FUZZY LOGIC CONTROLLER FOR AN AUTONOMOUS MOBILE ROBOT

VAMSI MOHAN PERI

Bachelor of Technology in Electrical and Electronics Engineering

Jawaharlal Nehru Technological University, India

May, 2002

Submitted in partial fulfillment of requirements for the degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

MAY, 2005

This thesis has been approved

by the Department of **ELECTRICAL AND COMPUTER ENGINEERING**

and the College of Graduate Studies by

_____

Dr. Dan Simon, Thesis Committee Chairperson

_____

Department/Date

_____

Dr. Ana V. Stankovic, Thesis Committee Member

_____

Department/Date

_____

Dr. Majid Rashidi, Thesis Committee Member

_____

Department/Date

To amma, dad, teja and ammamma…

ABSTRACT

In this thesis the development of an autonomous wall-following robot is presented. The wall following controller is a two input, two output system. The inputs are two proximity measurements to the wall, and the outputs are the speeds of the two rear wheels. For the embedded fuzzy logic controller, the behavior must be approximately encoded for the target processor, and then downloaded to the chip for execution. The target system is a small mobile robot equipped with an embedded microcontroller based on a Microchip PIC16F877 microcontroller. The robot is driven by two independent servo motors. Three ultrasonic range sensors are used by the robot – two on one side (the controller inputs) and one in the front (for emergency stop in case of an obstacle). Since all the control circuitry and computation are embedded in the robot, it is self contained and travels without the need for any data link to external processors such as a PC. For downloading the fuzzy control software and communicating with the microcontroller during testing, the robot must be connected to the PC via a standard RS232 serial link. The detection of a wall by the sensors activates the controller which simply attempts to align the robot with the wall at a specified reference distance. Once aligned, the robot follows the wall and attempts to maintain alignment by compensating for lateral drift. The fuzzy rule base and the generation of the processor-specific code was done using a compiler called PIC-C which provides the capability of converting C code into PIC16F877 assembly code.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE ..... PAGE

LIST OF TABLES

CHAPTER I

INTRODUCTION

Mobile robots are mechanical devices capable of moving in an environment with a certain degree of autonomy. Autonomous navigation is associated to the availability of external sensors that capture information of the environment through visual images or distance or proximity measurements. The most common sensors are distance sensors (ultrasonic, laser, etc) capable of detecting obstacles and of measuring the distance to walls close to the robot path. When advanced autonomous robots navigate within indoor environments (industrial or civil buildings), they have to be endowed the ability to move through corridors, to follow walls, to turn corners and to enter open areas of the rooms.

In attempts to formulate approaches that can handle real world uncertainty, researches are frequently faced with the necessity of considering tradeoffs between developing complex cognitive systems that are difficult to control, or adopting a host of assumptions that lead to simplified models which are not sufficiently representative of the system or the real world. The latter option is a popular one which often enables the formulation of viable

control laws. However, these control laws are typically valid only for systems that comply with imposed assumptions, and further more, only in neighborhoods of some nominal state. The option that involves complex systems has been less prevalent due to that lack of analytical methods that can adequately handle uncertainty and concisely represent knowledge in practical control systems. Recent research and application employing non-analytical methods of computing such as fuzzy logic, evolutionary computation, and neural networks have demonstrated the utility and potential of these paradigms for intelligent control of complex systems. In particular, fuzzy logic has proven to be a convenient tool for handling real world uncertainty and knowledge representation.

1.1 Literature Survey

As regards the corridor and wall-following navigation problem, some control algorithms based on artificial vision have been proposed. In [1], image processing is used to detect perspective lines to guide the robot along the centre axis of the corridor. In [2], two lateral cameras mounted on the robot are used, and the optical flow is computed to compare the apparent image velocity on both cameras in order to control robot motion. In [3, 4], one camera is used to drive the robot along the corridor axis or to follow a wall, by using optic flow computation and its temporal derivatives. In [5], a globally stable control algorithm for wall-following based on incremental encoders and one sonar sensor is developed. In [6], a theoretical model of a fuzzy based reactive controller for a non holonomic mobile robot is developed. In [7], an ultrasonic sensor is used to steer an autonomous robot along a concrete path using its edged as a continuous landmark. In [8],

a mobile robot control law for corridor navigation and wall following based on sensor and odometric sensorial information is proposed.

1.2 Applications

Wheeled mobile robots are becoming increasingly important in industry as a means of transport, inspection, and operation because of their efficiency and flexibility. In addition, mobile robots are useful for intervention in hostile environments. The motion of a wheeled mobile robot will, in general, be subject to nonholonomic constraints due to the rolling constraints of the wheels, which render a motion perpendicular to the wheels impossible. These nonholonomic constraints give rise to highly nonlinear mathematical models of the mobile robots, and the control problem is not trivial although the full state is measured. Feedback control of nonholonomic mobile robots is, therefore, a challenging problem which combines nonlinear control theory and differential geometry.

Fuzzy logic unlike classical logic is tolerant to imprecision, uncertainty, and partial truth. This makes it easier to implement fuzzy logic controller to nonlinear models than other conventional control techniques. In the context of mobile robot control, a fuzzy logic based system has the advantage that it allows intuitive nature of sensor-based navigation to e easily modeled using linguistic terminology. The computational loads of typical fuzzy inference systems are relatively light. As a result, reactive fuzzy control systems permit intelligent decisions to be made in the real time, thus allowing smooth and uninterrupted motion.

1.3 Thesis Organization

The main objective of this thesis is to develop a control system for an autonomous mobile robot for sensor base navigation. The main aim being, to build an intelligent autonomous robot able to dynamically interact with the real world. That is, it must be capable of sensing the real world, reasoning about the real world, and physically influencing the real world. In Chapter 2 we discuss the model of the robot, inspiration for choosing this particular problem, goals set to be achieved in this thesis the various techniques used and also the reasons for choosing fuzzy logic. Chapter 3 describes the basic concept of fuzzy logic, the use of fuzzy logic controller for path tracking of the robot, the hardware requirements and the constraints on the model developed. In Chapter 4 the Simulink model of robot is discussed and the results obtained from software simulations and real time runs are compared. Also the results obtains using a fuzzy logic controller are compared with that obtained using a proportional controller which is optimized using genetic algorithms. Chapter 5 includes conclusions and possible future work on autonomous robot path tracking.

CHAPTER II

MODEL AND CONTROL DESIGN OF ROBOT

In this chapter the robotic behavior and the model designed for that particular behavior are discussed. In Section 2.1 the various constraints imposed on the model are given. In Section 2.2 various types of robotic behavior that are generally used are discussed. Section 2.3 focuses on the mathematical modeling of the robot that was built for this thesis.

2.1 Introduction

The robot that is to be controlled was initially built to take part in an IEEE competition, held at Cleveland State University in 2004. The goal of this competition is to compete head to head on the playing board with an opponent and obtain the most points in an

allotted amount of time. The dimensions of the robot had to be within those specified in the competition. The robot had to fit within a 1.5-foot square and could not exceed 1.5 feet in height. It should be totally autonomous, shouldn't transmit or receive signals to or from the outside of the playing area, and shall not be equipped to intentionally harm its opponents. It also shouldn't carry any onboard cameras. The main goal of this work, keeping in mind the requirements of the competition, was to cover as much of the playing area as possible within the shortest time so that the maximum points can be scored. The robot that was built placed second among eleven other competitors from different schools.

2.2 Types of Robotic Behavior

A mobile robot could be modeled in numerous ways, but the most important factor for defining the model would be the application and the complexity involved. The mobile robot designed in this work is a wheeled robot intended for indoor use as opposed to other types (legged, airborne, and submersible mobile robots). This robot type is the easiest to model, control, and build. There are various behaviors that could be modeled, like wall following, collision avoidance, corridor following, goal seeking, adaptive goal seeking, etc., as shown in Figure 2.1. With the competition in mind we had thought of implementing a wall following robot. This robot would follow the boundaries of the playing area and cover a maximum area in a predefined path programmed into its onboard microcontroller.

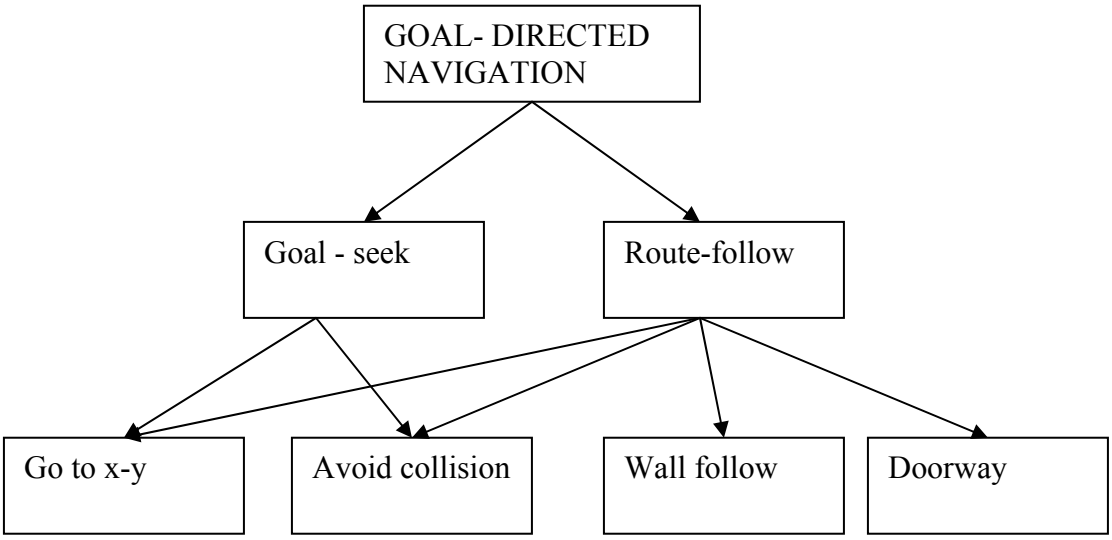


Figure 2.1 Hierarchical decomposition of mobile robot behavior [9]

Various control techniques have been proposed and are being researched. The control strategies of mobile robots can be divided into open loop and closed loop feedback strategies. In open loop control, the inputs to the mobile robots (velocities or torques) are calculated beforehand, from the knowledge of the initial and end position and of the desired path between them in the case of path following. This strategy cannot compensate for disturbances and model errors. Closed loop strategies however may give the required compensation since the inputs are functions of the actual state of the system and not only of the initial and the end point. Therefore disturbances and errors causing deviations from the predicted state are compensated by the use of the inputs. Of the many available closed loop control systems like P (proportional) control, PI (proportional integral) control, and PID (proportional integral derivative) control, fuzzy logic control was selected as it was easiest to implement for a highly nonlinear model.

Although a relatively new concept, fuzzy logic is being used in many engineering applications because it is considered by the designers to be the simplest solution available for the specific problem. What gives fuzzy logic advantages over more traditional solutions is that it allows computers to reason more like humans, responding effectively to complex inputs to deal with notions such as 'too hot', 'too cold' or 'just right'. Furthermore, fuzzy logic is well suited to low-cost implementations based on cheap sensors, low-resolution analog-to-digital converters, and 4-bit or 8-bit one-chip microcontroller chips. Such systems can be easily upgraded by adding new rules to improve performance or add new features. In many cases, fuzzy control can be used to improve existing traditional controller systems by adding an extra layer of intelligence to the current control method. In many cases, the mathematical model of the control process may not exist, or may be too "expensive" in terms of computer processing power and memory, and a system based on empirical rules may be more effective. Implementing the enhanced, traditional PID controller can be a challenge, especially if auto-tuning capabilities to help find the optimal PID constants are desired. However, the theory of PID control is very well known and widely used in many other control applications. On the other hand, fuzzy control seems to accomplish better control quality with less complexity (if tuning / gain scheduling needed for the PID approach). Approximation of the second-order switching curve used in time-optimal control systems by a polynomial of the first or higher order makes fuzzy control a better candidate for time-optimal control applications [10]. As a relatively new control method it provides more space for further improvements.

Originally advocated by Zadeh [11] and Mamdaniand Assilian [12], fuzzy logic has become a means of collecting human knowledge and experience and dealing with uncertainties in the control process. Now fuzzy logic is becoming a very popular topic in control engineering. Considerable research and applications of this new area for control systems have taken place. Control is by far the most useful application of fuzzy logic theory, but its successful applications to a variety of consumer products and industrial systems have helped to attract growing attention and interest.

Thereafter, a situation for which the vehicle tries to reach an end point is examined. Based on its design simplicity, its ease of implementation, and its robustness properties, a fuzzy logic controller is used in this thesis to control the navigation behavior of an autonomous mobile robot.

2.3 Mathematical Robot Model

This section describes the kinematics and dynamic equations of the wheeled mobile robot of the unicycle type and formulates the problem of controlling it to a point with a desired orientation. The vehicle has two identical parallel, non-deformable rear wheels which are controlled by two independent motors, and a steering front wheel. It is assumed that the plane of each wheel is perpendicular to the ground and the contact between the wheels and the ground is pure rolling and non-slipping, i.e., the velocity of the center of mass of the robot is orthogonal to the rear wheels' axis. It is further assumed that the masses and inertias of the wheels are negligible and that the center of mass of the mobile robot is located in the middle of the axis connecting the rear wheels.

2.3.1 Robotic Wheel

An idealized rolling wheel is shown in Figure 2.2**.** The wheel is free to rotate about its axis (y_w axis). The robot exhibits preferential rolling motion in one direction (x_w axis) and a certain amount of lateral slip. For lower velocities, rolling motion is dominant and slipping can be neglected.

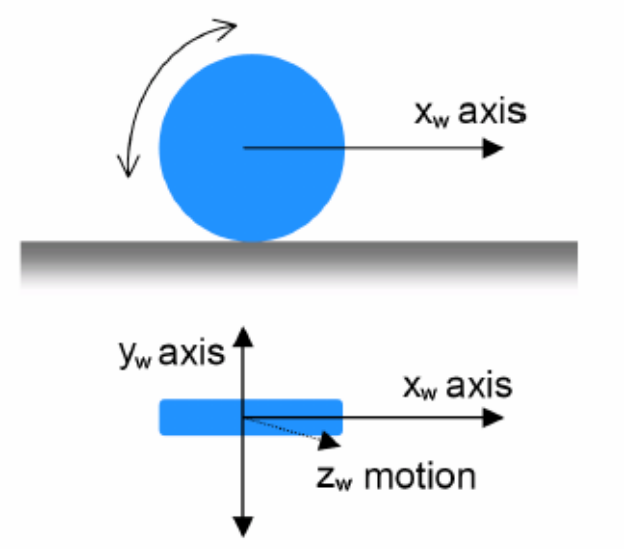


Figure 2.2 Idealized rolling wheel [13]

The wheel parameters are

r= wheel radius

v= wheel linear velocity

w= wheel angular velocity

2.3.2 Differential Drive

Kinematics is the study of the mathematics of motion without considering the forces that affect the motion. It deals with the geometric relationships that govern the system. It

develops a relationship between control parameters and the parameters and the behavior of a system in space. The model of the robot is as shown in Figure 2.3

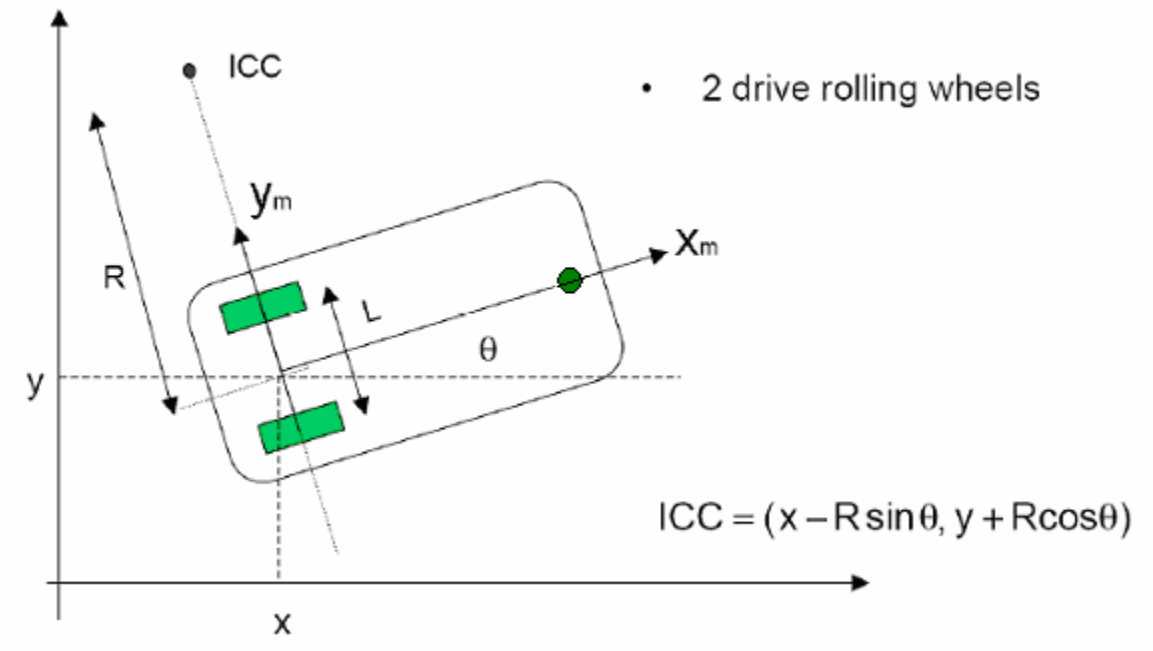


Figure 2.3 Kinematic model of the robot

2.3.3 Equations Defining the Robot

For a differential drive the kinematics equations in the world frame are as follows [13].

$v_r(t)$ = linear velocity of right wheel

$v_l(t)$ = linear velocity of left wheel

$w_r(t)$ = angular velocity of right wheel

$w_l(t)$ = angular velocity of left wheel

r = nominal radius of each wheel

L = distance between the two wheels

R = instantaneous curvature radius of the robot trajectory, relative to the mid-point axis

ICC = Instantaneous Center of Curvature

$R-\frac{L}{2}$ = Curvature radius of trajectory described by left wheel

$R+\frac{L}{2}$ = Curvature radius of trajectory described by right wheel

With respect to ICC the angular velocity of the robot is given as follows

$$w(t)=\frac{v_r(t)}{R+L/2} \tag{2.1}$$

$$w(t)=\frac{v_l(t)}{R-L/2} \tag{2.2}$$

$$w(t)=\frac{v_r(t)-v_l(t)}{L} \tag{2.3}$$

The instantaneous curvature radius of the robot trajectory relative to the mid-point axis is given as

$$R=\frac{L(v_l(t)+v_r(t))}{2(v_l(t)-v_r(t))} \tag{2.4}$$

Therefore the linear velocity of the robot is given as

$$v(t)=w(t)R=\frac{1}{2}(v_r(t)+v_l(t)) \tag{2.5}$$

The kinematics equations in the world frame can be represented as follows.

$$\begin{aligned}\dot{x}(t)&=v(t)\cos\theta(t)\\ \dot{y}(t)&=v(t)\sin\theta(t)\\ \dot{\theta}(t)&=w(t)\end{aligned} \tag{2.6}$$

This implies

$$x(t) = \int_0^t v(\sigma)\cos(\theta(\sigma))d\sigma$$
$$y(t) = \int_0^t v(\sigma)\sin(\theta(\sigma))d\sigma \qquad (2.7)$$
$$\theta(t) = \int_0^t w(\sigma)d\sigma$$

The above equations can also be represented in the following form.

$$\begin{bmatrix} v_x(t) \\ v_y(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ w\ (t) \end{bmatrix} = \begin{bmatrix} v(t)\cos\theta \\ v(t)\sin\theta \\ w(t) \end{bmatrix} \begin{bmatrix} v_l(t) \\ v_r(t) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2}(v_r + v_l)\cos\theta \\ \frac{1}{2}(v_r + v_l)\sin\theta \\ (v_r - v_l)/L \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ -1/L & 1/L \end{bmatrix} \begin{bmatrix} v_r \\ v_l \end{bmatrix} \qquad (2.8)$$

These are the equations that are used to build a model of the robot. These equations were used to simulate the robot in MATLAB Simulink. The fuzzy logic controller was tested and fine tuned on this model, as well as compared with other controllers for optimum results.

CHAPTER III

FUZZY LOGIC

The human brain interprets imprecise and incomplete sensory information provided by perceptive organs. Fuzzy set theory provides a systematic calculus to deal with such information linguistically, and it performs numerical computation by using linguistic labels stipulated by membership functions. A fuzzy inference system (FIS) when selected properly can effectively model human expertise in a specific application. In this chapter the basic terminology of fuzzy logic is discussed, giving a detailed explanation of all the aspects involved. Later the design of the fuzzy logic controller used in this thesis is given.

A classic set is a crisp set with a crisp boundary. For example, a classical set A of real numbers greater than 6 can be expressed as

$$A = \{x \mid x > 6\} \tag{3.1}$$

where there is a clear, unambiguous boundary 6 such that if x is greater than this number, then x belongs to this set A; or otherwise does not belong to this set. Although classical sets are suitable for various applications they do not reflect the mature of human concepts and thoughts, which tend to be abstract and imprecise.

In contrast to a classical set, a fuzzy set, as the name implies, is a set without a crisp boundary. That is, the transition from "belongs to a set" to "does not belong to a set" is gradual, and this smooth transition is characterized by membership functions that give fuzzy sets flexibility in modeling commonly used linguistic expressions, such as "the water is hot" or "the temperature is high". The fuzziness does not come from the randomness of the constituent members of the set, but from the uncertainties and imprecise nature of abstract thoughts and concepts. In Sections 3.1 to 3.5 the various notations and terminologies used in fuzzy logic are described. In Section 3.6 the fuzzy logic controller used is described. Finally in Section 3.7 the tuning of the fuzzy logic controller is discussed.

3.1 Basic Definitions and Terminology

Let X be a space of objects and x be a generic element of X. A classic set A , $A \subseteq X$ is defined as a collection of elements or objects $x \in X$, such that each x can either belong to or not belong to the set A. By defining a characteristic function for each element x in X, we can represent a classical set A by a set of ordered pairs (x, 0) or (x, 1), which indicates $x \notin A$ or $x \in A$, respectively.

Unlike the classical set, a fuzzy set expresses the degree to which an element belongs to a set. Hence the characteristic function of a fuzzy set is allowed to have values between 0 and 1, which denotes the degree of membership of an element in a given set.

If X is a collection of objects denoted generically by x, then a fuzzy set A in x is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\}, \tag{3.2}$$

where $\mu_A(x)$ is called the membership function (MF) for the fuzzy set A. The MF maps each element of x to a membership value between 0 and 1. It is obvious that if the value of the membership function $\mu_A(x)$ is restricted to either 0 or 1, then A is reduced to a classical set and $\mu_A(x)$ is the characteristic function of A. Usually X is referred to as the universe of discourse, or simply the universe, and it may consist of discrete (ordered or unordered) objects or continuous space.

The construction of a fuzzy set depends on two things: the identification of a suitable universe of discourse and the specification of an appropriate membership function. Therefore, the subjectivity and non-randomness of fuzzy sets is the primary difference between the study of fuzzy sets and probability theory.

In practice, when the universe of discourse X is a continuous space, we usually partition X into several fuzzy sets whose MFs cover x in a more or less uniform manner. These fuzzy sets, which usually carry names that confirm to adjectives appearing in our daily

linguistic usage, such as “large,” “medium,” or “negative” are called linguistic values or linguistic labels.

3.2 Membership Functions (MF)

As discussed above a fuzzy set is completely parameterized by its MF. Since most fuzzy sets have a universe of discourse *X* consisting of the real line R, it would be impractical to list all the pairs defining a membership function. So a MF is expressed with the help of a mathematical formula. A MF can be parameterized according to the complexity required. These also could be one dimensional or multi dimensional. Here are a few classes of parameterized MFs of one dimension that is MFs with a single input.

3.2.1 Triangular Membership Function

A triangular MF is specified by three parameters *{a, b, c}* as follows

$$triangle(x;a,b,c)=\begin{cases}0, & x<a.\\ \dfrac{x-a}{b-a}, & a\le x\le b.\\ \dfrac{c-x}{c-b}, & b\le x\le c.\\ 0, & c\le x.\end{cases} \tag{3.3}$$

By using min and max, we have an alternative expression for the preceding equation

$$triangle(x;a,b,c)=\max\left(\min\left(\frac{x-a}{b-a},\frac{c-x}{c-b}\right),0\right) \tag{3.4}$$

The parameters *{a, b, c}* (with $a<b<c$) determine the *x* coordinates of the three corners of the underlying triangular MF.

3.2.2 Trapezoidal Membership Function

A trapezoidal MF is specified by four parameters {a, b, c} as follows

$$trapezoid(x;a,b,c,d)=\begin{cases}0, & x<a.\\ \dfrac{x-a}{b-a}, & a\le x\le b.\\ 1, & b\le x\le c.\\ \dfrac{d-x}{d-c}, & c\le x\le d.\\ 0, & d\le x.\end{cases} \tag{3.5}$$

An alternative concise expression using min and max is

$$trapeziod(x;a,b,c)=\max\left(\min\left(\frac{x-a}{b-a},1,\frac{d-x}{d-c}\right),0\right). \tag{3.6}$$

The parameter *{a, b, c, d}* (with $a < b < c < d$) determine the *x* coordinates of the four corners of the underlying trapezoidal MF.

3.2.3 Gaussian Membership Function

A Gaussian MF is specified by two parameters $\{c,\sigma\}$

$$gaussian(x;c,\sigma)=e^{-\frac{1}{2}(\frac{x-c}{\sigma})^2} \tag{3.7}$$

A Gaussian MF is determined completely by *c* and σ; *c* represents the MFs center and σ determines the MFs width.

3.2.4 Generalized Bell Membership Function

A generalized bell MF (or bell MF) is specified by three parameters *{a, b, c}*

$$bell\{x;a,b,c\}=\frac{1}{1+\left|\dfrac{x-c}{a}\right|^{2b}} \tag{3.8}$$

where the parameter *b* is usually positive. It is also called as the Cauchy MF.

3.2.5 Sigmoidal Membership Function

A sigmoidal MF is defined by

$$sig\{x;a,c) = \frac{1}{1+\exp[-a(x-c)]} \tag{3.9}$$

where *a* controls the slope at the crossover point *x=c*. Sigmoidal functions are widely used as the activation function of artificial neural networks.

3.3 Linguistic Variables and Fuzzy If-Then Rules

In 1973, Professor Lotfi Zadeh proposed the concept of linguistic or "fuzzy" variables. Think of them as linguistic objects or words, rather than numbers. The sensor input is a noun, e.g. "temperature," "displacement," "velocity," "flow," "pressure," etc. Since error is just the difference, it can be thought of the same way. The fuzzy variables themselves are adjectives that modify the variable (e.g. "large positive" error, "small positive" error, "zero" error, "small negative" error, and "large negative" error). As a minimum, one could simply have "positive", "zero", and "negative" variables for each of the parameters. Additional ranges such as "very large" and "very small" could also be added to extend the responsiveness to exceptional or very nonlinear conditions, but aren't necessary in a basic system.

