**Project Report on**

# CLOSED LOOP PERFORMANCE ANALYSIS AND SPEED CONTROL OF A DC MOTOR

SUBMITTED IN PARTIAL FULFILLMENT FOR THE
AWARD OF THE DEGREEOF

**BACHELOR OF ENGINEERING**
**IN**
**ELECTRONICS ENGINEERING**
**(A.Y. 2014-2015)**

BY

KALPESH CHAUDHARI (11)
SHARANHEERCHOUDHARI(12)
ADITYA PALEKAR (41)
MANSI THAKOR(72)

**UNDER THE GUIDANCE OF**
**HEMANT KASTURIWALE**
ASSOCIATE PROFESSOR-ELECTRONICS DEPARTMENT


**UNIVERSITY OF MUMBAI**
**DEPARTMENT OF ELECTRONICS ENGINEERING**

THAKUR TRUSTS
Laxmi Singh Charitable Trust's (Regd.)
THAKUR COLLEGE OF ENGINEERING & TECHNOLOGY
(Approved by AICTE, Govt. of Maharashtra & Affiliated to University of Mumbai)
(Accredited by National Board of Accreditation, New Delhi)*
A - Block, Thakur Educational Campus,
Shyamnarayan Thakur Marg, Thakur Village,
Kandivali (East), Mumbai - 400 101.
Tel.: 6730 8000 / 8106 / 8107
Fax : 2846 1890
Email : tcet@thakureducation.org
Website : www.tcetmumbai.in • www.thakureducation.org
* Accredited Programmes : • Computer Engineering • Electronics & Telecommunication Engineering • Information Technology (w.e.f.: 16-09-2011 for 3 years)
tcet
ISO 9001 : 2008 Certified

# CERTIFICATE

This is to certify that Aditya Palekar, Sharanheer Choudhari, Mansi Thakor and Kalpesh Chaudhariare the bonafide students of Thakur College of Engineering and Technology, Mumbai. They have satisfactorily completed the requirements of project-II as prescribed by the University of Mumbai while working on the project titled "Closed Loop Performance Analysis And Speed Control of a DC Motor". The work has not been presented elsewhere for the award of any other degree or diploma prior to this.

| | | |
|---|---|---|
| **(Signature)** | **(Signature)** | **(Signature)** |
| **Prof.Hemant Kasturiwale**<br>**(Associate Professor)**<br>**(ETRX)** | **Prof. PoorvaWaingankar**<br>**(HOD ETRX)** | **Dr. B. K. Mishra**<br>**(Principal)** |

| | |
|---|---|
| **Internal Examiner**<br>**(Name and Signature with Date)** | **External Examiner**<br>**(Name and Signature with Date)** |

Thakur College of Engineering and Technology

Kandivali(E), Mumbai-400101.

PLACE: Mumbai

DATE:

# ACKNOWLEDGEMENT

Sincere and heartfelt thanks to Hemant Kasturiwale Sir for his affectionate guidance and constant support. His patient, yet thorough approach is the key reason that we were able to widen the scope of this project.

We would even like to extend our gratitude to fellow librarians and lab assistants for aiding us in collecting the necessary resources that would eventually lead to the fruition of the proposed project.

We extend our sincere thanks to our Head of Department, Mrs. Poorva Waingankar, our principal Dr. B.K. Mishra and to the management of THAKUR COLLEGE OF ENGINEERING & TECHNOLOGY for giving us required facilities that helped this project reach completion.

# DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature and name of all the students in the group

**(Aditya Palekar)**

**(Mansi Thakor)**

**(Sharanheer Choudhari)**

**(Kalpesh Chaudhari)**

# Abstract

The objective of our project is to do performance analysis and control the speed of a motor using PID controller. The PID controller is implemented using pic microcontroller and used in the speed control of motor.

Performance analysis of dc motor is done based on different types of controllers. Different controllers used are P, PD, PID controllers**.** A C code is written for the pic microcontroller to implement these controllers. A matlab GUI is created which communicates with pic microcontroller and also used to control pic microcontroller.

Performance analysis of dc motor is done using waveforms plotted in matlab GUI.A waveform of velocity v/s time is plotted in matlab GUI. The speed of the motor will be controlled based on the count obtained from the optical encoder and the speed calculated would be compared with the reference value and obtained error is projected over PID controller and the process continues till we get minimum error.

The effectiveness of controlling the speed of motor by using the software is, we are open to change the Kp, Kd and Ki constants. Based upon the error occurred and by changing the values suitably the required output from the system can be obtained.

.

# CONTENTS

# List of figures

# 1.INTRODUCTION

## 1.1 Importance and Background

DC Motor plays a crucial role in research and laboratory experiments because of their simplicity and low cost. The speed of the motor can be controlled by three methods namely terminal voltage control, armature rheostat control method and flux control method. Here in this project terminal voltage control method is employed.

A control system is an interconnection of components forming a system configuration that will provide a desired system response. A controlled DC-motor was developed allowing Arduino a hardware acts as the interface between the computer and the outside world. It primarily functions as a device that digitizes incoming analog signals so that the computer can interpret them. The user interface was developed in a Arduino environment.

## 1.2 Motivation and Scope

In this project, the aim is to explore the capability of the PID CONTROLLER - an application of DC motor speed control.

The objective of this project to setup a DC motor controlled with PID CONTROLLER. Modeling of the DC motor system and simulate with PROTEUS.

The project will include:

Developing the PID controller using PIC16F877A and interfacing the PIC16F877A hardware by dumping the program.

### Closed loop control:

A closed-loop control system is one in which an input forcing function is determined in part by the system response. The measured response of a physical system is compared with a desired response. The difference between these two responses initiates actions that will result in the actual response of the system to approach the desired response. This in turn drives the difference signal towards zero. Typically the difference signal is processed by another physical system, which is called a compensator, a controller, or a filter for real-time control system applications.

# Chapter 2
# Proposed work and Literature review

## 2.1 Problem Definition:

The performance of dc motor is to be looked in terms of various electrical and electronics parameters. Generally, it is very difficult to implement and test a digital controller design and also the hardware implementation is expensive. Our project emphasis is on controller which is more stable using simulation . But for different desired speed of the motor we have to change the constants and re-burn the program in the controller and make the task quiet tedious. Our project will try to find out most suitable algorithm and controller for hardware implementation.

## 2.2 Problem Solution:

The project provides a low cost solution that will allow a student to implement and test a digital controller design. In this manner by changing the Kp, Ki, Kd constants the set point can be changed and the desired speed of Motor can be obtained without burden of writing all of the microcontroller and MATLAB code from scratch.

## 2.3 Literature Review

The purpose of this is to provide a low cost solution that will allow a student to implement and test a digital controller design. In this manner the student can obtain hands-on experience with implementing a digital controller without burden of writing all of the microcontroller and MATLAB code from scratch.

For this project, the PID control method will be used. PID, meaning Proportional-Integral-Derivative, is one of the most used feedback control method.
A PID controller consists of a Proportional element, an Integral element and a Derivative element, all three connected in parallel. All of them take the error as input. Kp, Ki, Kd are the gains of P, I and D elements respectively, as seen in Figure 2.3. Where *u(t)* is the control signal sent to the system, *y(t)* is the measured output and *r(t)* is the desired output, and tracking error $e(t) = r(t) - y(t)$, thus a PID controller can be expressed as in Figure 2.3

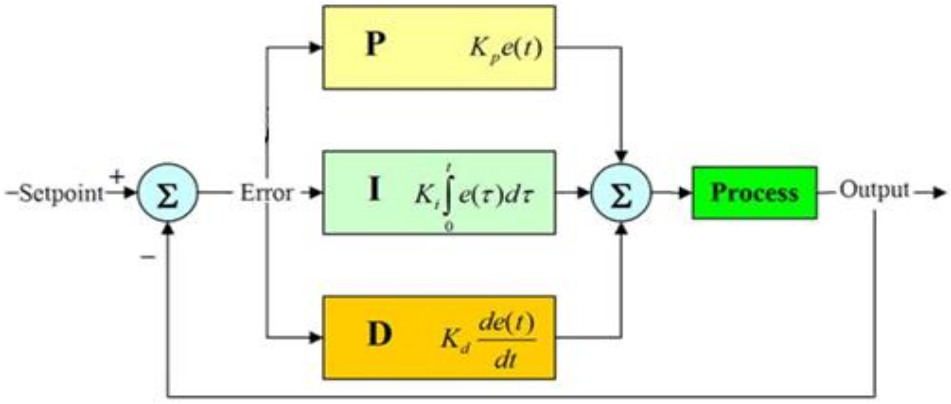



Fig 2.1 Feedback Circuit

# Chapter 3
# Analysis and Planning

## 3.1 Progress

After finalizing the project, we started searching for its applications. We searched for IEEE and various other research papers to know about the concepts related to our project and also the work that has been done in this field. We searched for various techniques and algorithms that would help us to implement our project in a simple and effective way. We came to know that PID controller gives better efficiency in controlling the speed of the d.c .motor. Hence we decided to control the speed of d.c. motor using a P.I.D controller. We made the general block diagram of how our project is going to work and started searching on the components required to build it. We prepared the synopsis and a presentation on the project. We referred IEEE and various other research papers for the same. We prepared an abstract and literature survey showing all the IEEE and various other research papers.

