Sudan University of Science and Technology
College of Postgraduate Studies

Speed Control of DC Motor Using Proportional–Integral–Derivative Controller and Fuzzy Logic

التحكم في سرعة محرك التيار المستمر باستخدام المتحكم التناسبي التكاملي التفاضلي و المنطق الغامض

A thesis Submitted in Partial Fulfillment of the Requirement for the Degree of M.Sc. in Mechatronics Engineering

Prepared By:
Abdelgaleel Ahamed Omer AlSheik

Supervisor:
Dr. Ebtihal Haider Gismalla Yousif

August 2018

الإستهلال

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

وَيَسـَٔلونَكَ عَنِ ٱلرّوحِ قُلِ ٱلرّوحُ مِنْ أَمْرِ رَبّى وَما أوتيتُمْ مِنَ ٱلْعِلْمِ إِلّا قَليلًا ﴿٨٥﴾

سورة ٱلإِسْراءِ

Acknowledgments

First of all thanks for my God who helped me to complete this project. I would like to express my deepest appreciation to my supervisor Dr. Ebtihal Haider Gismalla Yousif for all her assistance during this master thesis work , and everybody help me to success of this research.

Abstract

Fuzzy control systems have been successfully applied to a wide variety of practical problems. It has been shown that these controllers may perform better than conventional controllers, especially when applied to processes difficult to model, with nonlinearities. On the other hand, the direct current (DC) motor has been widely used as well because it is one of the most common actuators used in control systems. The main objective of this research is to control the speed of DC motor using a combination of the Fuzzy Logic Controller (FLC) and the proportional–integral–derivative (PID) controller. In this study, the targeted problem is the design of the circuit for control of the DC motor based on an Arduino unit. The employed tools are Protus and MatLab. The design and implementation steps will be shown and explained throughout the thesis and the results are discussed based on the speed of response. It will be shown that the speed of response and precision are better in terms of performance than the conventional PID model.

المستخلص

تم تطبيق انظمة التحكم الغامض بنجاح علي مجموعة واسعه من المشاكل العملية ولقد ثبت ان اداء هذه المتحكمات افضل من اداء المتحكمات التقليدية وخصوصا عندما تطبق علي عمليات ذات صعوبة في ايجاد نموذج لها او ذات طبيعة لا خطية. لقد تم استخدام محرك التيار المستمر لانه واحد من اكثر المشغلات شيوعا في انظمة التحكم. ان الهدف الاساسي من هذا البحث هو التحكم في سرعة محرك التيار المستمر باستخدام خليط من متحكم المنطق الغامض والمتحكم التناسبي التكاملي التفاضلي. تعالج هذه الدراسة مشكلة كيفية تصميم دائرة للتحكم بمحرك التيار المستمر اعتمادا على وحدة اردوينو. الادوات التي تم استخدامها تشمل برنامج بروتس وبرنامج ماتلاب. خلال هذا البحث سيتم عرض و مناقشة خطوات التصميم بالاضافة الى مناقشة النتائج المتحصلة اعتمادا على سرعة الاستجابة. سيتبين أن سرعة الاستجابة والدقة أفضل من حيث الأداء من نموذج المتحكم التناسبي التكاملي التفاضلي التقليدي.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CAutoD	Computer Automated Design
CPU	Central Processing Unit
D	Derivative
DC	Direct Current
FLC	Fuzzy Logic Controller
FOPDT	First Order Pulse Dead Time
GTO	Gate Turn Off
I	Integral
I/O	Input/ Output
IC	Integrated Circuit
IGBT	Insulated Gate Bipolar Transistor
IR	Infrared
LCD	Liquid Crystal Display
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MV	Manipulated Variable
NB	Negative Big
NM	Negative Middle
NS	Negative Small
P	Proportional
PB	Positive Big
PD	Proportional-Derivative
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PM	Positive Middle

PMDC	Permanent Magnet DC
PS	Positive Small
PWM	Pulse Width Modulation
SISO	Single Input Single Output
UV	Ultra-Violet

List of Symbols

t	time
V_a	Armature voltage (V)
R_a	Armature resistance (Ω)
L_a	Armature inductance (H)
I_a	Armature current (A)
e_b	Back emf (V)
ω	Angular speed (rad/s)
T	Motor torque (Nm)
θ	Angular position of rotor shaft (rad)
J_m	Rotor inertia (Kgm^2)
B_m	The viscous friction coefficient (Nms/rad)
K_T	Torque constant (Nm/A)
K_b	The back emf constant (Vs/rad)
K_p	Proportional gain
K_i	Integral gain
K_d	Derivative gain
B	Density of the magnetic field
L	length of conductor
i	Current flowing in the conductor
P	Total input power

Chapter one
Introduction

1.1 Introduction

Dc motor has high reliabilities, flexibilities and low costs, DC motors are widely used in industrial applications, robot manipulators and home appliances where speed and position control of motor are required. PID controllers are commonly used for motor control applications because of their simple structures and intuitionally comprehensible control algorithms. Controller parameters are generally tuned using hand-tuning or Ziegler-Nichols frequency response method [1]. Both of these methods have successful results but long time and effort are required to obtain a satisfactory system response. Two main problems encountered in motor control are the time-varying nature of motor parameters under operating conditions and existence of noise in system loop Analysis and control of complex, nonlinear and/or time-varying systems is a challenging task using conventional methods because of uncertainties. Fuzzy set theory which led to a new control method called Fuzzy Control which is able to cope with system uncertainties. One of the most important advantages of fuzzy control is that it can be successfully applied to control nonlinear complex systems using an operator experiences or control engineering knowledge without any mathematical model of the plant [2].

1.2 Problem Statement

The major problem is the quality of speed control in the DC motors. There are more than one method to control speed but have not perfect outcome so we will use a new technique to improve the results.

1.3 Proposed Solution

A combination of fuzzy logic control and the PID controller can be used to enhance the performance, instead of using PID control alone. The DC motor

shall connect to external circuit to control the speed and the direction, the circuit is controlled using a micro-controller unit.

1.4 Objectives

The project studies the characteristic of DC motor and control the DC motor using fuzzy and PID controller . The objectives are as follows:

1. To built a modeling simulation for DC motor using MATLAB/Simulink.
2. Design circuit to control of DC motor using Fuzzy Logic Controller (FLC).
3. Implementation of fuzzy logic controller on control speed of dc motor to increase response.

1.5 Methodology

Simulation of DC motor controller using MATLAB/Simulink. The design control speed of dc motor by fuzzy logic controller in practical include hardware and software component. Firstly the hardware component arduino, LCD. Software program include arduino and protreus . Reprogramming the arduino by fuzzy logic controller requirements.

1.6 Thesis Organization

The rest of this thesis will be organized as follows. Chapter two is composed of introduction to DC motors, types, components and how to convert electrical to mechanical energy. Chapter three discusses the control circuit and components and compare between the circuit controllers. Chapter four discuss the hardware and software design and implementation and introduces the final outcomes. Finally, Chapter five presents the conclusions and recommendation.

Chapter Two
Background and Literature Review

2.1 Introduction

DC motor is machine that converts DC power into mechanical power. It is operation is based on the principle that when a conductor carry current is placed in a magnetic field, the conductor experiences a mechanical force. The direction of this force is given by Fleming's left hand rule and magnitude is given by;

$$F = Bil \quad \text{(Newton)} \tag{2.1}$$

where B is the density of the magnetic field, l is the length of conductor, and i the value of current flowing in the conductor.

Basically, there is no constructional difference between DC motor and DC generator. The same DC machine can be run as a generator or motor. The direct current machine becomes more popular and more useful in the industry control area for a long time because of its features such as high start torque, high-speed response, portability, and conform with many types of control tuning methods, Nowadays DC motors are used widely in many control applications, including robots, electrics vehicle application, disk drivers, machine tools, and servo-valve actuators. Small brushed DC motors also known as permanent magnet DC (PMDC) motors find their applications in intelligent toys, power tools, robotics, and modern appliances such as printers, scanners, photocopiers, car windows and wiper controls and other industrial applications [3].

2.2 Electromechanical Energy Conversion

An electromechanical energy conversion device is essentially a medium of transfer between an input side and an output side. Three electrical machines (DC, induction and synchronous) are used extensively for electromechanical

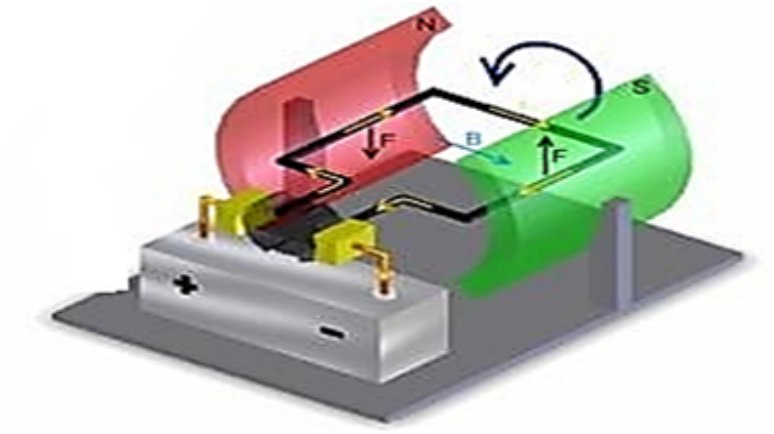

Figure 2.1: Force and Magnetic Field

energy conversion. Electromechanical energy conversion occurs when there is a change in magnetic flux linking a coil, associated with mechanical motion.

2.3 Electric Motor

The input is electrical energy (from the supply source), and the output is mechanical energy (to the load).

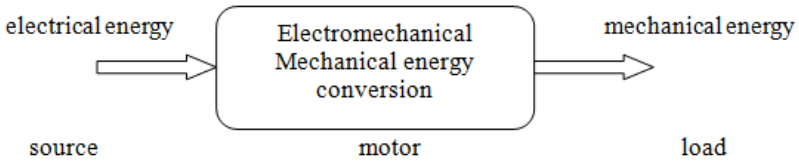

Figure 2.2: Electromechanical

2.4 Construction of DC Motor

DC motors consist of one set of coils, called armature winding, inside another set of coils or a set of permanent magnets, called the stator. Applying a voltage to the coils produces a torque in the armature, resulting in motion.

2.4.1 Stator

The *stator* is the stationary outside part of a motor. The stator of a permanent magnet dc motor is composed of two or more permanent magnet pole pieces,

The magnetic field can alternatively be created by an electromagnet. In this case, a DC coil (field winding) is wound around a magnetic material that forms part of the stator.

2.4.2 Rotor

The *rotor* is the inner part which rotates, The rotor is composed of windings (called armature windings) which are connected to the external circuit through a mechanical commutator. Both stator and rotor are made of ferromagnetic materials. The two are separated by air-gap.

2.4.3 Winding

A *winding* is made up of series or parallel connection of coils. The *armature winding* is the winding through which the voltage is applied or induced. The *field winding* is the winding through which a current is passed to produce flux (for the electromagnet). Windings are usually made of copper.

2.5 DC Motor Basic Principles

2.5.1 Energy Conversion

If electrical energy is supplied to a conductor lying perpendicular to a magnetic field, the interaction of current flowing in the conductor and the magnetic field will produce mechanical force and therefore, mechanical energy.

2.5.2 Value of Mechanical Force

There are two conditions which are necessary to produce a force on the conductor. The conductor must be carrying current, and must be within a magnetic field. When these two conditions exist, a force will be applied to the conductor, which will attempt to move the conductor in a direction perpendicular to the magnetic field. This is the basic theory by which all DC motors operate [4]. The force exerted upon the conductor can be expressed as given by equation (2.1). The direction of motion can be found using Fleming's Left Hand Rule The first finger points in the direction of the magnetic field (first - field), which goes from the North pole to the South pole. The second finger points in the direction of the current in the wire (second - current). The

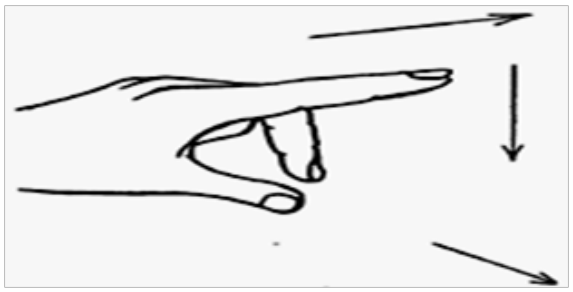

Figure 2.3: Fleming's Left Hand Rule

thumb then points in the direction the wire is thrust or pushed while in the magnetic field (thumb - torque or thrust).

2.6 DC Machine Classification

DC Machines can be classified according to the electrical connections of the armature winding and the field windings. The different ways in which these windings are connected lead to machines operating with different characteristics. The field winding can be either self-excited or separately-excited, that is, the terminals of the winding can be connected across the input voltage terminals or fed from a separate voltage source (as in the previous section). Further, in self-excited motors, the field winding can be connected either in series or in parallel with the armature winding. These different types of connections give rise to very different types of machines, as we will study in this section.