Once the linguistic variables and values are defined, the rules of the fuzzy inference system can be formulated. These rules map the fuzzy inputs to fuzzy outputs. This

mapping takes place through compositional rule of inference which is based on Zadeh's extension of *modus ponens* which is nothing more than the familiar if-then conditional form. A fuzzy if-then rule (also known as fuzzy rule) assumes the form

$$\text{If } x \text{ is } A \text{ then } y \text{ is } B, \tag{3.10}$$

where A and B are linguistic values defined by fuzzy sets on universe of discourse X and Y , respectively. "x is A " is called the antecedent or premise, while "y is B" is called the consequent or conclusion. This rule is also abbreviated as $A \rightarrow B$.

The antecedent normally consists of some combination of the inputs and the consequent consists of output variables. The antecedent takes the form

$$\tilde{u}_1 \text{ is } \tilde{A}_1^j \text{ and } \tilde{u}_2 \text{ is } \tilde{A}_1^k \text{ and}, ..., \tilde{u}_n \text{ is} A_n^1$$

where

$$\tilde{A}_i = \{\tilde{A}_i^j : j = 1, 2, ..., N_i\} \tag{3.11}$$

$\tilde{A}^j{}_i$ is the j^{th} linguistic value of the linguistic variable $\tilde{u}_i$ defined over the universe of discourse U_i. The linguistic variable $\tilde{u}_i$ and its linguistic value $\tilde{A}^j{}_i$ are combined with the other variables and values on the antecedent by the fuzzy AND or OR operators. The choice depends upon the desired inference system. The AND operator corresponds to the intersection of the fuzzy sets and the OR operator corresponds to the union of the fuzzy sets. The antecedent need not require all linguistic variables and indeed could contain as few as one.

The consequent takes the form

$\tilde{y}_i$ is $\tilde{B}_i^p$

where

$$\tilde{B}_i = \{\tilde{B}^p{}_i : p = 1, 2, ..., N_i\} \quad (3.12)$$

$\tilde{B}^p{}_i$ is the p^{th} linguistic value of he linguistic variable $\tilde{y}_i$ defined over the universe of discourse Y_i.

Assuming all premise terms are used in every rule and a rule is generated for each possible premise combination, there will be

$$\prod_{i=1}^{n} N_i = N_1.N_2....N_n \quad (3.13)$$

rules. n represents the number of linguistic variables in the antecedent and N_i is number of linguistic values per variable.

3.4 Fuzzy Inference Systems

There are three main fuzzy logic inference systems (fuzzy logic approximators): Mamdani type, Sugeno type, and Tsukamoto type. Of these Mamdani fuzzy inference system is used.

As shown in Figure 3.1, the Mamdani type of fuzzy logic controller contains four main parts, two of which perform transformations.

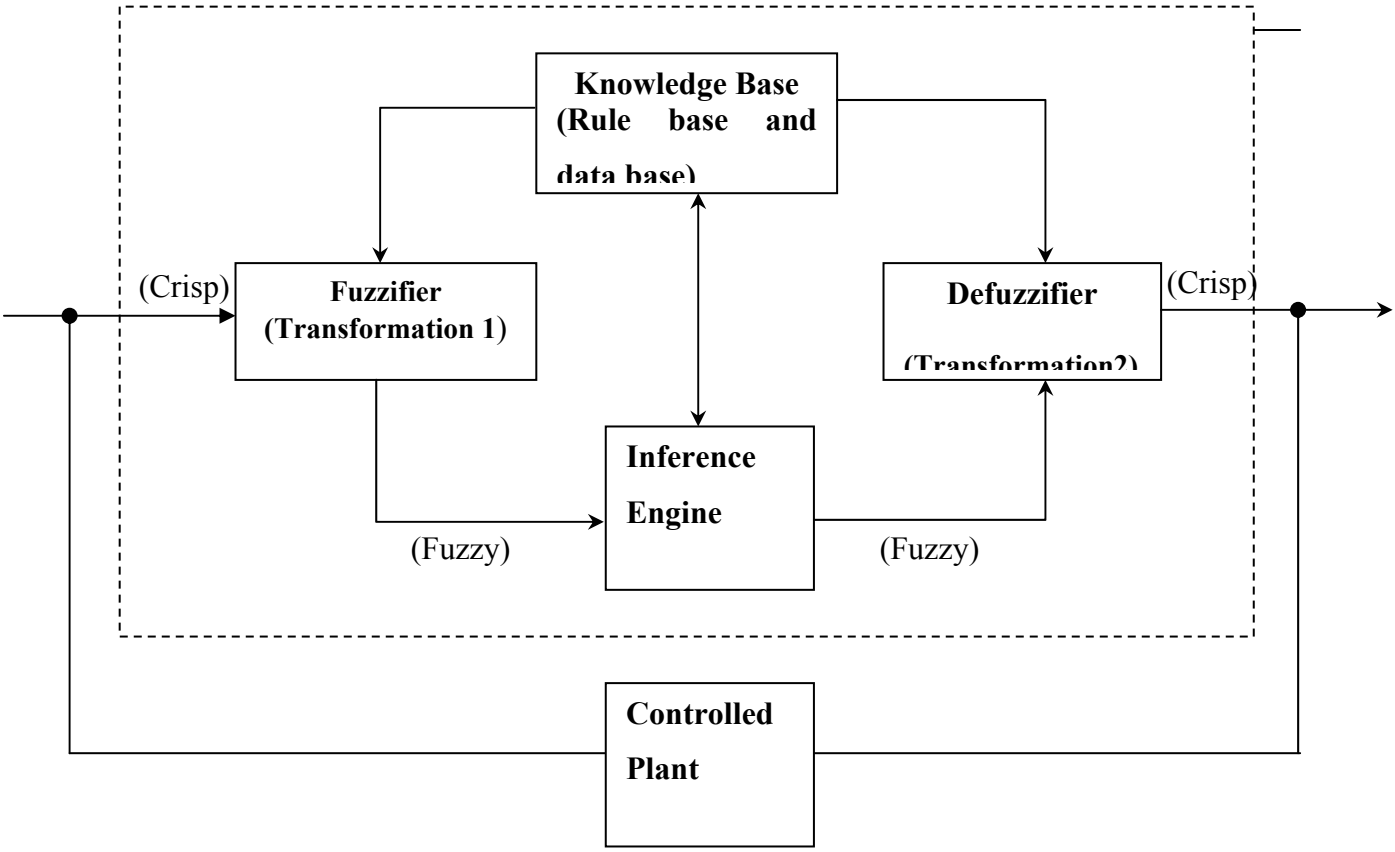


Figure 3.1 Fuzzy logic inference system.

The four parts are

- Fuzzifier (transformation 1);
- Knowledge base;
- Inference engine (fuzzy reasoning , decision-making logic);
- Defuzzifier (transformation 2).

The fuzzifier performs measurements of the input variables (input signals, real variables), scale mapping and fuzzification (transformation 1). Thus all the monitored signals are scaled, and fuzzification means that the measured signals (crisp input quantities which have numerical values) are transformed into fuzzy quantities (which are also referred to

as linguistic variables in the literature). This transformation is performed using membership functions. In a conventional fuzzy logic controller, the number of membership functions and the shapes of these are initially determined by the user. A membership function has a value between 0 and 1, and it indicates the degree of belongingness of a quantity to a fuzzy set. If it is absolutely certain that the quantity belongs to the fuzzy set, then its value is 1, but if it is absolutely certain that it does not belong to this set then its value is 0.

The membership functions can take many forms including triangular, Gaussian, bell-shaped, trapezoidal, etc. The knowledge base consists of the data base and the linguistic control rule base. The data base provides the information which is used to define the linguistic control rules and the fuzzy data manipulation in the fuzzy logic controller. The rule base defines (expert rules) specifies the control goal actions by means of a set of linguistic rules. In other words, the rule base contains rules such as would be provided by an expert. The FLC looks at the input signals and by using the expert rules determines the appropriate output signals (control actions). The rule base contains a set of if–then rules. The main methods of developing the rule base are:

- Using the experience and knowledge of an expert for the application and the control goals;
- Modeling the control action of the operator;
- Modeling the process;
- Using a self organized fuzzy controller;
- Using an artificial controller;

- Using artificial neural networks.

When the initial rules are obtained by using expert physical considerations, these can be formed by considering that the three main objectives to be achieved by the FLC are

- Removal of an significant errors in the process output by suitable adjustments of the control output;
- Ensuring a smooth control action near the reference value;
- Preventing the process output exceeding user specified values.

The inference engine (reasoning mechanism) is the kernel of FLC and has the capability both of simulating human decision making based on fuzzy concepts and inferring fuzzy control actions by using fuzzy implications and fuzzy logic rules of inference. In other words, once all the monitored input variables are transformed into their respective linguistic variables (by transformation 1), the inference engine evaluates the set of if-then rules (given in the rule base) and thus a result is obtained which is again a linguistic value for the linguistic variable. The linguistic result has to be then transformed into a crisp output value of the FLC and this is why there is a second transformation in the FLC. The second transformation is performed by the defuzzifier which performs scale mapping as well as defuzzification. The defuzzifier yields a non fuzzy, crisp control action from the inferred fuzzy control action by using the consequent membership functions of the rules.

3.5 Defuzzification

There are many defuzzification techniques some of which are discussed here.

3.5.1 Centroid of Area ${}^{z}COA$

$$ {}^{z}COA = \frac{\int_{z} \mu_A(z) z dz}{\int_{z} \mu_A(z) dz} \tag{3.14} $$

where $\mu_A(z)$ is the aggregated output MF and z is the output quantity. This is the most widely adopted defuzzification strategy. For discrete values, above equation can be put in the form

$$ {}^{z}COA = \frac{\sum_{k=1}^{n} \mu_A(z_k) z_k dz}{\sum_{k=1}^{n} \mu_A(z_k) dz} \tag{3.15} $$

where $\mu_A(z_k)$ are the k=1,2,…,n sampled values of the aggregated output membership function.

3.5.2 Mean-Max Method (Middle of Maxima MOM Method)

In this defuzzification technique, the average output value

$$ z = \frac{(z_1 + z_2)}{2} \tag{3.16} $$

is obtained, where z_1 is the first value and z_2 is the last value, where the output overall membership function, $\mu_A(z)$,is maximum.

3.5.3 First of Maxima (FOM) Method

When this defuzzification technique, the first value of the overall output membership function with maximum membership $\mu_A(z)$ degree is taken. It should be noted that this is equal to z_1 used in the MOM defuzzification method.

3.5.4 Last of Maxima (LOM) Method

When this defuzzification technique, the last value of the overall output membership function with maximum membership $\mu_A(z)$ degree is taken. It should be noted that this is equal to z_2 used in the MOM defuzzification method.

In general, the defuzzification operations are time consuming and they are not easily subject to rigorous mathematical analysis.

3.6 Fuzzy Logic Controller (FLC) Design

There has been some published worked regarding fuzzy logic motion control. However, this area is still relatively uncharted when compared to the amount of published research regarding other methods of control. Also, more work seems to have been done in applying fuzzy logic to process control applications where variables such as temperature or material level are being controlled.

Perhaps the lack of published fuzzy logic motion control research is because motion control can be very demanding due to computational power necessary. Inherent to the increased use of microcontrollers and digital forms of control are hardware limitations as to how quickly the feedback signal can be processed and updated into a control signal.

Fast motion indexing requires high sampling and update rates for stability. PID control requires, in general, less processor time to calculate control signals (one of PID's advantages) than a comparable FLC. The processing time required when using fuzzy logic control depends upon the number of rules that must be evaluated. Large systems with many rules would require very powerful and fast processors to compute in real time. The smaller the rule base, the less computational power needed. To reduce processing time, a static lookup table can be used to generate FLC control action. In some applications, that can greatly reduce processing time compared to performing fuzzy inference [9].

This thesis details the FLC designed to control the motion of a robot. The FLC used has two inputs: error in the position and error in the angle of the robot. Thus the FLC is a two input, two output system. MATLAB'S Fuzzy Logic Toolbox [14] was used to aid in the FLC design. The toolbox contains functions, graphical user interfaces and data structures that allow the user to quickly design, test, simulate and modify a fuzzy inference system.

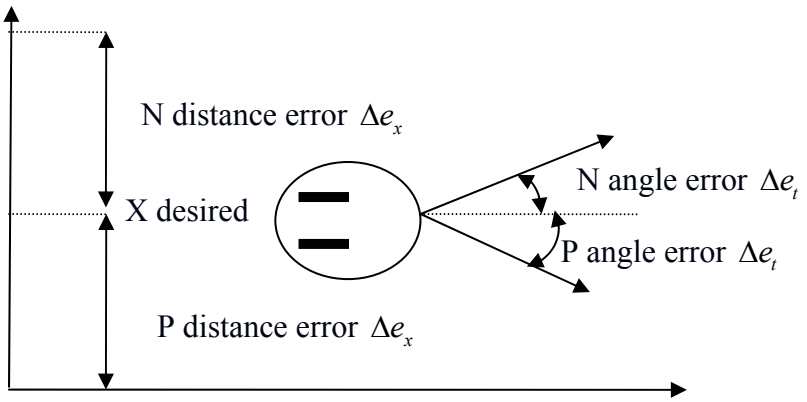


Figure 3.2 Robot in Cartesian space.

From Figure 3.2 it is clear that the two inputs are error in angle of orientation θ and error in distance x. The output of the controller (that is, the control signals) would be pulse-width-modulated signals to control the angular velocity of the two servo wheels. The block diagram of the whole system is given in Figure 3.3.

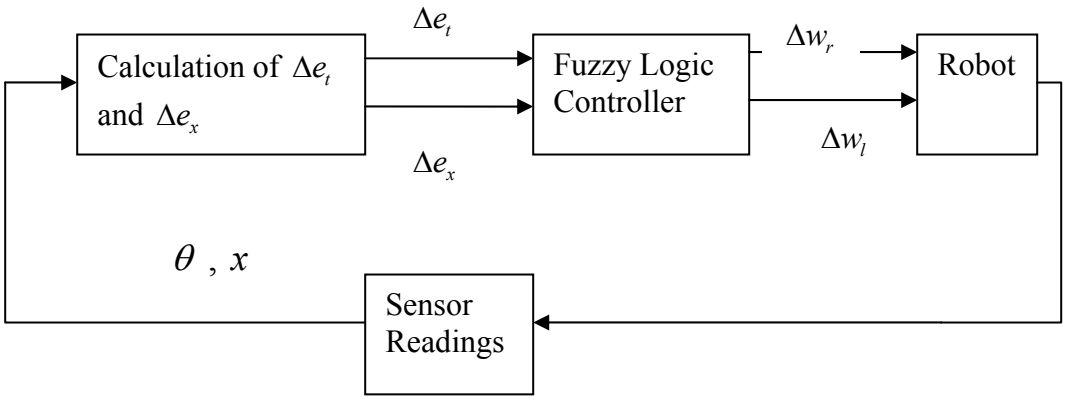


Figure 3.3 Block diagram of the whole system.

3.6.1 Universe of Discourse

The universe of discourse of the error in orientation angle Δe_t would normally span between plus and minus the maximum angle of orientation symmetric about zero. (θ = angle of orientation). To simplify the model the range of Δe_t was reduced to -0.3 to +0.3 radians.

The universe of discourse of the error in distance Δe_x spans from plus to minus of the reference distance to be maintained. Thus the range of Δe_x is -4 to 4. It is assumed that the reference distance wouldn't be set to more than 4 inches from the wall for simplicity.

The output of the FLC is the change in angular velocities of the two wheels Δw_r and Δw_l (which are the control signals). Their ranges would depend upon the mechanical constraints. Their ranges are -3 to 3 radians per second.

3.6.2 Linguistic Variables, Values and Membership Functions

The linguistic variables are error in angle Δe_t, error in distance Δe_x and change in angular velocities of the two wheels Δw_r and Δw_l. The error in distance and error in angle are rounded off as the sensor outputs a value rounded to the nearest inch.

All three linguistic variables have been assigned three linguistic values. The linguistic variables error in angle Δe_t and error in distance Δe_x have linguistic values N (negative), Z (zero), P (positive). The changes in angular velocities of the two wheels Δw_r and Δw_l have linguistic values S (slow), M (medium) and F (fast). Each linguistic value is assigned a triangular membership function. The membership functions are shown in Figures 3.4 to 3.7 below. The triangular membership functions are used for their simplicity as is quite commonly done. Calculating fuzzified inputs and the areas of these functions is simple and fast when compared to other membership functions such as Gaussian.

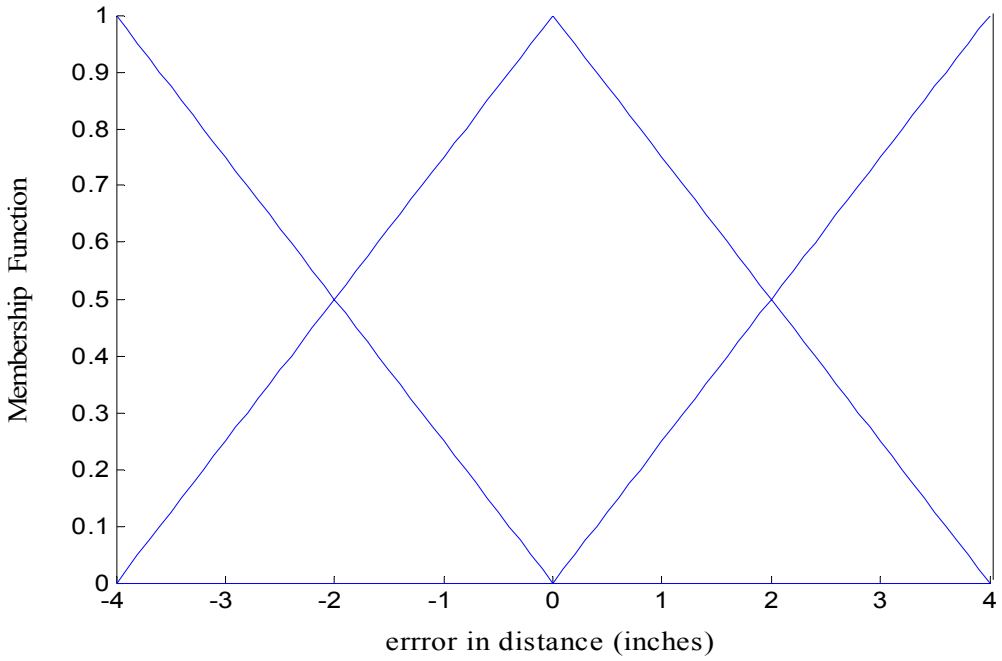


Figure 3.4 Sum normal membership function of error in distance Δe_x.

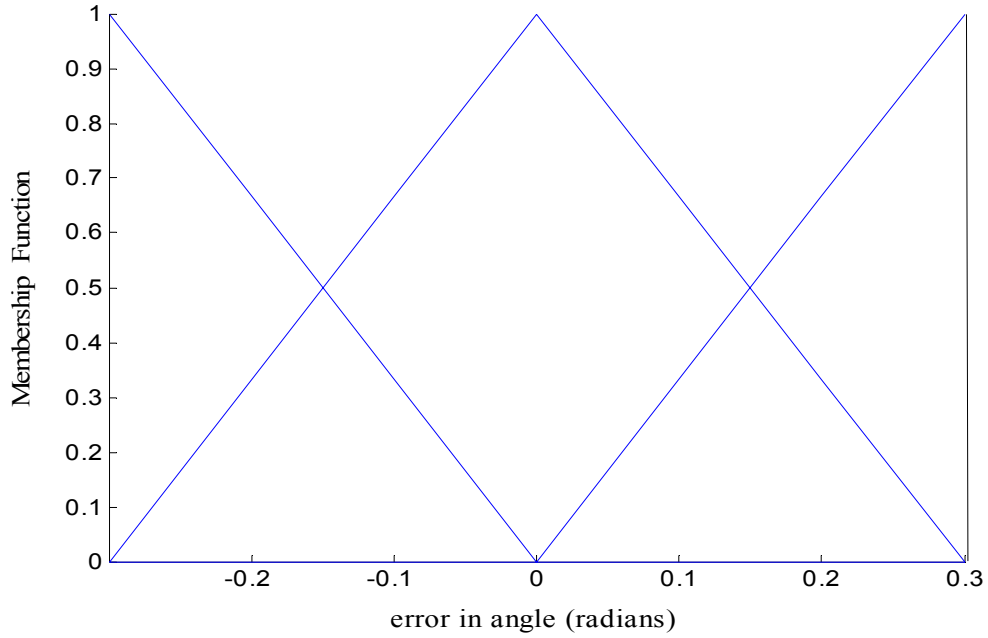


Figure 3.5 Sum normal membership function of error in theta Δe_t

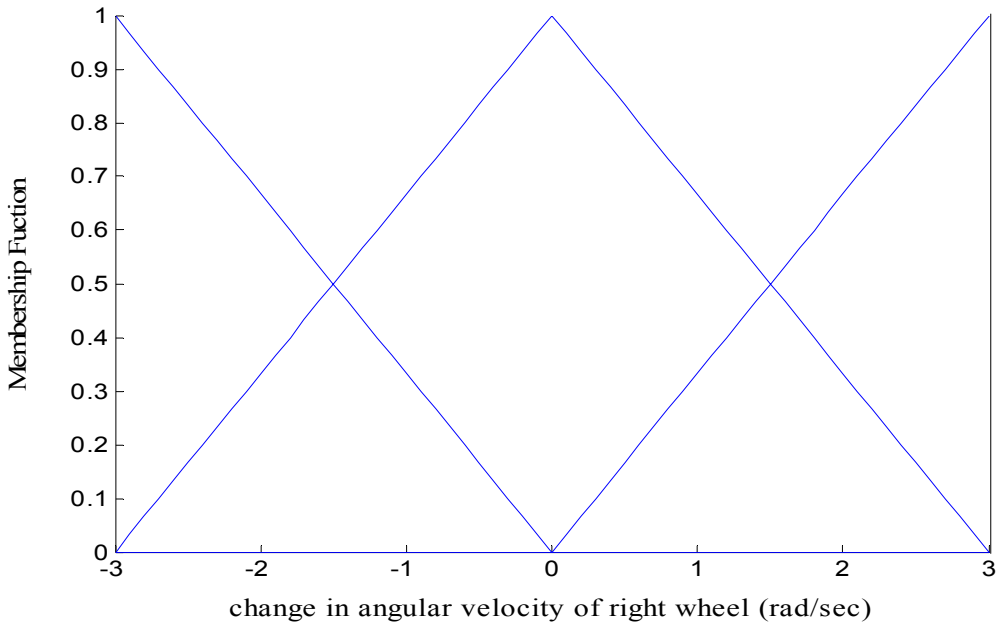


Figure 3.6 Change in velocities Δw_r

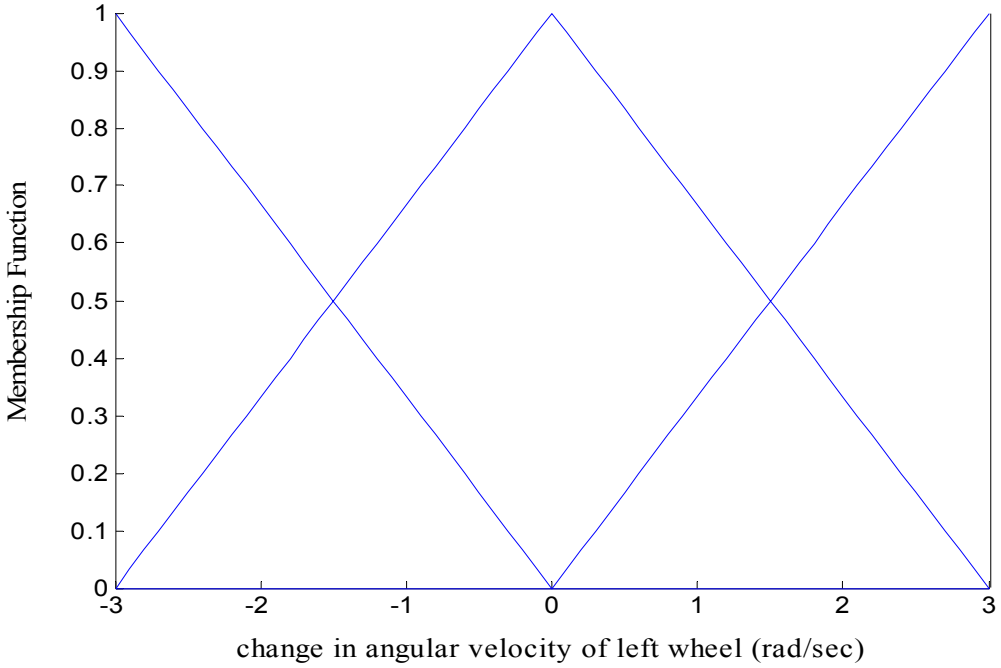


Figure 3.7 Change in velocities Δw_l

.

3.6.3 Rule Base

Generating the rules for fuzzy inference system is often the most difficult step in the design process. It usually requires some expert knowledge of the plant dynamics. This knowledge could be in the form of an intuitive understanding gained from experimenting, or it could come from a plant model which is then used in a computer simulation. The former method was used in this thesis. The rule base for the FLC proposed in this study is listed in Table 3.1 and 3.2 below. Thus we see that there are 18 total rules for the two wheels combined. Of these at any instant only two given rules are fired thus making it easier to understand and debug the rules. Figure 3.8 shows how for a given error in distance and angle the corresponding rules are fired and the defuzzified output is arrived at. In the situation described the error in distance (Δe_x) is -0.625 and the error in angle (Δe_t) is +0.25. Let us consider initially the defuzzified output of right wheel. We see that the error in distance is both negative and zero and error in angle is both positive and zero. The set of rules in column three represent the output for right wheel. Taking the case of rule one (row one in the Fig 3.8), we see that the membership function of Δe_x is 0.25 and membership function of Δe_t is zero. Thus the output of these two is the minimum of these two values which is zero, thus no rule in the third column representing the output for right wheel is fired. Now considering the third row in the figure which corresponds to the third rule, we see that membership function of Δe_x is 0.25 and membership function of Δe_t is 0.75. Thus the output of these two is the minimum of these two values that would be 0.25. This the value at which the output is clipped. Similarly as we notice the only other rule that is fired is rule six which clips the output at a value of 0.8. Now the two outputs are combined using the logical OR operator which results in a polygon. Now

using the centroid of area defuzzification the centroid of the polygon is calculated which is the crisp output for the right wheel. Similarly the output for left wheel is arrived at.

$\Delta e_x / \Delta e_t$	N(Negative)	Z(Zero)	P(Positive)
N(Negative)	S(Slow)	S(Slow)	S(Slow)
Z(Zero)	S(Slow)	F(Fast)	M(Medium)
P(Positive)	S(Slow)	M(Medium)	F(Fast)

Table 3.1 Rule base for change in angular velocity of right wheel Δw_r .

$\Delta e_x / \Delta e_t$	N(Negative)	Z(Zero)	P(Positive)
N(Negative)	F(Fast)	M(Medium)	S(Slow)
Z(Zero)	M(Medium)	F(Fast)	S(Slow)
P(Positive)	S(Slow)	S(Slow)	S(Slow)

Table 3.2 Rule base for change in angular velocity of left wheel Δw_l .