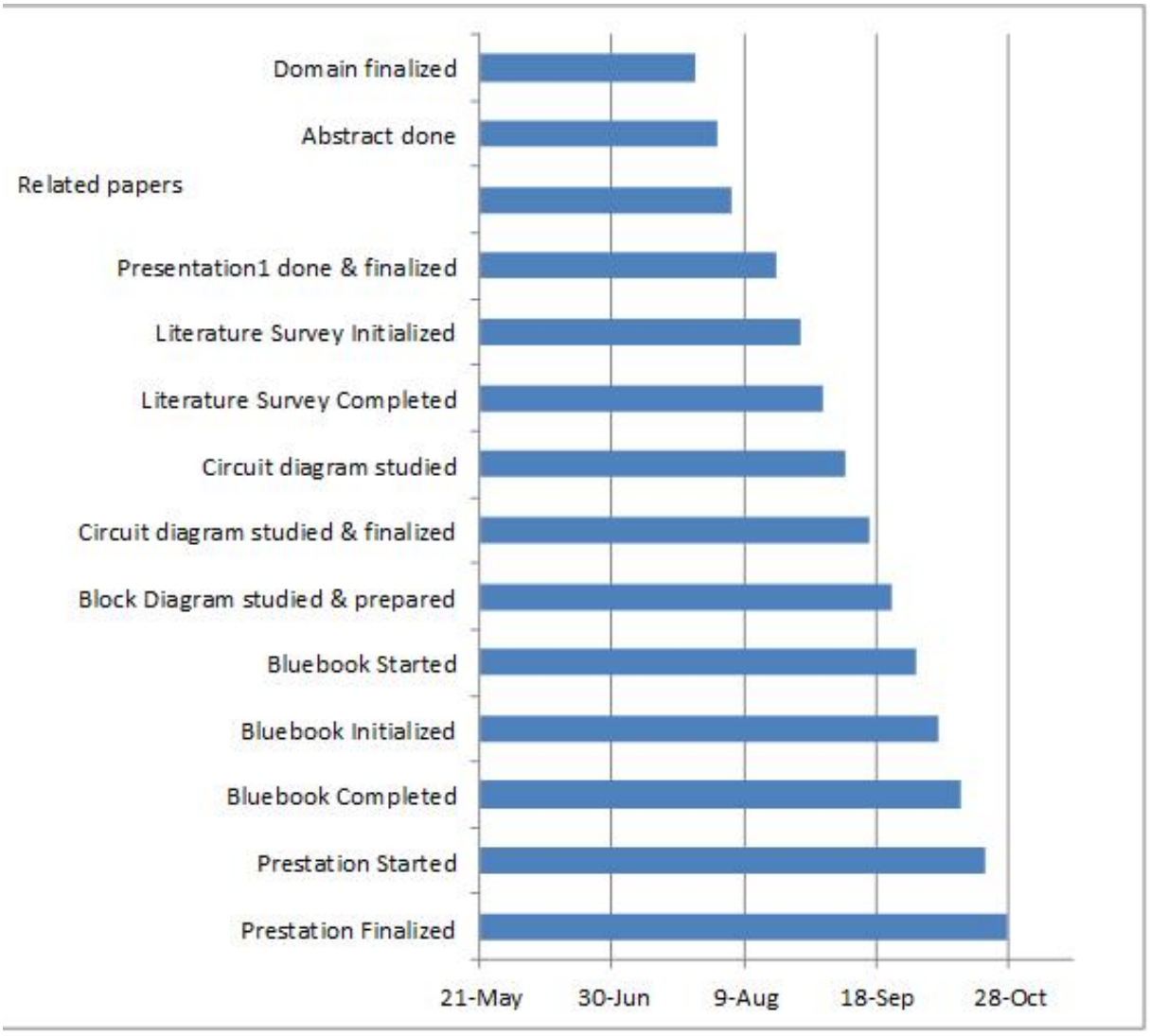
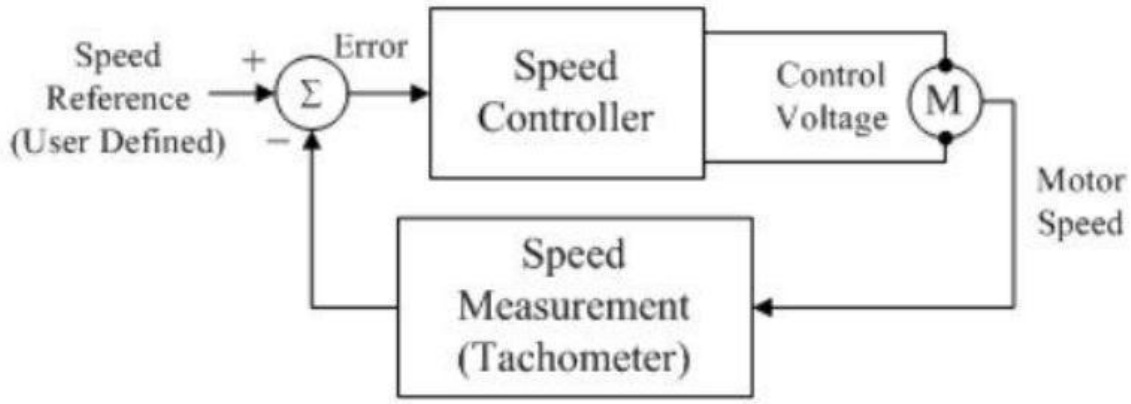



Fig 3.1 Graph of IEEE Papers

# Chapter 4

# Design, Implementation &Installation

## 4.1 Block Diagram:



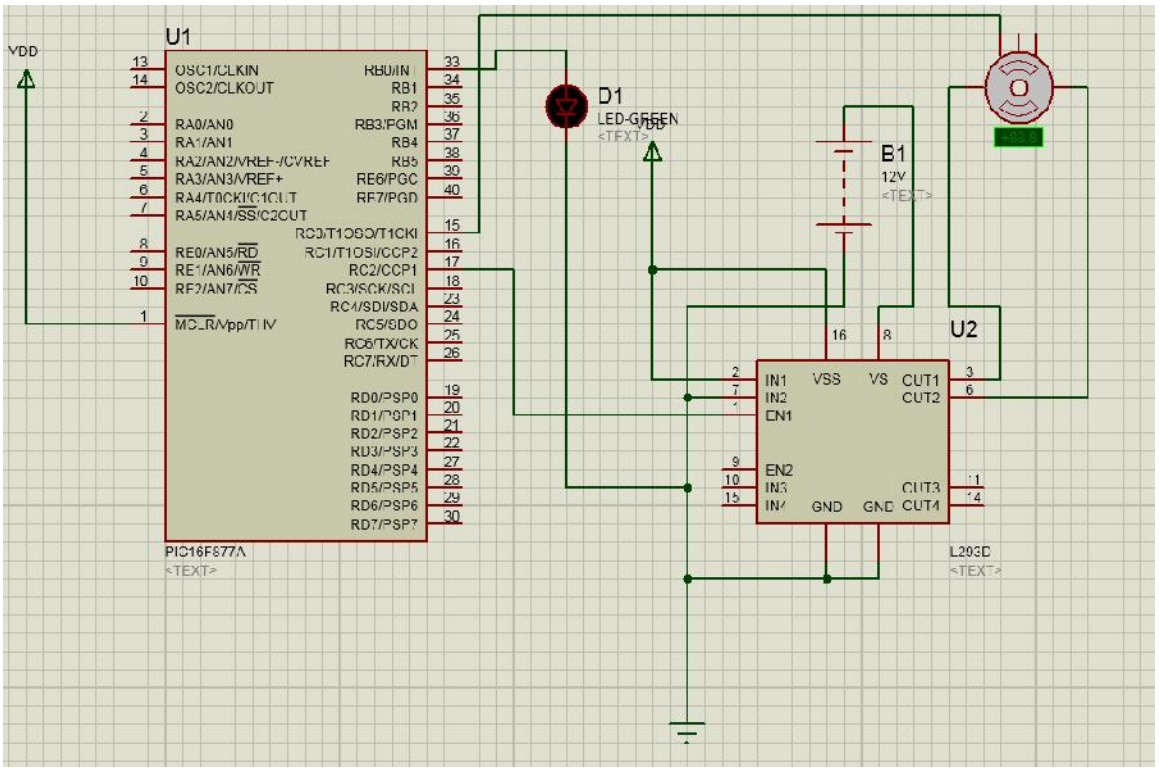




Fig.4.1 Block Diagram

## 4.2 Components

**D.C. Motor:**

An Electric motor is a machine which converts electric energy into mechanical energy. Its action is based on the principle of that when a current carrying conductor is placed in a magnetic field experiences a mechanical force whose direction is given by Fleming's Left-Hand Rule.

