# CONTROLLER FOR A PROSTHETIC ARM

**FOO JONG WEI**

**A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor (Hons.) of Mechanical Engineering**

**Faculty of Engineering and Science**
**Universiti Tunku Abdul Rahman**

**April 2011**

# DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : _____

ID No. : _____

Date : _____

## APPROVAL FOR SUBMISSION

I certify that this project report entitled **"CONTROLLER FOR A PROSTHETIC ARM"** was prepared by **FOO JONG WEI** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering Science (Hons.) Mechanical Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Prof. Dato Dr. Goh Sing Yau

Date : _____

# ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Prof. Dato Dr. Goh Sing Yau for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement throughout the whole process. Special thanks to Mr. Danny for his guidance and advice.

# CONTROLLER FOR A PROSTHETIC ARM

## ABSTRACT

Nowadays, there are many types of high accuracy controller which are being implemented in the field of medical and biomechatronics, especially for the prosthetic arm. Due to the advancement of the technologies in the field, the development of the prosthetic arm has achieved great advancement and the demand for the high accuracy controller is getting higher. Thus, it is vital to find out and learn the control techniques of these advance fields in order to adapt the controlling techniques into current project. The earlier batch of students has designed the manual tuning of PID and Fuzzy Logic controller for the mechanical arm. This project is the improvement of the previous project by implementing the improved PID and Fuzzy Logic controller to control the position of the mechanical arm. The algorithms of the controllers mentioned above are developed in the form of C language. It is then programmed into the microcontroller PIC18F2550 that is able to control the position of the DC motor. The control system is a closed loop real time system, where the encoder that mounted on the DC motor provides feedback of actual position of the mechanical arm to the microcontroller and error will be generated. The controller will process the error signal and perform correction action to bring the mechanical arm to the desired position. The microcontroller can either function in PID mode or Fuzzy Logic mode, depends on the choice of the user. For PID mode the three gain constants are tuned manually while for Fuzzy Logic mode the membership functions are tuned manually as well. The performance of the PID and Fuzzy Logic controller are then discussed and compared.

# TABLE OF CONTENTS

**DECLARATION** ii
**APPROVAL FOR SUBMISSION** iii
**ACKNOWLEDGEMENTS** v
**ABSTRACT** vi
**TABLE OF CONTENTS** vii
**LIST OF TABLES** x
**LIST OF FIGURES** xi
**LIST OF FIGURES** xii
**LIST OF FIGURES** xiii
**LIST OF SYMBOLS / ABBREVIATIONS** xiv
**LIST OF SYMBOLS / ABBREVIATIONS** xv
**LIST OF APPENDICES** xvi

**CHAPTER**

**1 INTRODUCTION** **1**
1.1 Background 1
1.2 Aims and Objectives 3
1.3 Scope of Work 3
1.4 Thesis Outline 4

**2 LITERATURE REVIEW** **5**
2.1 Introduction 5
2.2 Prosthetic Arm Development – Myoelectric Arm 5
2.3 PID Controller 7

| | | | |
|---|---|---|---|
| | 2.3.1 | Theory of PID controller | 7 |
| | 2.3.2 | Overview of Tuning methods for PID Controller | 10 |
| 2.4 | Fuzzy Logic Controller | | 13 |
| | 2.4.1 | History of Fuzzy Logic | 13 |
| | 2.4.2 | Fuzzy Logic | 14 |
| | 2.4.3 | Theory of Fuzzy Logic Controller | 14 |
| 2.5 | Adaptive Fuzzy Logic Controller | | 21 |
| | 2.5.1 | Genetic Algorithm | 22 |
| | 2.5.2 | Fuzzy-Neural Control System | 26 |
| **3** | **METHODOLOGY** | | **30** |
| 3.1 | Introduction | | 30 |
| 3.2 | Closed-Loop Control System | | 31 |
| 3.3 | Simulation of Fuzzy Logic Controller Using Matlab | | 32 |
| 3.4 | C Language Programming Using MPLAB IDE | | 35 |
| 3.5 | C Language Compiling Using HITECH C | | 36 |
| 3.6 | Graphic User Interface (GUI) Built Using Visual Basic | | 36 |
| 3.7 | Hardware Implementation | | 37 |
| | 3.7.1 | Microcontroller | 38 |
| | 3.7.2 | RS232 Communication | 38 |
| | 3.7.3 | PWM Signal Generator | 38 |
| | 3.7.4 | Application of PWM and Encoder | 39 |
| **4** | **RESULTS AND DISCUSSIONS** | | **41** |
| 4.1 | Simulation of Fuzzy Logic Controller | | 41 |
| 4.2 | Mathematical Representation of Fuzzy Logic Controller | | 45 |
| 4.3 | Results and Performance of Fuzzy Logic Controller1 | | 47 |
| 4.4 | Results and Performance of Fuzzy Logic Controller2 | | 49 |
| 4.5 | Results and Performance of PID Controller | | 50 |
| 4.6 | Comparison between 3 Types of Developed Controller | | 60 |
| **5** | **CONCLUSION AND RECOMMENDATIONS** | | **62** |
| 5.1 | Conclusion | | 62 |

5.2 Problems 62
5.3 Recommendations 63

**REFERENCES** **64**

**APPENDICES** **65**

# LIST OF TABLES


| TABLE | TITLE | PAGE |
|---|---|---|
| 2.1 | Comparison between Different Tuning Methods | 11 |
| 2.2 | Effects of Increasing a Parameter Independently | 12 |
| 2.3 | Ziegler–Nichols method | 13 |
| 2.4 | Fuzzy Control Rules for Speed and Current Controllers | 21 |
| 3.1 | Table base Controller | 34 |
| 4.1 | Range of Error for Various Input Conditions | 43 |
| 4.2 | PWM Values for Various Output Conditions | 44 |
| 4.3 | Rule Base | 45 |
| 4.4 | Comparison between 3 Types of Controller (no load) | 60 |
| 4.5 | Comparison between 3 Types of Controller (loaded) | 60 |

# LIST OF FIGURES


| FIGURE | TITLE | PAGE |
|---|---|---|
| 2.1 | Block Diagram of Microprocessor Based Multifunction Myoelectric Arm | 7 |
| 2.2 | Block Diagram of PID Controller | 8 |
| 2.3 | Fuzzy System Structure | 14 |
| 2.4 | Membership Functions | 15 |
| 2.5 | Various Types of Membership Functions | 16 |
| 2.6 | 1 input, 1 Output Rule Base with Non-Singleton Output Sets | 19 |
| 2.7 | A Practical Illustration | 20 |
| 2.8 | 8 Bit Binary String | 23 |
| 2.9 | Crossover with 8 Bit Binary String | 24 |
| 2.10 | Crossover with 8 Bit Binary String | 24 |
| 2.11 | General Flow Chart for GA | 25 |
| 2.12 | Simplified Schematic of the ANN Training Process | 26 |
| 2.12 | Structure of the ANFIS Network | 28 |
| 3.1 | Block Diagram of DC Motor Position Control System | 31 |
| 3.2 | Number of Output and Input is Set | 33 |
| 3.3 | Constructed Input Membership Functions | 33 |

# LIST OF FIGURES


| FIGURE | TITLE | PAGE |
|---|---|---|
| 3.4 | Constructed Output Membership Functions | 34 |
| 3.5 | Rule Base of Fuzzy Logic Controller | 34 |
| 3.6 | Simulation of Fuzzy Logic Controller to Obtain Output | 35 |
| 3.7 | GUI Built Using Visual Basic | 37 |
| 3.8 | Application of PWM | 40 |
| 4.1 | Fuzzy Logic Controller Interface | 41 |
| 4.2 | Simulation of Fuzzy Logic Controller | 42 |
| 4.3 | Simulation of Fuzzy Logic Controller | 42 |
| 4.4 | Optimized Input Membership Functions | 43 |
| 4.5 | Optimized Output Membership Functions | 44 |
| 4.6 | Rule Base of the Fuzzy Logic Controller | 45 |
| 4.7 | Response Graph under No Load Condition | 47 |
| 4.8 | Response Graph where the Arm under Loaded Condition (200g) | 48 |
| 4.9 | Response Graph under No Load Condition | 49 |
| 4.10 | Response Graph where the Arm under Loaded Condition (200g) | 50 |
| 4.11 | Response Graph of PID Controller | 51 |
| 4.12 | Response Graph of PID Controller | 51 |

# LIST OF FIGURES


| FIGURE | TITLE | PAGE |
|---|---|---|
| 4.13 | Response Graph of PID Controller | 52 |
| 4.14 | Response Graph of PID Controller | 52 |
| 4.15 | Response Graph of PID Controller | 53 |
| 4.16 | Response Graph of PID Controller | 53 |
| 4.17 | Response Graph of PID Controller | 54 |
| 4.18 | Response Graph of PID Controller | 54 |
| 4.19 | Response Graph of PID Controller | 55 |
| 4.20 | Response Graph of PID Controller | 55 |
| 4.21 | Response Graph of PID Controller | 56 |
| 4.22 | Response Graph of PID Controller | 56 |
| 4.23 | Response Graph of PID Controller | 57 |
| 4.24 | Response Graph of PID Controller | 58 |
| 4.25 | Response Graph of PID Controller | 58 |
| 4.26 | Response Graph of PID Controller | 59 |
| 4.27 | Response Graph of PID Controller | 59 |

## LIST OF SYMBOLS / ABBREVIATIONS

| | |
|---|---|
| $K_p$ | Proportional gain |
| $K_i$ | Integral gain |
| $K_d$ | Derivative gain |
| $T_i$ | Integral time, s |
| $T_d$ | Derivative time, s |
| $\mu$ | crisp output value |
| $X_i$ | running point in a discrete universe |
| $\mu\ (X_i)$ | membership value in the membership function |
| $S_i$ | position of the singleton $i$ in the universe |
| $\mu\ (S_i)$ | firing strength $\alpha_i$ of rule $i$ |

| | |
|---|---|
| EMG | Electromyography |
| ANN | Artificial Neural Network |
| NN | Neural Network |
| TMR | Targeted Muscle Reinnervation |
| TSR | Targeted Sensory Reinnervation |
| PID | Proportional, Integral, Derivative |
| FL | Fuzzy Logic |
| FLC | Fuzzy Logic Controller |
| PL | Positive Large |
| PM | Positive Medium |
| PS | Positive Small |
| NL | Negative Large |
| NM | Negative Medium |
| NS | Negative Small |
| ZE | Zero |

# LIST OF SYMBOLS / ABBREVIATIONS

| | |
|---|---|
| COG | Centre of Gravity |
| MOM | Mean of Maximum |
| GA | Genetic Algorithm |
| ANFIS | Adaptive Neurofuzzy Inference System |
| GUI | Graphic User Interface |
| PWM | Pulse Width Modulation |

# LIST OF APPENDICES


| APPENDIX | TITLE | PAGE |
|---|---|---|
| A | C Programming Source Code (Fuzzy Logic Controller1) | 65 |
| B | C Programming Source Code (Fuzzy Logic Controller2) | 71 |
| C | C Programming Source Code (PID Controller) | 79 |
| D | Visual Basic Source Code (GUI) | 85 |
| E | Figures of Prototype of Prosthetic Arm | 90 |

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

The number of aged or physically handicapped people requiring someone's assistance in everyday life has been increasing in recent years. These are the peoples who suffer from the lost of limbs such as arms or legs due to accidents, disease, trauma, congenital defects and so on. They faced a lot of difficulties and inconveniences in their daily life movements and activities. Hence, most of them require assistance from their family members. However, some of the handicapped peoples are very independent and they prefer to depend on the prosthetic devices to help them in life. The prosthetic device is a common device in the medical field today and it is quite convenient for the patients to be equipped with the suitable prosthetic limb since the development of robotic prosthetic limb has achieved great advancement and their availability is high.

In this modern era, with the aid of advance biomechatronics technologies and science knowledge, the development of prosthetic devices has achieved a significant improvement and advancement in the medical and biomechatronics field over the years. For example, the myoelectric limbs control the limbs by converting muscle movements to electrical signals through EMG (electromyography) pattern recognition. Besides, myoelectric limbs allow the amputees to more directly control the artificial limb and these have become much more common than cable operated limbs. On the other hand, techniques such as TMR and TSR also being introduced into the medical field. Targeted muscle reinnervation (TMR) is a technique in which

motor nerves which previously controlled muscles on an amputated limb are surgically rerouted such that they reinnervate a small region of a large, intact muscle, such as the pectoralis major. As a result, when a patient thinks about moving the thumb of his missing hand, a small area of muscle on his chest will contract instead. By placing sensors over the reinervated muscle, these contractions can be made to control movement of an appropriate part of the robotic prosthesis. The procedure of targeted sensory reinnervation (TSR) is similar to TMR, except that sensory nerves are surgically rerouted to skin on the chest, rather than motor nerves rerouted to muscle. The patient then feels any sensory stimulus on that area of the chest, such as pressure or temperature, as if it were occurring on the area of the amputated limb which the nerve originally innervated.

To ensure that the robotic prosthetic limb could work, it must have several components to integrate it into the body's function. Firstly, biosensors detect signals from the patient's nervous or muscular systems. It then relays this information to a controller located inside the device, and processes feedback from the limb and actuator such as position, force and so on and then sends it to the controller. Next, mechanical sensors process aspects affecting the prosthetic device and the control elements such as limb position, applied force and load and relay this information to the biosensor or controller.

The controller is connected to the user's nerve and muscular systems and the device itself. It sends intention commands from the user to the actuators of the device and the actuator mimics the actions of a muscle in producing force and movement. Besides, the controller also interprets feedback from the mechanical and biosensors to the user. Other than that, the controller is also responsible for the monitoring and control of the movements of the device.

The robotic prosthetic limb that installed as the patient’s arm must be implemented with control method of high accuracy and high precision in order to allow the patient to control the prosthetic limb in any ways and directions on their will with accurate speed and motion. Besides, this also enables them to control the prosthetic limb easily and more directly. Therefore, they can act alone independently and do things by their own will. Since the robotic prosthetic limbs used in the

medical field are being combined with high accuracy controller, it is interesting to find out the detail parts of the controller and at the same time learn the techniques used and then apply it as much as possible during the process of designing controller for the prosthetic device.

## 1.2 Aims and Objectives

The main objectives of the project are:

- To develop improved PID and Fuzzy Logic controller for the mechanical arm as prototype of prosthetic arm.
- To compare the performance of PID controller and Fuzzy Logic controller in controlling the mechanical arm.
- To design a user-friendly graphic user interface (GUI) for the real time display of positional control of the mechanical arm.

## 1.3 Scope of Work

Previously, the earlier batch of students have constructed the electronic circuit and they have also come out with two sets of control algorithms in C language, namely PID and Fuzzy Logic control C algorithms to control the position of the mechanical arm. Hence, the scope of work now is to improve the previous project by develop an improved controller for the mechanical arm. After tested their controller C code and desired to continue from that part to do further improvement, it is found that their code is not working, thus the prototype could not work as well, might be due to certain error or other factors. Thus, everything will be started from scratch and at the same time focus will be given into the developing of improved PID controller and Fuzzy Logic controller in controlling of the movement and position of the mechanical arm. The controller algorithms will be programmed into the microcontroller, PIC18F2550 by using HITECH C compiler. RS232 serial port communication protocol is used to communicate between the circuit and the

computer. On the other hand, a graphic user interface (GUI) will be designed to show the results and performances of the positional control of mechanical arm by PID and Fuzzy Logic controller.

## 1.4 Thesis Outline

This report has been organized in the following order:

- In chapter 1, the overview of the objectives and scope of the project is highlighted.
- In chapter 2, the literature review of the controlling of prosthetic arm by using PID and Fuzzy Logic controller.
- In chapter 3, the methodology in developing the improved PID and Fuzzy Logic Controller will be discussed.
- In chapter 4, the results and performances of PID and Fuzzy Logic controller will be compared and the results will be further discussed.
- In chapter 5, conclusion and suggestions for further improvement are made.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Introduction

This chapter includes the study of prosthetic arm development, PID controller, Fuzzy logic controller and adaptive Fuzzy Logic Controller.

## 2.2 Prosthetic Arm Development – Myoelectric Arm

Electromyography (EMG) is a medical technique for measuring muscle response to nervous stimulation. EMG is being performed using an instrument called an electromyograph to produce a record called an electromyogram. An electromyograph detects the electrical potential generated by muscle cells when these cells contract.

A myoelectric prosthesis uses EMG signals or potentials from voluntarily contracted muscles within a person's residual limb on the surface of the skin to control the movements of the prosthesis, such as elbow flexion/extension, wrist supination/pronation (rotation) or hand opening/closing of the fingers. Prosthesis of this type utilizes the residual neuro-muscular system of the human body to control the functions of an electric powered prosthetic hand, wrist or elbow. This is as opposed to an electric switch prosthesis, which requires straps and cables actuated by body movements to actuate or operate switches that control the movements of prosthesis or one that is totally mechanical. It has a self suspending socket with pick

up electrodes placed over flexors and extensors for the movement of flexion and extension respectively. Advances in myoelectric technology in recent years have made these upper extremity prosthetic components far superior to body powered equivalents. Myoelectric arm components perform better than conventional prostheses in terms of function, weight, comfort, and cosmetics.

The control system designed to control the movement of the prosthetic arm is made up of three main sections: input signal processing, motor control algorithms and motor driver sections. The input signal processing contains a real-time data retrieval section and history buffer section. These are used in combination to capture and measure EMG signals from the subject's bicep muscles and produce a stream of steady RMS values. This array of RMS values is then used to determine the next movement of the arm. Subsequently, it is converted into the motor control output voltage by the motor device drivers and is amplified to control the movement of the arm. Using the proper selection of threshold and fitness levels, any type of workout may be achieved for the subject.

The control system developed uses information collected from the amputee to train a pattern classifier in the control system to recognize the contraction patterns specific to each amputee. The basic operation of the control system is illustrated in Figure 2.1. The classifier uses features extracted from the first 200ms of myoelectric activity following the initiation of a contraction to determine the intent of the amputee. The classifier matches this feature set with the features sets obtained from the amputee during the initial system calibration. The closest match is used to select which device is to be controlled. Control of this device continues until the signal level returns to a predetermined low level.

As shown in Figure 2.1, the controller can operate in two modes, which are PC-Interface (training and configuration) and Prosthetic Control (normal operation). The PC-Interface mode is used to train the control system to recognize the myoelectric control inputs for each subsequent of training of the ANN. (Artificial Neural Network)

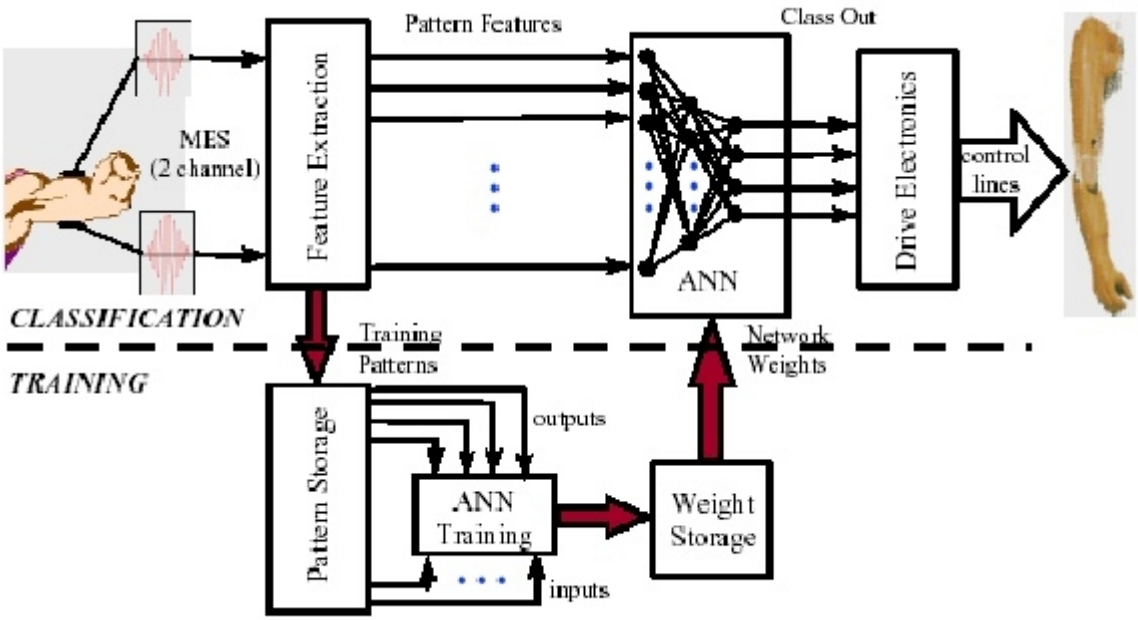


**Figure 2.1: Block Diagram of Microprocessor Based Multifunction Myoelectric Arm**

## 2.3 PID Controller

### 2.3.1 Theory of PID controller

A proportional–integral–derivative controller, as known as PID controller is the most common used feedback controller in industrial control systems. A PID controller calculates an error value as the difference between a measured process variable and a desired setpoint at the summing 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.

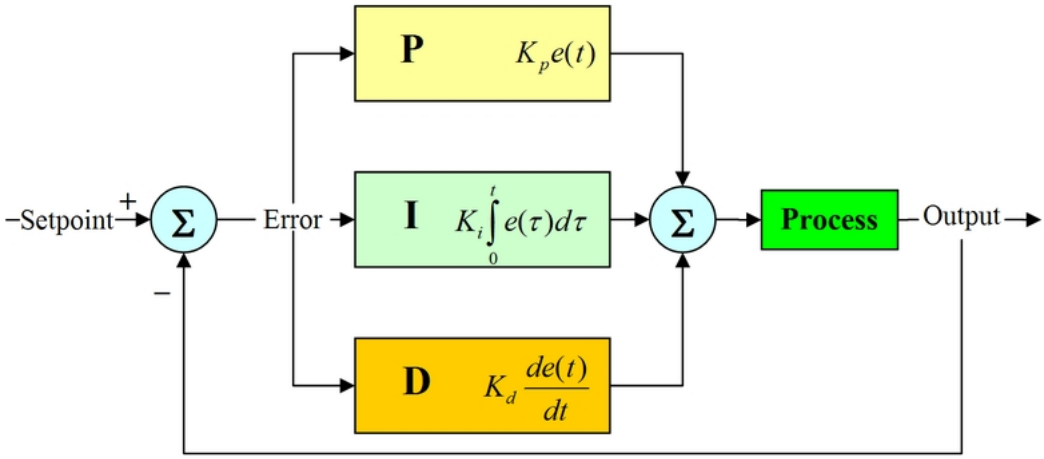


**Figure 2.2: Block Diagram of PID Controller**

The PID controller algorithm involves three separate parameters, also as known as three-term control: the proportional, the integral and derivative values, denoted 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 or an actuator such as the position and speed of the robotic arm, position of a control valve, power supply of a heating element and so on. 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. Some applications may require using only one or two modes to provide the appropriate system control. This is achieved by setting the gain of undesired control outputs to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions.

There are three main actions in the PID controller, namely proportional action, integral action and derivative action. The proportional action makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by proportional gain, $K_p$. The contribution from the integral action, sometimes called reset is proportional to both the magnitude of the error and the duration of the error. Integrating the error gives the accumulated

offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output. The magnitude of the contribution of the integral term to the overall control action is determined by the integral gain, $K_i$. The rate of change of the process error is calculated by determining the slope of the error over time (i.e., its first derivative with respect to time) and multiplying this rate of change by the derivative gain $K_d$. The magnitude of the contribution of the derivative action to the overall control action is termed the derivative gain, $K_d$.

