

UNIVERSITI MALAYSIA PAHANG

BORANG PENGESAHAN STATUS TESIS*
PID CONTROL SYSTEM IMPLEMENTATION IN
EMBEDDED SYSTEM FOR DC MOTOR SPEED
CONTROL

JUDUL:

SESI PENGAJIAN: 2008/2009

Saya ARIFF BIN CHE MOHD NOOR (850725-09-5029)
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/Sarjana /Doktor Falsafah)* ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang (UMP).
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (✓)

☐

SULIT

(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)

☐

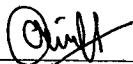
TERHAD

(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)

☒

TIDAK TERHAD

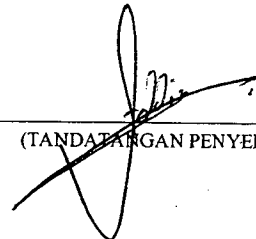
Disahkan oleh:



(TANDATANGAN PENULIS)

Alamat Tetap:

25 F, JALAN RAJA SYED ALWI,
01000, KANGAR,
PERLIS



(TANDATANGAN PENYELIA)

ADDIE IRAWAN BIN HASHIM
(Nama Penyelia)

Tarikh: **23 OCTOBER 2008**

Tarikh: : **23 OCTOBER 2008**

CATATAN:

*

Potong yang tidak berkenaan.

**

Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu dikelaskan sebagai atau TERHAD.

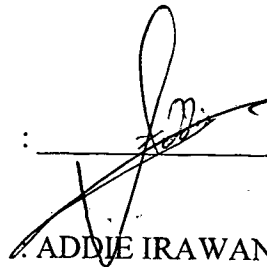
◆

Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).

"I hereby acknowledge that the scope and quality of this thesis is qualified for the award
of the Bachelor Degree of Electrical Engineering (Electronics)"

Signature

:



A handwritten signature in black ink, appearing to be 'Addie Irawan Hashim', is written over a horizontal line. The signature is stylized with a large loop at the top and a long horizontal stroke extending to the right.

Name

ADDIE IRAWAN HASHIM

Date

: 23 OCTOBER 2008

PID CONTROL SYSTEM IMPLEMENTATION IN EMBEDDED SYSTEM FOR DC
MOTOR SPEED CONTROL

ARIFF BIN CHE MOHD NOOR


This thesis is submitted as partial fulfillment of the requirements for the award of the
Bachelor of Electrical Engineering (Electronics)

Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

NOVEMBER, 2008

PERPUSTAKAAN UNIVERSITI MALAYSIA PAHANG	
No. Perolehan 037320	No. Panggilan TJ 223 PSS A75 2008
Tarikh 02 JUN 2009	11 B2.

“All the trademark and copyrights use herein are property of their respective owner. References of information from other sources are quoted accordingly; otherwise the information presented in this report is solely work of the author.”

Signature :  _____

Author : ARIFF BIN CHE MOHD NOOR

Date : 23 OCTOBER 2008

To my beloved mother and father

ACKNOWLEDGEMENTS

In preparing this thesis, I was in contact with many people, researchers, academicians, and practitioners. They have contributed towards my understanding and thoughts. In particular, I wish to express my sincere appreciation to my supervisor, En Addie Irawan Hashim, for encouragement, guidance, critics and friendship. Without his continued support and interest, this thesis would not have been the same as presented here.

I am also indebted to all the lecturers and staff of Faculty of Electrical & Electronics Engineering, Universiti Malaysia Pahang for their assistance in supplying the relevant equipment and also lending their knowledge for me to complete this project.

My fellow undergraduate students should also be recognized for their support and my sincere appreciation also extends to others who have provided assistance at various occasions. Unfortunately, it is not possible for me to list all of them in this limited space.

Thank you.

ABSTRACT

This project is focused on implementation of the Proportional (P), Integral (I) and Derivative (D) control system algorithms in microcontroller unit (MCU) for direct current (DC) Motor speed control. The PIC series, PIC18F2331 has been used to perform the processing of PID algorithms for DC motor control purpose. The focus is on 12 volt DC motor with 30 revolutions per minute (rpm) maximum speed. No-load case and loaded case are the scope for this research. Three experiments have been done to look how much PID control algorithms affect the performances on driving actual DC motor; PI algorithm experiment, PD algorithm experiment and PID algorithm experiment. The result shows that, implementation of PID algorithm in small scale MCU is possible. PID algorithm that has been implemented in MCU inside the DC motor controller module system can eliminate the steady state error and overshoot problem including settling time. By creating real time data acquisition software, the performance of the system is monitored and later on analyzed. It is later found out that the PID algorithm has been able to create faster settling time while the overshoot has been reduced to 5% and the steady-state has been successfully reduced. The impact of the load and no load application of the PID algorithm can be clearly seen by how the PID algorithm has helped the controller to drive a loaded DC motor to the desired speed which could not be achieved without the PID algorithm.

ABSTRAK

Projek ini memfokuskan kepada implementasi algoritma system kawalan “Proportional”, “Integral” dan “Derivative” di dalam mikropengawal untuk mengawal kelajuan motor arus terus. Mikropengawal yang kecil dan murah telah diprogramkan dengan sejenis algoritma untuk membetulkan masalah “steady-state error” untuk motor arus terus yang beroperasi menggunakan 12 Voltan arus terus dan dengan kelajuan 30 revolusi per minit (rpm). Skop projek ini adalah kawalan kelajuan terhadap kes tanpa beban. Tiga eksperimen dijalankan untuk melihat sejauh mana algoritma “PID” memainkan peranan dalam pemacuan motor arus terus. Ia terdiri dari pengawal “PI”, pengawal “PD” dan juga pengawal “PID”. Keputusan eksperimen menunjukkan, implementasi algoritma “PID” dalam mikropengawal adalah sesuatu yang boleh dilaksanakan. Algoritma “PID” yang telah dihasilkan diaplikasikan kedalam mikropengawal yang terdapat didalam modul pengawal kelajuan mampu melenyapkan “steady-state error” dan “overshoot” termasuk “settling time”. Dengan menghasilkan perisian “real time data acquisition” prestasi sistem boleh diawasi dan dianalisis. Didapati bahawa algoritma “PID” yang dihasilkan mampu mempercepatkan “settling time” dan juga mengurangkan masalah “overshoot” sebanyak 5 % dan “steady-state error” berjaya dikurangkan. Kesan algoritma “PID” tersebut dalam aplikasi yang menggunakan beban jelas kelihatan apabila algoritma tersebut berjaya membantu sistem pengawal untuk memacu motor arus terus ke tahap kelajuan yang diingini.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	DECLARATION	i
	DEDICATION	ii
	ACKNOWLEDGEMENTS	iii
	ABSTRACT	iv
	ABSTRAK	v
	TABLE OF CONTENTS	vi
	LIST OF TABLES	x
	LIST OF FIGURES	xi
	LIST OF SYMBOLS	xiv
	LIST OF APPENDICES	xv
 1	 INTRODUCTION	 1
	1.1 Background	1
	1.2 Objectives	2
	1.3 Scopes	2

2	LITERATURE REVIEW	3
2.1	DC Motor Characteristic	3
2.2	PID Implementation on DC Motor Close Loop Control	5
2.3	Adaptive PID	10
2.4	PID Tuning	11
2.4.1	Manual Tuning	12
2.5	Implementing a PID Controller Using a PIC18 MCU	13
3	IMPLEMENTATION OF PID CONTROLLER ALGORITHMS IN MICROCONTROLLER UNIT	15
3.1	Introduction	15
3.2	Encoder Configuration	16
3.3	DC Motor	18
3.3.1	Pulse Width Modulation	19
3.4	PID Algorithm	20
3.4.1	Error Calculations	22
3.4.2	Proportional Terms	23
3.4.3	Integral Terms	23
3.4.4	Derivative Terms	24
3.4.5	PID Output	25
3.5	Adaptive PID	25

4	GRAPHICAL USER INTERFACE	28
4.1	Introduction	28
4.2	PID Motor Control Panel	29
4.2.1	Data Transmission to the Controller	29
4.2.2	Performance Monitoring	32
5	RESULT, PERFORMANCE & ANALYSIS	33
5.1	Introduction	33
5.2	PID Tuning	34
5.3	Performance Without PID Controller	37
	Under No Load	
5.4	Performance With PID Controller	39
	Under No Load	
	5.4.1 PI Controller	39
	5.4.2 PD Controller	41
	5.4.3 PID Controller	43
5.5	Performance Without PID Controller	46
	Under Load	
5.6	Performance With PID Controller Under Load	49
	5.6.1 PI Controller	49
	5.6.2 PD Controller	51
	5.6.3 PID Controller	53

6	CONCLUSION & RECOMMENDATIONS	56
6.1	Conclusion	56
6.2	Costing & Commercialization	57
6.3	Recommendations	59
6.3.1	Real time sampling	59
6.3.2	Application of Better Tuning Method	59
6.3.3	Handling of Decimal Number	60
6.3.4	Application of Universal Serial Bus Interface (USB)	60
	REFERENCES	61
	Appendices A - D	62-65

LIST OF TABLES

TABLE NO.	TITLE	PAGE
2.1	Choosing a Tuning Method	11
2.2	Effect of Increasing Parameters	12
3.1	SPGH-150 DC Motor Specifications	18
5.1	Results for Finding Value of K_p	35
5.2	Results of the Three Controllers Working Under No Load	45
5.3	Results of the Three Controllers Working Under Load	55
6.1	Components Price List for Commercialization Purpose	58

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
2.1	Step Response of Open Loop System	4
2.2	Step Response With Proportional Control	7
2.3	PID Control With Small K_i and K_d	8
2.4	PID Control With Large K_i	9
2.5	PID Control	10
2.6	18F2331 Pin Diagram	13
3.1	Hardware Design	16
3.2	Sample of Output from Encoder	17
3.3	SPGH-150 DC Motor	18
3.4	Sample of a PWM Waveform	19
3.5	Flowchart of PID Algorithm Implemented In the MCU	21
3.6	Flowchart of the Adaptive PID Algorithm Implemented In the MCU	27
4.1	Data Transmission Panel	30

4.2	Flowchart of the Data Transmission Panel	31
4.3	Data Plotting Platform	32
5.1	Gain Tuning For $K_p = 5$, $K_i = 0$ and $K_d = 0$	34
5.2	Gain Tuning For $K_p = 10$, $K_i = 0$ and $K_d = 0$	34
5.3	Gain Tuning For $K_p = 20$, $K_i = 0$ and $K_d = 0$	35
5.4	Gain Tuning For $K_p = 10$, $K_i = 0$ and $K_d = 1$	36
5.5	Gain Tuning For $K_p = 10$, $K_i = 0.4$ and $K_d = 2$	36
5.6	Free Run on 10 RPM without Load	37
5.7	Free Run on 20 RPM without Load	38
5.8	Free Run on 30 RPM without Load	38
5.9	PI Controller for 10 RPM without Load	39
5.10	PI Controller for 20 RPM without Load	40
5.11	PI Controller for 30 RPM without Load	40
5.12	PD Controller for 10 RPM without Load	41
5.13	PD Controller for 20 RPM without Load	42
5.14	PD Controller for 30 RPM without Load	42
5.15	PID Controller for 10 RPM without Load	43
5.16	PID Controller for 20 RPM without Load	44
5.17	PID Controller for 30 RPM without Load	44
5.18	Figure Displaying How the Motor Is Connected To The Load	46
5.19	Free Run on 10 RPM with Load	47
5.20	Free Run on 20 RPM with Load	48

5.21	Free Run on 30 RPM with Load	48
5.22	PI Controller for 10 RPM with Load	49
5.23	PI Controller for 20 with Load	50
5.24	PI Controller for 30 RPM with Load	50
5.25	PD Controller for 10 RPM with Load	51
5.26	PD Controller for 20 RPM with Load	52
5.27	PD Controller for 30 RPM with Load	52
5.28	PID Controller for 10 RPM with Load	53
5.29	PID Controller for 20 RPM with Load	54
5.30	PID Controller for 30 RPM with Load	54

LIST OF SYMBOLS

J	-	Moment of inertia
K_e	-	Electromotive force constant
Ω	-	Electric resistance, ohm
L	-	Electric inductance
V	-	Voltage

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Flowchart of the Implemented PID algorithms for DC Motor speed control application	62
B	Complete Circuit Diagram	63
C	Screenshots of PID Motor Control Panel	64
D	Snapshots of the Hardware with the DC Motor	65

CHAPTER 1

INTRODUCTION

1.1 Background

In many industrial and general applications it is desired that the speed of a motor is restored and maintained during any disturbances to a set value. A scaled down model of this controlling scenario is created by inducing disturbances in this scaled down model and by taking feedback from the output, we will restore the system to a set value by using the Proportional Integral Derivative (PID) control scheme.

The PID controller calculation involves three separate parameters; the Proportional(P), the Integral(I) and Derivative(D) values. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the speed of the dc motor via the microcontroller.

1.2 Objective

The objective of this project is to design a firmware PID sub-routine in microcontroller for computer-based speed control. The PID algorithm written as a computer program will be embedded in a hardware device which is the microcontroller. This firmware is intended to be used to any kind of dc motor that needs to operate under PID control system.

1.3 Scope

The project consists of 3 scopes. The first scope is DC motor steady-state error correction under no load case and loaded case. Steady-state error is defined as the difference between the input and output of a system in the limit as the response has reached the steady state. Steady-state error determines the stability of a system and it is important that the steady-state error is kept at minimum as possible. The second scope is DC motor overshoot control under no load case and loaded case. Overshoot refers to an output exceeding its final, steady-state value. Once the motor start running, its momentum will drive it pass the speed it should be. Overshoot should be reduced by the controller at the expense of a longer rise time. The third scope is the creation of a Control Panel that allows data monitoring for performance analysis. It is important that these project features user friendly interface. By using a control panel, the user can simply insert the PID gains and the performance of the motor can be monitored from the control panel.

CHAPTER 2

LITERATURE REVIEW

2.1 DC Motor Characteristic

A common actuator in control systems is the DC motor. It directly provides rotary motion and, coupled with wheels or drums and cables to provide transitional motion. Some of its characteristics that can be addressed are:

1. Moment of inertia of the rotor (J) :0.01 kg.m²/s²
2. Damping ratio of the mechanical system (b) :0.1 Nms
3. Electromotive force constant (K_e) :0.01 Nm/Amp
4. Electric resistance I :1 Ω
5. Electric inductance (L) :0.5 H
6. Input (V) :Source Voltage
7. Output (theta) :Position of shaft
8. The rotor and shaft are assumed to be rigid

To meet with the design requirements, first the motor can only rotate at 0.1 rad/sec with an input voltage of 1 Volt. Since the most basic requirement of a motor is that it should rotate at the desired speed, the steady-state error of the motor speed should be less than 1%. The other performance requirement is that the motor must accelerate to its steady-state speed as soon as it turns on. In this case, the motor should have a settling time of 2 seconds. Since a speed faster than the reference may damage the equipment, it also need to have an overshoot of less than 5%. Using MATLAB, the original open-loop performances can be plotted as figure below.

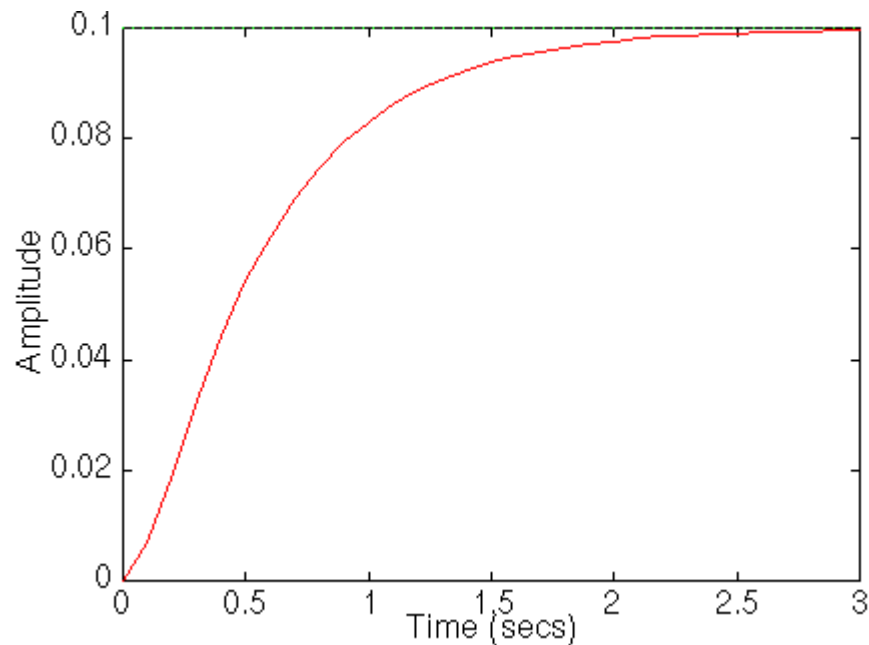


Figure 2.1 Step Response of Open Loop System

If the reference input is simulated by an unit step input, then the motor speed output should have:

1. Settling time less than 2 seconds
2. Overshoot less than 5%
3. Steady-state error less than 1% [1]

2.2 PID Implementation on DC Motor Close Loop Control

The closed-loop controller is a very common means of keeping motor speed at the required set point under varying load conditions. It is also able to keep the speed at the set point value where for example, the set point is ramping up or down at a defined rate.