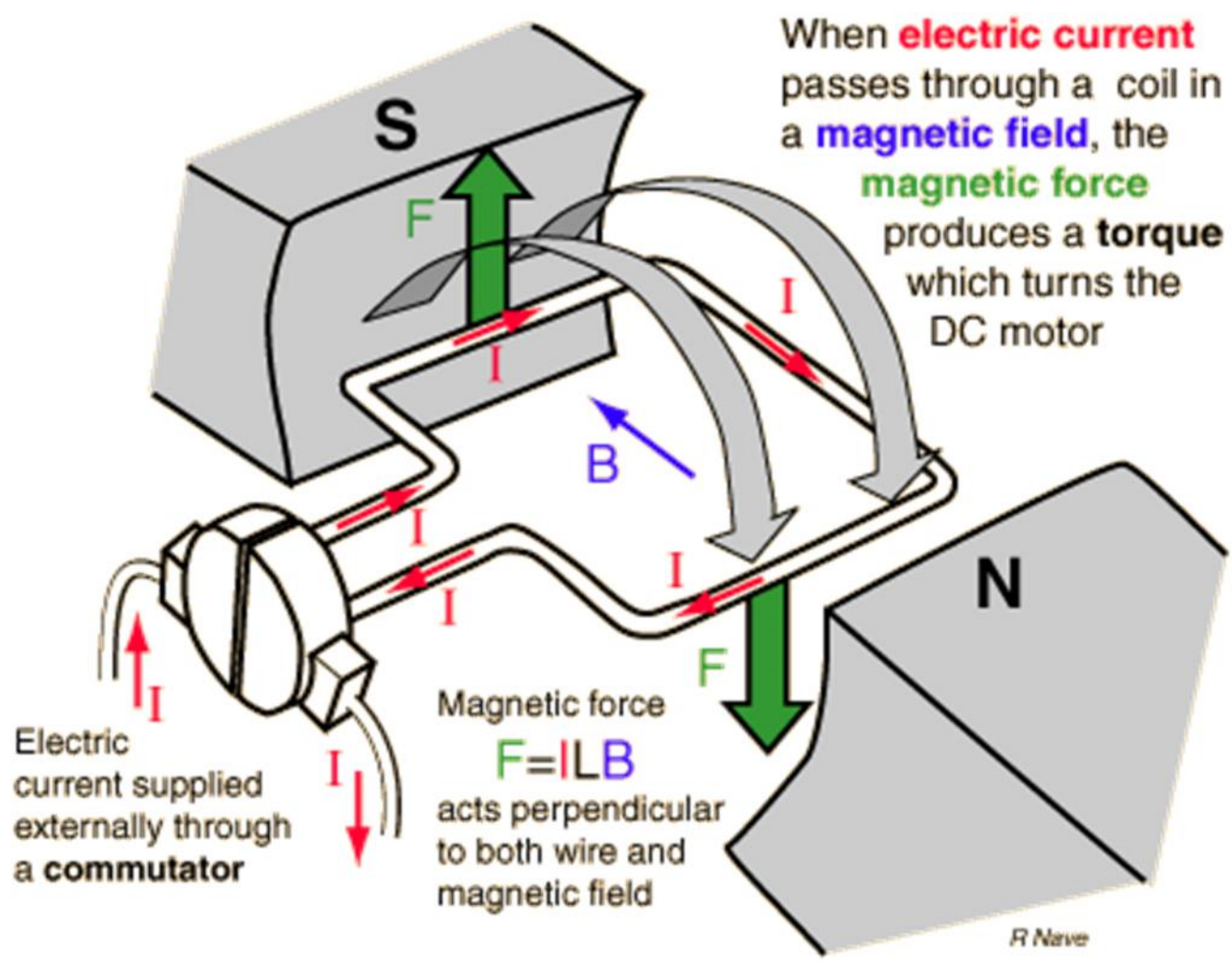


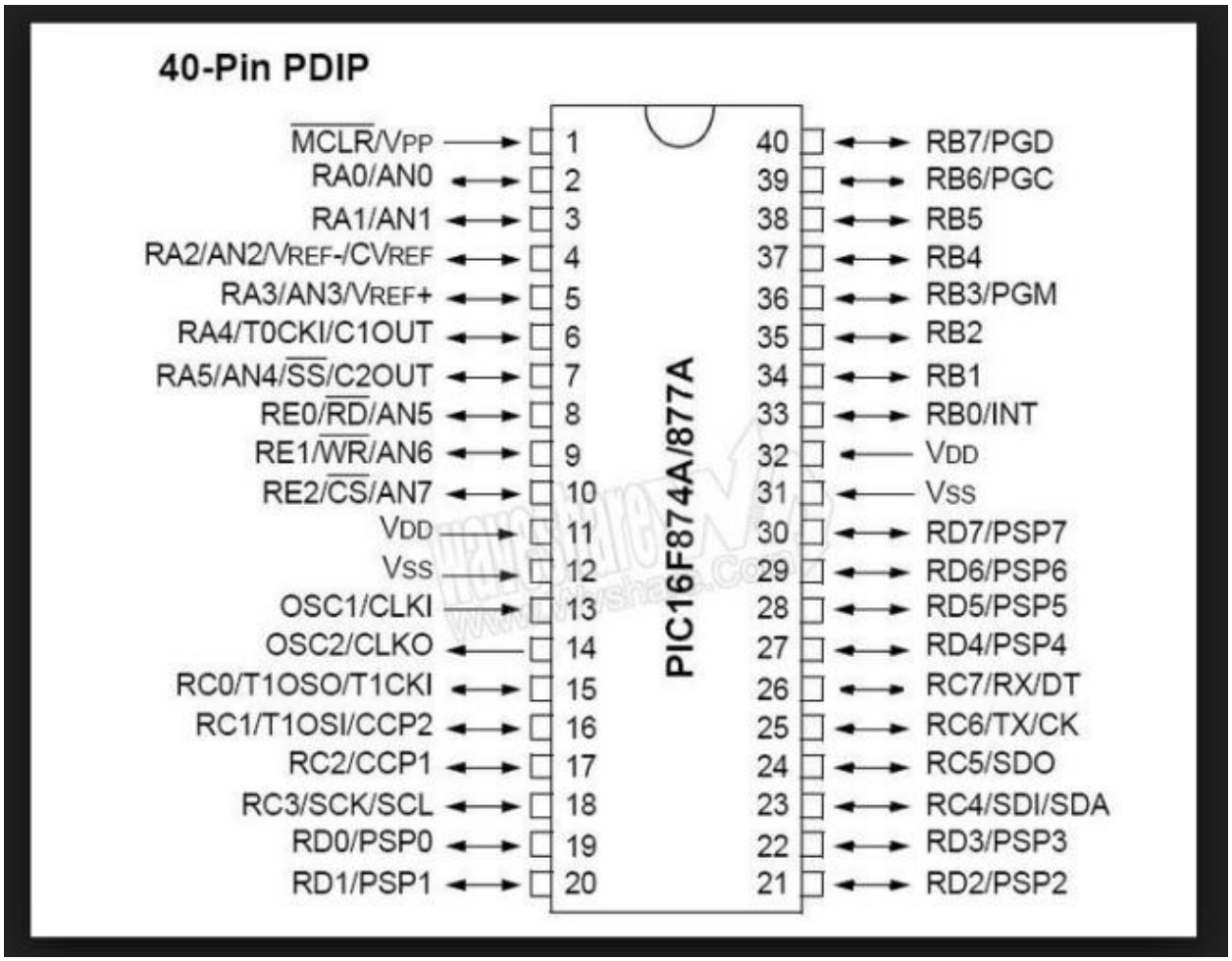

Fig. 4.2 DC Motor

### Factors Controlling Motor Speed:

The speed of the motor is given by the relation

$$N = \frac{V - IaRa}{Z\phi} \cdot \left(\frac{A}{P}\right) = K\frac{V - IaRa}{Z\phi} r.p.s$$

PIC16F877A

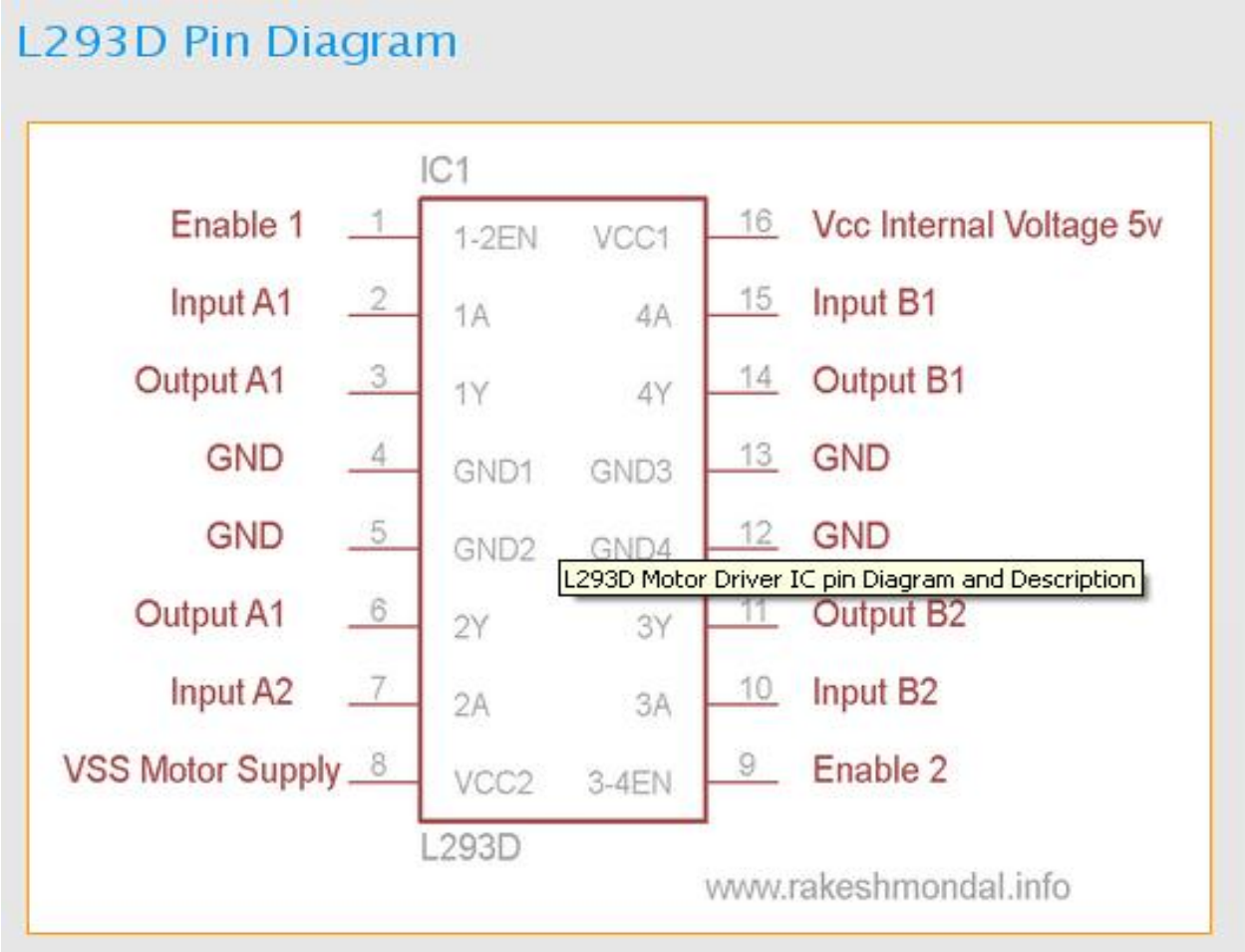



MICROCONTROLLER PINS USED

1) Pin 1 : mclkbar/ vpp – clear circuit
2) Pin 11,32 : vdd (+5v)
3) Pin 12,31 : gnd (0v)
4) Pin 13,14 : 20Mhz crystal oscillator
5) Pin 15 : feedback
6) Pin 17 : output(pulses)
7) Pin 25 : Transmit serial data
8) Pin 26 : Receive serial data.

`

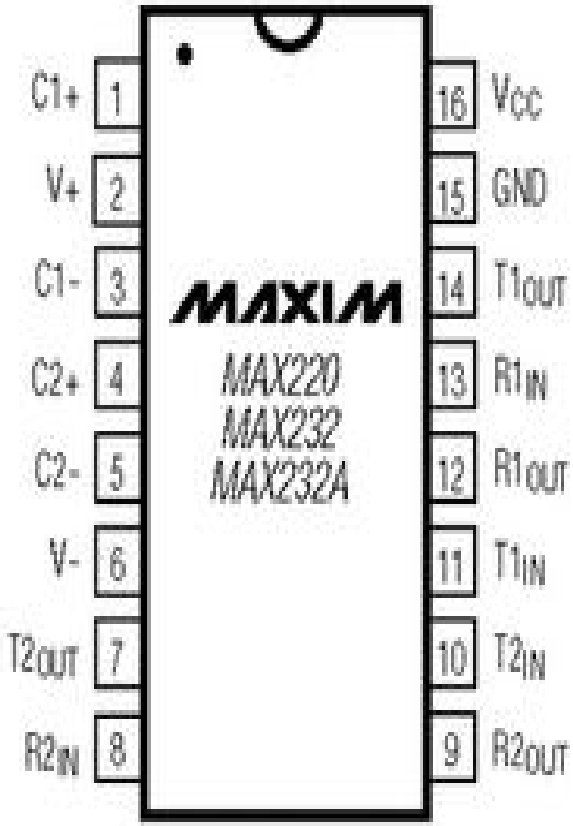
L293D



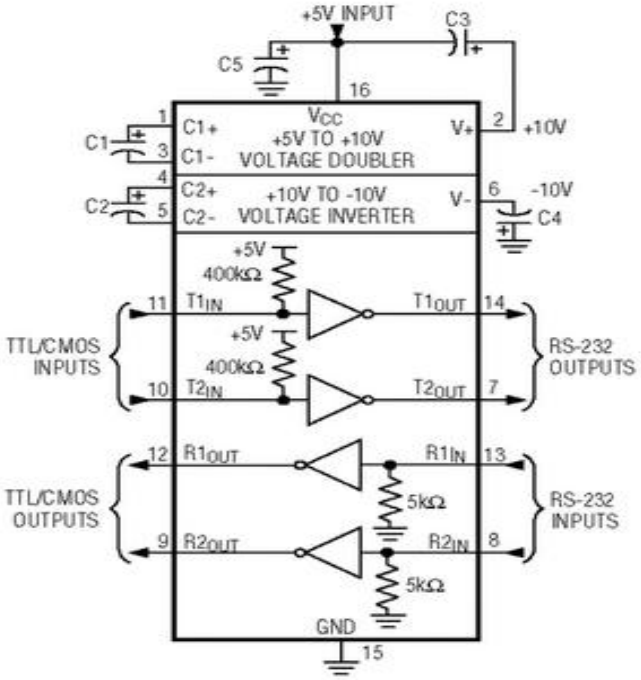

L293D Logic Table.:

- Pin 2 = Logic 1 and Pin 7 = Logic 0 | Clockwise Direction
- Pin 2 = Logic 0 and Pin 7 = Logic 1 | Anticlockwise Direction
- Pin 2 = Logic 0 and Pin 7 = Logic 0 | Idle [No rotation] [Hi-Impedance state]
- Pin 2 = Logic 1 and Pin 7 = Logic 1 | Idle [No rotation]

MAX232 DESCRIPTION

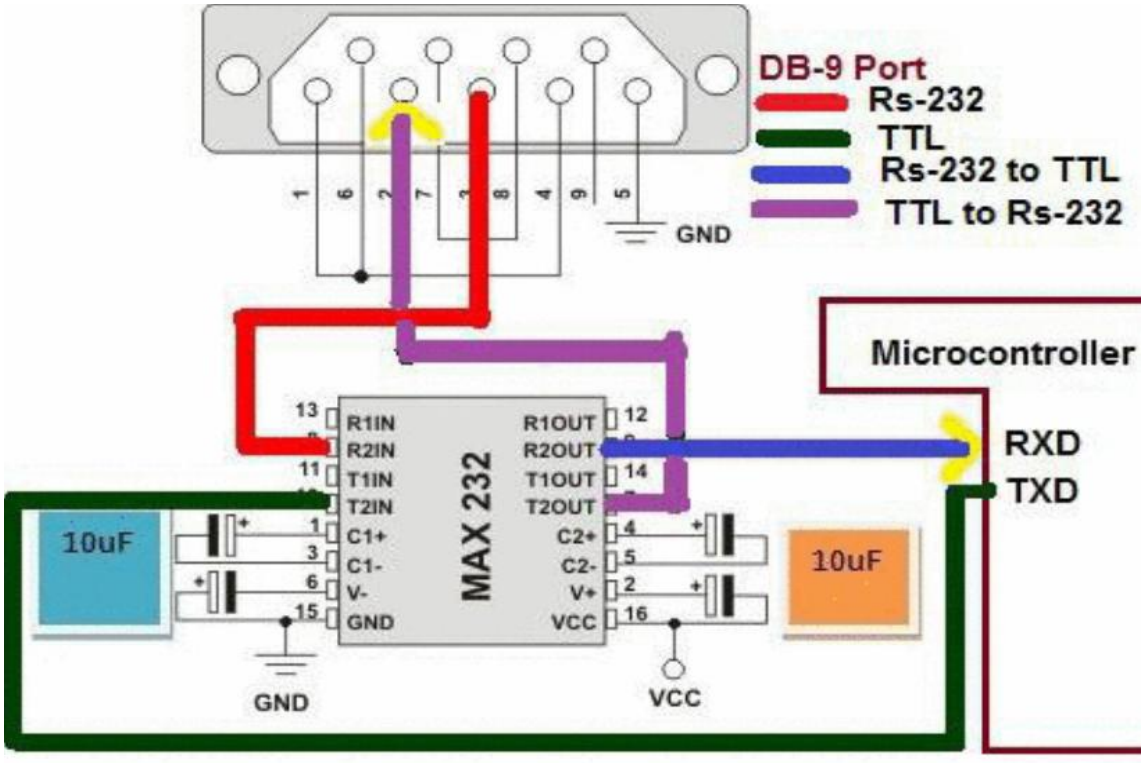



Pin Diagram




Pin Description

Max 232 is an ic (integrated circuit) that converts TTL(Transistor Transistor logic) logic signal in to its equivalent RS-232c level signal and Rs-232c level to its equivalent TTL level signal. This ic is very important in case when we need to make connection and transfer data between devices that works on different wave forms.
For example, Most of the microcontrollers 8051(89c51,89c52), PIC(16f877), AVR works on TTL logic wave form. These microcontrollers have a build in UART(Universal Synchronous-Asynchronous Receiver& Transmitter) which can send and receive serial data. Since they work on TTL level so they transmit and receive data on TTL wave from. Whereas Standard PC(Personal Computers) works on RS-232 level wave form. Now if we need to transfer data from microcontroller to PC(Personal computer) we need to convert data from TTL to RS-232 level and if we want to send data from PC to microcontroller we have to convert data from Rs-232 to TTL. MAx-232 is solution to this problem. There is wide range where we use max232 but its main purpose is explained above.

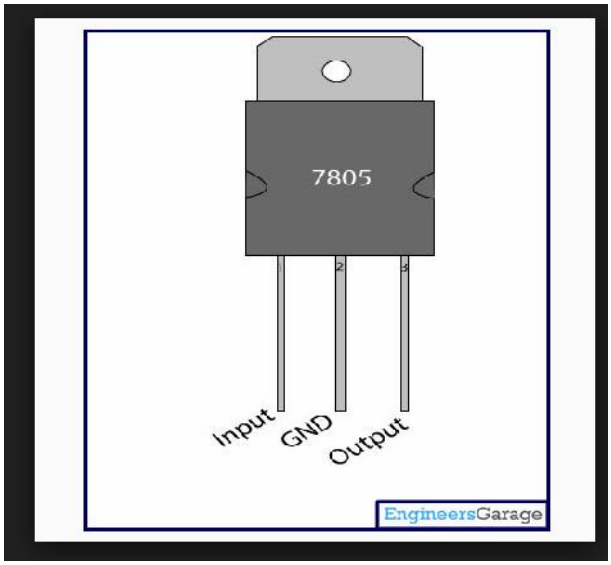


MAx232 has 16 pins. It requires four external capacitors for its proper configuration. Capacitors can range between 8uf to 10uf and are of up to 16v. Pin names With functions are listed below.
Suppose max-232 is connected to Pc or microcontroller.
PIN 1(C1+) Connect positive leg of a capacitor to it.
PIN 2(Vs+) Connect positive leg of a capacitor to it, and make negative leg of same capacitor ground.
PIN 3(C1-) Connect negative leg of a capacitor to it, whose positive leg is connected to Pin#1.
PIN 4(C2+) Connect positive leg of a capacitor to it.
PIN 5(C2-) Connect negative leg of a capacitor to it, whose positive leg is connected to Pin#4.
PIN 6(Vs-) Connect negative leg of a capacitor to it and apply 5 volts to positive leg of the same capacitor.
PIN 7(T2OUT)
Outputs the converted Transmitted signal. Signal is received from Pc or microcontroller etc at T1IN Pin. Connect this pin to Pin#2 of DB-9 serial port of your PC or Rxd Pin of youe microcontroller, actually this pin transmits the transformed signal from TTL to RS-232 level or Rs-232 to TTL. Pin#2 of DB-9 port is Rxd(Rxd means This pin receives Transmitted Signal(data)).
PIN 8(R2IN)
This pin Receives transmitted signal from Pc or microcontroller etc. This pin receives signal transmitted from Txd pin. Connect this pin to Pin#3 of DB-9 port of your PC or Txd pin of microcontroller. Pin#3 of DB-9 port is Txd (Txd means This pin transmits data).
PIN 9(R2OUT)
Outputs the converted received signal. Signal is recevied from Pc or microcontroller etc. at R1In Pin. Connect this pin to your module Rxd pin which receives the signal.
PIN 10(T2IN)
Recevies the transmitted signal from pc or microcontroller etc. Signal is transmitted from txd pin. Connect this pin to your module Txd pin

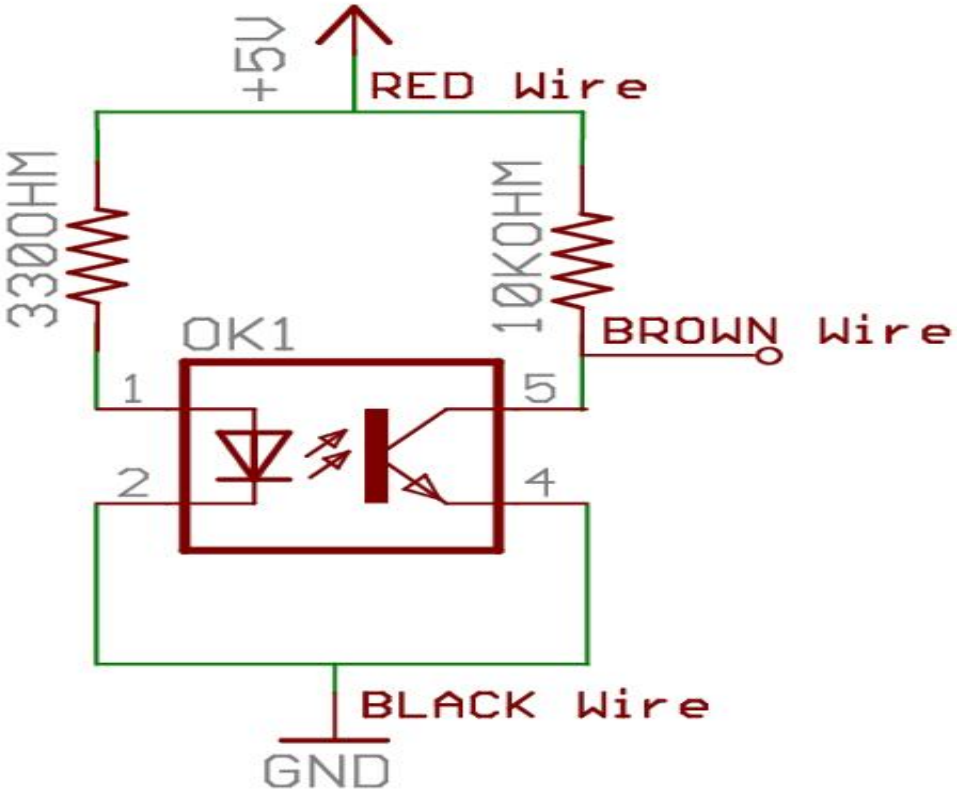
- PIN 11(T1IN) Woks same as T1IN.
- PIN 12(R1OUT) Works same as R2OUT.
- PIN 13(R1IN) Works same as R2IN.
- PIN 14(T1OUT) Works same as T2OUT.
- PIN 15(GND) Ground this pin.
- PIN 16(vcc) Apply 5 volts to this pin.