The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining u(t) as the controller output, the final form of the PID algorithm is:

$$\mathrm{u(t)} = \mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

Where the tuning parameters are:

$K_p$ = Proportional gain

$K_i$ = Integral gain

$K_d$ = Derivative gain

Or the function can be written as

$$\mathrm{U(t)} = K_p[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d \frac{de(t)}{dt}]$$

Where:

$T_i$ = Integral time

$T_d$ = Derivative time

By tuning the three constants in the PID controller algorithm, namely $K_p$,$K_i$ and $K_d$, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the

responsiveness of the controller to an error, the degree to which the controller overshoots the setpoint and the degree of system oscillation.

The effect of tuning the three constants:

Proportional gain, $K_p$:

Larger values typically mean faster response since the larger the error, the larger the proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

Integral gain, $K_i$:

Larger values imply steady state errors are eliminated more quickly. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before reaching steady state.

Derivative gain, $K_d$:

Larger values decrease overshoot, but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

The tuning of the three constant mentioned above is crucial in obtaining an optimum performance of the control system, in this case is to obtain high accuracy positioning of the DC motor.

### 2.3.2 Overview of Tuning methods for PID Controller

Tuning a control loop is the adjustment of its control parameters to the optimum values for the desired control response. Stability is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and some desiderata conflict. Furthermore, some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load; this can be corrected by gain scheduling, which is using different parameters in different operating regions. PID

controllers often provide acceptable control even in the absence of tuning, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning.

It is important to find out which tuning method is suitable and applicable to the process plant in the industrial. Besides, in medical and biomechatronics field, the tuning methods of the PID controller applied in controlling of prosthetic devices are very important since there is a must to ensure the high accuracy positioning of the prosthetic device.

There are several methods for tuning a PID control loop. The most effective methods generally involve the development of some form of process model, then choose P, I, and D based on the dynamic model parameters. Manual tuning methods can be relatively inefficient, particularly if the loops have response times on the order of minutes or longer.

**Table 2.1: Comparison between Different Tuning Methods**

| 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 models. | Some math. Offline method. Only good for first-order processes. |

The choice of method will depend largely 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 involves 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 of the three parameters in PID controller has been investigated and implemented by previous batch students in the control of prosthetic arm project. The main objective now is to develop an adaptive type controller that can automatically tune the parameters when the load of the arm is changing, in other words changing in angle and weight of the arm.

**Table 2.2: Effects of Increasing a Parameter Independently**

| Parameter | Rise time | Overshoot | Settling time | Steady-state error | Stability |
|---|---|---|---|---|---|
| $K_p$ | Decrease | Increase | Small change | Decrease | Degrade |
| $K_i$ | Decrease | Increase | Increase | Decrease significantly | Degrade |
| $K_d$ | Minor decrease | Minor decrease | Minor decrease | No effect in theory | Improve if $K_d$ is small |

It is important for us to understand the effects of tuning the three constant would have towards the transient response of the process and this might be helpful in making better tuning of the control loop with input of suitable values of $K_p$, $K_i$ and $K_d$. Besides, it is a basic knowledge that must learned before moving into the development of an adaptive PID controller.

#### 2.3.2.1 Ziegler–Nichols method

Ziegler–Nichols method is one of the adaptive tuning methods of PID controller, introduced by John G. Ziegler and Nathaniel B. Nichols in the 1940s. As in the method above, the $K_i$ and $K_d$ gains are first set to zero. The P gain is increased until it reaches the ultimate gain, $K_u$, at which the output of the loop starts to oscillate. $K_u$ and the oscillation period $P_u$ are used to set the gains as shown:

**Table 2.3: Ziegler–Nichols method**

| Control Type | $K_p$ | $K_i$ | $K_d$ |
|---|---|---|---|
| P | $0.50K_u$ | - | - |
| PI | $0.45K_u$ | $1.2K_p / P_u$ | - |
| PID | $0.60K_u$ | $2K_p / P_u$ | $K_pP_u / 8$ |

Ziegler–Nichols method is one of the auto-tuning methods that can be applied in the designing of the adaptive PID controller to control the position of the DC motor.

## 2.4 Fuzzy Logic Controller

### 2.4.1 History of Fuzzy Logic

The concept of Fuzzy Logic (FL) was conceived by Lotfi Zadeh in1965, a professor at the University of California at Berkley, and presented not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership. This approach to set theory was not applied to control systems until the 70's due to insufficient small-computer capability prior to that time. Professor Zadeh reasoned that people do not require precise, numerical information input, and yet they are capable of highly adaptive control. If feedback controllers could be programmed to accept noisy, imprecise

input, they would be much more effective and perhaps easier to implement. Unfortunately, U.S. manufacturers have not been so quick to embrace this technology while the Europeans and Japanese have been aggressively building real products around it.

### 2.4.2 Fuzzy Logic

Fuzzy logic is a form of multi-valued logic derived from fuzzy set theory to deal with reasoning that is approximate rather than precise. In contrast with crisp logic, where binary sets have binary logic, fuzzy logic variables may have a truth value that ranges between 0 and 1 and is not constrained to the two truth values of classic propositional logic. Furthermore, when linguistic variables are used, these degrees may be managed by specific functions.

### 2.4.3 Theory of Fuzzy Logic Controller

A simple block diagram of a Fuzzy Logic Controller (FLC) is shown in Figure 2.3. There are four major components in the Fuzzy Logic Controller, which are fuzzification block, fuzzy knowledge-base block, fuzzy inference engine and defuzzification block. The functions of the blocks and working principles of the fuzzy system are briefly summarized.

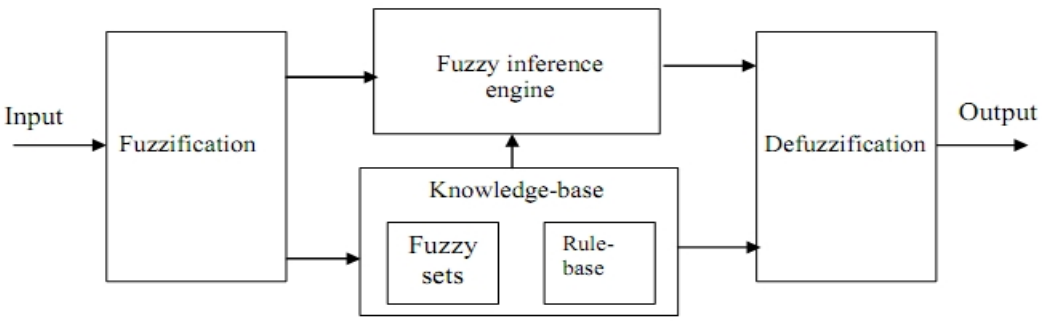


**Figure 2.3: Fuzzy System Structure**

### 2.4.3.1 Fuzzification and Membership Functions

The fuzzification block performs the following tasks:

- Measures the value of input variables.
- Performs a scale mapping that transfers the range of values of input variables into the corresponding universes of discourse.
- Performs the function of fuzzification, which converts input data into suitable linguistic values that may be viewed as labels of fuzzy sets.

The input signals to FLC are scaled using appropriate scaling factors. These scaled input data are then converted into linguistic variables, which may be viewed as labels of fuzzy sets. Fuzzy sets can be characterized by membership functions. The group of membership functions cannot be assigned arbitrarily, it depends on the characteristics of the system. Figure 2.4 shows one of the many types of membership function assignment, in which the number of reference fuzzy subset is seven: positive large (PL), positive medium (PM), positive small (PS), negative large (NL), negative medium (NM), negative small (NS), and zero (ZE).

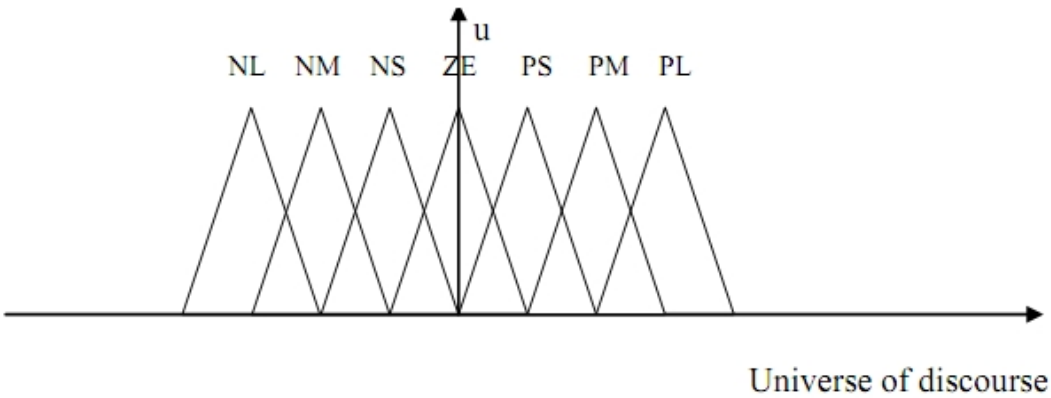


**Figure 2.4: Membership Functions**

These membership functions can be represented by using graphical representations, a fuzzy association pairs, matrices, and mathematical equations. There are many types of membership functions e.g., the bell-shaped, linear function, triangular function, trapezoidal function and exponential function. In assigning membership functions, a single function or combination of several functions may be

used in the same universe of discourse. The choice of membership function shape is mainly dependent on the designer preference. The triangular shape is comparatively easier to be represented inside a microcontroller hence it is suitable to be used in the designing of the Fuzzy Logic Controller algorithm.

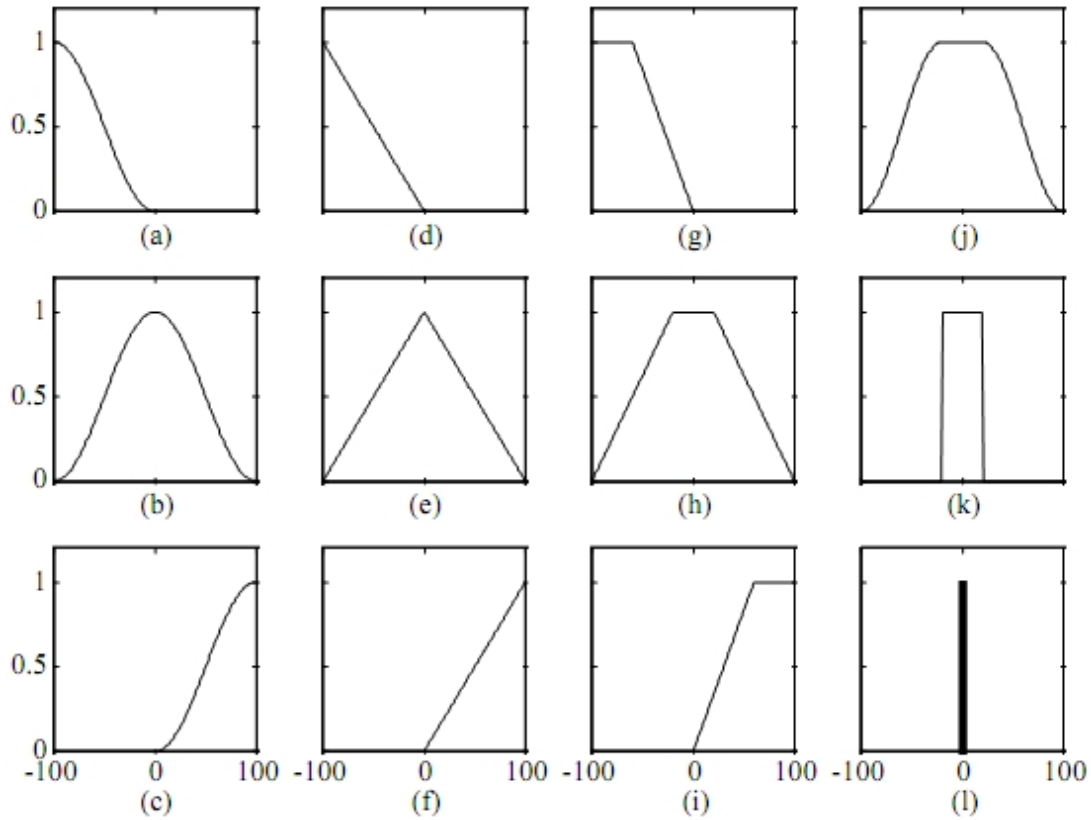


**Figure2.5: Various Types of Membership Functions**

### 2.4.3.2 Knowledge-Base

The knowledge base is comprised of two components namely called fuzzy sets (data base) and fuzzy control rule base. The concepts associated with fuzzy sets are used to characterize fuzzy control rules and fuzzy data manipulation in an FLC. These concepts are subjectively defined and based on experience. So, it should be noted that the correct choice of the membership functions of a term set plays an essential role in the success of an application.

The dynamic behavior of a fuzzy control logic system is characterized by a set of linguistic control rules based on expert knowledge. The fuzzy rule base consists of a set of linguistic control rules written in the form:

*IF a set of conditions are satisfied (premise), THEN a set of consequences are inferred*

The collection of fuzzy control rules that are expressed as fuzzy conditional statements forms the rule base or the rule set of an FLC. The selection of the linguistic variables has a substantial effect on the performance of an FLC. Experience and engineering knowledge play an important role during this selection stage. In particular, the choice of linguistic variables and their membership function have a strong influence on the linguistic structure of an FLC. Typically, the linguistic variables in an FLC are the state, state error, state error derivative, state error integral and so on.

One of the key problems is to find the appropriate fuzzy control rules. In general, there are four models of derivation of fuzzy control rules:

- Using the experience and knowledge of an expert.
- Modelling the control actions of the operator.
- Using a fuzzy model of a process.
- Using self-organized fuzzy controllers.

#### 2.4.3.3 Fuzzy Inference Engine

The fuzzy engine is the kernel of a fuzzy logic controller, which has capability of simulating human decision-making based on fuzzy concepts and of inferring fuzzy control actions using fuzzy implication (fuzzy relation) and the rules of inference in fuzzy logic. This means that the fuzzy inference engine handles rule inference where human experience can easily be injected through linguistic rules.

#### 2.4.3.4 Defuzzification

The defuzzification block performs the following functions:

- Scale mapping, which converts the range of values of output variables into corresponding universes of discourse.
- Transforms the fuzzy control actions to continuous (crisp) signals at the output, which can be applied to the physical plant.

A defuzzification strategy is aimed at producing a non-fuzzy control action that best represents the possibility of distribution of an inferred fuzzy control action. Unfortunately, there is no systematic procedure for choosing a defuzzification strategy. Zadeh first pointed out this problem and made tentative suggestions for dealing with it. At present, the commonly used strategies may be described as the max criterion, the mean of maximum and the centre of gravity.

For the method of centre of gravity (COG), the crisp output value $\mu$ (white line in Figure 2.5) is the abscissa under the centre of gravity of the fuzzy set:

$$u = \frac{\sum_i \mu(x_i)\, x_i}{\sum_i \mu(x_i)}$$

Where:
$X_i$ is a running point in a discrete universe
$\mu(X_i)$ is its membership value in the membership function

The expression can be interpreted as the weighted average of the elements in the support set. For the continuous case, replace the summations by integrals. It is a commonly used method although its computational complexity is relatively high. This method is also called centroid of area.

If the membership functions of the conclusions are singletons, the method of centre of gravity method for singletons (COGS) is used. The output value is:

$$u = \frac{\sum_i \mu(s_i)\, s_i}{\sum_i \mu(s_i)}$$

Where:

$S_i$ is the position of the singleton $i$ in the universe

$\mu(S_i)$ is equal to the firing strength $\alpha_i$ of rule $i$

This method has a relatively good computational complexity, and $\mu$ is differentiable with respect to the singletons $S_i$, which is useful in neuro-fuzzy system.

For the method of mean of maximum (MOM), an intuitive approach is to choose the point with the strongest possibility, in other words maximal membership. It may happen, though, that several such points exist, and a common practice is to take the mean of maximum (MOM). This method disregards the shape of the fuzzy set, but the computational complexity is relatively good.

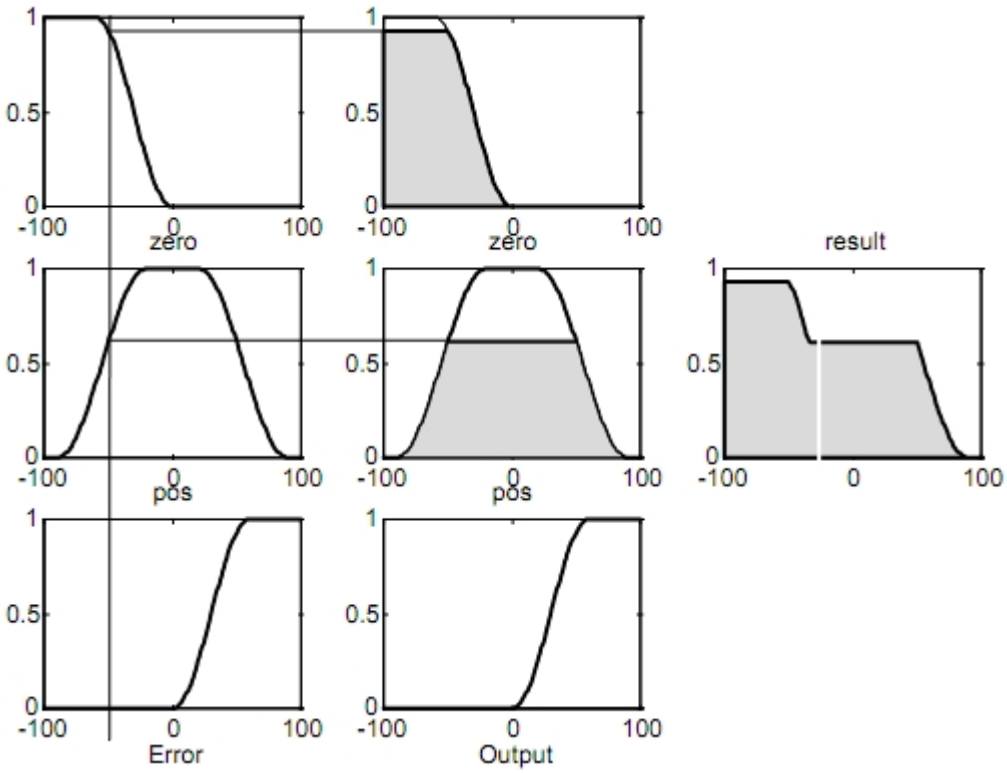


**Figure 2.6: 1 input, 1 Output Rule Base with Non-Singleton Output Sets**

A practical illustration of the operation of a fuzzy system is then given in Figure 2.6, for a multiple-input single-output fuzzy system with two inputs, e1 and e2, and one output, u. For each input or output, two fuzzy sets are shown, through usually there are more. e1, e2 and u are numerical variables associated with linguistic variables such as speed and torque, etc. ZE, PS, and PL are linguistic values of the linguistic variables. Given the values for e1 and e2 a shown, Singleton fuzzification process maps them to associated fuzzy sets with membership values: e1 is mapped into the fuzzy sets representing ZE with a membership value of 0.75, and mapped into the fuzzy sets representing PS with a membership value of 0.25; e2 is mapped into the fuzzy set representing PS with a membership value of 0.5. Then the following rules (assumed exist in the rule base) fire to find the output fuzzy sets that contains the output:

*If e1 is ZE and e2 is PS, then u is PS;*
*If e1 is PS and e2 is PS, then u is PL,*

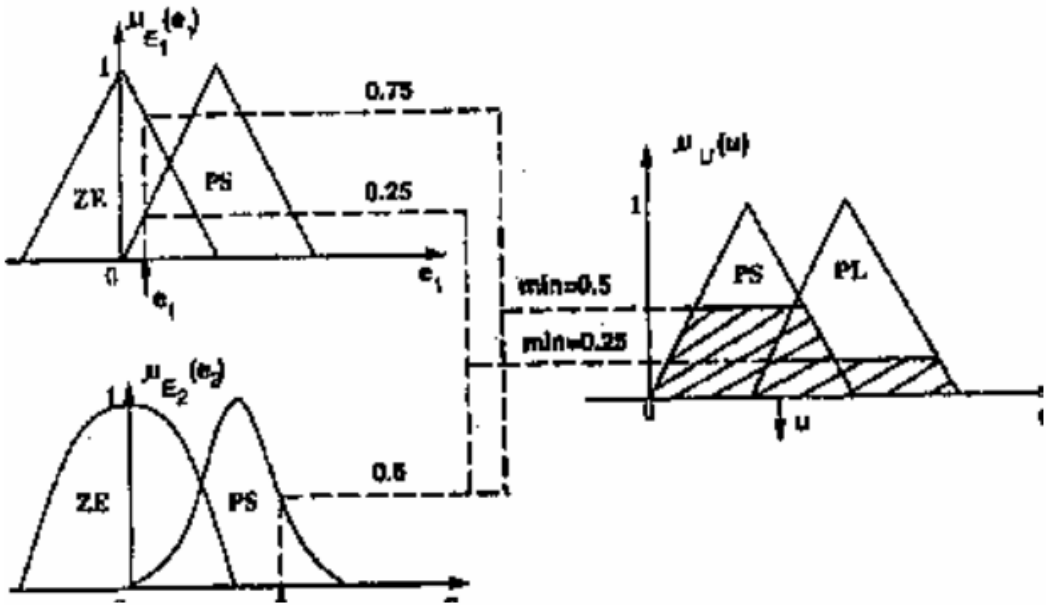


**Figure 2.7: A Practical Illustration**

By using Sup-Min inference method for both the premises and the fuzzy implication, as illustrated, and by using centre of gravity defuzzification method on the shaded area, the desired output value is then found.

#### 2.4.3.5 Table Based Controller

If the universes are discrete, it is always possible to calculate all thinkable combinations of inputs before putting the controller into operation. In a table based controller the relation between all input combinations and their corresponding outputs are arranged in a table. With two inputs and one output, the table is a two dimensional look-up table. With three inputs the table becomes a three-dimensional array. The array implementation improves execution speed, as the run-time inference is reduced to a table look-up which is a lot faster, at least when the correct entry can be found without too much searching. Below is an example of table based controller:

**Table 2.4: Fuzzy Control Rules for Speed and Current Controllers**

| Δe \ e | NB | NM | NS | ZE | PS | PM | PB |
|---|---|---|---|---|---|---|---|
| NB | NVB | NVB | NVB | NB | NM | NS | ZE |
| NM | NVB | NVB | NB | NM | NS | ZE | PS |
| NS | NVB | NB | NM | NS | ZE | PS | PM |
| ZE | NB | NM | NS | ZE | PS | PM | PB |
| PS | NM | NS | ZE | PS | PM | PB | PVB |
| PM | NS | ZE | PS | PM | PB | PVB | PVB |
| PB | ZE | PS | PM | PB | PVB | PVB | PVB |

A typical application area for the table based controller is where the inputs to the controller are the error and the change in error.

## 2.5 Adaptive Fuzzy Logic Controller

There are many kinds of adaptive Fuzzy Logic Controller that are being implemented in the control system in industrial and medical field as well. It is important to select a suitable adaptive Fuzzy Logic Controller which is easier to be learned and

implemented and at the same time bring the performance of the control system to an optimum operating condition.

The most important task in adaptive fuzzy control engineering is to build advanced tools for automated knowledge-base generation and tuning fuzzy controllers. Moreover, an improved approximate reasoning mechanism to speed up the on-line controller response is required. The tools for auto-generation of the knowledge-base will decrease the cost and time of fuzzy controller application developments. The tools for tuning the controller knowledge will provide the stability requirements for the operation of the controller.

The conventional method to optimize the FLC is the steepest cell descent. Nevertheless, this method required some prerequisites, such as fixing the number of rules, which has been obtained through trials of success and failure, been the method tedious and bored because it requires a lot of time. The Genetic Algorithm (GA) and neural network driven by fuzzy reasoning are the advanced methods for learning the FLC systems. The following paragraphs describe these optimization methods.

### 2.5.1 Genetic Algorithm

The GA is one of the most up-to-date artificial intelligence techniques. GA has been applied successfully in many engineering applications and optimization problems. The GA is an optimization method developed by biological evolution, which was used to determine the shapes and places of membership functions, getting the inference rules and output memberships. They presume that the potential solution of a problem is an individual and can be represented by a set of parameters. These parameters are regarded as the gens of chromosome and can be structured by a string of values in binary form or real form. Generally, a positive value know as fitness value, is used to reflect the degree of goodness of the chromosome for solving the problem.