In the closed loop speed controller, a signal proportional to the motor speed is fed back into the input where it is subtracted from the set point to produce an error signal. This error signal is then used to work out what the magnitude of controller output should be to make the motor run at the required set point speed. For example, if the error speed is positive, the motor is running too fast so that the controller output should be reduced and vice-versa.

If a load is applied, the motor slows down so that a positive error speed is produced. The output increases by a proportional amount to try and restore the speed. However, as the motor speed recovers, the error reduces and so therefore does the drive level. The result is that the motor speed will stabilize at some speed below the set point at which the load is balanced by the error speed times the gain. If the gain is very high so that even the smallest change in motor speed causes a significant change in drive level, the motor speed may oscillate. This basic strategy is known as “proportional control” and on its own has only limited use as it can never force the motor to run exactly at the set point speed.

The next improvement is to introduce a correction to the output which will keep adding or subtracting a small amount to the output until the motor reaches the set point, at which point no further changes are made. In fact a similar effect can be had by keeping a running total of the error speed speeds observed for instance, every 25ms and multiplying this by another gain before adding the result the proportional correction found above. This new term is based on what is effectively the integral of the error speed.

The proportional term is a fast-acting correction which will make a change in the output as quickly as the error arises. The integral takes a finite time to act but has the ability to remove all the steady-state speed error.

A further refinement uses the rate of change of error speed to apply an additional correction to the output drive. This means that a rapid motor deceleration would be counteracted by an increase in drive level for as long as the fall in speed continues. This final component is the “derivative” term and it is a useful means of increasing the short-term stability of the motor speed. A controller incorporating all three strategies is the well-known Proportional-Integral-Derivative, or “PID” controller.

Creating PID algorithm involves lots of concern in terms of the programming. The main issue on implementing PID control system is on how to program the algorithm and correctly functioning as true PID behavior. For the error calculation results, the plant variables might be bigger than Set point value and gives negative Error result. As a solution, the program must have conversion subroutine to ensure the Error result is in positive value. Another aspect to consider is the Integral Windup. Integral term is based on the sum of all previous observed error speeds. However the integral can continuous to integrate indefinitely, thus the microcontroller program must check for overflow on the resulting integral term. [2]

For best performance, the proportional and integral gains need careful tuning. For example, too much integral gain and the control will tend to over-correct for any speed error resulting in oscillation about the set point speed. Integral gains ensure that under steady state conditions that the motor speed almost exactly matches the set point speed. A low gain can make the controller slow to push the speed to the set point but excessive gain can cause hunting around the set point speed. In less extreme cases, it can cause overshoot whereby the speed passes through the set point and then approaches the required speed from the opposite direction. Unfortunately, sufficient gain to quickly achieve the set point speed can cause overshoot and even oscillation but the other terms can be used to damp this out. Proportional gains gives fast response to sudden load changes and can reduce instability caused by high integral gain. This gain is typically many times higher than the integral gain so that relatively small deviations in speed are corrected while the integral gain slowly moves the speed to the set point. Like integral gain, when set too high, proportional gain can cause an oscillation of a few Hertz in motor speed.

There are many ways for an initial setting of the gains. One of it is to set the set point to maximum speed and with the integral and derivative gains at zero, increase the proportional gain so that the speed reaches the maximum possible before a speed oscillation sets in. Reduce the set point to zero. Repeatedly apply a step change in set point to 75% of full speed and increase the integral gain gradually until the speed starts to overshoot.

The speed should rise quickly with the step change and settle at the set point without significant overshoot. The integral gain setting will be particularly influenced by the moment of inertia of the load and some experimentation will be required. The controller is configured as a proportional-integral controller which should quickly correct speed errors without oscillation. [3]

A simulation of how PID controller works can be done through MATLAB. First, the proportional control was put to the test. By using a gain of 100, and by using MATLAB m-file, the following plot is generated.

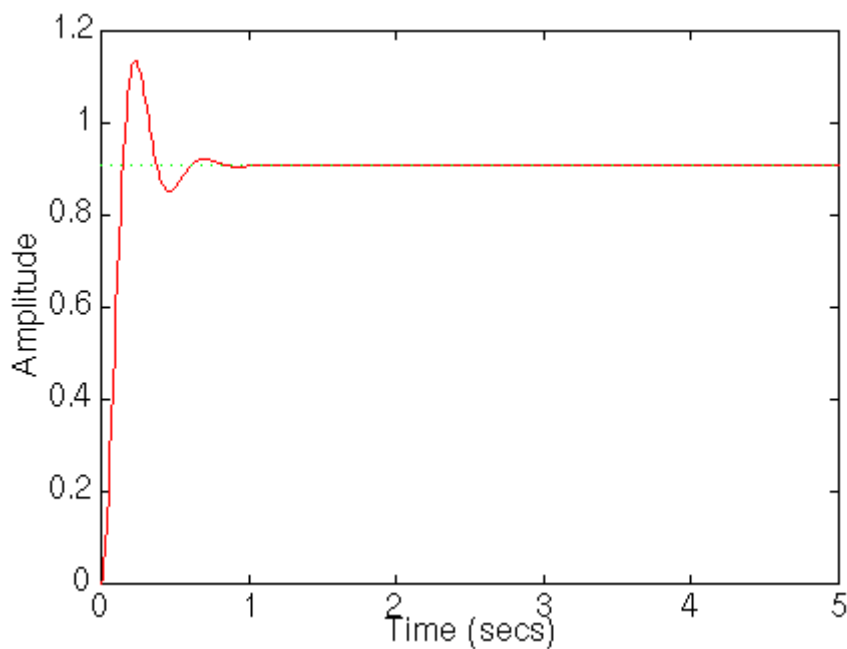


Figure 2.2 Step Response with Proportional Control

From the plot above, the steady-state error and the overshoot are too large. Adding an integral term will eliminate the steady-state error and a derivative term will reduce the overshoot. Inserting a small K_i and K_d to the system and the plot as figure below is obtained.

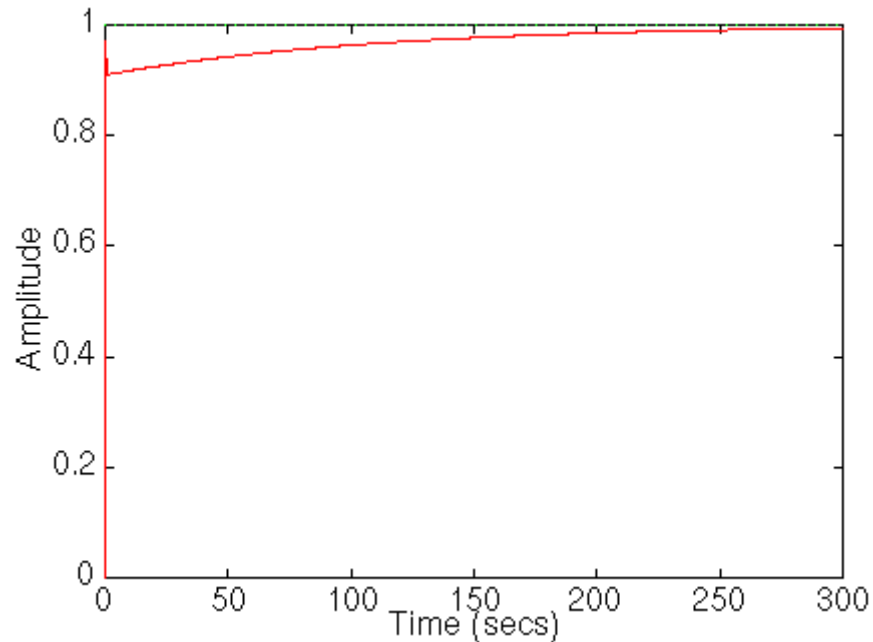


Figure 2.3 PID Control with Small K_i and K_d

From the figure above, it is seen that the settling time is too long. Increasing K_i will reduce the settling time as the figure below.

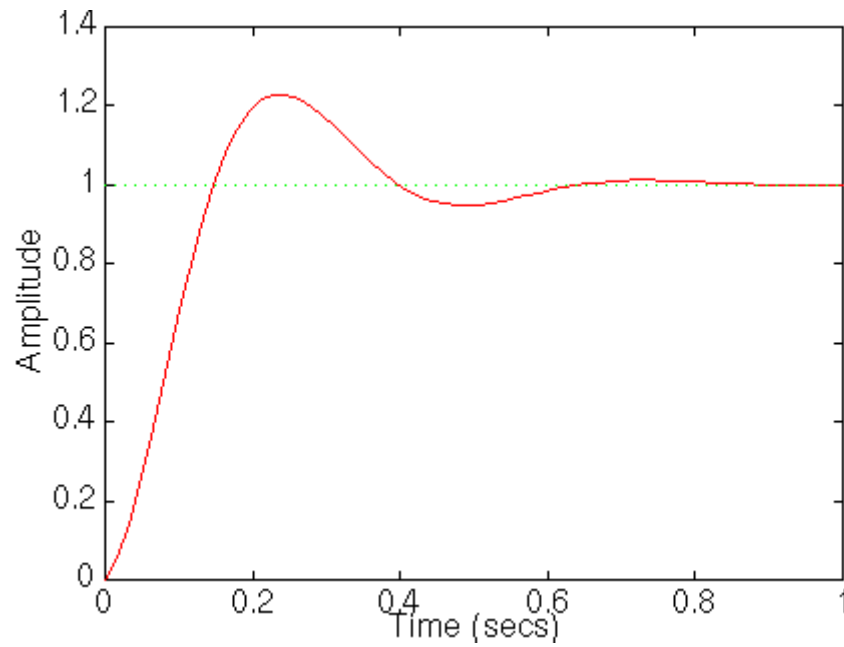


Figure 2.4 PID Control with Large K_i

From the figure above, it is seen that the response is much faster than before, but the large K_i has worsened the transient response and result in big overshoot. Increasing K_d will reduce the overshoot and figure as below is obtained. From the figure above, the design requirements has been achieved. [4]

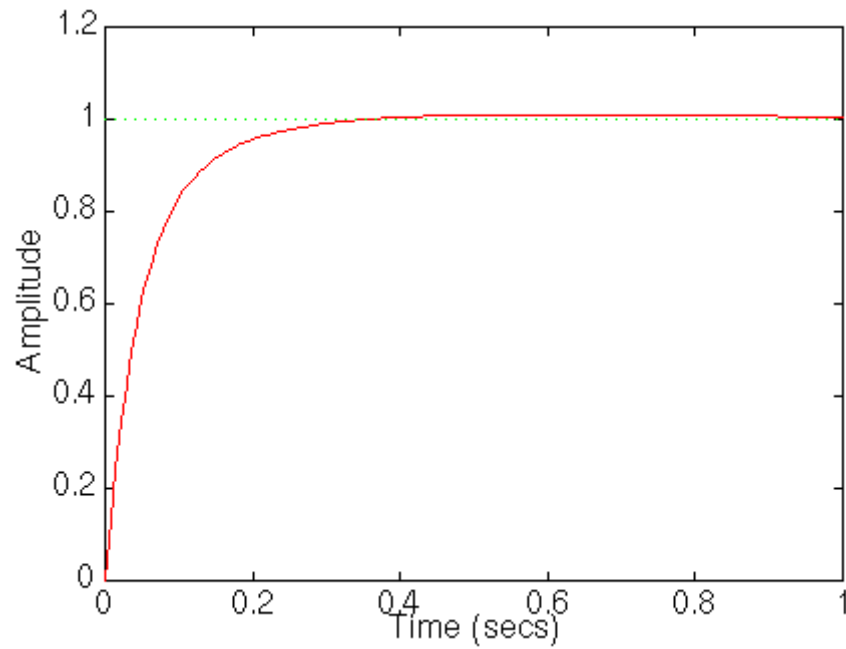


Figure 2.5 PID Control

2.3 Adaptive PID

The term adaptive system implies that the system is capable of accommodating unpredictable environmental changes, whether these changes arise within the system or external to it. The adaptive control scheme consists of two parts. The first part is using initial or updated PID parameters, the controller will be taking in input samples, processing them, and sending them out to the motor. The second part is updating the controller parameters. This process continues until the error signal approach zero. [5]

2.4 PID Tuning

Tuning a PID is the adjustment of its control parameters to the optimum values for the desired control response. The optimum behavior of a process varies depending on the application. There are several methods for tuning a PID. The most effective methods generally involve the development of some form of process model, then choosing P, I and D based on the dynamic model parameters.

Table 2.1: Choosing a Tuning Method

Choosing a Tuning Method		
Method	Advantages	Disadvantages
Manual Tuning	No math required. Online Method	Requires experienced personnel
Ziegler-Nichols	Proven method. Online method	Process upset, some trial-and-error, very aggressive tuning
Software Tools	Consistent tuning. Online or offline method. May include valve and sensor analysis. Allow simulation before downloading	Some cost and training involved
Cohen-Coon	Good process model	Some math. Offline method. Only good for first-order processes.

2.4.1 Manual Tuning

If the system must remain online, one tuning method is to first set the I and D values to zero. Increase P until the output of the loop oscillates, then the P should be left set to be approximately half of that value. Then increase D until any offset is correct insufficient time for the process. However, too much D will cause instability. Finally, increase I, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much I will cause excessive response and overshoot. A fast PID tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot, in which case an over-damped closed-loop system is required, which will require a P setting significantly less than half of that of the P setting causing oscillation.[6]

Table 2.2: Effect of Increasing Parameters

Effects of increasing parameters				
Parameter	Rise Time	Overshoot	Settling Time	Steady-state error
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Decrease	Decrease	Decrease	None

2.5 Implementing a PID Controller Using a PIC18 MCU

As the controller for the system, the microprocessor is chosen due to its simplicity in designing and also interfacing with other input or output devices. Below is the pin diagram of the microprocessor.

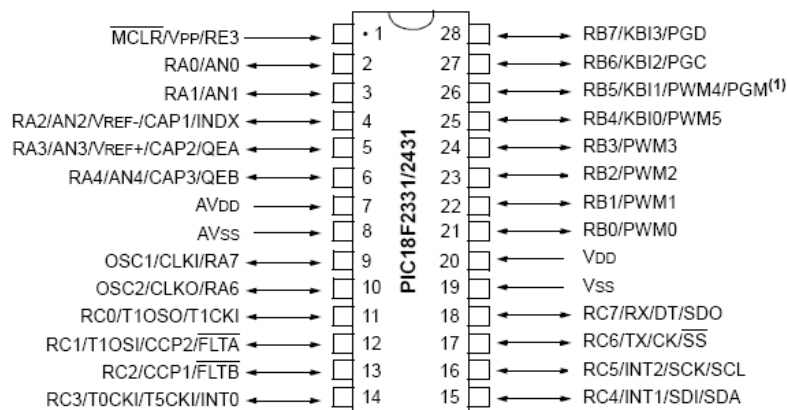


Figure 2.6 18F2331 pin Diagram

The microprocessor consists of 28 pin. At an economical price, with the addition of high endurance enhanced Flash program memory and a high speed 10-bit A/D converter. On top of these features, the PIC18F2331/2431/4331/4431 family introduces design enhancements that make these microcontrollers a logical choice for many high performance, power control and motor control applications. These special peripherals include 14-bit resolution Power Control PWM Module (PCPWM) with programmable dead time insertion Motion Feedback Module (MFM), including a 3-channel Input Capture (IC) Module and Quadrature Encoder Interface (QEI) High-speed 10-bit A/D Converter (HSADC) The PCPWM can generate up to eight complementary PWM outputs with dead-band time insertion. The MFM Quadrature Encoder Interface provides precise rotor position feedback and velocity measurement.