2.6.1 Separately Excited Machines

The armature and field winding are electrically separate from each other. The field winding is excited by a separate DC source. The voltage and power equations for this machine are same as those derived in the previous section.

$$\text{The total input power} = V_f I_f + V_T I_a \tag{2.2}$$

2.6.2 Self Excited Machines

In these machines, instead of a separate voltage source, the field winding is connected across the main voltage terminals.

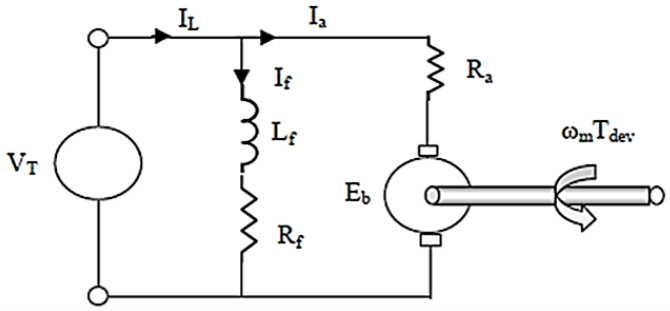

Figure 2.4: Shunt DC machine

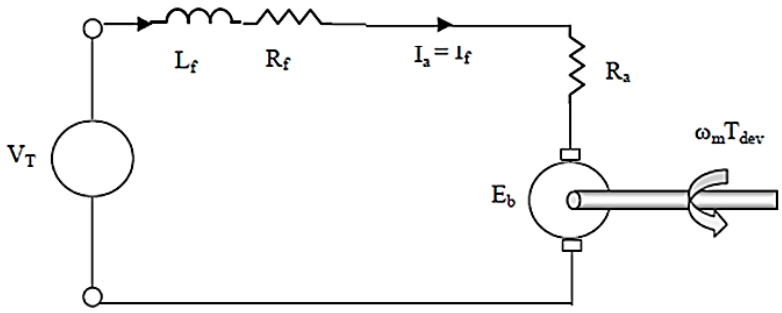

Figure 2.5: Series DC machine

2.6.2.1 Shunt Machine

A shunt DC macine is shown in Figure 2.4. The armature and field winding are connected in parallel. The armature voltage and field voltage are the same. The total current drawn from the supply,

$$I_L = I_f + I_a, \tag{2.3}$$

and the total input power is

$$P = V_T I_L. \tag{2.4}$$

2.6.2.2 Series DC Machine

A series DC macine is shown in Figure 2.5. The field winding and armature winding are connected in series. The field winding carries the same current as the armature winding. A series wound motor is also called a universal motor. It is universal in the sense that it will run equally well using either an ac or a dc voltage source. Reversing the polarity of both the stator and the rotor cancel out. Thus the motor will always rotate the same direction irregardless of the voltage polarity.

2.6.2.3 Compound DC Machine

If both series and shunt field windings are used, the motor is said to be compounded. In a compound machine, the series field winding is connected in series with the armature, and the shunt field winding is connected in parallel. Two types of arrangements are possible in compound motors:

1. Cumulative compounding - If the magnetic fluxes produced by both series and shunt field windings are in the same direction (i.e., additive), the machine is called cumulative compound.

2. Differential compounding - If the two fluxes are in opposition, the machine is differential compound. In both these types, the connection can be either short shunt or long shunt.

2.6.3 Principle of Operation

Consider a coil in a magnetic field of flux density B . When the two ends of the coil are connected across a DC voltage source, current I flows through it. A force is exerted on the coil as a result of the interaction of magnetic field and electric current. The force on the two sides of the coil is such that the coil starts to move in the direction of force.

In an actual DC motor, several such coils are wound on the rotor, all of which experience force, resulting in rotation. The greater the current in the wire, or the greater the magnetic field, the faster the wire moves because of the greater force created.At the same time this torque is being produced, the conductors are moving in a magnetic field. At different positions, the flux linked with it changes, which causes an emf to be induced as $e = d\phi/dt$, as shown in Figure 2.6. This voltage is in opposition to the voltage that causes current flow through the conductor and is referred to as a counter-voltage or back emf.

The value of current flowing through the armature is dependent upon the difference between the applied voltage and this counter-voltage. The current due to this counter-voltage tends to oppose the very cause for its production according to Lenz's law. It results in the rotor slowing down. Eventually, the rotor slows just enough so that the force created by the magnetic field, $F = Bil$, equals the load force applied on the shaft. Then the system moves at constant velocity.

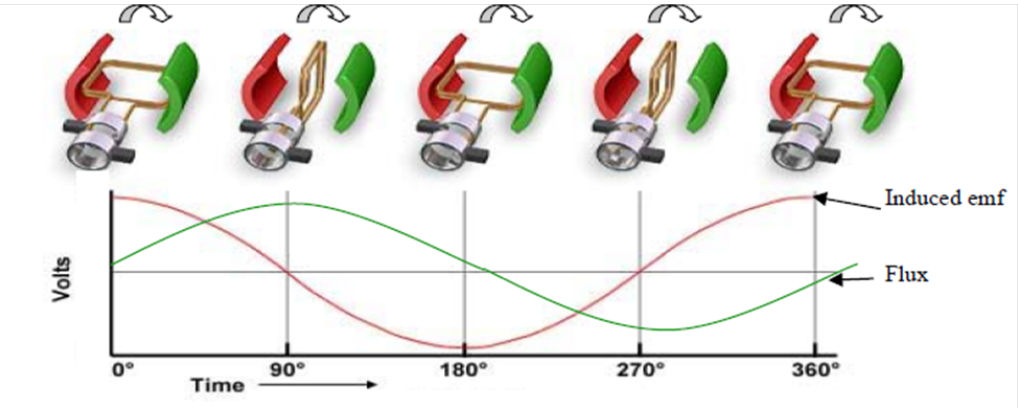

Figure 2.6: Induced voltage in the armature winding of DC motor

2.7 Speed Control Method of DC Motor

The speed of DC motor can be varied by controlling the field flux, the armature resistance or the terminal voltage that applied to the armature circuit (armature voltage). The three most common speed control methods are field resistance control, armature voltage control, and armature resistance control.

2.7.1 Field resistance control method

In the field resistance control method, a series resistance is inserted in the shunt-field circuit of the motor in order to change the flux by controlling the field current. It is theoretically expected that an increase in the field resistance will result in an increase in the load speed of the motor and in the slope of torque speed curve.

2.7.2 Armature voltage control method

In the armature voltage control method, the voltage applied to the armature circuit, is varied without changing the voltage applied to the field circuit of the motor. Therefore, the motor must be separately excited to use armature voltage control. When the armature voltage is increased, the no-load speed of the motor increases while the slope of torque speed curve remains unchanged since the flux is kept constant.

2.7.3 Armature resistance control method

The armature resistance control is the less commonly used method for speed control in which an external resistance is inserted in series with the armature circuit. An increase in the armature resistance results in a significant increase in the slope of the torque speed characteristic of the motor while the no-load speed remains constant.

2.8 DC Motor Model

In armature control of separately excited DC motors, the voltage applied to the armature of the motor is adjusted without changing the voltage applied to the field. Figure 2.7 shows a separately excited DC motor equivalent model

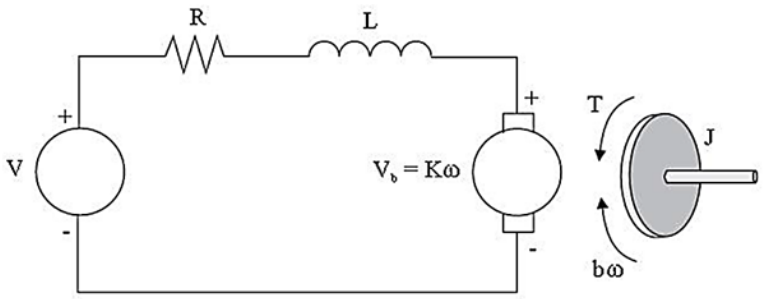

Figure 2.7: DC motor model

$$v_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} + e_b(t) \tag{2.5}$$

$$e_b(t) = K_b w(t) \tag{2.6}$$

$$T_m(t) = K_T i_a(t) \tag{2.7}$$

$$T_m(t) = J_m \frac{dw(t)}{dt} B_m \omega(t) \tag{2.8}$$

where V_a is Armature voltage (V), R_a is the armature resistance (Ω), L_a is the armature inductance (H) , I_a is the armature current (A) e_b is Back emf (V), ω is angular speed (rad/s), T is the motor torque (Nm) θ is angular position of rotor shaft (rad) J_m is Rotor inertia (Kgm^2), B_m is the viscous friction coefficient (Nms/rad), K_T is torque constant (Nm/A), K_b is the back

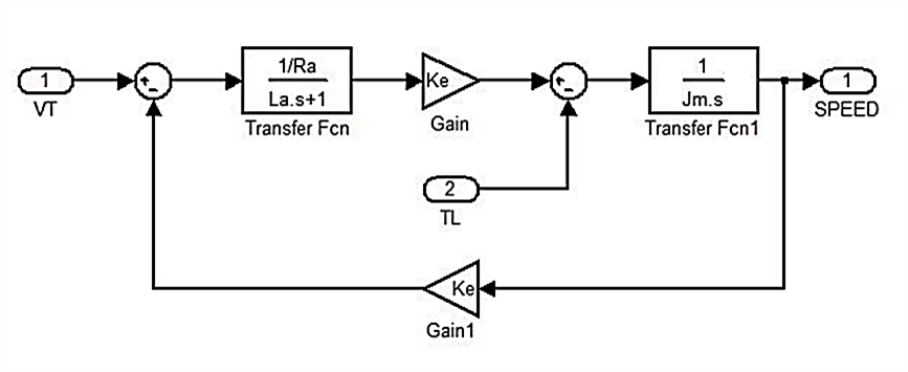

Figure 2.8: Simulink Model

emf constant (Vs/rad). We also have that

$$v_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} + K_b \omega(t), \tag{2.9}$$

$$K_T i_a(t) = J_m \frac{dw(t)}{dt} + B_m \omega(t). \tag{2.10}$$

The Laplace transforms of the previous equations are

$$V_a(s) = R_a I_a(s) + L_a I_a(s) + K_b \omega(s) c \tag{2.11}$$

$$K_T I_a(s) = J_m \omega(s) s + B_m \omega(s) \tag{2.12}$$

$$V_a(s) = \frac{\omega(s)}{K_T \left[L_a J_m s^2 + (R_a J_m + L_a B_m)s + (R_a B_m + K_b K_T)\right]} \tag{2.13}$$

The relation between rotor shaft speed and applied armature voltage by transfer function:

$$\frac{\omega(s)}{V_a(s)} = \frac{K_T}{L_a J_m s^2 + (R_a J_m + L_a B_m)s + (R_a B_m + K_b K_T)} \tag{2.14}$$

The relation between position and speed is:

$$\theta(s) = \frac{1}{s\omega(s)} \tag{2.15}$$

Then the transfer functions between shaft position and armature voltage at no-load is:

$$\frac{\theta(s)}{V_a(s)} = \frac{K_T}{L_a J_m s^3 + (R_a J_m + L_a B_m)s^2 + (K_T K_b + R_a B_m)s} \tag{2.16}$$

Figure 2.8 shows an example Simulink model, while Table 2.1 shows the value of variable of DC motor.

Table 2.1: Values of variables of DC motor

Parameter	Value
Moment of inertia of the rotor	$J = 0.0000885$ Nm/rad/s^2
Damping (friction) of the mechanical system	$b = 0.00130$ Nm/rad/s
Terminal resistance	$R_a = 0.66\,\Omega$
Terminal inductance (L_a)	$L_a = 0.011791815$ H
Electromotive force constant (K)	$K = 0.0941$ Nm/A

2.9 Related Studies

The speed of DC motors can be adjusted within wide boundaries so that this provides easy controllability and high performance. DC motors used in many applications such as still rolling mills, electric trains, electric vehicles, electric cranes and robotic manipulators require speed controller to perform their tasks. Speed controller of DC motors is carried out by means of voltage control in 1981 firstly by Ward Leonard [5]. The regulated voltage sources used for DC motor speed control have gained more importance after the introduction of thyristor as switching devices in power electronics. Then semiconductor components such as MOSFET, IGBT, and GTO have been used as electric switching devices [6]. DC motor systems are indispensable in modern industries. DC motors are used in a variety of applications in industrial electronics and robotics that includes precision positioning as well as speed control. DC motors use feedback controller to control the speed or the position, or both.