A genetic algorithm mimics the evolution of populations. First, different possible solutions to a problem are generated. They are tested for their performance, that is, how good a solution they provide. A fraction of the good solutions is selected, and the others are eliminated. Then the selected solutions undergo the processes of reproduction, crossover and mutation to create a new generation of possible solutions, which is expected to perform better than the previous generation. Finally, production and evaluation of new generations is repeated until convergence. Such an algorithm searches for a solution from a broad spectrum of possible solutions, rather than where the results would normally be expected. The penalty is computational intensity. The elements of a genetic algorithm are explained next.

1. Encoding. The parameter set of the problem is encoded into a bit string representation. For instance, a point (x,y)=(11,6), can be represented as a chromosome, which is a concatenated bit string


**Figure 2.8: 8 Bit Binary String**

Each coordinate value is a gene of four bits. Other encoding schemes can be used, and arrangements can be made for encoding negative and floating point numbers.

2. Fitness evaluation. After creating a population the fitness value of each member is calculated. For a maximization problem, the fitness value of the *i*th member is the value of the objective function at point *i*. Normally, strictly positive objective functions are employed. Another possible fitness measure is to use a ranking of the members of the population so the objective function can be inaccurate as long as it provides the correct ranking.

3. Selection. The algorithm selects which parents should participate in producing off-springs for the next generation. Usually the probability of selection for a member is proportional to its fitness value. The idea is to let

members with above-average fitness reproduce and replace members with below-average fitness.

4. Crossover. Crossover operators generate new chromosomes that hopefully retain good features from the previous generation. Crossover is usually applied to selected pairs of parents with a probability equal to a given crossover rate. In one-point crossover a crossover point on the genetic code is selected at random and two parent chromosomes interchange their bit strings to the right of this point. Take for example two chromosomes

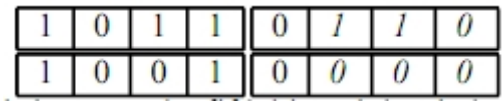


**Figure 2.9: Crossover with 8 Bit Binary String**

If the crossover point is between the fifth bit and the sixth, the digits written in italics will swap places vertically. The two new chromosomes will be

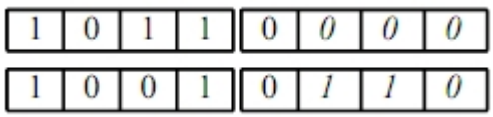


**Figure 2.10: Crossover with 8 Bit Binary String**

In two-point crossover, two crossover points are selected and the part of the chromosome string between these two points is swapped to generate two children and so on. In effect, parents pass segments of their own chromosomes on to their children, and some children will be able to outperform their parents if they get good genes from both parents.

5. Mutation. A mutation operator can spontaneously create new chromosomes. The most common way is to flip a bit with a probability equal to a very low, given mutation rate. The mutation prevents the population from converging

towards a local minimum. The mutation rate is low in order to preserve good chromosomes.

The flow chart of a simple GA is shown in Figure 2.10. There are three genetic operators used to generate and explore the population and select new generations. These operators are selection, crossover and mutation.

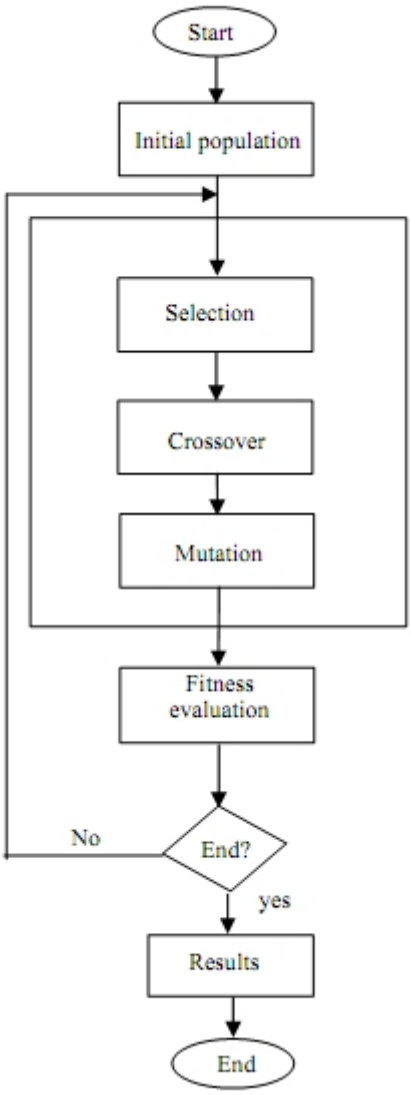


**Figure 2.11: General Flow Chart for GA**

### 2.5.2 Fuzzy-Neural Control System

#### 2.5.2.1 Artificial-Neural Network (ANN)

In the year of 1943, it is often considered the initial year for the development of artificial neural network (ANN). McCulloch and Pitts outlined the first formal model of an elementary computing neuron. The model includes all necessary elements to perform logic operations, and thus it contained function as an arithmetic-logic computing element. ANN consists of many interconnected neurons, nodes or processing elements arranged in layers. The purpose of the network is to map some input vector to an output vector. The mapping may be linear or non-linear, generally it is nonlinear.

ANN can be classified into two main categories based on their connection structures, which are feedforward and feedback respectively. It is trained, not programmed such as the rule-based FLC, to learn by example from the sets of input-output training data fed into it. The ANN results from the interconnection of artificial neurons via weights that act like gains. A non-linear activation function contributes to the non-linear transfer characteristics of a neuron, which permits non-linear input-output mapping in a neural network (NN). The learning (or training) process of the network has been illustrated simply in Figure 2.11, where the training algorithm adjusts the values of the weights until the network output matches the target.

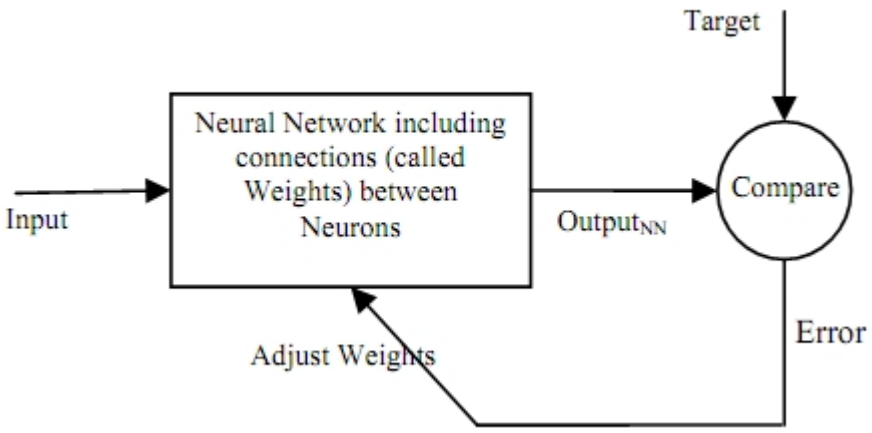


**Figure 2.12: Simplified Schematic of the ANN Training Process**

#### 2.5.2.2 Adaptive Neurofuzzy Inference System (ANFIS)

ANFIS is an architecture which is functionally equivalent to a Sugeno type fuzzy rule base. Under certain minor constraints the ANFIS architecture is also equivalent to a radial basis function network. Loosely speaking ANFIS is a method for tuning an existing rulebase with a learning algorithm based on a collection of training data. This allows the rulebase to adapt. The requirements for the radial basis function network to be equivalent to a fuzzy rule bases is summarized in the following:

- Both must use the same aggregation method (weighted average or weighted sum) to derive their overall outputs.
- The number of activation functions must be equal to the number of fuzzy if-then rules.
- When there are several inputs in the rule base, each activation function must be equal to a composite input membership function. One way to achieve this is to employ Gaussian membership functions with the same variance in the rule base, and apply product for the DQG operation. The multiplication of the Gaussian membership functions becomes a multi-dimensional Gaussian radial basis function.
- Corresponding activation functions and fuzzy rules should have the same functions on the output side of the neurons and rules respectively.

If the training data are contained in a small region of the input space, the centre of the neurons in the hidden layer can be concentrated within the region and sparsely cover the remaining area. Thus only a local model will be formed and if the test data lie outside the region, the performance of the network will be poor. On the other hand, if one distributes the basis function centre evenly throughout the input space, the number of neurons depends exponentially on the dimension of the input space.

### 2.5.2.3 ANFIS Architecture

The fuzzy neural controller structure used in represented by a neural network consisting of five layers. Figure 2.12 shows an example of the network structure for a controller with two inputs and a single output. The construction of the five-layer network is described next.

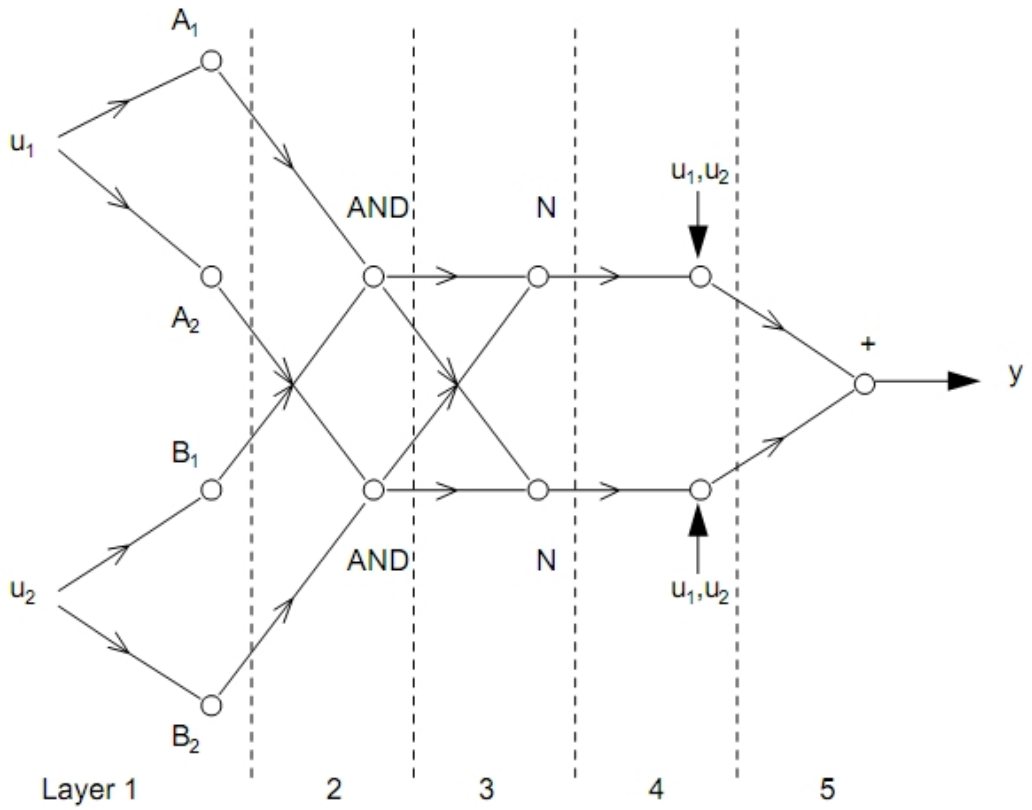


**Figure 2.13: Structure of the ANFIS Network**

The first layer is an input layer with one node for each controller input variable. The nodes in this layer act as single-input, multi output, 'fan-out' nodes distributing each input variable to each of its associated membership function nodes in the second layer. The interconnection weights between the first and second layers are all unity and constant. The second layer is made up of nodes representing Gaussian membership functions. The total number of nodes in this is equal to the total number of fuzzy sets associated with the input variables. The interconnection weights between the second and third layers are all unity and constant. The third

layer is made up of nodes implementing the fuzzy intersection form of the fuzzy AND operator.

The interconnection weights between the third and fourth layers are also all unity and constant. The fourth layer is made up of nodes implementing the bounded sum form of the fuzzy OR operator. The number of nodes is equal of nodes is equal to the total number of fuzzy sets associated with the controller output variables. The fifth and final layer comprises nodes implementing a center-of-gravity defuzzification algorithm, with one node for each output variable. The weights of the interconnections between the nodes in the fourth and fifth layers are the products of the center and width of the membership function associated with the fuzzy set for each layer four-node output variable.

# CHAPTER 3

# METHODOLOGY

## 3.1 Introduction

Generally, the controlling of the position of the mechanical arm is done by the microcontroller of the control system. The microcontroller does not act independently where it needs to work together with other vital components in order to control the position of the DC motor. The other components include hardware such as other electronic components, DC motor driver, encoder, RS232 communication and so on and software such as Matlab, MPLAB IDE and HITECH C. The main thing is the controller algorithms are to be developed in the form of C programming language in MPLAB IDE and being compiled into hex file using HITECH C. The compiled C code is then programmed into the microcontroller to be implemented in controlling the position of the DC motor. Before this, simulation of the controllers is made using certain software such as Matlab to see the simulated performance of the controller. It is an essential step to implement suitable controllers in controlling the position of the DC motor.

Before that, the mechanical arm could only raise up to 120 degree and if the movement of the arm is to exceed this angle, the movement of the arm will be very sluggish and not smooth at all. Now, the improved PID and Fuzzy Logic controller is being developed to overcome such problems and at the same time to optimize the controlling of the mechanical arm. The methodology of developing the improved PID and Fuzzy Logic controller is being discussed in this chapter.

## 3.2 Closed-Loop Control System

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 toward 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.

In developing and designing the improved PID and Fuzzy Logic controller for the mechanical arm, closed–loop control system is being applied instead of open-loop control system since the output, which is position of the arm will tend to oscillate and feedback signal is required to compare with desired input angle so that error can be generated and being send to controller. Next, the controller will process the error and send out correction signal to correct the position of the arm. Basically, controlling of the position of the mechanical arm is the controlling of the position of the DC motor.

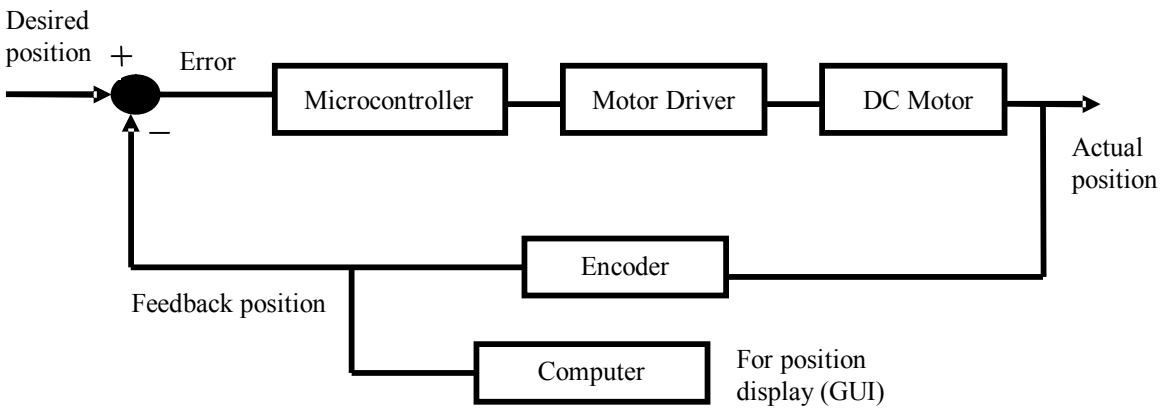


**Figure 3.1: Block Diagram of DC Motor Position Control System**

The block diagram of the control system is shown in Figure 3.1. It is a closed-loop control system with real time monitoring. As shown in the figure, the microcontroller acts like the brain of the DC motor position control system. It is used

as the controller to control the position and speed of the DC motor to be at the desired level.

The set-point value, or desired position (angle) is being input to the controller by the user and the control signal is being sent to the motor driver to drive the DC motor to the desired position. The output, which is the actual position (angle) of the DC motor will be measured by the sensor, which is the encoder. The feedback position is then sent to the summing point to be compared with the desired position. At the summing point, the difference between the desired angle and actual angle will be calculated and it serves as an error signal. The error signal will enter the microcontroller to be processed. Next, the microcontroller will decide which action to be taken to correct the error and bring the arm to desired position. In other words, the microcontroller will send correction signal to actuate the DC motor driver as well as the DC motor to bring the arm to desired position. The output of the controller will be the variation of pulse width modulation (PWM) to either accelerate or decelerate the speed of motor to reach desired position. At the same time, the control system also links to graphic user interface (GUI) in the user's computer for the purpose of real-time monitoring.

## 3.3 Simulation of Fuzzy Logic Controller Using Matlab

Before proceed to write the C code of Fuzzy Logic controller in MPLAB, simulation for Fuzzy Logic Controller is done in Matlab using Fuzzy Logic Toolbox. It is an essential step as it is desired to obtain optimized membership function for the Fuzzy Logic Controller in controlling the position of the DC motor. First, the number of input and output of the controller is determined. Next, the number of membership functions of the input and output is determined and the range for each input and output membership functions is set as well. After that, rules of the Fuzzy Logic Controller will be set and the simulation can be run to see the results. The simulation is to be run till the optimized membership function is obtained.

