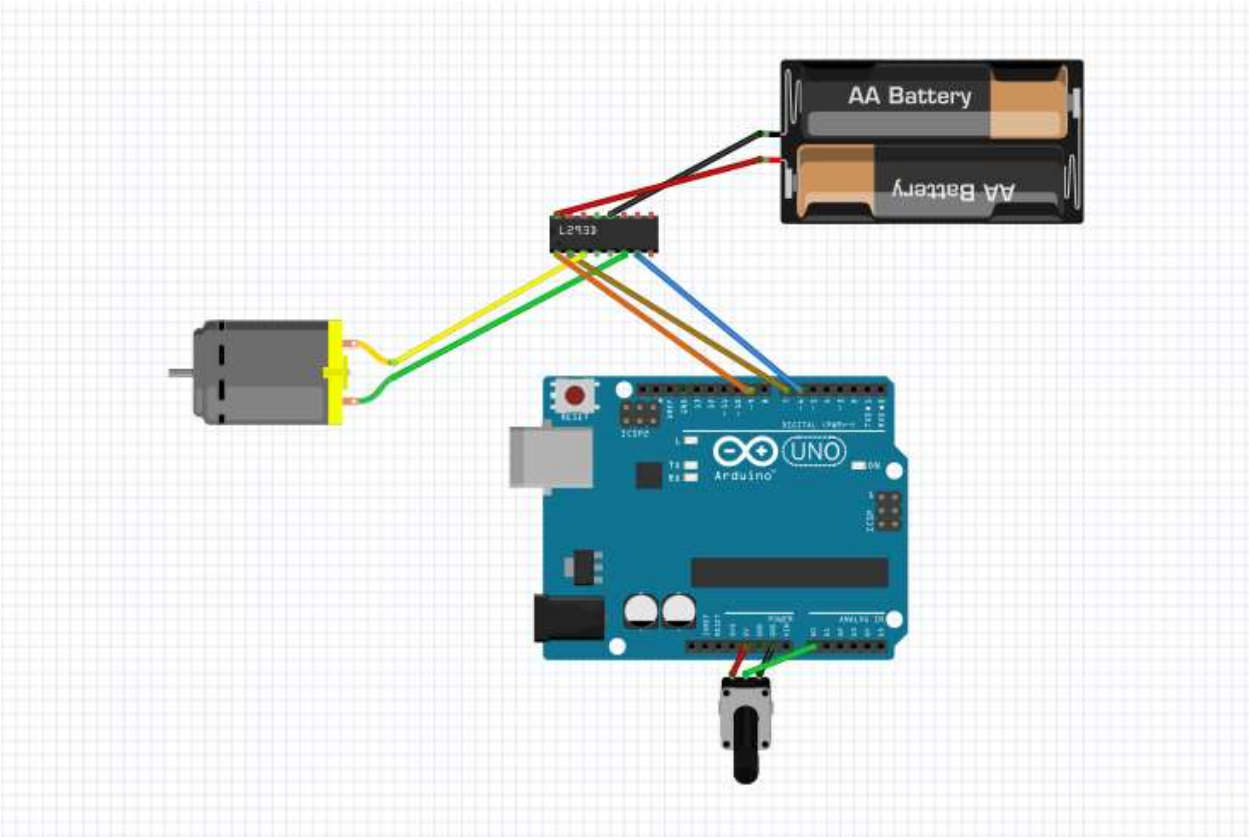
L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

Enable pins 1 and 9 (corresponding to the two motors) must be high for motors to start operating. When an enable input is high, the associated driver gets enabled. As a result, the outputs become active and work in phase with their inputs. Similarly, when the enable input is low, that driver is disabled, and their outputs are off and in the high-impedance state.

Potentiometer (10K)

Coreless DC motor

Circuit Diagram of the controller:



Required Software:

- The Arduino interfaces with the propeller arm by sending pulse-width modulated (pwm) signals to the transistor motor driver and by reading the angle sensor and motor coil voltages. Its pwm output signal is derived from sensor and motor readings in a feedback control loop (that you will modify). The control loop's behavior is adjusted by values/parameters sent to it from a python-based local server running on your computer
using serial communication. In addition, at regular intervals, the Arduino sends its measurements back to the Python server.

- The python-based local server running on your computer is an interface. It communicates with the Arduino via serial link, and with a Browser-based GUI via WebSockets.

- The Browser-based GUI plots measured data from the Arduino, passed to it by the local server, and provides simple interface for users to change the Arduino feedback loop parameters, which it passes on to the local server.

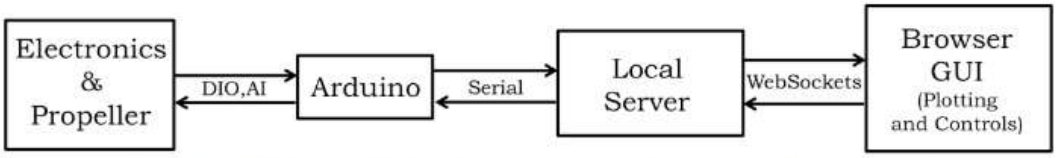


Figure 47: The basic workflow of our software for controlling and analyzing things!

Parts of the Software

The entire package of software is broken down into three pieces:

- **Microcontroller Code:** This is the code uploaded to the Arduino (the .ino file). It sets the Arduino up for listening to and controlling the electronics as well as taking in parameters and reporting values to your computer over serial (the USB cable). It is written in Arduino's C/C++ type language.