The PID routine is configured in a manner that makes it modular. It is intended to be plugged into an existing piece of firmware, where the PID routine is passed the 8-bit or 16-bit error value. Therefore, the actual error value is calculated outside of the PID routine. If necessary, the code could be easily modified to do this calculation within the PID routine. The PID can be configured to receive the error in one of two ways, either as a percentage with a range of 0 to 100% (8-bit), or a range of 0 to 4000 (16-bit). PID source code with the PID's variable declarations. The gains for proportional, integral and derivative all have a range of 0 to 15. For resolution purposes, the gains are scaled by a factor of 16 with an 8-bit maximum of 255. A general flow showing how the PID routine would be implemented in the main application code is presented in Figure 2. There were two methods considered for handling the signed numbers. The first method was to use signed math routines to handle all of the PID calculations. The second was to use unsigned math routines and maintain a sign bit in a status register. [7]

CHAPTER 3

IMPLEMENTATION OF PID CONTROLLER ALGORITHMS IN MICROCONTROLLER UNIT

3.1 Introduction

In this system, the PID controller is designed using PIC microcontroller 18F2331. This microcontroller provide motion feedback module that is useful in designing a close loop control system. Furthermore the microcontroller also provide up to 4 PWM channels that allow the user to control more motor. To provide feedback to the microcontroller, a quadrature encoder is used. The quadrature encoder will provide the actual speed references to allow the microcontroller to calculate the error. The full hardware diagram is as below.

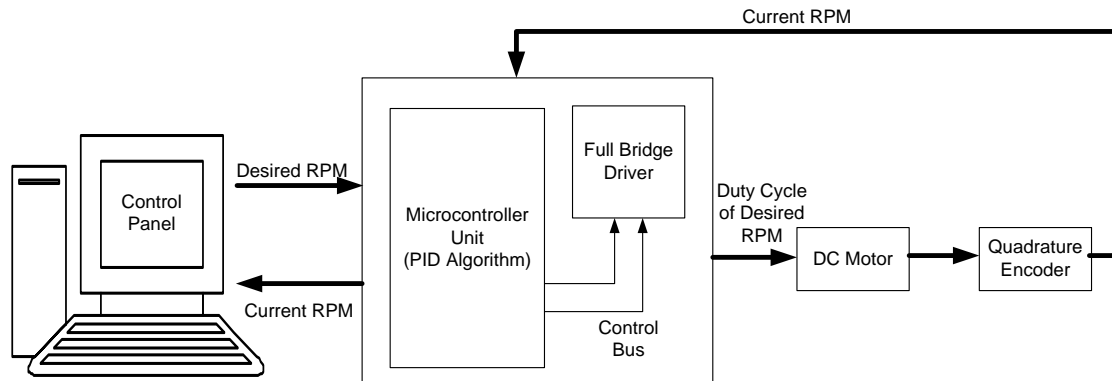


Figure 3.1 Hardware Design

From the figure above, it should not be confused that the computer does not work as a controller, instead it just a monitoring device that allow the user to monitor the performance. Furthermore, the computer is also used to upload the PID algorithm into the 18F2331 firmware. The computer is connected to the microcontroller using RS-232 serial data communication. The motor driver works as an actuator in providing the desired duty cycle to the motor.

3.2 Encoder configuration

The feedback module consists of a quadrature encoder and a flexible coupling. The quadrature encoder will enable the system to acquire the feedback and later performing the required operations to effectively use the information coming from the encoder. The two quadrature encoder output signals channel A and channel B. The position counter can be used either for position or speed measurement. To measure motor position, we must know the relationship between the displacement and the number of phase pulses we get from the encoder. This relation can be known in advance,

or can be measured during initialization by accumulating the total count for the maximum allowed displacement by using the formula below.

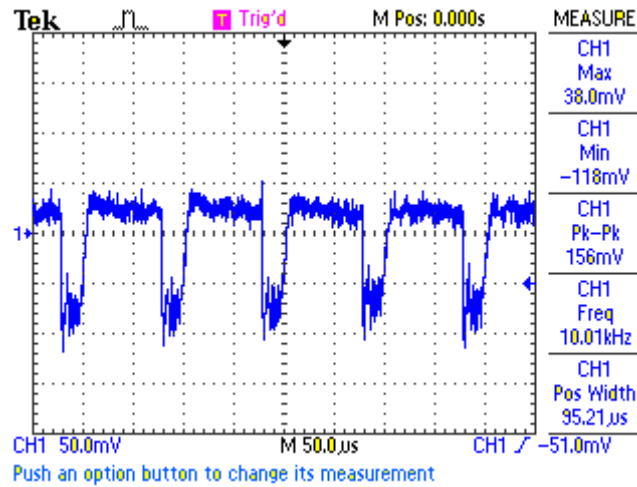


Figure 3.2 Sample of Output From Encoder

To calculate the angular velocity under a fixed time interval, the value of encoder pulses and pulses per revolution of the encoder need to be known. Mathematically, it can be derived from the formula above. By running the motor at full speed and record the number of pulses under 1 second sampling time, the value of pulses per revolution can be obtained. Below is the sample of output from the encoder.

$$\text{Velocity} = \frac{\frac{\text{Encoder Pulses}}{\text{Pulses Per Revolution}} \times \frac{60\text{sec}}{1\text{min}}}{\text{Fixed Time Interval (sec)}} \text{ (RPM)}$$

By using flexible coupling, the single joint allows for minor misalignments such as installation errors and changes in shaft alignment due to operating conditions.

3.3 DC Motor

In this system, the dc motor will be run under no load. The dc motor used is a brushless geared dc motor. The specification is as below.

Table 3.1: SPGH-150 DC Motor Specification

Rated Voltage (V)	No Load		Load				Maximum Efficiency		Output Power(W)	Number of gear trains	Gearbox length (mm)
			Current (Ma)	Speed (RPM)	Current (Ma)	Speed (RPM)					
	Kgf.cm	N-m					Kgf.cm	N-m			
12	≤220	30	≤900	23	10	0.98	30	2.94	3.4	4	26



Figure 3.3 SPGH-150 DC Motor

3.3.1 Pulse Width Modulation

Pulse-width modulation (PWM) or duty-cycle variation methods are commonly used in speed control of DC motors. The duty cycle is defined as the percentage of digital 'high' to digital 'low' plus digital 'high' pulse-width during a PWM period.

The average DC voltage value for 0% duty cycle is zero; with 25% duty cycle the average value is 3V (25% of 12V). With 50% duty cycle the average value is 6V, and if the duty cycle is 75%, the average voltage is 9V and so on. The maximum duty cycle can be 100%, which is equivalent to a DC waveform. Thus by varying the pulse-width, we can vary the average voltage across a DC motor and hence its speed.

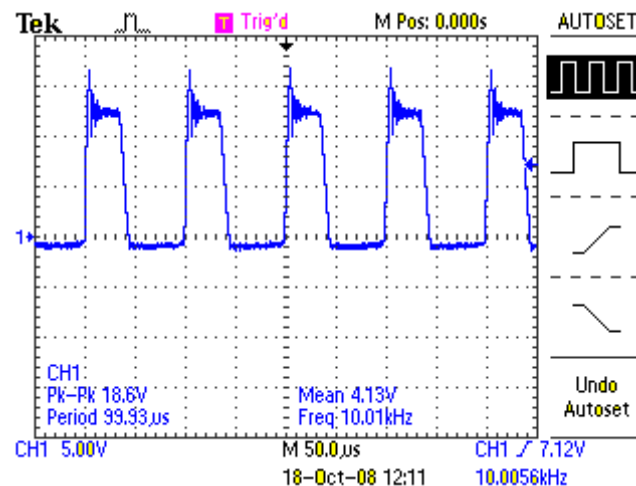


Figure 3.4 Sample of a PWM Waveform

3.4 PID Algorithms

PID algorithms consist of three parameters which are Proportional, Integral and Derivative terms. All this three terms later on added to create an output which will be inserted into the plant or in this case the motor. From previous chapter, it is known that the motor run on a generated PWM from the microcontroller and this PWM waveform is controlled through its duty cycle. The duty cycle plays an important role in the whole system. The PID output itself will be inserted together with the duty cycle to create an adjustment so that the motor can be brought back to its desired speed. To do this, the system need to know the rated speed, or the maximum speed the motor can handle. This rated speed value is used with the desired speed value so that it can be converted into a duty cycle.

$$\text{Duty Cycle} = (\text{Desired Speed} / \text{Rated Speed}) \times 255$$

This formula is converted into a PICBasic language as below:

```
Duty = Setpoint * 255  
Duty = Duty/Rated Speed  
HPWM 2,Duty,10000
```

Once the motor already run on the desired speed, it will be left to the PID algorithm to correct its speed by adjusting the output of PID into the duty cycle. The flowchart below explains how the PID algorithm works.

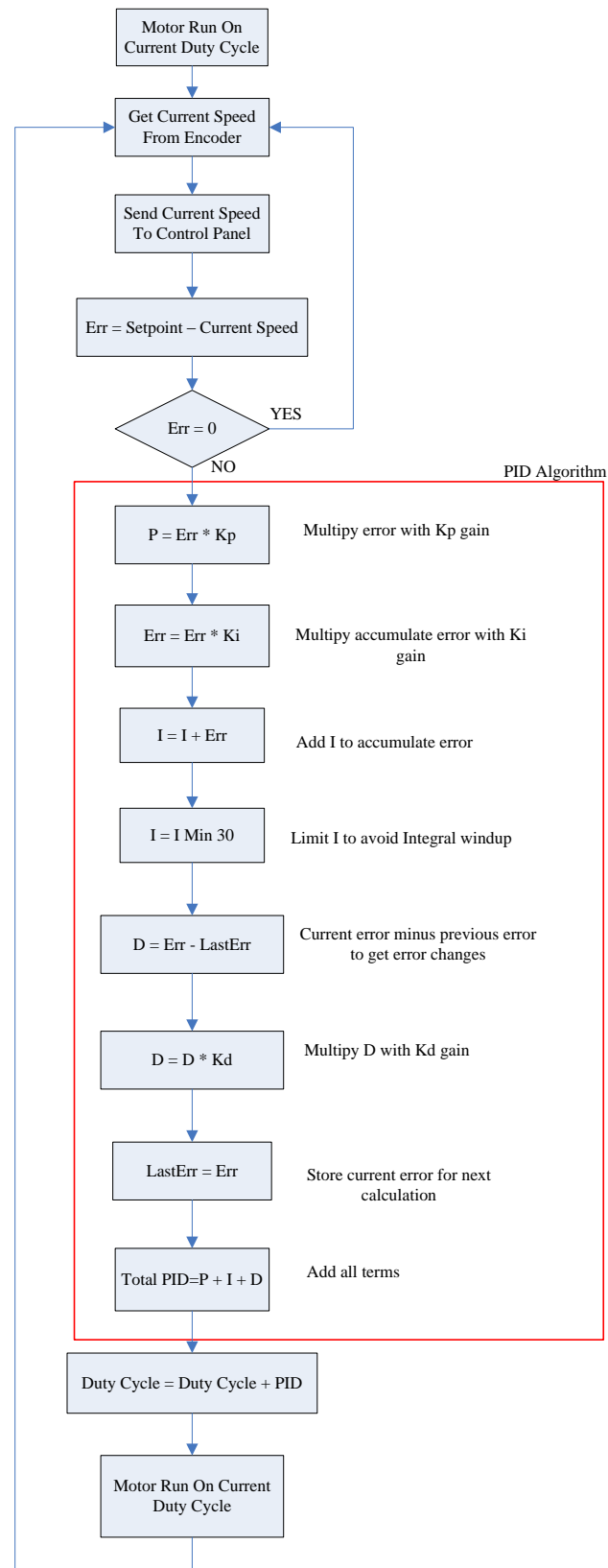


Figure 3.5 Flowchart of PID Algorithm Implemented In the MCU

3.4.1 Error Calculation

The error calculation is basically the difference between the desired speed or setpoint and the actual speed of the motor. The actual speed from encoder will be used as Current speed. In the programming, a subroutine is used to calculate the error.

```
Error = Setpoint - actual RPM
```

However, under certain condition, the actual speed might be bigger than the set point. Thus, the controller must be made to ensure that it know the sign of the error. This can be done by checking the highest bit of the error variable.

```
Error = ABS error  
IF Error.15 = 1 THEN error = -error
```

Furthermore, it is important to use the absolute value of error in all the calculations later on.

3.4.2 Proportional Terms

Proportional parameter is simply the multiplication between the proportional gain, K_p with the Error. The proportional gain is inserted into the program since it is not an auto tuning PID routine.

```
P = Error * Kp  
IF Error.15 = 1 THEN P=-P
```

3.4.3 Integral Terms

Unlike proportional control, which looks at the present error, integral control looks at past errors. This is the accumulative error (sum of all past errors) which is used to calculate the integral term, but at fixed time intervals. By using a program, the program simply records the value of E at fixed time interval of T (sampling time). Since Integral terms looks at past errors, the new integral term is obtained by adding the old integral term with accumulated errors which has been multiplied by the integral gain. However, to prevent integral windup, a limit must be used for calculating the accumulated errors to avoid the accumulated errors to keep on adding.

```

IF Err.15 =1 THEN Err_2 = -Err_2
Ei = Ei + Err_2
Sign = Ei.15
Ei_2 = (ABS Ei) * Ki
IF Sign = 1 THEN Ei_2 = -Ei_2
I = I + Ei_2
Sign = I.15
I = ABS I
I = I MIN 100
IF Sign = 1 THEN I = -I

```

In the code snippet above, the current error, Ei_2 will be added to accumulated error, Ei. The accumulated error is then multiplied by Ki. Later on the result of the multiplication is added with the I terms.