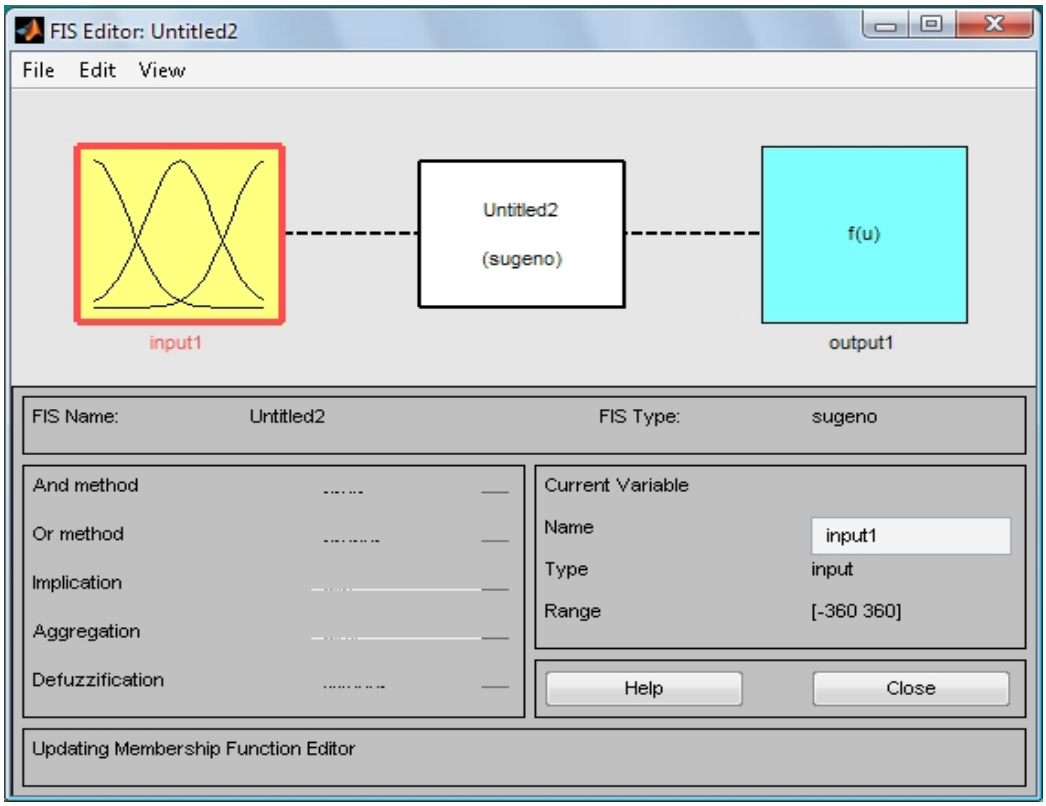


**Figure 3.2: Number of Input and Output is Set**

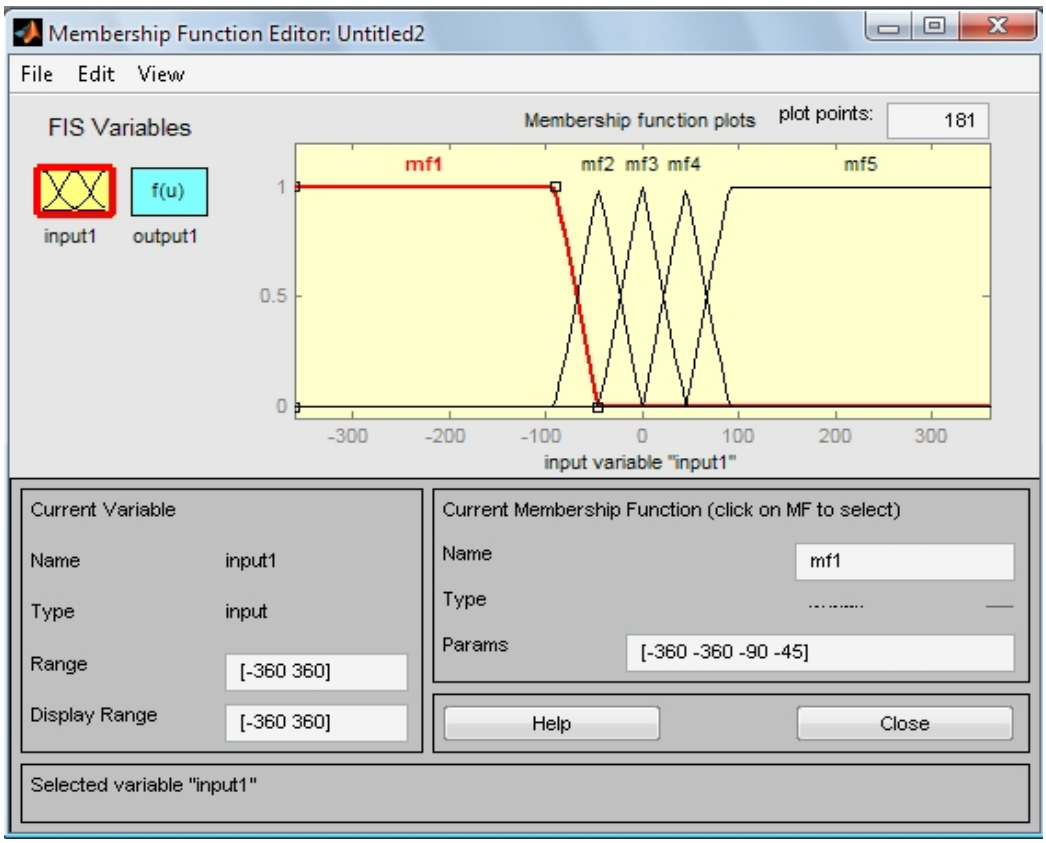


**Figure 3.3: Constructed Input Membership Functions**

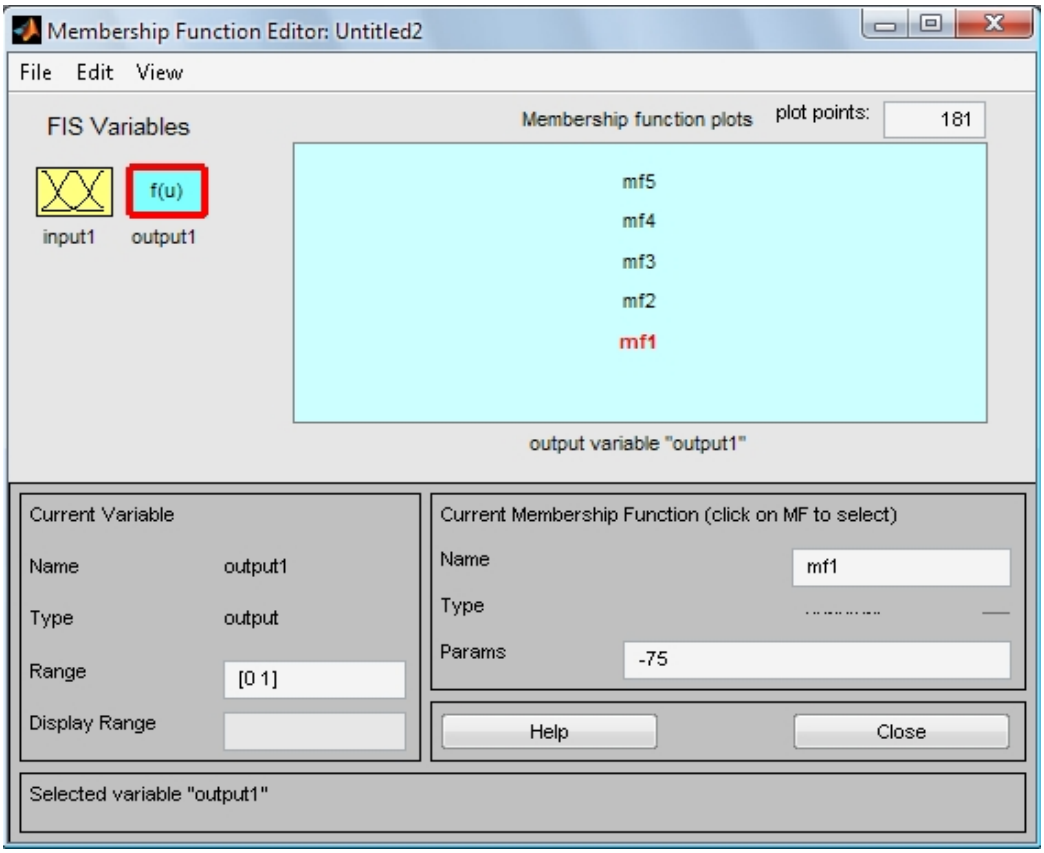


**Figure 3.4: Constructed Output Membership Functions**

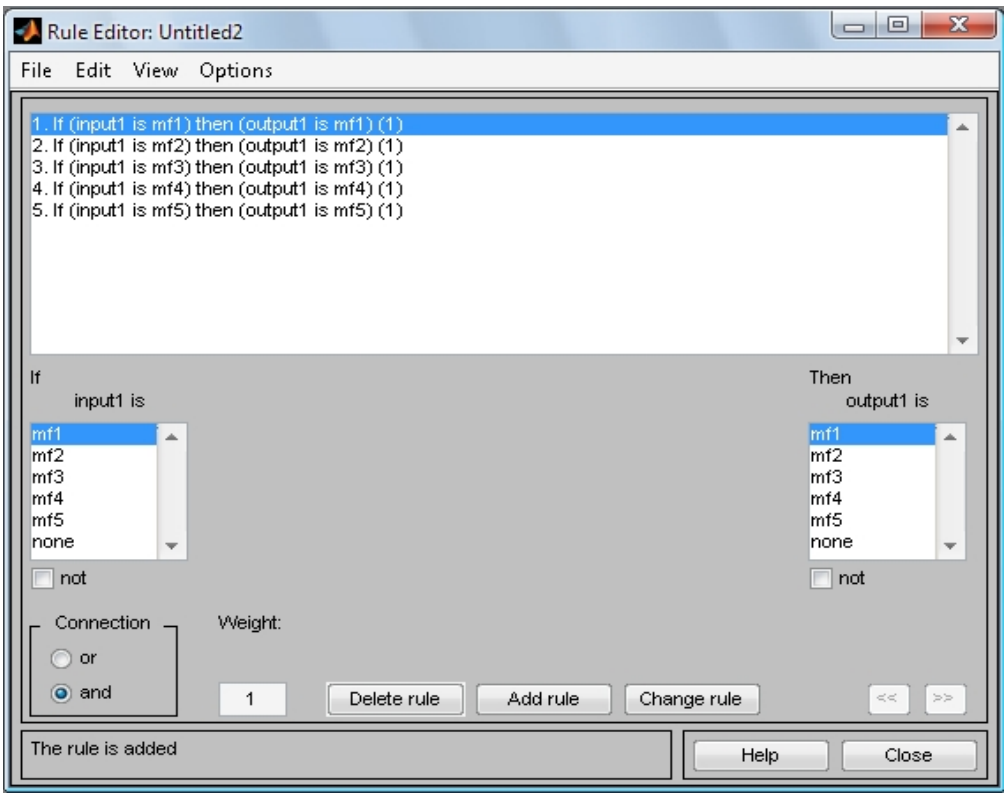


**Figure 3.5: Rule Base of Fuzzy Logic Controller**

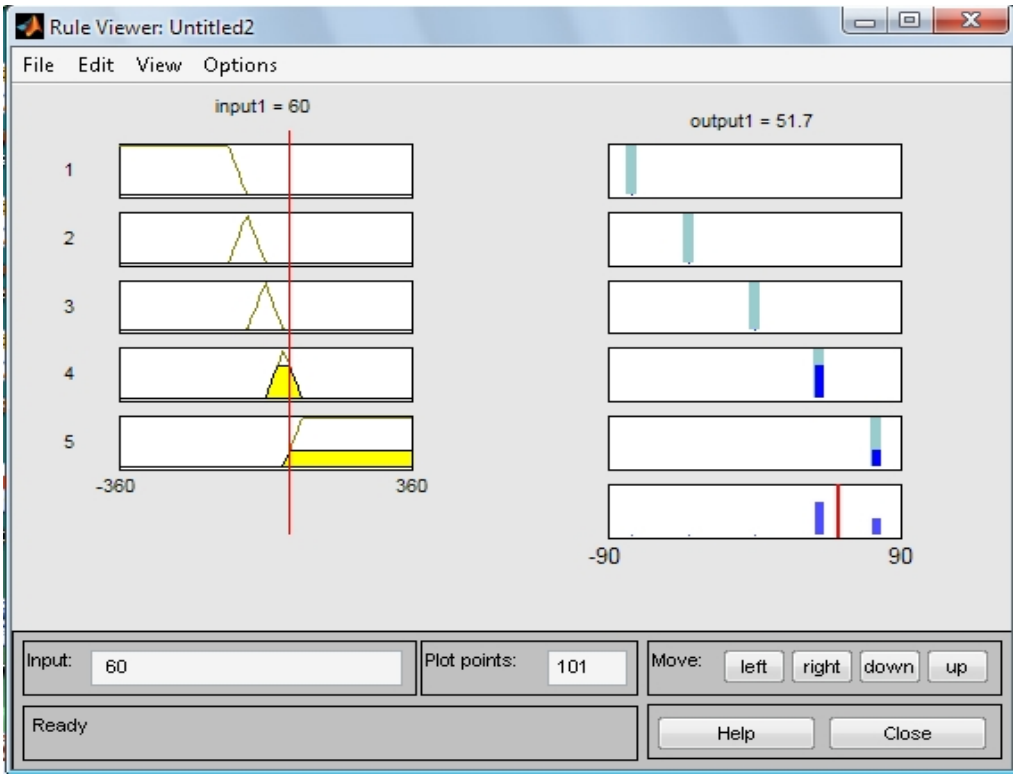


**Figure 3.6: Simulation of Fuzzy Logic Controller**

In controlling the position of the DC motor, there will be 1 controller input and 1 controller output. There are 5 membership functions for both input and output. The membership functions of the input are in triangular form while the membership functions of the output are in the form of singleton. After that, the Fuzzy Logic controller algorithm is being developed in the form of mathematical representation before being written out in C language in MPLAB.

## 3.4 C Language Programming Using MPLAB IDE

MPLAB Integrated Development Environment (IDE) is a free, integrated toolset for the development of embedded applications employing Microchip's PIC® and dsPIC® microcontrollers. MPLAB IDE runs as a 32-bit application on MS Windows®, is easy to use and includes a host of free software components for fast application development and super-charged debugging. MPLAB IDE also serves as a single,

unified graphical user interface for additional Microchip and third party software and hardware development tools. Moving between tools is a snap, and upgrading from the free software simulator to hardware debug and programming tools is done in a flash because MPLAB IDE has the same user interface for all tools.

The PID and Fuzzy Logic Controller algorithm that used to control the position of the mechanical arm are to be developed in the form of C programming language. To generate the controller algorithms in C language programming, MPLAB is being used for this purpose. The controlling methods are first studied and then being written out in C code before being programmed into PIC18F2550.

## 3.5 C Language Compiling Using HITECH C

After the controller C algorithm has been generated in MPLAB, next will be the compiling of the C algorithms into hex file before programmed into microcontroller, PIC18F2550. This task is being done by using HITECH C compiler.

HI-TECH C is a world class brand of compilers featuring Omniscient Code Generation™, whole-program compilation technology, for Microchip Technology's 8-, 16-, and 32-bit PIC® microcontroller and dsPIC® digital signal controller architectures.

## 3.6 Graphic User Interface (GUI) Built Using Visual Basic

In order to monitor the performance of the controllers in real time manner, a GUI is developed using Visual Basic 6.0. Within the GUI, types of controller to be applied can be select, the desired angle of the mechanical arm can be input, the tuning gain constant can be input, error can be obtained and the response graph of the controller can be observed.

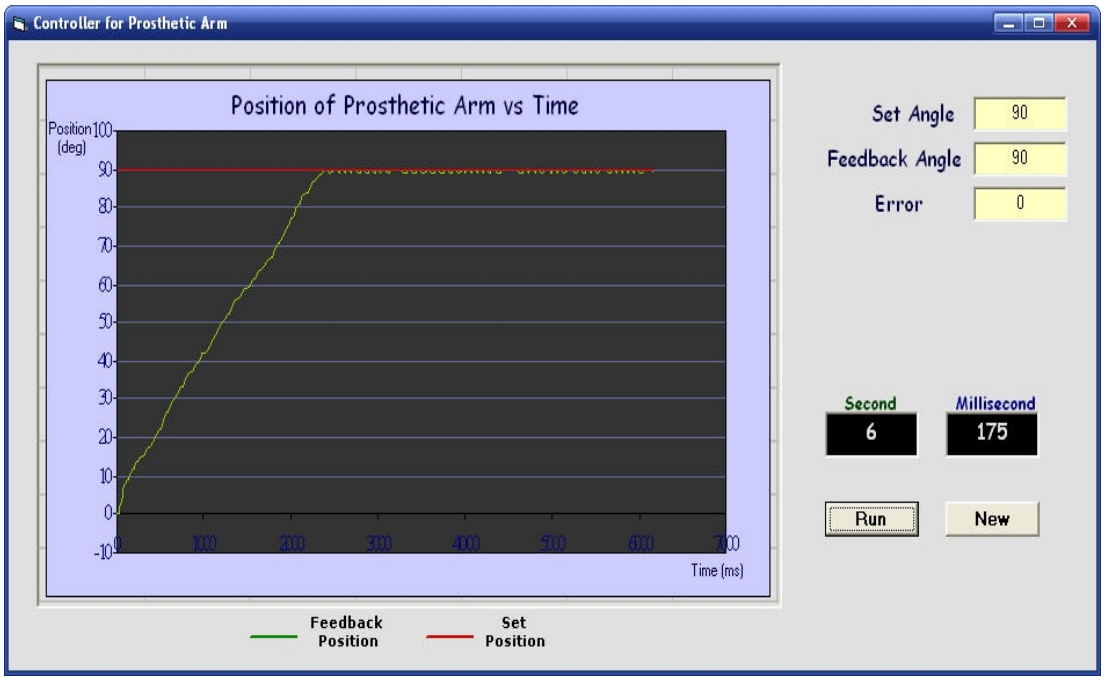


**Figure 3.7: GUI built using Visual Basic**

## 3.7 Hardware Implementation

Although the hardware part, such as circuit has been designed by earlier batch of students, but during this project, the previous hardware included circuit is being used back in this project. Basically, same hardware implementation will be used but still a good understanding towards the hardware built is required in order to make the works smooth during doing the project.

A controller board that is developed earlier consists of many electronic parts. It is used to apply the PID and Fuzzy Logic Controller in controlling the position of the DC motor and consists of microcontroller (PIC18F2550), RS232 communication, DC motor driver, data acquisition and encoder.

### 3.7.1 Microcontroller

The microcontroller is a very important component in the control system and it acts like the brain of the DC motor position control system. It receive the feedback of the actual value and generate error and do the automatically tuning of either gain constant or membership functions, depend on the controller mode selected. The microcontroller chip PIC18F2550 has been selected for the purpose of controlling the position of the DC motor is manufactured by Microchip. It has the most of the motor control elements that are embedded as build in modules. Other build in modules used in this application include PWM signal generator and ADC module.

### 3.7.2 RS232 Communication

The communication system is a vital element for a position control system. The RS232 protocol is applied on the embedded USART module. The RS232 protocol is a commonly used communication method for microcontroller – computer interface. This protocol is easy to use, and most microcontrollers nowadays support this type of communication. Level converter MAX232 is used in this communication system to provide a proper voltage level for microcontroller – computer communication.

### 3.7.3 PWM Signal Generator

Pulse width modulation (PWM) is a method to control power by adjusting PWM duty cycle. Here, letter “x” is used to designate PWM channel. Referring to the PIC datasheet, the duty cycle is adjusted by changing 10-Bit duty cycle registers represented by CCPRxL:CCPxCON<5:4>. A proper relationship between the registers and the effective PWM duty cycle output is determined by:

PWM_duty_cycle = (CCPRxL:CCPxCON<5:4>)*TOSC* (TMR2_prescale_value)

Where the Tosc is 1/(oscillator frequency), and the TMR2_prescale_value is a constant pre-scale value set in TMR2 pre-scale register. The PWM frequency is calculated as follows:

PWM_freq = 1/PWM_Period,
PWM_Period = [(PR2) + 1]*4*(TMR2_prescale_value)

### 3.7.4 Application of PWM and Encoder

Encoder serves as the sensor in this control system. It is important to have a sensor in the control system as it serves as it will receive the actual variable value in an analogue sense and converts it to a digital form and after that the data is being feedback to the microcontroller and by doing so error and rate of change or error can only be generated. It senses the angle and speed produce by the rotation of the DC motor and output it in the pulse signal, which is an electrical signal that can be communicate with the microcontroller. The signal is then delivered to the microcontroller to be processed. In other words, a closed loop microcontroller will regulate the power delivered to the motor to reach the required velocity. If the motor is to turn faster than the required velocity, the controller will deliver less power to the motor. Controlling the electrical power delivered to the motor, is usually done by Pulse Width Modulation.

In the closed-loop system used in this project, the microcontroller will constantly adjust the average power delivered to the motor to reach the required velocity and precisely calculate the position/angle of the motor's output shaft. From figure 3.2, the encoder will provide the microcontroller's internal counter with a sequence of pulses that correspond to the rotation of the motor. A timer is set to execute two software routine every 1/10 th of a second (which is just an arbitrary value). One of those software routines is to recalculate the actual angle of the shaft or the total number of revolutions. Another software routine is executed to control the speed of the motor by comparing the number of counted pulses with a fixed number which is referred to as the "required pulses". The "required pulses" corresponds to

the desired speed, and the "counted pulses" corresponds to the actual speed of the motor.

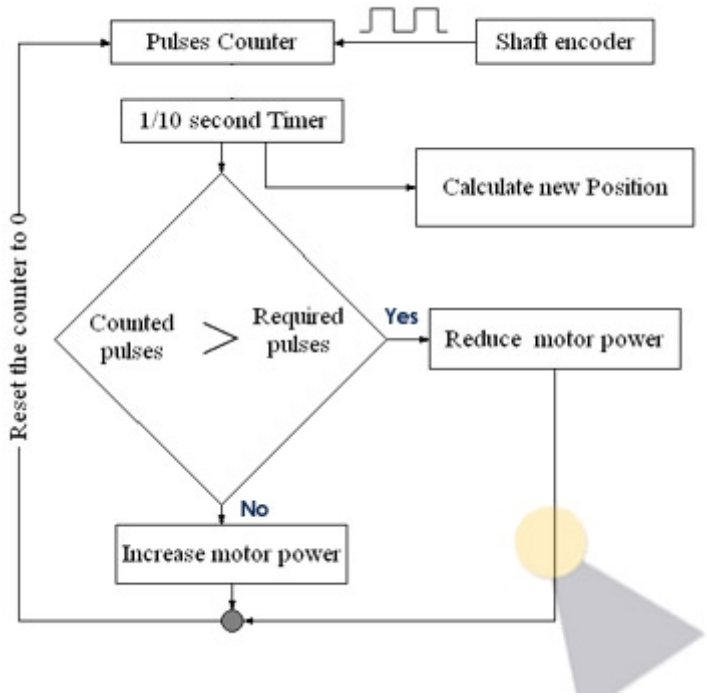


**Figure 3.8: Application of PWM**

Hence, in this project, the measured angle of the DC motor is being converted to pulse signal by encoder, which is in digital form. In order to obtain the difference between desired value and actual value which is error, when writing C programming language, a program that used to convert the pulse signal back to analogue data is required. The formula of converting the pulse signal into angle will be include in the programming code and the error could be generated within the microcontroller. Next, controller action can be done.

Below is the formula to convert pulse signal into angle:

angle = n_pulse/45000.0 * 360.0

where there is 45000 pulses detected in 1 revolution of the DC motor.

# CHAPTER 4

# RESULTS AND DISCUSSIONS

## 4.1 Simulation of Fuzzy Logic Controller

The simulation of Fuzzy Logic controller is done using Matlab. In order to obtain optimized membership functions, the input and output membership functions of the Fuzzy Logic controller is being tuned continuously base on the result of the simulation. At the same time, the simulation of the controller is being run continuously to obtain optimum performance. During the simulation, the input represents the error in terms of angle while the output represents the PWM to either accelerate or decelerate the speed of the motor.

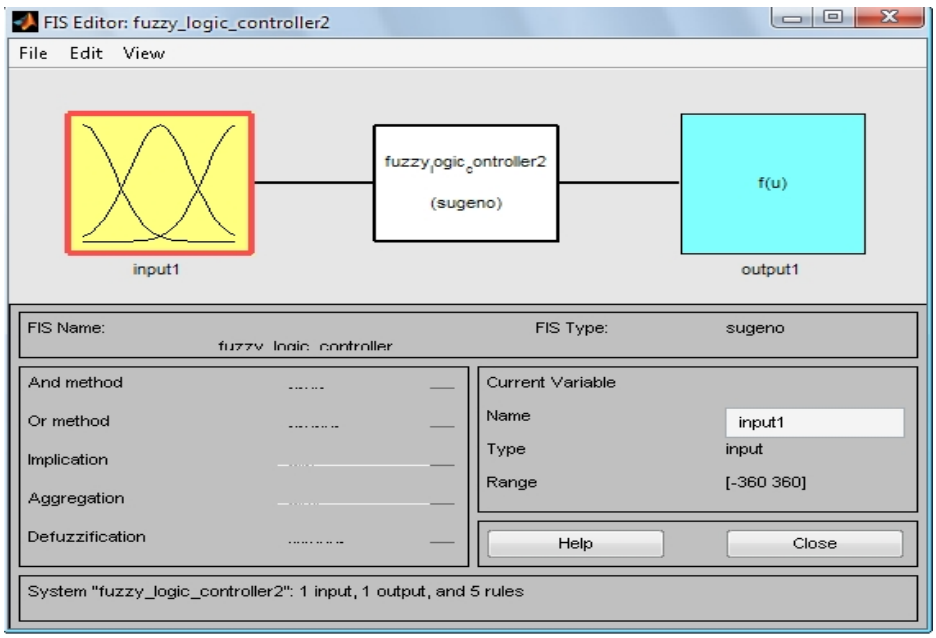


**Figure 4.1: Fuzzy Logic Controller Interface**

Figures below show the simulation of Fuzzy Logic controller:

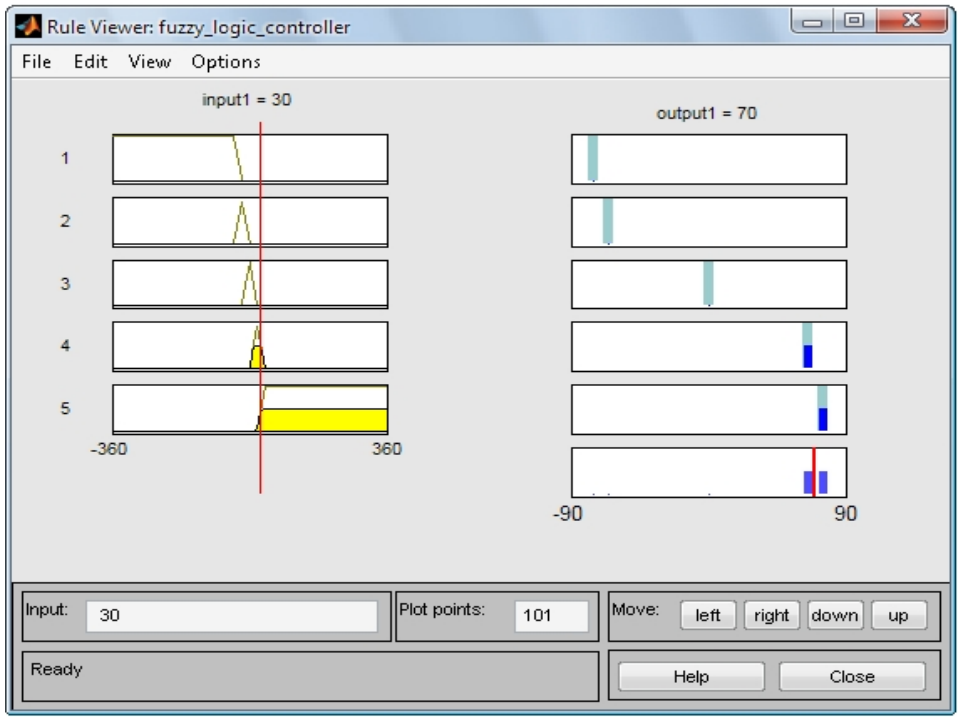


**Figure 4.2: Simulation of Fuzzy Logic Controller**

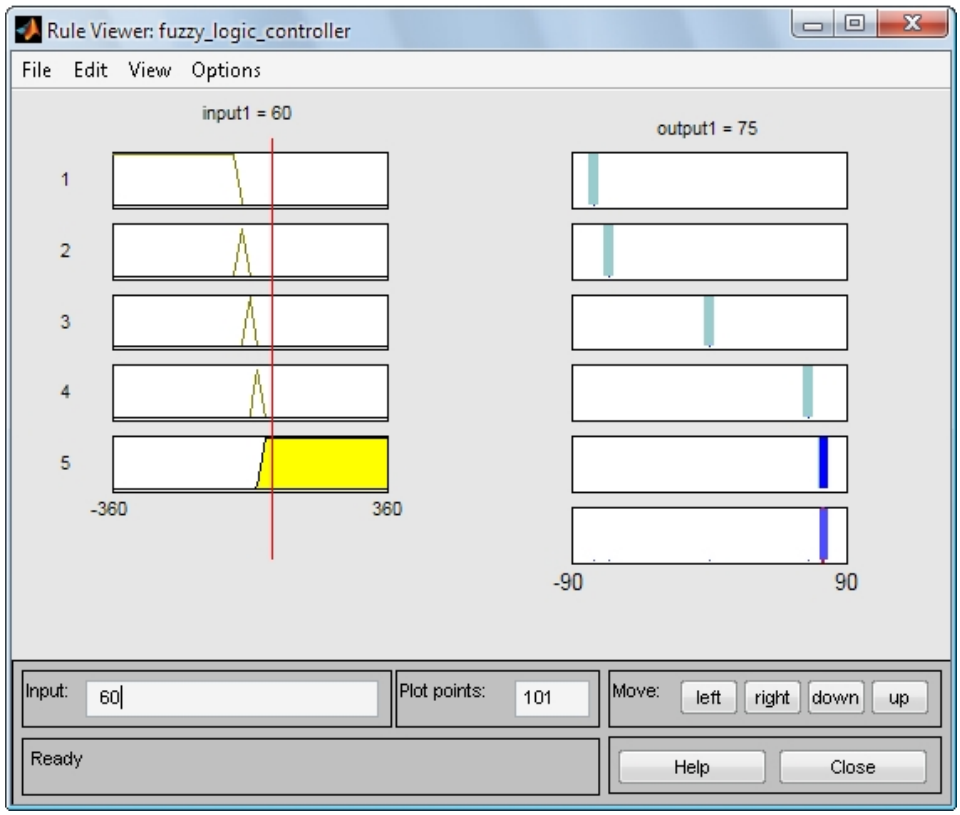


**Figure 4.3: Simulation of Fuzzy Logic Controller**

As indicated by the figure, during the simulation, different desired angles are being input into the controller and the corresponding PWM is obtained. From figure 4.2, the input angle is 30 and the corresponding PWM is 70 while from figure 4.3, the input angle is 60 and the corresponding PWM is 75. Base on the knowledge of the relationship between the magnitude of PWM and error, the optimum input and output membership functions can be obtained.