Today most famous and most frequently used type of controller in industry is PID controller [7], but PID controllers don't offer satisfactory results when adaptive algorithm are required [8, 9]. Fuzzy Logic Controller offers some solutions. Basic advantages of Fuzzy Logic Controller is that it does not require knowing complete mathematical model of system [10–13]. Popularity of FLC is explained with fact that it puts clear and simple implementation of human thinking into controlling algorithm [14]. Fuzzy controllers are robust regarding dynamic changes and have wide stability range [15]. FLC only based on approximate and linguistic information [10,11,16]. The basic continuous feedback controller is PID controller which possesses good performance. However is adaptive enough only with flexible tuning. Although many advanced con-

trol techniques such as self-tuning control, model reference adaptive control, sliding mode control and fuzzy logic control have been proposed to improve system performances. In this project, Fuzzy Logic Controller has been proposed for improvement and analysis of the system performance. The study in [17] reviewed the control DC motor is modeled using MATLAB/Simulink. Using both PID controller and fuzzy logic techniques, the results are compared for different speed values.

The study in [18] reviewed the design the control to speed and direction control of brushed DC motor using a microcontroller has been developed successfully. The speed control of the DC motor is achieved using standard PWM techniques in a sensor-less closed loop feedback control fashion. Motor functions such as start-stop, forward and reverse direction and speed increment and decrement are controlled through keyboard commands. The control software for the microcontroller is written in Assembly language. The present paper discusses the experimental system and the test results.

Also, the study in [19] reviewed the design circuit to control of dc motor by using PWM. Speed control of DC motor is very crucial in application where required speed is precision and correcting signal representing and to operate motor at constant speed, so we used PWM method which are fulfill all requirements to speed control of DC motor. PWM based speed control system consists of electronic components (integrated circuits, Sensors etc.).

In [20], to control the speed of DC motor using Pulse Width Modulation (PWM) method. Microcontroller AT89S52 is used to generate PWM. L293D IC is used to drive the motor which is made up of two H-Bridge. 555 IC is used with optocoupler to sense the speed of DC motor. Rectifier circuit is used for power supply to circuit and motor. The study in [21], shows that precise and accurate control of small DC motors can be done efficiently without using costly components and complicated circuit.

Chapter Three
Fuzzy PID Control System Model

3.1 Proposed Control System

This chapter discuses the components of the control circuit, used in the project and discuses the PID controller (type and tuning method), fuzzy logic, Arduino, required LED and sensors.

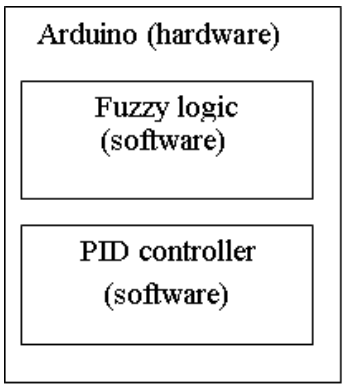

Figure 3.1: A combined Fuzzy PID Controller

3.2 PID Controller

A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems. A PID is the most commonly used feedback controller. A PID controller calculates an ”error” value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs. In the absence of knowledge of the underlying process, PID controllers are the best controllers. 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.

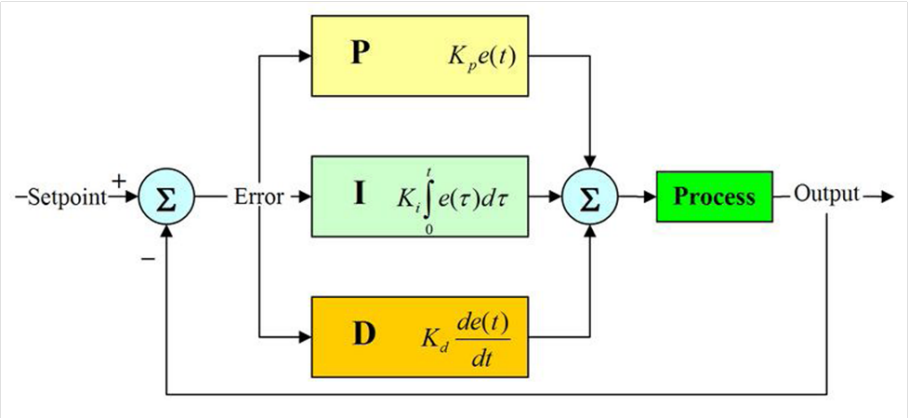

Figure 3.2: Block Diagram of PID Controller

The PID controller calculation (algorithm) involves three separate parameters, and is accordingly sometimes called 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 such as the position of a control valve or the power supply of a heating element. Heuristically, these values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change. By tuning the three constants in the PID controller algorithm, 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. Note that the use of the PID algorithm for control does not guarantee optimal control of the system or system stability.

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. PI controllers are fairly common, since derivative action is sensitive to measurement noise, whereas the absence of an integral value may prevent the system from reaching

its target value due to the control action. More detail on PID control theory can be found in [22] .

3.2.1 PID Controller Theoretical Model

The PID controller is probably the most-used feedback control design. If $u(t)$ is the control signal sent to the system, $y(t)$ is the measured output and $r(t)$ is the desired output, and tracking error $e(t) = r(t)-y(t)$, a PID controller has the general form

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d(e(t))}{dt} \tag{3.1}$$

where the desired closed loop dynamics is obtained by adjusting the three parameters K_p , K_i and K_d, often iteratively by "tuning" and without specific knowledge of a plant model. Stability can often be ensured using only the proportional term. The integral term permits the rejection of a step disturbance (often a striking specification in process control). The derivative term is used to provide damping or shaping of the response. PID controllers are the most well established class of control systems: however, they cannot be used in several more complicated cases, especially if MIMO systems are considered. Applying Laplace transformation results in the transformed PID controller equation as

$$u(s) = K_p e(s) + K_i \frac{1}{s} e(s) + K_d s e(s) \tag{3.2}$$

$$= \left(K_p + K_i \frac{1}{s} + K_d s\right) e(s) \tag{3.3}$$

with the PID controller transfer function

$$c(t) = K_p + K_i \frac{1}{s} + K_d s \tag{3.4}$$

In other words, The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). Hence:

$$\mathrm{MV} = P_{out} + I_{out} + D_{out} \tag{3.5}$$

where P_{out}, I_{out}, and D out are the contributions to the output from the PID controller from each of the three terms, as defined in the following subsections.

3.2.1.1 Proportional Term

The proportional term (sometimes called gain) 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 a constant K_p, called the *proportional gain*. The proportional term is given by:

$$P_{out} = K_p e(t) \tag{3.6}$$

where P_{out} is the Proportional term of output, K_p is the proportional gain, a tuning parameter, $e = SP–PV$ represents error and t is instantaneous time (the present).

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error, and a less responsive (or sensitive) controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

In the absence of disturbances, pure proportional control will not settle at its target value, but will retain a steady state error (known as droop) that is a function of the proportional gain and the process gain. Specifically, if the process gain – the long-term drift in the absence of control, such as cooling of a furnace towards room temperature, which is denoted by G and assumed to be approximately constant in the error, then the droop is when this constant gain equals the proportional term of the output, P_{out}, which is linear in the error, $G = K_p e$, so $e = G/K_p$. This is when the proportional term, which is pushing the parameter towards the set point, is exactly offset by the process gain, which is pulling the parameter away from the set point. If the process gain is down, as in cooling, then the steady state will be below the set point, hence the term ”droop”.

Only the drift component (long-term average, zero-frequency component) of process gain matters for the droop – regular or random fluctuations above or below the drift cancel out. The process gain may change over time or in the presence of external changes, for example if room temperature changes, cooling may be faster or slower. Droop is proportional to process gain and inversely proportional to proportional gain, and is an inevitable defect of purely proportional control. Droop can be mitigated by adding a bias term (setting the setpoint above the true desired value), or corrected by adding

an integration term (in a PI or PID controller), which effectively computes a bias adaptively. Despite the droop, both tuning theory and industrial practice indicate that it is the proportional term that should contribute the bulk of the output change.

3.2.1.2 Integral Term

The contribution from the integral term (sometimes called reset) is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (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 integral term is given by:

$$I_{out} = K_i \int_0^t e(\tau)d\tau \tag{3.7}$$

where I_{out} is the integral term of output, K_i: Integral gain, a tuning parameter, $e = SP–PV$ represents error and t is instantaneous time (the present) and τ is a dummy integration variable.

The integral term (when added to the proportional term) accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the setpoint value (cross over the setpoint and then create a deviation in the other direction). For further notes regard ingintegral gain tuning and controller stability, see the section on loop tuning

3.2.1.3 Derivative Term

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 term (sometimes called rate) to the overall control action is termed the derivative gain, K_d. The derivative term is given by

$$D_{out} = K_d \frac{d(e(t))}{dt} \tag{3.8}$$

where D_{out} is the derivative term of output, K_d is the derivative gain, a tuning parameter, $e = SP–PV$ represents error and t is instantaneous time (the present).

The derivative noticeable close term slows the rate of change of the controller output and this effect is most to the controller setpoint. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability. However, differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term, and can cause a process to become unstable if the noise and the derivative gain are sufficiently large. Hence an approximation to a differentiator with a limited bandwidth is more commonly used. Such a circuit is known as a *Phase-Lead compensator*.

3.2.1.4 PID Output

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:

$$u(t) = MV + K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d(e(t))}{dt} \tag{3.9}$$

where the tuning parameters are stated as follows.

1. The first tuning parameter is the *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.

2. The second tuning parameter is the *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.

3. The third tuning parameter is the *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. Table 3.1 showing the Effect of increasing parameter independently

Table 3.1: Effect of increasing parameters independently

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
K_p	Decrease	Increase	Small Change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor Change	Decrease	Decrease	No Effect	Improve if K_d small

3.2.2 Loop Tuning

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. Stability (bounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and some desiderata conflict. Further, 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 (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. PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the limitations of PID control. There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject of patents; this section describes some traditional manual methods for loop tuning

3.2.2.1 Stability

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by excess

gain, particularly in the presence of significant lag. Generally, stability of response (the reverse of instability) is required and the process must not oscillate for any combination of process conditions and setpoints, though sometimes marginal stability (bounded oscillation) is acceptable or desired.

3.2.2.2 Optimum behavior

The optimum behavior on a process change or setpoint change varies depending on the application. Two basic desiderata are regulation (disturbance rejection – staying at a given setpoint) and command tracking (implementing setpoint changes) – these refer to how well the controlled variable tracks the desired value. Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the setpoint if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new setpoint.

3.3 Tuning Methods

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing P, I, and D based on the dynamic model parameters. Manual tuning methods can be relatively inefficient, particularly if the loops have response times on the order of minutes or longer. The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and on 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. Table 3.2 discusses the pros/cons of choosing a tuning method.

3.3.1 Open Loop Method

In these methods, the PID is being tuned in open loop, isolated from the process plant. First a step input is applied to the plant and the process reaction curve is obtained. Using the process reaction curve with one of the First Order Plus Dead Time (FOPDT) estimation methods an approximation of the process is calculated. Knowing K_m, τ_m and t_d the PID parameters can

Table 3.2: Choosing a Tuning Method

Method	Advantages	Disadvantages
Manual Tuning	No math required, Online	Requires experienced personnel
Ziegler-Nichols	Proven Method, Online	Process upset, some trial-and-error, very aggressive tuning
Cohen-Coon	Good process models Some math; offline	only good for first-order processes
Software Tools	Consistent tuning; online or offline - can employ computer-automated control system design (CAutoD) techniques;	Some cost or training involved

be evaluated from the related correlations according to the method used. First Order Plus Dead Time (FOPDT) is given by

$$G(s) = \left(\frac{K_m}{\tau_m s} + 1\right) e - t_d s \tag{3.10}$$

3.3.2 Ziegler-Nichols Open Loop Method

In the 1940's, Ziegler and Nichols devised two empirical methods for obtaining controller parameters. Their methods were used for first order plus dead time situations, and involved intense manual calculations. With improved optimization software, most manual methods such as these are no longer used. However, even with computer aids, the following two methods are still employed today, and are considered among the most common. This method remains a popular technique for tuning controllers that use proportional, integral, and derivative actions. The Ziegler-Nichols open-loop method is also referred to as S-shaped curve method, because it tests the open-loop reaction of the process to a change in the control variable output. This basic test requires that the response of the system be recorded, preferably by a plotter or computer. Once certain process response values are found, they can be plugged into the Ziegler-Nichols equation with specific multiplier constants for the gains of a controller with either P, PI, or PID actions. In this

Table 3.3: Open-loop Calculation of (K_p, T_i, T_d)

Controller Type	K_p	T_i	T_d
P	$\frac{X_0}{K_m}\frac{\tau_m}{T_d}$	∞	0
PI	$0.9\frac{X_0}{K_m}\frac{\tau_m}{T_d}$	$3.3t_d$	∞
PID	$1.2\frac{X_0}{K_m}\frac{\tau_m}{T_d}$	$2t_d$	$0.5t_d$

method, we obtain experimentally the open loop response of the FOPDT to a unit step input. This method only applied if the response to a step input exhibits an s-shaped curve as shown in figure . This means that if the plant involves integrators (like 2nd order prototypes system) or complex-conjugate poles (general 2nd order system), then this method can't be applied since s-shaped will not be obtained. This method remains a popular technique for tuning controllers that use proportional, integral, and derivative actions. The Ziegler-Nichols open-loop method is also referred to as a process reaction method, because it tests the open-loop reaction of the process to a change in the control variable output.