3.4.5 Derivative Terms

The derivative term works on the present errors to forecast a future response of the system. The derivative term makes an adjustment based on the rate at which the Plant output is changing from its Setpoint. A notable characteristic in this type of control is when the error is constant, or at the maximum limit, the effect is minimal. To get the derivative term, the previous error is subtracted from the current error and multiplied by the derivative gain, Kd. Then the program must save the current error so that at next time, it will be the old error.

```

D = (Error-LastError)
D = ABS D * Kd
IF D.15 = 1 THEN D = -D
LastError = Error
IF Error.15 =1 THEN LastError = -LastError

```

3.4.6 PID Output

The PID output is calculated after the proportional, integral and derivative terms have been determined. It is done by adding the current motor duty cycle with the PID. This result will later be inserted in the duty cycle variable of the motor.

```

Duty = actual RPM * 255/Maximum Motor RPM
Duty = Duty + (P + I + D)
Duty = Duty MIN 255
HPWM 2,Duty,10000

```

3.5 Adaptive PID

By understanding how PID algorithm works, it is later found out that each of the three gains can be created by increasing them to their ratio. This understanding however is obtained after the Manual Tuning of PID algorithm had been working perfectly. The flowchart below explains how the gains are increased up to a certain value. The term Adaptive PID means that the gains will be increased to meet the performance requirement until steady-state error has been fully corrected from the system.

```
IF RPM = Setpoint THEN RETURN  
Accept_Value = RPM  
Automin_value = Setpoint - 1  
Automax_value = Setpoint + 1  
IF (Accept_Value <= Automin_Value OR Accept_value >=  
Automax_value) THEN  
Kp = Kp + 1  
Kp = Kp MIN 10  
Ki = Ki + 1  
Ki = Ki MIN 2  
Kd = Kd + 1  
Kd = Kd MIN 2
```

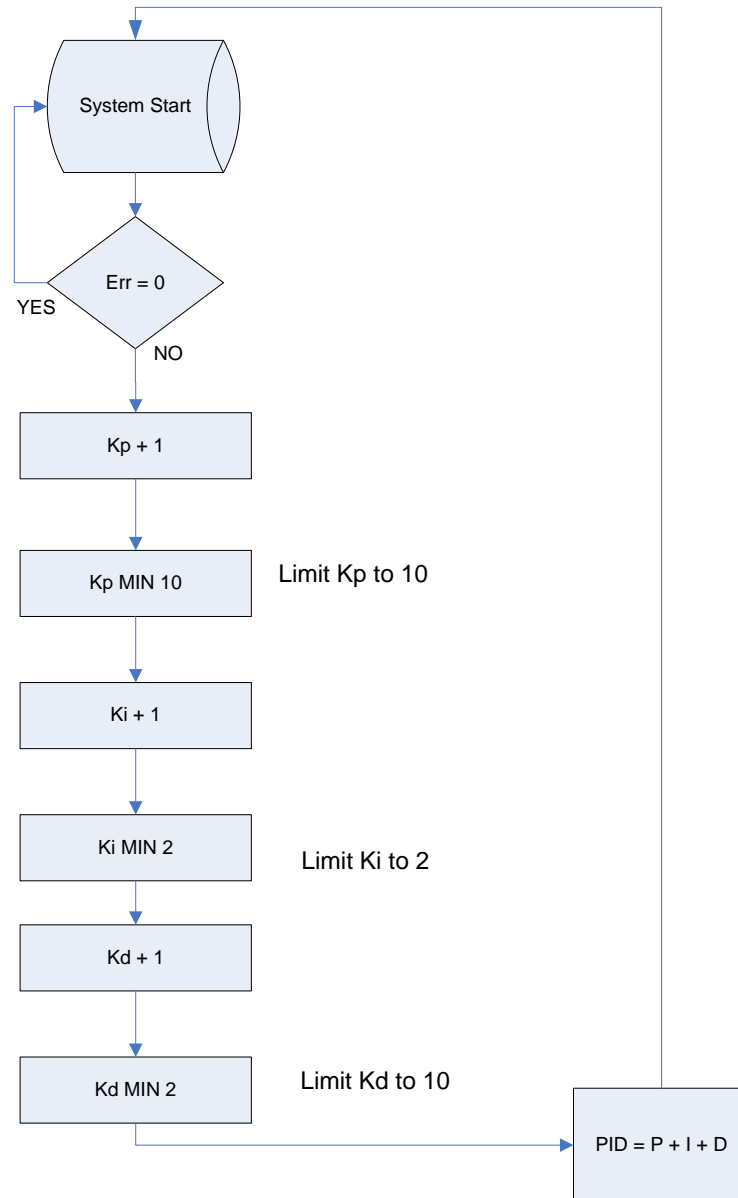


Figure 3.6 Flowchart of the Adaptive PID Algorithm Implemented In the MCU

CHAPTER 4

GRAPHICAL USER INTERFACE DESIGN

4.1 Introduction

Real-time performance monitoring to identify performances has become an integral part of process control solutions. Nowadays automatic process control solutions that incorporate real-time monitoring and performance analysis are fulfilling the market need. By applying real time performance monitoring, the project will gain several advantages. It will allow the build up to data collection that can create statistics that can be applied to improve the performance.

4.2 PID Motor Control Panel

The control panel is built on two purposes;

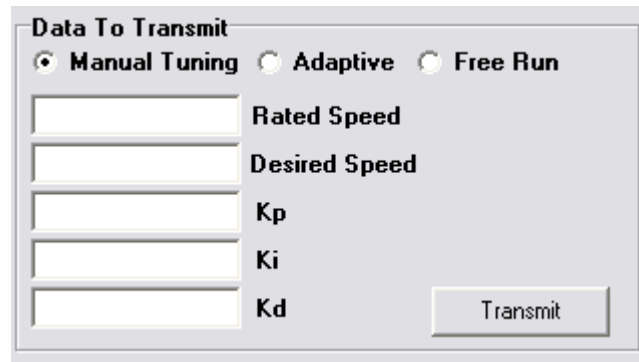
- i) Data transmission to the controller
- ii) Performance monitoring

4.2.1 Data Transmission to the Controller

For the controller to work perfectly, it needs to know the environment it is working on. Thus, the control panel needs to send all the information related for the controller. This information are:

- i) Rated Speed
- ii) Desired Speed
- iii) K_p gains
- iv) K_i gains
- v) K_d gains

All this data can be inserted through a dedicated text box for each of it as shown from figure below.



The screenshot shows a software window titled "Data To Transmit". Inside the window, there are three radio buttons: "Manual Tuning" (which is selected), "Adaptive", and "Free Run". Below the radio buttons, there are five input fields arranged vertically. To the right of each input field is a label: "Rated Speed", "Desired Speed", "Kp", "Ki", and "Kd". At the bottom right of the window is a button labeled "Transmit".

Figure 4.1 Data Transmission Panel

The screenshot of the PID Motor Control Panel is shown in Appendix C.

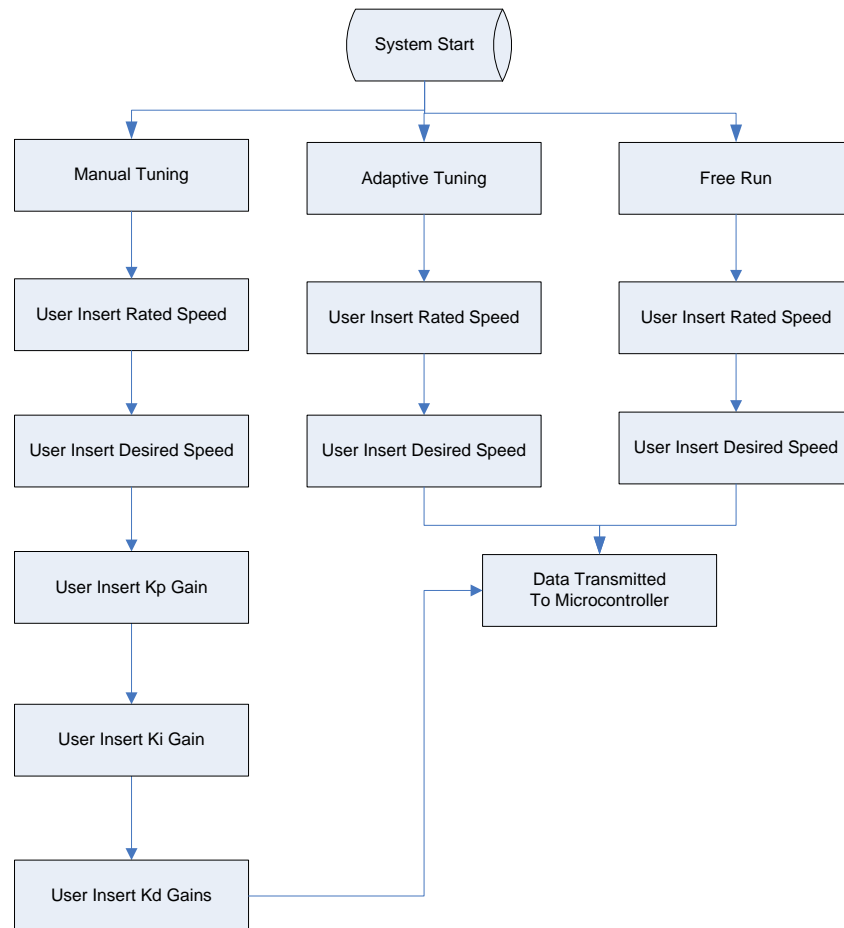


Figure 4.2 Flowchart of the Data Transmission Panel

The microcontroller unit and the PID Motor Control Panel communicate through RS232 serial data communication. The control panel will send the required data to the microcontroller unit continuously until all the required data needed is gathered. As shown, the control panel allows the user to choose three types of mode. The first is the Auto Tune mode that will cause the motor to run under adaptive PID control system. The later is the Manual Tune mode which requires the user to insert the value of K_p , K_i , and K_d . The last mode is the Free Run mode that will set the motor to run on a desired set point without any control system affecting it.

4.2.2 Performance Monitoring

The analysis of percentage of overshoot and steady-state error is important in this system as it is the objectives of this project to control the DC motor overshoot and steady-state error. Thus, data plotting is important as it allow visual monitoring of the performance of the system. The control panel plots any data coming from the microcontroller through a graph as shown below.

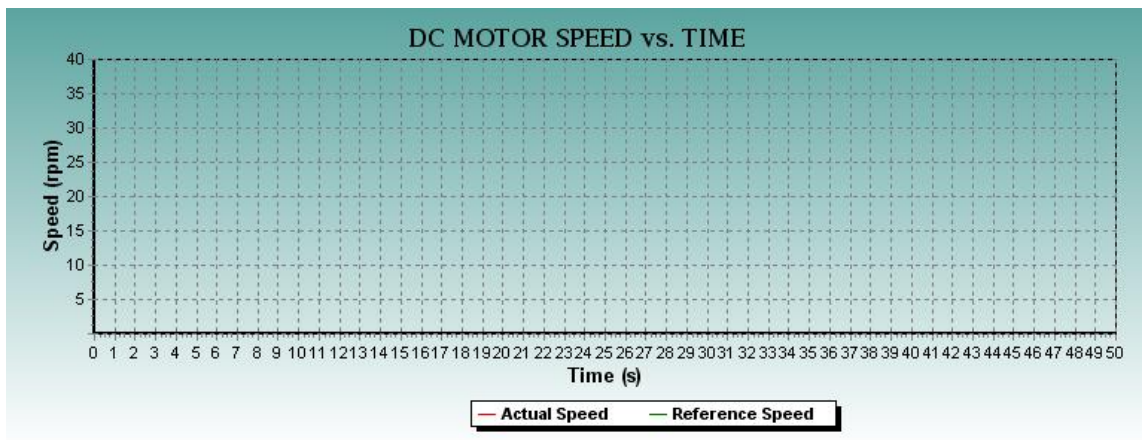


Figure 4.3 Data Plotting Platform

As stated before, the reason of the motor performance analysis is to come up with the analysis of:

- i) Percentage of overshoot
- ii) Steady-state error

CHAPTER 5

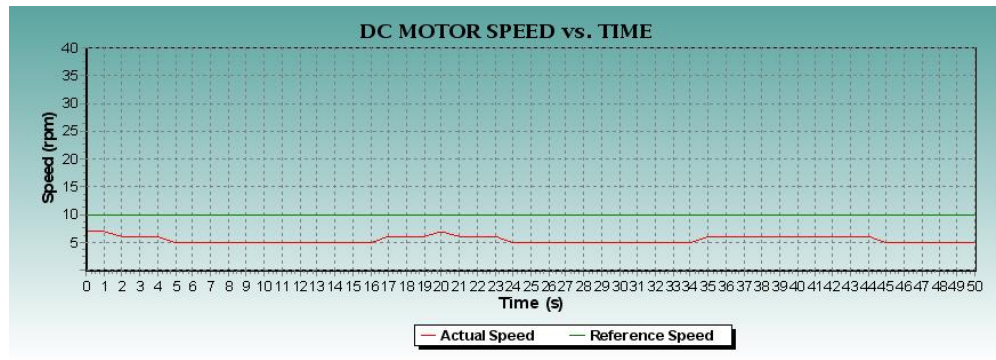
RESULTS, PERFORMANCES AND ANALYSIS

5.1 Introduction

Performance analysis is crucial in determining the response of a close-loop control system. For a PID control system, it is important in determining which part of it that affects the response of the system. Thus, performance analysis is done graphically by using the DC Motor Speed vs Time graph that was plotted by PID Motor Control Panel.

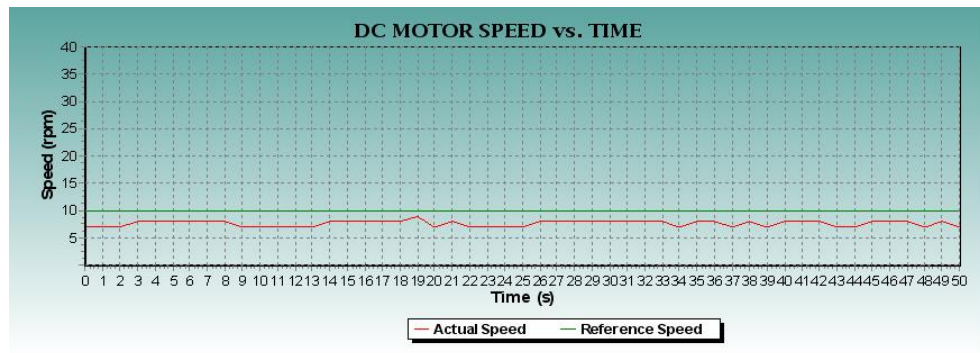
5.2 PID Tuning

To find the K_p gain, the derivative and integral part is turned off. Then K_p is increased to max or until oscillation occurs. If system oscillates, K_p is divided by 2.



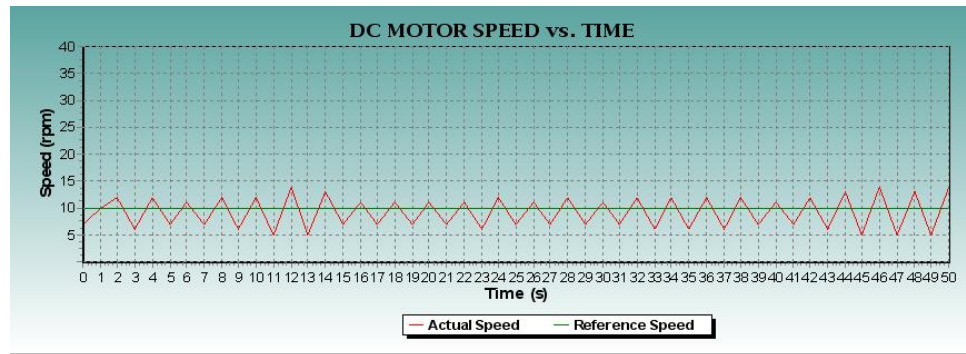
Desired Speed(RPM) = 10 $K_p = 05$ $K_i = 00$ $K_d = 00$

Figure 5.1 Gain Tuning For $K_p = 5$, $K_i = 0$ and $K_d = 0$



Desired Speed(RPM) = 10 $K_p = 10$ $K_i = 00$ $K_d = 00$

Figure 5.2 Gain Tuning For $K_p = 10$, $K_i = 0$ and $K_d = 0$



Desired Speed(RPM) = 10

Kp = 20

Ki = 00

Kd = 00

Figure 5.3 Gain Tuning For Kp = 20, Ki = 0 and Kd = 0

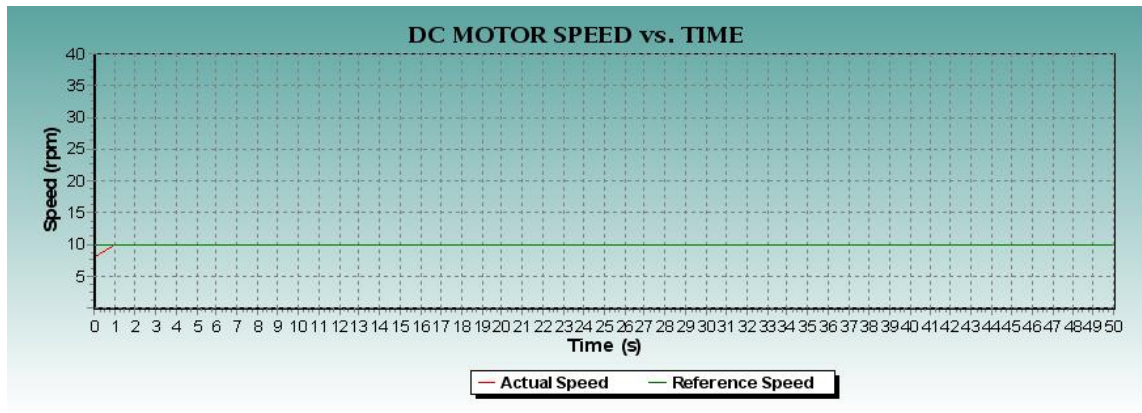
From the graph plotted, it is discovered that oscillation occurred when Kp is set to 20. Table below shows the results.