After running the simulation in Matlab, the optimized membership functions have been obtained. Below is the optimized input membership function:

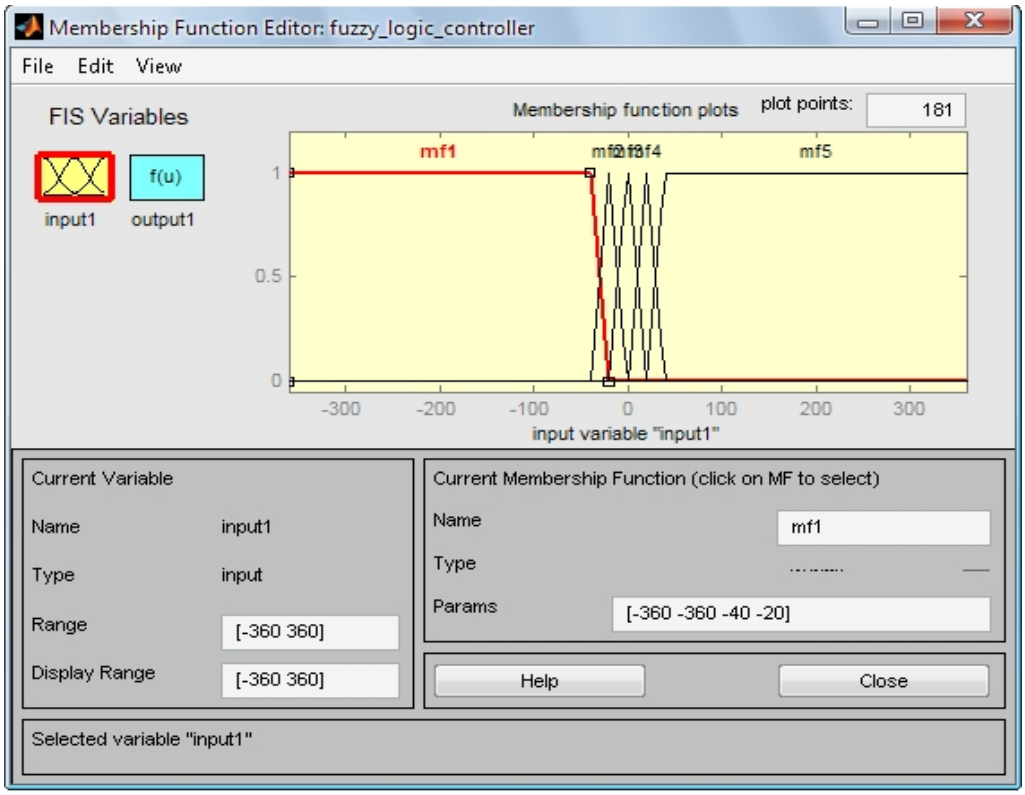


**Figure 4.4: Optimized Input Membership Function (Triangular)**

**Table 4.1: Range of Error for Various Input Conditions**

| Conditions | Range of Error |
|---|---|
| Negative Big (NB) | -360 < x < -20 |
| Negative Small (NS) | -40 < x < 0 |
| Zero (Z) | -20 < x < 20 |
| Positive Small (PS) | 0 < x < 40 |
| Positive Big (PB) | 20 < x < 360 |

Below is the optimized output membership function:

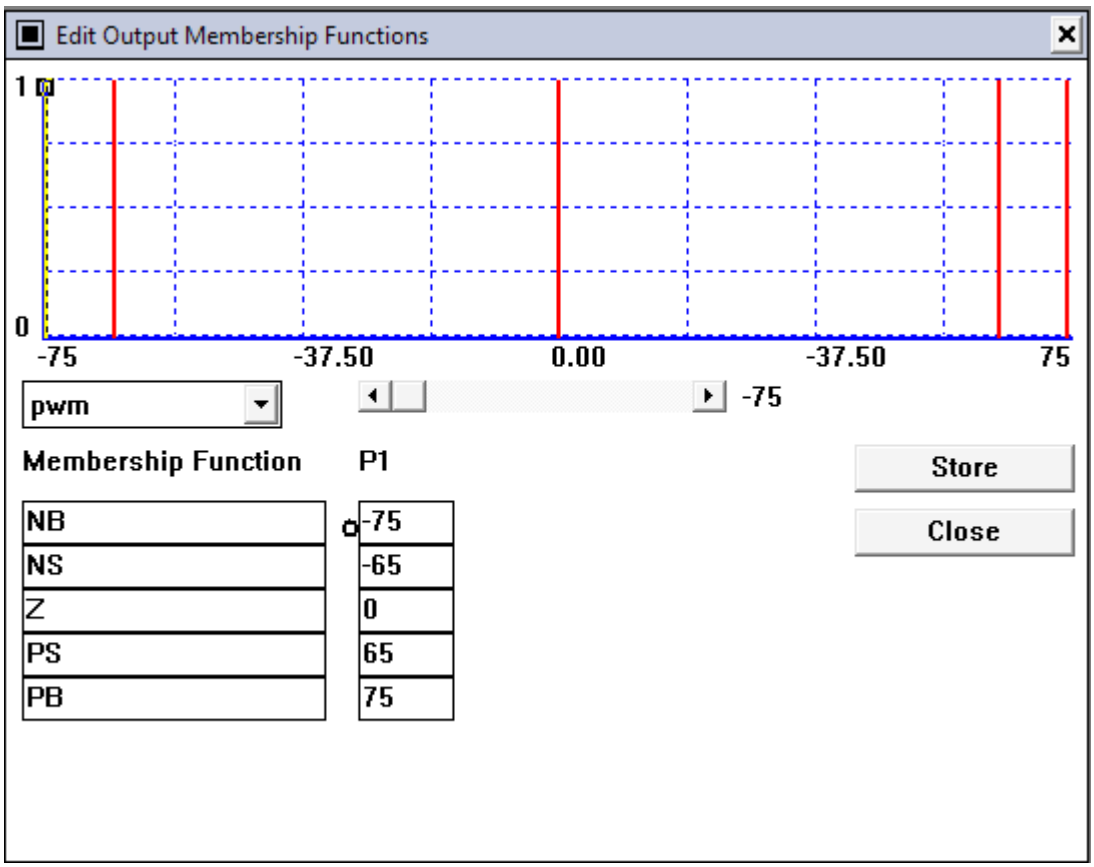


**Figure 4.5: Optimized Output Membership Functions (Singleton)**

**Table 4.2: PWM Values for Various Output Conditions**

| Conditions | Value of PWM |
|---|---|
| Negative Big (NB) | -75 |
| Negative Small (NS) | -65 |
| Zero (Z) | 0 |
| Positive Small (PS) | 65 |
| Positive Big (PB) | 75 |

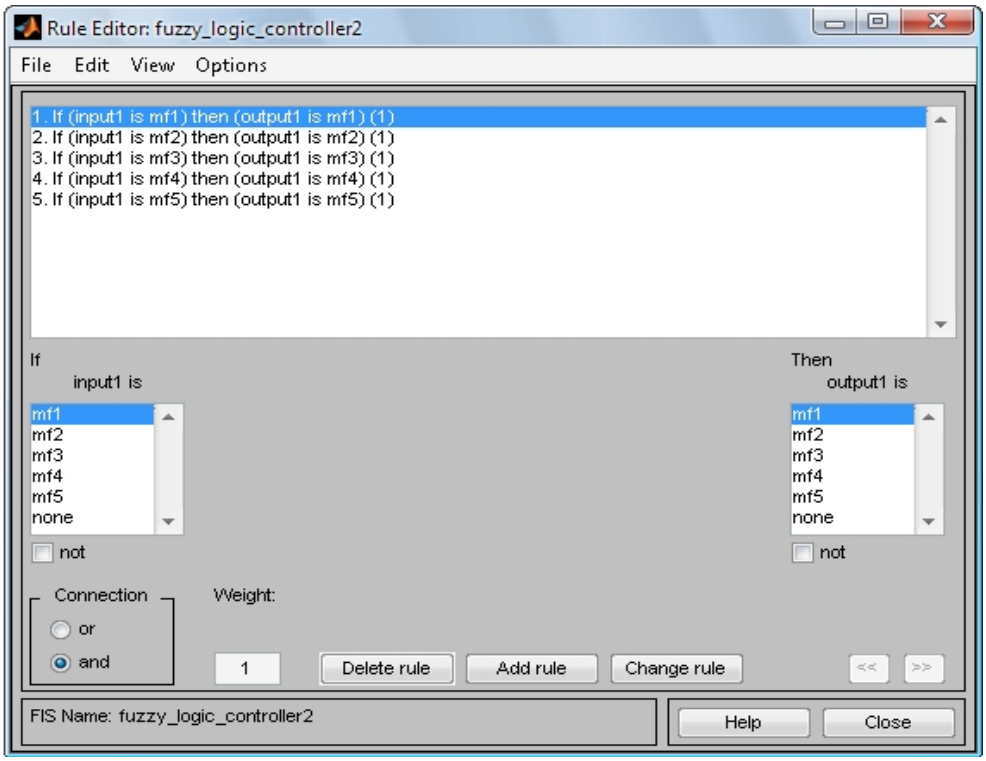


**Figure 4.6: Rule Base of the Fuzzy Logic Controller**

There are 5 rules in the rule base and the rules are to be applied in fuzzification process. Below is the linguistic rule base:

**Table 4.3: Rule Base (5 rules base)**

| Input (error) | Output (PWM) |
|---|---|
| Negative Big (NB) | Negative Big (NB) |
| Negative Small (NS) | Negative Small (NS) |
| Zero (Z) | Zero (Z) |
| Positive Small (PS) | Positive Small (PS) |
| Positive Big (PB) | Positive Big (PB) |

## 4.2 Mathematical Representation of Fuzzy Logic Controller

Before the algorithm of Fuzzy Logic controller is being developed in C programming language, the mathematical representation of the Fuzzy Logic controller is generated

based on the membership functions. The membership functions are in triangular form and they are linear as well, hence mathematical representation can be developed conveniently.

Below shows the mathematical representation of the Fuzzy Logic controller:

Case 1:

$X<-40$

$Z = -75$

Case 2:

$-40 \leq x<-20$

$a = x/20 + 2$

$b = -x/20 – 1$

$Z = a(-65) + b(-75)$

Case 3:

$-20 \leq x<0$

$a = x/45 + 1$

$b = -x/45$

$Z = a(0) + b(-65)$

Case 4:

$0 \leq x<20$

$a = -x/45 + 1$

$b = x/45$

$Z = a(0) + b(65)$

Case 5:

$20 \leq x<40$

$a = -x/45 + 2$

$b = x/45 – 1$

$Z = a(65) + b(75)$

Case 6:

x>90

Z = 75

## 4.3 Results and Performance of Fuzzy Logic Controller

Experiments are run to observe the response of the Fuzzy Logic controller. The transient response graph of the Fuzzy Logic controller is being displayed in the GUI developed by Visual Basic 6.0. Response graph of no load and loaded condition will be compared. The desired angle will be 90 degree. (moving from top)

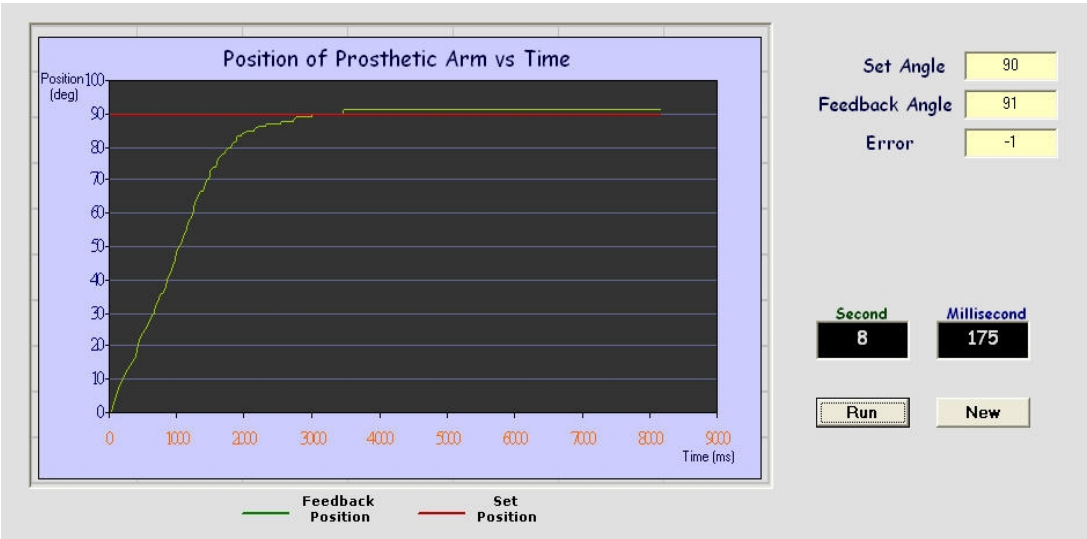


**Figure 4.7: Response Graph under No Load Condition**

From the graph above, we could obtain important information:

Steady state error: -1

%ss = (90-91)/90*100 = -1.11%

Rise time, $T_r$ = 1.8 – 0.2 = 1.6s

Settling time $T_s$: 2.75s

Overshoot: 0

Tolerance of error: +/-2 (+/-2.22%)

The condition shown in the graph is a critically damped control system. This shows that the response of the control system did not overshoot and reach the desired target in steady and stable manner. Since there is no transfer function of the control system, the calculated values is base on the estimation from the graph.

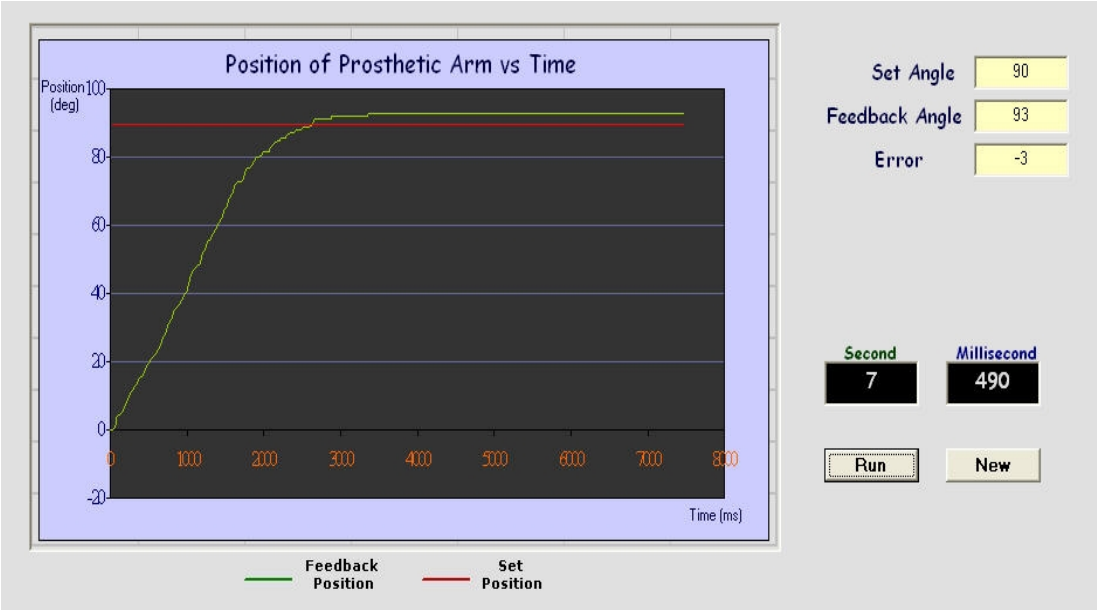


**Figure 4.8: Response Graph where the Arm under Loaded Condition (200g)**

From the graph above, we could obtain important information:

Steady state error: -3

%ss = (90-93)/90*100 = -3.33%

Rise time, $T_r$ = 2.1 – 0.3 = 1.8s

Settling time $T_s$: 2.8s

Overshoot: 0

Tolerance of error: +/-3 (+/-3.33%)

The condition shown in the graph is a critically damped control system. Since there is no transfer function of the control system, the calculated values is base on the estimation from the graph.

To compare the 2 conditions, it is obvious that loaded condition will have larger steady state error due to the additional weight that the arm is holding and make it tend to further slight down. Besides, the settling time and rise time of the loaded condition is slightly higher than the no load condition since additional weight of the arm make it slower to reach desired target.

## 4.4 Results and Performance of another Fuzzy Logic Controller

Here are the results obtained from another Fuzzy Logic controller for both no load and loaded conditions after running the experiments to see its performance.

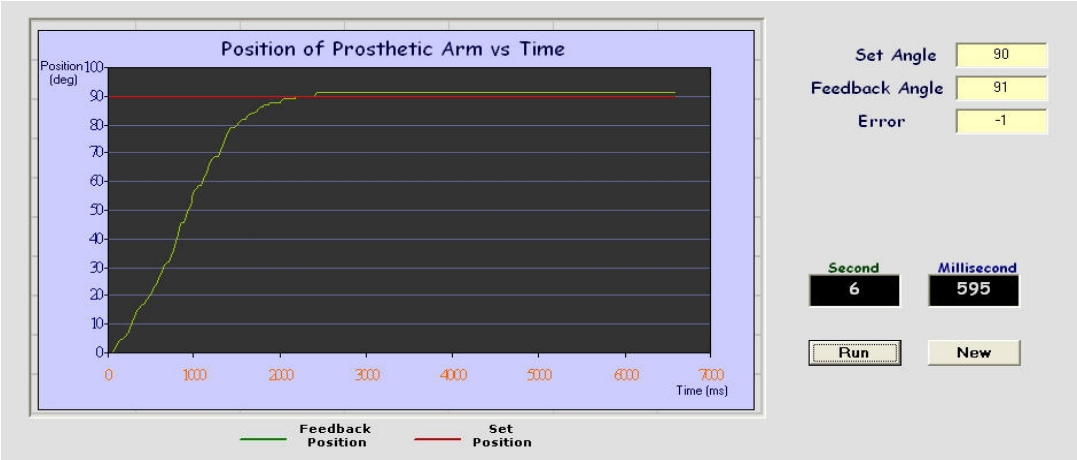


**Figure 4.9: Response Graph under No Load Condition**

From the graph above, we could obtain important information:

Steady state error: -1

%ss = (90-91)/90*100 = -1.11%

Rise time, $T_r$: 1.2s

Settling time $T_s$: 2.05s

Overshoot: 0

Tolerance of error: +/-2 (+/-2.22%)

Condition: Critically damped

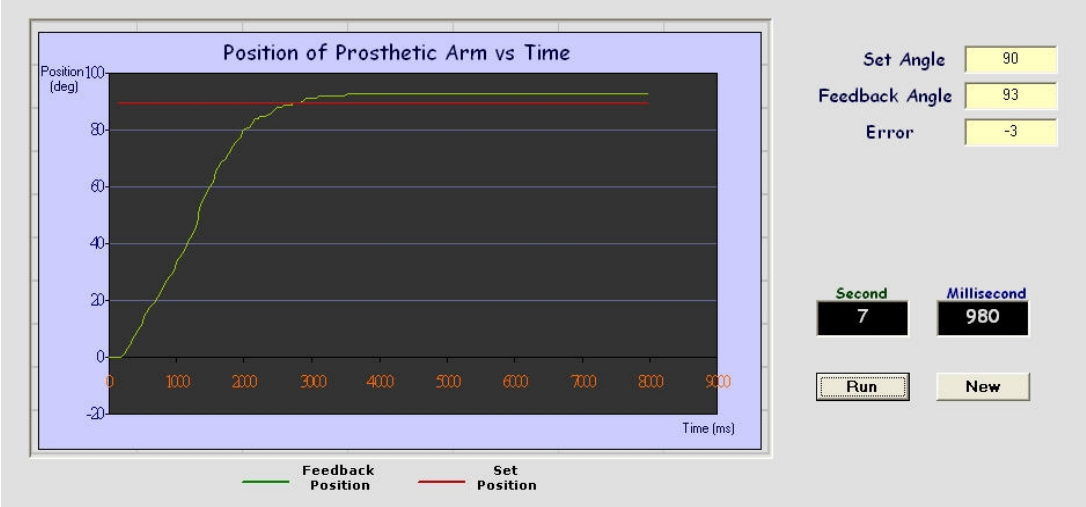


**Figure 4.10: Response Graph where the Arm under Loaded Condition (200g)**

From the graph above, we could obtain important information:

Steady state error: -3

%ss = (90-93)/90*100 = -3.33%

Rise time, $T_r$: 1.7s

Settling time $T_s$: 2.95s

Tolerance of error: +/-3 (+/-3.33%)

Condition: Critically damped

After comparing both graphs, it is observed that the rise time, settling time and steady state error for loaded condition are larger than no load condition due to the reasons explained in previous section.

## 4.5 Results and Performance of PID Controller

Experiments are run to obtain the response of the PID controller. The experiments in tuning the PID controller will be conducted with various combinations of gain constant. The PID controller is being tune manually to study the effect of each gain constant in order to obtain optimum response.

First, the PID controller is being tuned until the response graph is oscillating under no load condition. It is to be done by purely tuning the integral gain, $K_i$.