The Tuning Procedure to use the Ziegler-Nichols open-loop tuning method, you must perform the following steps

1. Make an open loop step test

2. From the process reaction curve determine the transportation lag or dead time, t_d, the time constant or time for the response to change, τ_m, and the ultimate value that the response reaches at steady-state, K_m, for a step change of X_0.

3. Determine the loop tuning constants. Plug in the reaction rate and lag time values to the Ziegler-Nichols open-loop tuning equations for the appropriate controller (P, PI, or PID) to calculate the controller constants. Use table 3.3 for open loop calculation of the parameters.

3.3.3 Cohen-Coon Method

The Cohen-Coon tuning rules are suited to a wider variety of processes than the Ziegler-Nichols tuning rules. The Ziegler-Nichols rules work well only on processes where the dead time is less than half the length of the time constant. The Cohen-Coon tuning rules work well on processes where the dead

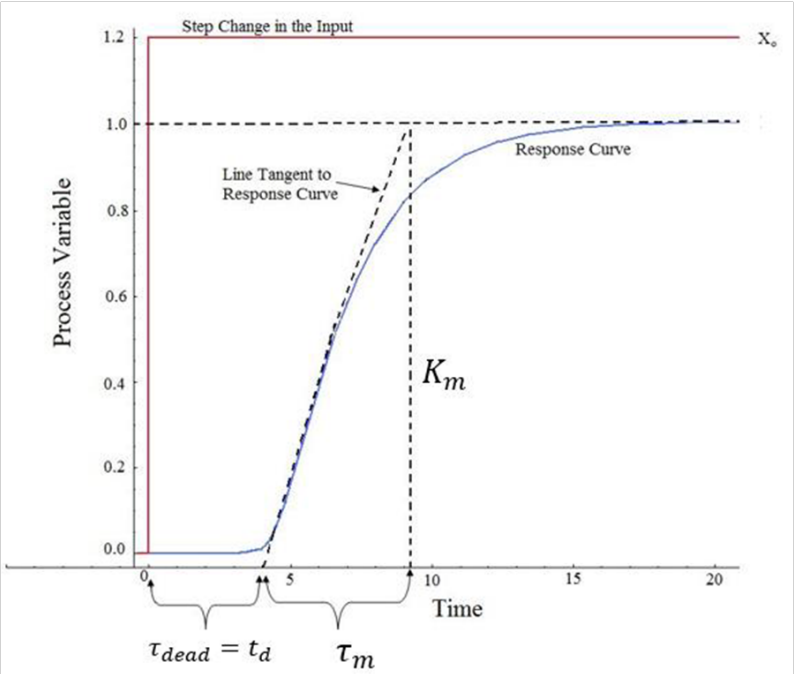

Figure 3.3: Open Loop of first order system plus dead Time (s-shaped curve)

time is less than two times the length of the time constant (and you can stretch this even further if required). Also it provides one of the few sets of tuning rules that has rules for PD controllers. Like the Ziegler-Nichols tuning rules, the Cohen-Coon rules aim for a quarter-amplitude damping response. Although quarter-amplitude damping-type of tuning provides very fast disturbance rejection, it tends to be very oscillatory and frequently interacts with similarly-tuned loops. Quarter-amplitude damping-type tuning also leaves the loop vulnerable to going unstable if the process gain or dead time doubles in value. In this method the process response curve is obtained first, by an open loop test a and then the process dynamics is approximated by a first order plus dead time model, with following parameters:

$$\tau_m = \frac{3}{2}(t_2 - t_1) \tag{3.11}$$

$$t_d = t_2 - \tau_m \tag{3.12}$$

Again the particular rules for this method are used to calculate the PID parameters are listed in table 3.4

3.3.4 Ziegler-Nichols Closed - Loop Tuning Method

The Ziegler-Nichols closed-loop tuning method allows you to use the critical gain value, K_{cr}, and the critical period of oscillation, P_{cr}, to calculate K_p . It

Table 3.4: The parameters of Cohen-Coon method

Controller Type	K_p	T_i	T_d
P	$\frac{\tau_m}{Kt_d}\left(1+\frac{t_d}{3\tau_m}\right)$	-	-
PI	$\frac{\tau_m}{Kt_d}\left(0.9+\frac{t_d}{12\tau_m}\right)$	$t_d\left(\frac{30+\frac{3t_d}{\tau_m}}{9+\frac{20t_d}{\tau_m}}\right)$	-
PD	$\frac{\tau_m}{Kt_d}\left(1.25+\frac{t_d}{6\tau_m}\right)$	-	$t_d\left(\frac{6-\frac{2t_d}{\tau_m}}{22+\frac{3t_d}{\tau_m}}\right)$
PID	$\frac{\tau_m}{Kt_d}\left(1.33+\frac{t_d}{4\tau_m}\right)$	$t_d\left(\frac{32+\frac{6t_d}{\tau_m}}{13+\frac{8t_d}{\tau_m}}\right)$	$t_d\left(\frac{4}{11+\frac{2t_d}{\tau_m}}\right)$

is a simple method of tuning PID controllers and can be refined to give better approximations of the controller. You can obtain the controller constants K_p , T_i , and T_d in a system with feedback. The Ziegler-Nichols closed-loop tuning method is limited to tuning processes that cannot run in an open-loop environment.

Determining the ultimate gain value, K_{cr} is accomplished by finding the value of the proportional-only gain that causes the control loop to oscillate indefinitely at steady state. This means that the gains from the I and D controller are set to zero so that the influence of P can be determined. It tests the robustness of the K_p value so that it is optimized for the controller. Another important value associated with this proportional-only control tuning method is the critical period (P_{cr}). The ultimate period is the time required to complete one full oscillation while the system is at steady state. These two parameters, K_{cr} and P_{cr}, are used to find the loop-tuning constants of the controller (P, PI, or PID). To find the values of these parameters, and to calculate the tuning constants, use the following procedure. The Tuning Procedure:

- Remove integral and derivative action. Set integral time T_i to ∞ or its largest value and set the derivative controller T_d to zero.
- Create a small disturbance in the loop by changing the set point. Adjust the proportional, increasing and/or decreasing, the gain until the oscillations have constant amplitude.
- Record the gain value (K_{cr}) and period of oscillation (P_{cr}).
- Plug these values into the Ziegler-Nichols closed loop equations and

determine the necessary settings for the controller.

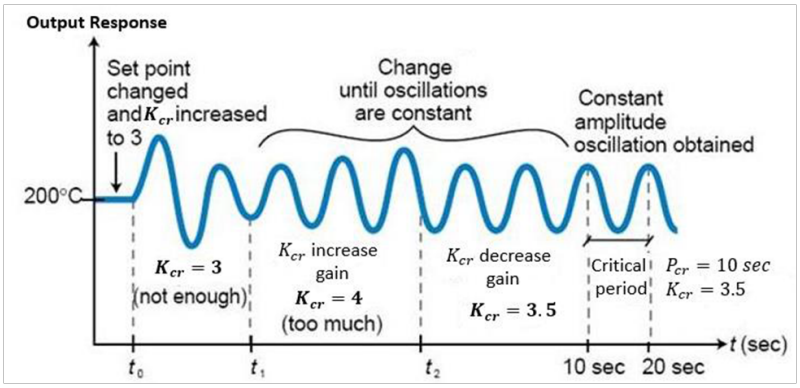

Figure 3.4: System tuned using the Ziegler-Nichols closed-loop tuning method

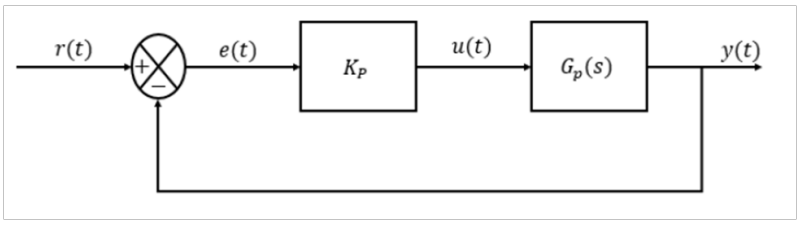

Figure 3.5: The Control system with Gain K_p

These values can be found from the crossing points of the root locus branches with the jw axis. This method doesn't apply if the root locus doesn't cross the jw axis. Advantages of the Ziegler-Nichols closed-loop tuning methods are as follows.

- Easy experiment; only need to change the P controller
- Includes dynamics of whole process, which gives a more accurate picture of how the system is behaving

The disadvantages of the Ziegler-Nichols Closed-Loop tuning methods are as follows.

- Includes dynamics of whole process, which gives a more accurate picture of how the system is behaving
- Experiment can be time consuming
- Can venture into unstable regions while testing the P controller, which could cause the system to become out of control

Table 3.5: Closed-Loop calculation of (K_p, T_i, T_d)

controller type	Kp	ki	Kd
P	$0.5K_{cr}$	∞	0
PI	$K_{cr}/2.2$	$P_{cr}/1.2$	∞
PID	$K_{cr}/1.7$	$P_{cr}/2$	$P_{cr}/8$

3.3.5 Software Method (PID Tuning Toolbox In MATLAB)

PID tuning is the process of finding the values of proportional, integral, and derivative gains of a PID controller to achieve desired performance and meet design requirements. PID controller tuning appears easy, but finding the set of gains that ensures the best performance of your control system is a complex task. Traditionally, PID controllers are tuned either manually or using rule-based methods. Manual tuning methods are iterative and time-consuming, and if used on hardware, they can cause damage. Rule-based methods also have serious limitations: they do not support certain types of plant models, such as unstable plants, high-order plants, or plants with little or no time delay. You can automatically tune PID controllers to achieve the optimal system design and to meet design requirements, even for plant models that traditional rule-based methods cannot handle well. An automated PID tuning workflow involves:

- Identifying plant model from input-output test data .
- Modeling PID controllers in MATLAB using PID objects or in Simulink using PID Controller blocks
- Automatically tuning PID controller gains and fine-tune your design interactively
- Tuning multiple controllers in batch mode
- Tuning single-input single-output PID controllers as well as multi loop PID controller architectures

3.4 Fuzzy Logic

Fuzzy systems are an alternative to traditional notions of set membership and logic that has its origins in ancient Greek philosophy, and applications at the

Table 3.6: Difference between conventional logic and fuzzy logic

Boolean and Conventional Logic	Fuzzy Logic
Uses sharp distinctions. Exp: max height for short = 170 cm. 170.5 cm is considered tall.	No distinct values. Exp: John is very tall
Draw lines between members of a class & non-members.	Do not distinguish members of a class & non-members.
Can be absurd. Is 170 cm really short?	Avoids absurdities. Reflects on how people think. Creates a model based on sense of words. Decision making. Common sense.
Has only two values (0 or 1).	Has an extended range of values between 0 and 1.

leading edge of Artificial Intelligence. Yet, despite its long-standing origins, it is a relatively new field.

Fuzzy logic is a multi-valued logic which deals with vague and indecisive ideas. It has been described as an extension to the conventional Aristotelian and Boolean logic as it deals with ”degrees of truth” rather than absolute values of ”0 and 1” or ”true/false”. Fuzzy logic is not like a computer software which understands only binary functions or concrete values like 1.5, 2.8, etc; instead, it is similar to human thinking and interpretation and gives meaning to expressions like ”often”, ”smaller” and ”higher”. Fuzzy logic takes into account that real world is complex and there are uncertainties; everything cannot have absolute values and follow a linear function [22].

3.4.1 Advantage and disadvantages of fuzzy logic

Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making. It looks into all shades of gray and answers uncertainties and ambiguities. created by human language where everything cannot be described in precise and discrete terms. Fuzzy systems help define disease extent and severity and answer ques-

tions related to individual patients taking into account their risk factors and co-morbidities. On the other hand, it has a number of disadvantages too. It is tedious to develop fuzzy rules and membership functions and fuzzy outputs can be interpreted in a number of ways making analysis difficult. In addition, it requires lot of data and expertise to develop a fuzzy system. It does not give generalizable results and the program has to be run for each individual patient. Therefore, its clinical applicability and utilization is difficult without the availability of preprogrammed soft ware's for different pathologies and the basic training of clinicians to use these programs.

3.4.2 Fuzzy expert system

A fuzzy expert system as show in figure 3.6: is a form of artificial intelligence that uses a collection of membership functions (fuzzy logic) and rules (instead of Boolean logic) to reason about data [23].

3.4.3 Rules of fuzzy expert system

The rules in a fuzzy expert system are usually of a form similar to this:

```
If x is low and y is high then z=medium
```

where x and y are input variables(names for noun data values), z is an output variable(a name for a data value to be computed), `low` is a membership function (fuzzy subset) defined on x, `high` is a membership function defined on y, and `medium` is a membership function defined on z.

