Genetic Tuning of Fuzzy Controller

Conrado F. Ostia, Jr.
Oscar Y. Chuy, Jr.

Abstract

A methodology of applying Genetic Algorithm (GA) to optimize a fuzzy controller in a speed control using the integral of the square of the error (ISE) criterion is presented in this study. Motivated by the claim that fuzzy controller is hard to train, this study is conducted on the stretch of a computer simulation using MATLAB.

A cursory exploration has been done to determine the suitable combination of operators to be used in the optimization proper. Since GA is a random process, several trials were done in every exploration, as well as in the optimization proper.

Results showed that GA was able to tune the fuzzy controller. This derivative-free algorithm returned a lower average ISE than what is reported in Chuy's work, that is, using derivative-based optimization method, by 2.7 percent.

CHAPTER 1

INTRODUCTION

Genetic Algorithms (GAs) have gained an increasing popularity as an optimization tool. In fact, several published researches have already reported successful applications on optimization problems such that of fuzzy controllers. GAs are directed random search optimization routines modeled after nature's evolutionary process [7]. They perform a parallel, stochastic , but guided search to evolve the most fit population. Demonstrating the unique coding capability to represent parameters of membership functions, GAs can be used to tune fuzzy logic controller (FLC), fuzzy controller for short .

Fuzzy controller known for its applicability on controllable systems with complicated math-model, or even without math-model, shows feasibility to application that allows a bigger room for error. As one of its limitations, a trained fuzzy controller is applicable only to a set of inputs and outputs which is part of the universal training space.

This paper presents the results of the optimization of a fuzzy controller using GA. Speed control system is used as the testing set-up of the simulation. Integral of the square of the error (ISE) is the performance criterion used. The convergence rate is measured by the number of *generation* produced during the whole optimization process.

In this chapter, problems, objectives, significance, scope and limitations and the theoretical framework of the study are specified and discussed. Related literatures are reviewed in Chapter 2. Theoretical background on fuzzy controllers and genetic algorithms is discussed thoroughly in Chapter 3. Chapter 4 describes the methodology of the study. Design and Implementation are presented in Chapter 5. Results are discussed in Chapter 6 and the conclusion and recommendations in Chapter 7.

1.1 Statement of the Problem

Recent studies on optimization showed that fuzzy controllers led proportional-integral-derivative (PID) controllers with insignificant magnitude of ISE. Thus, the advantage of fuzzy controllers over PID controllers is inconspicuous. It is, therefore, necessary to try another optimization technique to further enhance the performance of fuzzy controllers.

The main problem of this study is how to optimize a fuzzy controller using GA. Specifically, it is the concern of this research on how to genetically vary the width of membership functions to obtain minimum ISE.

1.2 Objectives of the Study

The main objective of this study is to establish a methodology on how to apply GA in tuning fuzzy controller. This tuning process determines the optimum fuzzy controller parameters which would yield minimum ISE.

It specifically intends to seek solutions on how to apply GA to automatically determine the width of the input membership functions that would result to minimum ISE. Further, it investigates the convergence rate, the lowest ISE ever attained, and the consistency of the optimization process.

1.3 Significance of the Study

The result of this study will establish a sound methodology in optimizing a fuzzy controller using GA. It will also serve as a benchmark of any effort to further verify the validity of the work on an actual set-up. Furthermore, it will serve as a basis of any study regarding GA search and optimization inside and outside fuzzy controller.

1.4 Theoretical Framework

Fuzzy controllers, on one hand, outperformed classical controllers as claimed by many researchers. Chuy [4] verified this in his study. He made a comparative study of classical and pure fuzzy controllers using a speed control system to evaluate its ISE-based performance. Both controllers were optimized by Levenberg-Marquardt Method, a derivative-based optimization. As a result, the fuzzy controller leads to a smaller ISE compared to the PID controller by 2 percent. The lowest ISE ever arrived of the fuzzy controller was about 0.052. However, the former took more iterations than the latter. This means that the fuzzy controller is harder to train than the PID controller using a derivative-based optimization method.

Genetic Algorithm, on the other hand, has been proven as an effective optimization tool. Braunstingl et. al. [3] and Akbarzadeh et. al. [1] showed this in their work. In the study of Braunstingl et. al. [3], they demonstrated that a wall following robot with a fuzzy controller optimized by a GA performed satisfactorily. Akbarzadeh et. al. [1] compared model-based PID, fuzzy-PID, and GA-optimized fuzzy-PID controllers. The third controller, outperformed the other first two controllers.