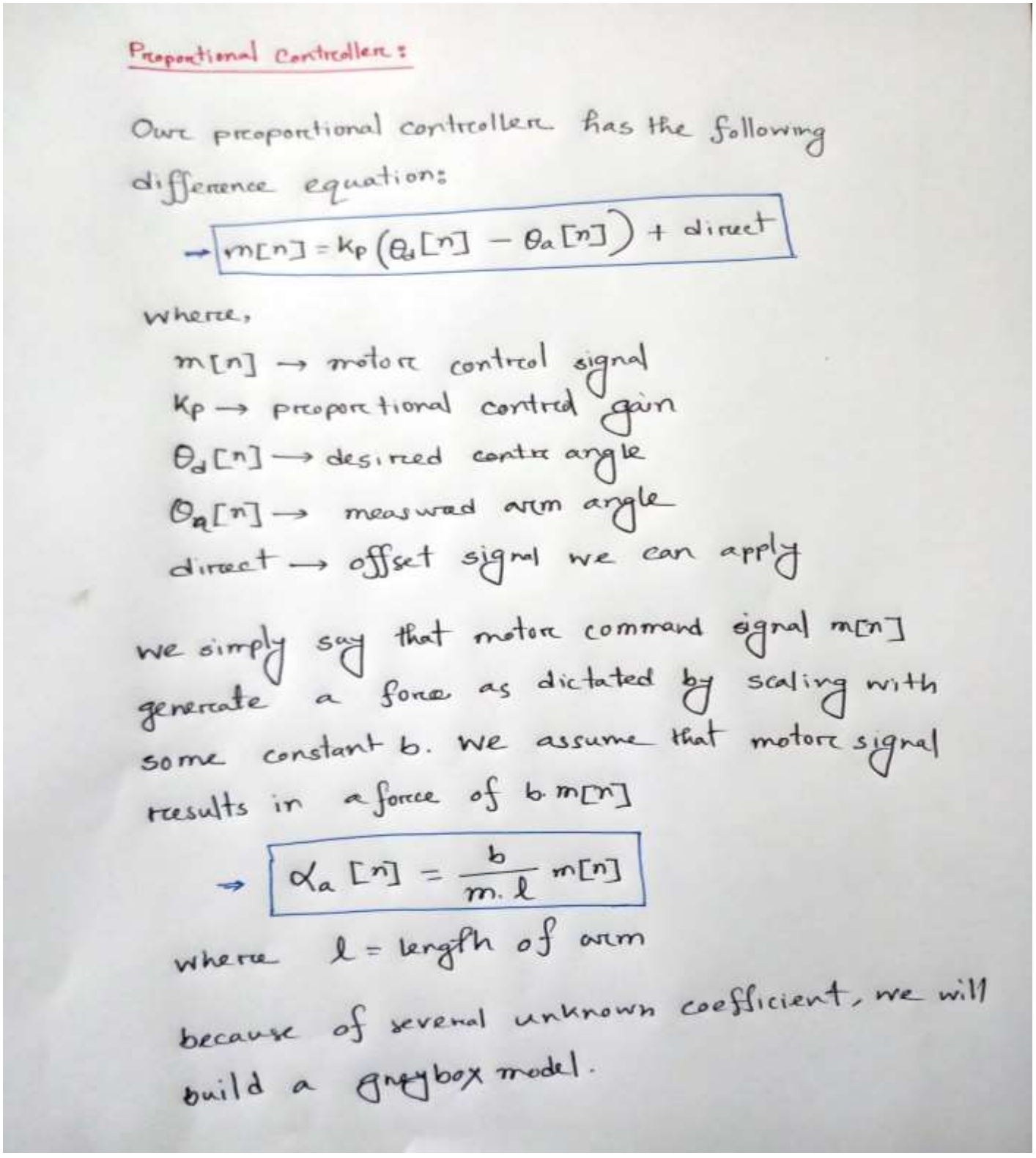
- **Server Code:** This is a Python script that talks with the Arduino over serial, does some calculations, and then talks with the GUI code in your browser. It is the "middle-person" of our software stack.

- **GUI Code:** Your GUI (Graphical User Interface) is basically just a web-page that is hosted on your machine. It is written in standard html/css/javascript. Don't worry, it is locally hosted and only you have access to it! It lives at localhost:3000 in your browser url field when the server script is running.

GUI Interface:

Controller Analysis:

Mathematical Derivation of Proportional Controller



Proportional Controller:

Our proportional controller has the following difference equation:

$$m[n] = k_p\left(\theta_d[n] - \theta_a[n]\right) + \text{direct}$$

Where,

$m[n]$ → motor control signal
K_p → proportional control gain
$\theta_d[n]$ → desired centre angle
$\theta_a[n]$ → measured arm angle
direct → offset signal we can apply

we simply say that motor command signal $m[n]$ generate a force as dictated by scaling with some constant b. we assume that motor signal results in a force of $b \cdot m[n]$

$$\alpha_a[n] = \frac{b}{m \cdot l} m[n]$$

where l = length of arm

because of several unknown coefficient, we will build a greybox model.

Instead of deriving each of the unknown coefficient from first principle, we will lump them togather and estimate one coefficient experimentally.

We have figured out that plant can be described with following difference equation relating angular acceleration $\ddot{\theta}[n]$ to angle $\theta[n]$

$$\theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2] = \alpha_a[n-2] \cdot T^2$$

If our angular acceleration is now dicated by expression because of proportional control:

$$\alpha_a[n] = \gamma m[n]$$

we find that

$$\theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2] = T^2\gamma K_p \theta_d[n-2] - T^2\gamma K_p \theta_a[n-2] + T^2\gamma . direct$$

After rearranging,

$$\theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2](1 + T^2\gamma K_p) = T^2\gamma K_p \theta_d[n-2] + T^2\gamma . direct$$

Analyzing this expression for its natural frequency we end up with a quadratic expression. In this analysis we disgreated the direct terem as it is constant overall time and input. It is therefore part of our operating point.

$$\lambda_{1,2} = \frac{2 \pm \sqrt{4 - 4(1 + T^2 \cdot \gamma \cdot K_P)}}{2}$$

Realizing that

→ T must be positive

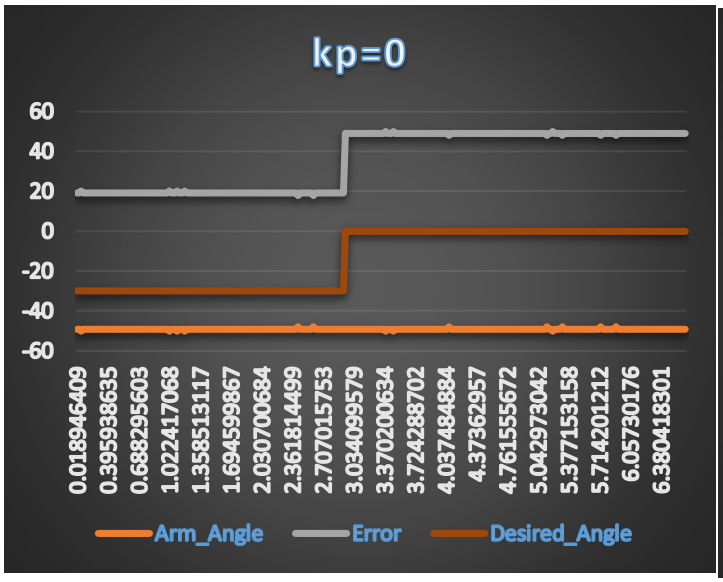
→ K_P is limited to being real, it should become apparent that for all K_P values, the system guarateed that to have natural frequencies with magnitude of at least 1