3.4.4 Fuzzy Logic Controller

Fuzzy Logic is a form of logical reasoning that can be incorporated into automation systems typically human reasoning schemes. Fuzzy theory was first proposed and investigated by Prof Zadeh in 1965 [20]. One of the main features of fuzzy logic is its ability to operate with vague or ambiguous concepts typical of qualitative reasoning, based on a mathematical support quantitative conclusion can be drawn from a set of observations and qualitative rules. Fuzzy logic control is the application of fuzzy inference process automation. A typical fuzzy controller infers the consequent of more or less large simple rules, this process of reasoning can be performed in parallel, yielding the result with a simple logical sum. This parallel processing capability allows even

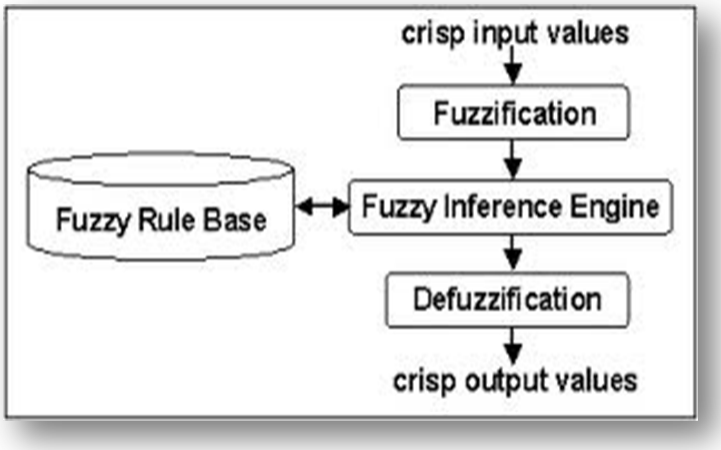

Figure 3.6: Fuzzy logic controller

relatively complex controllers to perform the fuzzy inference in a minimum computation time.

In a fuzzy logic system, the inference mechanism decides what rules to apply for the corresponding inputs by matching the fuzzified inputs to the premises of the rules in the rule base. The inference mechanism provides a fuzzy set that indicates the certainty that the plant input should take the various values. The defuzzification is used to convert the fuzzy set produced by the inference mechanism into a crisp output to be used by the plant . The most important specifications of fuzzy logic control method are their fuzzy logical ability in the quality perception of system dynamics and the application of these quality ideas simultaneously for control system. A simple block diagram of a fuzzy logic system is shown in Figure 3.7

3.4.4.1 Fuzzy Input

The inputs are most often hard or crisp measurement from some measuring equipment is converted into fuzzy values for each input fuzzy set with the fuzzification block.

3.4.4.2 Fuzzification

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

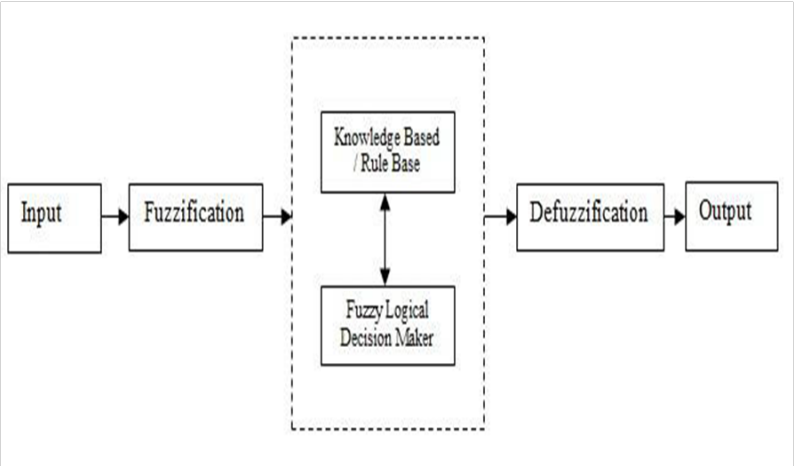

Figure 3.7: Block diagram of a fuzzy logic system

function of fuzzification, which converts input data into suitable linguistic values that may be viewed as labels of fuzzy sets.

3.4.4.3 Knowledge based / rule based

The collection of rules is called a *rule base*. The rules are in "`If Then`" format and formally the `If` side is called the *conditions* and the `Then` side is called the *conclusion*. The computer is able to execute the rules and compute a control signal depending on the measured inputs error (e) and change in error (ce). In a rule based controller the control strategy is stored in a more or less natural language. A rule base controller is easy to understand and easy to maintain for a non-specialist end user and an equivalent controller could be implemented using conventional techniques.

3.4.4.4 Fuzzy Logical

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.

3.4.4.5 Defuzzification

Defuzzification is when all the actions that have been activated are combined and converted into a single non-fuzzy output signal which is the control signal of the system. The output levels are depending on the rules that the systems have and the positions depending on the non-linearities existing to the systems. To achieve the result, develop the control curve of the system representing the I/O relation of the systems and based on the information, define the output degree of the membership function with the aim to minimize the effect of the non-linearity.

3.4.4.6 Fuzzy Output

The output is output gain that can be tuned and also become as an integrator. The output crisp value can be calculated by the centre of gravity or the weighted average.

3.5 Hardware Requirements

The control circuit will use an Arduino board as a control unit. Another hardware items are required such as resistors, push-buttons, sensors and LCD unit. The hardware implementation will employ a DC Motor fan to test the combined fuzzy-PID controller.

3.5.1 Arduino Uno

The Arduino Uno now uses an ATmega16U2 instead of the 8U2 found on the Uno (or the FTDI found on previous generations). This allows for faster transfer rates and more memory. No drivers needed for Linux or Mac (INF file for Windows is needed and included in the Arduino IDE), and the ability to have the Uno show up as a keyboard, mouse, joystick, etc.

The Uno R3 also adds SDA and SCL pins next to the AREF. In addition, there are two new pins placed near the RESET pin. One is the IOREF that allow the shields to adapt to the voltage provided from the board. The other is a not connected and is reserved for future purposes. The Uno R3 works with all existing shields but can adapt to new shields which use these additional pins.

Arduino is an open-source physical computing platform based on a simple I/O board and a development environment that implements the Processing/Wiring language. Arduino can be used to develop stand-alone interactive objects or can be connected to software on your computer (e.g. Flash, Processing, and MaxMSP). The open-source IDE can be downloaded for free (currently for Mac OS X, Windows, and Linux). The Arduino Uno R3 requires the Arduino 1.0 drivers' folder in order to install properly on some computers. We have tested and confirmed that the R3 can be programmed in older versions of the IDE. However, the first time using the R3 on a new computer, you will need to have Arduino 1.0 installed on that machine [6].

- ATmega32 microcontroller
- input voltage - 7-12V
- 4 Digital I/O Pins (6 PWM outputs)
- Analog Inputs
- 32k Flash Memory
- 6Mhz Clock Speed

3.5.2 Sensors

A sensor is a device that detects and responds to some type of input from the physical environment. The specific input could be light, heat, motion, moisture, pressure, or any one of a great number of other environmental phenomena. The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing. Here are a few examples of the many different types of sensors :In a mercury-based glass thermometer, the input is temperature. The liquid contained expands and contracts in response, causing the level to be higher or lower on the marked gauge, which is human-readable. An oxygen sensor in a car's emission control system detects the gasoline/oxygen ratio, usually through a chemical reaction that generates a voltage. A computer in the engine reads the voltage and, if the mixture is not optimal, readjusts the balance. Motion sensors in various systems including home security lights, automatic doors and bathroom fixtures typically send out some

type of energy, such as microwaves, ultrasonic waves or light beams and detect when the flow of energy is interrupted by something entering its path. A photo sensor detects the presence of visible light, infrared transmission (IR), and/or ultraviolet (UV) energy.

3.5.3 LCD Display

This component is specifically manufactured to be used with microcontrollers which means that it cannot be activated by standard IC circuit .it is used for displaying different messages on a miniature liquid crystal display .The model described here is for its low price and great capabilities most frequently used in practice. it can display messages in two lines with 16 characters each .it can display all the letters of alphabet , Greek letters ,punctuation marks, mathematical symbols etc. It is also possible to display symbols made up by the user and it include automatic message shift (left and right) Cursor appearance [23].

3.5.4 DC Motor Fan

With the increase in magnitude of the usage of computers to more than eighteen hours a day, there are additional demands placed on the processor. The processor then starts generating a lot of heat which is caused by the internal electrical energy of the components, and gets warmer the harder the components have to work . Improper maintenance of heat and overheating of the processor can reduce the lifespan or cause irreparable damage to the components in the processor, including components like circuits, microchips, RAM, or hard drives, and makes the computer inoperable. In order to avoid damaging the components and prevent loss of data, it becomes essential to have good cooling equipment like a CPU fan [8].

Chapter Four
Design, Implementation and Discussion

4.1 Introduction

This chapter aims to discuss the hardware and software components, functional block diagrams and discuss the types of the component, their use, and the operation of the circuit. Some cases of the circuit explaining the circuit work and the flow of circuit functions and explain the fuzzy logic and finally explain the difference between PID controller and fuzzy PID.

4.2 Circuit Components

Arduino Uno is basic component in the control circuit. Its content is the code of PID fuzzy logic and receive the signal from the sensor and send the signal to change the speed of DC motor (by using PWM). The Arduino calculates and finds the error. The error changes the value of PID and fuzzy rule. Figure 4.1 below show the Arduino Uno.

The DC motor operates by 12 voltage (CPU computer fan). it consist of four wires (two wires line and ground (connected with adapter 12 V) and a wire gets the speed sensor connected with Arduino, and the final wire takes

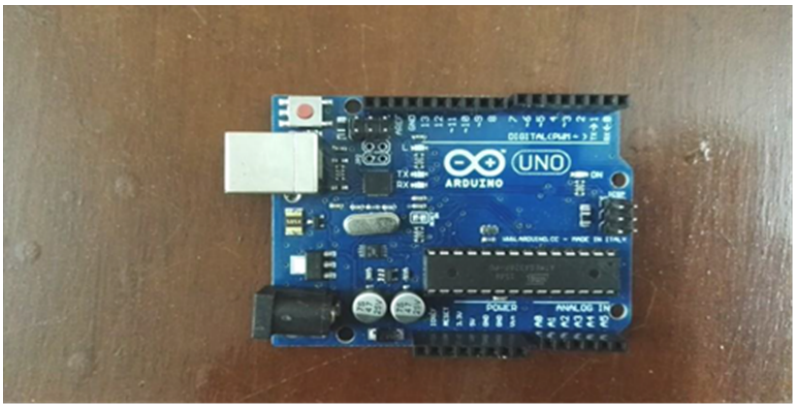

Figure 4.1: Arduino Uno

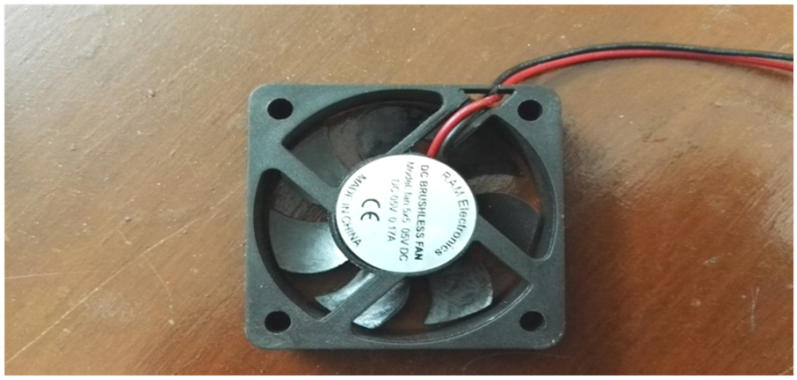

Figure 4.2: CPU fan

the signal to Arduino to change speed by change PWM. the figure 4.2 shows the CPU fan.

A Push button is a type of switch used in the circuit. The circuit contains three Push button switches, that used to increase or decrease the speed and OK switch. The switches one side is connected with resistance (10 KΩ) work pull up resistance and resistances connected with aground and another side connected with Arduino. It is an input device that means give the order to Arduino. In the circuit we need 12V voltage to supply the DC motor in hardware used the adapter to convert the voltage form the main supply to 12V voltage in the software (Protous program) using transistor (IRF 540) to rise up voltage from 5V to 12V to operate the fan.

In the circuit we need to show the result and need to show the speed of motor. An LCD (16×2) is used to show the speed of DC motor and speed required. The LCD in hardware connected with variable resistance to get clear show. The LCD is connected with Arduino to show the speeds (actual speed, required speed). In the circuit we need to connect all these component together to complete the circuit, using male–male wires and used the project board to connect the devices. Finally, Diodes used for the DC motor voltage safety and protection connected in parallel with motor.