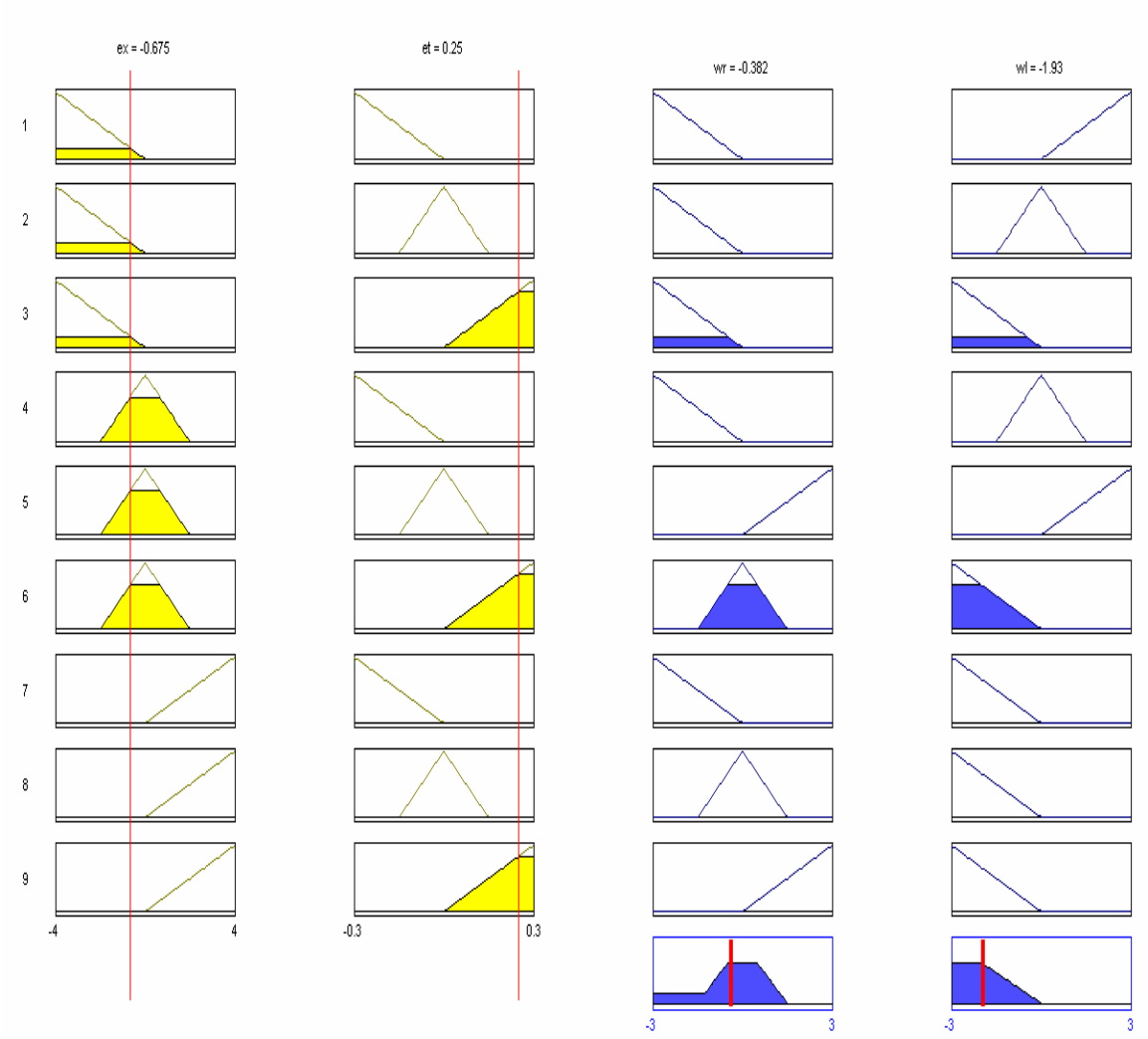


Figure 3.8 Firing of rule base.

Figure 3.9 shown below shows a scenario when Δe_t (error in angle) = N (that is, the robot is farther from the wall than desired) and Δe_x (error in distance) = N (that is, the angle of orientation of the robot is such that at the robot has to be rotated clockwise to bring it back to normal course). Therefore the control signals (output of the FLC) have to be such that the robot is aligned back to a straight course so that errors in distance and angle of orientation will be zero. The rules are defined as Δw_r = S and Δw_l = F, that is the angular velocity of the right wheel is slower than the nominal speed and the left

wheel's angular velocity is faster than the nominal speed. Thus the robot's resultant angular velocity would be to rotate it to its right, thus taking it closer to the wall.

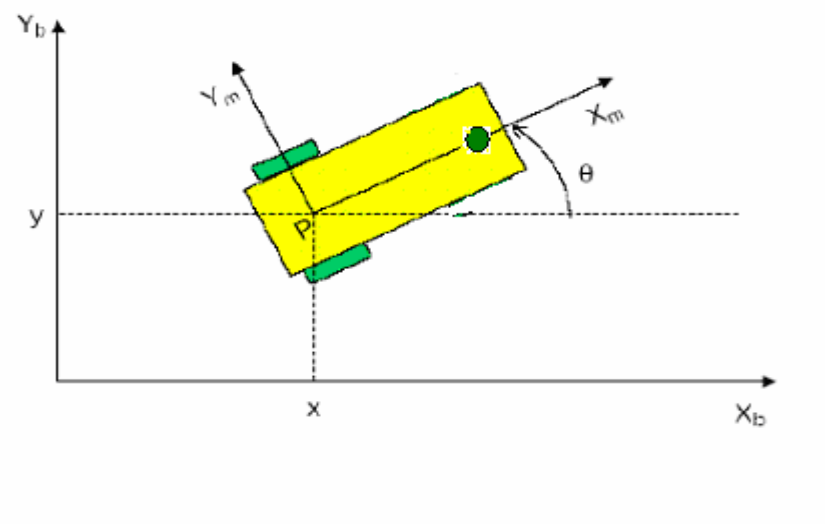


Figure 3.9 A scenario of the robot in motion[13]

The FLC's control surfaces are shown in Figures 3.10 and 3.11**.** A control surface is a plot of the controller's control signal as a function of controller's inputs. In this case, these are plots of the control signals (angular velocities of both wheels) as a function of error in angle Δe_t and error in distance Δe_x. For a linear PD controller, this surface is a plane, that is, it is completely described by a linear equation. Therefore, in order to improve upon the performance of the PD controller, the FLC control surface would be intuitively non-linear.

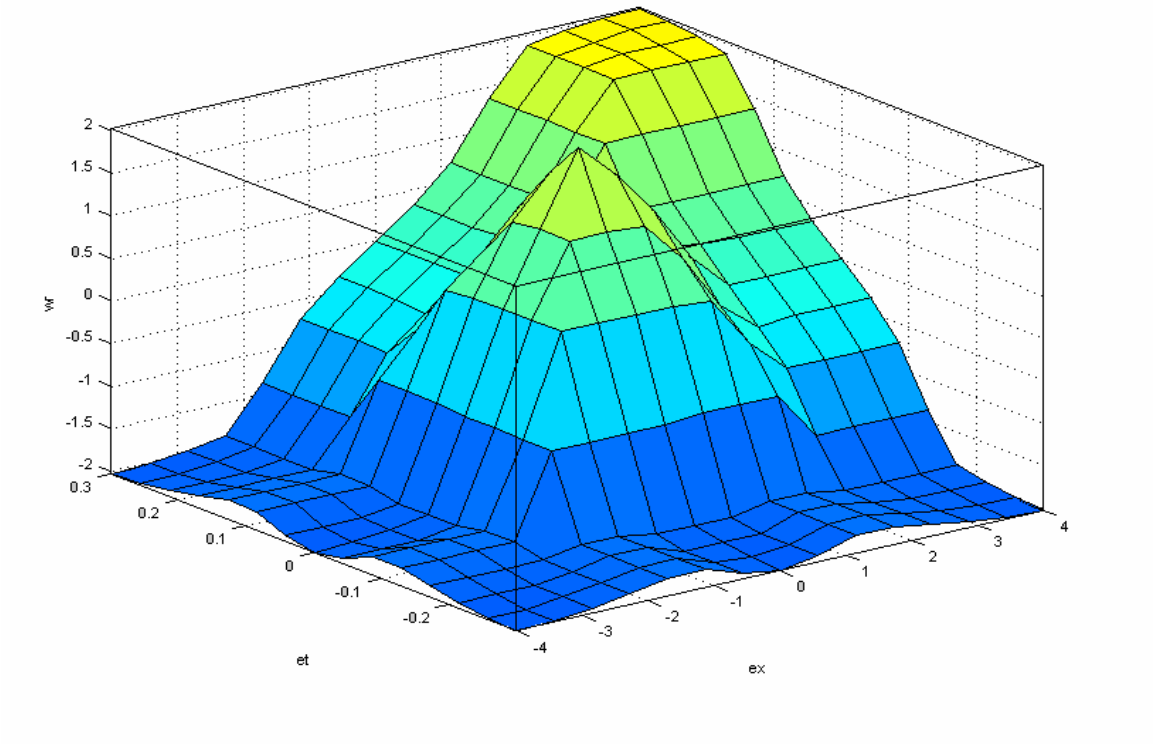


Figure 3.10 Control surface of Δw_r .

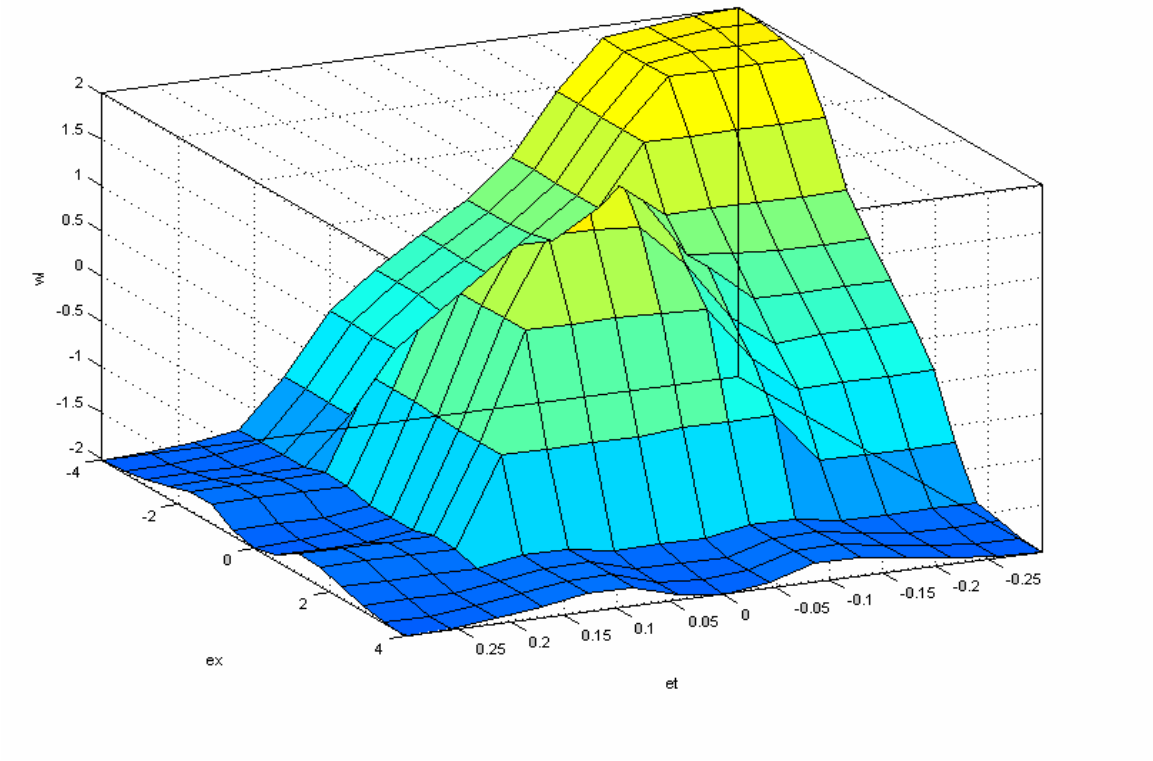


Figure 3.11 Control surface of Δw_l .

3.6.4 Fuzzification, Implication, Aggregation and Defuzzification

As described earlier, in a general fuzzy system the method for fuzzyfying inputs is to simply apply the equations in Tables 3.1 and 3.2. Once the inputs are fuzzified, the FLC knows the value of membership, or degree of truth, in each of the inputs membership functions. The antecedent of each rule must then be resolved by a fuzzy operator. The operator chosen for this application is the AND (or minimum) operator. It takes the smallest value of the maximum of the two membership functions found following fuzzification of the inputs. The minimum value is then applied to the corresponding output membership function in that particular rule. The membership function is then scaled to this value. All the final output membership functions are then aggregated together using OR (or Maximum) operator. The centroid of area (COA) method is used to produce a crisp output. This method was chosen as it produced the best results of all those tested in MATLAB. The other methods for defuzzification that were used are bisector method, mean max method, first of maxima method and last of maxima method.

3.6.5 FLC Tuning

Tuning an FLC is a daunting task as there are many parameters that can be adjusted. These include the rules, membership functions and any other gains within the control system. The overall FLC tuning process consists of the following two steps.

- Gross adjustment of the system is done by iteratively adjusting rules, membership functions and number of variables needed.
- Once gross tuning is accomplished, the FLC is fine tuned. This involves slight adjustments of individual membership functions and their ranges.

3.7 Real Time Implementation of Fuzzy Logic

In the fuzzy inference system designed in our system, the membership functions of the linguistic variables are sum normal triangular functions. That means that the sum of the membership functions of any variables at any given point is one.

3.7.1 Sum Normal Fuzzification [15]

Consider the *i* th fuzzy membership function of the *j* th input z_j. Its modal pint is denoted as c_{ij}, its lower half width as b_{ij}^-, and its upper half width as b_{ij}^+. The membership function attains a vaule of 1 when the input is c_{ij}. As the input decreases from c_{ij}, the membership function value decreases linearly to 0 at c_{ij} - b_{ij}^- and remains at 0 for all inputs les than c_{ij} - b_{ij}^-. As the input increases from c_{ij}, the membership function value decreases linearly to 0 at c_{ij} + b_{ij}^+, and remains at 0 for all inputs greater than c_{ij} + b_{ij}^+. The membership function of the *j* th crisp input z_j in its *i* th fuzzy set is given by

$$f_{ij}(z_j) = \begin{cases} 1+(z_j - c_{ij})/b_{ij}^- \\ 1-(z_j - c_{ij})/b_{ij}^+ \\ 0 \end{cases} \tag{3.17}$$

Our fuzzy system has two outputs of which we are considering only one output for notational convenience. Suppose there are a total of M rules in FLS. The consequent of the *j* th rule is a triangular fuzzy set with modular point γ_j, lower half-width as β_j^-, and upper half width β_j^+. That is, the fuzzy set of the consequent of the *j* th rule is given as

$$m_j(y) = \begin{cases} 1+(y-\gamma_j)/\beta_j^- \\ 1-(y-\gamma_j)/\beta_j^+ \\ 0 \end{cases} \tag{3.18}$$

Suppose that the *j* th rule is a consequent of z_1 belonging to fuzzy set *i* and z_2 belonging to fuzzy set k. Then the activation level of the consequent of the *j* th rule is w_j, which is given as

$$w_j = \min[f_{i1}(z_1), f_{k2}(z_2)] \tag{3.19}$$

So the fuzzy output when z_1 belongs to fuzzy set *i* and z_2 belongs to fuzzy set *k* is given as

$$\bar{m}_j(y) = w_j m_j(y) \tag{3.20}$$

The over all fuzzy output *m(y)* takes into account the possibility that each input falls into more than one fuzzy set so more than one rule can be fired at the same time.

$$m(y) = \sum_{j=1}^{M} \bar{m}_j(y) \tag{3.21}$$

The fuzzy output is mapped to a crisp number y using centroid defuzzification.

$$\hat{y} = \frac{\sum_{j=1}^{M} w_j \Gamma_j J_j}{\sum_{j=1}^{M} w_j J_j} \tag{3.22}$$

Γ_j and J_j are the centroid and the area of the *j* th output fuzzy membership function. The centroid of $m_j(y)$, the *j* th output fuzzy set, is defined as

$$\Gamma_j = \frac{\int y m_j(y) dy}{\int m_j(y) dy} \tag{3.23}$$

After substituting (2) into the above equation we obtain

$$\Gamma_j = \frac{\beta_j^+(3\gamma_j + \beta_j^+) + \beta_j^-(3\gamma_j - \beta_j^-)}{3(\beta_j^+ + \beta_j^-)} \tag{3.24}$$

Thus the final fuzzy output is $\hat{y}$ which is the speed of the right wheel in rad/sec. Similarly speed of the left wheel is also calculated.

3.7.2 Fuzzy Logic Lookup Table

Due to the hardware limitations it is not possible to achieve good results by doing these calculations on board of the autonomous robot as the system clock is only of 4MHz. It takes about 0.4 seconds to complete the whole process of fuzzification and defuzzification, thus the response of the system in real time is too slow to allow the robot to attain a steady state. Thus we have used a look up table instead. The fuzzy calculations are done offline using MATLAB for all the possible values of errors in distance and angle of orientation, and a lookup table has been compiled. Thus now only the sensor readings are to be taken and the errors in distance and angle are to be calculated. Comparing these values from the lookup table already stored on board the robot, we come up with the dynamic speeds of the two wheels thus providing a more fast and accurate response to real time changes.

CHAPTER IV

HARDWARE AND SOFTWARE IMPLEMENTATION

Embedded controller realizations are generally subject to stronger constraints than their PC-based counterparts regarding speed, memory, and the environment in which the embedded system operates. In this chapter details of the hardware used in building the robot along with the software implementation are given. The basic hardware components used, servo motors for driving the robot, ultrasonic sensors for distance measurements and microcontroller PIC16F877 are discussed in detail in Setion 4.1. In Section 4.2 the software development and implementation are discussed.

4.1 Hardware Description

The major hardware components used are described in detail in the following sections.

4.1.1 Servo Motor

A servo is a motor that is attached to a position feedback device. Generally there is a circuit that allows the motor to be commanded to go to a specified "position". Servos are constructed from three basic pieces: a motor, a feedback device, and a control board. Servos all use the same basic concept that the length of the pulse received controls the output shafts rotational position. The basic operation of the servo is illustrated in Figure 4.1.

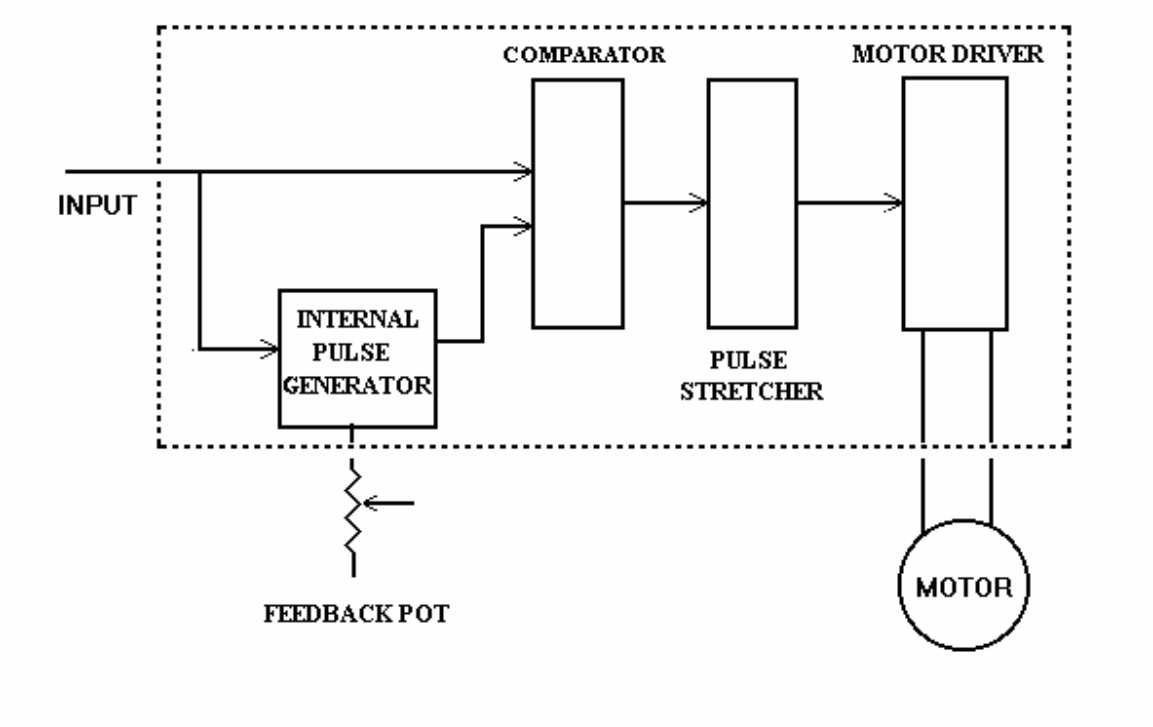


Figure 4.1 Block diagram of typical servo motor [16].

Internally within a servo is its own pulse generator which generates a pulse which varies in length based on where the output shaft is. The output shaft moves a variable resistor (pot). This varies the internal pulse generators pulse length from 1.0 to 2.0 ms with the pulse being 1.5 ms when the shaft is in the center.

The pulse generator within the servo only generates a pulse when an incoming pulse is received from the receiver. The receiver pulse triggers the servo's pulse generator. Now within the servo there are two pulses, the pulse from the receiver and the servo's internally generated pulse. Both pulses start at the same time, but depending upon the time periods, they may not end at the same time.

Let’s first look at the operation when the receiver pulse is of length 1.5 ms, and the servo shaft is centered. Under this condition the incoming pulse would trigger the servo pulse and both pulses would start at the same time, and end together 1.5 ms later. When this occurs the servo does nothing.

Now let’s say that the receiver pulse became longer. As before, the input pulse triggers the servo's internal pulse generator and the two pulses start together, but this time the input pulse is longer than the servo's internally generated pulse and they don't end together. Then the second circuit in the servo called a "pulse length comparator" comes into action. This circuit gets input from the incoming signal and the internal pulse generator, and as its name implies, compares them. The output of the comparator will be a signal which either indicates that both pulses are equal in length, the incoming pulse is longer, or the incoming pulse is shorter. How much longer or shorter the incoming pulse is versus the internally generated pulse does not matter. Now the motor driver circuit is put into action. This is the circuit which applies power to the servo motor. The servo motor is a DC motor, which means, if we connect the power to its terminals one way it will turn in one direction, and if we reverse the polarity of the connection it will run in

the opposite direction. The motor driver circuit does just that. Depending upon the signal it receives from the pulse comparator, it will either run the motor clockwise, or counter clockwise.

Let us assume that for this condition, where the input pulse is longer than the internal pulse, the motor is run clockwise. This in turn rotates the output shaft which is connected to the "feedback pot". As the output shaft rotates, the length of the servo generated pulse increases. The servo will continue to rotate until the "pulse length comparator" senses that they are both equal in length, at which time the motor driver circuitry will turn off the motor.

If we were to move the shaft a little further in the same direction, the pulse comparator would once again detect a longer input pulse and signal the motor driver circuitry to start the motor running clockwise again until the pulse lengths match.

If we were to return the shaft to neutral, the input pulse would be shortened. The pulse comparator would detect this shorter input pulse and signal the motor driver to run the motor counter-clockwise until the pulses matched.

Thus we see is that servos are "error correcting" and not "self centering". This means that the servo only moves when it receives an input pulse which does not match its internal generated pulse. If we were to lose the signal to the servo it will remain wherever it was last positioned. In order for our servos to faithfully track the transmitter shaft position, we

must constantly provide it a stream of input pulses for it to compare and adjust to. The servo receives a input pulse about every 20 ms, or about 50 times a second.

The output of the "pulse length comparator" feeds a "pulse stretcher" which in turn feeds the "motor driver circuitry". The function of this pulse stretcher is to "stretch" the small differential pulse generated by the comparator long enough for the next pulse to arrive.

4.1.2 Ultrasonic Sensors

Figure 4.2 Ultrasonic sensor SRF04 [17].

The Devantech SRF04 ultrasonic range finder, shown in the above Figure 4.2, offers precise ranging information from roughly 3 cm to 3 meters. This range, easy interfacing, and minimal power requirements make it an ideal ranger for robotics applications.

Theory of operation

The transmitter works by transmitting a pulse of sound outside the range of human hearing at 40 KHz. This pulse travel at the speed of sound away from the ranger in a cone shape and the sound reflects back to the ranger from any object in the path of sonic wave. The ranger pauses for a brief interval after the sound is transmitted and then awaits the reflected sound in the form of an echo. If received, the ranger reports this echo to the

controller and the controller can then compute the distance to the object based on the elapsed time.

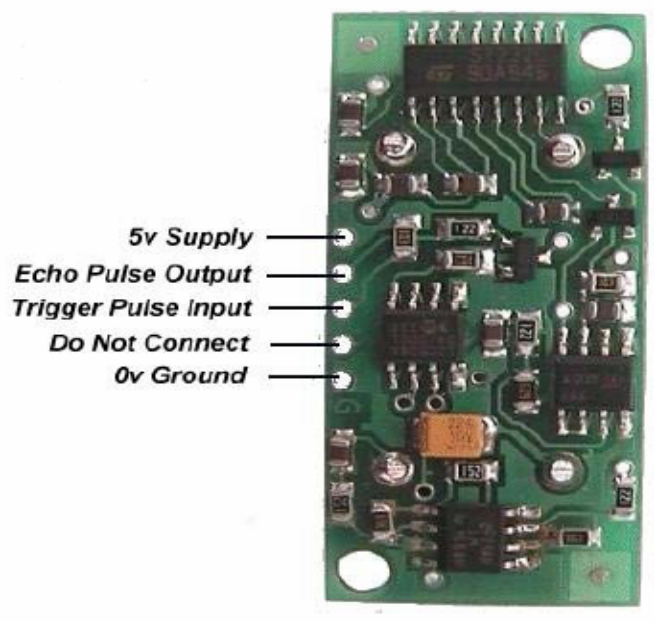


Figure 4.3 Pin connections of SRF04 ultrasonic sensor [17].

The ranger requires four connections to operate as show in Figure 4.3. Two of them are the power and ground lines. The ranger requires 5V power supply capable of handling roughly 50 mA of continuous output. The remaining two wires are the signal wires. For the ranger to work there are a couple of requirements for the input trigger and output pulse generated as illustrated in Figure 4.4. The input line should be held low (logic 0) and then brought high for a minimum of 10 microseconds to initiate the sonic pulse. The pulse is generated on the falling edge of the input trigger. The ranger's receive circuitry is held in a short blanking interval 100 microseconds to avoid noise from the internal ping and then it is enabled to listen for the echo. The echo line is low until the receive circuitry

is enabled. Once the receive circuitry is enabled, the falling edge of the echo line signals either an echo detection or timeout (if no object is detected).

The controller will begin timing on the falling edge of the trigger and end timing on the falling edge of the echo line. The duration determines the distance to the first object the echo is received from. If no object is detected, the echo pulse will timeout and return an echo at approximately 36 milliseconds. The timing diagram is shown in Figure 4.4.