Table 5.1: Results for Finding Value of Kp

Kp	5	10	20
Oscillation	No	No	Yes

$$Kp = \frac{Kp'}{2} = \frac{20}{2} = 10$$

Then, Kd is increased and its behavior as desired speed changed by about 5% is observed. Value Kd is choose from which it gives a fast damped response.



Desired Speed(RPM) = 10

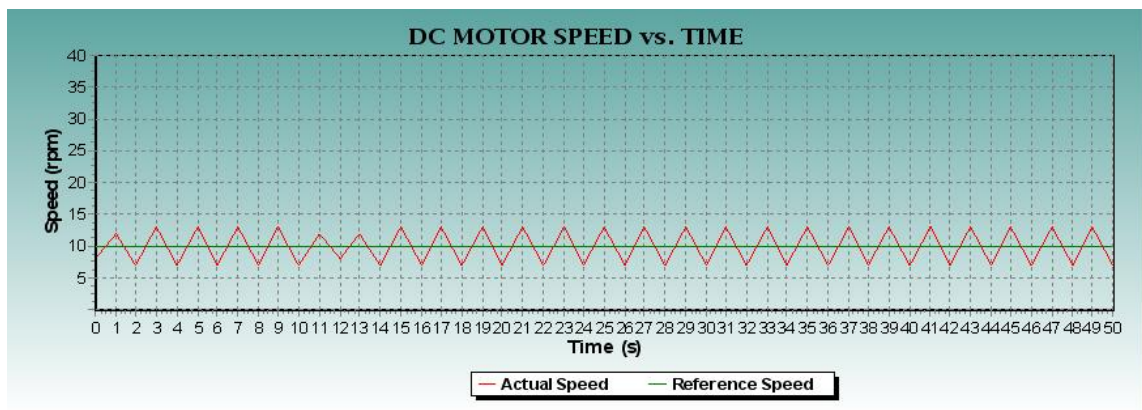
$K_p = 10$

$K_i = 00$

$K_d = 02$

Figure 5.4 Gain Tuning For $K_p = 10$, $K_i = 0$ and $K_d = 1$

To find the value of K_i gains, its value is increased until oscillation occurs. This value will then be divided by 2 or 3 to obtain the real K_i value.



Desired Speed(RPM) = 10

$K_p = 10$

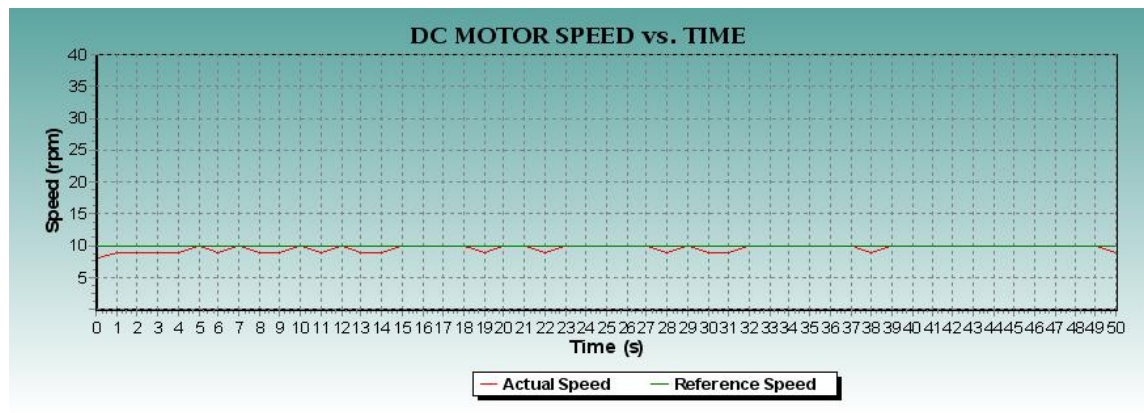
$K_i = 04$

$K_d = 02$

Figure 5.5 Gain Tuning For $K_p = 10$, $K_i = 04$ and $K_d = 2$

5.3 Performance without PID Under No Load

Before a control system is integrated into a system, the user must first know what type of control loop that should be implemented and which part of the system that need to be repaired. Thus it is important the performance of the system without close-loop control system is obtained first.



Desired Speed(RPM) = 10

K_p =

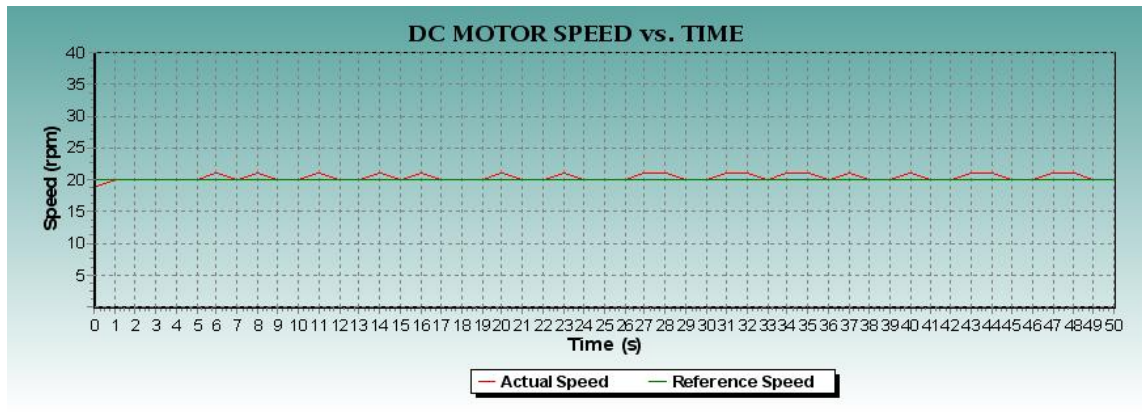
K_i =

K_d =

%Overshoot: 0%

Steady-state error: System does not even reached steady-state

Figure 5.6 Free Run on 10 RPM



Desired Speed(RPM) = 20

$K_p =$

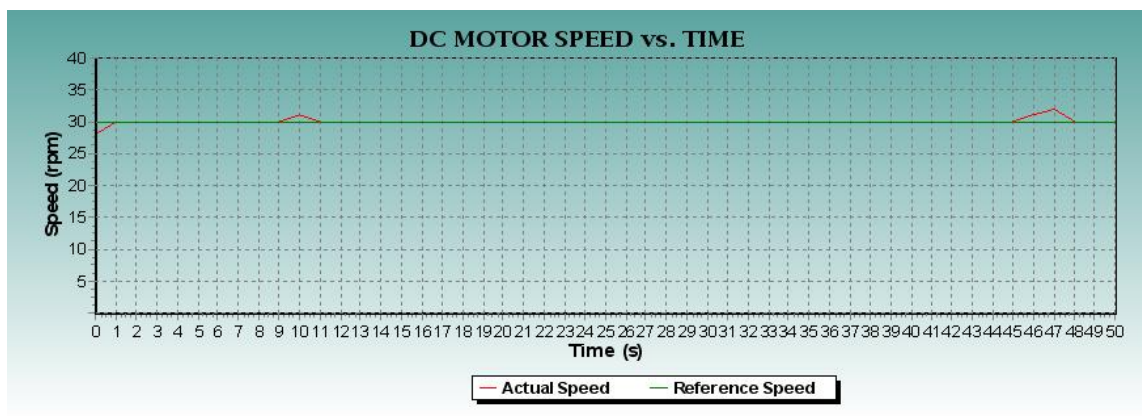
$K_i =$

$K_d =$

%Overshoot: 0%

Steady-state error: Still occur

Figure 5.7 Free Run on 20 RPM



Desired Speed(RPM) = 30

$K_p =$

$K_i =$

$K_d =$

%Overshoot: 0%

Steady-state error: Still occur

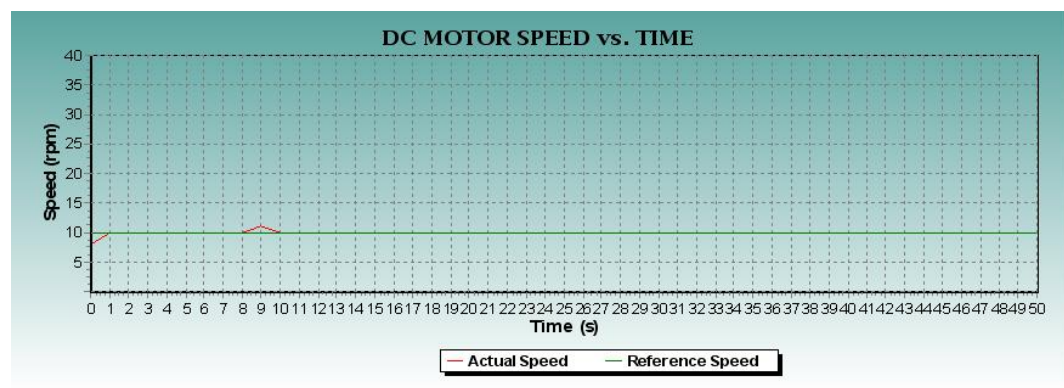
Figure 5.8 Free Run on 30 RPM

5.4 Performance With PID Under No Load

Under no load, the system is tested through three controllers which are PI, PD and also PID.

5.4.1 PI Controller

PI Controller is a special case in which the Derivative term is not used. The integral is used to react to any occurrence of error in the system. In other words, if the occurrence of error is getting rapid, the integral output will grow.



Desired Speed(RPM) = 10

$K_p = 10$

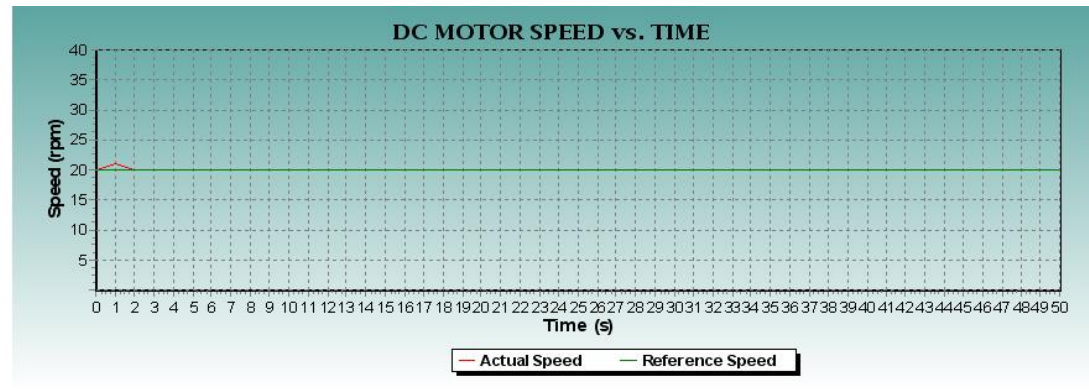
$K_i = 0.2$

$K_d = 0.0$

%Overshoot: 0%

Steady-state error: Still occur

Figure 5.9 PI Controller for 10 RPM



Desired Speed(RPM) = 20

$K_p = 10$

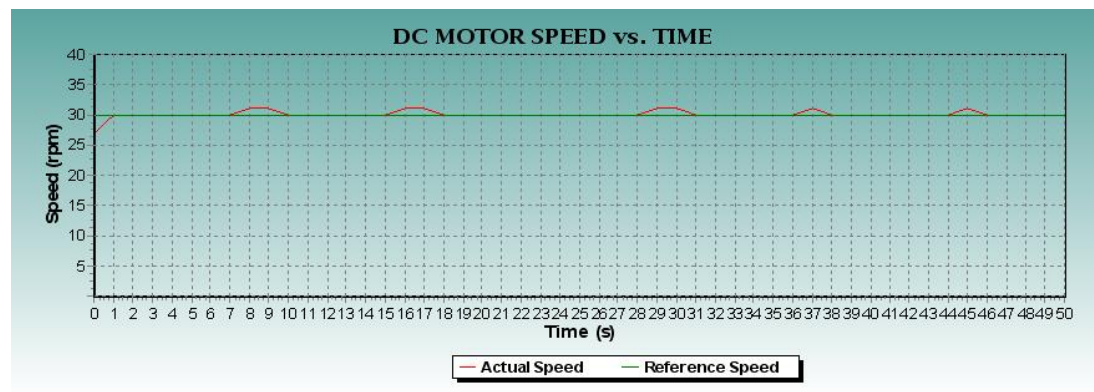
$K_i = 0.2$

$K_d = 0.0$

%Overshoot: 5%

Steady-state error: Eliminated

Figure 5.10 PI Controller for 20RPM



Desired Speed(RPM) = 30

$K_p = 10$

$K_i = 0.2$

$K_d = 0.0$

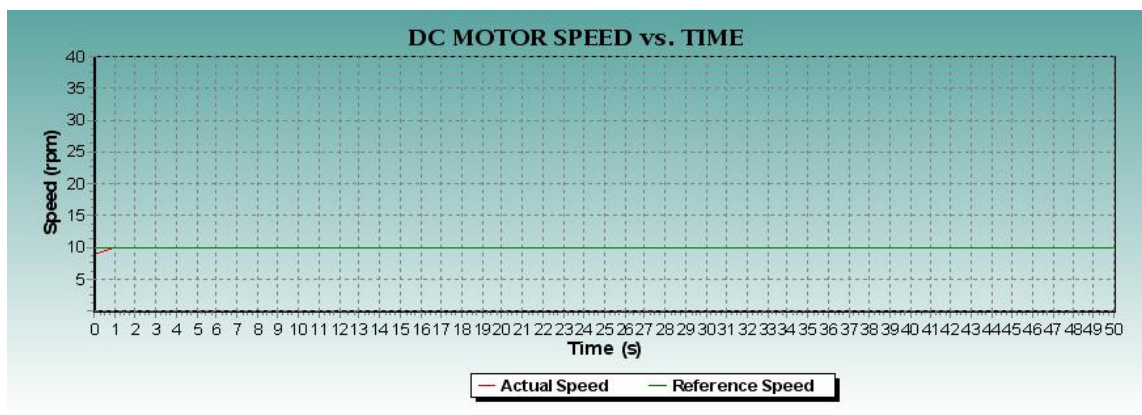
%Overshoot: 0%

Steady-state error: Still occur

Figure 5.11 PI Controller for 30 RPM

5.4.2 PD Controller

PD controller is useful in reacting to sudden changes throughout the system. From the figure below, however when the derivative term is working alone without the integral term, it does not have the ability to overcome the occurrence of the same error and thus we could see that large amount of steady-state occurrence.



Desired Speed(RPM) = 10

$K_p = 10$

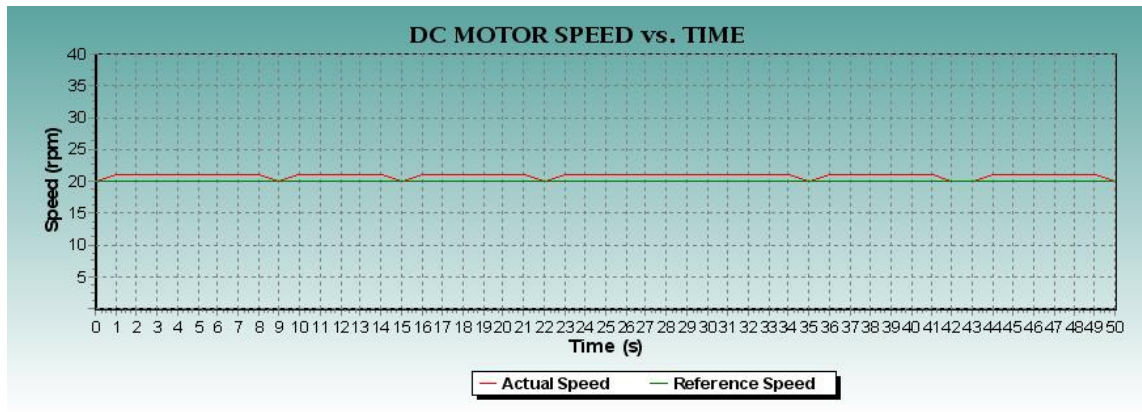
$K_i = 00$

$K_d = 02$

%Overshoot: 0%

Steady-state error: Eliminated

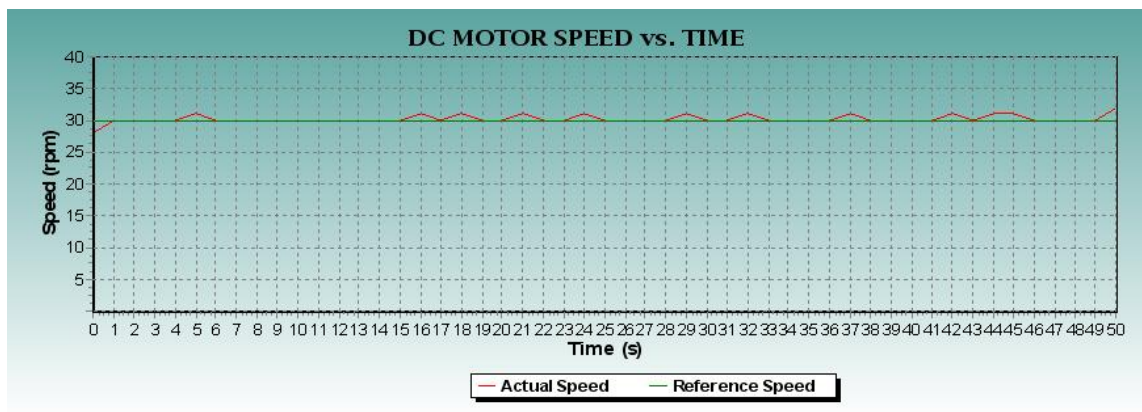
Figure 5.12 PD Controller for 10 RPM



Desired Speed(RPM) = 20 $K_p = 10$ $K_i = 00$ $K_d = 02$
 %Overshoot: 5%

Steady-state error: System does not even reach steady-state.