Max232 has two line drivers. You can make connections between four uarts at a time. Diagram below will clear you about line drivers, pin functions and connections. Data flow is clearly visible.

## 7805




## Optical Encoder



# PID CONTROLLER

## 4.3 Introduction:

A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems– a PID is the most commonly used feedback controller. 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. In the absence of knowledge of the underlying process, a PID controller is the best controller. However, for best performance, the PID parameters used in the calculation must be tuned according to the nature of the system – while the design is generic, the parameters depend on the specific system.

The PID controller calculation involves three separate parameters, and is accordingly sometimes called three-term control: the proportional, the integral and derivative values, denoted as P,I, and D. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element. Heuristically, these values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change.

## 4.4 The Three Elements of PID:

In simple terms, the each serves a purpose in the control mechanism.

### 4.4.1 Proportional Controller ($K_P$)

As the name suggests, a proportional controller applies power to the heater in proportion to the difference in temperature between the output and the reference. Thus, the P term is referred to as the proportional gain of the controller. On its own, the characteristic of the resulting output temperature will be such that it will typically stabilize just below the desired reference temperature. This is so because; as its gain is increased, the system responds by applying more power to the output, and as a result the temperature rises quickly and approaches closer to the set-point. But as this happens, the system will react by lowering the gain since the gap to the reference is now getting smaller. This causes the response to become progressively under-damped as the output temperature gets closer to the set-point. This difference between the stabilized output and the reference is called the Steady State Error. However, if the P gain is set too high, there will be excessive initial overshoot, after which the output will oscillate near the reference level, and eventually becomes unstable. On the other hand, a gain set too low will never enable the output to even reach near the reference level. The responses can be seen in Figure 8.6 below. While the proportional gain method will perform better than the unavoidable oscillation in a simple on-off control system, on its own, it will never stabilize on the set-point. The reason is, acting alone, if temperature equals set-point, then there is no error, subsequently, the gain will be zero, and thus the output will also be zero.

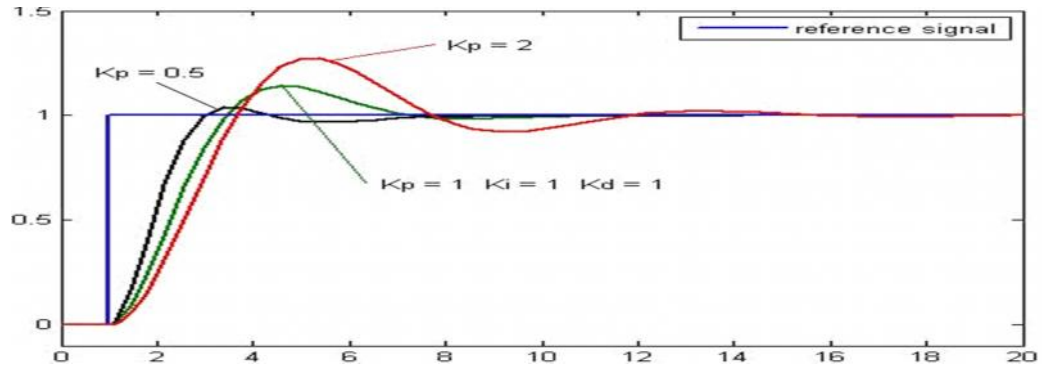



Fig 4.3 Proportional Control Response

**4.4.2 Integral Controller** ($K_I$):

To resolve the issue of the steady state error with Kp alone, the integral term, Ki, has to be used. The characteristic of the integral control is that it performs an integration of the past error values and applies a gain to minimize this error. The effect of this action is that it changes the heater power continuously based on past performance until the time-averaged value of the temperature error is zero. The downside to this is that the initial overshoot may be increased since during the initial moments of heating, the difference between actual temperature and set-point is usually greater. And since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the set-point value. As such, settling time may also take longer due to this temperature oscillation, and as past averaged error values will take time to decrease as the error gets smaller over a period of time. However, together with Kp, a PI controlled system is able to achieve the temperature requested by the set-point by eliminating the steady state error, and also have a shorter rise time.

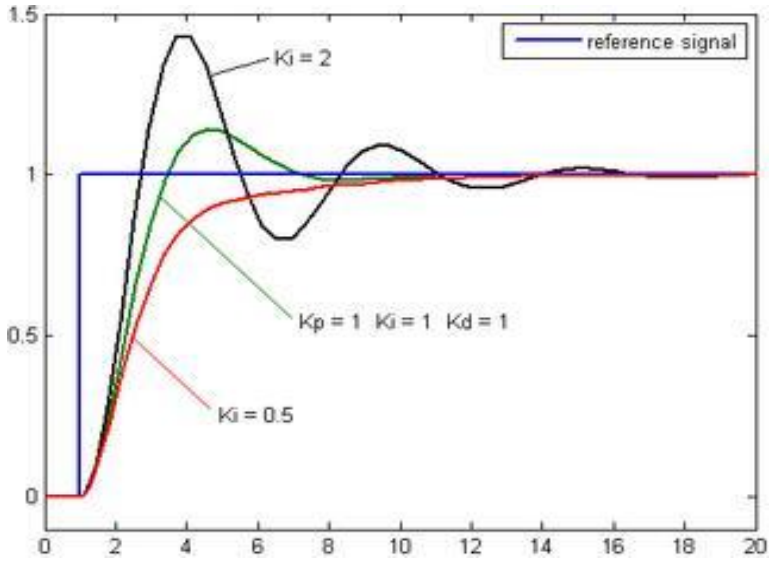



Fig 4.4 system variation with change in integral control

**4.4.3 Derivative Controller** (Kd):

The derivative term comes into play when theovershoot and oscillation need to be addressed. The rate of change of the error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain Kd. The magnitude derivative term introduces damping tothe overall system output by slowing down the rate of change of the controller output such that the overshoot and oscillation can be reduced. But one side effect is that is that the damping may also increase the rise time slightly. Together with P and I control terms, the derivative term can help to improve the process stability of the system. But according to some studies, differentiation of a signal amplifies noise and can cause a system to become unstable, so if the system is susceptible to a lot of noise or is sensitive to noise, then it may be necessary to leave out the derivative control. Figure 8.7 illustrates the effects of a PD controlled response.

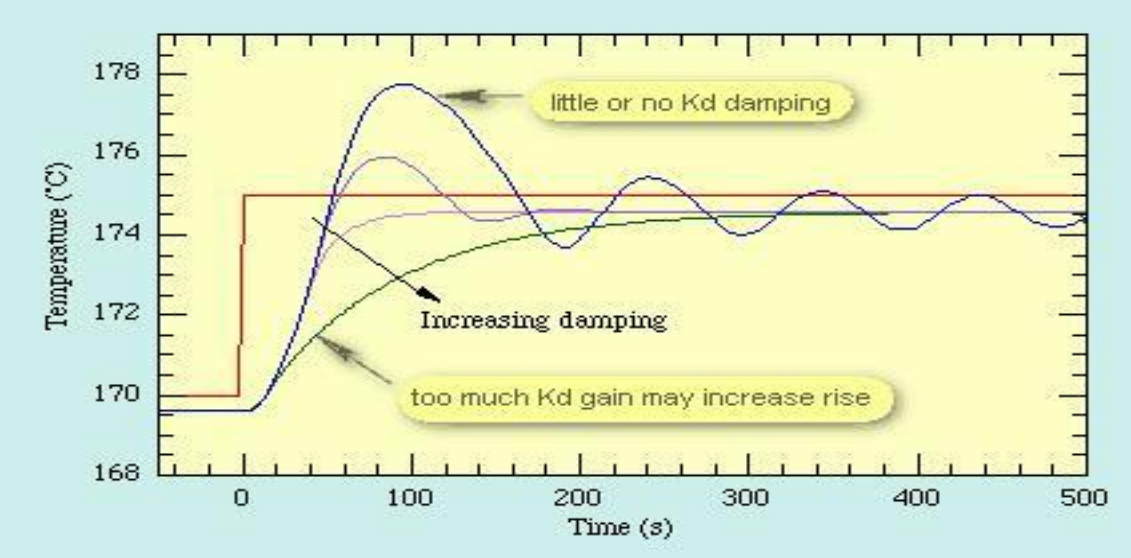


Fig 4.5 Proportional-Derivative Control Response

## 4.5 CLOSED LOOP SYSTEM WITH PROPORTIONATE, INTEGRAL AND DERIVATIVE CONTROL:

To summarize all three controls, proportional control causes an input signal to change as a direct ratio of the error signal variation. It responds immediately to the current tracking error but it cannot achieve the desired set-point accuracy without an unacceptably large gain.

Thus, proportional term usually needs the other terms. Integral control causes an output signal to change as a function of the integral of the error signal over time duration. Integral term yields zero steady-state error in tracking a constant set-point. It also rejects constant disturbances. Derivative action reduces transient errors and causes an output signal to change as a function of the rate of change of the error signal. The contributions of the three terms will yield the control output, or the control variable:

Control Variable = Pout + Iout + Dout

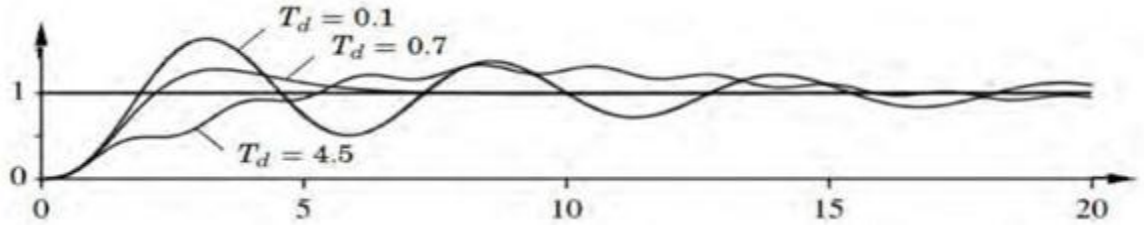


Fig 4.6 Closed loop system with proportional, integral, and derivative control.