Since no known study applied GA to tune a pure fuzzy controller, this work proposed a methodology on how to apply this derivative-free optimization technique to tune such controller in a MATLAB environment. In this study, a speed control system whose transfer functions are modeled after Chuy's [4] study, is used as a testing set-up. The fuzzy controller is designed to have a Gaussian input membership function with a variable width, a fixed seven (7) fuzzy rules and a fixed triangle output membership function. A genetic optimization program is developed to automatically tune the controller. As illustrated in Figure 1.1 below, this program will update the controller's parameters which are the width of the input membership functions. It will then simulate the model developed in *Simulink*. The model will get the parameters for simulation and return ISE. This iterative process will go on until the desired minimum ISE is attained.

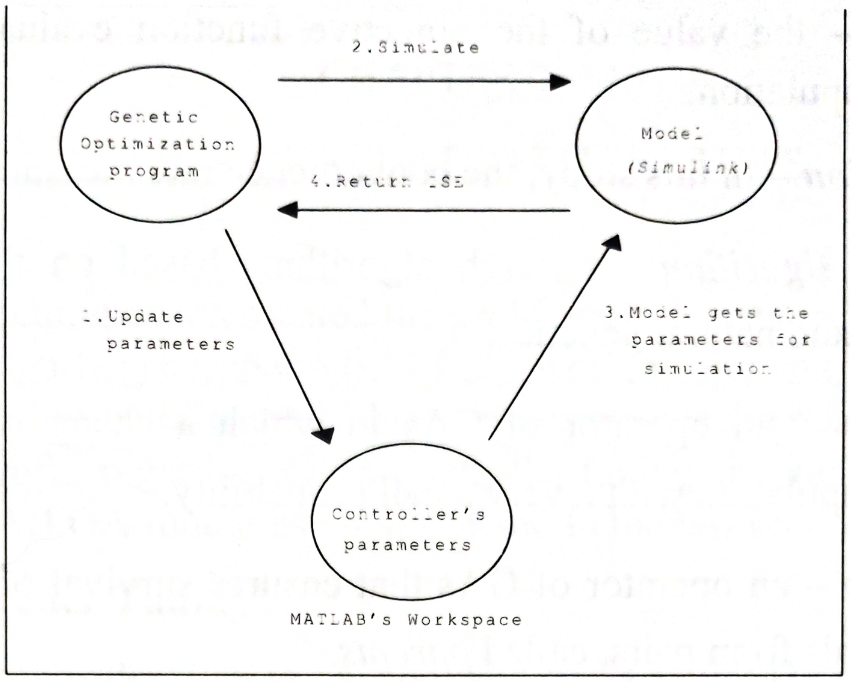


Figure 1. Fuzzy controller tuning diagram

1.5 Scope and Limitations

The study focuses on the application of GA in the optimization of a fuzzy controller used in a speed control system. In order to minimize the parameter set and reduce simulation's processing time, only the input membership function of the controller is varied. The output membership function and fuzzy rules are fixed. The optimization processes will then be evaluated using the ISE performance criterion.

The available function in Genetic Algorithm Optimization Toolbox (GAOT) and Fuzzy Logic Toolbox (FLT) for MATLAB are being used.

The work is limited to computer simulation only. Physical experimentation is beyond the scope of this study.

1.6 Definition of Terms

Chromosome – the binary bit string that represents a parameter (or solution).

Crossover – an operator used in GAs to generate new chromosomes (or offspring). It combines portions of two parents to create two offsprings.

Fitness – the value of the objective function evaluated at a certain member in the population.

Generation – In this study, the convergence rate measurement in GAs.

Genetic Algorithms – search algorithms based on the mechanics of natural selection and natural genetics.

Mutation – an operator of GAs in which a change is made in each member of the population with a very small probability.

Selection – an operator of GAs that ensures survival of the fittest. The selected individuals form pairs, called *parents*.

Singleton – a membership function whose support is a single point with a membership value of one.

Stochastic – In this study, a process that is random in nature.

Support – the support of a fuzzy set which is the set of all points in the set whose membership value is greater than zero.

Tuning – the optimization of the parameters to get minimum error.