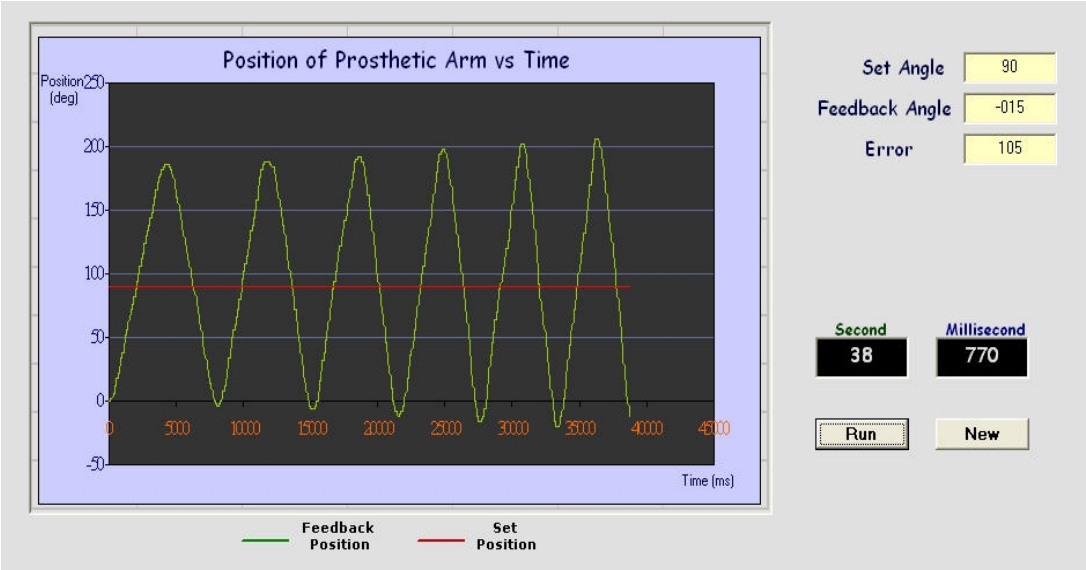


**Figure 4.11: Response Graph of PID Controller (Kp=0, Ki=0.01, Kd=0)**

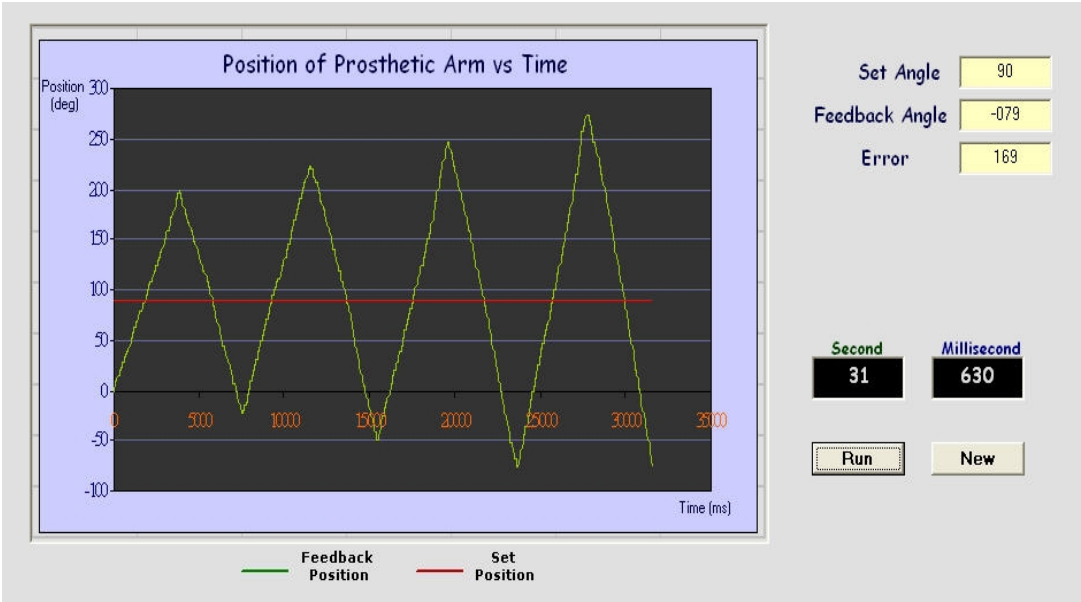


**Figure 4.12: Response Graph of PID Controller (Kp=0, Ki=0.1, Kd=0)**

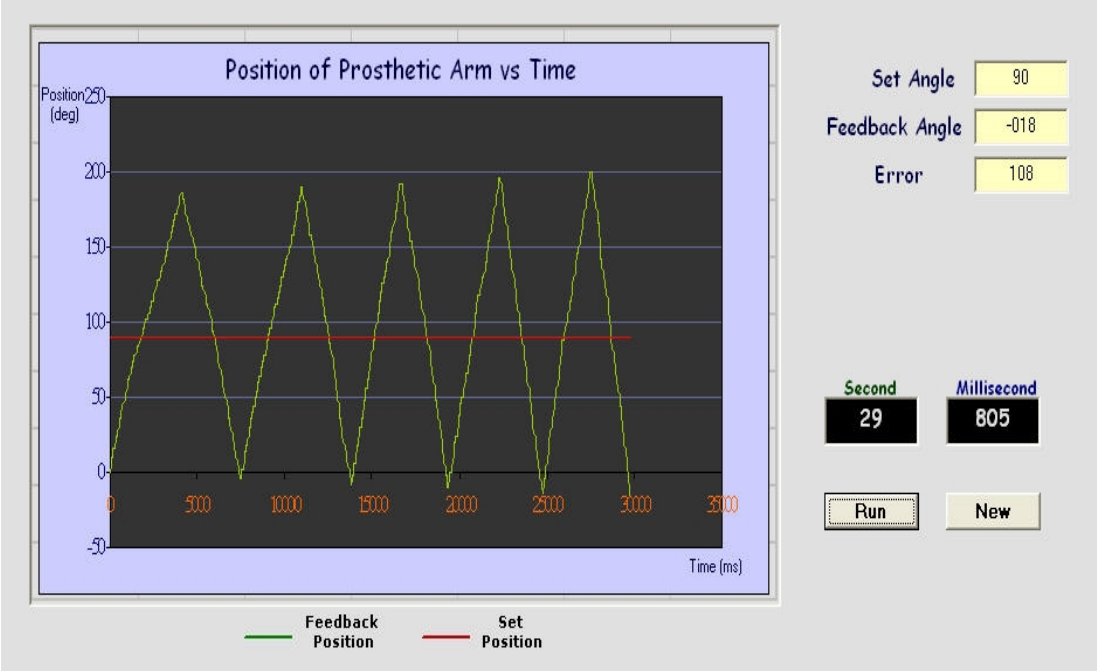


**Figure 4.13: Response Graph of PID Controller (Kp=0, Ki=0.5, Kd=0)**

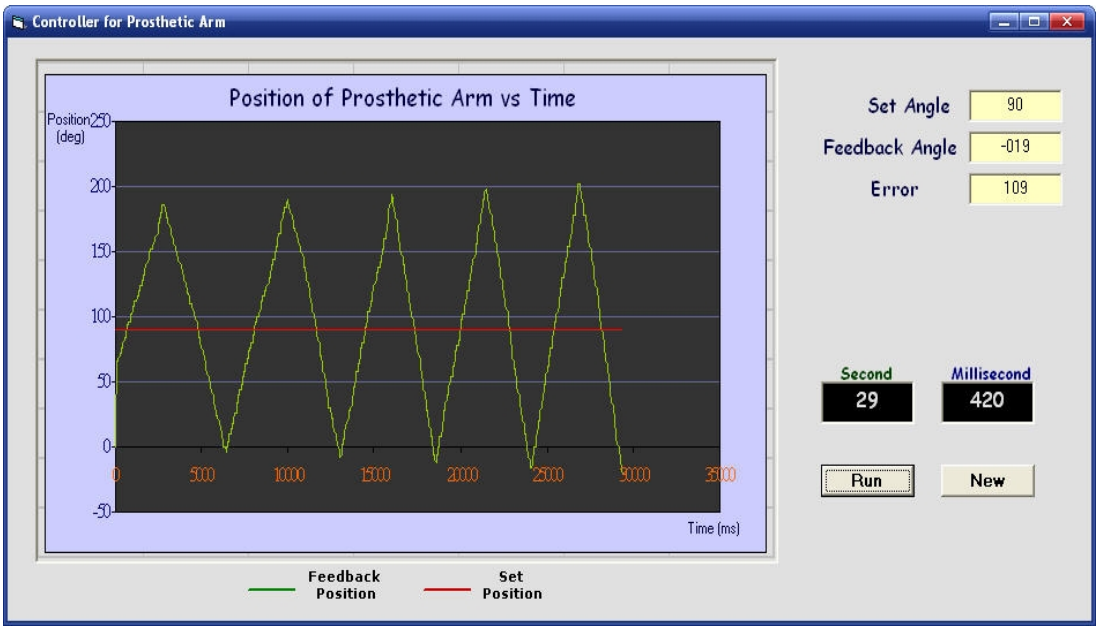


**Figure 4.14: Response Graph of PID Controller (Kp=0, Ki=1, Kd=0)**

Hence, the above results shows that by purely increase $K_i$ will cause the response graph to oscillate without stopping thus additional action need to be taken to stop the output of the system from oscillating too much.

Next, the PID controller is being tuned until the response of the system reaches critically damp condition. (No load condition)

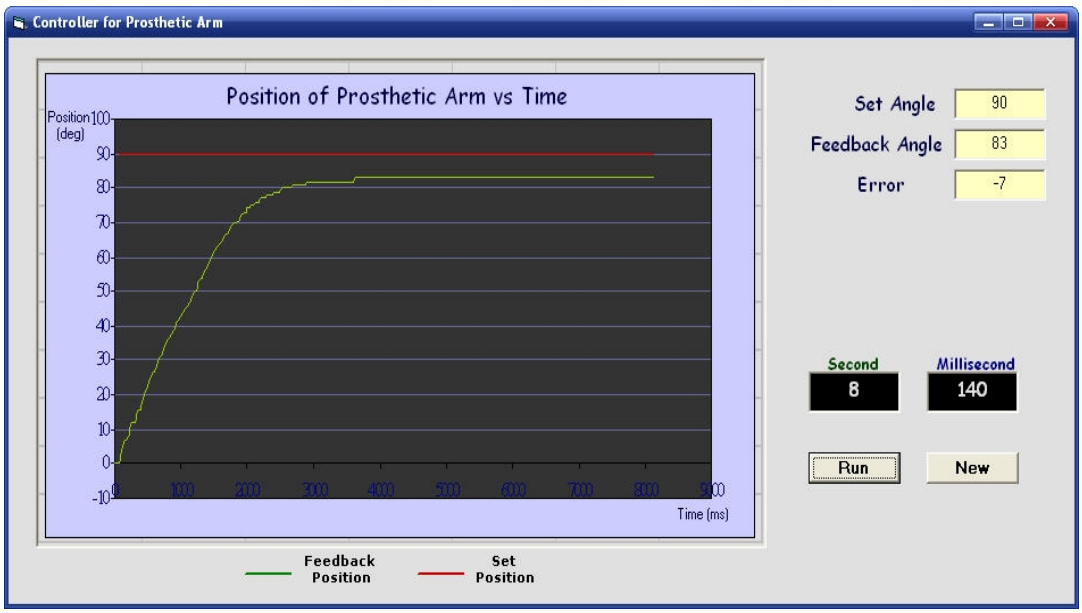


**Figure 4.15: Response Graph of PID Controller (Kp=2, Ki=0, Kd=0)**

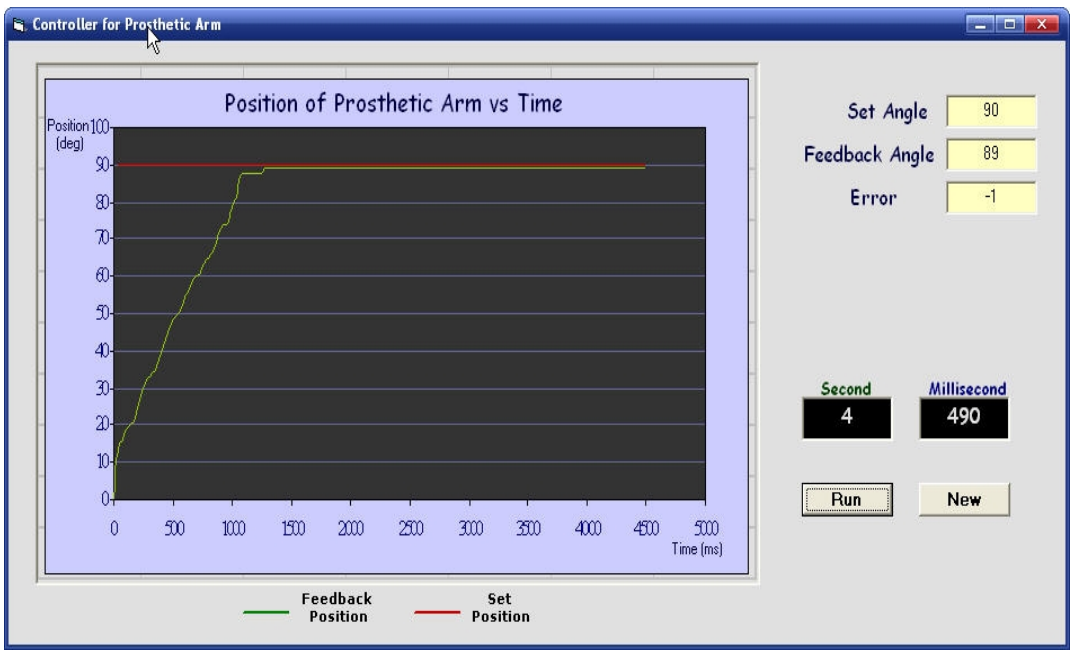


**Figure 4.16: Response Graph of PID Controller (Kp=9, Ki=0, Kd=0)**

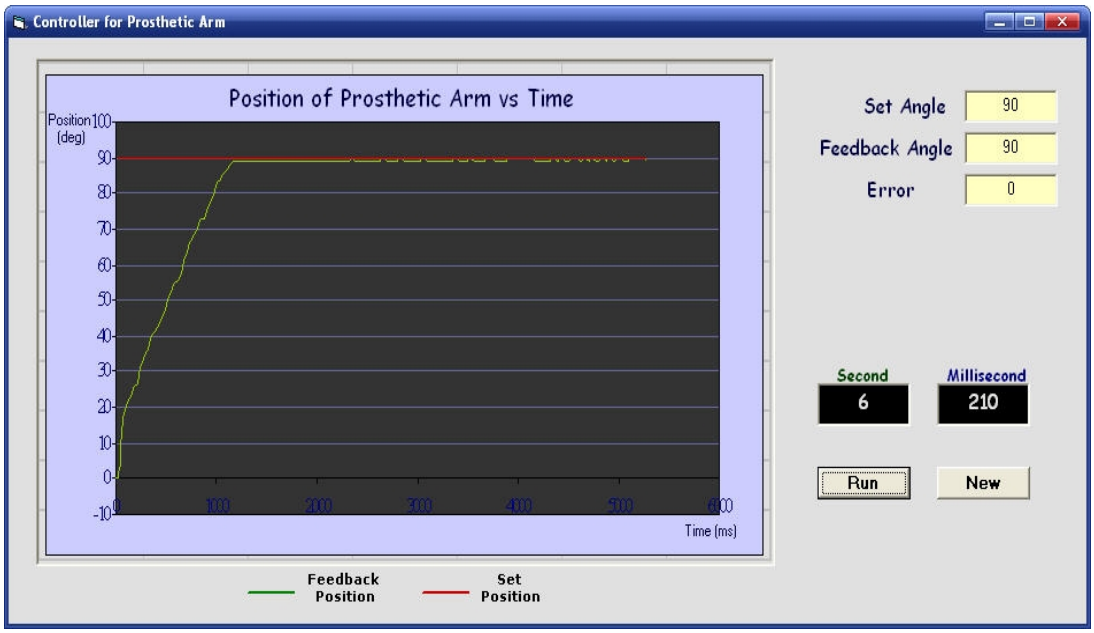


**Figure 4.17: Response Graph of PID Controller (Kp=15, Ki=0, Kd=0)**

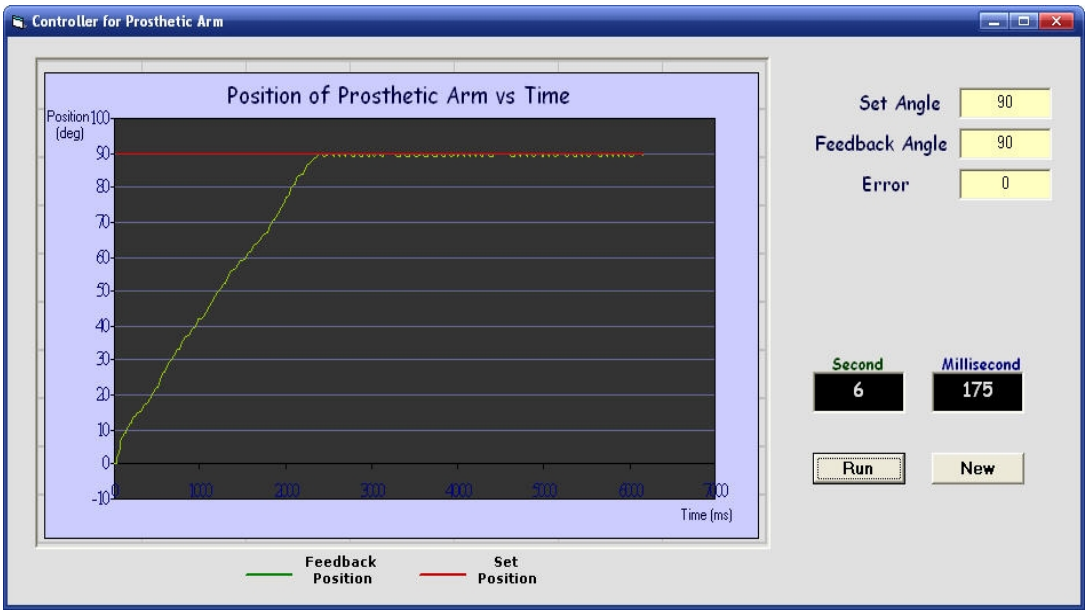


**Figure 4.18: Response Graph of PID Controller (Kp=20, Ki=0, Kd=0)**

From the above 4 graphs, it is obvious that the systems are in critically damp conditions. This shows that the action of proportional gain can be use to stop the response of the system from oscillating too much.

Now, for no load condition, the mechanical arm is to move 90 degree from top. The proportional gain, integral gain and derivative gain are tuned properly to achieve optimum performance of transient response.