## 4.5.1TUNING THE PID CONTROLLER:

The setting up a PID controller is to tune or choose numerical values for the PID parameters. PID controllers are tuned in terms of their P, I, and D terms. Tuning the control gains can result in the following improvement of responses:

**Proportional gain (Kp):**

Larger proportional gain typically means faster response, since the larger the error, the larger the proportional term compensation. However, an excessively large proportional gain may result in process instability and oscillation.

**Integral gain (Ki):**

Larger integral gain implies steady-state errors are eliminated faster. However, the trade-off may be a larger overshoot, since any negative error integrated during transient response must be integrated away by positive error before steady state can be reached.

**Derivative gain (Kd):**

Larger derivative gain decreases overshoot but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

The following table lists some common tuning methods and their advantages and disadvantages. The choice of method will mostly depend on whether or not the loop can be taken offline for tuning, and the response time of the system. If the system can be taken offline, the best tuning method often involve s subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters. Manual tuning methods can be quite inefficient, especially if the loops have response times on the order of minutes or longer.

## Methods of Determining RPM:

There are two methods for determining RPM:

1. The Frequency measurement method.
2. The period measurement method.

Frequency measurement is better for fast-moving devices such as motors and turbines that typically turn in thousands of revolutions per minute.

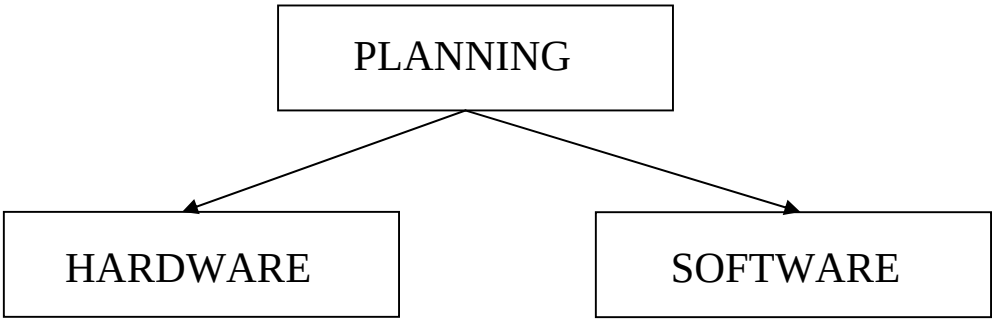
Period measurement is better for devices that move more slowly, such as shafts that turn in less than 10 RPM.

High PPR Solutions Using the Frequency Measurement Method:

For this discussion, high PPR is considered to be at least 60 PPR. When using high PPR sensors, such as shaft encoders or proximity sensors sensing gear teeth, the easiest way to determine RPM is to monitor the pulse frequency from the sensor using a digital input module and the Get Frequency command in PAC Control Professional. Then calculate the RPM using this equation:

$$\text{RPM} = \frac{(\text{Pulse Frequency in pulses/sec}) \times (60 \text{ sec/min})}{(\text{Sensor pulses/revolution})} = \frac{\text{Revolutions}}{\text{Minute}}$$

## 4.6 Project Planning:




## 4.6.1 Software Planning

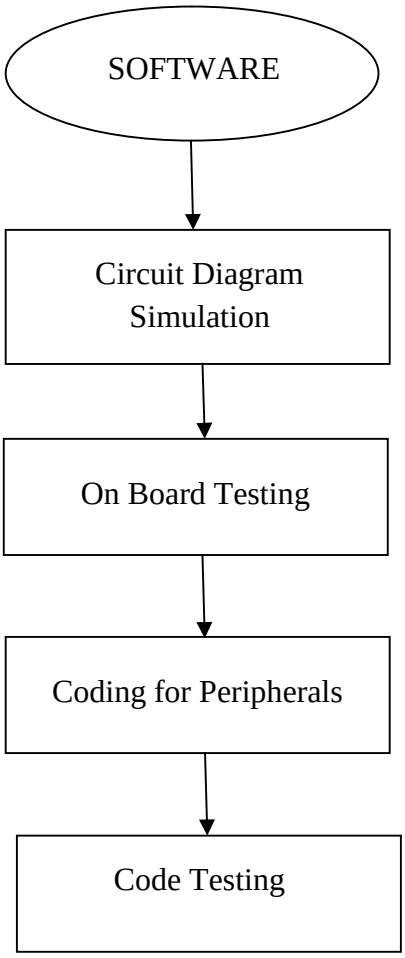



Fig 4.7 Flowchart for Software Planning

### 4.6.2 Hardware Planning

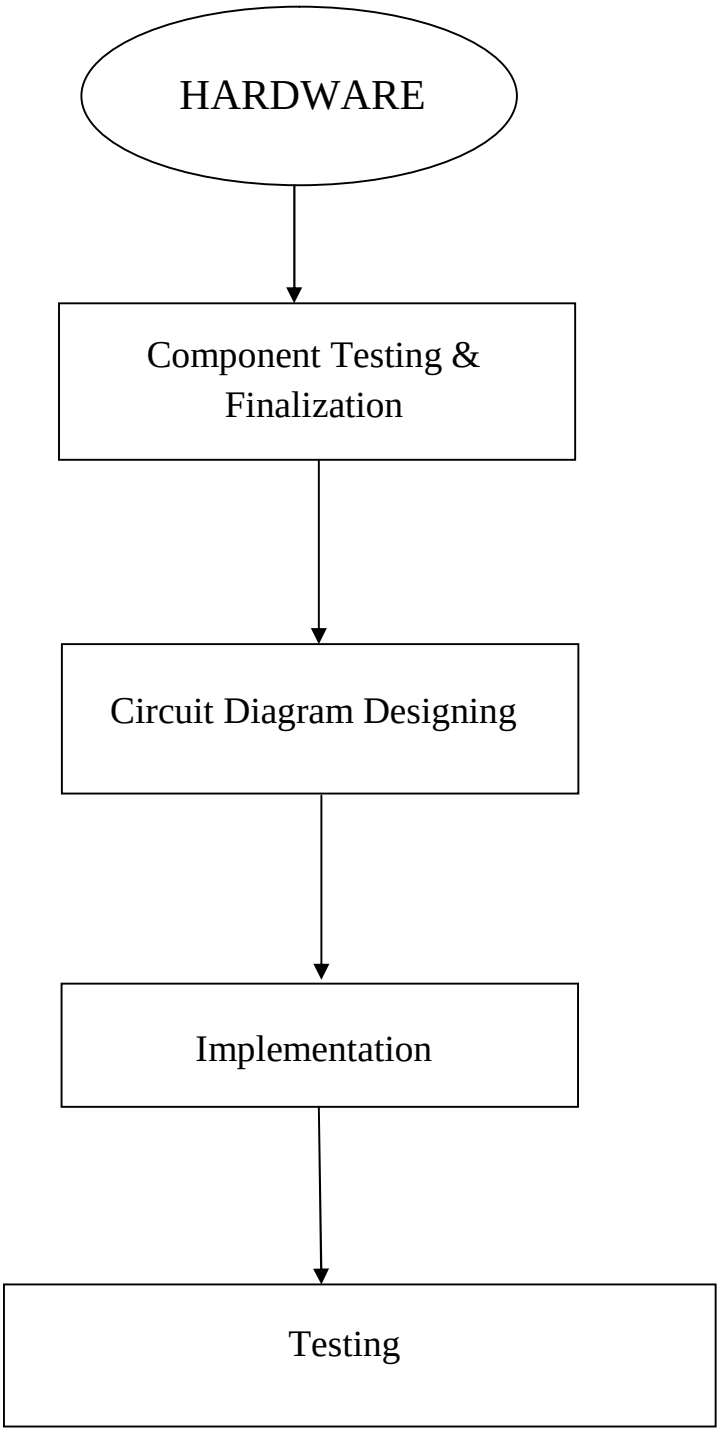



Fig 4.8 Flowchart for Hardware Planning

# Chapter 5

# Code Algorithm

## 5.1 Algorithm 1 (microcontroller):

- Get serial data from computer on pin 26

- Generate PMW pulses on pin 17 given to L293D

- Obtain feedback on pin 15 from optical encoder

- Enable the microcontroller to work in automatic mode

-Create a log file to store data from each node

- Calculate the error

- Adjust the frequency of PWM pulses on pin 17 accordingly

## 5.2 Algorithm 2(for matlab):

- Run the GUI code on MATLAB

- Enter external Kp, Kd, and Ki values

- Start the simulation

- Observe the waveform

- Check is the desired speed is obtained

- If the desired speed is not obtained then change the value of constants

- If yes then stop the simulation

# CHAPTER 6

# PERFORMANCE AND RESULTS

## 6.1 SOFTWARE

GUI to set point and vary values of proportional, integral, derivative gain.
It consists of controls like start (to start the motor) , stop (to stop the motor) , refresh (to vary the values of PID constants) , Set point (to enter the speed desired).
It also consists of graphical display which plots a graph of actual motor speed versus time which is obtained on the hardware kit.
The GUI window is as shown below.