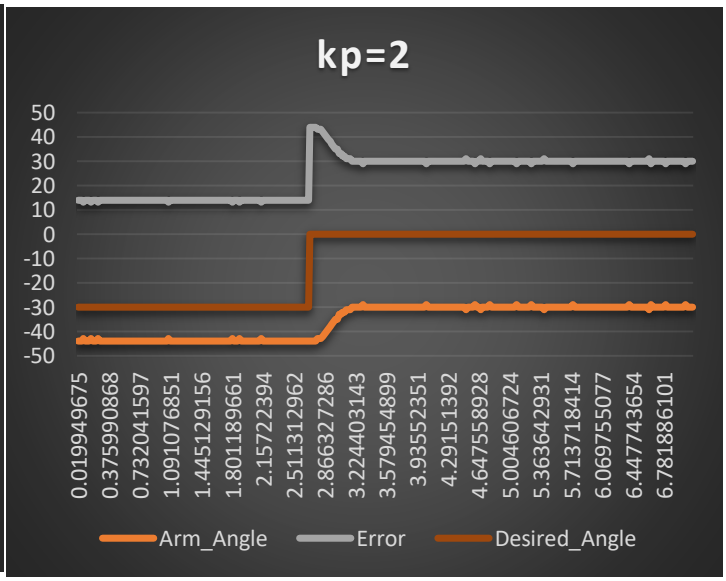
Arduino code for proportional controller

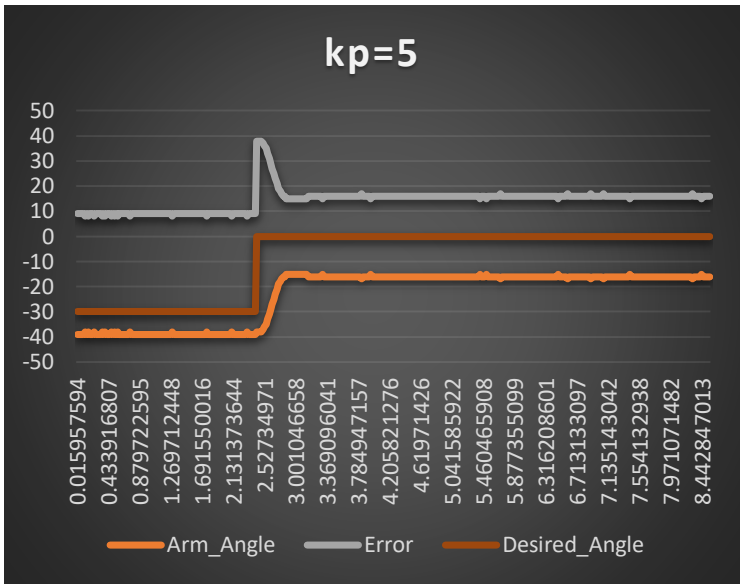
```
float Vangle = float (analogRead(angleSensorPin)
                    + analogRead(angleSensorPin) + analogRead(angleSensorPin) - 3*adcCenter)/12.0;

// Compute the error, which is the difference between desired and measured angle.
float error = desired - Vangle;

// Compute and limit the motor output command.
int motorCmd = int(Kp *error + direct);
motorCmd = min(max(motorCmd, 0), dacMax);
analogWrite(motorEnablePin,motorCmd);
```

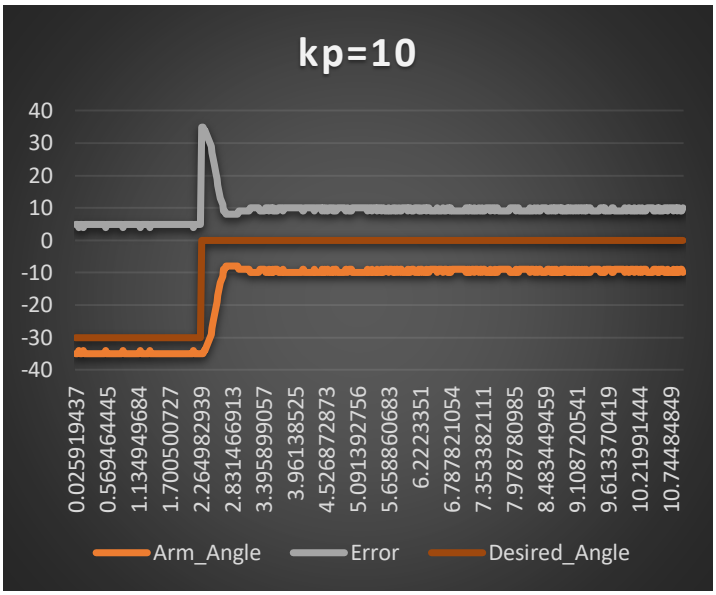
Plotting Graph from CSV file

By run the code with arduino and gui interface. We generate CSV file by python gui. Then plot the required data(arm_angle, desired_angle and error) in y axis and time in x axis with help of execl software. Then we got the graphs.









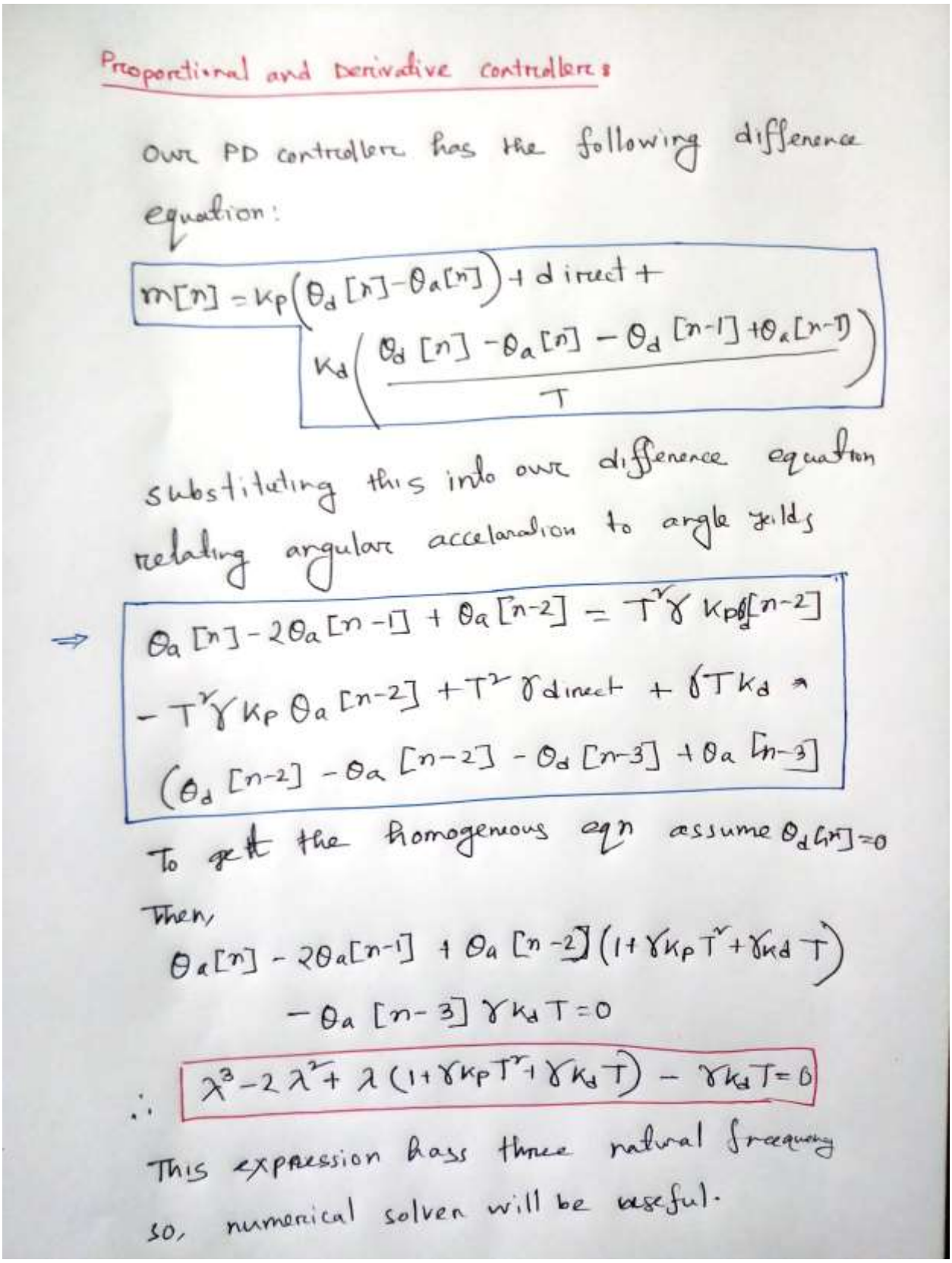
Observations:

As a general rule, increasing proportional gain decreases the steady state error. However, the
actual performance of P controller depends on the order of the plant.
from the following graph we observe that:

- Increasing gain decreases rise time (Advantage)
- Increasing gain increases percent overshoot and number of oscillations (Disadvantage)
- Increasing gain decreases steady state error (Advantage)
- Steady state is never zero if only-P type controller is used (Disadvantage)
- In order to have zero steady state error gain should be infinity (Physically impossible)

The discussion above shows that only-P control is not enough to control second order plants. P control is easy to implement.

Mathematical Derivation of Proportional - Derivative Controller



Proportional and Derivative controllers

Our PD controllers has the following difference equation:

$$m[n] = K_p\left(\theta_d[n] - \theta_a[n]\right) + \text{direct} + K_d\left(\frac{\theta_d[n] - \theta_a[n] - \theta_d[n-1] + \theta_a[n-1]}{T}\right)$$

substituting this into our difference equation relating angular accelaration to angle yields

$$\Rightarrow \theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2] = T^2\gamma K_p\theta_d[n-2] - T^2\gamma K_p\theta_a[n-2] + T^2\gamma\,\text{direct} + \gamma T K_d * \left(\theta_d[n-2] - \theta_a[n-2] - \theta_d[n-3] + \theta_a[n-3]\right)$$

To get the homogeneous eqn assume $\theta_d[n] = 0$

Then,

$$\theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2]\left(1 + \gamma K_p T^2 + \gamma K_d T\right) - \theta_a[n-3]\gamma K_d T = 0$$

$$\therefore \lambda^3 - 2\lambda^2 + \lambda\left(1 + \gamma K_p T^2 + \gamma K_d T\right) - \gamma K_d T = 0$$

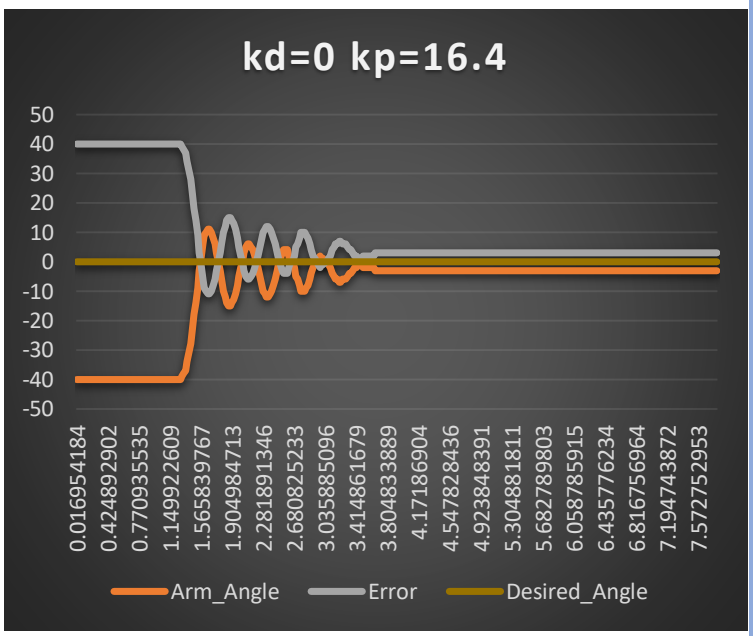
This expression has three natural frequency so, numerical solver will be useful.

Arduino code for PD controller:

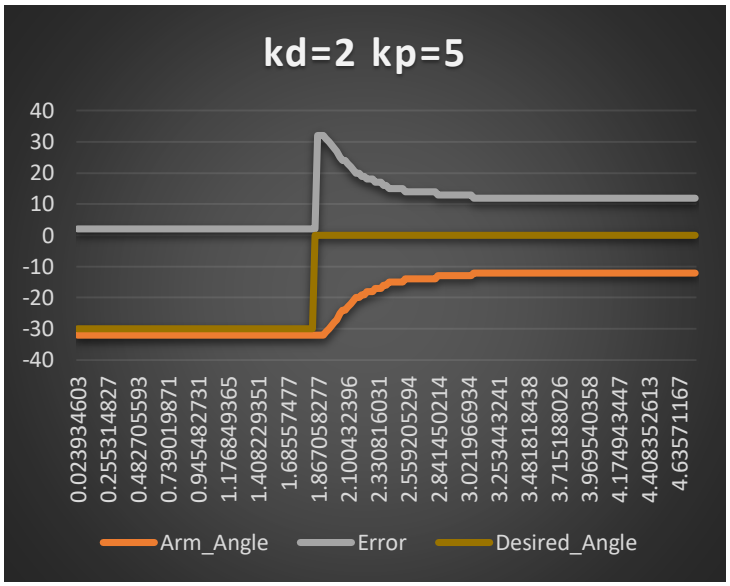
```
// Compute the error, which is the difference between desired and measured angle.
for(int i =pastSize-1; i>0; i--){
  pastError[i] = pastError[i-1];
}
float error = desired - Vangle;
pastError[0] = error;
float delta = (pastError[0] - pastError[pastSize-1])/(pastSize*(float(deltaT)*1e-6));
// Compute and limit the motor output command.
int motorCmd = int(Kp *error + Kd*delta + direct);
motorCmd = min(max(motorCmd, 0), dacMax);
analogWrite(motorEnablePin,motorCmd);
```