Figure 5.13 PD Controller for 20 RPM



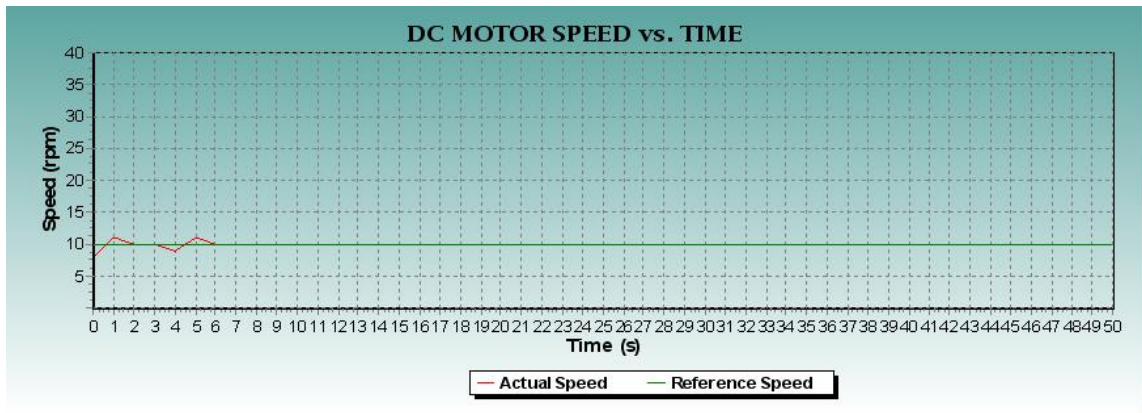
Desired Speed(RPM) = 30 $K_p = 10$ $K_i = 00$ $K_d = 02$
 %Overshoot: 0%

Steady-state error: Still occur

Figure 5.14 PD Controller for 30 RPM

5.4.3 PID Controller

When all the three terms are put together, it is hoped that the best result can be achieved. From the figure below it is seen that the output of all the three terms cause overshoot on all the three tested speed. However we can clearly see that the steady-state error is eliminated as the system goes. This is due to the effect of integral and derivative that counters any occurrence of error and also changes in error.



Desired Speed(RPM) = 10

$K_p = 10$

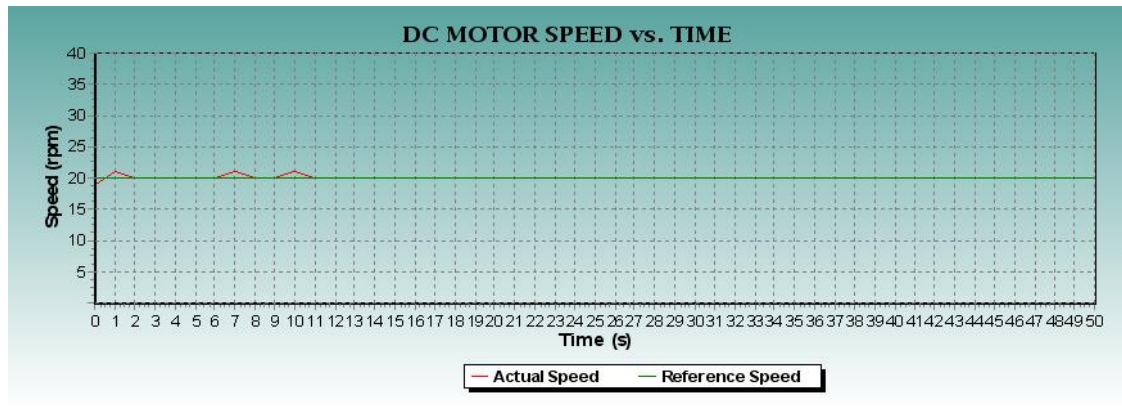
$K_i = 0.2$

$K_d = 0.2$

%Overshoot: 5%

Steady-state error: Still occur but eliminated through time

Figure 5.15 PID Controller for 10 RPM



Desired Speed(RPM) = 20

$K_p = 10$

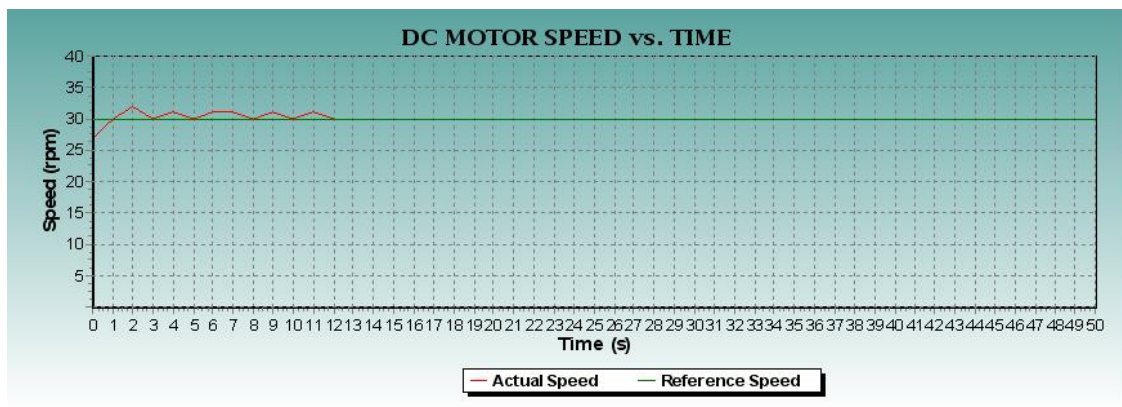
$K_i = 02$

$K_d = 02$

%Overshoot: 5%

Steady-state error: Still occur but eliminated through the time

Figure 5.16 PID Controller for 20 RPM



Desired Speed(RPM) = 30

$K_p = 10$

$K_i = 02$

$K_d = 02$

%Overshoot: 6.67%

Steady-state error: Still occur but eliminated through time

Figure 5.17 PID Controller for 30 RPM

Table 5.2: Results of the Three Controllers Working Under No Load

Controller	PI			PD			PID		
Speed (RPM)	10	20	30	10	20	30	10	20	30
% Overshoot	0 %	5 %	5 %	5 %	5 %	0 %	5 %	5 %	6.67%
Steady-State Error	Yes	No	Yes	No	Yes	Yes	No	No	No

From the table above, the PID algorithm that was used was able eliminate steady-state error but only after a few seconds once the system has reached steady-state. One conclusion that can be made of this is due to the buildup of integral terms that has been able to detect the trend or occurrence of error as the system works. As a result, after a few seconds, steady-state error is totally eliminated.

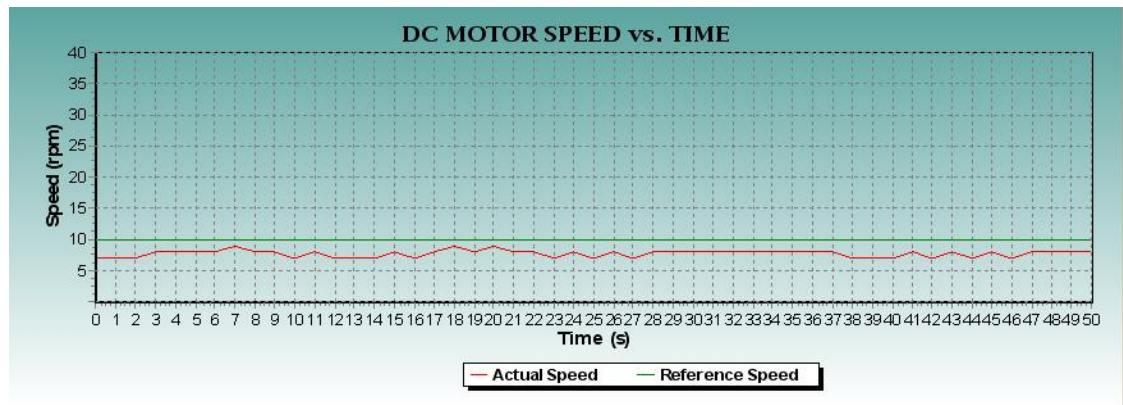
5.5 Performance Without PID Controller Under Load

For more analysis on the effect of PID on the system, it is tested under load. The load used is another DC motor which has a lower torque than the main motor. The reason is so that the main motor will be capable to drag the load.



Figure 5.18 Figure Displaying How The Motor Is Connected To The Load

As tried on no load, the system is then tested to run freely under the influences of load without the PID controller.



Desired Speed(RPM) = 10

$K_p = 10$

$K_i = 02$

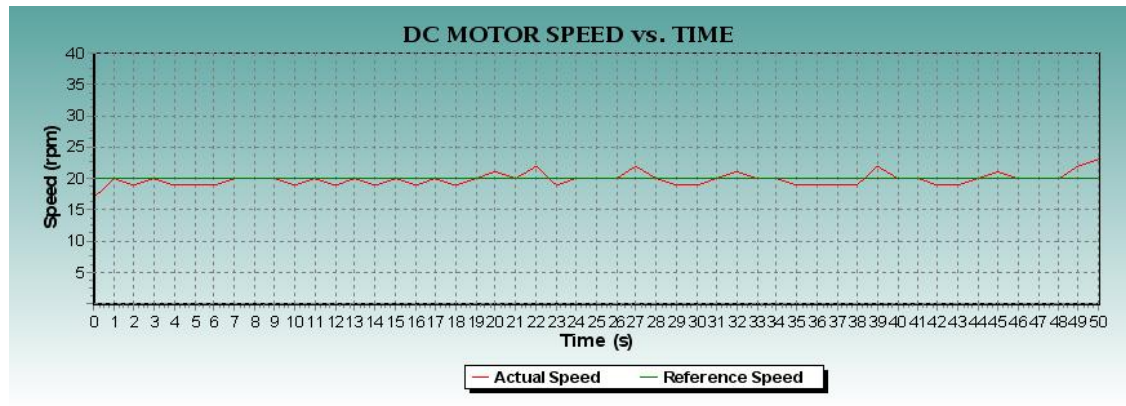
$K_d = 02$

%Overshoot: 0%

Steady -state error: System does not even reach steady-state condition

Figure 5.19 Free Run on 10 RPM

From the figure above, we can see that without any PI, PD or PID controller, the system could not even achieve the desired speed. Even if the system reach the desired speed, the occurrence of steady-state error is so huge and out of control. This happened to the other system tested on 20 rpm and 30 rpm.



Desired Speed(RPM) = 20

$K_p = 10$

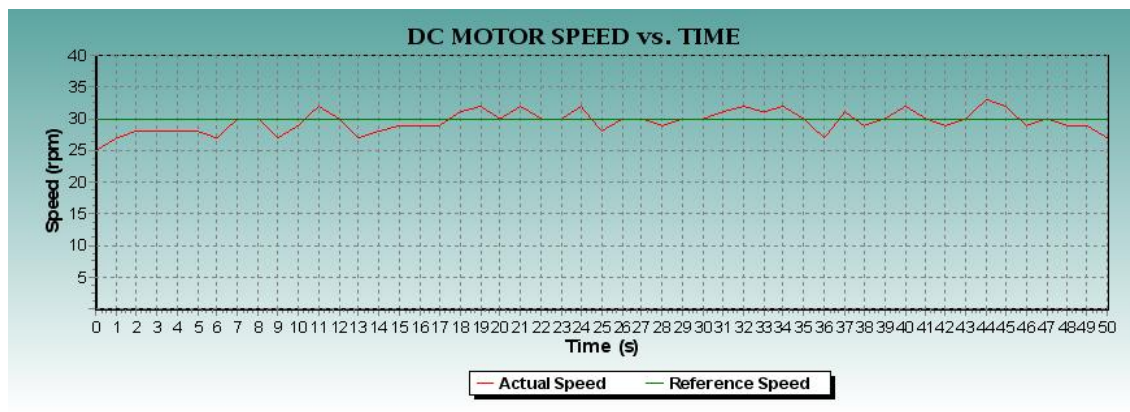
$K_i = 02$

$K_d = 02$

%Overshoot: 0%

Steady-state error: Still occur

Figure 5.20 Free Run on 20 RPM



Desired Speed(RPM) = 30

$K_p = 10$

$K_i = 02$

$K_d = 02$

%Overshoot: 0%

Steady-state error: Still occur

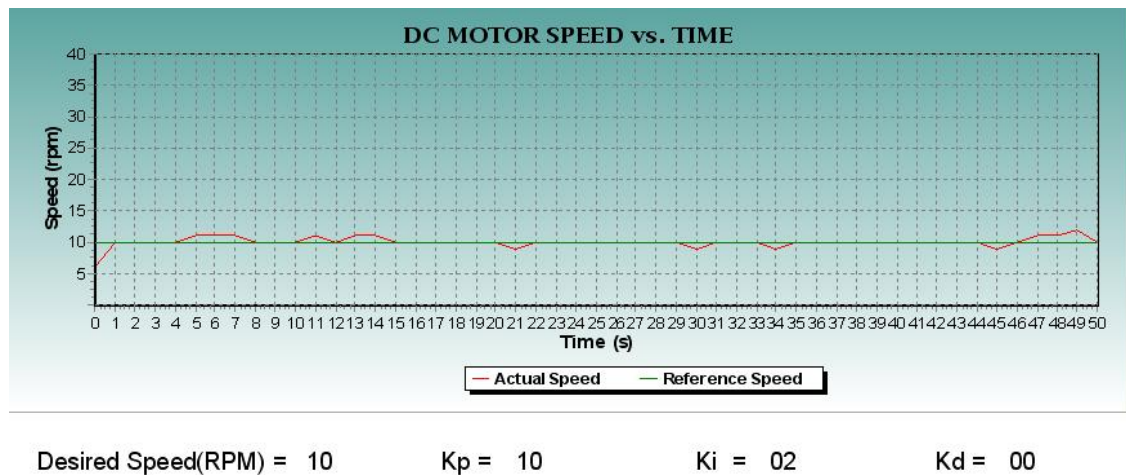
Figure 5.21 Free Run on 30 RPM

5.6 Performance With PID Controller Under Load

Once the system is inserted with load, it is later tested with the PI, PD and PID controller.