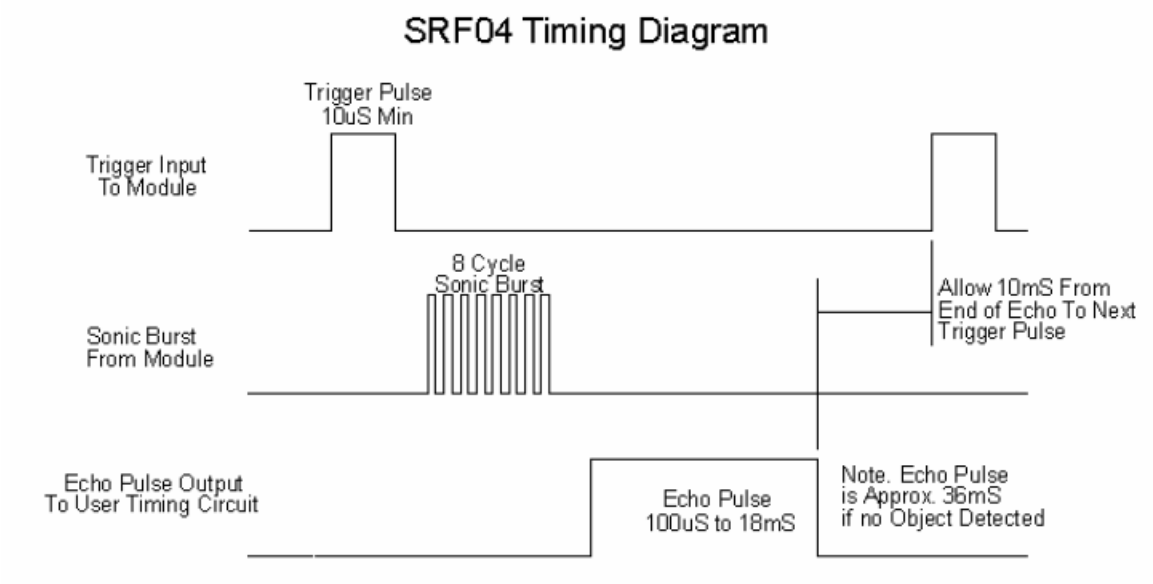


Figure 4.4 Timing diagram of the ultra sonic sensor [17].

4.1.3 PIC16F877 Microcontroller

In order to run the real-time control algorithm and create PWM signals for the two servo motors, two PIC16F877 (Microchip 8 bit microcontrollers) are used. The PIC16F877 consists of fixed-point and floating-point capability. This microcontroller combines this real-time processing capability with controller peripherals to create a suitable solution for

a variety of control system applications. The following characteristics make the PIC family a suitable choice for a wide range of processing applications:

• Flexible instruction set,

• Inherent operational flexibility,

• Comparative high-speed performance,

• Cost effectiveness.

Core Features:

The PIC16F877 features can be summarized as follows.

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input, DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory
- Up to 368 x 8 bytes of Data Memory (RAM)
- Up to 256 x 8 bytes of EEPROM Data Memory
- Pin out compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection

- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0 V to 5.5 V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:

  < 0.6 mA typical @ 3 V, 4 MHz

  20 μA typical @ 3 V, 32 kHz

  < 1 μA typical standby current

Figure 4.5 shows the pin diagram for the PIC16F877. Table 4.1 describes key features of PIC16F877.

Key Features	PIC16F877
Operating Frequency	20 MHz
Resets	POR, BOR
Flash Program Memory	8 K
Data Memory	368
EEPROM Data Memory	256
Interrupts	14
I/O Ports	Ports A,B,C,D,E
Timers	3
Capture/Compare/PWM modules	2
Serial Communication	USART
Parallel Communication	PSP
10 bit A/D module	8 input channels
Instruction set	35 instructions

Table 4.1 Key Features of the PIC16F877.

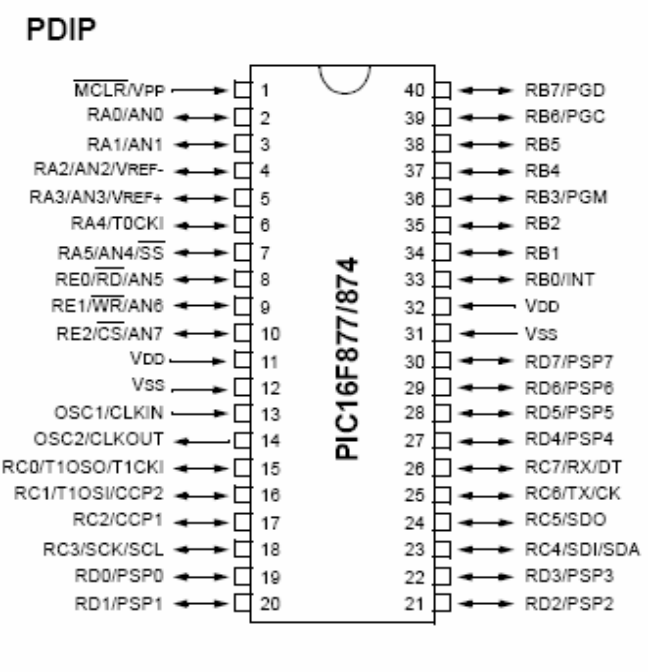


Figure 4.5 Pin Connection of PIC16F877 [18].

4.2 Software Description

This section deals specifically with the implementation of the fuzzy logic controller in the microcontroller as well as in MATLAB. The mobile robot was first simulated in SIMULINK for testing as well as tuning the fuzzy logic controller. Initially the fuzzy logic toolbox that was available was used. Then later on the required M-files were written to simulate the whole system. Initially the fuzzy logic controller was built using the fuzzy logic toolbox, taking advantage of its graphical interface. Different types of membership functions were tested and also the ranges of the membership functions can be tweaked easily to search for an optimum result. The rules developed also can be changed and the effect observed easily. Different defuzzification methods were tested and finally centroid of area method was chosen. After determining the type of membership functions, fuzzy rules and defuzzification method, an M-file was created. This brings us much closer to

programming in C language but it is possible to use this code in SIMULINK and thus possible to check for bugs if any by observing the behavior of the robot model. After it was confirmed that the code was working fine, the fuzzy logic controller was coded in C. Using C to program a microcontroller would drastically decrease the speed of execution, thus further improvement assembly language can be used.

Initially the fuzzy logic controller developed was implemented with variable ranges for membership functions. But when this was implemented in the real time system then some memory and speed constraints were faced, because the response in real time was too slow for the system to respond. Also this was necessitated as the fuzzy logic method that we implemented namely sum normal fuzzification wasn't available in the toolbox. The MATLAB code is in Appendix A.

When implementing the fuzzy logic controller in the PIC16F877, there were some memory as well as speed restrictions to be taken care of. The microcontroller was programmed using a CCS C-compiler with debugger. The C code is in Appendix B. The microcontroller uses a 4 MHz clock thus taking a large amount of time to implement the fuzzy logic controller. To overcome this problem, two PIC16F877 microcontrollers were used. One is used for reading the inputs from the sensors and calculating the fuzzy outputs, and the other for running the two motors. The algorithms for the two microcontrollers are given in Figure 4.6 and Figure 4.7. The use of two microcontrollers not only increases the speed of execution but also provides modularity to the code. It becomes easier to debug any hardware or software problems encountered. The

microcontroller connected to the sensors reads the current input, calculates the error in distance and angle of orientation and does the fuzzy calculations. If the ultrasonic sensor placed in the front reads a value less than 5 inches then the robot stops moving so as to avoid a collision. The output of the fuzzy logic controller (i.e., the pulse widths to the servo motors) are then written to the external EEPROM shared by the two microcontrollers. The second microcontroller then reads the values from the common EEPROM and changes the speed of the servo motors accordingly.

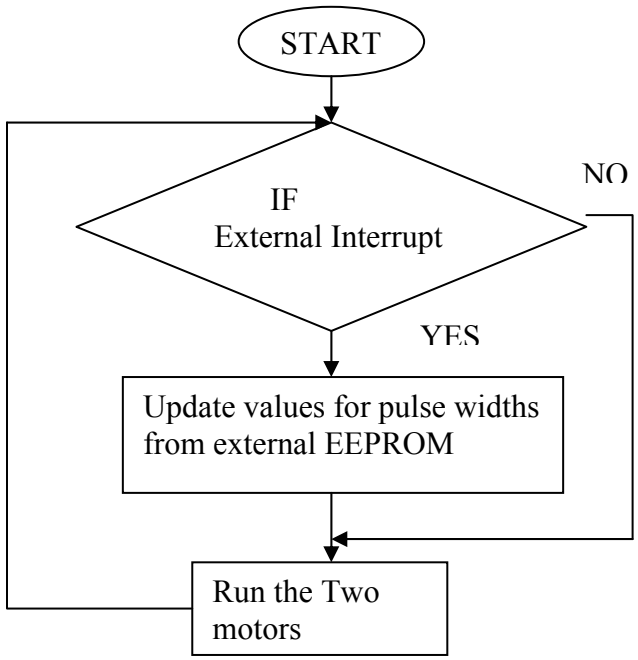


Figure 4.6 Flow chart for PIC16F877 controlling the servo motors.

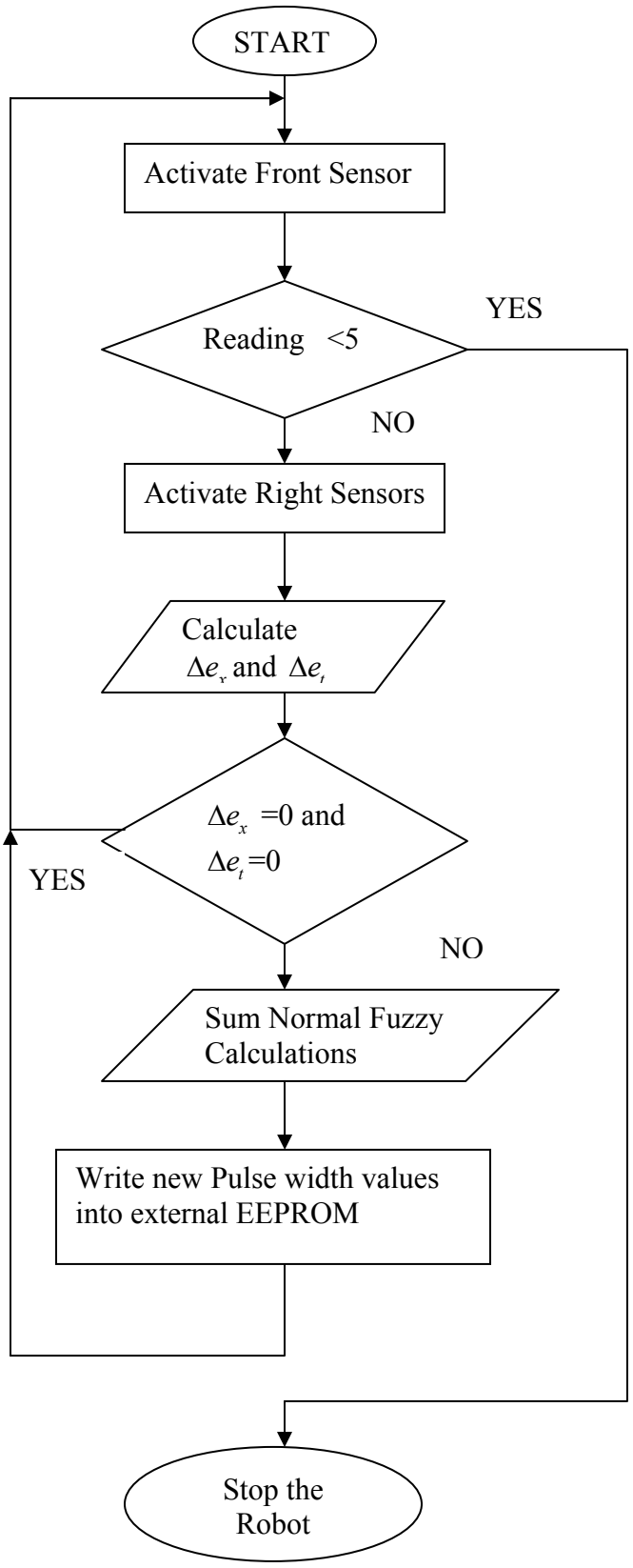


Figure 4.7 Flow chart for PIC16F877 for fuzzy calculation and sensor control.

CHAPTER V

RESULTS

In this chapter the various results obtained during the course of the thesis are presented and discussed in detail. In Section 5.1 the Simulink model of the robot, built with the help of the mathematical equations discussed in Chapter 2, is given. In Section 5.2 two types of controllers (FLC and P controller) are compared. In Section 5.3 the results obtained while using non sum normal FLC are discussed. In Section 5.4 the results obtained while using sum normal fuzzy rules are discussed. In Section 5.5 the results with the two different types of membership functions are compared. Finally in Section 5.6 the results observed using all three controllers (namely P controller, FLC with non sum normal MFs, and FLC with sum normal MFs) are compared.

5.1 Simulink Model of the Autonomous Robot

The mathematical modeling of the robot was discussed in detail in Chapter 2. Using the kinematic and dynamic equations of the robot a Simulink model was developed to test the

feasibility of using a controller and for validating the results seen in real time. The Simulink model developed is shown below in Figure 5.1. The fuzzy block in the Simulink model is a customized MATLAB M-file function block replacing the fuzzy logic toolbox. The measurement noise observed in the sensors is shown in the Simulink block diagram as white noise, with a standard deviation of 1 inch. The error in angle is measured as the difference between the two sensors on the side of the robot, hence it is measured in inches. This was done because calculating the actual angle would require using trigonometric functions which would take a long time to calculate or require large memory space.

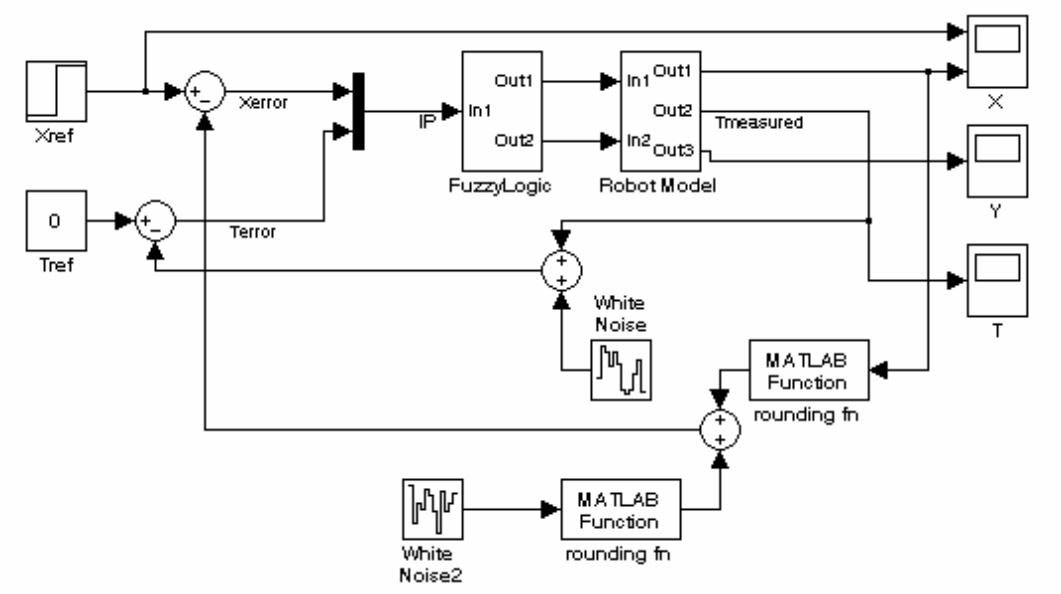


Figure 5.1 Simulink model of the autonomous robot.

The output variables *X*, *Y*, and *T* are the two Cartesian coordinates of the robot position, along with its angular orientation. The *Xref* input is intended to keep the robot a certain distance away from the wall which it is following. The *Tref* input is zero, as it is intended that the orientation of the robot be maintained at a zero angle relative to the wall.

The Simulink model shown above in Figure 5.1 was developed using the graphical interface offered by the MATLAB. To represent the whole system more realistically an M-file was written, to have further flexibility to model the sensors and any delays that we might want to incorporate, taking into account the robot's real time implementation. Also the fuzzy logic block inside the Simulink model was initially built using the fuzzy logic toolbox and later an M-file was written to simulate the actual process taking place in the microcontroller.

5.2 Comparison of FLC and P Controller

Figures 5.2 and 5.3 compare the response of the robot model when using a Fuzzy Logic Controller (FLC) and P (proportional) controller (trained using a genetic algorithm). The P controller was tuned for optimum results using a genetic algorithm. The robot is asked to maintain a distance of 4 inches from the wall. It is seen that the robot starts from zero and aligns itself to the reference distance. Figure 5.2 shows the angle of orientation of the robot as a function of time. Figure 5.3 shows the distance from the wall as the robot tries to reach the reference distance. The solid lines represent the response of the robot with a FLC and the dashed lines represent the response of the robot with a P controller. It is obvious that the FLC is performing much better than the P controller, as it is seen that the robot reaches a steady state much earlier with a FLC than a P controller. Thus the FLC controller was pursued with interest [19].

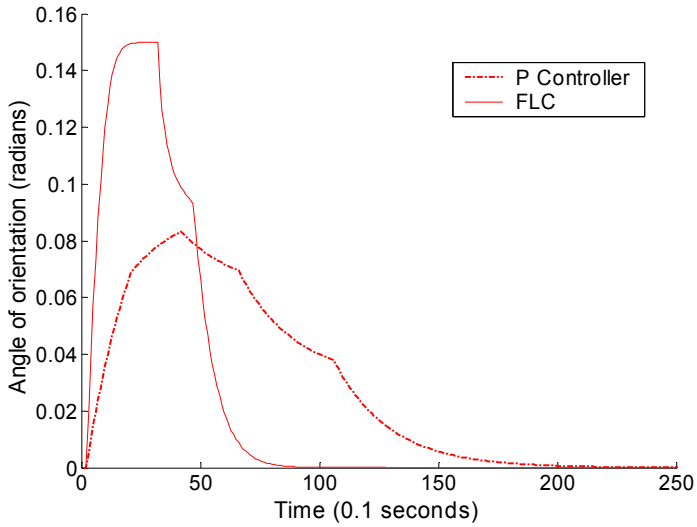


Figure 5.2 Angle of orientation as a function of time.

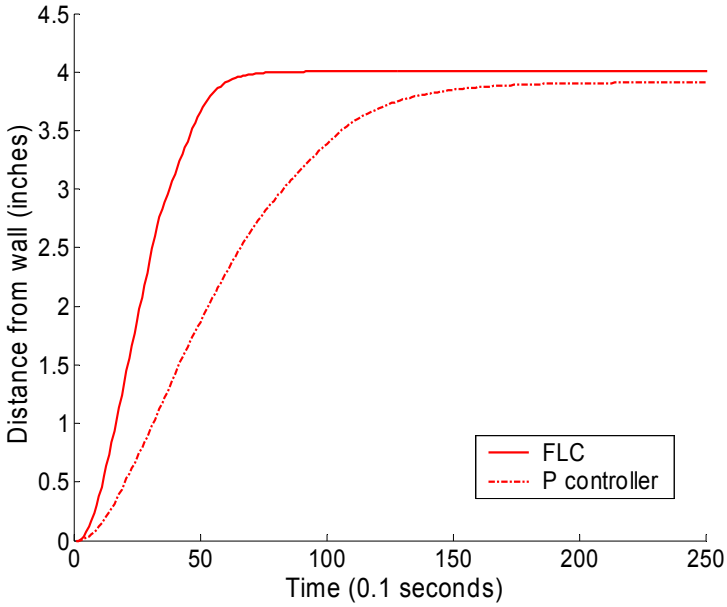


Figure 5.3 Distance from wall to the robot as a function of time.

5.3 Results Obtained Using FLC with Non Sum Normal Fuzzy Membership Functions

In this section the results observed using the various combinations of robot model and fuzzy controller are shown. The membership functions used here are not sum normal. That is the membership functions have variable ranges thus increasing the computational effort of the fuzzy logic controller. Initially a Simulink model was built for the robot and later for more generalization an M-file was written using the mathematical model. Also the fuzzy logic controller was initially modeled using the fuzzy logic toolbox and later an M-file was created to be more problem specific and to bring us nearer to real time implementation of the controller on the PIC16F877 microcontroller.

5.3.1 Simulink Model for Robot and Fuzzy Logic Toolbox for Fuzzy Rules

Figure 5.4 shows that the robot reaches the reference which is set a distance of 4 inches from the wall, very smoothly. The response of the system is slow probably because of the fuzzy toolbox that has been used to simulate the fuzzy logic controller. This could be caused because of the membership functions used in the toolbox. The membership functions of used in the toolbox have variable ranges thus it takes longer to solve the fuzzy rules and calculate the fuzzy output.

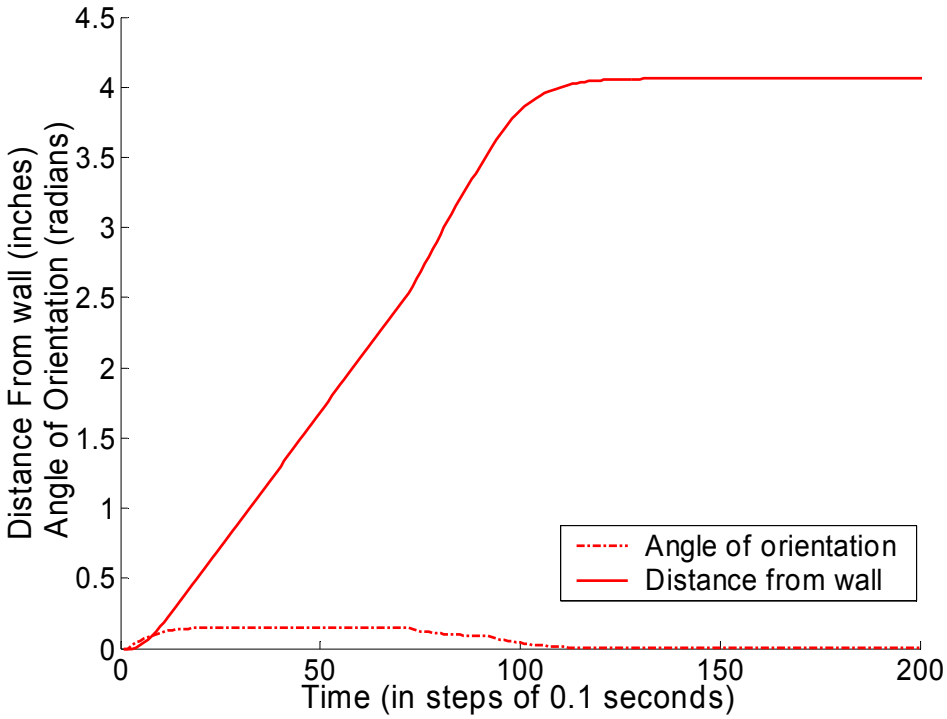


Figure 5.4 Distance from wall and angle of orientation of the robot as a function of time.

5.3.2 Simulink Model for Robot and M-File for Fuzzy Rules

Figure 5.5 shows simulation results when the fuzzy logic controller is built using an M-file and the Simulink model of the robot is used. There is a noticeable decrease in time to achieve steady state (seen from comparing Figures 5.4 and 5.5), because of the use of M-file instead of the fuzzy toolbox. This model simulates the scenario where the robot in real time is running with a lookup table in the microcontroller, as there is no time delay added, to take into account the processing time of the fuzzy rules in the microcontroller. Also the sensors are not simulated according to real time; i.e., there are no restrictions placed on the sensor readings thus giving a very smooth control. The physical sensors'

minimum resolution is in inches but the system above uses the real analog values calculated from the mathematical equations at each time step.

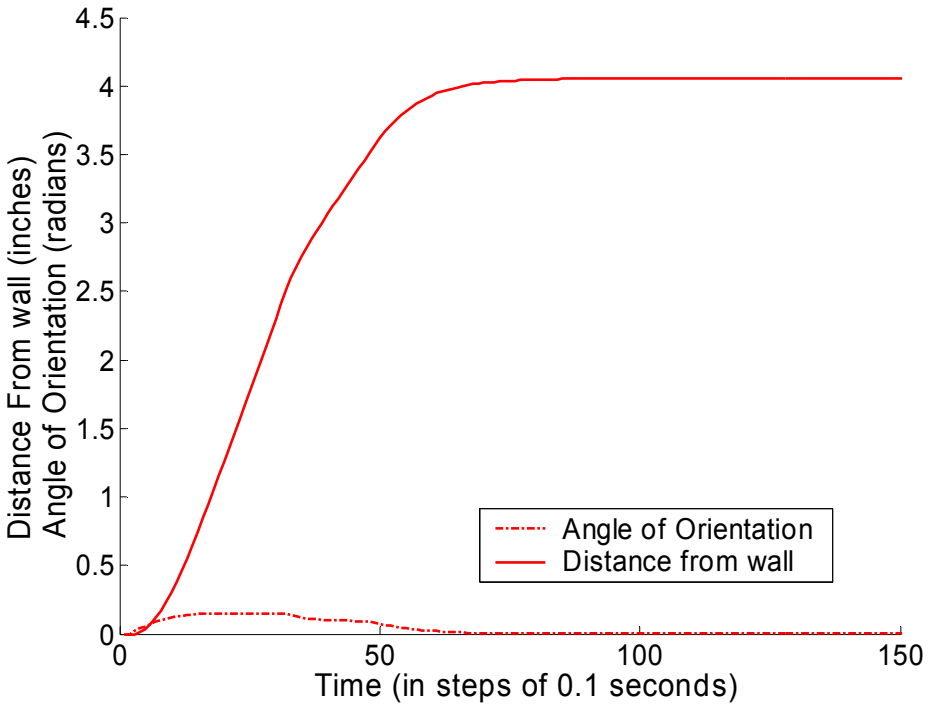


Figure 5.5 Distance from wall and angle of orientation of the robot as a function of time.

5.4 Results Obtained Using FLC with Sum Normal Fuzzy Rules

The results obtained using non sum normal fuzzy rules as seen above are fine, but due to the computational effort required and the time delay, we need to use sum normal fuzzy membership functions in our real time implementation. In this section the results obtained using sum normal membership functions in the fuzzy logic controller are discussed. The sensor and microcontrollers hardware limitation are added in the simulations to observe the response of the system in real time.

5.4.1 Zero Time Delay for Fuzzy Logic and Analog Sensor Values

Figure 5.6 shows results obtained using sum normal fuzzy logic membership functions. This simulates the scenario where the robot in real time is running with lookup table in the microcontroller, as there is no time delay added, to take into account the processing time of the fuzzy rules in the microcontroller. Also the digital character of the sensor readings are not simulated, thus giving a very smooth control. The physical sensors' minimum resolution is in inches but the system above uses the real values calculated from the mathematical equations at each time step.