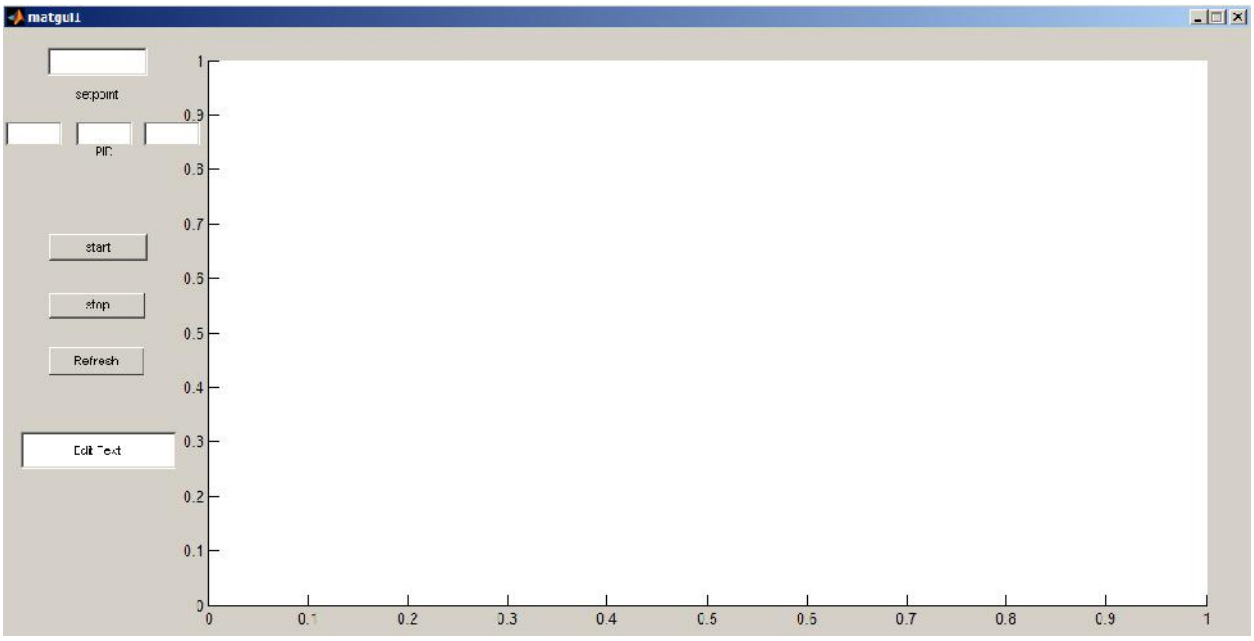



Fig 6.1

## 6.2 HARDWARE

The hardware is as shown below:

Components used – 1) PIC16F877A

2) L293D

3) MAX232

4) Optical Encoder

5) Voltage Regulator (7805)

6) 12v 150rpm DC Motor

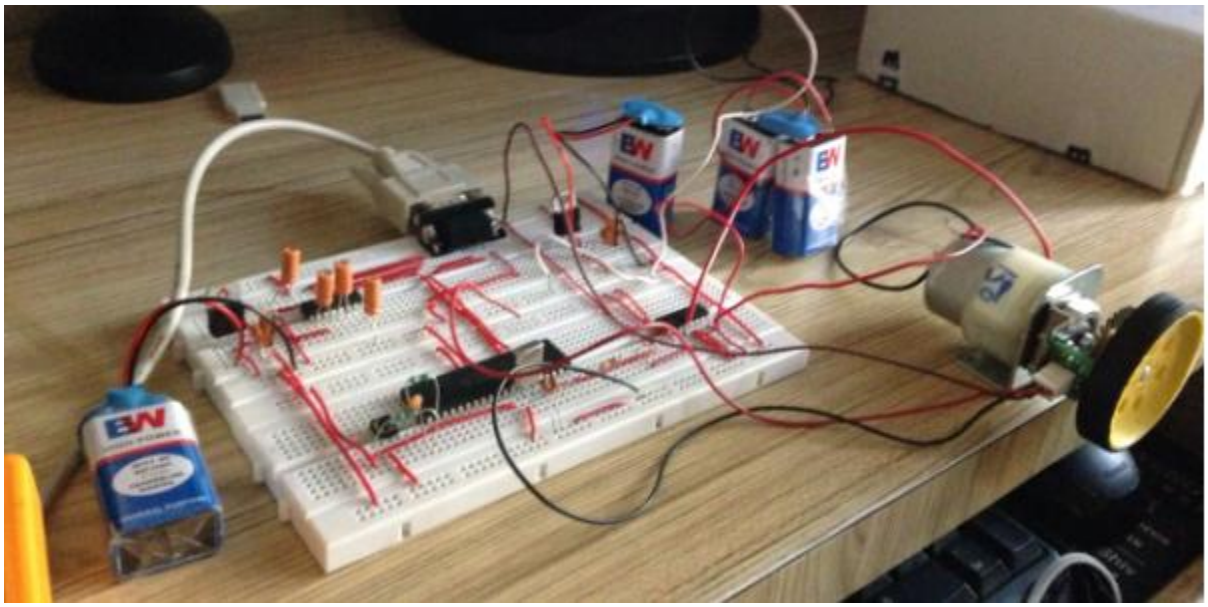


Fig 6.2 Hardware

## 6.3 Output on GUI

As shown in the graph below, the desired speed of the motor (set point) is set to 100 rpm.

The actual speed obtained is plotted which is close to the desired value.

The PID controller takes the necessary action to maintain the desired speed of the motor

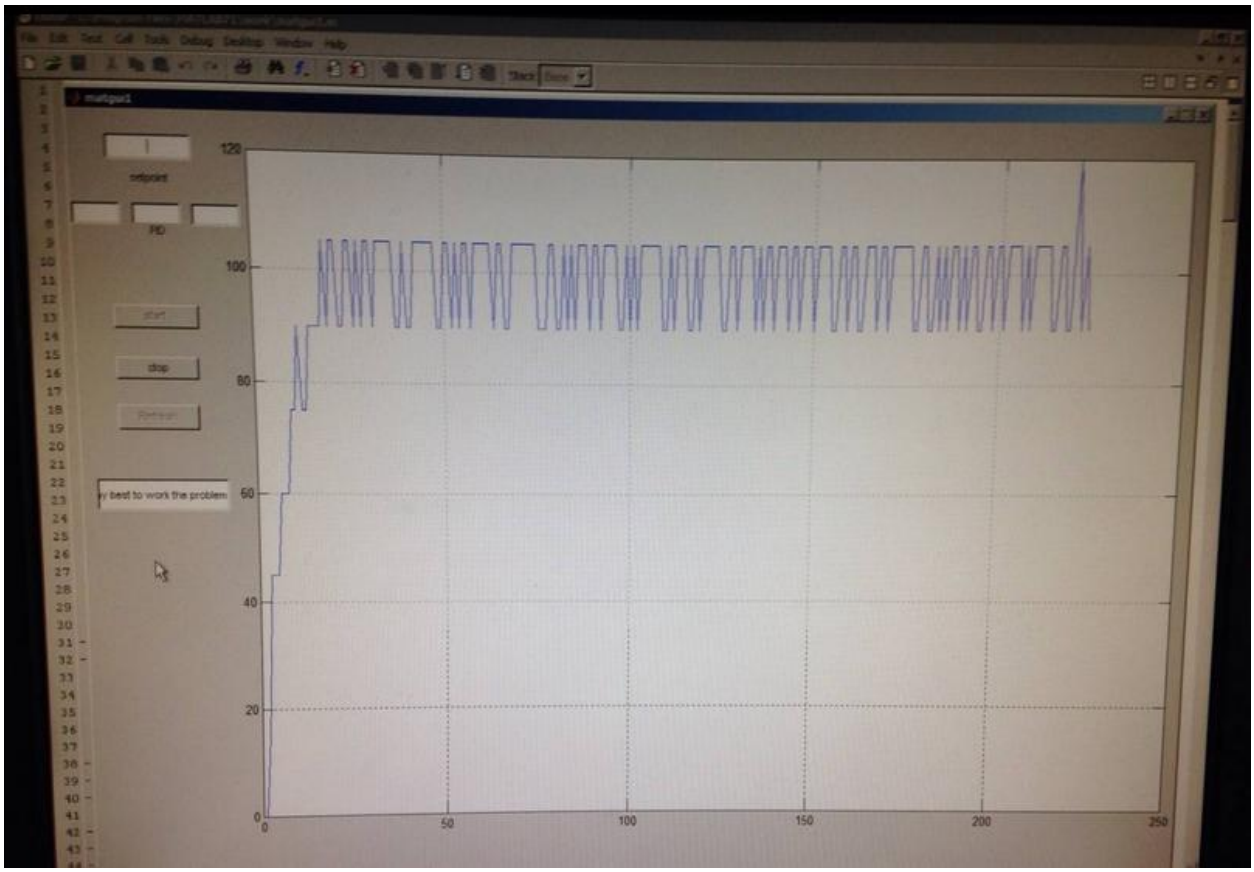



Fig 6.3 Output on GUI

PERFORMANCE:

Drawbacks of open loop control:

1) Since feedback is not present , it is difficult to obtain the desired speed.

2) The steady state error and transient errors increase significantly.

Advantages of closed loop control:

1) Due to feedback provided through optical encoder, the settling time is less i.e. the desired speed is obtained faster.

2) Even if the load on the motor shaft increases, to maintain the desired speed, the PID controller takes the necessary action on the basis of feedback provided by the closed loop system.

Analysis of DC Motor using different controllers – P, PI, PID

P: The actual speed of the motor reaches close to the desired speed but there exists a steady state error.

PI: The I term eliminates the steady-state error but introduces a initial peak overshoot.

PID: It eliminates the steady-state error , peak overshoot and also reduces the settling time.

# CHAPTER 7

# CONCLUSION AND SCOPE

# Conclusion

We made an attempt to design PID controller operation using pic16f877A though there are various types of other processes which can execute the function of PID controller like lab view software, hardware circuit, microprocessor etc.,

Reason for doing this project is to show that this method of PID operation decreases labor when compared to other methods but what we felt during processing our project is there is lot of possibility to short circuit as it includes large number of connections and there is also the possibility of pic failure as it could be overloaded when l293d fails.

# Future Scope

In this project only simple controllers were implemented. These were PD position PID position and PI rate controllers. In the future it would be interesting to develop and implement more advanced controllers such as two degree of freedom and robust controllers. Even a neutral net controller may be implemented if it is simple enough to fit in the remaining ROM space on the PIC. Handshaking between matlab and the pic maybe helpful in ensuring more reliable operation.

# REFERENCES

## REFERENCES:


[1] Power Electronics Book by Muhammad H. Rashid.

[2] Fundamental of Microcontrollers and Applications in Embedded Systems by Ramesh S. Gaonkar, Tata McGraw Hill, Third Publication.

[3] PIC Microcontroller Projects In C: Basic to Advance by Dogan Ibrahim

[4] Alleyne, Andrew G., Block, Daniel J., Meyn, Sean P., Perkins, William R., and Spong, Mark W. "An Interdisciplinary, Interdepartmental Control Systems Laboratory." IEEE Control Systems Magazine, v25, n1, February 2005.

[5] Choi, Chiu H. "Undergraduate Controls Laboratory Experience." Proceedings of the 2004 American Society for Engineering Education Annual Conference and Exposition, Jun 20-23 2004.

[6] Dixon, Warren E., Dawson, Darren M., Costic, B. T., Queiroz, Marcio S. "A MATLAB-Based Control Systems Laboratory Experience for Undergraduate Students: Toward Standardization and Shared Resources." IEEE Transactions on Education, v 45, n 3, Aug. 2002.

[7] Berstein, Dennis S. "The Quanser DC Motor Control Trainer." IEEE ControlSystems Magazine, v25, n3, June 2005.

[8] "User Guide: Quanser Engineering Trainer DC Motor Control", QuanserConsulting Inc.

[9] "PIC16F87X Data Sheet." Microchip Technology Inc., 2001.