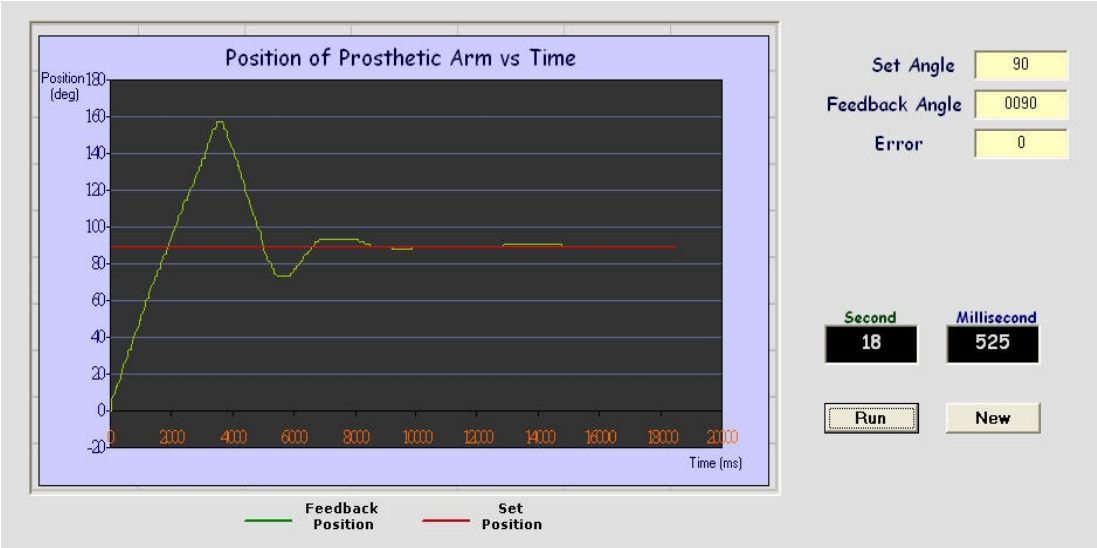


**Figure 4.19: Response Graph of PID Controller (Kp=2, Ki=0.05, Kd=0)**

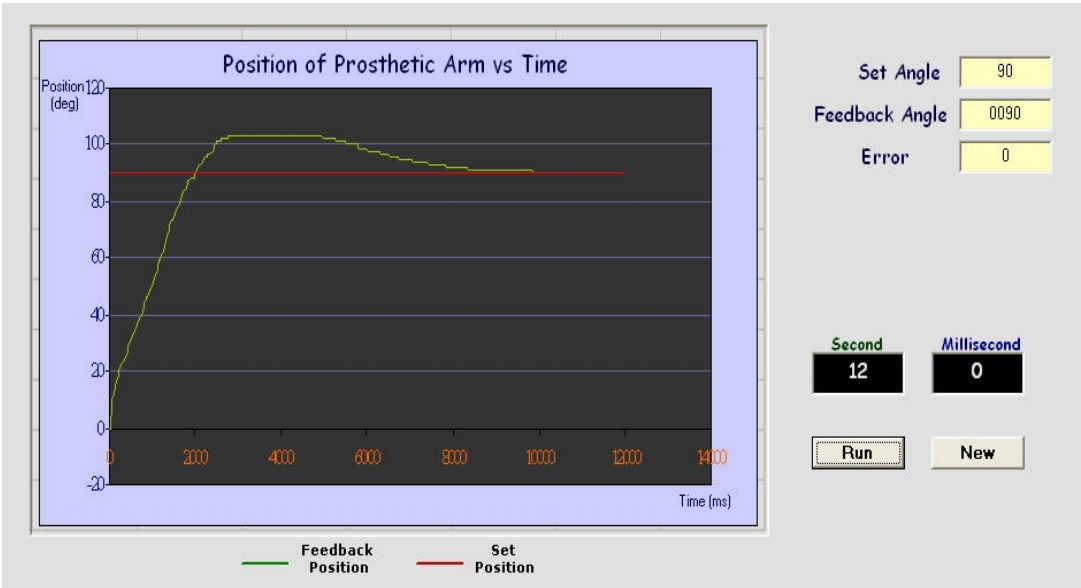


**Figure 4.20: Response Graph of PID Controller (Kp=2, Ki=0.007, Kd=0)**

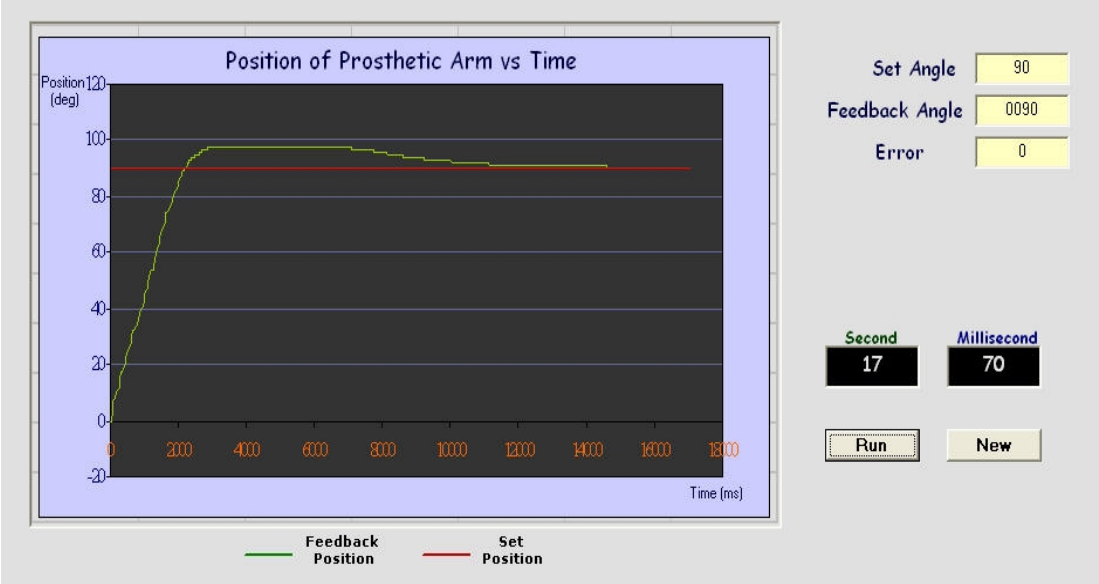


**Figure 4.21: Response Graph of PID Controller (Kp=2, Ki=0.005, Kd=0)**

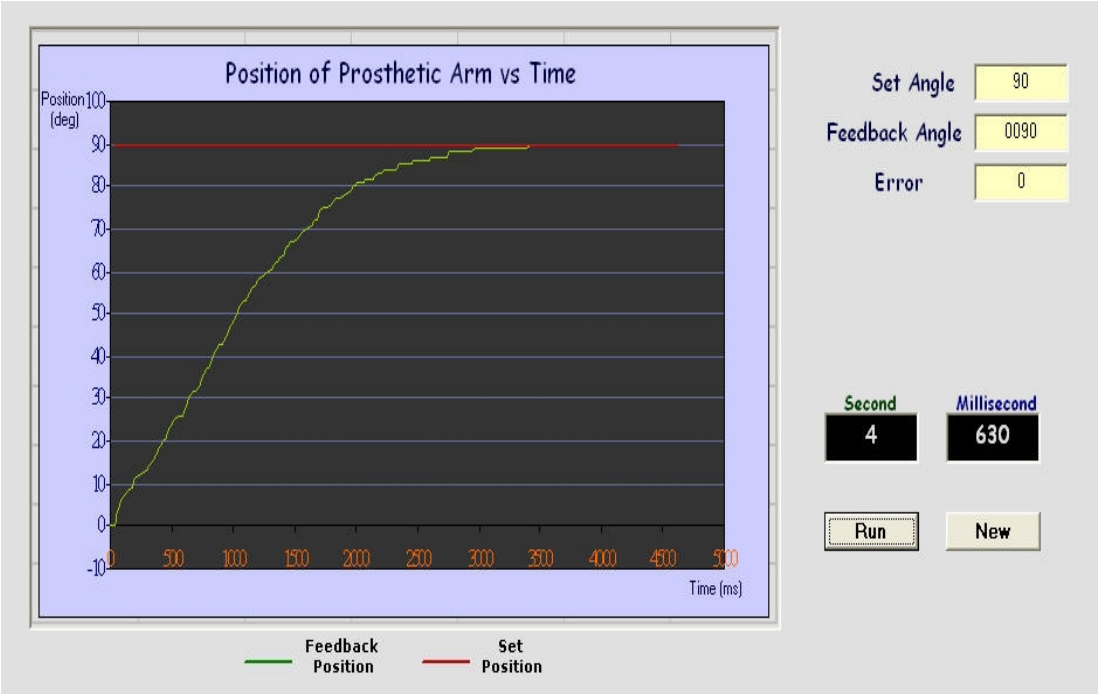


**Figure 4.22: Response Graph of PID Controller (Kp=2, Ki=0.002, Kd=10)**

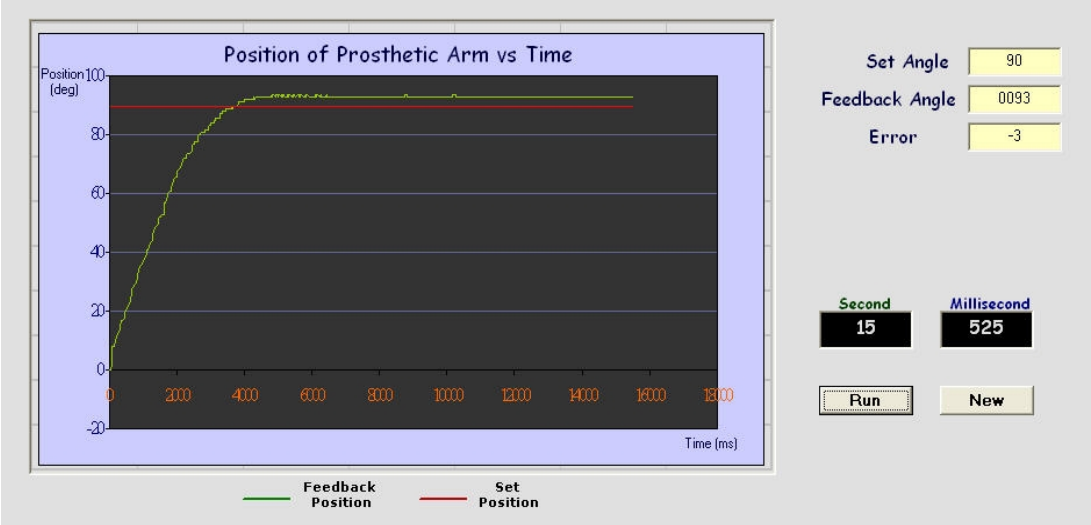


**Figure 4.23: Response Graph of PID Controller (Kp=2, Ki=0.002, Kd=100)**

Form the tuning process, at first, $K_p$ is set to be 2 and at $K_i$ is set to be 0.05. As observed from the graph, the overshoot is large and the settling time is large as well. Next, $K_p$ is kept constant and $K_i$ is being reduced slowly to further decrease the overshoot or oscillating. Next, as $K_i$ is slowly reduced until 0.002 with $K_d$=10, the response of the control system become smooth and does not have any overshoot. This is the best response graph so far. However, if further increase $K_d$ until 100 as shown in fifth graph, the system become less stable as only low derivative gain improve the stability of the system. Hence, after compare to other 4 graphs, graph in figure 4.22 seems to be most optimized with smooth response and without any overshoot. Thus, graph in figure 4.22 will be used to compare with another two types of developed Fuzzy Logic controller in terms of performance.

Important information of the graph in figure 4.22:

Steady state error: 0

%ss = 0

Rise time, $T_r = 2 – 0.2 = 1.8s$

Settling time $T_s$: 2.65s

Overshoot: 0

Tolerance of error: +/-2 (+/-2.22%)

Condition: Critically damped

Next, the mechanical arm is attached with the weight of 200g and the experiments are run to obtain the response graph of the control system. The gain constants are being tuned manually to obtain steady and stable response of the control system.

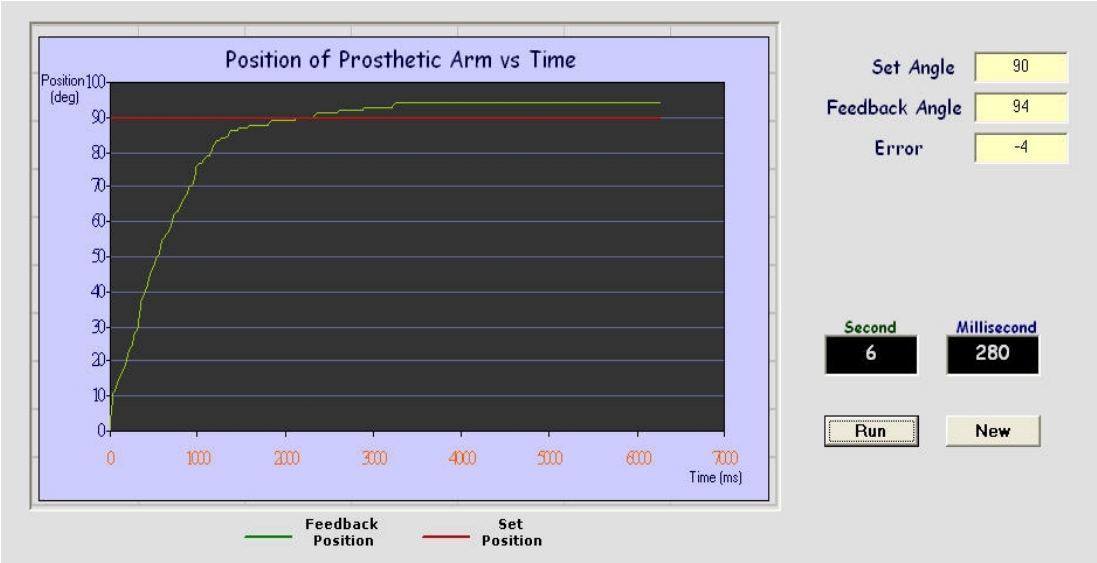


**Figure 4.24: Response Graph of PID Controller (Kp=2, Ki=0.02, Kd=10)**

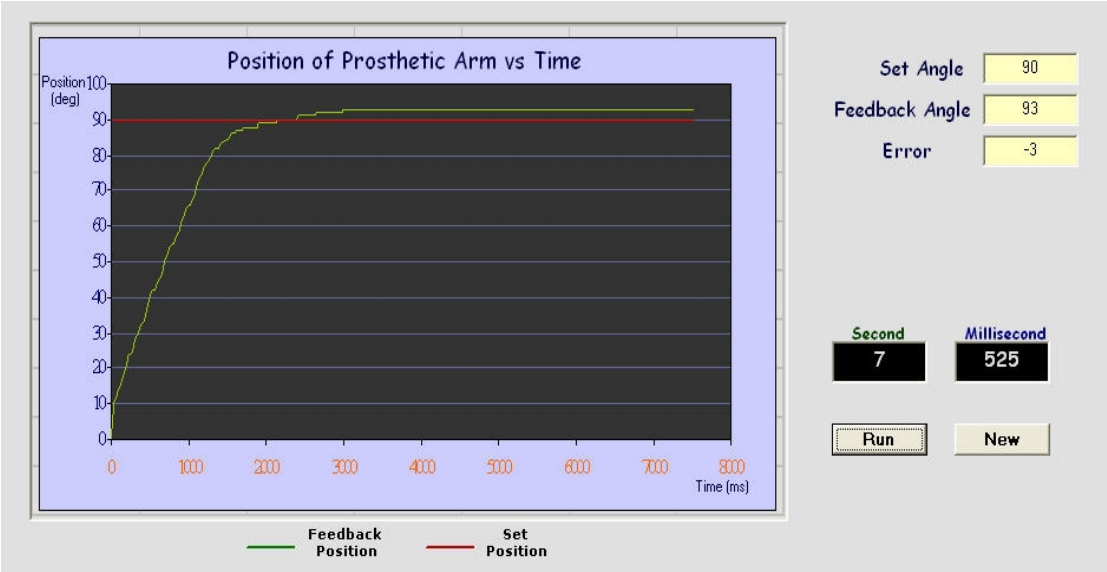


**Figure 4.25: Response Graph of PID Controller (Kp=3, Ki=0.02, Kd=10)**

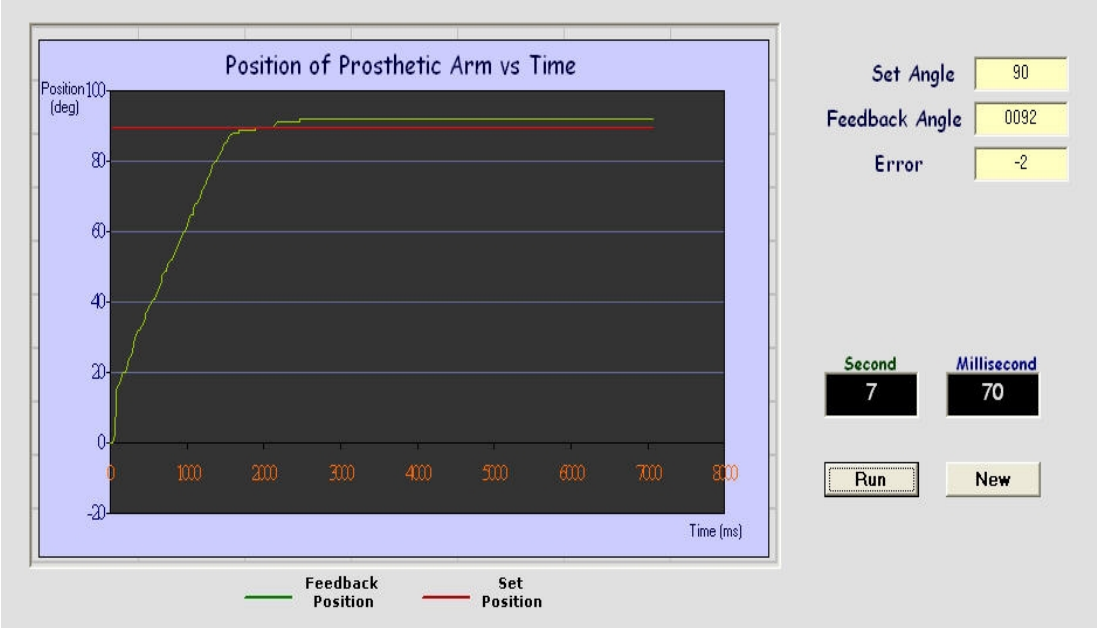


**Figure 4.26: Response Graph of PID Controller (Kp=5, Ki=0.02, Kd=10)**

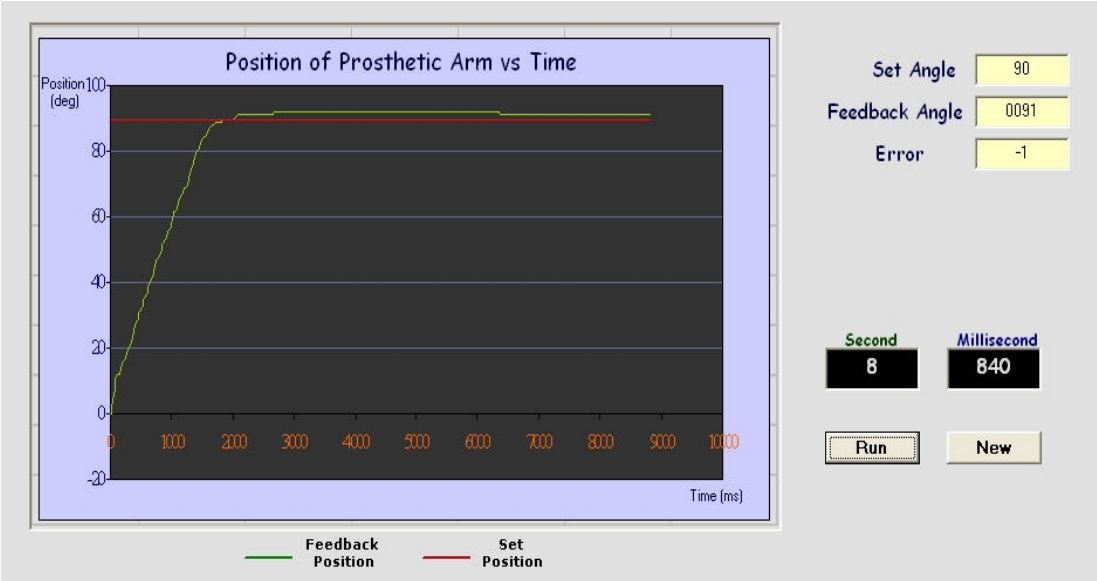


**Figure 4.27: Response Graph of PID Controller (Kp=7, Ki=0.02, Kd=10)**

Under loaded condition, the $K_i$ and $K_d$ remain unchanged ($K_i$=0.02, $K_d$=10) while the $K_p$ is being tuned to reduce the steady state error. As indicated in the graphs above, the increasing of $K_p$ from 2 to 7 has reduce the steady state error from -4 to -1. Compare the no load and loaded conditions, additional weight of the arm will make the arm further slight down and contributed to steady state error and it will reach the target slower as well.

## 4.6 Comparison between 3 Types of Developed Controller

The results and performance of Fuzzy Logic Controller 1, Fuzzy Logic Controller 2 and PID controller are being compared.

No load Conditions:

**Table 4.4: Comparison between 3 Types of Controller (no load)**

| Time | FLC #1 | FLC #2 | PID controller |
|---|---|---|---|
| Rise Time | 1.6s | 1.2s | 1.8s |
| Settling Time | 2.75s | 2.05s | 2.65s |
| Overshoot | 0 | 0 | 0 |
| Steady State Error | -1 | -1 | 0 |

Loaded Conditions (200g):

**Table 4.5: Comparison between 3 Types of Controller (loaded)**

| Time | FLC #1 | FLC #2 | PID controller |
|---|---|---|---|
| Rise Time | 1.8s | 1.7s | 1.85s |
| Settling Time | 2.8s | 2.95s | 2.7s |
| Overshoot | 0 | 0 | 0 |
| Steady State Error | -3 | -3 | -1 |

Under no load condition, the rise time of PID controller is highest and settling time of FLC1 is highest while FLC2 has lowest rise time and settling time. Hence, FLC2 has the tendency to reach the set-point fastest. All the controllers have no overshoot. In terms of steady state error, PID controller has the lowest value which is 0 while both FLC have the steady state error of -1. This comparison shows that PID controller has slightly better performance and accuracy than the Fuzzy Logic controller.

Under loaded condition, the rise time and settling time of the 3 controllers is almost the same while they have no overshoot. The critical difference is the steady state error. The steady state error of PID controller is -1 while the steady state error of both FLC is -3. Once again this shows the PID controller has better performance and accuracy.

Overall, from the comparison above, it is observed that the PID controller has slightly better performance and higher accuracy compare to the two Fuzzy Logic controllers in this control system. Fuzzy Logic controller will have shorter rise time and settling time which means the desired position can be reached faster than that of PID controller. For no load condition, PID controller can reach zero steady state error while Fuzzy Logic controller cannot. For loaded condition, PID controller has lowest steady state error while Fuzzy Logic controller has slightly higher steady state error. In terms of overshoot, Fuzzy Logic controller will not have high overshoot while PID controller if not tuned properly will have high overshoot. Furthermore, Fuzzy Logic controller is able to deal with more changing conditions.

# CHAPTER 5

# CONCLUSION AND RECOMMENDATIONS

## 5.1 Conclusion

Fuzzy Logic controller for the prosthetic arm has been developed. By select either controller mode with proper tuning, the prosthetic arm can reach the desired position.

The response graphs for different controller approach are obtained. The results and performance of PID and Fuzzy Logic controllers are compared and it is found that PID controller has better performance and higher accuracy while Fuzzy Logic controller has less overshoot and able to deal with more changing conditions.

A user-friendly graphic user interface (GUI) for the real time display of the positional control of the prosthetic arm is developed.

## 5.2 Problems

Although a GUI has been designed to do real time monitoring of the positional control of the prosthetic arm, but the GUI developed using Visual Basic 6.0 is not stable enough and keep on encountered errors.

## 5.3 Recommendations

For future works, some recommendations have been proposed based on the problems encountered in order to improve the performance of the controller.

Adaptive controlling method:
Advance controlling method such as adaptive PID controller as well as adaptive Fuzzy Logic controller can be designed to obtain better performance of the control system. Adaptive controller such as Fuzzy-PID Controller, Neural-Network Fuzzy Logic Controller, Genetic Algorithm Fuzzy Logic Controller and so on can be developed to have better control of the prosthetic arm.

Graphic User Interface:
A user-friendly GUI is very important in real time monitoring of the response of the control system. Hence, a more stable GUI with no errors should be developed by using appropriate software, such as Visual Basic 2010.

# REFERENCES

Foundations of Fuzzy Control, Jan Jantzen<jj@iau.dtu.dk>, Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK.Tech. report no 98-E 864 (design), 19 Aug 1998.

Design of Fuzzy Controllers, Jan Jantzen<jj@iau.dtu.dk>, Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK.Tech. report no 98-E 864 (design), 19 Aug 1998.

Tuning Of Fuzzy PID Controllers, Jan Jantzen<jj@iau.dtu.dk>, Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK.Tech. report no 98-E 864 (design), 19 Aug 1998.

Neurofuzzy modelling, Jan Jantzen<jj@iau.dtu.dk>, Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK. Tech. report no 98-E 864 (design), 19 Aug 1998.

Principle of adaptive Fuzzy Logic Controller for DC motor drives, E. E. El-kholy, A. M. Dabroom. General Organization for Technical Education and Vocational Training College of Telecom & Electronics R & D Center, Jeddah, Saudi Arabia. Adel E. El-kholy, Phd, Faculty of Engineering, Cairo University, Egypt.

Xia Changliang, Guo PeiJian, Shi Tingna, and wang Mingehao; "Speed Control of Brushless DC Motor Using Genetic Algorithm Based Fuzzy Controller," International Conference on Intelligent Mechatronics and Automation, pp. 460-464, Aug. 26-31, 2004.

FUZZY LOGIC for Embedded Systems Applications, by Ahmad M. Ibrahim, Ph.D. Senior Member, IEEE. Copyright © 2004, Elsevier Science (USA). All rights reserved.

A Human-Assisting Manipulator Teleoperated by EMG Signals and Arm Motions, Osamu Fukuda, Toshio Tsuji, Member, IEEE, Makoto Kaneko, Senior Member, IEEE, and Akira Otsuka. IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 19, NO. 2, APRIL 2003.

http://www.ikalogic.com/tut_closed_loop_spd_ctrl.php#0

# APPENDICES

## APPENDIX A: C Programming Source Code (Fuzzy Logic Controller1)

```
#include <htc.h>

__CONFIG(1,0x0C00);
__CONFIG(2,0X1E1F);
__CONFIG(3,0X8100);
__CONFIG(4,0X00C1);
__CONFIG(5,0XC00F);

#define motor_dir RC1
#define motor_speed CCPR1L
#define encoderA  RA0
#define encoderB  RA1

signed long n_pulse=0, feed_ang=0, set_ang=0;
float Rx_VB=0;
signed long pre_error=0, integral=0, derivative=0, output=0;
float error_value=0;

void UART_setup(void)
{
        /* relates crystal freq to baud rate

        BRGH=1, Fosc=4MHz
        -----------------
        Baud            SPBRG
        9600            25
        19200           12
        38400           6
        57600           3
        115200          1
        250000          0
        */
        SYNC  = 0;          // select asynchronous mode
        SPBRG = 12;         // baud rate 19200
```

```
        BRGH  = 1;                  // high data rate for sending
        SPEN  = 1;                  // enable serial port pins

        TXIE  = 0;                  // disable tx interrupts
        TX9   = 0;                  // 8-bit transmission
        TXEN  = 1;                  // enable the transmitter

        RCIE  = 0;                  // disable rx interrupts
        RX9   = 0;                  // 8-bit reception
        CREN  = 1;                  // Enables continuous receive
}

void putChar(unsigned char ch)
{
        while(!TXIF)            //set when TXREG is empty
       ;
        TXREG=ch;
}

unsigned char getChar(void)
{
        while(!RCIF)
      ;
        return RCREG;
}

void putString(register const char *Str)
{
  while((*Str)!=0)
  {
     putChar(*Str);
          if (*Str==10) putChar(10);
          if (*Str==13) putChar(13);
     Str++;
   }
}

void USARTWriteInt(int val,unsigned char field_length)
{
        //Convert Number To String and pump over Tx Channel.
        char str[5]={0,0,0,0,0};

        if(val<0)
        {
                putChar('-');           //Write '-' sign for negative numbers.
                val=(val*(-1));         //Make it positive.
        }

        int i=4,j=0;
        while(val)
```

```
        {
                str[i]=val%10;
                val=val/10;
                i--;
        }
        if(field_length>5)
                while(str[j]==0)
                {
                        if(j==4) break;
                        else j++;
                }
        else
                j=5-field_length;

        for(i=j;i<5;i++)
        {
                putChar('0'+str[i]);
        }
}

// Rotate the motor to the right.
void motor_right(unsigned char speed)
{
        motor_dir = 0;                  // Set the direction of the motor to right.
        motor_speed = speed; // Set the PWM value to control the speed.
}


// Rotate the motor to the left.
void motor_left(unsigned char speed)
{
        motor_dir = 1;                  // Set the direction of the motor to left.
        motor_speed = speed; // Set the PWM value to control the speed.
}


// Brake the motor.
void motor_stop(void)
{
        motor_speed = 0;                // Set the PWM value to 0 and brake the motor.
}

void interrupt isr(void)
{
        if(INT0IF)
        {
                INT0IF=0;                               //clear interrupt flag

                if(encoderB)
                        n_pulse+=1;
```

```
                else
                    n_pulse-=1;
            }
}

void Wait(unsigned char delay)
{
  unsigned char i;
  for(i=0;i<delay;i++)
     _delay(1000);
}

void main(void)
{

   ADCON1=0b00111110;
   TRISA=0b00000111;
   TRISB=0b00000001;
   TRISC=0b10000000;
   LATA=LATB=LATC=0;

   UART_setup();

//setup external interrupt
   INTEDG0=0;                          //interrupt on falling edge
   INT0IE=1;                           //enable external interrupt
   PEIE=1;                             //enable peripheral interrupt
   GIE=1;                              //enable global interrupts

 // Setup PWM mode.
     CCP1CON = 0b00001100;             // Set as PWM Mode.
     T2CON = 0b00000101;               // Turn on Timer 2 with prescaler = 4.
     PR2 = 0xFF;                       // Set the PWM period.
     CCPR1L = 0;                       // No PWM Duty Cycle during startup.

         //Variables Declaration
         unsigned char charRxTx=0, Rx_digit[10], i=0, j=0, stop_Rx=0,k=2, r=0;
         unsigned long dec_division=1;
         float a=0, b=0;

         while(1)
         {

            charRxTx = getChar();

             if(charRxTx==13)
                  {

                     if(i==1)
                       Rx_VB=Rx_digit[0];
```

```
      if(i==2)
        Rx_VB=Rx_digit[0]*10+Rx_digit[1];

      if(i==3)
        Rx_VB=Rx_digit[0]*100+Rx_digit[1]*10+Rx_digit[2];

set_ang = Rx_VB;

while(stop_Rx != 'S')
{

  feed_ang=(n_pulse*360)/45000;

  Wait(5);

  if(feed_ang < 0)
  USARTWriteInt(feed_ang,3);

  else
  USARTWriteInt(feed_ang,4);

  //putString(" ");

  error_value = set_ang - feed_ang;      // Calculate the error.

  if(error_value>=0 && error_value<=20)
  {
    a=(-1*error_value)/20 + 1;
    b=error_value/20;
    output=a*0+b*65;
  }

  else if(error_value>20 && error_value<=40)
  {
    a=(-1*error_value)/20 + 2;
    b=error_value/20 - 1;
    output=a*65+b*75;
  }

  else if(error_value>40)
  {
     output=75;
  }

  else if(error_value>=-20 && error_value<0)
  {
    a=error_value/20 + 1;
    b=error_value/20;
    output=a*0+b*(-65);
  }
```

```
                    else if(error_value>=-40 && error_value<-20)
                    {
                      a=error_value/20 + 2;
                      b=(-1*error_value/20) - 1;
                      output=a*(-65)+b*(-75);
                    }

                    else if(error_value<-40)
                    {
                       output=-75;
                    }

                    if (output > 0) motor_right((unsigned char)output);
                    // When output is positive, turn right.
                    else if (output < 0) motor_left((unsigned char)(-output));
                    // When error is negative, turn left.
                    else motor_stop();
                    // When output is 0, stop the motor.

                    if(RCIF)
                    stop_Rx=RCREG;
                    }
                    motor_stop();

                    TMR1ON=0;

                    set_ang=0;

                    feed_ang=0;

                    n_pulse=0;

                    i=0;

                    j=0;

                    stop_Rx=0;
                    }
                    else
                    {
                      if(charRxTx>=48 && charRxTx<=57)
                      Rx_digit[i]=charRxTx-48;

                      else if(charRxTx==46)
                      Rx_digit[i]=charRxTx;

                      i++;
                     }
            }
}
```