CHAPTER 2

REVIEW OF RELATED LITERATURE

The preceding chapter stated the problem of this study. It also presented the objectives, significance, theoretical framework, scope and limitations of the study, and the definition of terms. This chapter reviews the related literatures. It looks over the works regarding fuzzy controller tuning and genetic optimization. Existing MATLAB GA functions are discussed in the last section of this chapter.

2.1 Fuzzy Controller Tuning

Fuzzy controller tuning, as defined by Chuy [4], is the process of determining the parameters of the fuzzy controller such that the desired response or specification is achieved. Moler and Costa [16] illustrated fuzzy controller tuning by changing the parameters in the input membership function to get the desired response.

In his work, Chuy [4] compares the performances of PID and fuzzy controllers both tuned with the Levenberg-Marquardt Method using the ISE criterion. The latter returned an ISE smaller than the former by two (2) percent. The fuzzy controller was tuned to emulate the non-linear input-output relation of a controller by changing the width of the Gaussian input membership function. Chuy [4] made the centers of the input membership function, the triangle output membership function, and the seven (7) fuzzy rules fixed. He used a step signal as an input to the controllers and a direct current (DC) motor as an actuator of the system. A computer simulation was conducted before doing an actual experimentation.

MATLAB has a toolbox function, *anfis,* which stands for adaptive neuro-fuzzy inference system. As described by Gulley and Jang [8], it is an off-line tuning of the parameters associated with the input membership function of the fuzzy controller. Using for example a given input/output data of the equation z = sinc(x)*sinc(y), where x and y are the inputs and z is the output, it constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone, or in combination with a least squares type of method. Figure 2.1 shows the initial and final forms of the input membership function after the training. As shown in the

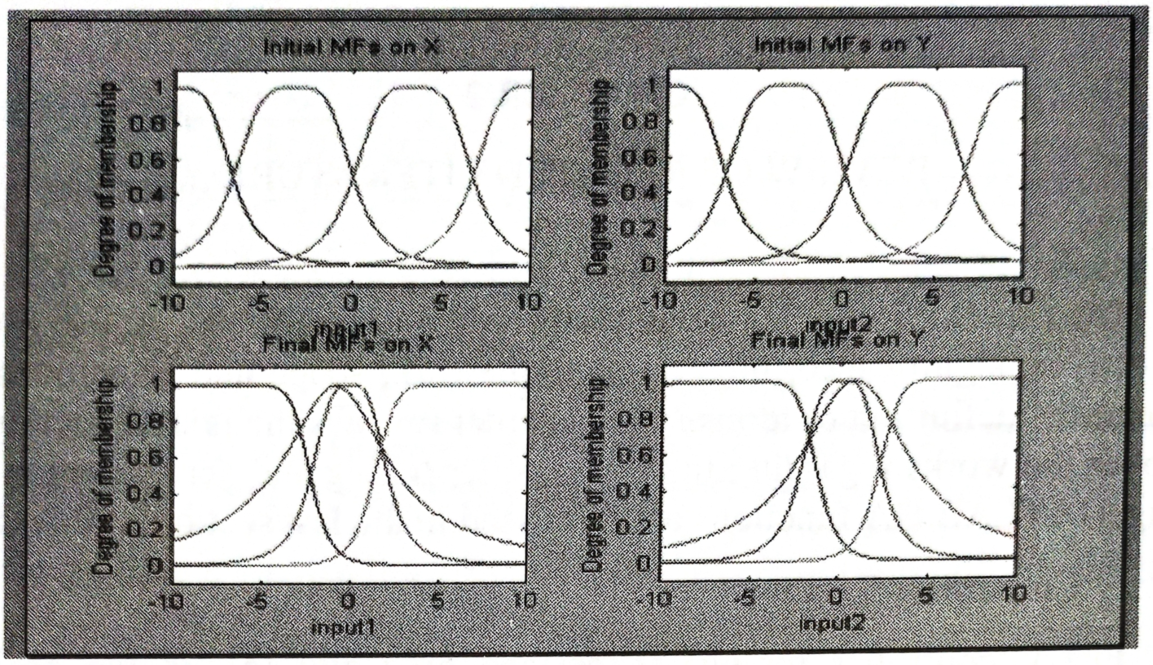


Figure 2.1. Initial and final input membership function optimized by ANFIS.

figure, the centers and the width of the Gaussian input membership function are varied (See the initial and final forms.) to optimum values. Figure 2.2 indicates that the fuzzy system (shown by the ANFIS output graph, top-right) is able to emulate the training data (top-left).