[10] Engle, Benjamin. "WSU QET Interface Help System." November 2007.

[11] http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[12] http://electronics.stackexchange.com/questions/156475/float-to-short-conversion-error-in-c

[13] http://electronics.stackexchange.com/users/66917/dcmotor?tab=questions&sort=votes

[14] http://www.nex-robotics.com/

# APPENDIX

## Code 1 (for microcontroller):

```
floatpro,i,der,pOut,iOut,dOut,out;
interror,lastError,count=0,sel=1,overflow_bit=0,timer_zero_bit=0;
unsigned short duty=0;
unsignedint setPoint,actualOut,storeOne,higherBit=0,lowerBit=0,refresh,adder,reader;
char start;
int apple[4];

floatpid(intactualOut)
{
error=setPoint-actualOut;

pOut=error*pro;
iOut=iOut+(i*error);
if(iOut>255)
iOut=255;
else if(iOut<0)
iOut=0;
dOut=-(actualOut-lastError)*der;
out = pOut+iOut+dOut;
lastError=actualOut;
if(out>255){
out=255;
    }
else if(out<0){
out=0;
    }

return out;
  }
voidpwm(short duty){
  PWM1_Init(5000);
  PWM1_Start();
  PWM1_Set_Duty(duty);
 }
unsignedint feedback()
   {
overflow_bit=0;
       OPTION_REG=0;
       T1CON=0;
       INTCON=0;
       ADCON0=0;
       CMCON = 0x07;
sel=1;
```

```
        TMR0=0;
        TMR1H=0x00;
        TMR1L=0x00;
        INTCON=0xE0;
        OPTION_REG.PSA=0;
        OPTION_REG.PS2=1;
        OPTION_REG.PS1=0;
        OPTION_REG.PS0=1;
        OPTION_REG.T0CS=0;
        T1CON=0x07;
        TMR1IE_bit = 1;
        TMR1IF_bit = 0;

while(sel==1){
        }
storeOne=(higherBit*256)+lowerBit;
actualOut=(60*storeOne*2)/24;                   //24 pulses per rev

returnactualOut;
        }
void interrupt(){
if(TMR0IF_bit==1){
overflow_bit++;

if(overflow_bit==38){                           //timer0 will overflow 38 times
lowerBit = TMR1L;
higherBit = TMR1H;
timer_zero_bit=TMR0;
       OPTION_REG=0;
       T1CON=0;
       TMR0IF_bit = 0;
       TMR1IF_bit = 0;
       TMR0IE_bit = 0;
       TMR1IE_bit = 0;
sel=0;
       }
       TMR0IF_bit = 0;
       }
      }
void main() {
   UART1_Init(9600);
   PWM1_Init(5000);          //added on 3/17
   PWM1_Start();
   PWM1_Set_Duty(0);
start='i';                 //added on 3/17
    TXSTA=0x24;
    RCSTA=0x90;
```

```
pro=3,i=0.8,der=1.2;    // proportional,integral,derivative gains
setPoint=100,sel=1;       // setpoint
actualOut=0;                //actualoutput
pOut=0,iOut=0,dOut=0;
out=16;
error=0,lastError=0;
  OPTION_REG=0;
  T1CON=0;
  INTCON=0;
  ADCON0=0;
  CMCON = 0x07;
    TRISC.RC0=1;
    TRISC.RC2=0;
    TRISC.RC6=0;
     PORTC=0;

while(1){
if (UART1_Data_Ready()||start=='s'){
if (UART1_Data_Ready()){
start = UART1_Read();
          PORTB=1;
          TRISB=0;
delay_ms(500);
          PORTB=0;
           }
if(start=='s'){

out = pid(actualOut);

duty=(unsigned short)out;
pwm(duty);
actualOut = feedback();

       UART1_Write(actualOut);


   }
else if(start=='g'){
       PWM1_Init(5000);
       PWM1_Start();
       PWM1_Set_Duty(0);
actualOut=0;
error=0,lastError=0;
pOut=0,iOut=0,dOut=0;
   }
```

```
else if(start=='r'){
    UART1_Write(1);
adder=0;
while(adder!=4){
refresh=1;

while(refresh==1){
if(UART1_Data_Ready()){
apple[adder]=UART1_Read();      // reads data from matlab

refresh=0;
    UART1_Write(apple[adder]);      //transmits back to matlab
adder++;
    }
 }
 }
setPoint=apple[0];
pro=apple[1];
i=apple[2];
der=apple[3];

 }
   }
```

## Code 2 (for matlab)

```
functionvarargout = matgui1(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @matgui1_OpeningFcn, ...
'gui_OutputFcn',  @matgui1_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
ifnargin&&ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end

ifnargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
function matgui1_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;

guidata(hObject, handles);

set(handles.stop,'Enable','off');

s = serial('COM1');
set(s,'BaudRate',9600);
set(s,'Timeout',20);
set(s,'ReadAsyncMode','continuous');
fopen(s);
handles=guidata(hObject);
handles.set=s;
guidata(hObject,handles);



functionvarargout = matgui1_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

/Executes on button press in start.
functionstart_Callback(hObject, eventdata, handles)

set(handles.start,'Enable','off');
set(handles.Refresh,'Enable','off');
set(handles.stop,'Enable','on');
drawnow;
```

```
  s=handles.set;
flushinput(s);          //input buffer to be flushed
fprintf(s,'%c','s');
i=1;
 string1='start';
set(handles.edit5, 'String', string1);
drawnow;
condition=1;
handles=guidata(hObject);
handles.con=condition;
guidata(hObject,handles);
while(condition==1)
if(s.BytesAvailable>0)
       a = fread(s,1,'uint8');
t(i)=a;
drawnow;
plotData=plot(t,'linewidth',1);
drawnow;

gridon;
drawnow;
i=i+1;
      string1='inside the start callback';
set(handles.edit5, 'String', string1);
end
     string1='i am trying my best to work the problem';
set(handles.edit5, 'String', string1);
handles=guidata(hObject);
condition=handles.con;
end
     string1='i did it';
set(handles.edit5, 'String', string1);
drawnow;
-
function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)
ifispc&&isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
```

```
function edit2_Callback(hObject, eventdata, handles)


function edit2_CreateFcn(hObject, eventdata, handles)
ifispc&&isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
function edit3_CreateFcn(hObject, eventdata, handles)
ifispc&&isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
function edit4_CreateFcn(hObject, eventdata, handles)
ifispc&&isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

//--- Executes on button press in stop.
functionstop_Callback(hObject, eventdata, handles)
condition=0;
handles=guidata(hObject);
handles.con=condition;
guidata(hObject,handles);
set(handles.stop,'Enable','off');
set(handles.start,'Enable','on');
set(handles.Refresh,'Enable','on');
drawnow;
setup=handles.set;
fprintf(setup,'%c','g');
flushinput(setup);
 string1='stop';
set(handles.edit5, 'String', string1);
debugger=0;
while(debugger~=50)
       string1='stopping na ';
set(handles.edit5, 'String', string1);
debugger=debugger+1;
end
```

```
% --- Executes on button press in Refresh.
functionRefresh_Callback(hObject, eventdata, handles)
   string1='inside refresh';
set(handles.edit5, 'String', string1);
drawnow;
setup=handles.set;
if( isempty(get(handles.edit1,'String'))||
isempty(get(handles.edit2,'String'))||isempty(get(handles.edit3,'String'))||isempty(get(handles.edit
4,'String')) )
     string1='please fill all the values';
set(handles.edit5, 'String', string1);
drawnow;
else
set(handles.stop,'Enable','off');
set(handles.start,'Enable','on');
set(handles.Refresh,'Enable','off');
drawnow;
t(1)=str2num(get(handles.edit1,'String'));
t(2)=str2num(get(handles.edit2,'String'));
t(3)=str2num(get(handles.edit3,'String'));
t(4)=str2num(get(handles.edit4,'String'));

flushinput(setup);
fprintf(setup,'%c','r');

refresh=0;
while(refresh==0)
if(setup.BytesAvailable>0)
refresh = fread(setup,1,'uint8');
end
end
adder=1;

while(adder~=5)
fwrite(setup,t(adder),'uint8');
chaos=0
while(chaos==0)
if(setup.BytesAvailable>0)
setp(adder)=fread(setup,1,'uint8');
chaos=1;
end
end
```

```
adder=adder+1;

end
  string1='done';
set(handles.edit5, 'String', string1);
drawnow;
set(handles.Refresh,'Enable','on');
end


function edit5_Callback(hObject, eventdata, handles)
function edit5_CreateFcn(hObject, eventdata, handles)
ifispc&&isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
```

# PIC16F877A SPECIFICATIONS

| Parameter Name | Value |
|---|---|
| Program Memory Type | Flash |
| Program Memory (KB) | 14 |
| CPU Speed (MIPS) | 5 |
| RAM Bytes | 368 |
| Data EEPROM (bytes) | 256 |
| Digital Communication Peripherals | 1-UART, 1-A/E/USART, 1-SPI, 1-I2C1-MSSP(SPI/I2C) |
| Capture/Compare/PWM Peripherals | 2 CCP |
| Timers | 2 x 8-bit, 1 x 16-bit |
| ADC | 8 ch, 10-bit |
| Comparators | 2 |
| Temperature Range (C) | -40 to 125 |
| Operating Voltage Range (V) | 2 to 5.5 |
| Pin Count | 40 |

| Specification | Value |
|---|---|
| Family | PIC16F877A |
| Number of Timers | 3 |
| On-Chip ADC | 8-chx10-bit |
| Program Memory Size | 14.3KB |
| Program Memory Type | Flash |
| RAM Size | 368Byte |
| Interface Type | CCP/I2C/MSSP/SPI/USART |
| Family Name | PIC16 |
| Minimum Operating Supply Voltage | 4V |
| Typical Operating Supply Voltage | 5V |
| Maximum Operating Supply Voltage | 5.5V |
| Minimum Operating Temperature | -40°C |
| Maximum Operating Temperature | 85°C |
| Screening Level | Industrial |
| Mounting | Through Hole |

| Specification | Value |
|---|---|
| Device Core | PIC |
| Packaging | Tube |
| Pin_Count | 40 |
| Supplier_Package | PDIP |
| Product Width | 13.84mm |
| Product Height | 3.81mm |
| Product Length | 52.26mm |
| Product Type | Microcontroller |
| Data Bus Width | 8Bit |
| Instruction Set Architecture | RISC |
| Maximum Clock Rate | 20MHz |
| Maximum Speed | 20MHz |
| Number of Programmable I/Os | 33 |

Report a problem
Suggest a product