APPENDIX B: C programming Source Code (Fuzzy Logic Controller2)

```
#include <htc.h>

__CONFIG(1,0x0C00);
__CONFIG(2,0X1E1F);
__CONFIG(3,0X8100);
__CONFIG(4,0X00C1);
__CONFIG(5,0XC00F);

#define motor_dir RC1
#define motor_speed CCPR1L
#define encoderA  RA0
#define encoderB  RA1

const float      a=-90;
const float      b=-60;
const float      c=-30;
const float      d=0;
const float      e=30;
const float      f=60;
const float      g=90;
const float      PWM_PB=75;
const float      PWM_PM=70;
const float      PWM_PS=65;
const float       PWM_ZO=0;
const float      PWM_NS=-65;
const float      PWM_NM=-70;
const float      PWM_NB=-75;

signed long n_pulse=0, feed_ang=0, set_ang=0;
float Rx_VB=0;
signed long pre_error=0, integral=0, derivative=0, output=0;
float error_value=0;

void UART_setup(void)
{
        /* relates crystal freq to baud rate

        BRGH=1, Fosc=4MHz
        ----------------
        Baud            SPBRG
```

```
        9600            25
        19200           12
        38400           6
        57600           3
        115200          1
        250000          0
        */
        SYNC  = 0;                  // select asynchronous mode
        SPBRG = 12;                 // baud rate 19200
        BRGH  = 1;                  // high data rate for sending
        SPEN  = 1;                  // enable serial port pins

        TXIE  = 0;                  // disable tx interrupts
        TX9   = 0;                  // 8-bit transmission
        TXEN  = 1;                  // enable the transmitter

        RCIE  = 0;                  // disable rx interrupts
        RX9   = 0;                  // 8-bit reception
        CREN  = 1;                  // Enables continuous receive
}

void putChar(unsigned char ch)
{
        while(!TXIF)            //set when TXREG is empty
       ;
        TXREG=ch;
}

unsigned char getChar(void)
{
        while(!RCIF)
      ;
        return RCREG;
}

void putString(register const char *Str)
{
   while((*Str)!=0)
   {
     putChar(*Str);
        if (*Str==10) putChar(10);
        if (*Str==13) putChar(13);
     Str++;
    }
}

void USARTWriteInt(int val,unsigned char field_length)
{
        //Convert Number To String and pump over Tx Channel.
        char str[5]={0,0,0,0,0};
```

```
        if(val<0)
        {
                //str[1]=45;
                putChar('-');           //Write '-' sign for negative numbers.
                val=(val*(-1));         //Make it positive.
        }

        int i=4,j=0;
        while(val)
        {
                str[i]=val%10;
                val=val/10;
                i--;
        }
        if(field_length>5)
                while(str[j]==0)
                {
                        if(j==4) break;
                        else j++;
                }
        else
                j=5-field_length;

        for(i=j;i<5;i++)
        {
                putChar('0'+str[i]);
        }
}

// Rotate the motor to the right.
void motor_right(unsigned char speed)
{
        motor_dir = 0;                  // Set the direction of the motor to right.
        motor_speed = speed; // Set the PWM value to control the speed.
}


// Rotate the motor to the left.
void motor_left(unsigned char speed)
{
        motor_dir = 1;                  // Set the direction of the motor to left.
        motor_speed = speed; // Set the PWM value to control the speed.
}


// Brake the motor.
void motor_stop(void)
{
        motor_speed = 0;                // Set the PWM value to 0 and brake the motor.
}
```

```
void interrupt isr(void)
{
//      signed long error_value, pre_error, integral, derivative, output=0;

        if(INT0IF)
        {
                INT0IF=0;                               //clear interrupt flag

                if(encoderB)
                        n_pulse+=1;
                else
                   n_pulse-=1;
        }
}

void Wait(unsigned char delay)
{
  unsigned char i;
  for(i=0;i<delay;i++)
    _delay(1000);
}

float NB_func(float x, float a, float b)
{
        if (x<=-90)
        return 1;

        else if ((x>-90)&&(x<-60))
        return ((b-x)/(b-a));

        else if(x>=-60)
        return 0;
}

float NM_func(float x, float a, float b, float c)
{
        if (x<=-90)
        return 0;

        else if ((x>-90)&&(x<=-60))
        return ((x-a)/(b-a));

        else if ((x>-60)&&(x<-30))
        return ((c-x)/(c-b));

        else if (x>=-30)
        return 0;
}

float NS_func(float x, float b, float c, float d)
```

```
{
        if (x<=-60)
        return 0;

        else if ((x>-60)&&(x<=-30))
        return ((x-b)/(c-b));

        else if ((x>-30)&&(x<0))
        return ((d-x)/(d-c));

        else if (x>=0)
        return 0;
}

float ZO_func(float x, float c, float d, float e)
{
        if (x<=-30)
        return 0;

        else if ((x>-30)&&(x<=0))
        return ((x-c)/(d-c));

        else if ((x>0)&&(x<30))
        return ((e-x)/(e-d));

        else if (x>=30)
        return 0;
}

float PS_func(float x, float d, float e, float f)
{
        if (x<=0)
        return 0;

        else if ((x>0)&&(x<=30))
                return ((x-d)/(e-d));

        else if ((x>30)&&(x<60))
                return ((f-x)/(f-e));

        else if (x>=60)
        return 0;
}

float PM_func(float x, float e, float f, float g)
{
        if (x<=30)
        return 0;

        else if ((x>30)&&(x<=60))
```

```
                return ((x-e)/(f-e));

        else if ((x>60)&&(x<90))
                return ((g-x)/(g-f));

        else if (x>=90)
        return 0;
}

float PB_func(float x, float f, float g)
{
        if (x<=60)
        return 0;

        else if ((x>60)&&(x<90))
                return ((x-f)/(g-f));

        else if (x>=90)
        return 1;
}

void main(void)
{

   ADCON1=0b00111110;
   TRISA=0b00000111;
   TRISB=0b00000001;
   TRISC=0b10000000;
   LATA=LATB=LATC=0;

   UART_setup();

        //setup external interrupt
   INTEDG0=0;                          //interrupt on falling edge
   INT0IE=1;                        //enable external interrupt
   PEIE=1;                             //enable peripheral interrupt
   GIE=1;                              //enable global interrupts

  // Setup PWM mode.
        CCP1CON = 0b00001100;    // Set as PWM Mode.
        T2CON = 0b00000101;             // Turn on Timer 2 with prescaler = 4.
        PR2 = 0xFF;                     // Set the PWM period.
        CCPR1L = 0;                  // No PWM Duty Cycle during startup.

        //variables declaration
        unsigned char charRxTx=0, Rx_digit[10], i=0, stop_Rx=0;
        signed float NB, NM, NS, ZO, PS, PM, PB, weighed_pwm, weight;

        while(1)
        {
```

```
charRxTx = getChar();

if(charRxTx==13)
{
  if(Rx_digit[0]=='-')
  {
    if(i==2)
    Rx_VB=(-1)*Rx_digit[1];

    if(i==3)
    Rx_VB=(-1)*(Rx_digit[1]*10+Rx_digit[2]);

    if(i==4)
    Rx_VB=(-1)*(Rx_digit[1]*100+Rx_digit[2]*10+Rx_digit[3]);
  }

  else
  {
     if(i==1)
     Rx_VB=Rx_digit[0];

      if(i==2)
      Rx_VB=Rx_digit[0]*10+Rx_digit[1];

      if(i==3)
      Rx_VB=Rx_digit[0]*100+Rx_digit[1]*10+Rx_digit[2];
   }

  set_ang = Rx_VB;

  while(stop_Rx != 'S')
  {

     feed_ang=(n_pulse*360)/45000;

     Wait(5);

     if(feed_ang < 0)
     USARTWriteInt(feed_ang,3);

     else
     USARTWriteInt(feed_ang,4);

     error_value = set_ang - feed_ang;     // Calculate the error.

     NB = NB_func(error_value, a, b);
     NM = NM_func(error_value, a, b ,c);
     NS = NS_func(error_value, b, c, d);
     ZO = ZO_func(error_value, c, d, e);
     PS = PS_func(error_value, d, e, f);
```

```
                PM = PM_func(error_value, e, f, g);
                PB = PB_func(error_value, f, g);

weighed_pwm
=(NB*PWM_NB)+(NM*PWM_NM)+(NS*PWM_NS)+(ZO*PWM_ZO)+(PS*PW
M_PS)+(PM*PWM_PM)+(PB*PWM_PB);

                weight = NB+NM+NS+ZO+PS+PM+PB;

                output = weighed_pwm / weight;

                if (output > 0)motor_right((unsigned char)output);
                // When output is positive, turn right.
                else if (output < 0) motor_left((unsigned char)(-output));
                // When error is negative, turn left.
                else motor_stop();
                // When output is 0, stop the motor.

                if(RCIF)
                stop_Rx=RCREG;
                }

                motor_stop();

                TMR1ON=0;

                set_ang=0;

                feed_ang=0;

                n_pulse=0;

                i=0;

                stop_Rx=0;
                }

                else
                {
                  if(charRxTx>=48 && charRxTx<=57)
                  Rx_digit[i]=charRxTx-48;

                  else if(charRxTx==45)
                  Rx_digit[i]=charRxTx;

                   i++;
                  }
          }
}
```

# APPENDIX C: C Programming Source Code (PID controller)

```
#include <htc.h>

__CONFIG(1,0x0C00);
__CONFIG(2,0X1E1F);
__CONFIG(3,0X8100);
__CONFIG(4,0X00C1);
__CONFIG(5,0XC00F);

//#define _XTAL_FREQ 4000000
#define motor_dir RC1
#define motor_speed CCPR1L
#define encoderA  RA0
#define encoderB  RA1

signed long n_pulse=0, feed_ang=0, set_ang=0;
float Rx_VB=0;
float kp=7; //oscillate at 14, 60% at 9
float ki=0.002;
float kd=10;
signed long error_value=0, pre_error=0, integral=0, derivative=0, output=0;

void UART_setup(void)
{
        /* relates crystal freq to baud rate

        BRGH=1, Fosc=4MHz
        -----------------
        Baud            SPBRG
        9600            25
        19200           12
        38400           6
        57600           3
        115200          1
        250000          0
        */
        SYNC  = 0;                  // select asynchronous mode
        SPBRG = 12;                 // baud rate 19200
        BRGH  = 1;                  // high data rate for sending
        SPEN  = 1;                  // enable serial port pins
```

```
        TXIE  = 0;                  // disable tx interrupts
        TX9   = 0;                  // 8-bit transmission
        TXEN  = 1;                  // enable the transmitter

        RCIE  = 0;                  // disable rx interrupts
        RX9   = 0;                  // 8-bit reception
        CREN  = 1;                  // Enables continuous receive
}

void putChar(unsigned char ch)
{
        while(!TXIF)            //set when TXREG is empty
       ;
        TXREG=ch;
}

unsigned char getChar(void)
{
        while(!RCIF)
      ;
        return RCREG;
}

void putString(register const char *Str)
{
  while((*Str)!=0)
  {
    putChar(*Str);
        if (*Str==10) putChar(10);
        if (*Str==13) putChar(13);
    Str++;
   }
}

void USARTWriteInt(int val,unsigned char field_length)
{
        //Convert Number To String and pump over Tx Channel.
        char str[5]={0,0,0,0,0};

        if(val<0)
        {
                //str[1]=45;
                putChar('-');           //Write '-' sign for negative numbers.
                val=(val*(-1));         //Make it positive.
        }

        int i=4,j=0;
        while(val)
        {
                str[i]=val%10;
```

```
		val=val/10;
		i--;
	}
	if(field_length>5)
		while(str[j]==0)
		{
			if(j==4) break;
			else j++;
		}
	else
		j=5-field_length;

	for(i=j;i<5;i++)
	{
		putChar('0'+str[i]);
	}
}

// Rotate the motor to the right.
void motor_right(unsigned char speed)
{
	motor_dir = 0;			// Set the direction of the motor to right.
	motor_speed = speed; // Set the PWM value to control the speed.
}


// Rotate the motor to the left.
void motor_left(unsigned char speed)
{
	motor_dir = 1;			// Set the direction of the motor to left.
	motor_speed = speed; // Set the PWM value to control the speed.
}


// Brake the motor.
void motor_stop(void)
{
	motor_speed = 0;		// Set the PWM value to 0 and brake the motor.
}

void interrupt isr(void)
{

	if(INT0IF)
	{
		INT0IF=0;				//clear interrupt flag

		if(encoderB)
			n_pulse+=1;
		else
```

```
                n_pulse-=1;
        }
}

void main(void)
{

   ADCON1=0b00111110;
   TRISA=0b00000111;
   TRISB=0b00000001;
   TRISC=0b10000000;
   LATA=LATB=LATC=0;

   UART_setup();

   //setup external interrupt
   INTEDG0=0;                          //interrupt on falling edge
   INT0IE=1;                           //enable external interrupt
   PEIE=1;                             //enable peripheral interrupt
   GIE=1;                              //enable global interrupts

  // Setup PWM mode.
        CCP1CON = 0b00001100;    // Set as PWM Mode.
        T2CON = 0b00000101;             // Turn on Timer 2 with prescaler = 4.
        PR2 = 0xFF;                     // Set the PWM period.
        CCPR1L = 0;                  // No PWM Duty Cycle during startup.

        unsigned char charRxTx=0, Rx_digit[10], i=0, j=0, stop_Rx=0,k=2, r=0;
        unsigned long dec_division=1;

        while(1)
        {

            charRxTx = getChar();


            if(charRxTx==13)
            {


              if(i==1)
              Rx_VB=Rx_digit[0];

              if(i==2)
              Rx_VB=Rx_digit[0]*10+Rx_digit[1];

              if(i==3)
              Rx_VB=Rx_digit[0]*100+Rx_digit[1]*10+Rx_digit[2];

              set_ang = Rx_VB;
```

```
//setup Timer1
TMR1H = 0;                              //initialize timer 1
TMR1L = 0;
T1CON = 0b00110001;           //enable Timer1 with 1:8 prescale value

 while(stop_Rx != 'S')
{

   if(TMR1H && TMR1L)
   {

      feed_ang=(n_pulse*360)/45000;

       _delay(400);

       if(feed_ang < 0)
       USARTWriteInt(feed_ang,3);

       else
       USARTWriteInt(feed_ang,4);

       //putString(" ");

       error_value = set_ang - feed_ang;          // Calculate the error.
       integral = integral + error_value;         // Calculate integral.
       derivative = error_value - pre_error;      // Calculate derivative.

       output = (kp * error_value) + (ki * integral) + (kd * derivative);
       // Calculate the output, pwm.

       if (output > 75) output = 75; // Limit the output to maximum 75.
      else if (output < -75) output = -75;

       if (output > 0)       motor_right((unsigned char)output);
       // When output is positive, turn right.
       else if (output < 0) motor_left((unsigned char)(-output));
       // When error is negative, turn left.
       else motor_stop();
       // When output is 0, stop the motor.

       pre_error = error_value;       // Save as previous error.

       if(RCIF)
       stop_Rx=RCREG;
        }
   }

   motor_stop();

   TMR1ON=0;
```

```
                set_ang=0;

                feed_ang=0;

                n_pulse=0;

                i=0;

                j=0;

                stop_Rx=0;

                }

                else
                {
                  if(charRxTx>=48 && charRxTx<=57)
                  Rx_digit[i]=charRxTx-48;

                  else if(charRxTx==46)
                  Rx_digit[i]=charRxTx;

                  i++;
                   }
        }

}
```

# APPENDIX D: Visual Basic Source Code (GUI)

```
Option Explicit
    Dim set_ang As Double
    Dim feed_ang As Variant             ' Feedback angle from microcontroller
    Dim error_ang As Double
    Dim objExcel As Excel.Application   ' Setup for OLE graph
    Dim wExcel As Excel.Workbook
    Dim xlchart As Excel.Chart
    Dim j As Integer
    Dim error_adj As Integer
    Dim secs As Long
    Dim msecs As Long
    'Dim Kp As Double
    'Dim Ki As Double
    'Dim Kd As Double

Private Sub Form_Load()

    '  Use COM8
    MSComm1.CommPort = 8

    ' 19200 baud, no parity, 8 data bits, 1 stop bit
    MSComm1.Settings = "19200,N,8,1"

    ' Fire Rx Event Every 4 Bytes
    MSComm1.RThreshold = 4

    ' When Inputting Data, Input 4 Bytes at a time
    MSComm1.InputLen = 4

    ' Disable DTR
    MSComm1.DTREnable = False

    ' Open the port
    MSComm1.PortOpen = True

    'Kp_txt.Text = ""
    'Ki_txt.Text = ""
    'Kd_txt.Text = ""
    set_ang_txt.Text = ""
```

```
    feed_ang_txt.Text = 0
    feed_ang_txt.Locked = True

    error_txt.Text = 0
    error_txt.Locked = True

    error_adj = 0

    ' Setup for excel file
    Set objExcel = GetObject("", "Excel.Application")

    Set wExcel = objExcel.Workbooks.Open(App.Path & "\Motor_pos_.xls") 'Motor_pos as excel file

    objExcel.Visible = False

    j = 2

    With wExcel.Sheets("Sheet2")
        .Cells(1, 1) = 0
        .Cells(1, 2) = 0
        .Cells(2, 1) = 0
    End With

        OLE_pos.Update

End Sub

Private Sub Form_Unload(Cancel As Integer)

    MSComm1.PortOpen = False
    Call wExcel.Close(False)
    objExcel.Quit
    Set objExcel = Nothing

End Sub

Private Sub MSComm1_OnComm()

    If MSComm1.CommEvent = comEvReceive Then

        ' This is used when data is received
        feed_ang = MSComm1.Input                    'Get the feedback angle from microcontroller

      If (j = 2 And (feed_ang > 9 And feed_ang < 99)) Or (j = 3 And (feed_ang > 9 And feed_ang < 99)) Or error_adj = 1 Then

            feed_ang = Val(feed_ang / 10)
            feed_ang_txt.Text = feed_ang
```

```
        error_ang = Round(set_ang - feed_ang)        'Calculate the error between
feedback angle with the set angle
        error_txt.Text = error_ang
        error_adj = 1

      ElseIf (j = 2 And feed_ang > 99) Or (j = 3 And feed_ang > 99) Or error_adj = 2
Then

        feed_ang = Val(feed_ang / 100)
        feed_ang_txt.Text = feed_ang
        error_ang = Round(set_ang - feed_ang)        'Calculate the error between
feedback angle with the set angle
        error_txt.Text = error_ang
        error_adj = 2

      Else

        feed_ang_txt.Text = Val(feed_ang)
        error_ang = Round(set_ang - feed_ang)
        error_txt.Text = error_ang

      End If

  End If

  If (secs_lbl.Caption * 1000) + msecs_lbl.Caption >
wExcel.Sheets("Sheet2").Cells(j - 1, 1).Value Then

      'Plotting graph
      With wExcel.Sheets("Sheet2")

        .Cells(j, 1) = (secs_lbl.Caption * 1000) + msecs_lbl.Caption
        .Cells(j, 2) = feed_ang
        .Cells(j, 3) = set_ang

      End With

        j = j + 1

        OLE_pos.Update  ' Update the graph



  End If

End Sub



Private Sub new_btn_Click()
```

```
    'Clear the graph
    wExcel.Sheets("Sheet2").Range("A1:C3000") = ""
    j = 2
    OLE_pos.Update
    msecs_lbl.Caption = "0"
    secs_lbl.Caption = "0"
    'Kp_txt.Text = ""
    'Ki_txt.Text = ""
    'Kd_txt.Text = ""
    set_ang_txt.Text = ""
    feed_ang_txt.Text = 0
    error_txt.Text = 0
    error_adj = 0

End Sub

Private Sub run_btn_Click()

    If run_btn.Caption = "Run" Then

        'Kp = Kp_txt.Text
        'Ki = Ki_txt.Text
        'Kd = Kd_txt.Text
        set_ang = set_ang_txt.Text

        If set_ang = 0 Then

            MsgBox "You must select a desired angle first!", vbOKOnly, "Invalid Data"
' If no set angle received from user

        Else

            MSComm1.Output = set_ang & Chr$(13)
            'MSComm1.Output = Kp & Chr$(13)
            'MSComm1.Output = Ki & Chr$(13)
            'MSComm1.Output = Kd & Chr$(13)
            Timer1.Enabled = True
            run_btn.Caption = "Stop"
            new_btn.Enabled = False
            'Kp_txt.Locked = True
            'Ki_txt.Locked = True
            'Kd_txt.Locked = True
            set_ang_txt.Locked = True


        End If

    ElseIf run_btn.Caption = "Stop" Then

        MSComm1.Output = "S"
```

```
        Timer1.Enabled = False
        run_btn.Caption = "Run"
        new_btn.Enabled = True
        'Kp_txt.Locked = False
        'Ki_txt.Locked = False
        'Kd_txt.Locked = False
        set_ang_txt.Locked = False

    End If

End Sub

Private Sub Timer1_Timer()

    msecs_lbl.Caption = msecs_lbl.Caption + 35

    If msecs_lbl.Caption >= 1000 Then

        msecs_lbl.Caption = 0
        secs_lbl.Caption = secs_lbl.Caption + 1

    End If

End Sub
```

APPENDIX E: Figures of Prototype of Prosthetic Arm