4.3 Control Chain Block Diagram

The block diagram in figure 4.10 shows the basic idea of the project. When the speed required is the input of the system , the system is containing of feedback finds the error $e(t)$ fuzzy logic takes the value of the error depending on the error to get the value of (K_i ,K_p, K_d) and change PWM to increase or

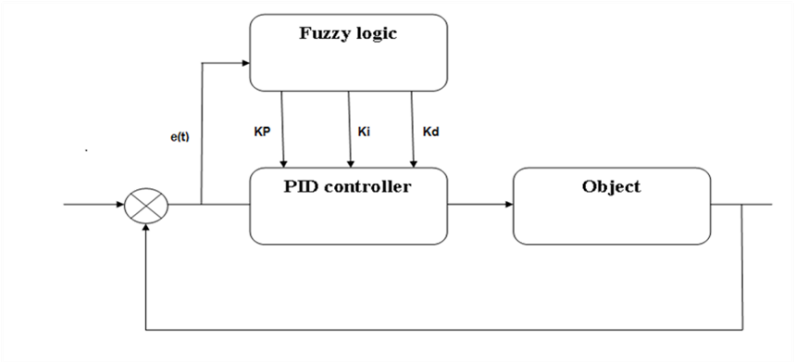

Figure 4.3: block diagram

decrease the speed of DC motor.

4.4 Software Utilization

The Protous program is used to simulate control of the DC motor. The Protous program can show the result of the motor speed. Figure 4.11 shows the basic components (Arduino, LCD, switches, resistance, photodiode, transistor) and shows the connection of the component together, explaining that the LCD is output devise, bush bottom is input device and Arduino the drive and control in DC motor containing of code (fuzzy PID, PWM, Calculate error) and sensor is input device.

4.5 Control Flow Chart

The flow chart explaining the step of control in DC motor shown in figure 4.4. First, choose the speed and press the Ok switch, sending a signal to the Arduino unit to increase or decrease speed. The Arduino determines the error and the fuzzy logic controller classify the error and determines the rule used in this case, and calculates K_p ,K_i, K_d and changes the speed of DC motor. The sensor sends the speed to Arduino and repeats this operation until to give the speed required .

The difference between the PID controller and the fuzzy PID controller is that for PID, the value of K_p ,K_i, K_d is constant value. However, the fuzzy PID controller classify the error and type of the error and takes the value of K_p ,K_i, K_d depending on the fuzzy rules.

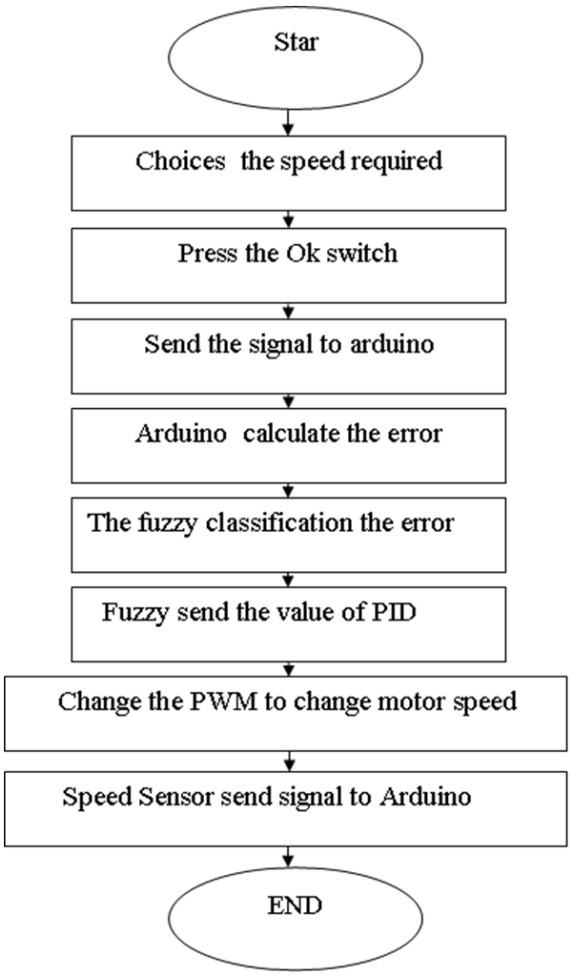

Figure 4.4: flow chart

4.6 Operating The Circuit

The circuit contains three switches the first is to increase or decrease speed and second switch is ok switch, pressing on first switch shown in LED and then the speed required obtained by pressing on the OK switch, which sends signal to Arduino, the Arduino calculates the error (In design, the error range is taken between 50 ,-50), and used PID depend of Ziegler and Nichols proposed rules for determining values of K_c, T_i and T_d based in the transient response characteristics of a given plant. Table below gives experimental tuning rules based on closed loop oscillation method (the value of K_p is between (0-1) , K_i is (0-0.1) and K_d is 0.

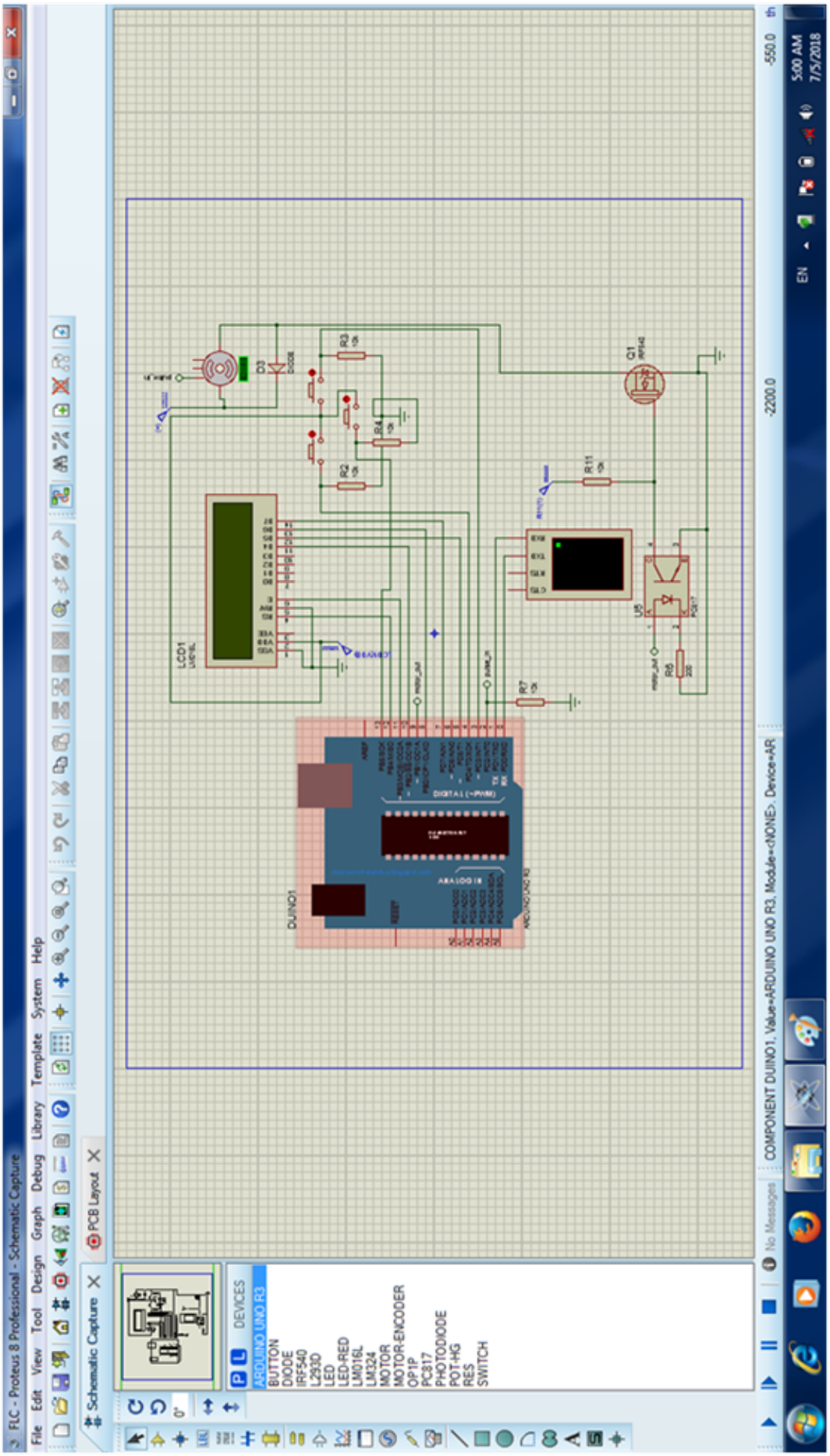

Figure 4.5: circuit software component

Table 4.1: Simulation Parameters: (K_p, T_i, T_d)

Type of Controller	K_p	T_i	T_d
P	$0.5K_c$	∞	0
PI	$0.45K_c$	$0.83T$	0
PID	$0.6K_c$	$6K$	$0.5T$

Table 4.2: Simulation Parameters: definition and range of error variables

Error Variable	Definition	Range
NL	Negative Large	-50 - 20
NM	Negative Medium	-30 - 10
NS	Negative Small	-20 - 0
ZE	Zero	-10 - 10
PS	Positive Small	0 - 20
PM	Positive Medium	10 - 30
PL	Positive Large	30 - 50

The classification of the error is in seven parts depending on the values of the error. When classification the error, the fuzzy controller also classify the value of K_p in the three values. Table 4.3 show that the classification of K_p takes the range between (0-1) and three variables (`small`, `medium`, `large`).

Table 4.3: Simulation Parameters: K_p values classification

K_p Value	Range
`small`	0 - 0.3
`medium`	0.1 - 0.5
`large`	0.3 - 1

When classifying the error and K_p, the fuzzy controller also classify the value of K_i in three values. Table 4.4 shows the classification of K_i takes the range between (0-0.1) and three variables (`small`, `medium`, `large`). To connect between error and K_i , K_d that used fuzzy logic rules (If ... then ...), we take seven rules of logic .

```
1. If Error is PL then Kp is Small and Ki is Small.
```

2. If Error is PM then Kp is Medium and Ki is Medium.

3. If Error is PS then Kp is Big and Ki is Big.

4. If Error is ZE then Kp and Ki are Not change.

5. If Error is NS then Kp is Big and Ki is Big.

6. If Error is NM then Kp is Medium and Ki is Medium.

7. If Error is NL then Kp is Small and Ki is Small.

Table 4.4: Simulation Parameters: K_i values classification

K_i Value	Range
small	0 -0.04
medium	0.02 - 0.08
large	0.06 - 0.1

The Arduino changes the PWM to get the required speed and sensor takes the speed and this operation is repeated till to reach an error equal zero

4.6.1 Case(1)

In case (1) explaining the basic component of the hardware component , and showing the speed of DC motor that means the small value of speed (1204 r.p.m) that can be change the speed in a range between(50,-50 r.p.m).

4.6.2 Case(2)

In case (2), the speed required is 1894 rpm and it can be seen that the achieved motor speed is above than 1894 rpm. When pressing the Ok switch, the speed decreases and the error increase. The fuzzy controller determines the value for K_p and K_i and change the value of speed. The number 1900 rpm in LCD which is the current motor speed, will decrease until te value is the 1894 rpm required speed.

4.6.3 Case(3)

In case (3), the speed required is 1204 rpm, but the achieved motor speed is less (1201 rpm). When pressing the Ok switch the speed decreases and

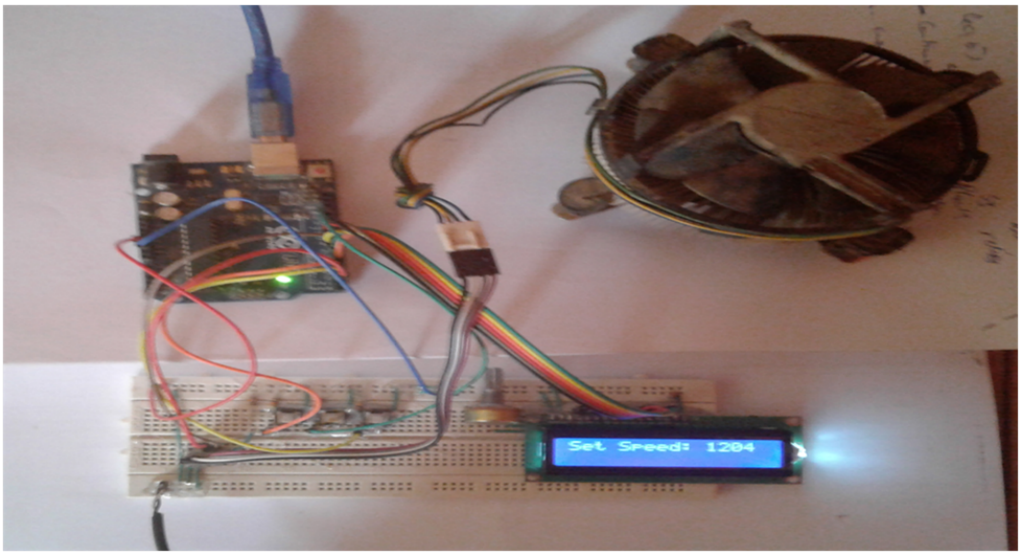

Figure 4.6: Case(1): The set speed is 1204 rpm)