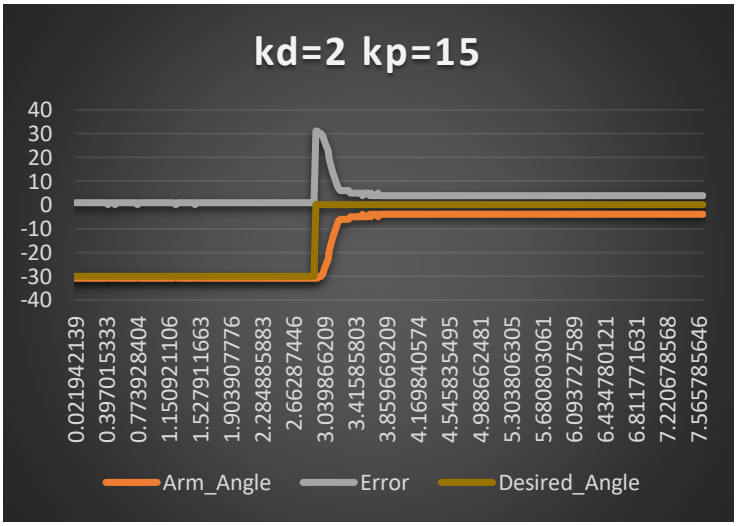
Plotting Graph from CSV file

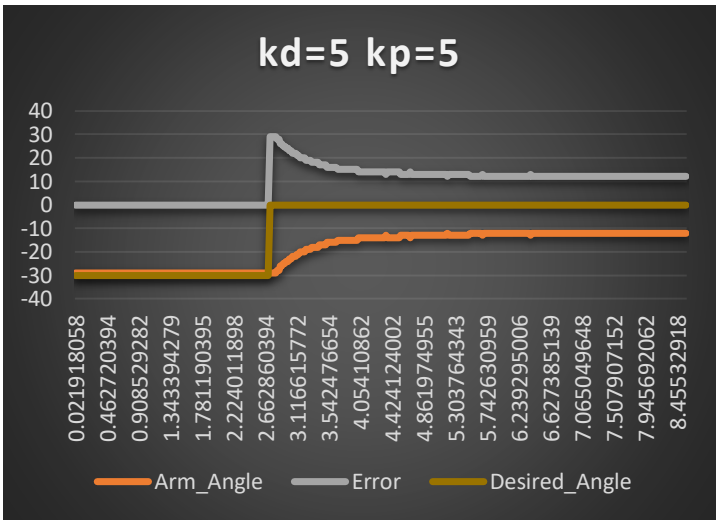
By run the code with arduino and gui interface. We generate CSV file by python gui. Then plot the required data(arm_angle, desired_angle and error) in y axis and time in x axis with help of execl software. Then we got the graphs.

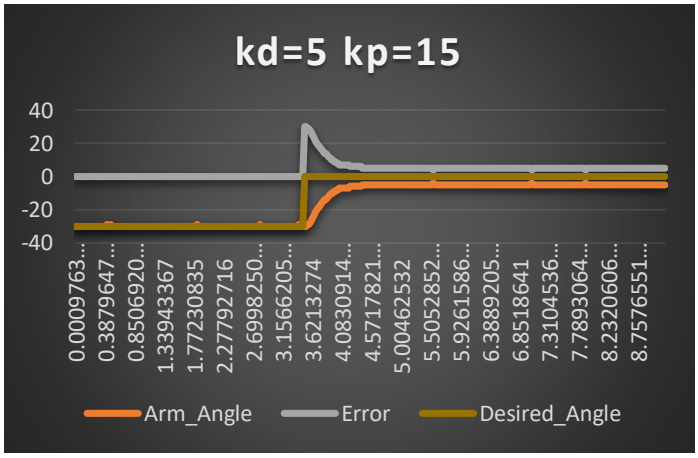



kd=0 kp=18.6
50
40
30
20
10
0
-10
-20
-30
-40
-50
0.012962103
0.283208132
0.584401131
0.827748775
1.108013391
1.373304129
1.699431896
1.921853065
2.192112207
2.466377974
2.777546406
3.012914896
3.28219533
3.556480169
3.866630554
4.101004601
4.375270367
4.645547628
4.979652882
5.195075035
5.467346668
Arm_Angle
Error
Desired_Angle

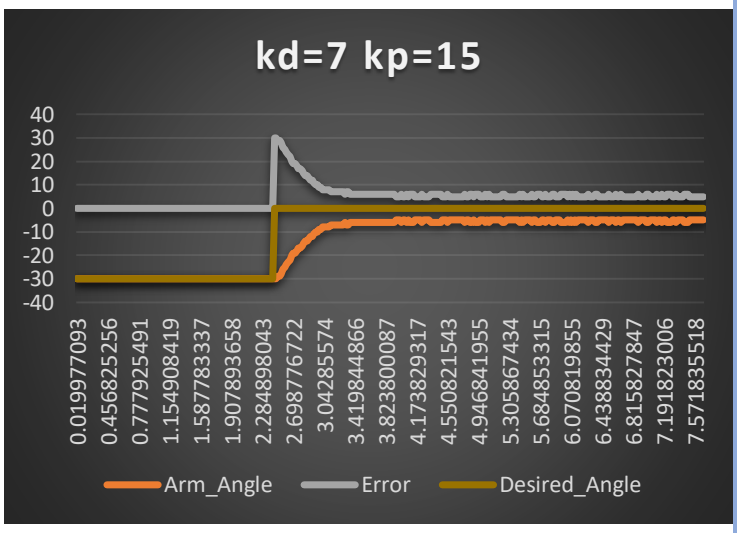

kd=2 kp=5
40
30
20
10
0
-10
-20
-30
-40
0.023934603
0.255314827
0.482705593
0.739019871
0.945482731
1.176849365
1.408229351
1.68557477
1.867058277
2.100432396
2.330816031
2.559205294
2.841450214
3.021966934
3.253443241
3.481818438
3.715188026
3.969540358
4.174943447
4.408352613
4.63571167
Arm_Angle
Error
Desired_Angle