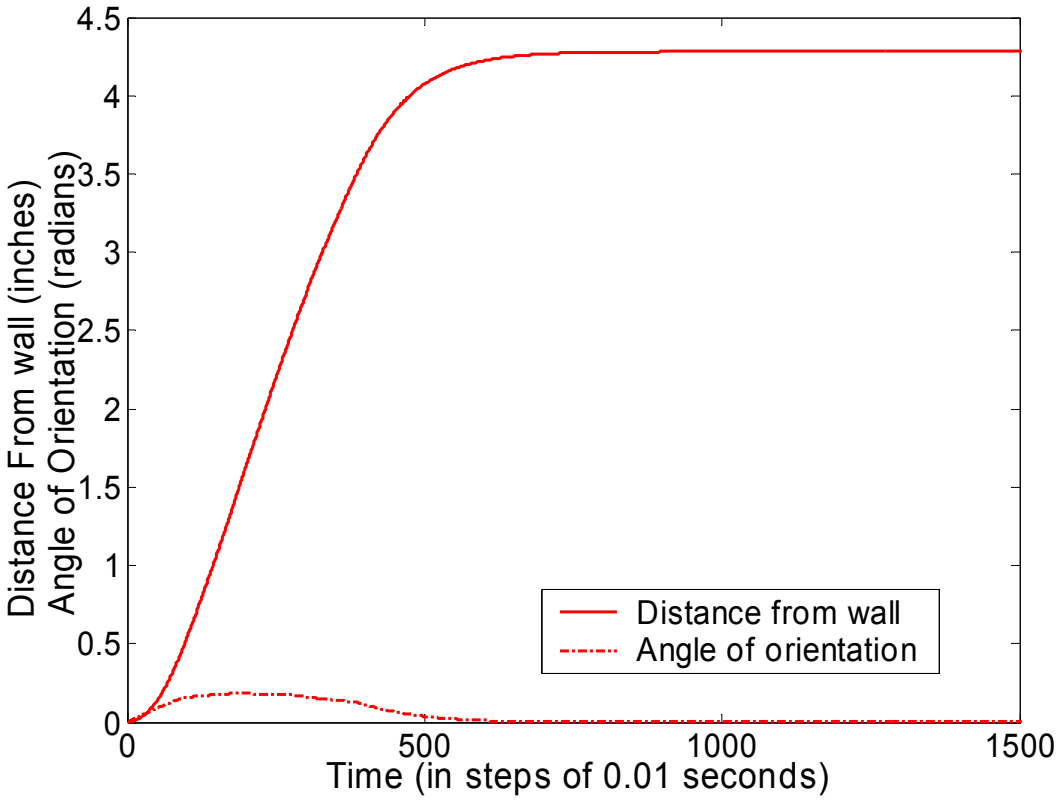


Figure 5.6 Distance from wall and angle of orientation of the robot as a function of time.

5.4.2 Zero Time Delay for Fuzzy Logic and Rounded Sensor Values

Figure 5.7 shows results obtained while simulating the hardware limitations of the ultrasonic sensors. The output from the sensors are rounded to the nearest inch. In this scenario the robot in real time is running with a lookup table in its microcontroller, as there is no time delay added to take into account the processing time of the fuzzy rules in the microcontroller. The offset noticed from the reference value of 4 inches is due to the sensors' resolution limitations.

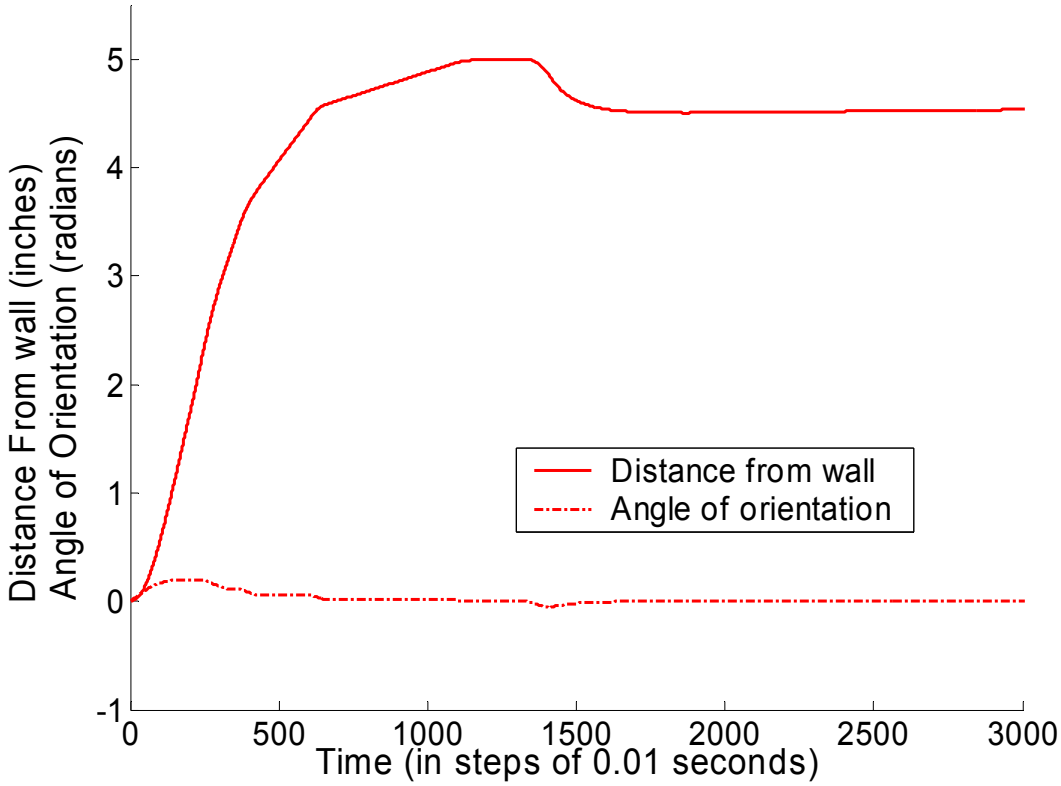


Figure 5.7 Distance from wall and angle of orientation of the robot as a function of time.

5.4.3 Time Delay for Fuzzy Logic and Rounded Sensor Values

Figure 5.8 shows results of the scenario where the robot in real time is running with the actual fuzzy calculations done onboard the robot in the microcontroller. This is made possible by adding a delay in the model of 360 milliseconds, which is the time taken by the microcontroller to complete the fuzzy calculations. Thus we see that the system takes a longer time and also has some oscillations.

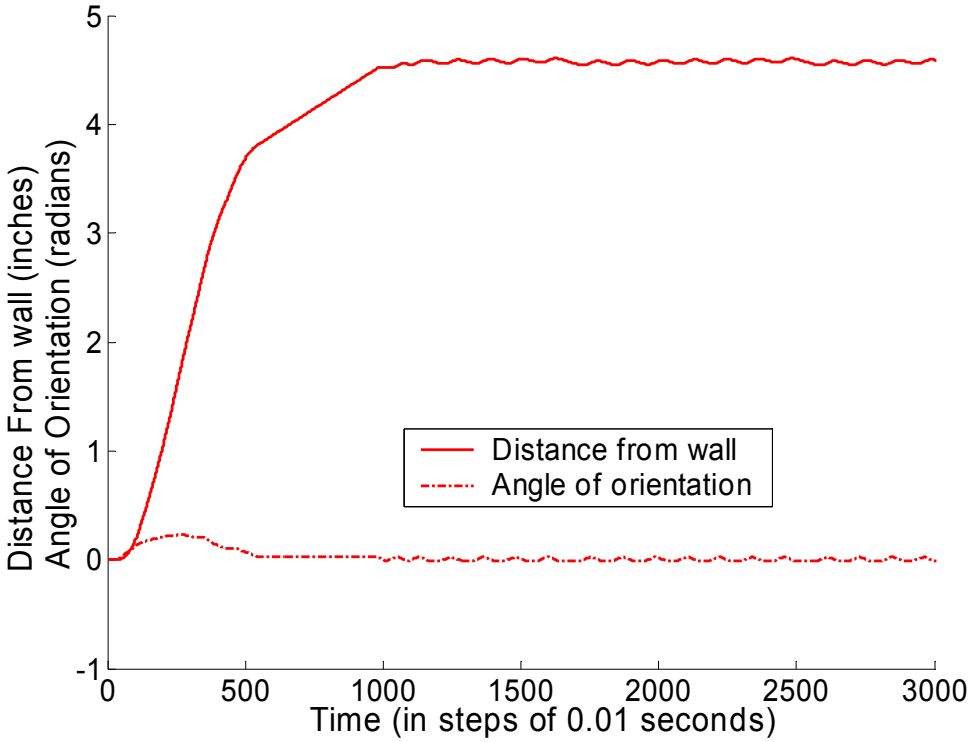


Figure 5.8 Distance from wall and angle of orientation of the robot as a function of time.

5.5 Comparison of Results Obtained Using FLC with Sum Normal Membership Functions and Non Sum Normal Membership Functions

In the course of the thesis two different types of membership functions were used with the same set of rules. These are shown in Figures 5.9 and 5.10. Their results are compared when applied to the robot model we have. The plots in Figures 5.11 and 5.12 compare the response of the robot model when using a Fuzzy Logic Controller (FLC) with sum normal and non sum normal membership functions. Figure 5.11 shows the angle of orientation of the robot as a function of time. Fig. 5.12 shows the distance from the wall as the robot tries to reach the reference distance. The solid lines represent the response of the robot with a sum normal MFs and the dashed lines represent the response of the robot with non sum normal MFs. It can be easily noticed that the non sum normal MFs outperform the sum normal MFs but the time and space restraints posed by the non sum normal MFs make sum normal MFs more attractive for real time applications where speed is more of a factor than accuracy. The PIC16F877 microcontroller needs around 800 milliseconds to complete fuzzy calculations when using non sum normal MFs when compared to 370 milliseconds needed with sum normal MFs.

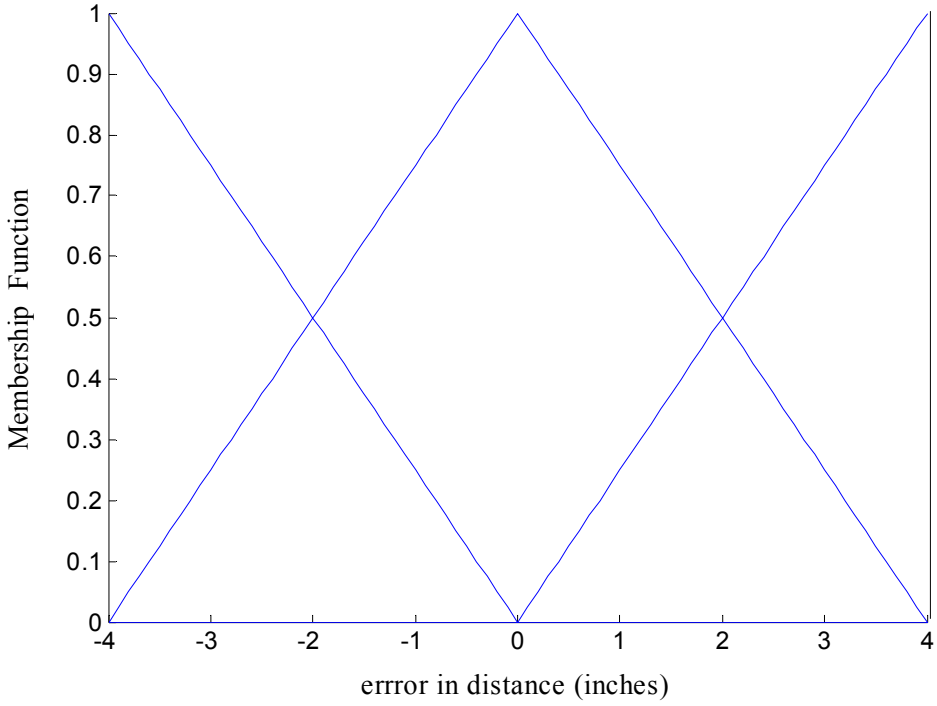


Figure 5.9 Sum normal membership function of error in distance Δe_x.

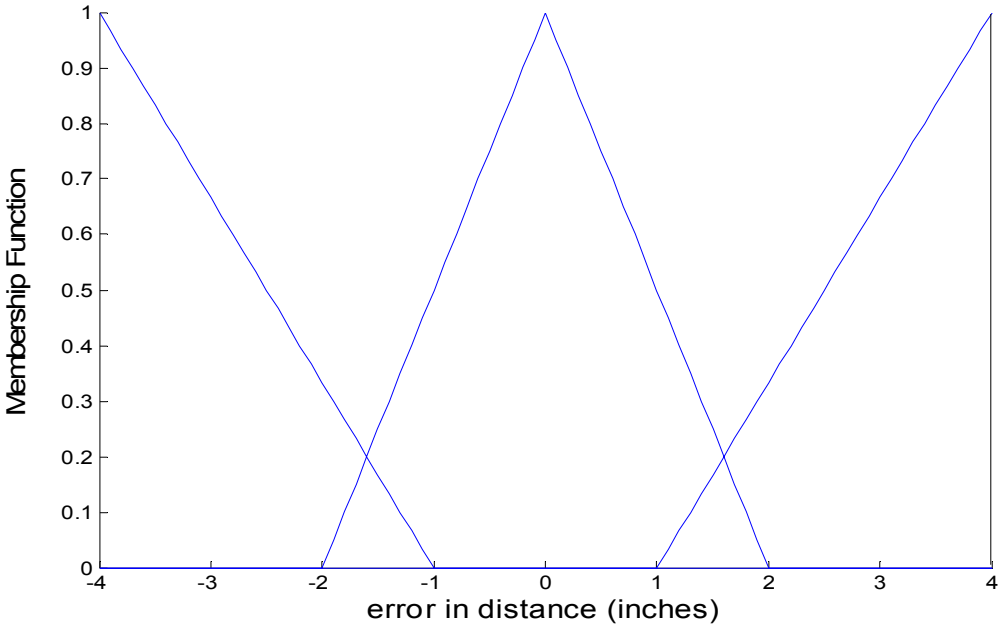


Figure 5.10 Non sum normal membership function of error in distance Δe_x.

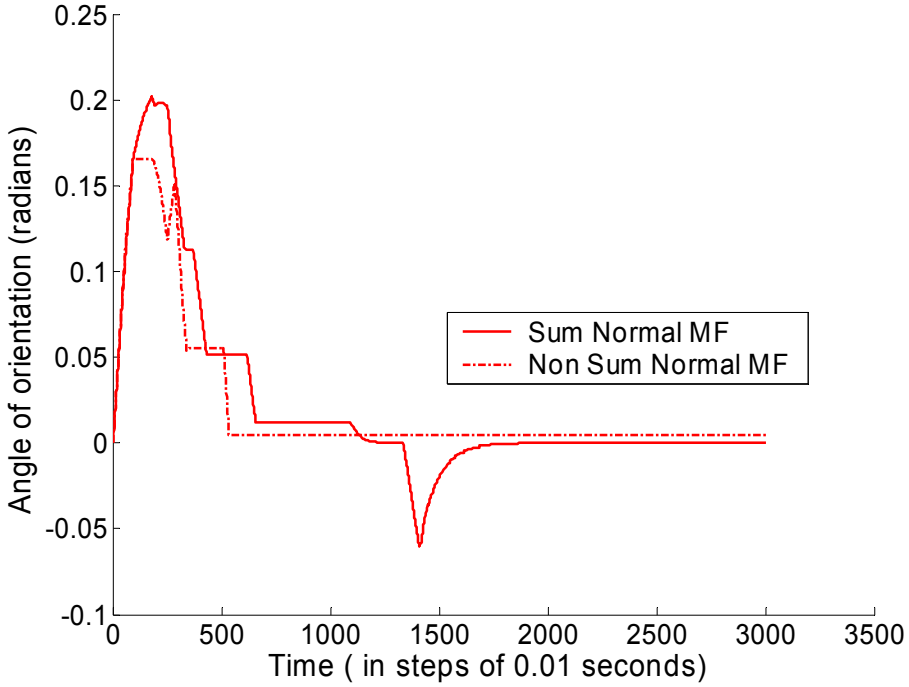


Figure 5.11 Angle of orientation of the robot as a function of time.

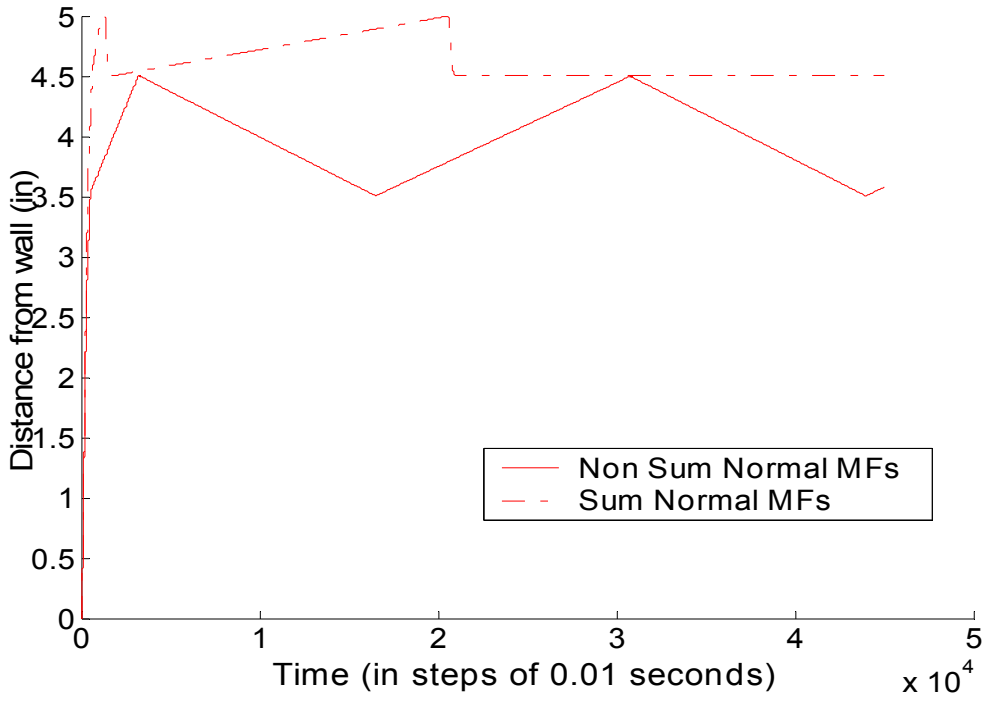


Figure 5.12 Distance from wall to the robot as a function of time.

5.6 Comparison of P Controller, FLC with Sum Normal Membership Functions and FLC with Non Sum Normal Membership Functions

The results obtained using the various controllers and different implementation methods are tabulated below in Table 5.1

		Without lookup table		With lookup table	
	P Controller	FLC (Non sum normal MFs)	FLC (Sum normal MFs)	FLC (Non sum normal MFs)	FLC (Sum normal MFs)
Rise time x (distance from wall) (seconds)	7.32	4.23	3.73	3.49	3.49
Rise time θ (angle of orientation) (seconds)	4.26	0.53	0.78	0.77	0.89
Peak value Δe_t (radians)	0.0832	0.1657	0.2070	0.1500	0.1860
Peak value Δe_x (inches)	3.91	4.5004	4.9992	4.0034	4.2809
Steady state error (Δe_x) (inches)	0.0896	0.4971	0.4972	0.0034	0.2809
Steady state error (Δe_t) (radians)	0	0	0	0	0
Computational time (MATLAB) (seconds)	0.0005	1.1344	0.088	0.0013	0.0013
Computational time (PIC16F877)(seconds)	N/A	1.904	0.186	0.051	0.051
Memory requirements (MATLAB) (lines)	10	584	237	420	420

Table 5.1 Comparison of P, non sum normal and sum normal fuzzy controllers

The first two rows in Table 5.1 compare the rise times of the response of the autonomous robot with different controllers. The rise time is taken as 70% of the peak value reached by both x (distance from wall in inches) and θ (angle of orientation in radians). It can be seen that the FLC with sum normal and non sum normal membership functions when

used take a large amount of time when compared to the case when a lookup table is used. The P controller has a higher rise time than a FLC controller.

Peak values reached by both x (distance from wall in inches) and θ (angle of orientation in radians) are lower in the case when non sum normal membership functions are used. It is seen that P controller has the lowest peak value of 3.91 but since the reference value to be reached is 4 inches the system is overdamped.

The steady state error is seen to be very large in the case of FLC without using the lookup table. This is because of the delay in the response of the system caused by the fuzzy calculations on board the robot. When the lookup tables are used it seen that the steady state error is reduced and smaller than the P controller.

The computational time in MATLAB is given for an iteration of the controller. The FLC with non sum normal fuzzy rules is the slowest with each simulation step taking over 1.344 seconds. This is due to the fuzzy calculation with membership functions with variable ranges. The computational time for both FLC controllers with lookup tables is the same with an average time of 0.0013 seconds. The P controller takes the least time as it has very few equations to solve and also it doesn't have to go through a lookup table to give an output.

The memory requirements are the actual code length in the M-files needed to implement that particular controller. The P controller needs the least amount of lines as there are

only three equations to be used. But the actual task of training a P controller takes a long time when done manually or using a genetic algorithm which uses a complex algorithm to find the best gains for the controller. The FLC with non sum normal membership functions is huge compared to the FLC using sum normal membership functions.

From the above results we can conclude that a controller should be selected according to the constraints posed by a particular problem. If we want a small steady state error and we want to reduce the overshoot in the system, then FLC with non sum normal fuzzy membership functions should be used. If memory size is the primary consideration then a P controller can be used as it has minimum memory requirements. If computational time is of paramount interest then a P controller or any of the FLCs with a lookup table could be used.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This thesis's main objective was to develop an optimal path controller for an autonomous mobile robot. The first step in approaching this problem was to build the highly nonlinear dynamic model of the mobile robot. This was done using the Simulink toolbox available in MATLAB. The next step was to decide which type of controller was to be used for the path control of the robot. Comparing the traditional P controller and the fuzzy logic controller it was noticed that the fuzzy logic controller outperforms the P controller. The P controller was tuned by a genetic algorithm for optimum gains. Thus it was decided to use the fuzzy logic controller.

Initially when the fuzzy logic controller was designed non sum normal fuzzy membership functions were used. The resultant controller worked fine but when it was implemented on hardware some inherent problems came to light. It was observed that the PIC16F877 microcontroller took a large amount of time to complete the fuzzy calculations thus

degrading the performance of the controller. The response of the system was too slow in real time to actually control the motion of the robot. This was mainly because of the variable ranges of the membership functions used in the fuzzy rules. New sum normal membership functions were used to build the fuzzy logic controller to overcome this problem. It was seen that this improved the performance of the mobile robot, but still the calculations were taking too long a time due to the speed limitations of the microcontroller used. Thus a lookup table was developed offline and was programmed into the microcontroller, thereby reducing the processing time tenfold. Thus a fuzzy logic controller was implemented in real time to build an autonomous wall following robot.

The modeling of a fuzzy controller is an iterative process of trial and error. It involves fine tuning of the shapes of membership functions as well as the fuzzy rule base. For relatively complex control problems this synthesis procedure can turn out to be lengthy depending on the availability of expert knowledge or intuition. This serves to weaken the scalability of one of the strongest attributes of fuzzy logic applications in control – fast development time. Moreover, it is not always obvious what the best combination of fuzzy input-output resolution, membership functions, and rules should be for a given system. Hence, the likelihood of obtaining an optimal fuzzy behavior as a result of trial and error synthesis is low. The establishment of more systematic approaches to fuzzy controller synthesis in the absence of an expert, or sufficient knowledge of the problem domain, is currently an open problem. Various attempts have been made to address this issue. These include the determination of fuzzy membership functions and rules by optimization of search using genetic algorithms, and by learning using neural networks.

The fuzzy logic controller was initially built with its two inputs being the distance from wall at that instant and the rate of change of error in distance. The response of the system wasn't as expected thus the two inputs were changed in this thesis to error in distance and error in angle of orientation. Later the membership functions were changed from variable ranges to sum normal membership functions.

In the implementation of the fuzzy logic controller the microcontroller used was an eight bit PIC16F877 with a 4 MHz clock. To improve the results obtained we could use a better microcontroller like a Microchip dsPIC which is a 16 bit microcontroller with larger RAM and higher speed of instruction execution. Also the microcontroller was programmed using a C compiler thus reducing the efficiency of the code. To increase its efficiency the microcontroller could be programmed in assembly language. Genetic algorithms could be used to improve the rules and also the membership functions for optimum results.

BIBILIOGRAPHY

[1] R. Frizera Vassallo, H. J. Schneebeli, J. Santos-Victor, Visual navigation: combining visual servoing and appearance based methods, 6th International Symposium on Intelligent Robotic Systems, SIRS'98, Edinburgh, Scotland, July 1998.

[2] J. Santos-Victor, G. Sandini, F. Curotto, S. Garibaldi, Divergent stereo in autonomous navigation: from bees to robots, International Journal of Computer Vision 14 (1995) 159–177.

[3] R. Carelli, C. Soria, O. Nasisi, E. Freire, Stable AGV corridor navigation with fused vision-based control signals, Conference of the IEEE Industrial Electronics Society, IECON, Sevilla, Spain, November 5–8, 2002.

[4] A. Dev, B. Kröse, F. Groen, Navigation of a mobile robot on the temporal development of the optic flow, IEEE/RSJ/GI International Conference on Intelligent Robots and Systems IROS'97, Grenoble, September 1997, pp. 558–563.

[5] R. Benporad, M. Di Marco, A. Tesi, Wall-following controllers for sonar-based mobile robots, 36th IEEE Conference on Decision and Control, San Diego, CA, 10-12 Dec. 1997, pp. 3063-3068.

[6] Ole Jakob Sordalen, Feedback Control of Nonholonomic Mobile Robots, doctoral thesis, Department of Engineering Cybernetics, The Norwegian Institute of Technology, 1993.

[7] Danny Ratner, Phillip McKerrow, Navigating an outdoor robot along continuous landmarks with ultrasonic sensing, Robotics and Autonomous Systems 45 (2003) 73–82.

[8] Ricardo Carelli Eduardo Oliveira Freire, Corridor navigation and wall-following stable control for sonar-based mobile robots, Robotics and Autonomous Systems 45 (2003) 235–247

[9] Mohammad Jamshidi et al. (eds.), Applications of fuzzy logic: Towards high machine intelligence quotient systems, Prentice-Hall, NJ, 1997.

[10] http://www.manufacturing.net/ctl/article/ca372359

[11] L. A. Zadeh, Fuzzy sets, Information and Control 8 (1965) 338–353.

[12] E. H. Mamdani, S. Assilian, Application of fuzzy algorithms for control of simple dynamic plant, Proceedings of the Institute of Electrical Engineers 121 (1974) 1585–1588.

[13] M.I. Rieiro, P. Lima "Kinematics Models of Mobile Robots," http://omni.isr.ist.utl.pt/~mir/cadeiras/robmovel/Kinematics.pdf

[14] J. S. R. Jang, N. Gulley, Fuzzy Logic Toolbox User's Guide, The Math Works, Inc., 1995

[15] Dan Simon, "Sum normal optimization of fuzzy membership functions," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10, no. 4 (2002) 363-384.

[16] http://www.rchelibase.com

[17] http://www.acroname.com

[18] http://www.microchip.com

[19] Vamsi Mohan Peri and Dan Simon, Fuzzy Logic Controller for an Autonomous Robot, North American Fuzzy Information Processing Society Conference, 2005.