5.6.1 PI Controller

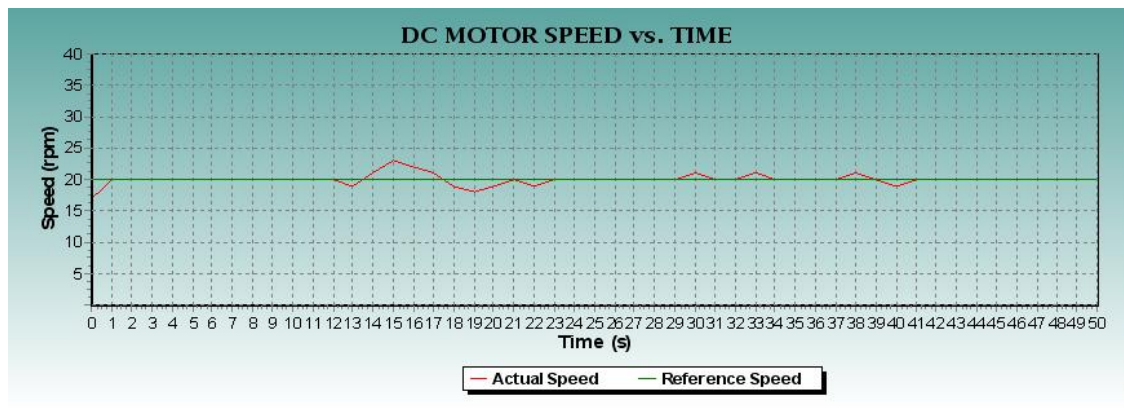
PI Controller is a special case in which the Derivative term is not used. The integral is used to react to any occurrence of error in the system. In other words, if the occurrence of error is getting rapid, the integral output will grow.



%Overshoot: 0%

Steady-state error: Still occur but reduced

Figure 5.22 PI Controller for 10 RPM With Load



Desired Speed(RPM) = 20

$K_p = 10$

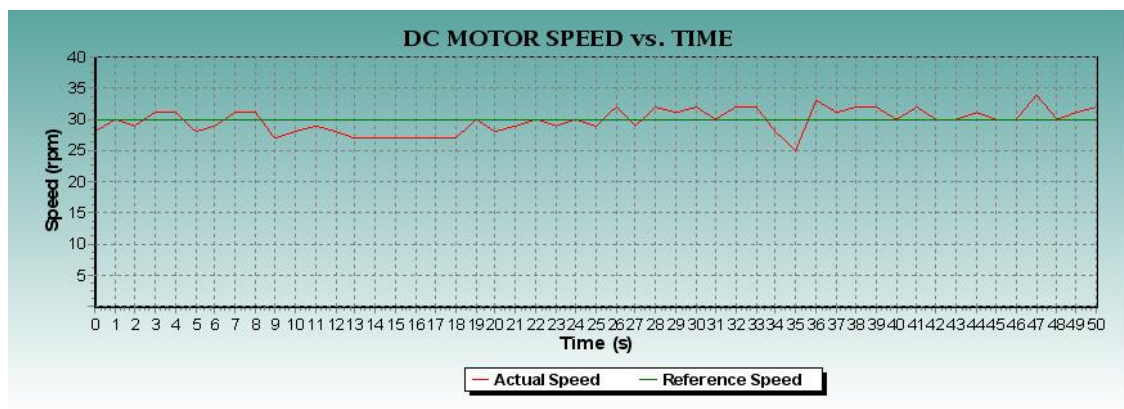
$K_i = 02$

$K_d = 00$

%Overshoot: 0%

Steady-state error: Still Occur

Figure 5.23 PI Controller for 20 RPM With Load



Desired Speed(RPM) = 30

$K_p = 10$

$K_i = 04$

$K_d = 00$

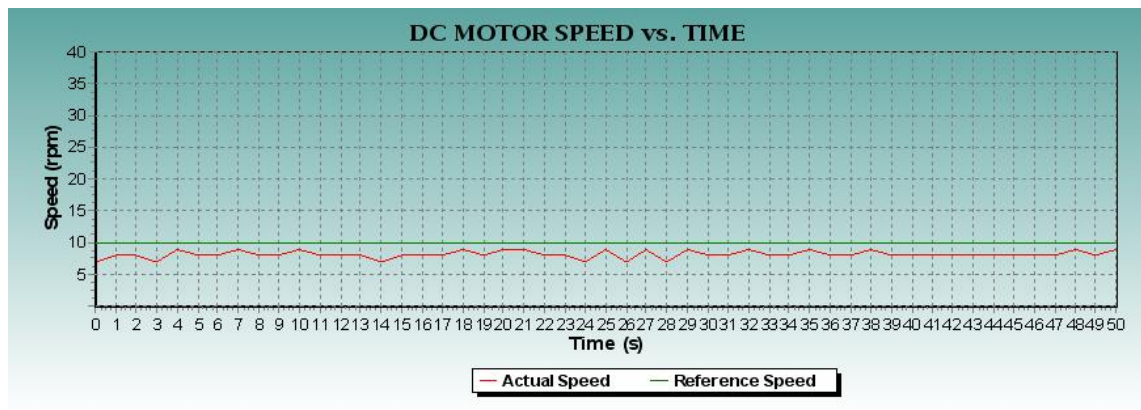
%Overshoot: 0%

Steady-state error: Still occur

Figure 5.24 PI Controller for 30 RPM With Load

5.6.1 PD Controller

PD controller is useful in reacting to sudden changes throughout the system. From the figure below, however when the derivative term is working alone without the integral term, it does not have the ability to overcome the occurrence of the same error and thus we could see that large amount of steady-state occurrence.



Desired Speed(RPM) = 10

$K_p = 10$

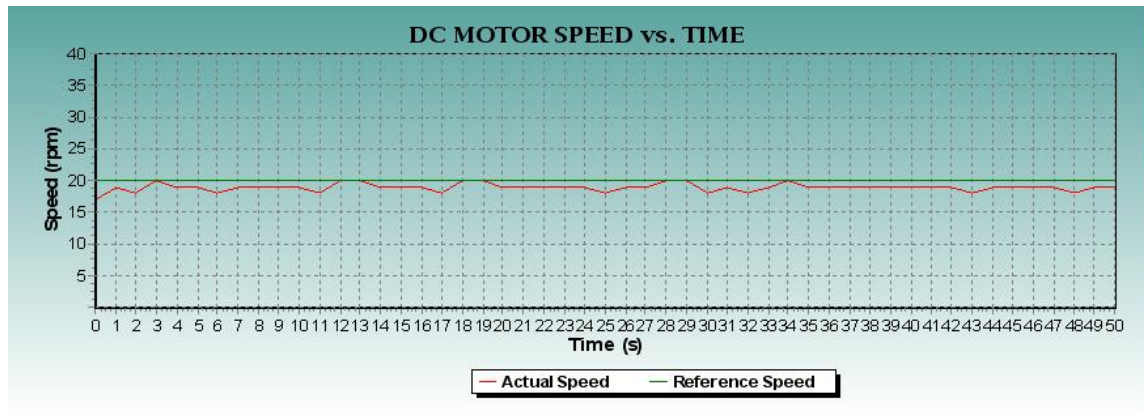
$K_i = 00$

$K_d = 02$

% Overshoot: 0%

Steady-state error: System does not even reached steady-state

Figure 5.25 PD Controller for 10 RPM With Load



Desired Speed(RPM) = 20

$K_p = 10$

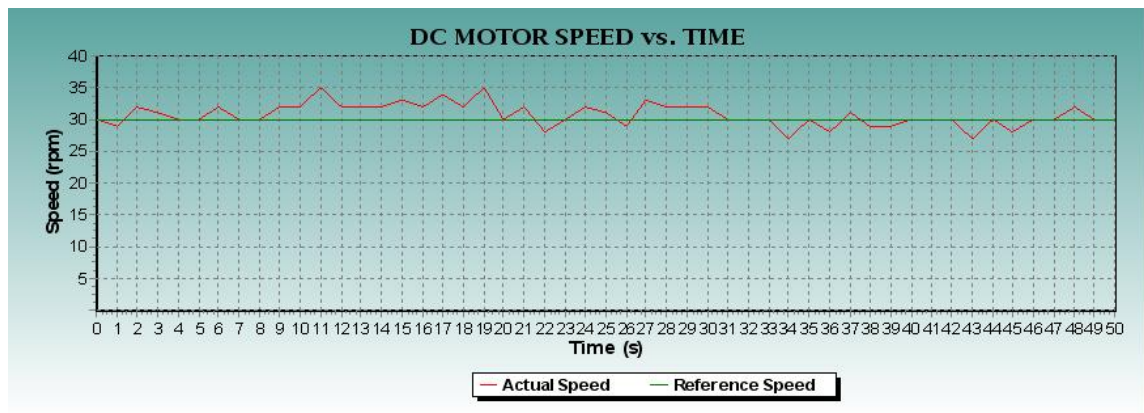
$K_i = 00$

$K_d = 02$

%Overshoot: 0%

Steady-state error: System does not even reached steady-state error

Figure 5.26 PD Controller for 20 RPM With Load



Desired Speed(RPM) = 30

$K_p = 10$

$K_i = 00$

$K_d = 02$

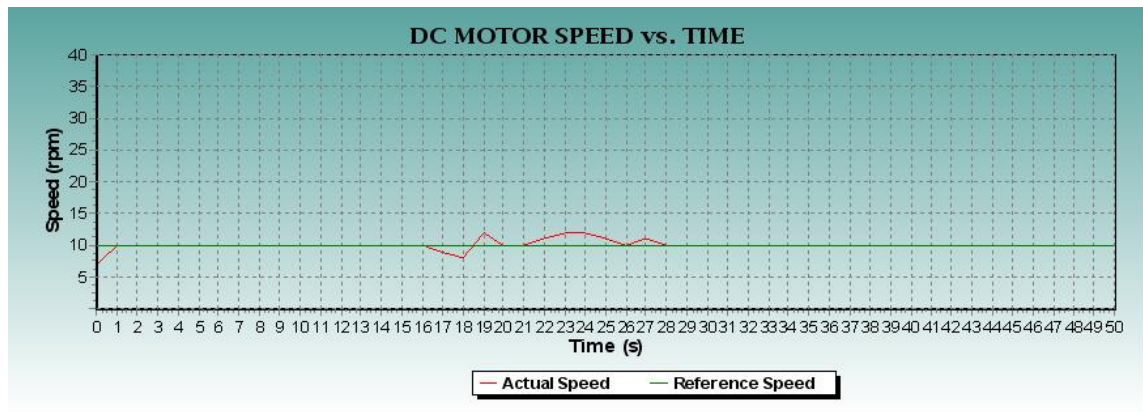
%Overshoot: 6.67%

Steady-state error: Still occur

Figure 5.27 PD Controller for 30 RPM With Load

5.6.2 PID Controller

When all the three terms are put together, the best result can be achieved. From the figure below it is seen that the output of all the three terms cause overshoot on all the three tested speed. However we can clearly see that the steady-state error is eliminated as the system goes. This is due to the effect of integral and derivative that counters any occurrence of error and also changes in error.



Desired Speed(RPM) = 10

$K_p = 10$

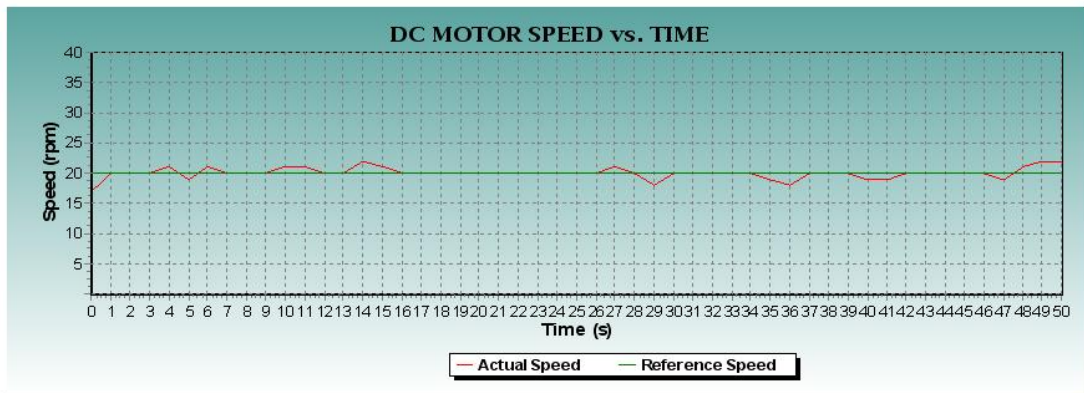
$K_i = 02$

$K_d = 02$

%Overshoot: 0%

Steady-state error: Still occur

Figure 5.28 PID Controller for 10 RPM With Load



Desired Speed(RPM) = 20

$K_p = 10$

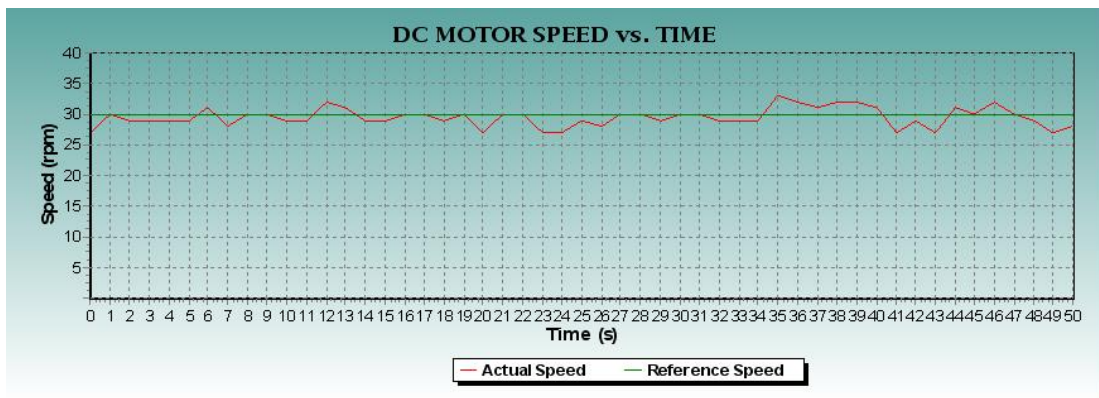
$K_i = 02$

$K_d = 02$

%Overshoot: 0%

Steady-state error: Still occur

Figure 5.29 PID Controller for 20 RPM With Load



Desired Speed(RPM) = 30

$K_p = 10$

$K_i = 02$

$K_d = 02$

%Overshoot: 0%

Steady-state error: Still occur

Figure 5.30 PID Controller for 30 RPM With Load

Table 5.3: Results of the Three Controller Working Under Load

Controller	PI			PD			PID		
Speed (RPM)	10	20	30	10	20	30	10	20	30
% Overshoot	0 %	0 %	0 %	5 %	5 %	6.67 %	0 %	0%	0%
Steady-State Error	Yes	Yes	Yes	-	-	-	Yes	Yes	Yes

From the table above, it is clearly understood that under the effect of load, the PD controller does not seem to be working at all. This is proved by the fact that the motor could not even reach steady-state. However, the PID controller has been able to drive the motor up to steady-state but still not been able to keep the motor stays at that steady condition. One reason for this is because of the disturbance in the mechanical connection between the load and the motor that caused the load to variant and thus it is hard for the PID algorithm to keep the motor at a steady-state.

CHAPTER 6

CONCLUSION & RECOMMENDATIONS

6.1 CONCLUSION

From all the experiment conducted on the implementation of the PID algorithm, there were several conclusions that can be made. The first is that the algorithm created all has played their effect on the system although it is the overshoot or the steady-state error. The steady-state error has been reduced or eliminated throughout the system while the overshoot has also been reduced. However, some of the experiment shows that the overshoot has been increased to 6.67% which is not meeting the exact requirement. However, for no load case, all the steady-state error has been eliminated as the system works.

Furthermore, from Table 5.2 above, the PID algorithm that was used was able eliminate steady-state error but only after a few seconds once the system has reached steady-state. One conclusion that can be made of this is due to the buildup of integral terms that has been able to detect the trend or occurrence of error as the system works. As a result, after a few seconds, steady-state error is totally eliminated.

In the Table 5.3, it can be concluded that under the effect of load, the PD controller does not seem to be working at all. This is proved by the fact that the motor could not even reach steady-state. However, the PID controller has been able to drive the motor up to steady-state but still not been able to keep the motor stays at that steady condition. One reason for this is because of the disturbance in the mechanical connection between the load and the motor that caused the load to variant and thus it is hard for the PID algorithm to keep the motor at a steady-state. This condition is similar to other controller run with the speed of 30 RPM. One conclusion that can be made from this problem is that the mechanical gear seems to be out of proportion when used under bigger speed. This cause variation on the speed itself. This variation has caused the algorithms problem in keeping the system to steady-state.

Overall, it can be said that the PID algorithm can be created by using low-cost MCU and applied to a system in controlling the speed of a loaded or unloaded DC motor.