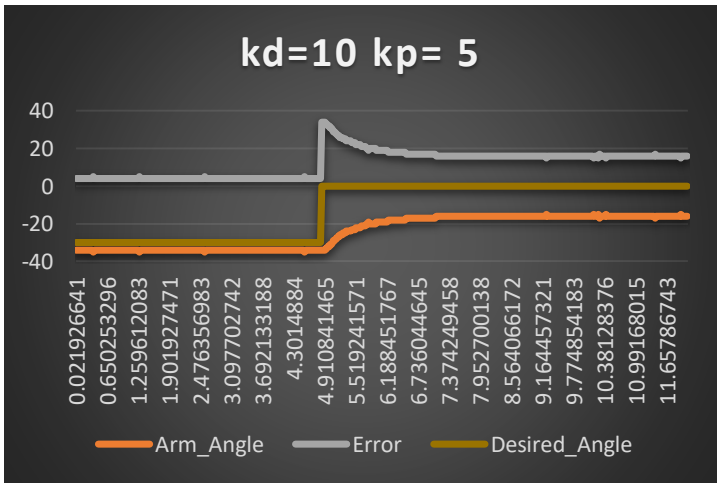

kd=2 kp=15
40
30
20
10
0
-10
-20
-30
-40
0.021942139
0.397015333
0.773928404
1.150921106
1.527911663
1.903907776
2.284885883
2.66287446
3.039866209
3.41585803
3.859669209
4.169840574
4.545835495
4.988662481
5.303806305
5.680803061
6.093727589
6.434780121
6.811771631
7.220678568
7.565785646
Arm_Angle
Error
Desired_Angle

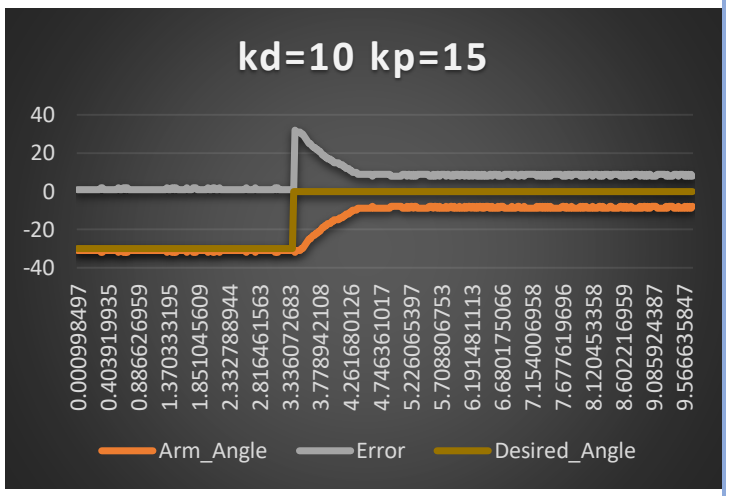

kd=5 kp=5
40
30
20
10
0
-10
-20
-30
-40
0.021918058
0.462720394
0.908529282
1.343394279
1.781190395
2.224011898
2.662860394
3.116615772
3.542476654
4.05410862
4.424124002
4.861974955
5.303764343
5.742630959
6.239295006
6.627385139
7.065049648
7.507907152
7.945692062
8.45532918
Arm_Angle
Error
Desired_Angle

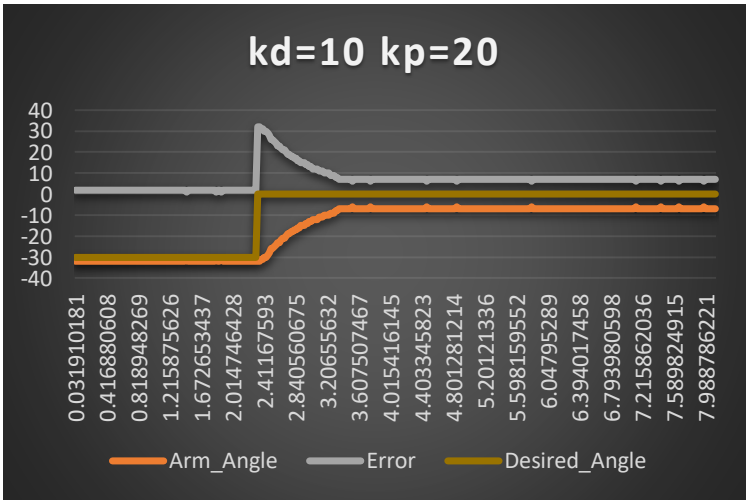

kd=5 kp=15
40
20
0
-20
-40
0.0009763...
0.3879647...
0.8506920...
1.33943367
1.77230835
2.27792716
2.6998250...
3.1566205...
3.6213274
4.0830914...
4.5717821...
5.00462532
5.5052852...
5.9261586...
6.3889205...
6.8518641
7.3104536...
7.7893064...
8.2320606...
8.7576551...
Arm_Angle
Error
Desired_Angle


kd=7 kp=5
40
30
20
10
0
-10
-20
-30
-40
0.015957117
0.455780745
0.939500809
1.334427834
1.777441978
2.216268778
2.660042048
3.137762785
3.538722515
3.980509043
4.419369221
4.862182617
5.332982063
5.741794586
6.180620193
6.622436762
7.061295986
7.516046524
7.943912506
8.384773016
Arm_Angle
Error
Desired_Angle


kd=7 kp=15
40
30
20
10
0
-10
-20
-30
-40
0.019977093
0.456825256
0.777925491
1.154908419
1.587783337
1.907893658
2.284898043
2.698776722
3.04285574
3.419844866
3.823800087
4.173829317
4.550821543
4.946841955
5.305867434
5.684853315
6.070819855
6.438834429
6.815827847
7.191823006
7.571835518
Arm_Angle
Error
Desired_Angle


kd=10 kp= 5
40
20
0
-20
-40
0.021926641
0.650253296
1.259612083
1.901927471
2.476356983
3.097702742
3.692133188
4.3014884
4.910841465
5.519241571
6.188451767
6.736044645
7.374249458
7.952700138
8.564066172
9.164457321
9.774854183
10.38128376
10.99168015
11.65786743
Arm_Angle
Error
Desired_Angle


kd=10 kp=15
40
20
0
-20
-40
0.000998497
0.403919935
0.886626959
1.370333195
1.851045609
2.332788944
2.816461563
3.336072683
3.778942108
4.261680126
4.746361017
5.226065397
5.708806753
6.191481113
6.680175066
7.154006958
7.677619696
8.120453358
8.602216959
9.085924387
9.566635847
Arm_Angle
Error
Desired_Angle