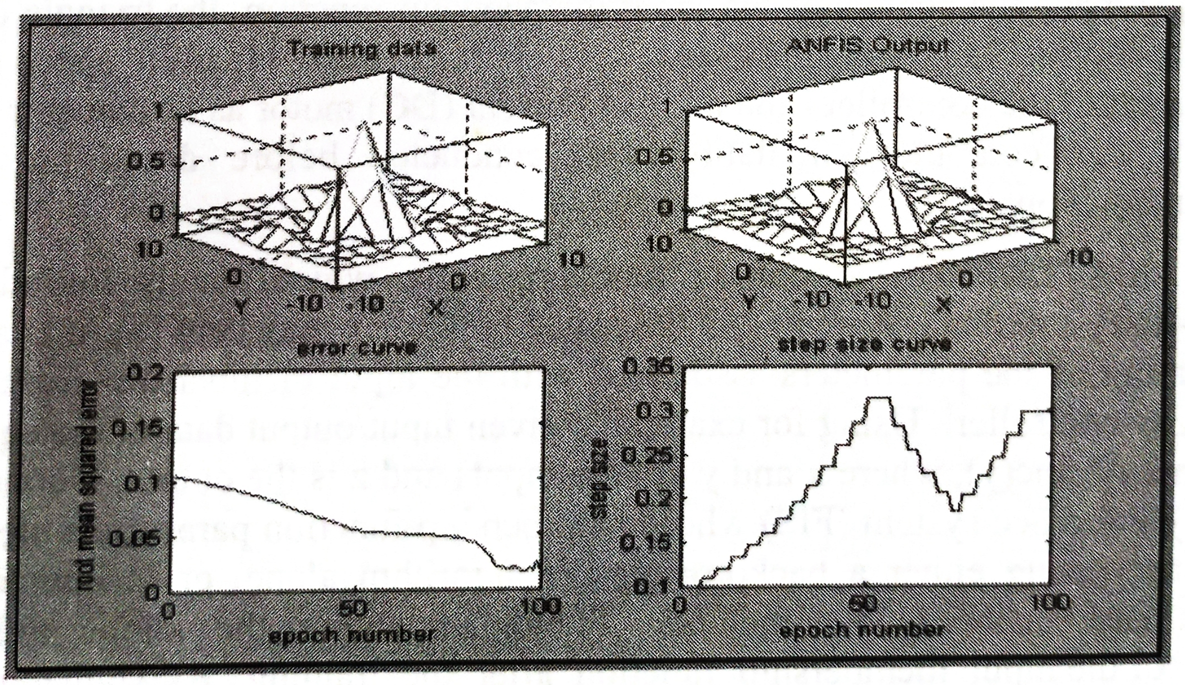


Figure 2.2. ANFIS output

2.2 Genetic Optimization

Braunstingl et. al. [3] applied GA in the optimization of a fuzzy controller used in a wall following robot. In their work, they used thirty-three (33) fuzzy rules. Triangles and trapezoids were chosen to describe the input membership functions and singletons for the output membership functions. They set up the rule base and type of membership functions by hand. However, the shape of the membership functions was left undecided, to be determined by GA in such a way as to reach the highest performance of the robot. It is to move at constant distance to the wall, at high speed and as smoothly as possible.

Akbarzadeh et. al. [1] compared the model-based PID, fuzzy-PID, and GA-optimized fuzzy-PID, using the returned angular velocities of the controllers as the performance criterion. In the optimization of the fuzzy-PID controller, they incorporated the expert knowledge in generating the initial population to make the process converge faster. They minimized, as well, the parameter set by allowing only the parameters associated with the input membership function to vary as to reduce the simulation's processing time. As a result, the GA-optimized fuzzy-PID performed significantly better than both the initial crude fuzzy-PID and model-based PID controllers.