APPENDIX

APPENDIX A

PROGRAM IN C IMPLEMENTING FUZZY LOGIC CONTROL WITH SUM NORMAL MEMBERSHIP FUNCTIONS IN THE PIC16F877

```
// code for fuzzy logic and sensors
// This sends the new values to motors to control their speeds.
// this is the main program as of now march 23rd
#include "C:\PICC\robot\r2.h"
#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <2401.c>

//Global Variables
int sflag=0;

// FUNCTIONS FOR ACCESSING INTERNAL AND EXTERNAL EEPROMS
#SEPARATE void write_float_eeprom(long int ,float);
#SEPARATE float read_float_eeprom(long int );
#SEPARATE void write_float_exteeprom(BYTE,long int);
#SEPARATE void write_int_exteeprom(BYTE,int);


// FUNCTIONS FOR MANIPULATION OF DATA
#SEPARATE float min(float,float);
#SEPARATE float max(float,float);
#SEPARATE float getdistance(float,float);
#SEPARATE float getangle(float,float);
#SEPARATE long int conv_pulse(float,short int);

//FUNCTION FOR ACTIVATING ULTRASONIC SENSORS

#SEPARATE int ultrapulse(int);

//FUNCTIONS USED IN SUMNORMAL FUZZIFICATION

#SEPARATE float calcmfex(float ,float ,int);
#SEPARATE void calcmfet(int ,float ,int , float);
#SEPARATE void rulebase(int ,float , int);



void main()
```

```
{
  float r1=0,r2=0,edistance,etheta;
  int control,selectwheel;
  long int pulse,pulse2;
  BYTE address;

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_1);
  setup_timer_2(T2_DIV_BY_1,0,1);

  output_a(1);
  while(true)
  {
    output_a(control);

    r1=ultrapulse(5);    // distance in front
    if(r1<4)
    {
      output_high(PIN_D4);    //switches ON the transistors
      control=1;        //STOP the robot
      address=0x08;
      write_int_exteeprom(address,control);
      output_low(PIN_D4);     //switches OFF the transistors
      output_high(PIN_D5); //connected to RB0 of second PIC
      output_low(PIN_D5);
    }
    else
    {
      r1=ultrapulse(1);    //Rfront distances
      r2=ultrapulse(0);    //Rback  distances

      edistance=getdistance(r1,r2);     //edistance calculated
      etheta=getangle(r1,r2);        //etheta  calculated

      if(edistance==0 && etheta==0)
      {
        pulse = 1400;      //for rmotor
        pulse2= 1700;    //for lmtor
        address=0x00;
        write_float_exteeprom(address,pulse);
        address=0x04;
        write_float_exteeprom(address,pulse2);
        control=5;
```

```
          address=0x08;
          write_int_exteeprom(address,control);
          output_low(PIN_D4);     //switches OFF the transistors
        }
        else
        {
          selectwheel=0;
          r1=calcmfex(edistance,etheta,selectwheel);         //change in WR is calc
          selectwheel=1;
          r2=calcmfex(edistance,etheta,selectwheel);         //change in WL is calc

          r1=r1+4;            //actual WR in radians
          r2=r2+4;            //actual WL in radians

          output_high(PIN_D4);    //switches ON the transistors
          pulse=  conv_pulse(r1,0);      //for rmotor
          pulse2= conv_pulse(r2,1);    //for lmtor
          address=0x00;
          write_float_exteeprom(address,pulse);
          address=0x04;
          write_float_exteeprom(address,pulse2);
          control=5;
          address=0x08;
          write_int_exteeprom(address,control);
          output_low(PIN_D4);     //switches OFF the transistors
        }
      }

      output_high(PIN_D5); //connected to RB0 of second PIC
      output_low(PIN_D5);
    }       //end of while
}         //end of main

// INTERRUPT FOR SENSORS
#int_ccp1
void isr()
{
  sflag=1;
}

//CALCULATES THE ERROR IN DISTANCE OF THE ROBOT WITH SENSOR
INPUTS
#SEPARATE float getdistance(float r1,float r2)
{
  float distance,edistance;
```

```
  distance= max(r1,r2)+abs(r1-r2)/2;
  edistance=8-distance; //8 is ref distance

  if(edistance>4)
    edistance=4;
  if(edistance<-4)
    edistance=-4;
  return(edistance);
}

//CALCULATES THE ERROR IN ANGLE OF THE ROBOT WITH SENSOR INPUTS
#SEPARATE float getangle(float r1,float r2)
{
  float etheta,theta;

  theta=r1-r2;
  etheta=0-theta;
  if(etheta>3)
    etheta=3;
  if(etheta<-3)
    etheta=-3;

  return(etheta);
}

//CONVERTS FUZZY OUTPUT(FLOAT) TO PULSE (LONG INT)
#SEPARATE long int conv_pulse(float r,short int f)
{
  long int k;
   if(f==0)
  {
    //k=-50*r+1550;     //for Rmotor (1500-1200)
    k=((6-r)*200/6)+1400;
    if(k>1500)
      k=1500;
  }
  else
  {
    //k=50*r+1550;      // for Lmotor (1600-1900)
    k=((r-6)*200/6)+1700;
    if(k<1600)
      k=1600;
  }
  return(k);
}
```

```
//ROUTINE FOR SELESCTING AND RUNNING THE CORRECT SENSOR
RETURNS DISTANCE(INCHES)
#SEPARATE int ultrapulse(int sensor_no)
{
  long int i=0;
  int distance;
  ldiv_t f;

  switch(sensor_no)
  {
    case 0:
    output_high(PIN_C6);              // RIGHT BACK SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C6);
    break;

    case 1:
    output_high(PIN_C1);              // RIGHT FRONT SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C1);
    break;

    case 5:
    output_high(PIN_C5);              // FRONT SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C5);
    break;

    case 4:
    output_high(PIN_C4);              // BACK SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C4);
    break;
  }
  setup_ccp1(CCP_CAPTURE_FE);             // set capture on falling edge
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_1); // set prescaler of tiemr1 to 8
  set_timer1(0);                  // re initialize timer to zero
  enable_interrupts(INT_CCP1);   // Setup interrupt on falling edge
  enable_interrupts(GLOBAL);
  while(i<10000 && sflag==0)
  {
    i=i+1;
  }


  if(i>6000)
```

```
  {
    distance=i;
    return(distance);
  }
  else
  {
    sflag=0;            //reinitialize flag to zero
    f= ldiv(CCP_1,150);
    distance=f.quot;

    delay_ms(10);             // probably not needed
    return(distance);
  }
}

//CALCULATES MEM.FN OF EDISTANCE AN ALSO FINALLY RETURNS TEH
FUZZY OUTPUT
#SEPARATE float calcmfex(float ex,float et,int selectwheel)
{

  float const edistance[3]={ -4.0, 0.0 ,4.0};
  int rule_no,selectwheel1;
  float num,denum,answer,mfex,et1;
  int num1_addr=20, denum1_addr=30;
  int num_addr=40, denum_addr=50;

  et1=et;
  selectwheel1=selectwheel;

  if(ex >= edistance[0] & ex <= edistance[1])
  {
    mfex=((ex-edistance[0])/(edistance[0]-edistance[1]))+1;  //ex is N
    rule_no=0;
    calcmfet(rule_no,mfex,selectwheel1,et1);

    num=read_float_eeprom(num1_addr);
    denum=read_float_eeprom(denum1_addr);


    mfex=1-mfex;   //ex is Z
    rule_no=3;
    calcmfet(rule_no,mfex,selectwheel1,et1);

    num=num+ read_float_eeprom(num1_addr);
    denum=denum+read_float_eeprom(denum1_addr);
  }
```

```
  else
  {
    mfex=((ex-edistance[1])/(edistance[1]-edistance[2]))+1;  //ex is Z
    rule_no=3;
    calcmfet(rule_no,mfex,selectwheel1,et1);

    num=read_float_eeprom(num1_addr);
    denum=read_float_eeprom(denum1_addr);

    mfex=1-mfex;  //ex is P
    rule_no=6;
    calcmfet(rule_no,mfex,selectwheel1,et1);

    num=num+ read_float_eeprom(num1_addr);
    denum=denum+read_float_eeprom(denum1_addr);
  }

  answer=num/denum;

  if(answer < -3)
    answer= -3;
  else if(answer > 3)
    answer= 3;

  return(answer);

}

//CALCULATES MEM.FN OF ETHETA

#SEPARATE void calcmfet(int rule_no,float mfex,int selectwheel1, float et1)
{
  float const etheta[3]={ -3, 0.0 , 3};
  int rule_no1,rule_no2,selectwheel2;
  float mfet,w,num1,denum1;
  int num2_addr=0, denum2_addr=10;
  int num1_addr=20, denum1_addr=30;

  rule_no1=rule_no;
  num1=0;
  denum1=0;
  selectwheel2=selectwheel1;

  if (et1 >= etheta[0] & et1 <= etheta[1])
  {
    mfet=((et1-etheta[0])/(etheta[0]-etheta[1]))+1;  //et is N
```

```
    rule_no2=rule_no1+1;
    w=min(mfex,mfet);
    rulebase(rule_no2,w,selectwheel2);

    num1=read_float_eeprom(num2_addr);
    denum1=read_float_eeprom(denum2_addr);

    mfet=1-mfet;   //et is Z
    rule_no2=rule_no1+2;
    w=min(mfex,mfet);
    rulebase(rule_no2,w,selectwheel2);

    num1=num1+read_float_eeprom(num2_addr);
    denum1=denum1+read_float_eeprom(denum2_addr);
  }

  else
  {
    mfet=(et1-etheta[1])/(etheta[1]-etheta[2])+1;  //et is Z
    rule_no2=rule_no1+2;
    w=min(mfex,mfet);
    rulebase(rule_no2,w,selectwheel2);

    num1=read_float_eeprom(num2_addr);
    denum1=read_float_eeprom(denum2_addr);

    mfet=1-mfet;   //et is P
    rule_no2=rule_no1+3;
    w=min(mfex,mfet);
    rulebase(rule_no2,w,selectwheel2);

    num1=num1+read_float_eeprom(num2_addr);
    denum1=denum1+read_float_eeprom(denum2_addr);
  }


  write_float_eeprom(num1_addr,num1);
  write_float_eeprom(denum1_addr,denum1);

}

//THE RULE BASE FOR SUMNORMAL FUZZYLOGIC

#SEPARATE void rulebase(int rule_no2,float w, int selectwheel2)
{
```

```
int num2_addr=0, denum2_addr=10;
float num2,denum2;

//float Narea=1.5,Zarea=3.0,Parea=1.5;
//int Ncentroid=-2,Zcentroid=0,Pcentroid=2;

switch (rule_no2)
{
  case 1:
  if (selectwheel2==0)          //RWHEEL
  {

    num2=w*1.5*(-2.0);
    denum2=w*1.5;
    // num2=w*Narea*Ncentroid;
    // denum2=w*Narea;
  }
  else                    //LWHEEL
  {
    num2=w*1.5*2;
    denum2=w*1.5;
    //num2=w*Parea*Pcentroid;
    //denum2=w*Parea;
  }
  break;
  case 2:
  if (selectwheel2==0)
  {
    num2=w*3*0;
    denum2=w*3;
    //num2=w*Zarea*Zcentroid;
    //denum2=w*Zarea;
  }
  else
  {
    num2=w*1.5*2.0;
    denum2=w*1.5;
    //num2=w*Parea*Pcentroid;
    //denum2=w*Parea;
  }
  break;
  case 3:
  if (selectwheel2==0)
  {
    num2=w*1.5*2.0;
    denum2=w*1.5;
```

```
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
else
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
break;
case 4:
if (selectwheel2==0)
{
  num2=w*3*0;
  denum2=w*3;
  //num2=w*Zarea*Zcentroid;
  //denum2=w*Zarea;
}
else
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
break;
case 5:
if (selectwheel2==0)
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
else
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
break;
case 6:
if (selectwheel2==0)
{
```

```
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
else
{
  num2=w*3*0;
  denum2=w*3;
  //num2=w*Zarea*Zcentroid;
  //denum2=w*Zarea;
}
break;
case 7:
if (selectwheel2==0)
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
else
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
break;
case 8:
if (selectwheel2==0)
{
  num2=w*1.5*2.0;
  denum2=w*1.5;
  //num2=w*Parea*Pcentroid;
  //denum2=w*Parea;
}
else
{
  num2=w*3*0;
  denum2=w*3;
  //num2=w*Zarea*Zcentroid;
  //denum2=w*Zarea;
}
break;
case 9:
```

```
    if (selectwheel2==0)
    {
      num2=w*1.5*2.0;
      denum2=w*1.5;
      //num2=w*Parea*Pcentroid;
      //denum2=w*Parea;
    }
    else
    {
      num2=w*1.5*(-2.0);
      denum2=w*1.5;
      //num2=w*Parea*Pcentroid;
      //denum2=w*Parea;
    }
    break;

  }

  write_float_eeprom(num2_addr,num2);
  write_float_eeprom(denum2_addr,denum2);


}

//CALCULATES MINIMUM OF TWO FLOATING NUMBERS

#SEPARATE float min(float r, float g)      // minimum function
{
  if (r>g)
    return(g);
  else
    return(r);
}

//CALCULATES MAXIMUM OF TWO FLOATING NUMBERS

#SEPARATE float max(float r, float g)      // maximum function
{
  if (r>g)
    return(r);
  else
    return(g);
}


// EXTERNAL EEPROM (SHARED BY BOTH PICS)
```

```
// WRITING FLOAT DATA TO EXTERNAL EEPROM

#SEPARATE void write_float_exteeprom(BYTE address,long int pulse)
{
  int i;
  long int a;
  a=pulse;
  for(i=0;i<2;i++)
  {
    WRITE_EXT_EEPROM(address+i,*(&a +i));

  }

}

// WRITING INTIGER DATA TO EXTERNAL EEPROM

#SEPARATE void write_int_exteeprom(BYTE address,int control)
{

  int a;
  a=control;

  WRITE_EXT_EEPROM(address,*(&a));


}

//  INTERNAL EEPROM
// writing FLAOT DATA to eeprom

#SEPARATE void write_float_eeprom(long int address ,float data)
{
  int p;
  for( p=0;p<4;p++)
  {
    write_eeprom(address+p,*(&data +p));
  }
}


// reading FLOAT DATA from eeprom

#SEPARATE float read_float_eeprom(long int address)
{
  int p;
```

```
  float data;
  for( p=0;p<4;p++)
  {
    *(&data+p)=read_eeprom(address+p);
  }
  return(data);
}
```

APPENDIX B

PROGRAM IN C IMPLEMENTING FUZZY LOGIC CONTROL WITH NON SUM NORMAL MEMBERSHIP FUNCTIONS IN THE PIC16F877

```
// code for fuzzy logic and sensors
// This sends the new values to motors to control their speeds.

#include "C:\PICC\robot\r2.h"
#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <2401.c>

//Global Variables
int select_wheel,sflag=0;

#SEPARATE void write_float_eeprom(long int ,float);
#SEPARATE float read_float_eeprom(long int );
#SEPARATE void write_float_exteeprom(BYTE,long int);
#SEPARATE void write_int_exteeprom(BYTE,int);

#SEPARATE void w_slow(float);
#SEPARATE void w_medium(float);
#SEPARATE void w_fast(float);
#SEPARATE void initialarray();
#SEPARATE float min(float,float);
#SEPARATE float max(float,float);
#SEPARATE float fuzzy();
#SEPARATE float equation_up(float ,float ,float );
#SEPARATE float equation_down(float ,float ,float );
#SEPARATE float defuzzification();
#SEPARATE void et();
#SEPARATE void ex(float,int);
#SEPARATE void rulebase(int,float);

#SEPARATE int ultrapulse(int);
#SEPARATE void getvalues(float,float);
#SEPARATE long int conv_pulse(float,short int);

void main()
{
  float r1=0,r2=0;
  int control;
```

```
long int pulse,pulse2;
int repeat;
BYTE address;

setup_adc_ports(NO_ANALOGS);
setup_adc(ADC_OFF);
setup_psp(PSP_DISABLED);
setup_spi(FALSE);
setup_counters(RTCC_INTERNAL,RTCC_DIV_1);
setup_timer_2(T2_DIV_BY_1,0,1);

output_a(1);
while(true)
{
  output_a(control);
  r1=ultrapulse(5);    // distance in front
  if(r1<10)
  {
    output_high(PIN_D4);    //switches ON the transistors
    control=1;        //STOP the robot
    address=0x08;
    write_int_exteeprom(address,control);
    output_low(PIN_D4);     //switches OFF the transistors
  }
  else
  {
    r1=ultrapulse(1);    //Rfront distances
    r2=ultrapulse(0);    //Rback  distances

    getvalues(r1,r2);    //edistance,etheta are calculated

    select_wheel=0;
    r1=fuzzy();          //change in WR is calc
    select_wheel=1;
    r2=fuzzy();          //change in WL is calc

    r1=r1+4;             //actual WR in radians
    r2=r2+4;             //actual WL in radians

    output_high(PIN_D4);    //switches ON the transistors
    pulse=  conv_pulse(r1,0);      //for rmotor
    pulse2= conv_pulse(r2,1);    //for lmtor
    address=0x00;
    write_float_exteeprom(address,pulse);
    address=0x04;
    write_float_exteeprom(address,pulse2);
```

```
        control=5;
        address=0x08;
        write_int_exteeprom(address,control);
        output_low(PIN_D4);     //switches OFF the transistors

      }

      output_high(PIN_D5); //connected to RB0 of second PIC
      output_low(PIN_D5);

   }        //end of while

}          //end of main


// INTERRUPT FOR SENSORS

#int_ccp1
void isr()
{

   sflag=1;
}

//CALCULATES THE ERRORS IN ANGLE AND DISTANCE OF THE ROBOT
WITH SENSOR INPUTS

#SEPARATE void getvalues(float r1,float r2)
{

   //edistance=(r1+r2)/2;      //r1=rfront
   edistance=4-sqrt((r1+r2)/2); //4 is ref distance
   etheta=0-(asin((r1-r2)/6))/4;
   //etheta=etheta/2;            // r2=rback
   if(edistance>4)
      edistance=4;
   if(edistance<-4)
      edistance=-4;
   if(etheta>0.3)
      etheta=0.3;
   if(etheta<-0.3)
      etheta=-0.3;
}

//CONVERTS FUZZY OUTPUT(FLOAT) TO PULSE (LONG INT)
```

```
#SEPARATE long int conv_pulse(float r,short int f)
{
  long int k;
    if(f==0)
    k=-50*r+1150;     //for Rmotor
  else
    k=50*r+1350;      // for Lmotor
  return(k);
}

#SEPARATE int ultrapulse(int sensor_no)
{

  long int i=0;
  int distance;
  ldiv_t f;

  switch(sensor_no)
  {
    case 0:
    output_high(PIN_C0);              // RIGHT BACK SENSOR
    delay_us(6);                    //apprx 10us delay is givem
    output_low(PIN_C0);
    break;

    case 1:
    output_high(PIN_C1);              // RIGHT FRONT SENSOR
    delay_us(6);                    //apprx 10us delay is givem
    output_low(PIN_C1);
    break;
    case 5:
    output_high(PIN_C5);              // FRONT SENSOR
    delay_us(6);                    //apprx 10us delay is givem
    output_low(PIN_C5);
    break;

    case 4:
    output_high(PIN_C4);              // BACK SENSOR
    delay_us(6);                    //apprx 10us delay is givem
    output_low(PIN_C4);
    break;
  }

  setup_ccp1(CCP_CAPTURE_FE);           // set capture on falling edge
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_1); // set prescaler of tiemr1 to 8
  set_timer1(0);                // re initialize timer to zero
```

```
  enable_interrupts(INT_CCP1);   // Setup interrupt on falling edge
  enable_interrupts(GLOBAL);
  while(i<10000 && sflag==0)
  {
    i=i+1;
  }

  if(i>6000)
  {
    distance=i;
    return(distance);
  }
  else
  {
    sflag=0;            //reinitialize flag to zero
    f= ldiv(CCP_1,150);
    distance=f.quot;

    delay_ms(10);              // probably not needed
    return(distance);
  }
}

#SEPARATE float fuzzy()
{
  int flag,p,X1_addr,Y1_addr,Y1_final_addr;
  float mfex,mfet,centroid1;
  initialarray();
  et();
  centroid1=defuzzification();
}

#SEPARATE void et()
{
  float mfet;
  int flag;
  float const ip2[5]={ -0.3,-0.15,0.0 ,0.15,0.3};

  if (etheta <=ip2[2])
  {
    mfet=equation_down(ip2[0],ip2[2],etheta);      //eqn 1
    flag=1;
    ex( mfet,flag);
  }
  if(ip2[1] <= etheta && etheta <= ip2[2])
  {
```

```
    mfet=equation_up(ip2[1],ip2[2],etheta);     //eqn 2
    flag=2;
    ex( mfet,flag);
  }
  if(ip2[2]<=etheta && etheta<=ip2[3])
  {
    mfet=equation_down(ip2[2],ip2[3],etheta);     //eqn 3
    flag=2;
    ex( mfet,flag);
  }
  if(etheta >=ip2[2])
  {
    mfet=equation_up(ip2[2],ip2[4],etheta);       //eqn 4
    flag=3;
    ex( mfet,flag);
  }
}

#SEPARATE void ex(float mfet,int flag)
{
  float mfex,cpt=-1;
  int rule;
  float const ip1[5]={ -4.0,-2.0,0.0 ,2.0,4.0};
  if (edistance <=ip1[2])         //ex is N
  {
    mfex=equation_down(ip1[0],ip1[2],edistance);     //eqn A
    cpt=min(mfex,mfet);
    if(cpt >0)
    {
      if(flag==1)             //et is N
      {
        rule=1;
        rulebase(rule,cpt);
      }
      if(flag==2 )       //et is Z
      {
        rule=2;
        rulebase(rule,cpt);
      }
      if(flag==3)          //et is P
      {
        rule=3;
        rulebase(rule,cpt);
      }
    }
  }
```

```
if(ip1[1] <= edistance && edistance <= ip1[2])  //ex is Z
{
  mfex=equation_up(ip1[1],ip1[2],edistance);     //eqn B
  cpt=min(mfex,mfet);
  if(cpt >0)
  {
    if(flag==1)             //et is N
    {
      rule=4;
      rulebase(rule,cpt);
    }
    if(flag==2 )       //et is Z
    {
      rule=5;
      rulebase(rule,cpt);
    }
    if(flag==3)          //et is P
    {
      rule=6;
      rulebase(rule,cpt);
    }
  }
}

if(ip1[2]<=edistance && edistance<=ip1[3])   //ex is Z
{
  mfex=equation_down(ip1[2],ip1[3],edistance);     //eqn C
  cpt=min(mfex,mfet);
  if(cpt >0)
  {
    if(flag==1)             //et is N
    {
      rule=4;
      rulebase(rule,cpt);
    }
    if(flag==2 )       //et is Z
    {
      rule=5;
      rulebase(rule,cpt);
    }
    if(flag==3)          //et is P
    {
      rule=6;
      rulebase(rule,cpt);
    }
```

```
    }
  }

  if(edistance >=ip1[2])          //ex is P
  {
    mfex=equation_up(ip1[2],ip1[4],edistance);       //eqn D
    cpt=min(mfex,mfet);
    if(cpt >0)
    {
      if(flag==1)             //et is N
      {
        rule=7;
        rulebase(rule,cpt);
      }
      if(flag==2 )       //et is Z
      {
        rule=8;
        rulebase(rule,cpt);
      }
      if(flag==3)          //et is P
      {
        rule=9;
        rulebase(rule,cpt);
      }
    }
  }

}

#SEPARATE void rulebase(int rule,float cpt)
{
  switch(rule)
  {
    case 1:  if(select_wheel==0)
    {
      w_slow(cpt);
    }
    else
      w_fast(cpt);
    break;

    case 2:  if(select_wheel==0)
    {
      w_slow(cpt);
    }
    else
```

```
   w_medium(cpt);
break;

case 3:  if(select_wheel==0)
{
   w_slow(cpt);
}
else
   w_slow(cpt);
break;

case 4:  if(select_wheel==0)
{
   w_slow(cpt);
}
else
   w_medium(cpt);
break;

case 5:  if(select_wheel==0)
{
   w_fast(cpt);
}
else
   w_fast(cpt);
break;

case 6:  if(select_wheel==0)
{
   w_medium(cpt);
}
else
   w_slow(cpt);
break;

case 7:  if(select_wheel==0)
{
   w_slow(cpt);
}
else
   w_slow(cpt);
break;

case 8:  if(select_wheel==0)
{
   w_medium(cpt);
```

```
    }
    else
      w_slow(cpt);
    break;
    case 9:  if(select_wheel==0)
    {
      w_fast(cpt);
    }
    else
      w_slow(cpt);
    break;
  }
}

#SEPARATE float defuzzification()
{
  float sum,area,X1,Y1,Y1final,centroid1;
  int p,X1_addr,Y1_addr,Y1_final_addr;
  area=0;
  p=0;

  while(p<=20)
  {
    Y1_final_addr=p*4+168;
    Y1final=read_float_eeprom(Y1_final_addr);
    area=(Y1final*0.3)+area;
    p=p+1;
  }

  p=0;
  sum=0;
  while(p<=20)
  {
    X1_addr=p*4;
    Y1_final_addr=p*4+168;
    Y1final=read_float_eeprom(Y1_final_addr);
    X1=read_float_eeprom(X1_addr);
    sum=(Y1final*X1*0.3)+sum;
    p=p+1;
  }
  centroid1=sum/area ;
  return(centroid1);
}

//////////////////////////////////////////////////////////////////
//// INITIALIZING HTE ARRAY TO ZERO ////////////////////////////////
```

```
#SEPARATE void initialarray()
{
  int X1_addr,Y1_addr,Y1_final_addr;
  int p;
  float X1,Y1,Y1final;
  float const op1[5]={-3.0,-1.5,0.0,1.5,3.0};

  for(p=0;p<=20;p++)   //i<= (3-(-3))/stepsize;
  {
    X1=p*0.3+op1[0];
    Y1=0;
    Y1final=0;
    X1_addr=p*4;
    Y1_addr=p*4+84;
    Y1_final_addr=p*4+168;
    write_float_eeprom(X1_addr,X1);
    write_float_eeprom(Y1_addr,Y1);
    write_float_eeprom(Y1_final_addr,Y1final);
  }
}

////////////////////////////////////////////
///        w_slow                      ////
#SEPARATE void w_slow(float cpt)
{
  int X1_addr,Y1_addr,Y1_final_addr;
  int p=0;       //wr is slow
  float X1,Y1,Y1final;
  float const op1[5]={-3.0,-1.5,0.0,1.5,3.0};

  X1_addr=p*4;
  X1=read_float_eeprom(X1_addr);

  while(equation_down(op1[0],op1[2],X1)>=cpt)
  {
    Y1=cpt;
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }

  while(equation_down(op1[0],op1[2],X1)>=0)
  {
```

```
    Y1=equation_down(op1[0],op1[2],X1);  //eqn a
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }


  for (p=0;p<=20;p++)
  {
    Y1_final_addr=p*4+168;
    Y1_addr=p*4+84;
    Y1final=read_float_eeprom(Y1_final_addr);
    Y1=read_float_eeprom(Y1_addr);
    Y1final=max(Y1final,Y1);
    write_float_eeprom(Y1_final_addr,Y1final);

  }
  cpt=-1;
}

////////////////////////////////////
//      w_medium()        ////
#SEPARATE void w_medium(float cpt)
{
  int p=0;       //wr is medium
  int X1_addr,Y1_addr,Y1_final_addr;
  float cutoff1,cutoff2,X1,Y1,Y1final;
  float const op1[5]={-3.0,-1.5,0.0,1.5,3.0};


  X1_addr=p*4;
  X1=read_float_eeprom(X1_addr);

  while(X1 < op1[1])
  {
    Y1=0;
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }

  cutoff1=cpt*(op1[2]-op1[1])+op1[1];
```

```
  cutoff2=cpt*(op1[2]-op1[3])+op1[3];

  while(X1<=cutoff1)
  {
    Y1=equation_up(op1[1],op1[2],X1);     //eqn b
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }
  while(X1<=cutoff2)
  {
    Y1=cpt;
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }
  while(X1<=op1[3])
  {
    Y1=equation_down(op1[2],op1[3],X1); //eqn c
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }
  for (p=0;p<=20;p++)
  {
    Y1_final_addr=p*4+168;
    Y1_addr=p*4+84;
    Y1final=read_float_eeprom(Y1_final_addr);
    Y1=read_float_eeprom(Y1_addr);
    Y1final=max(Y1final,Y1);
    write_float_eeprom( Y1_final_addr,Y1final);

  }
  cpt=-1;
}

////////////////////////////////////
///          W FAST            /////

#SEPARATE void w_fast(float cpt)
```

```
{
  int p=0;        //wr is fast
  int X1_addr,Y1_addr,Y1_final_addr;
  float X1,Y1,Y1final;
  float const op1[5]={-3.0,-1.5,0.0,1.5,3.0};


  X1_addr=p*4;
  X1=read_float_eeprom(X1_addr);

  while(X1 < op1[2])
  {
    Y1=0;
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }

  while(equation_up(op1[2],op1[4],X1)<=cpt)
  {
    Y1=equation_up(op1[2],op1[4],X1);
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;
    X1_addr=p*4;
    X1=read_float_eeprom(X1_addr);
  }

  while(p<=20)
  {
    Y1=cpt;
    Y1_addr=p*4+84;
    write_float_eeprom(Y1_addr,Y1);
    p=p+1;

  }

  for (p=0;p<=20;p++)
  {
    Y1_final_addr=p*4+168;
    Y1_addr=p*4+84;
    Y1final=read_float_eeprom(Y1_final_addr);
    Y1=read_float_eeprom(Y1_addr);
    Y1final=max(Y1final,Y1);
```

```
      write_float_eeprom(Y1_final_addr, Y1final);

   }
   cpt=-1;
}

// writing float data to external eeprom
#SEPARATE void write_float_exteeprom(BYTE address,long int pulse)
{
   int i;
   long int a;
   a=pulse;
   for(i=0;i<2;i++)
   {
      WRITE_EXT_EEPROM(address+i,*(&a +i));

   }

}

// writing int data to external eeprom
#SEPARATE void write_int_exteeprom(BYTE address,int control)
{
   int a;
   a=control;

   WRITE_EXT_EEPROM(address,*(&a));
}


// writing to eeprom
#SEPARATE void write_float_eeprom(long int address ,float data)
{
   int p;
   for( p=0;p<4;p++)
   {
      write_eeprom(address+p,*(&data +p));
   }
}


// reading from eeprom
#SEPARATE float read_float_eeprom(long int address)
{
   int p;
   float data;
```

```
  for( p=0;p<4;p++)
  {
    *(&data+p)=read_eeprom(address+p);
  }
  return(data);
}

#SEPARATE float min(float r, float g)      // minimum function
{
  if (r>g)
    return(g);
  else
    return(r);
}


#SEPARATE float max(float r, float g)      // maximum function
{
  if (r>g)
    return(r);
  else
    return(g);
}

#SEPARATE float equation_down(float lowlimit,float uplimit,float value)
{
  float g;
  g=((value-lowlimit)/(lowlimit-uplimit))+1;
  return(g);
}

#SEPARATE float equation_up(float lowlimit,float uplimit,float value)
{
  float g;
  g=(value-lowlimit)/(uplimit-lowlimit);
  return(g);
}
```