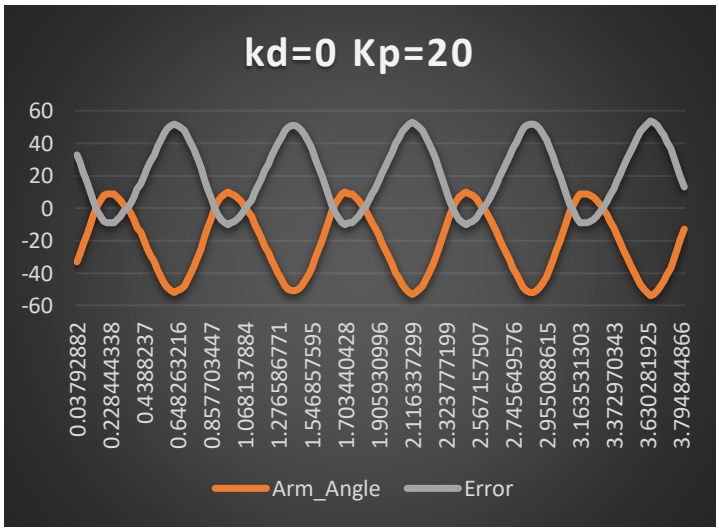

kd=10 kp=20
40
30
20
10
0
-10
-20
-30
-40
0.031910181
0.416880608
0.818948269
1.215875626
1.672653437
2.014746428
2.41167593
2.840560675
3.20655632
3.607507467
4.015416145
4.403345823
4.801281214
5.20121336
5.598159552
6.04795289
6.394017458
6.793980598
7.215862036
7.589824915
7.988786221
Arm_Angle
Error
Desired_Angle

Observations:

From the above graph, we observe the response the PD controller. By increasing parameter of the proportional derivative controller we find that

	Rise Time	Overshoot	Settling time	Steady-state error
Kp	Decrease	Increase	Very small effect	Decrease
Kd	Decrease	Increase	Increase	Eliminate

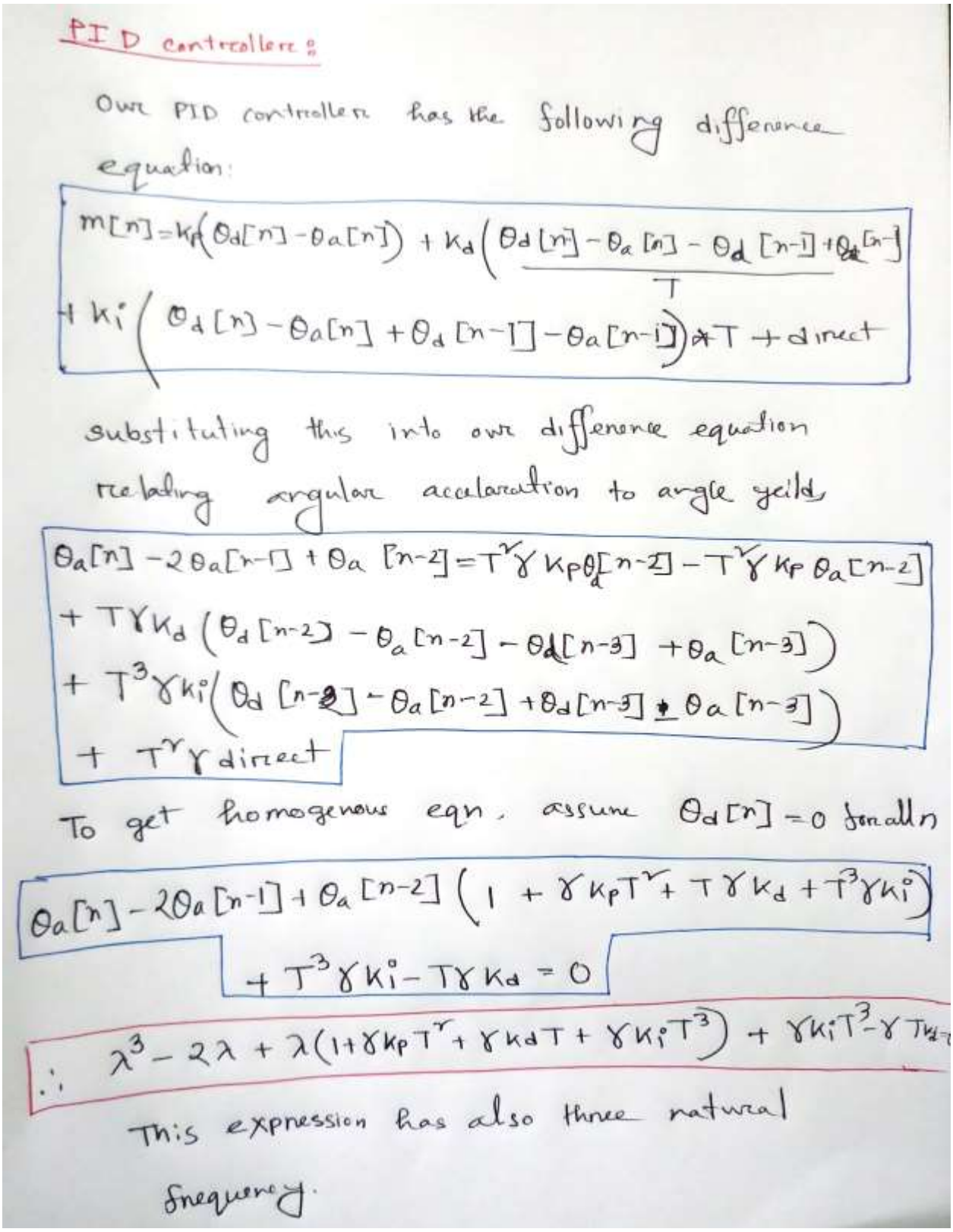
Special case:



In PD controller when **kd = 0 and kp = 20**, we observe that the arm is oscillating up and down showing in the graph. This is the **marginally stable** point for the PD controller

Perfect value of kp = 16.4, Kd = 7.45 and Motor speed= 65

Mathematical Derivation of PID Controller



PID controllers:

Our PID controller has the following difference equation:

$$m[n] = K_p(\theta_d[n] - \theta_a[n]) + K_d\left(\frac{\theta_d[n] - \theta_a[n] - \theta_d[n-1] + \theta_a[n-1]}{T}\right) + K_i\left(\theta_d[n] - \theta_a[n] + \theta_d[n-1] - \theta_a[n-1]\right)*T + \text{direct}$$

substituting this into our difference equation relating angular accelaration to angle yeild

$$\theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2] = T^2\gamma K_p\theta_d[n-2] - T^2\gamma K_p\theta_a[n-2] + T\gamma K_d\left(\theta_d[n-2] - \theta_a[n-2] - \theta_d[n-3] + \theta_a[n-3]\right) + T^3\gamma K_i\left(\theta_d[n-2] - \theta_a[n-2] + \theta_d[n-3] + \theta_a[n-3]\right) + T^2\gamma\,\text{direct}$$

To get homogenous eqn, assume $\theta_d[n] = 0$ for all n

$$\theta_a[n] - 2\theta_a[n-1] + \theta_a[n-2]\left(1 + \gamma K_p T^2 + T\gamma K_d + T^3\gamma K_i\right) + T^3\gamma K_i - T\gamma K_d = 0$$

$$\therefore \lambda^3 - 2\lambda + \lambda(1 + \gamma K_p T^2 + \gamma K_d T + \gamma K_i T^3) + \gamma K_i T^3 - \gamma T K_d$$

This expression has also three natural frequency.