6.2 Costing & Commercialization

The design of the PID Controller comes with the approach of simplicity and also user friendly. The controller uses PIC 18F2331 from the 18F family series of Microchip's PIC which already widely used in application regarding motor control. By using Printed Circuit Board (PCB) and also Surface-Mounted technology (SMT), it gives better performance under shake and vibration conditions and also robust design by its protective plastic container. Its small and lightweight design also allow the controller to be more mobile while offering less work space it its usage. Furthermore, with its simple and user friendly control panel the PID Controller can meets the demand of such controller in the industry. By coming with its own Control Panel, the PID Controller can be practically used on any system that needs the performance of its DC Motor to be

monitored and analyzed. Furthermore, the PID Controller also comes with a 30 A DC Motor Driver that is capable of driving up a DC Motor of 30 Amp peak current. The optional RS-232 interface feature also can provide full remote operation of the controller. As any standard electronic component, it also consists of a DC Supply Input and also a supply input for conventional power supply usage.

For commercialization purpose, the price of the unit is created based on the equipment or component used in the controller. This price list of components used is important if the product needs to be commercialized in the future. The complete list of prices is as below.

Table 6.1: Components Price List for Commercialization Purpose

Main Board	Price/Unit (RM)	Quantity	Price (RM)
Voltage Regulator LM7805	1.00	1	1.00
Voltage Regulator LM7812	1.00	1	1.00
PIC 18F2331	45.00	1	45.00
Headers	0.50	8	4.00
Diode D1N4148	0.10	1	0.10
Capacitor 4.7uF	0.10	4	0.40
Capacitor 0.1uF	0.10	1	0.10
Resistor 10K	0.04	1	0.04
Crystal 20MHZ	1.50	1	1.50
Plastic Casing	15.00	1	15.00
Input & Output Port	1.00	6	6.00
Motor Driver			
Cytron 30A DC Motor Driver	120.00	1	120.00
Total Price (RM)			194.14

The price above really meets the low-cost approach in the design of the controller. It will allow the controller to be sold at a cheaper price comparing to most of the PID Controller in the market. Furthermore, since the controller application is computer based, it should have a significant advantage over other PID controller in the market that relies on any other approach. This computer based application also allows the performance analysis to be made easily.

6.3 Recommendations

As previously stated, there are several improvement that can be made to the system. Some of the improvement is additional while others is crucial in making the system stay reliable in the future.

6.3.1 Real-time sampling

The MCU has been working perfectly in sampling the speed and making corrections. However, the PID Motor Control Panel works by displaying all the data sent from the MCU without displaying the real-time of data received. As a result, the PID Motor Control Panel will simply increased one second for any data received and later on creates problems in data sampling.

6.3.2 Application of Better Tuning Method

Better tuning method such as Ziegler- Nichols method can be applied to the current algorithm. In this method, the I and D gains are first set to zero. The “P” gain is increased until it reaches the “critical gain” K_c at which the output of the loop starts to oscillate. K_c and the oscillation period, P_c are used to set the gains.

6.3.3 Handling of Decimal Number

Currently, in the system, the sampling time used is only 500ms. However, this can be increased as long as the accuracy of the system is also increased. This can be done by allowing the controller to handle decimal number and not only integer.

6.3.4 Application of Universal Serial Bus Interface (USB)

The PID controller interface with the computer by using RS 232 Serial Communication protocol. However, nowadays, most computer manufacturers no longer install the RS 232 port on their motherboard. The usage of USB to RS 232 converter itself also sometimes create some error in accuracy of the data. One of the solution is to have an USB connections between the controller and the computer. This is possible as most PIC nowadays comes with features that allow USB interface.

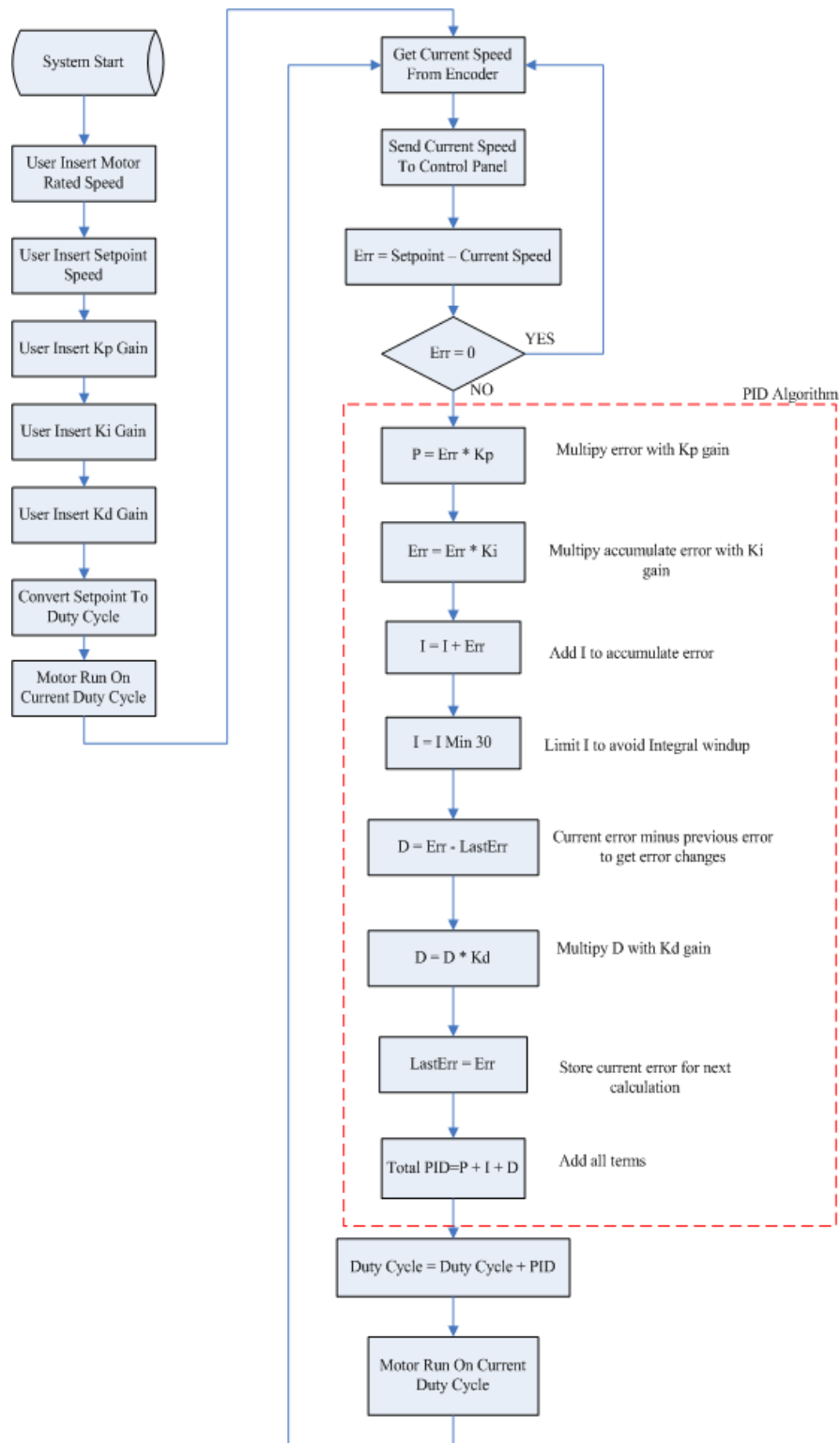
REFERENCES

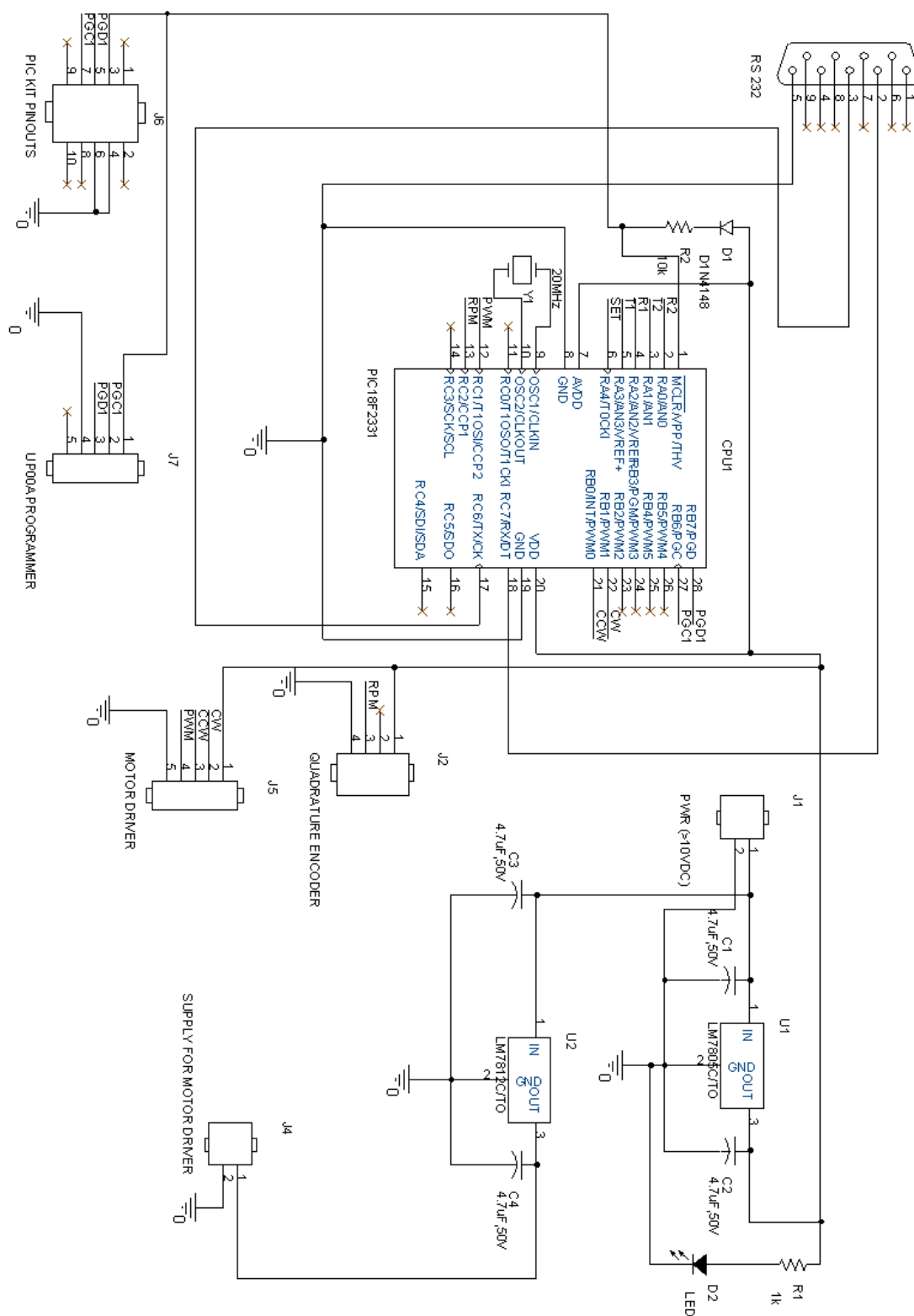
1. 14th January 2008, Citing Internet Sources, PID Design Method for DC Motor Speed Control,
URL <http://www.library.cmu.edu/ctms/ctms/examples/motor/pid2.htm#pid>
2. A.Irawan, M.F.Abas, M.S.Najib, S.Razali, N.Hasan, A.H.Mohd Hanafi, M.S.Jadin, B.Muhammad & M.R.Md Rejab, RDU070330, *Development of Multiple Motor Drive System Module with Real-time Data Acquisition Channels*, 1st Cycle Report, Faculty of Electrical & Electronics Engineering, Universiti Malaysia Pahang.
3. 21st January 2008, Citing Internet Sources, Basic *DC Motor Speed PID Control* with the *Infineon C167 Family*, URL www.hitex.co.uk/C166/pidex.html
4. 3rd August 2008, Citing Internet Sources, DC Motor Speed Modeling,
URL
<http://www.engin.umich.edu/group/ctm/examples/motor/motor.html#Problem>
5. Jianxin Tang, Rulph Chassaing, Walter J. Gomes III, Real-Time Adaptive PID Controller Using the TMS320C31 DSK, Dept. of Electrical and Computer Engineering, University of Massachusetts Dartmouth, North Dartmouth.
6. 2nd September 2008, Citing Internet Sources, PID controller
URL http://en.wikipedia.org/wiki/PID_loop
7. Chris Valenti, AN937 Implementing a PID Controller Using a PIC18 MCU, Microchip Technology Inc.

APPENDIX A

Flowchart of the Implemented PID algorithms for DC Motor speed control

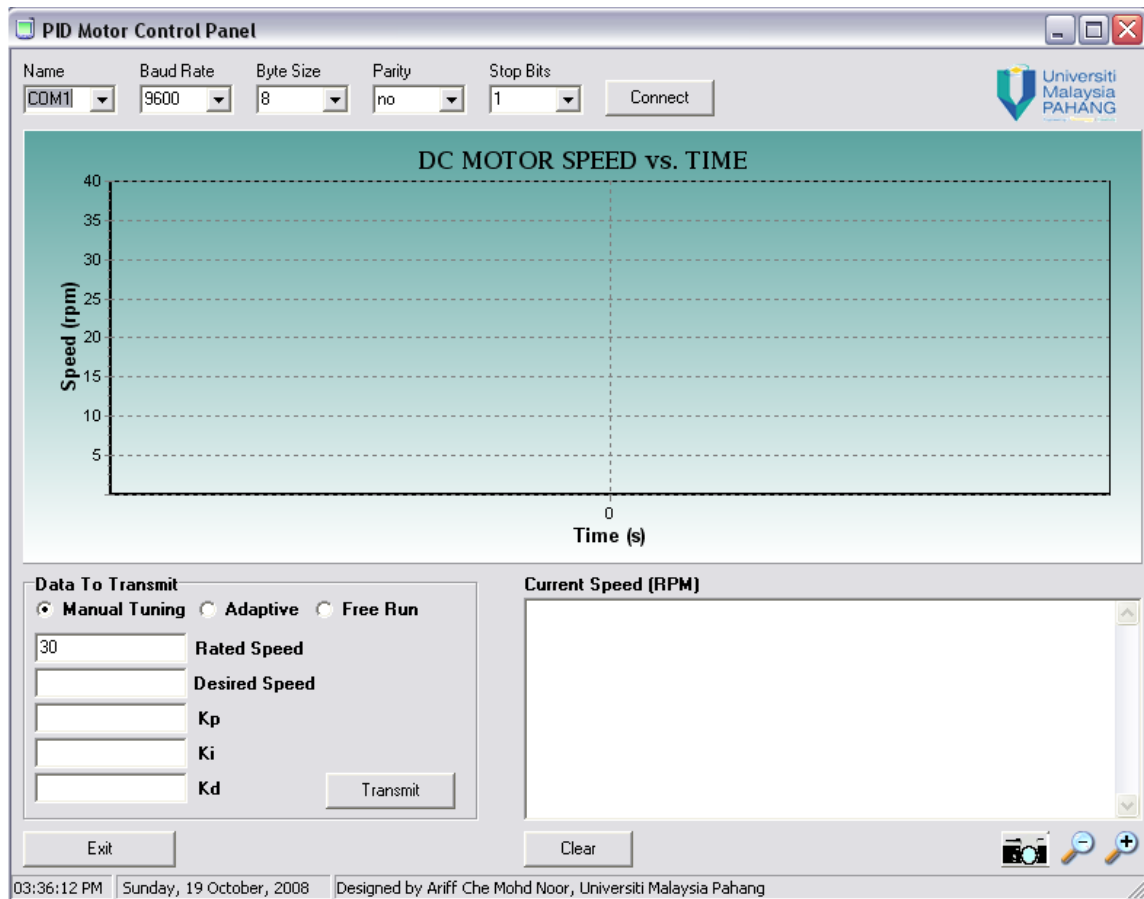
Application





APPENDIX C

Screenshots of PID Motor Control Panel



APPENDIX D

Snapshots of the Hardware with the DC Motor