2.3 MATLAB Functions

Houck et. al. [9] developed the Genetic Algorithm Optimization Toolbox (GAOT) for MATLAB. This toolbox contains the genetic optimization function, ***ga***. This function can do a maximization task. Also available in this toolbox are the several operators for different representations which could readily be used during simulated evolution. The following are the operators built for float representation:

Selection Operators

normGeomSelect Normal Geometric Select is a ranking selection function based on the normalized geometric distribution.

roulette Roulette is the traditional selection function with the probability of surviving equal to the fitness of the ith individual divided by the sum of the fitness of all individuals.

tournSelect Tournament Select performs a tournament selection.

Crossover Operators

arithXover Arithmetic crossover takes two parents and performs an interpolation along the line formed by the two parents.

heuristicXover Heuristic crossover takes parents and performs an extrapolation along the line formed by the two parents outward in the direction of the better parent.

simpleXover Simple crossover takes two parents and performs simple single point crossover.

Mutation Operators

boundaryMutation Boundary Mutation changes one of the parameters of the parent and changes it randomly either to its upper or lower bound.

multiNonUnifMutation Multi-non-uniform mutation changes all of the parameters of the parent based on a non-uniform probability distribution. This Gaussian distribution starts wide and narrows to a point distribution as the current generation approaches the maximum generation.

nonUnifMuation Non-uniform mutation changes one of the parameters of the parent based on a non-uniform probability distribution. This Gaussian distribution proceeds like the multiNonUnifMutation.

unifMutation Uniform mutation changes one of the parameters of the parent based on a uniform probability distribution.

CHAPTER 3
THEORETICAL BACKGROUND

This chapter discusses the theoretical background of fuzzy controllers and genetic algorithms (GAs). It starts with the discussion of the theory of fuzzy controllers, followed by fuzzy optimal control. It ends with the thorough discussion on GA.

3.1 Theory on Fuzzy Controllers

Fuzzy controllers have the capability of emulating the human decision-making process. They have shown robustness in task-oriented applications. To understand the strength of a fuzzy controller, the basic concepts and fundamentals will be discussed.

3.1.1 Classical Sets and Logic Theory

For example, a product quality, x, is divided into only two categories: 'qualified' and 'not qualified'. A classical subset A of quality variable space for the qualified product which has crisp or precise boundaries is described as $A = \{x|\ x_{min} \le x \le x_{max}\}$. The product quality then can be simply identified by the following bivalent membership (or characteristic indicator) function [11]:

$$m_A(x) = \begin{cases} 1 & x_{min} \le x \le x_{max} \\ 0 & \text{otherwise,} \end{cases} \qquad 3.1$$

where $m_A(x)$ denotes that x either is in A or is not. Since m_A maps x into two numbers $\{0, 1\}$; this set corresponds to two-valued logic. Some events or fact can be described by multi-valued logic, for instance, products sometimes are required to be divided into a number of grades based on given criteria. But imprecise information and its approximate reasoning, such as fuzzy linguistic statements existing in many real-world problems, are deal by fuzzy logic.

3.1.2 Fuzzy Sets and Fuzzy Membership

When we say 'the product quality is good', we are clearly making a typical fuzzy language statement describing the quality. To represent a fuzzy statement under a numerical framework, the fuzzy membership function, $m_A(x)$ can be defined and introduced to measure the 'degree of goodness' for any given product and is generally written as

$$m_A(x) = \text{degree } (x \subset A). \qquad 3.2$$

The fuzzy subset A is then denoted as follows:

$$A = \{(x, m_A(x)) | \ x \in X\}, \qquad 3.3$$

where X is referred to as the universe of discourse (or simply universe) or the space of objects [10].

Figure 3.1 shows an example of a membership function on a continuous universe.