Arduino code for PID controller:

```
float Vangle = float (analogRead(angleSensorPin)
                      + analogRead(angleSensorPin) + analogRead(angleSensorPin) - 3*adcCenter)/12.0;

// Compute the error, which is the difference between desired and measured angle.
for(int i =pastSize-1; i>0; i--){
  pastError[i] = pastError[i-1];
}
float error = desired - Vangle;
pastError[0] = error;
float delta = (pastError[0] - pastError[pastSize-1])/(pastSize*(float(deltaT)*1e-6));
sum = min(max(sum + error*float(deltaT)*1e-6,-1*maxSum),maxSum);
// Compute and limit the motor output command.
int motorCmd = int(Kp *error + Kd*delta + Ki*sum + direct);
motorCmd = min(max(motorCmd, 0), dacMax);
analogWrite(motorEnablePin,motorCmd);
```

Effects of PID parameters when increasing them:

	Rise Time	Overshoot	Settling time	Steady-state error	Stability
Kp	Decrease	Increase	Very small effect	Decrease	Degrade
Kd	Decrease	Increase	Increase	Eliminate	Degrade
Ki	Very small effect	Decrease	Decrease	No effect	Improve if K_d small

PID Tuning techniques

There are many tuning techniques that can be employed in order to tune a PID controller. Each technique uses a set of parameters that can be extracted from the system either by running an experiment (closed-loop or open-loop) or by using the transfer function of the system. Those parameters are then used in rules (different for each technique) that produce the proportional (Kp), integral (Ki) and derivative (Kd) gains.

Closed-loop Ziegler Nichols

The most famous PID tuning technique, which is the basis for many new tuning methods, is the Ziegler Nichols method. The Ziegler Nichols method was developed by J.G Ziegler and N.B Nichols in 1942 but still remains an important PID
tuning method. The Ziegler Nichols closed-loop method is applied to the transfer function of the complete system and is essentially a trial and error technique. The transfer function of the complete system is the combination of equation 2.5 and the transfer function of the process, which can be obtained by the two empirical methods. To tune a controller using the Ziegler Nichols method this procedure is followed:

1. Kp in the equation 2.5 is set to a value close to zero and Ki and Kd are turned off completely (set to zero).
2. The Kp value is increased until the output of the control loop is steadily oscillating. Ku is equal to the Kp used to produce steady oscillation.
3. The period of the steady oscillation Pu is then measured.
4. Using the table 2.3 the values for τI and τD can be calculated (Kp, Ki and Kd
   can be produced from the τ values, Kp = Kc, Ki = Kc / τi, Kd = Kc * τd)

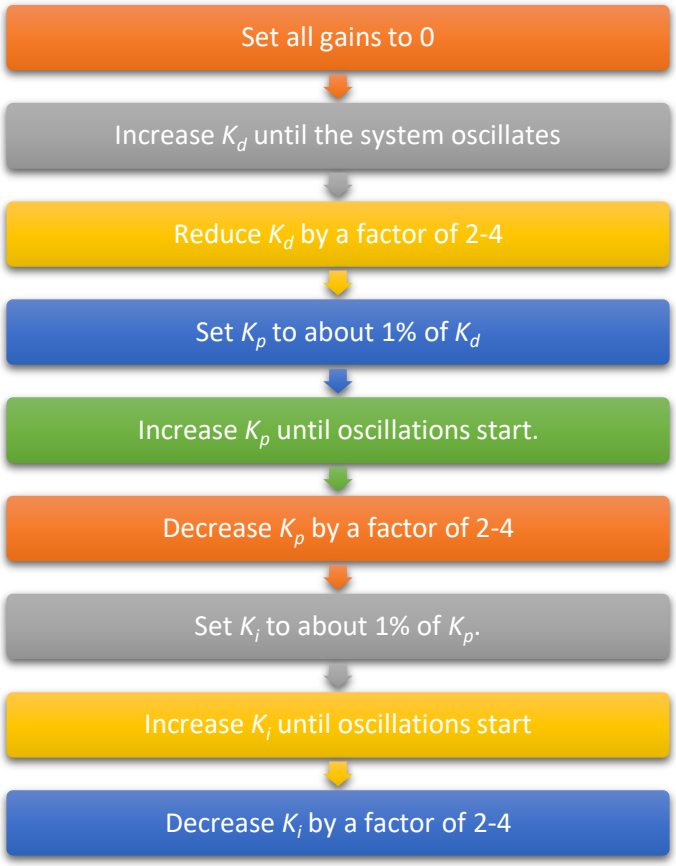
The closed-loop Ziegler Nichols method is best suited to high order systems (i.e. systems that their transfer function is cubic or of a higher degree) or systems with time delay.

	K_c	τ_I	τ_D
P control	$K_u/2$		
PI control	$K_u/2.2$	$P_u/1.2$	
PID control	$K_u/1.7$	$P_u/2$	$P_u/8$

Table 2.3, closed –loop Ziegler Nichols PID tuning chart, from ref [18]

Experimental method

A similar experimental method we used to tune our PID controller for levitated copter-arm. The procedure of this method is describing here



Optimum Value of Kp Kd and Ki

By this method we found that the optimum value of kp is 12.8, kd is 5 and ki is 4.2 and sum is 200

Conclusions:

P-I-D control and its variations are commonly used in the industry. They have so many applications. Control engineers usually prefer P-I controllers to control first order plants. On the other hand, P-I-D control is vastly used to control two or higher order plants. In almost all cases fast transient response and zero steady state error is desired for a closed loop system. Usually, these two specifications conflict with each other which makes the design harder. The reason why P-I-D is preferred is that it provides both of these features at the same time. In this recitation, it was aimed to explain how one can successfully use P-I-D controllers in their prospective projects. We tried to focus on almost all aspects of P-I-D control. However, it is almost impossible to fit the explanation of P-I-D controllers within one hour. We suggest for the future Discrete Time Control System students to split the P-I-D controller subject into pieces and explain it more than one recitation hour. Being prospective control engineers, we feel lucky to give a presentation on the P-I-D subject. Finally, we encourage prospective control engineers to use P-I-D controllers wherever necessary, especially, when a great controller is required.