APPENDIX C

PROGRAM IN C IMPLEMENTING FUZZY LOGIC CONTROL WITH A LOOK UP TABLE IN THE PIC16F877

```
fuzzy code using sum normal but with lookuptable
// code for fuzzy logic and sensors
// This sends the new values to motors to control thier speeds.
// using a look uptable with ranges
// rwheel 1400-1500
// lwheel 1700-1600
//neutral point being 1550
#include <16F877.h>
#device ICD=TRUE

//#device adc=8
#device *=16 ICD=TRUE ADC=10            // This allows auto variables over location
0xFF
#use delay(clock=4000000)
#fuses NOWDT,XT, NOPUT, NOPROTECT, BROWNOUT, LVP, NOCPD, NOWRT,
NODEBUG

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <2401.c>

//BYTE const RWHEEL[100]={
//long int const RWHEEL[9]={1420,1430,1440.1450,1460,1470,1480,1490};
//long int const LWHEEL[9]={1620,1630,1640.1650,1660,1670,1680,1690};
int sflag=0;
long int rwheel,lwheel;


// FUNCTIONS FOR ACCESSING INTERNAL AND EXTERNAL EEPROMS

#SEPARATE void write_float_eeprom(long int ,float);
#SEPARATE float read_float_eeprom(long int );
#SEPARATE void write_float_exteeprom(BYTE,long int);
#SEPARATE void write_int_exteeprom(BYTE,int);


// FUNCTIONS FOR MANIPILATION OF DATA
```

```
#SEPARATE float min(float,float);
#SEPARATE float max(float,float);
#SEPARATE float getdistance(float,float);
#SEPARATE float getangle(float,float);


//FUNCTION FOR ACTIVATING ULTRASONIC SENSORS

#SEPARATE int ultrapulse(int);

//FUNCTIONS USED IN SUMNORMAL FUZZIFICATION
#SEPARATE void Pexwheel(float ,float );
#SEPARATE void Nexwheel(float ,float );



void main()
{
  float r1=0,r2=0,edistance,etheta;
  int control,selectwheel;
  long int pulse,pulse2;
  BYTE address;

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_1);
  setup_timer_2(T2_DIV_BY_1,0,1);

  //pulse=RWHEEL[1];
  //pulse2=RWHEEL[2];

  output_a(1);
  while(true)
  {
    output_a(control);

    r1=ultrapulse(5);    // distance in front
    if(r1<4)
    {
      output_high(PIN_D4);    //switches ON the transistors
      control=1;        //STOP the robot
      address=0x08;
      write_int_exteeprom(address,control);
      output_low(PIN_D4);     //switches OFF the transistors
```

```
  output_high(PIN_D5); //connected to RB0 of second PIC
  output_low(PIN_D5);
}

else
{
  r1=ultrapulse(1);    //Rfront distances
  r2=ultrapulse(0);    //Rback  distances



  edistance=getdistance(r1,r2);     //edistance calculated
  etheta=getangle(r1,r2);        //etheta  calculated



  if(edistance==0 && etheta==0)
  {
    pulse = 1400;      //for rmotor
    pulse2= 1700;    //for lmtor
    address=0x00;
    write_float_exteeprom(address,pulse);
    address=0x04;
    write_float_exteeprom(address,pulse2);
    control=5;
    address=0x08;
    write_int_exteeprom(address,control);
    output_low(PIN_D4);     //switches OFF the transistors
  }


  else
  {

    if (edistance>=0)
      Pexwheel(edistance,etheta);
    else
      Nexwheel(edistance,etheta);

    pulse=rwheel;
    pulse2=lwheel;

    //pulse2=rightwheel(int r1,int r2);

    output_high(PIN_D4);    //switches ON the transistors
```

```
          //pulse=  conv_pulse(r1,0);      //for rmotor
          //pulse2= conv_pulse(r2,1);    //for lmtor
          address=0x00;
          write_float_exteeprom(address,pulse);
          address=0x04;
          write_float_exteeprom(address,pulse2);
          control=5;
          address=0x08;
          write_int_exteeprom(address,control);
          output_low(PIN_D4);     //switches OFF the transistors
        }

      }

      output_high(PIN_D5); //connected to RB0 of second PIC
      output_low(PIN_D5);

   }       //end of while

}          //end of main



// INTERRUPT FOR SENSORS

#int_ccp1
void isr()
{

   sflag=1;
}

//CALCULATES THE ERROR IN DISTANCE OF THE ROBOT WITH SENSOR
INPUTS

#SEPARATE float getdistance(float r1,float r2)
{
   float distance,edistance;

   distance= max(r1,r2)+abs(r1-r2)/2;

   edistance=8-distance; //4 is ref distance//8

   edistance=floor(edistance);
```

```
  if(edistance>4)
    edistance=4;
  if(edistance<-4)
    edistance=-4;
  return(edistance);

}

//CALCULATES THE ERROR IN ANGLE OF THE ROBOT WITH SENSOR INPUTS

#SEPARATE float getangle(float r1,float r2)
{
  float etheta,theta;

  theta=r1-r2;
  etheta=0-theta;

  if(etheta>3)
    etheta=3;
  if(etheta<-3)
    etheta=-3;

  return(etheta);

}



//LOOKUP TABLE
#SEPARATE void Pexwheel(float ex,float et)
{

  if(ex==4 && et==-3)
  {
    rwheel=1400;
    lwheel=1700;
  }
  else if (ex==4 && et==-2)
  {
    rwheel=1400;
    lwheel=1680;
  }
  else if (ex==4 && et==-1)
  {
    rwheel=1400;
    lwheel=1670;
```

```
}
else if (ex==4 && et==0)
{
  rwheel=1400;
  lwheel=1670;
}
else if (ex==4 && et==1)
{
  rwheel=1400;
  lwheel=1660;
}
else if (ex==4 && et==2)
{
  rwheel=1400;
  lwheel=1650;
}
else if (ex==4 && et==3)
{
  rwheel=1400;
  lwheel=1630;
}

// ex is 3

else if(ex==3 && et==-3)
{
  rwheel=1410;
  lwheel=1700;
}
else if (ex==3 && et==-2)
{
  rwheel=1410;
  lwheel=1690;
}
else if (ex==3 && et==-1)
{
  rwheel=1410;
  lwheel=1680;
}
else if (ex==3 && et==0)
{
  rwheel=1400;
  lwheel=1670;
}
else if (ex==3 && et==1)
{
```

```
  rwheel=1400;
  lwheel=1670;
}
else if (ex==3 && et==2)
{
  rwheel=1400;
  lwheel=1660;
}
else if (ex==3 && et==3)
{
  rwheel=1400;
  lwheel=1650;
}

// ex is 2

else if(ex==2 && et==-3)
{
  rwheel=1420;
  lwheel=1700;
}
else if (ex==2 && et==-2)
{
  rwheel=1420;
  lwheel=1690;
}
else if (ex==2 && et==-1)
{
  rwheel=1410;
  lwheel=1690;
}
else if (ex==2 && et==0)
{
  rwheel=1400;
  lwheel=1680;
}
else if (ex==2 && et==1)
{
  rwheel=1400;
  lwheel=1670;
}
else if (ex==2 && et==2)
{
  rwheel=1400;
  lwheel=1660;
}
```

```
else if (ex==2 && et==3)
{
  rwheel=1400;
  lwheel=1660;
}

// ex is 1

else if(ex==1 && et==-3)
{
  rwheel=1430;
  lwheel=1700;
}
else if (ex==1 && et==-2)
{
  rwheel=1420;
  lwheel=1690;
}
else if (ex==1 && et==-1)
{
  rwheel=1410;
  lwheel=1690;
}
else if (ex==1 && et==0)
{
  rwheel=1400;
  lwheel=1690;
}
else if (ex==1 && et==1)
{
  rwheel=1400;
  lwheel=1670;
}
else if (ex==1 && et==2)
{
  rwheel=1400;
  lwheel=1670;
}
else if (ex==1 && et==3)
{
  rwheel=1400;
  lwheel=1660;
}

//ex is 0
```

```
  else if(ex==0 && et==-3)
  {
    rwheel=1430;
    lwheel=1700;
  }
  else if (ex==0 && et==-2)
  {
    rwheel=1430;
    lwheel=1700;
  }
  else if (ex==0 && et==-1)
  {
    rwheel=1410;
    lwheel=1700;
  }
  else if (ex==0 && et==0)
  {
    rwheel=1400;
    lwheel=1700;
  }
  else if (ex==0 && et==1)
  {
    rwheel=1400;
    lwheel=1680;
  }
  else if (ex==0 && et==2)
  {
    rwheel=1400;
    lwheel=1670;
  }
  else if (ex==0 && et==3)
  {
    rwheel=1400;
    lwheel=1670;
  }
}

#SEPARATE void Nexwheel(float ex,float et)
{
  //long int rwheel;

  if(ex==-1 && et==-3)
  {
    rwheel=1440;
    lwheel=1700;
  }
```

```
else if (ex==-1 && et==-2)
{
  rwheel=1430;
  lwheel=1700;
}
else if (ex==-1 && et==-1)
{
  rwheel=1420;
  lwheel=1700;
}
else if (ex==-1 && et==0)
{
  rwheel=1400;
  lwheel=1700;
}
else if (ex==-1 && et==1)
{
  rwheel=1400;
  lwheel=1690;
}
else if (ex==-1 && et==2)
{
  rwheel=1400;
  lwheel=1680;
}
else if (ex==-1 && et==3)
{
  rwheel=1400;
  lwheel=1670;
}

// ex is -2

else if(ex==-2 && et==-3)
{
  rwheel=1440;
  lwheel=1700;
}
else if (ex==-2 && et==-2)
{
  rwheel=1440;
  lwheel=1700;
}
else if (ex==-2 && et==-1)
{
  rwheel=1430;
```

```
  lwheel=1700;
}
else if (ex==-2 && et==0)
{
  rwheel=1420;
  lwheel=1700;
}
else if (ex==-2 && et==1)
{
  rwheel=1420;
  lwheel=1690;
}
else if (ex==-2 && et==2)
{
  rwheel=1410;
  lwheel=1690;
}
else if (ex==-2 && et==3)
{
  rwheel=1400;
  lwheel=1680;
}

//ex is -3

else if(ex==-3 && et==-3)
{
  rwheel=1450;
  lwheel=1700;
}
else if (ex==-3 && et==-2)
{
  rwheel=1440;
  lwheel=1700;
}
else if (ex==-3 && et==-1)
{
  rwheel=1440;
  lwheel=1700;
}
else if (ex==-3 && et==0)
{
  rwheel=1430;
  lwheel=1700;
}
else if (ex==-3 && et==1)
```

```
{
  rwheel=1420;
  lwheel=1690;
}
else if (ex==-3 && et==2)
{
  rwheel=1410;
  lwheel=1690;
}
else if (ex==-3 && et==3)
{
  rwheel=1400;
  lwheel=1690;
}

//ex is -4

else if(ex==-4 && et==-3)
{
  rwheel=1470;
  lwheel=1700;
}
else if (ex==-4 && et==-2)
{
  rwheel=1450;
  lwheel=1700;
}
else if (ex==-4 && et==-1)
{
  rwheel=1440;
  lwheel=1700;
}
else if (ex==-4 && et==0)
{
  rwheel=1430;
  lwheel=1700;
}
else if (ex==-4 && et==1)
{
  rwheel=1430;
  lwheel=1700;
}
else if (ex==-4 && et==2)
{
  rwheel=1420;
  lwheel=1700;
```

```
  }
  else if (ex==-4 && et==3)
  {
    rwheel=1400;
    lwheel=1700;
  }
}

//ROUTINE FOR SELESCTING AND RUNNING THE CORRECT SENSOR
RETURNS DISTANCE(INCHES)

#SEPARATE int ultrapulse(int sensor_no)
{

  long int i=0;
  int distance;
  ldiv_t f;

  switch(sensor_no)
  {
    case 0:
    output_high(PIN_C6);              // RIGHT BACK SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C6);
    break;

    case 1:
    output_high(PIN_C1);              // RIGHT FRONT SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C1);
    break;
    case 5:
    output_high(PIN_C5);              // FRONT SENSOR
    delay_us(6);                      //apprx 10us delay is givem
    output_low(PIN_C5);
    break;

  }

  setup_ccp1(CCP_CAPTURE_FE);         // set capture on falling edge
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_1); // set prescaler of tiemr1 to 8
  set_timer1(0);                  // re initialize timer to zero
  enable_interrupts(INT_CCP1);   // Setup interrupt on falling edge
  enable_interrupts(GLOBAL);
  while(i<10000 && sflag==0)
  {
```

```
    i=i+1;
  }

  if(i>6000)
  {
    distance=i;
    return(distance);
  }
  else
  {
    sflag=0;            //reinitialize flag to zero
    f= ldiv(CCP_1,150);
    distance=f.quot;

    delay_ms(10);             // probably not needed
    return(distance);
  }
}

//CALCULATES MINIMUM OF TWO FLOATING NUMBERS

#SEPARATE float min(float r, float g)      // minimum function
{
  if (r>g)
    return(g);
  else
    return(r);
}

//CALCULATES MAXIMUM OF TWO FLOATING NUMBERS
#SEPARATE float max(float r, float g)      // maximum function
{
  if (r>g)
    return(r);
  else
    return(g);
}

// EXTERNAL EEPROM (SHARED BY BOTH PICS)
// WRITING FLOAT DATA TO EXTERNAL EEPROM

#SEPARATE void write_float_exteeprom(BYTE address,long int pulse)
{
  int i;
  long int a;
  a=pulse;
```

```
  for(i=0;i<2;i++)
  {
    WRITE_EXT_EEPROM(address+i,*(&a +i));
  }
}

// WRITING INTIGER DATA TO EXTERNAL EEPROM

#SEPARATE void write_int_exteeprom(BYTE address,int control)
{
  int a;
  a=control;

  WRITE_EXT_EEPROM(address,*(&a));
}

//  INTERNAL EEPROM
// writing FLAOT DATA to eeprom
#SEPARATE void write_float_eeprom(long int address ,float data)
{
  int p;
  for( p=0;p<4;p++)
  {
    write_eeprom(address+p,*(&data +p));
  }
}

// reading FLOAT DATA from eeprom
#SEPARATE float read_float_eeprom(long int address)
{
  int p;
  float data;
  for( p=0;p<4;p++)
  {
    *(&data+p)=read_eeprom(address+p);
  }
  return(data);
}
```

APPENDIX D

PROGRAM IN C TO RUN THE TWO SERVO MOTORS WITH SYNCHRONOUSLY GENERATED PWM IN PIC16F877

code for motors (same for both sum normal and non sum normal)

```
// code for motors only
//this receives interrupt from fuzzypic and gets new values for pulse for motors

#include "C:\PICC\robot\r2.h"
#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <2401.c>

int control,repeat;
long int pulse,pulse2,rpulse,lpulse;
BYTE address;


//FUNCTIONS TO RUN THE MOTORS

#SEPARATE void right_turn(long);
#SEPARATE void left_turn(long);
#SEPARATE void straight(void);
#SEPARATE void run(long int ,long int);
#SEPARATE void working(long int );
//FUNCTIONS TO ACCESS EXTERNAL EEPROM

#SEPARATE long int read_float_exteeprom(BYTE);
#SEPARATE long int read_int_exteeprom(BYTE);

//TO STOP THE MOTOR INFINITELY

#SEPARATE void infiniteloop();


// calculations

#SEPARATE float getdistance(float,float);
#SEPARATE float getangle(float,float);
#SEPARATE long int conv_pulse(float,short int);
```

```
void main()
{

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_1);
  setup_timer_2(T2_DIV_BY_1,0,1);

  enable_interrupts(INT_EXT);
  enable_interrupts(GLOBAL);

  control=5;  //initially

  pulse=1530;  //initially 1550 neutral
  pulse2=1570;   //initially 1550 neutral

  //pulse=1450;  //initially 1550 neutral
  //pulse2=1680;   //initially 1550 neutral

  while(true)
  {

    output_a(control);
    rpulse=pulse+20;  // 20 required as offset
    lpulse=pulse2;

    switch(control)
    {
      case 1:                   //stop
      output_low(PIN_D2);    //OFF LMTR
      output_low(PIN_D3);    //OFF RMTR
      disable_interrupts(INT_EXT);     //loops infinitely
      infiniteloop();              //loops infinitely
      break;

      case 2:                   // straight run
      for(repeat=1;repeat<=20;++repeat)
      {
        straight();
      }
      break;

      case 3:                   //rt turn LMTR on
      for(repeat=1;repeat<=250;++repeat)
```

```
        {
          pulse=1700;
          right_turn(pulse);
        }
        for(repeat=1;repeat<=250;++repeat)
        {
          pulse=1700;
          right_turn(pulse);
        }
        break;

        case 4:                    //lt turn RMTr on
        for(repeat=1;repeat<=200;++repeat)
        {
          pulse=900;
          left_turn(pulse);
        }
        break;

        case 5:
        //for(repeat=1;repeat<=50;++repeat)
        //{
          run(rpulse ,lpulse);
        //}
        break;

        case 6:
        for(repeat=1;repeat<=50;++repeat)
        {
          working(lpulse);
        }
        break;

     }   //end of switch
  }       //end of while
}         //end of main



// CODE JUMPS HERE IF AN EXTERNAL INTERRUPT IS RECEIVED

#INT_EXT
void EXT_isr()
{
  output_high(PIN_D4);    //switches ON the transistors
  address=0x00;
```

```
  pulse=read_float_exteeprom(address);

  address=0x04;
  pulse2=read_float_exteeprom(address);

  address=0x08;
  control=read_int_exteeprom(address);
  output_low(PIN_D4);     ////switches OFF the transistors
}

//INFINITE LOOP NEVER COMES OUT CALLED WHEN STOPPING THE
WHEELS

#SEPARATE void infiniteloop()
{
  int i;
  while(true)
  {
    i=i+1;
  }
}

// READS FLAOTING DATA FROM EXTERNAL EEPROM

#SEPARATE long int read_float_exteeprom(BYTE address)
{
  int i;
  long int val;

  for(i=0;i<2;i++)
  {
    *(&val+i)=READ_EXT_EEPROM(address+i);

  }
  return(val);
}

// READS INTIGER FROM EXTERNAL EEPROM

#SEPARATE  int read_int_exteeprom(BYTE address)
{
  int val;

  *(&val)=READ_EXT_EEPROM(address);

  return(val);
```

```
}

//CASE(5) FUZZY INTPUTS ARE TAKEN AND DIFFERNTIAL DRIVE OF 2
WHEELS TAKES PALCE

#SEPARATE void run(long int pulse ,long int pulse2)
{
  long temp,rpulse,lpulse;
  //ldiv_t t,c;
  int j,i,Rrep,Lrep,Drep,Trep,repeat_low;

  Rrep=(pulse-11)/23;
  Lrep=(pulse2-11)/23;
  Drep=Lrep-Rrep;

  temp=2500-pulse2;
  Trep=(temp-11)/23;

  repeat_low=1;

  output_d(0x00);
  output_high(PIN_D2);        // ON LMOTOR
  output_high(PIN_D3);        // ON RMOTOR

  for(i=1;i<=Rrep;++i)
    {
      for(j=1;j<=repeat_low;++j)
      {
      }
    }

  output_low(PIN_D3);         //OFF RMOTOR

  for(i=1;i<=Drep;++i)
    {
      for(j=1;j<=repeat_low;++j)
      {
      }
    }

  output_low(PIN_D2);         //OFF LMOTOR


  for(i=1;i<=Trep;++i)
    {
      for(j=1;j<=repeat_low;++j)
```

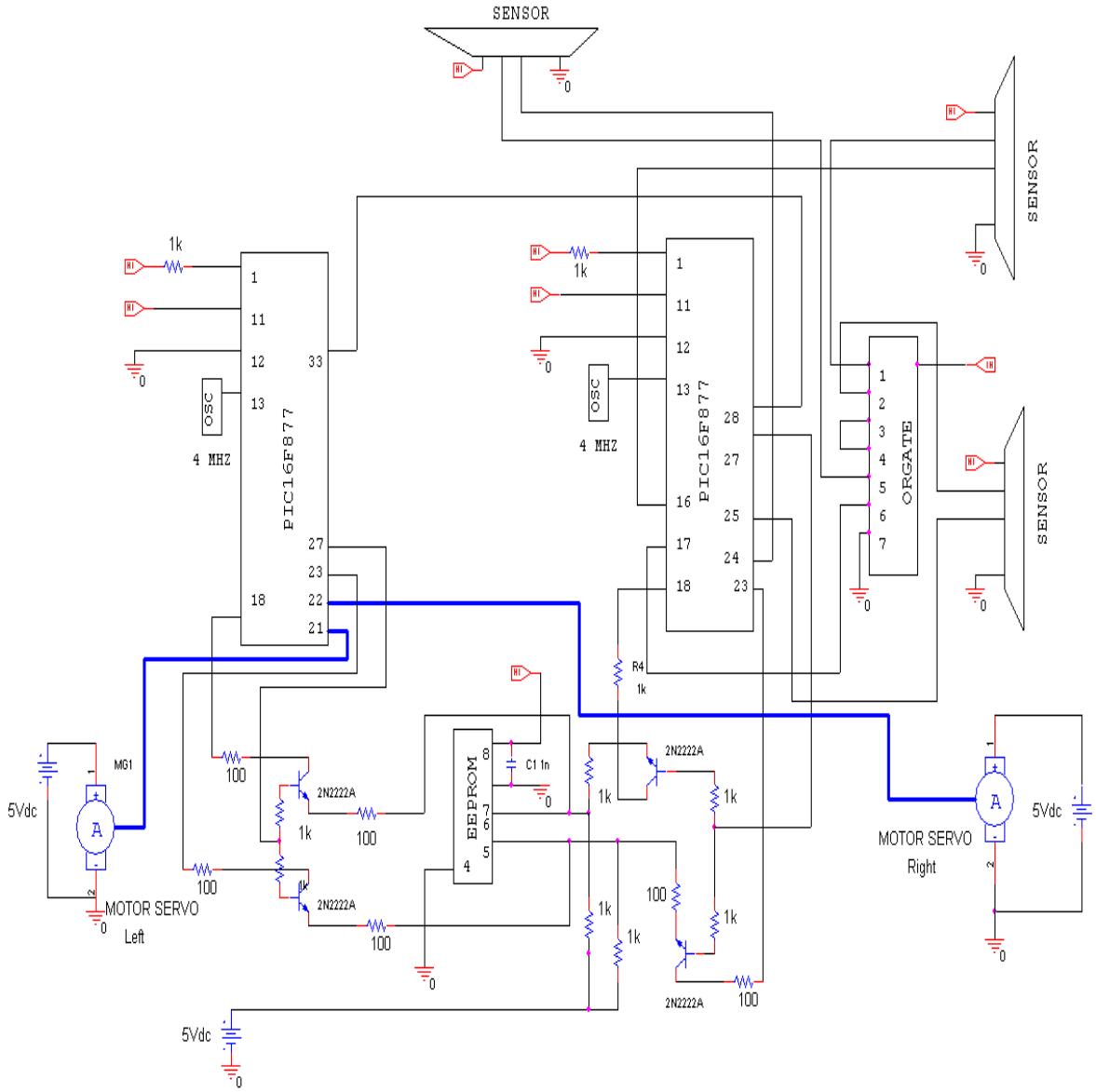
```
        {
        }
      }

  delay_ms(15);

}

//CASE (3) RIGHT TURN

#SEPARATE void right_turn (long pulse)        //turns lmotor i.e RIGHT TURN
{
  long int temp;
  ldiv_t f,t,c;     //***** structure for division in stlib

  f=ldiv(pulse,8);
  output_d(0x00);
  output_high(PIN_D2);            //D2 is connected to lmtr

  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);

  output_low(PIN_D2);
  temp=8880-pulse;
  c=ldiv(temp,1000);
  delay_ms(c.quot);
  t=ldiv(c.rem,4);
  delay_us(t.quot);
  delay_us(t.quot);
  delay_us(t.quot);
  delay_us(t.quot);
  delay_us(t.rem);
}

//CASE (4) LEFT TURN

#SEPARATE void left_turn (long pulse)        //turns rmotor i.e LEFT TURN
{                               //D3 is connected to rmtr
  long  temp;
  ldiv_t f,t,c;     //***** structure for division in stlib
```

```
  f=ldiv(pulse,8);
  output_d(0x00);
  output_high(PIN_D3);

  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);
  delay_us(f.quot);

  output_low(PIN_D3);
  temp=8880-pulse;
  c=ldiv(temp,1000);
  delay_ms(c.quot);
  t=ldiv(c.rem,4);
  delay_us(t.quot);
  delay_us(t.quot);
  delay_us(t.quot);
  delay_us(t.quot);
  delay_us(t.rem);
}

//CASE (2) RUNS THE TWO MOTORS AT CONSTANT SPEED

#SEPARATE void straight(void)        //turns rmotor & lmotor
{
  output_d(0x00);
  output_high(PIN_D2);         // ON LMOTOR
  output_high(PIN_D3);         // ON RMOTOR
  delay_us(900);
  output_low(PIN_D3);          //OFF RMOTOR
  delay_us(800);
  output_low(PIN_D2);          //OFF LMOTOR
  delay_ms(7);
  delay_us(180);
}

#SEPARATE void working(long int lpulse)
{

  int a=0,i=0,y,x,qt;
  long temp,pulse=0;
```

```
  ldiv_t f,t,c;

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_1);
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
  setup_timer_2(T2_DIV_BY_1,0,1);
  setup_ccp1(CCP_CAPTURE_FE);

  pulse=lpulse;
  //pulse=888; //clockwise
  //pulse=1300;//neutral1290-1310
  //pulse=1776;//anti clockwise

  f=ldiv(pulse,8);
  while(TRUE)
  {
    output_d(0x00);
    output_high(PIN_D2);

    delay_us(f.quot);
    delay_us(f.quot);
    delay_us(f.quot);
    delay_us(f.quot);
    delay_us(f.quot);
    delay_us(f.quot);
    delay_us(f.quot);
    delay_us(f.quot);
    output_low(PIN_D2);

    temp=8880-pulse;
    c=ldiv(temp,1000);
    delay_ms(c.quot);
    t=ldiv(c.rem,4);
    delay_us(t.quot);
    delay_us(t.quot);
    delay_us(t.quot);
    delay_us(t.quot);
    delay_us(t.rem);
  }
}
```