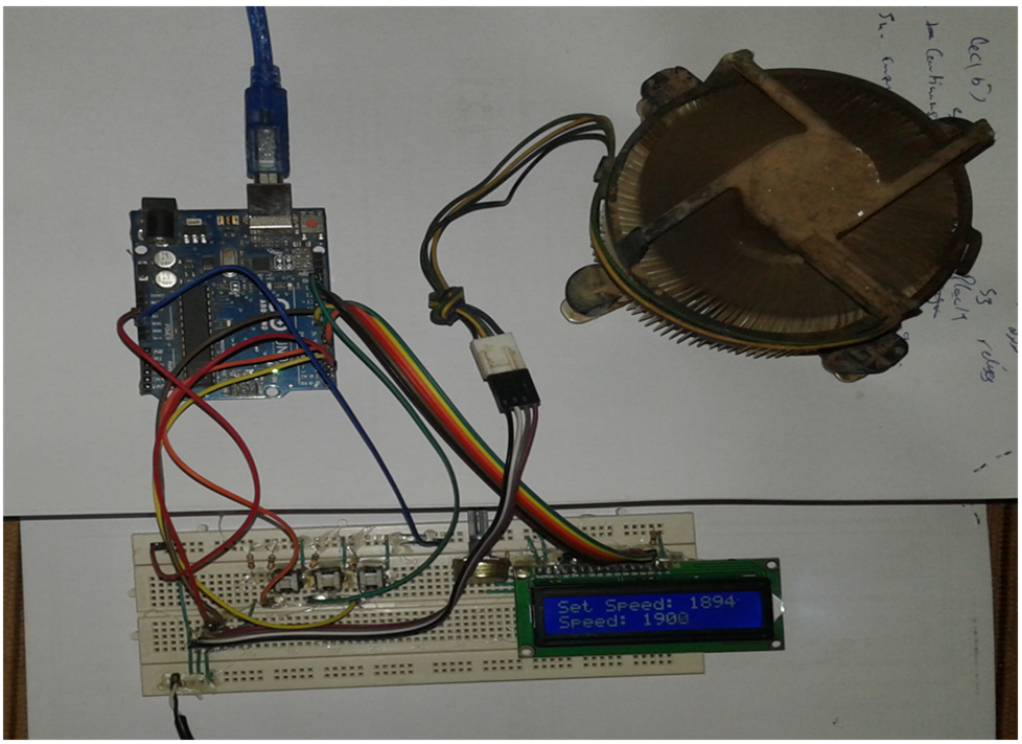

Figure 4.7: Case (2): The achieved speed is 1900 rpm

error decreases. The fuzzy controller determines the value for K_p and K_i and change the value of speed. The speed displayed by the LCD (1201 rpm), is increased until reaching a value of (1204 rpm) required speed.

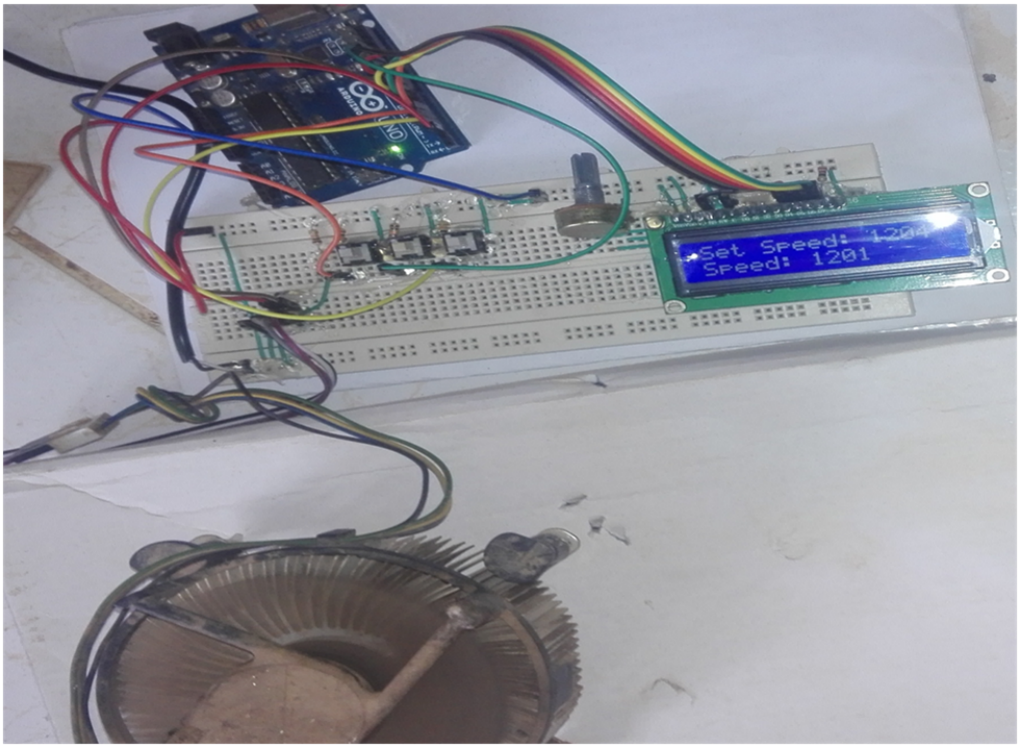

Figure 4.8: Case(3): The achieved speed is 1201 rpm

4.6.4 MATLAB

Matlab is a high performance language for technical computing. It integrates computation visualization, and programming in an easy to use environment where problems and solutions are expressed in familiar mathematical notation.

4.6.4.1 Simulation

Shows the DC motor model built in Simulink. Motor model was converted to a 2-in 2-out subsystem. Input ports are armature voltage (Va) and load torque (T load) and the output ports are angular speed in (w) and position (teta).

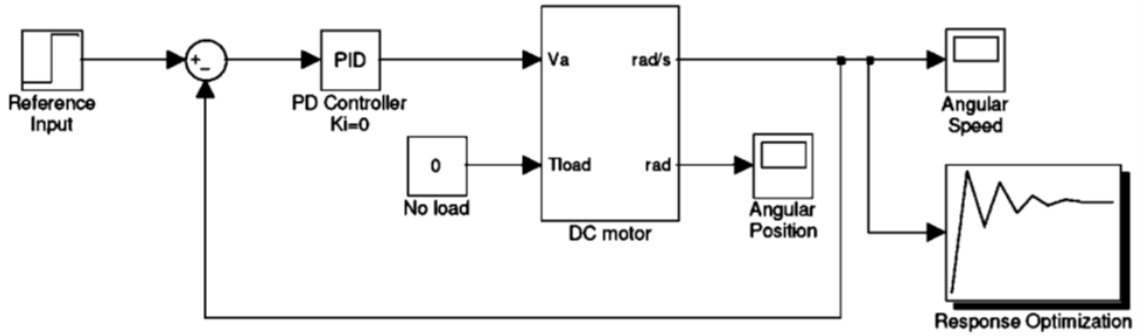

Figure 4.9: Crisp PD control system

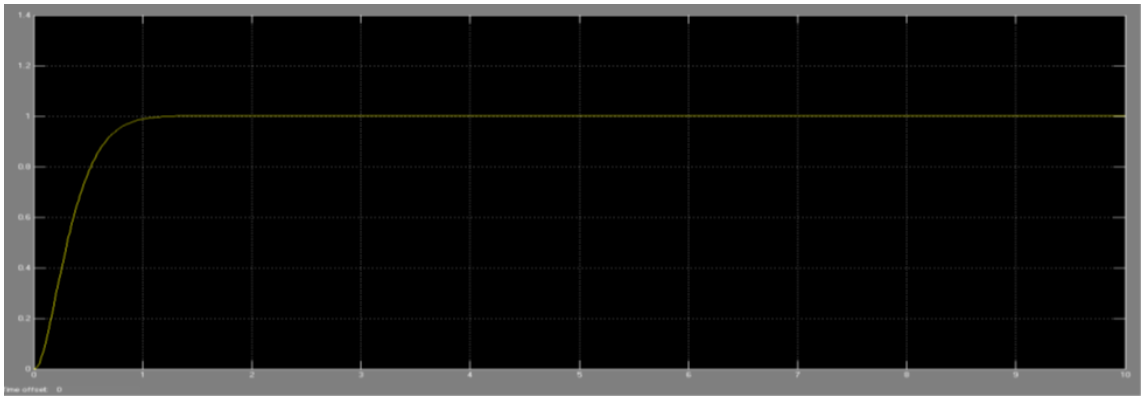

Figure 4.10: The figure below show the output simulation

4.6.4.2 PID Controller

Figure shows the overall simulation circuit for PID controller. PID controller will control input to the Pulse Width Modulation (PWM) terminal at buck converter. At several set points, output from PID controller will generate the different value of duty cycle for PWM. So that, average voltage supply to the voltage terminal of DC motor and speed of DC motor can be varied as shown in Figure4.9

the figure below explain the fuzzy PID controller in DC motor using matlab simulation and the result and explain the benefit of fuzzy PID controller.

The output response of PID fuzzy model of DC motor in matlab/simulink as shown in figure

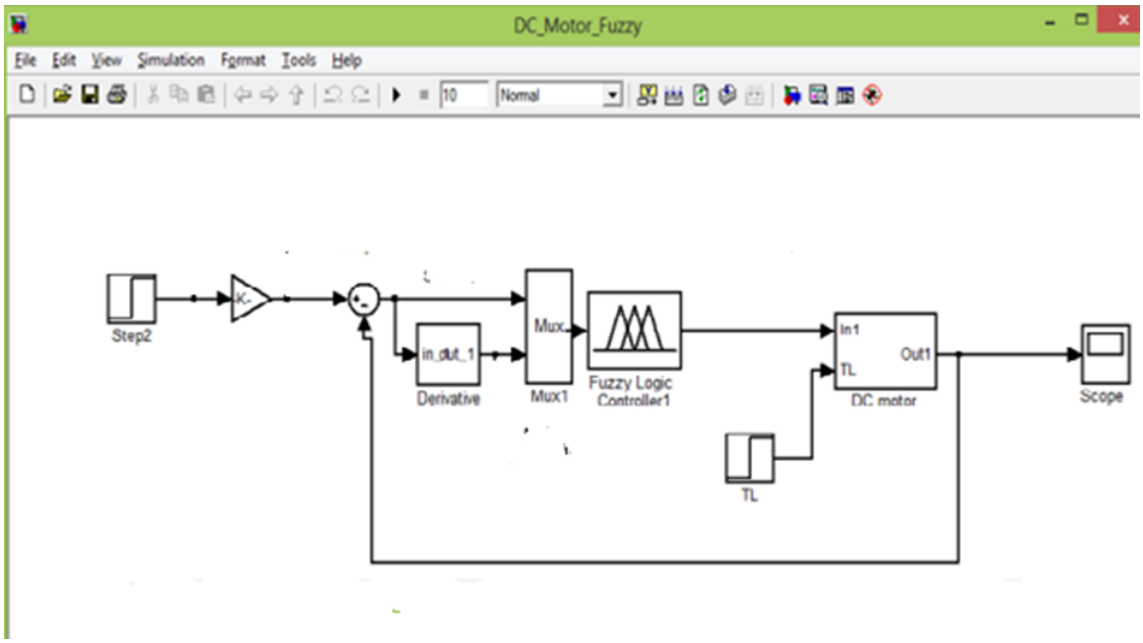

Figure 4.11: PID fuzzy model of DC motor.

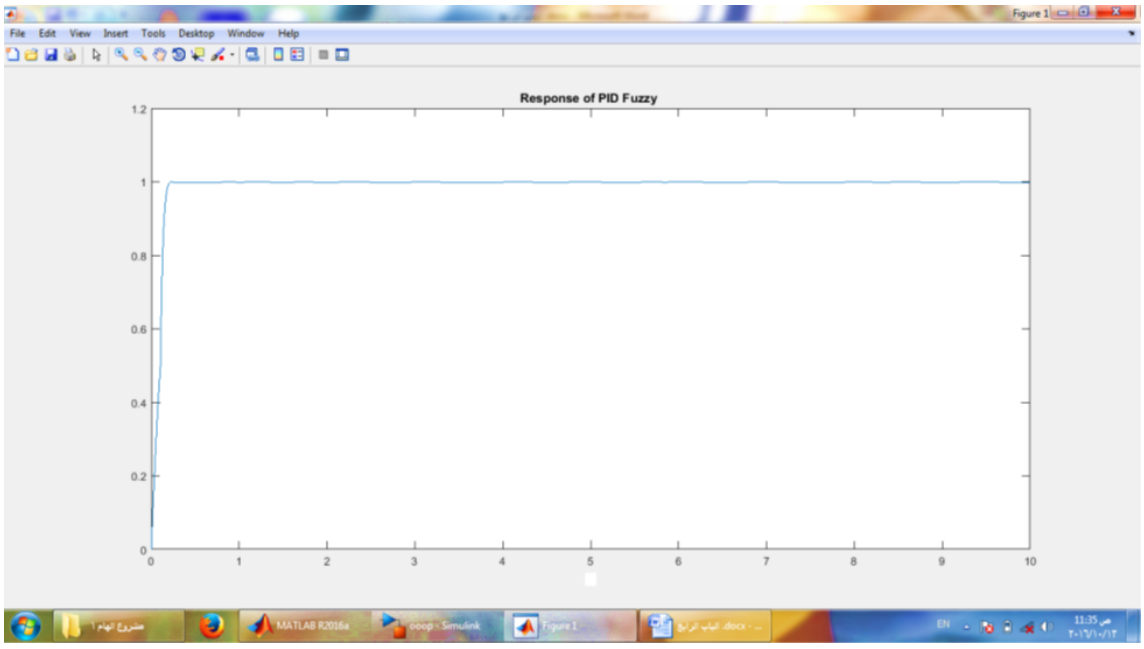

Figure 4.12: Output response of PID fuzzy model of DC motor

Chapter Five
Conclusion and Recommendations

5.1 Conclusions

The automatic control has played a vital role in the advance of engineering and science. This study targeted the problem of enhancement of speed control for the dc motor via a combination of the fuzzy logic and the PID controller. A The problem is formulated and has been modeled by derivation of the mathematical model with differential equations. Simulations were conducted using the Protus software. The theory of operation was tested using a CPU fan controlled by an Arduino unit. The circuit operation was tested for various cases and it was shown that the performance in terms of speed control is better.