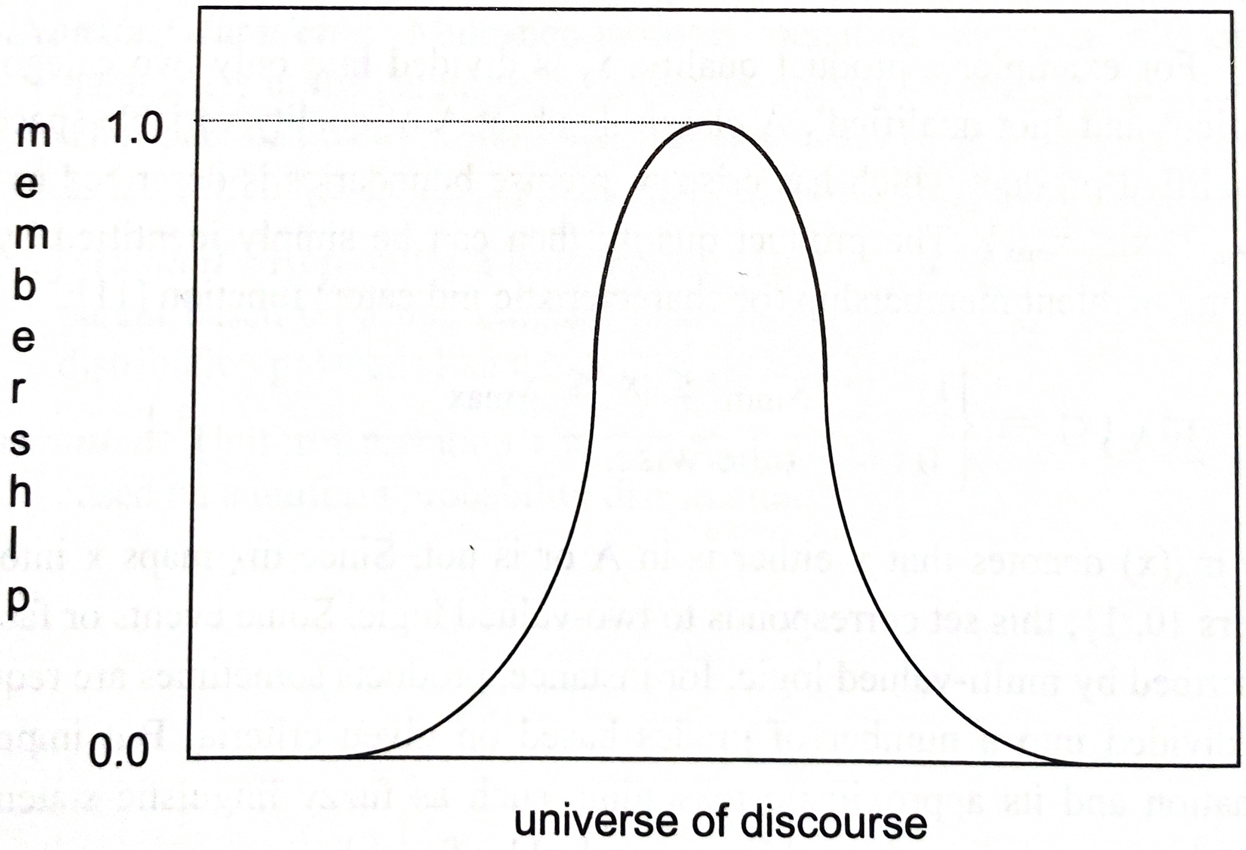


Figure 3.1. An example of membership function on a continuous universe.

3.1.2.1 Basic Operations on Fuzzy Sets

Let A and B be the fuzzy subsets of X, then the operations on fuzzy sets are introduced as follows [2]:

Equality

A and B are said to be equal if (if and only if) their membership functions are equal, i.e.

$m_A(x) = m_B(x), x \in X$ (x is included in X). 3.4

Inclusion

A is included in B, i.e.

$A \subset B$, iff $m_A(x) \leq m_B(x), x \in X$. 3.5

Proper subset

A is a proper subset of B, i.e.

$A \subset B$; iff $m_A(x) \leq m_B(x), x \subset X$ and $m_A(x) < m_B(x)$, for at least one x, $x \in X$. 3.6

Complementation

The complement of A, i.e. A^c, is defined as

$A^c = \{1 - m_A(x)\}, x \in X$. 3.7

Intersection

The intersection of A and B, i.e. $A \cap B$, is defined as

$m_{A \cap B}(x) = \min(m_A(x), m_B(x)), x \in X$. 3.8

Union

The union of A and B, i.e. $A \cup B$, is defined as

$m_{A \cup B}(x) = \max(m_A(x), m_B(x)), x \in X$. 3.9

3.1.2.2 Fuzzy Relations

The Cartesian product can be defined as

$$A \times B = \{(x,y) \mid x \in A, y \in B\}, \qquad 3.10$$

where A and B are subsets of the universal sets X_1 and X_2, respectively. A fuzzy relation A x B denoted by R(x, y), or R is defined as

$$R = \{((x,y), m_R(x,y)) \mid (x,y) \in A \times B, m_R(x,y) \in [0,1]\}, \qquad 3.11$$

where $m_R(x,y)$is a function in two variables called membership function [2]. It indicates the degree to which x is in relation with y.