APPENDIX E

SCHEMATIC DIAGRAM OF THE ROBOT HARDWARE



APPENDIX F

PROGRAM IN MATLAB IMPLEMENTING FUZZY LOGIC CONTROL WITH SUM NORMAL MEMBERSHIP FUCTIONS

```
%sum normal fuzzy logiccontroller

%function F_speed = snfuzzyrules(IP)
%ex=IP(1);
%et=IP(2);

 ex=2;
 et=2;

selectwheel=0;
speed1=calcmfex(selectwheel,ex,et);

selectwheel=1;
speed2=calcmfex(selectwheel,ex,et);

F_speed=[speed1;speed2];
% end

function answer=calcmfex(selectwheel,ex,et)
%INPUT1
edistance=[-4 0 4];

ex=ex;
et=et;
numerator=0;
denominator=0;
selectwheel=selectwheel;
% calculating membership function of error in distance
if(ex >= edistance(1) & ex <= edistance(2))
   mfex=((ex-edistance(1,1))/(edistance(1,1)-edistance(1,2)))+1;  %ex is N
   rule_no=0;
   A=calcmfet(rule_no,mfex,selectwheel,et);

   numerator= A(1);
   denominaor=A(2);

   mfex=1-mfex;   %ex is Z
   rule_no=3;
   A=calcmfet(rule_no,mfex,selectwheel,et);
```

```
   numerator= numerator+A(1);
   denominaor=denominaor+A(2);

else

   mfex=((ex-edistance(1,2))/(edistance(1,2)-edistance(1,3)))+1;  %ex is Z
   rule_no=3;
   A = calcmfet(rule_no,mfex,selectwheel,et);

   numerator= A(1);
   denominaor=A(2);

   mfex=1-mfex;  %ex is P
   rule_no=6;
   A=calcmfet(rule_no,mfex,selectwheel,et);

   numerator= numerator+A(1);
   denominaor=denominaor+A(2);
end
answer=numerator/denominaor;

   if(answer < -3)
      answer= -3;
   elseif(answer > 3)
      answer= 3;
   end


%*********************************************************
% calculating membership function of error in angle
function B = calcmfet(rule_no,mfex,selectwheel,et)
%INPUT2
% etheta=[-.3 0 0.3];
etheta=[-3 0 3];
et=et;
rule_no1=rule_no;
num1=0;
denum1=0;
selectwheel=selectwheel;

if (et >= etheta(1) & et <= etheta(2))

   mfet=((et-etheta(1,1))/(etheta(1,1)-etheta(1,2)))+1;  %et is N
   W=[mfex mfet];
   rule_no=rule_no1+1;
   w=min(W);
```

```
   B=rulebase(rule_no,w,selectwheel);

   num1=B(1);
   denum1=B(2);

   mfet=1-mfet;   %et is Z
   W=[mfex mfet];
   rule_no=rule_no1+2;
   w=min(W);
   B=rulebase(rule_no,w,selectwheel);

   num1=num1+B(1);
   denum1=denum1+B(2);

else
   mfet=(et-etheta(1,2))/(etheta(1,2)-etheta(1,3))+1;  %et is Z
   W=[mfex mfet];
   rule_no=rule_no1+2;
   w=min(W);
   B=rulebase(rule_no,w,selectwheel);

   num1=num1+B(1);
   denum1=denum1+B(2);

   mfet=1-mfet;   %et is P
   W=[mfex mfet];
   rule_no=rule_no1+3;
   w=min(W);
   B=rulebase(rule_no,w,selectwheel);

   num1=num1+B(1);
   denum1=denum1+B(2);
end
B=[num1 denum1];
% end
%*********************************************

function fout=rulebase(rule_no,w,selectwheel)
selectwheel=selectwheel;
%OUTPUT1
changeW=[-3 0 3];
%OUTPUT2
% changeW=[-3 0 3];
num=0;
denum=0;
%centroid= (br(3*c+br) +bl(3c-bl))/3(br+bl);
```

```
%for Small i.e right triangle at left end centroid is
%[bl =-3; c =-3; br= 0]
%[changeW(1);changeW(1);changeW(2)]
bl=0;
c=-3;
br=3;
Ncentroid =(((br*(3*c+br)) +(bl*(3*c-bl)))/(3*(br+bl)));

% for medium  isoscles trangle
%[bl =-3; c =0; br= +3]
%[changeW(1);changeW(2);changeW(3)]
bl=3;
c=0;
br=3;
Zcentroid = (((br*(3*c+br)) +(bl*(3*c-bl)))/(3*(br+bl)));

% for positve right triangle
%[bl =0; c =3; br= 3]
%[changeW(2);changeW(2);changeW(3)]
bl=3;
c=3;
br=0;
Pcentroid = (((br*(3*c+br)) +(bl*(3*c-bl)))/(3*(br+bl)));

Narea = 0.5*abs(changeW(1)-changeW(2));
Zarea = 0.5*abs(changeW(1)-changeW(3));
Parea = 0.5*abs(changeW(3)-changeW(2));

fout=[];
switch rule_no
   case 1,
      if selectwheel==0          %RWHEEL
         num=w*Narea*Ncentroid;
         denum=w*Narea;
      else                       %LWHEEL
         num=w*Parea*Pcentroid;
         denum=w*Parea;
      end
   case 2,
      if selectwheel==0
         num=w*Zarea*Zcentroid;
         denum=w*Zarea;
      else
         num=w*Parea*Pcentroid;
         denum=w*Parea;
```

```
    end
case 3,
    if selectwheel==0
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    else
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    end
case 4,
    if selectwheel==0
        num=w*Zarea*Zcentroid;
        denum=w*Zarea;
    else
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    end
case 5,
    if selectwheel==0
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    else
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    end
case 6,
    if selectwheel==0
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    else
        num=w*Zarea*Zcentroid;
        denum=w*Zarea;
    end
case 7,
    if selectwheel==0
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    else
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    end
case 8,
    if selectwheel==0
        num=w*Parea*Pcentroid;
        denum=w*Parea;
    else
```

```
            num=w*Zarea*Zcentroid;
            denum=w*Zarea;
        end
    case 9,
        if selectwheel==0
            num=w*Parea*Pcentroid;
            denum=w*Parea;
        else
            num=w*Narea*Ncentroid;
            denum=w*Narea;
        end
end
fout=[num denum];
% end
```

APPENDIX G

PROGRAM IN MATLAB IMPLEMENTING FUZZY LOGIC CONTROL WITH NON SUM NORMAL MEMBERSHIP FUNCTIONS

```
%For both wheels only with variable ranges TS2

function [centroid]=  fmodel(IP)

Input1=[-4 4];
ip1n=[-8 -4 0];
ip1z=[-2 0 2];
ip1p=[0 4 8];

Input2=[-0.3 0.3];
ip2n=[-0.6 -0.3 0];
ip2z=[-0.15 0 0.15];
ip2p=[0 0.3 0.6];

Output1=[-3 3];
op1s=[-5.4 -3 0];
op1m=[-2 0 2];
op1f=[0 3 5.4];

Output2=[-3 3];
op2s=[-5.4 -3 0];
op2m=[-2 0 2];
op2f=[0 3 5.4];

IP1=IP(1);
IP2=IP(2);

i=1;
k=1;
flag=0;
count=0;
stepsize=0.01;
% stepsize=0.4;
cpt=-1;

p=1;
x1(p)=Output1(1);
y1(p)=0;
y1final=y1;
for i= Output1(1):stepsize:Output1(2)
```

```
   p=p+1;
   x1(p)=x1(p-1)+stepsize;
   y1(p)=0;
end
[d,sofy1]=size(y1);

 q=1;
 x2(q)=Output2(1);
 y2(p)=0;
 y2final=y2;
 for i= Output2(1):stepsize:Output2(2)
    q=q+1;
    x2(q)=x2(q-1)+stepsize;
    y2(q)=0;
 end
 [d,sofy2]=size(y2);

% RULE 1 wr is slow & wl is fast
if (IP1 <=ip1n(3) & IP2 <= ip2n(3))
   mfex=((IP1-ip1n(2))/(ip1n(2)-ip1n(3)))+1;     %eqn A
   mfet=((IP2-ip2n(2))/(ip2n(2)-ip2n(3)))+1;     %eqn 1
   cpt=min(mfex,mfet);
end

if(cpt>0)
   p=1;       %wr is slow
   while(((x1(p)-op1s(2))/(op1s(2)-op1s(3)))+1 >=cpt)
      y1(p)=cpt;
      p=p+1;
   end
   while(((x1(p)-op1s(2))/(op1s(2)-op1s(3)))+1>=0)
      y1(p)=((x1(p)-op1s(2))/(op1s(2)-op1s(3)))+1;  %eqn a
      p=p+1;
   end
      for p=1:sofy1
      y1final=max(y1final,y1);
      end

  q=1;       %wl is fast
   while(x2(q) < op2f(1))
      y2(q)=0;
      q=q+1;
   end
   while((x2(q)-op2f(1))/(op2f(2)-op2f(1))<cpt)
      y2(q)=(x2(q)-op2f(1))/(op2f(2)-op2f(1));
      q=q+1;
```

```
    end
    while(q<=sofy2)
        y2(q)=cpt;
        q=q+1;
    end
        for q=1:sofy2
        y2final=max(y2final,y2);
        end
       cpt=-1;
end


% RULE 2.1 wr is medium & wl is fast
if(IP1 <= ip1n(3) & IP2 >=ip2z(1) & IP2 <= ip2z(2))
    mfex=((IP1-ip1n(2))/(ip1n(2)-ip1n(3)))+1;      %eqn A
    mfet=(IP2-ip2z(1))/(ip2z(2)-ip2z(1));     %eqn 2
    cpt=min(mfex,mfet);
 % RULE 2.2
elseif(IP1 <= ip1n(3) & IP2 >= ip2z(2) & IP2 <= ip2z(3))
    mfex=((IP1-ip1n(2))/(ip1n(2)-ip1n(3)))+1;      %eqn A
    mfet=((IP2-ip2z(2))/(ip2z(2)-ip2z(3)))+1;     %eqn 3
    cpt=min(mfex,mfet);
end
if(cpt>0)
    p=1;        %wr is medium
    while(x1(p) < op1m(1))
        y1(p)=0;
        p=p+1;
    end

    cutoff1=cpt*(op1m(2)-op1m(1))+op1m(1);
    cutoff2=(cpt-1)*(op1m(2)-op1m(3))+op1m(2);

    while(x1(p)<=cutoff1)
        y1(p)=(x1(p)-op1m(1))/(op1m(2)-op1m(1));      %eqn b
        p=p+1;
    end
    while(x1(p)<=cutoff2)
        y1(p)=cpt;
        p=p+1;
    end
    while(x1(p)<=op1m(3))
        y1(p)=((x1(p)-op1m(2))/(op1m(2)-op1m(3)))+1;  %eqn c
        p=p+1;
    end
         for p=1:sofy1
```

```
        y1final=max(y1final,y1);
      end

   q=1;       %wl is fast
   while(x2(q) < op2f(1))
      y2(q)=0;
      q=q+1;
   end
   while((x2(q)-op2f(1))/(op2f(2)-op2f(1))<cpt)
      y2(q)=(x2(q)-op2f(1))/(op2f(2)-op2f(1));
      q=q+1;
   end
   while(q<=sofy2)
      y2(q)=cpt;
      q=q+1;
   end
      for q=1:sofy2
      y2final=max(y2final,y2);
      end
   cpt=-1;
end


% RULE 3 wr is fast & wl is fast
if(IP1 <= ip1n(3) &   IP2 >=ip2p(1))
   mfex=((IP1-ip1n(2))/(ip1n(2)-ip1n(3)))+1;      %eqn A
   mfet=(IP2-ip2p(1))/(ip2p(2)-ip2p(1));     %eqn 4
   cpt=min(mfex,mfet);
end

if(cpt>0)
       p=1;       %wr is fast
   while(x1(p) < op1f(1))
      y1(p)=0;
      p=p+1;
   end
   while((x1(p)-op1f(1))/(op1f(2)-op1f(1))<cpt)
      y1(p)=(x1(p)-op1f(1))/(op1f(2)-op1f(1));
      p=p+1;
   end
   while(p<=sofy1)
      y1(p)=cpt;
      p=p+1;
   end
      for p=1:sofy1
      y1final=max(y1final,y1);
```

```
        end


 q=1;        %wl is fast
    while(x2(q) < op2f(1))
        y2(q)=0;
        q=q+1;
    end
    while((x2(q)-op2f(1))/(op2f(2)-op2f(1))<cpt)
        y2(q)=(x2(q)-op2f(1))/(op2f(2)-op2f(1));
        q=q+1;
    end
    while(q<=sofy2)
        y2(q)=cpt;
        q=q+1;
    end
        for q=1:sofy2
        y2final=max(y2final,y2);
        end
    cpt=-1;
end


%rule 4.1  wr is medium & wl is fast
if(ip1z(1) <= IP1 & IP1 <= ip1z(2) & IP2<=ip2n(3))
    mfex=(IP1-ip1z(1))/(ip1z(2)-ip1z(1));      %eqn B
    mfet=((IP2-ip2n(2))/(ip2n(2)-ip2n(3)))+1;     %eqn 1
    cpt=min(mfex,mfet);

%rule 4.2
elseif(ip1z(2) <= IP1 & IP1 <=ip1z(3) & IP2<=0)
   mfex=((IP1-ip1z(2))/(ip1z(2)-ip1z(3)))+1;     %eqn C
    mfet=((IP2-ip2n(2))/(ip2n(2)-ip2n(3)))+1;     %eqn 1
    cpt=min(mfex,mfet);
end

if(cpt>0)
      p=1;        %wr is medium
    while(x1(p) < op1m(1))
        y1(p)=0;
        p=p+1;
    end

    cutoff1=cpt*(op1m(2)-op1m(1))+op1m(1);
    cutoff2=(cpt-1)*(op1m(2)-op1m(3))+op1m(2);
```

```
   while(x1(p)<=cutoff1)
      y1(p)=(x1(p)-op1m(1))/(op1m(2)-op1m(1));      %eqn b
      p=p+1;
   end
   while(x1(p)<=cutoff2)
      y1(p)=cpt;
      p=p+1;
   end
   while(x1(p)<=op1m(3))
      y1(p)=((x1(p)-op1m(2))/(op1m(2)-op1m(3)))+1;  %eqn c
      p=p+1;
   end
       for p=1:sofy1
          y1final=max(y1final,y1);
       end
q=1;       %wl is fast
   while(x2(q) < op2f(1))
      y2(q)=0;
      q=q+1;
   end
   while((x2(q)-op2f(1))/(op2f(2)-op2f(1))<cpt)
      y2(q)=(x2(q)-op2f(1))/(op2f(2)-op2f(1));
      q=q+1;
   end
   while(q<=sofy2)
      y2(q)=cpt;
      q=q+1;
   end
      for q=1:sofy2
      y2final=max(y2final,y2);
      end


   cpt=-1;
end

%rule 5.1   wr is fast & wl is fast
if(ip1z(1)<=IP1 & IP1<=ip1z(2) & ip2z(1)<=IP2 & IP2<=ip2z(2))
    mfex=(IP1-ip1z(1))/(ip1z(2)-ip1z(1));      %eqn B
  mfet=(IP2-ip2z(1))/(ip2z(2)-ip2z(1));     %eqn 2
   cpt=min(mfex,mfet);
%rule 5.2
elseif(ip1z(1)<=IP1 & IP1<=ip1z(2) & ip2z(2) <= IP2 & IP2 <= ip2z(3))
   mfex=(IP1-ip1z(1))/(ip1z(2)-ip1z(1));      %eqn B
   mfet=((IP2-ip2z(2))/(ip2z(2)-ip2z(3)))+1;     %eqn 3
   cpt=min(mfex,mfet);
```

```
%rule 5.3
elseif(ip1z(2)<=IP1 & IP1<=ip1z(3) & ip2z(1)<=IP2 & IP2<=ip2z(2))
   mfex=((IP1-ip1z(2))/(ip1z(2)-ip1z(3)))+1;     %eqn C
   mfet=(IP2-ip2z(1))/(ip2z(2)-ip2z(1));     %eqn 2
   cpt=min(mfex,mfet);
%rule 5.4
elseif(ip1z(2)<=IP1 & IP1<=ip1z(3) & ip2z(2) <= IP2 & IP2 <= ip2z(3))
   mfex=((IP1-ip1z(2))/(ip1z(2)-ip1z(3)))+1;     %eqn C
   mfet=((IP2-ip2z(2))/(ip2z(2)-ip2z(3)))+1;     %eqn 3
   cpt=min(mfex,mfet);
end

if(cpt>0)
   p=1;       %wr is fast
   while(x1(p) < op1f(1))
      y1(p)=0;
      p=p+1;
   end
   while((x1(p)-op1f(1))/(op1f(2)-op1f(1))<cpt)
      y1(p)=(x1(p)-op1f(1))/(op1f(2)-op1f(1));
      p=p+1;
   end
   while(p<=sofy1)
      y1(p)=cpt;
      p=p+1;
   end
      for p=1:sofy1
      y1final=max(y1final,y1);
      end

   q=1;       %wl is fast
   while(x2(q) < op2f(1))
      y2(q)=0;
      q=q+1;
   end
   while((x2(q)-op2f(1))/(op2f(2)-op2f(1))<cpt)
      y2(q)=(x2(q)-op2f(1))/(op2f(2)-op2f(1));
      q=q+1;
   end
   while(q<=sofy2)
      y2(q)=cpt;
      q=q+1;
   end
      for q=1:sofy2
      y2final=max(y2final,y2);
```

```
        end

    cpt=-1;
end


%rule 6.1 wr is fast & wl is medium
if(ip1z(1)<=IP1 & IP1<=ip1z(2) & IP2>=ip2p(1))
     mfex=(IP1-ip1z(1))/(ip1z(2)-ip1z(1));      %eqn B
     mfet=(IP2-ip2p(1))/(ip2p(2)-ip2p(1));     %eqn 4
    cpt=min(mfex,mfet);
%rule 6.2
elseif(ip1z(2)<=IP1 & IP1<=ip1z(3) & IP2>=ip2p(1))
     mfex=((IP1-ip1z(2))/(ip1z(2)-ip1z(3)))+1;     %eqn C
     mfet=(IP2-ip2p(1))/(ip2p(2)-ip2p(1));     %eqn 4
    cpt=min(mfex,mfet);
end

if(cpt>0)

        p=1;        %wr is fast
    while(x1(p) < op1f(1))
        y1(p)=0;
        p=p+1;
    end
    while((x1(p)-op1f(1))/(op1f(2)-op1f(1))<cpt)
        y1(p)=(x1(p)-op1f(1))/(op1f(2)-op1f(1));
        p=p+1;
    end
    while(p<=sofy1)
        y1(p)=cpt;
        p=p+1;
    end
        for p=1:sofy1
        y1final=max(y1final,y1);
        end
    q=1;        %wl is medium
    while(x2(q) < op2m(1))
        y2(q)=0;
        q=q+1;
    end

    cutoff1=cpt*(op2m(2)-op2m(1))+op2m(1);
    cutoff2=(cpt-1)*(op2m(2)-op2m(3))+op2m(2);

    while(x2(q)<=cutoff1)
```

```
      y2(q)=(x2(q)-op2m(1))/(op2m(2)-op2m(1));      %eqn b
      q=q+1;
   end
   while(x2(q)<=cutoff2)
      y2(q)=cpt;
      q=q+1;
   end
   while(x2(q)<=op2m(3))
      y2(q)=((x2(q)-op2m(2))/(op2m(2)-op2m(3)))+1;  %eqn c
      q=q+1;
   end
       for q=1:sofy2
          y2final=max(y2final,y2);
       end

   cpt=-1;

end



%rule 7 wr is fast & wl is fast
if(IP1>=ip1p(1) & IP2<=ip2n(3))
    mfex=(IP1-ip1p(1))/(ip1p(2)-ip1p(1));     %eqn D
    mfet=((IP2-ip2n(2))/(ip2n(2)-ip2n(3)))+1;     %eqn 1
   cpt=min(mfex,mfet);
end

if(cpt>0)
     p=1;       %wr is fast
   while(x1(p) < op1f(1))
      y1(p)=0;
      p=p+1;
   end
   while((x1(p)-op1f(1))/(op1f(2)-op1f(1))<cpt)
      y1(p)=(x1(p)-op1f(1))/(op1f(2)-op1f(1));
      p=p+1;
   end
   while(p<=sofy1)
      y1(p)=cpt;
      p=p+1;
   end
      for p=1:sofy1
      y1final=max(y1final,y1);
      end
q=1;       %wl is fast
```

```
    while(x2(q) < op2f(1))
       y2(q)=0;
       q=q+1;
    end
    while((x2(q)-op2f(1))/(op2f(2)-op2f(1))<cpt)
       y2(q)=(x2(q)-op2f(1))/(op2f(2)-op2f(1));
       q=q+1;
    end
    while(q<=sofy2)
       y2(q)=cpt;
       q=q+1;
    end
       for q=1:sofy2
       y2final=max(y2final,y2);
       end


    cpt=-1;
end

%rule 8.1   wr is fast & wl is medium
if(IP1>=ip1p(1) & ip2z(1)<=IP2 & IP2<=ip2z(2))
   mfex=(IP1-ip1p(1))/(ip1p(2)-ip1p(1));     %eqn D
     mfet=(IP2-ip2z(1))/(ip2z(2)-ip2z(1));     %eqn 2
    cpt=min(mfex,mfet);
%rule 8.2
elseif(IP1>=ip1p(1) & ip2z(2)<=IP2 & IP2<=ip2z(3))
    mfex=(IP1-ip1p(1))/(ip1p(2)-ip1p(1));     %eqn D
   mfet=((IP2-ip2z(2))/(ip2z(2)-ip2z(3)))+1;     %eqn 3
    cpt=min(mfex,mfet);
end
if(cpt>0)
    p=1;        %wr is fast
    while(x1(p) < op1f(1))
       y1(p)=0;
       p=p+1;
    end
    while((x1(p)-op1f(1))/(op1f(2)-op1f(1))<cpt)
       y1(p)=(x1(p)-op1f(1))/(op1f(2)-op1f(1));
       p=p+1;
    end
    while(p<=sofy1)
       y1(p)=cpt;
       p=p+1;
    end
       for p=1:sofy1
```

```
      y1final=max(y1final,y1);
      end
  q=1;       %wl is medium
   while(x2(q) < op2m(1))
      y2(q)=0;
      q=q+1;
   end

   cutoff1=cpt*(op2m(2)-op2m(1))+op2m(1);
   cutoff2=(cpt-1)*(op2m(2)-op2m(3))+op2m(2);

   while(x2(q)<=cutoff1)
      y2(q)=(x2(q)-op2m(1))/(op2m(2)-op2m(1));      %eqn b
      q=q+1;
   end
   while(x2(q)<=cutoff2)
      y2(q)=cpt;
      q=q+1;
   end
   while(x2(q)<=op2m(3))
      y2(q)=((x2(q)-op2m(2))/(op2m(2)-op2m(3)))+1;  %eqn c
      q=q+1;
   end
       for q=1:sofy2
          y2final=max(y2final,y2);
      end

   cpt=-1;
end

%rule 9 wr is fast & wl is slow
if(IP1>=ip1p(1) & IP2>=ip2p(1))
   mfex=(IP1-ip1p(1))/(ip1p(2)-ip1p(1));     %eqn D
   mfet=(IP2-ip2p(1))/(ip2p(2)-ip2p(1));     %eqn 4
   cpt=min(mfex,mfet);
end
if(cpt>0)

   p=1;       %wr is fast
   while(x1(p) < op1f(1))
      y1(p)=0;
      p=p+1;
   end
   while((x1(p)-op1f(1))/(op1f(2)-op1f(1))<cpt)
      y1(p)=(x1(p)-op1f(1))/(op1f(2)-op1f(1));
      p=p+1;
```

```
    end
    while(p<=sofy1)
       y1(p)=cpt;
       p=p+1;
    end

       for p=1:sofy1
       y1final=max(y1final,y1);
       end

    q=1;       %wl is slow
    while(((x2(q)-op2s(2))/(op2s(2)-op2s(3)))+1 >=cpt)
       y2(q)=cpt;
       q=q+1;
    end
    while(((x2(q)-op2s(2))/(op2s(2)-op2s(3)))+1>=0)
       y2(q)=((x2(q)-op2s(2))/(op2s(2)-op2s(3)))+1;  %eqn a
       q=q+1;
    end
       for q=1:sofy2
       y2final=max(y2final,y2);
       end
    cpt=-1;
end

% OUTPUT1 defuzzification
area=0;
p=1;
while(p<=sofy1)
    area=(y1final(p)*stepsize)+area;
    p=p+1;
end
p=1;
sum=0;
while(p<=sofy1)
    sum=(y1final(p)*x1(p)*stepsize)+sum;
    p=p+1;
end
centroid1=sum/area ;

%OUTPUT2 DEFUZZIFICATION
area=0;
q=1;
while(q<=sofy2)
    area=(y2final(q)*stepsize)+area;
    q=q+1;
```

```
end
q=1;
sum=0;
while(q<=sofy2)
   sum=(y2final(q)*x2(q)*stepsize)+sum;
   q=q+1;
end
centroid2=sum/area;
centroid=[centroid1; centroid2];
```