5.2 Recommendations

For enhancement for this study, a microcontroller unit with better capabilities can be used to accommodate a larger number of fuzzy rules. Also, for further future enhancement, instead of using fuzzy logic,the following ideas can be investigated.

- The design of self-tuning algorithms based on Artificial Neural Networks (ANNs) to automatically tune the gains of a PID controller
- The problem of implementing a Genetic Algorithm (GA) in determining PID controller parameters for PID-controller tuning optimization
- The problem of designing a knowledge-based expert system to tune the parameters of the PID controller

.

Bibliography

[1] K. Ogata and Y. Yang, *Modern control engineering*. Prentice hall India, 2002, vol. 4.

[2] B. Chalmers, "Influence of saturation in brushless permanent-magnet motor drives," in *IEE Proceedings B-Electric Power Applications*, vol. 139, no. 1. IET, 1992, pp. 51–52.

[3] A. R. Hambley, "Electrical engineering principles and applications," *Chapter*, vol. 6, p. 255, 2009.

[4] G. Rizzoni and J. Kearns, *Principles and applications of electrical engineering*. McGraw-Hill Higher Education, 2004.

[5] C. Chan, "Low-cost electronic-controlled variable-speed reluctance motors," *IEEE transactions on industrial electronics*, no. 1, pp. 95–100, 1987.

[6] A. Khoei and K. Hadidi, "Microprocessor based closed-loop speed control system for dc motor using power mosfet," in *Electronics, Circuits, and Systems, 1996. ICECS'96., Proceedings of the Third IEEE International Conference on*, vol. 2. IEEE, 1996, pp. 1247–1250.

[7] T. Hägglund and K. J. Åström, "Automatic tuning of pid controllers," *The control handbook*, 1996.

[8] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[9] Y. D. Landau, "Adaptive control: The model reference approach," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 1, pp. 169–170, 1984.

[10] L. Reznik, *Fuzzy controllers handbook: how to design them, how they work*. Elsevier, 1997.

[11] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, "Neuro-fuzzy and soft computing; a computational approach to learning and machine intelligence," 1997.

[12] R.-J. Wai, C.-M. Lin, and C.-F. Hsu, "Adaptive fuzzy sliding-mode control for electrical servo drive," *Fuzzy sets and systems*, vol. 143, no. 2, pp. 295–310, 2004.

[13] R.-J. Wai, "Robust fuzzy neural network control for nonlinear motor-toggle servomechanism," *Fuzzy sets and systems*, vol. 139, no. 1, pp. 185–208, 2003.

[14] C.-h. Chen, *Fuzzy logic and neural network handbook*. McGraw-Hill, Inc., 1996.

[15] R. Palm, D. Driankov, and H. Hellendoorn, *Model based fuzzy control: fuzzy gain schedulers and sliding mode fuzzy controllers*. Springer Science & Business Media, 2013.

[16] P. Vas, *Artificial-intelligence-based electrical machines and drives: application of fuzzy, neural, fuzzy-neural, and genetic-algorithm-based techniques*. Oxford university press, 1999, vol. 45.

[17] K. M. Passino, S. Yurkovich, and M. Reinfrank, *Fuzzy control*. Citeseer, 1998, vol. 20.

[18] J. Yen, R. Langari, and L. A. Zadeh, *Industrial applications of fuzzy logic and intelligent systems*. IEEE press, 1995.

[19] E. Monmasson and M. N. Cirstea, "Fpga design methodology for industrial control systems—a review," *IEEE transactions on industrial electronics*, vol. 54, no. 4, pp. 1824–1842, 2007.

[20] L. A. Zadeh, "Probability measures of fuzzy events," *Journal of mathematical analysis and applications*, vol. 23, no. 2, pp. 421–427, 1968.

[21] M. Salami and G. Cain, "An adaptive pid controller based on genetic algorithm processor," 1995.

[22] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE transactions on control systems technology*, vol. 13, no. 4, pp. 559–576, 2005.

[23] W. Tang, G. Chen, and R. Lu, "A modified fuzzy pi controller for a flexible-joint robot arm with uncertainties," *Fuzzy sets and systems*, vol. 118, no. 1, pp. 109–119, 2001.

Appendix A

```
// include the library code:
#include <LiquidCrystal.h>
#include <FuzzyRule.h>
#include <FuzzyComposition.h>
#include <Fuzzy.h>
#include <FuzzyRuleConsequent.h>
#include <FuzzyOutput.h>
#include <FuzzyInput.h>
#include <FuzzyIO.h>
// Step 1 - Instantiating an object library
Fuzzy* fuzzy = new Fuzzy();
// initialize the library with the numbers of the interface ...
    pins
LiquidCrystal lcd(12, 11, 10, 5, 8, 7);
// initiate input pins;
int ypin = 4, npin = 3, enpin = 9, in1pin = 6, in2pin = 2, ...
    flippin = 13;
// initiate program flow variables
int val;
long last = 0;
int stat = LOW;
int stat2;
int contar = 0;
int setSpeed = 127;
int sens = 60; // this value indicates the limit reading ...
    between dark and light,
// it has to be tested as it may change acording on the
// distance the leds are placed.
int nPalas = 7; // the number of blades of the propeller
boolean fDirection = true;
int millis_period = 500; // the time it takes each reading
// set fuzzy pid controls
double Setpoint, Input, Output;
//Define the aggressive and conservative Tuning Parameters
double aggKp=4, aggKi=0.2, aggKd=1;
```

```
33 double consKp=1, consKi=0.05, consKd=0.25;
34 void setup()
35 {
36     Serial.begin(9600);
37
38     // set up the LCD's number of columns and rows:
39     lcd.begin(16, 2);
40     // Print a message to the LCD.
41     lcd.print("Welcome ...");
42     pinMode(ypin, INPUT);
43     pinMode(npin, INPUT);
44     pinMode(flippin, INPUT);
45     pinMode(in1pin, OUTPUT);
46     pinMode(in2pin, OUTPUT);
47     // initilizing fuzzy papmeters
48     FuzzyInput* speed = new FuzzyInput(1);
49     FuzzySet* low = new FuzzySet(0, 0, 0, 1280);
50     speed->addFuzzySet(low);
51     FuzzySet* normal = new FuzzySet(320, 1600, 1600, 2880);
52     speed->addFuzzySet(normal);
53     FuzzySet* high = new FuzzySet(1920, 3200, 3200, 3200);
54     speed->addFuzzySet(high);
55     FuzzyOutput* pwm = new FuzzyOutput(1);
56     FuzzySet* L = new FuzzySet(0, 0, 0, 102);
57     pwm->addFuzzySet(L);
58     FuzzySet* N = new FuzzySet(25, 127, 127, 230);
59     pwm->addFuzzySet(N);
60     FuzzySet* H = new FuzzySet(153, 255, 255, 255);
61     pwm->addFuzzySet(H);
62     fuzzy->addFuzzyOutput(pwm);
63     // initiating fuzzy rules
64     FuzzyRuleAntecedent* ifSpeedLow = new ...
           FuzzyRuleAntecedent();
65     ifSpeedLow->joinSingle(low);
66     FuzzyRuleConsequent* thenPwmL = new FuzzyRuleConsequent();
67     thenPwmL->addOutput(L);
68     FuzzyRule* fuzzyRule01 = new FuzzyRule(1, ifSpeedLow, ...
           thenPwmL);
69     fuzzy->addFuzzyRule(fuzzyRule01);
70     FuzzyRuleAntecedent* ifSpeedNormal = new ...
           FuzzyRuleAntecedent();
71     ifSpeedNormal->joinSingle(normal);
72     FuzzyRuleConsequent* thenPwmN = new FuzzyRuleConsequent();
```

```
73      thenPwmN->addOutput(N);
74      FuzzyRule* fuzzyRule02 = new FuzzyRule(2, ...
            ifSpeedNormal, thenPwmN);
75      fuzzy->addFuzzyRule(fuzzyRule02);
76      FuzzyRuleAntecedent* ifSpeedHigh = new ...
            FuzzyRuleAntecedent();
77      ifSpeedHigh->joinSingle(high);
78      FuzzyRuleConsequent* thenPwmH = new FuzzyRuleConsequent();
79      51
80      thenPwmH->addOutput(H);
81      FuzzyRule* fuzzyRule03 = new FuzzyRule(3, ifSpeedHigh, ...
            thenPwmH);
82      fuzzy->addFuzzyRule(fuzzyRule03);
83      // initiate fuzzty PID controller
84      Setpoint = setSpeed*3200.0/255.0;
85      fuzzyPIDController.SetMode(AUTOMATIC);
86      delay(2000);
87  }
88  void loop()
89  {
90      /// ---------------------- fetch 500ms revolutions ...
            ------------------------------------------
91      val = map(1023 - analogRead(0), 265, 272, 0, 100); // ...
            read world version
92      // val = map(analogRead(0), 205, 760, 100, 0); ...
            //simulation version
93      Serial.println(val);
94      if (val < sens)
95      stat = LOW;
96      else
97      stat = HIGH;
98      digitalWrite(13, stat); //as iR light is invisible for ...
            us, the led on pin 13
99      //indicate the state of the circuit.
100     if (stat2 != stat) { //counts when the state change, ...
            thats from (dark to light) or
101         //from (light to dark), remmember that IR light is ...
                invisible for us.
102         contar++;
103         stat2 = stat;
104     }
105     // --------------------- calculate RPM ...
            ------------------------------------------------------
```

```
106     $
107     if (millis() - last ≥ millis_period) {
108         double rps = ((double)contar / nPalas) / 2.0 * ...
                1000.0 / millis_period;
109         double rpm = ((double)contar / nPalas) / 2.0 * ...
                60000.0 / (millis_period);
110         lcd.clear();
111         lcd.setCursor(0, 0);
112         lcd.print("Set %Speed: ");
113         lcd.print(map(setSpeed, 0, 255, 0, 100));
114         lcd.setCursor(0, 1);
115         lcd.print("Speed: ");
116         lcd.print(round(rpm));
117         // button press chance ...
                window----------------------------
118         int window = 1000;
119         int now = millis();
120         boolean done = false;
121
122         While ((millis() - now < 1000) && !done) {
123             if (digitalRead(ypin) == HIGH ) {
124                 while (digitalRead(ypin) == HIGH);
125                 setSpeed += 5;
126                 done = true;
127             }
128             52
129             else if (digitalRead(npin) == HIGH ) {
130                 while (digitalRead(npin) == HIGH);
131                 setSpeed -= 5;
132                 done = true;
133             }
134             else if (digitalRead(flippin) == HIGH ) {
135                 while (digitalRead(flippin) == HIGH);
136                 fDirection = !fDirection;
137                 done = true;
138             }
139             if (setSpeed > 255) setSpeed = 255;
140             if (setSpeed < 1) setSpeed = 0;
141         }
142         //feed fuzzy logic controller ...
                ------------------------------
143         Setpoint = setSpeed*3200.0/255.0;
144         Input = rpm;
```

```
145             double gap = abs(Setpoint-Input);
146             if(gap < 100) fuzzyPIDController.SetTunings(consKp, ...
                    consKi, consKd);
147             else fuzzyPIDController.SetTunings(aggKp, aggKi, ...
                    aggKd);
148             fuzzyPIDController.Compute();
149             fuzzy->setInput(1, Output);
150             fuzzy->fuzzify();
151             // control motor direction ...
                    ------------------------------------
152             if (fDirection) {
153                 analogWrite(enpin, fuzzy->defuzzify(1));
154                 digitalWrite(in1pin, HIGH), ...
                        digitalWrite(in2pin, LOW);
155             }
156             else {
157                 analogWrite(enpin, fuzzy->defuzzify(1));
158                 53
159                 digitalWrite(in1pin, LOW), digitalWrite(in2pin, ...
                        HIGH);
160             }
161             // reset scope control parameters
162             contar = 0;
163             last = millis();
164         }
165 }
```