The following are the common linguistic relations that can be described by appropriate fuzzy relations:

- X is much greater than y
- X is close to y
- X is relevant to y
- X and y are almost equal X
- X and y are very far.

Fuzzy relations in different product spaces can be combined through a composition operation. The best known is the max-min composition.

Max-min Composition

Consider the following relations:

$$R_1(x,y) = \{((x,y), m_{R_1}(x,y)) \mid x \in A, y \in B, (x,y) \in A \times B\}, \qquad 3.12$$

$$R_2(y,z) = \{((y,z), m_{R_2}(y,z)) \mid y \in B, z \in C, (y,z) \in B \times C\}, \qquad 3.13$$

The max-min composition denoted R₁□R₂ with membership function $m_{R_2 \circ R_1}$ is defined by

$$R_1 \square R_2 = \{((x,z), \max_y(\min(m_{R_1}(x,y), m_{R_2}(y,z))))|(y,z) \square AxC\}$$

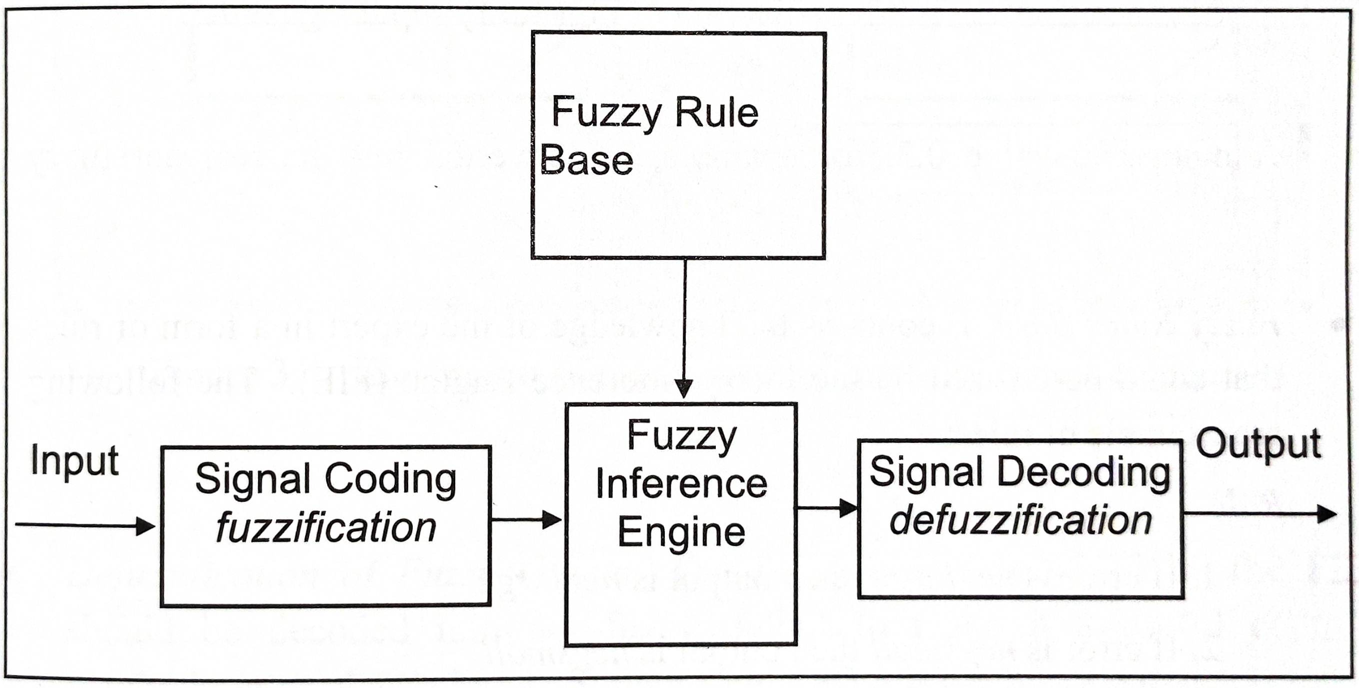
3.14

where $\max_y$ means maximum with respect to y.

3.1.3 Architecture of Fuzzy Logic Controls

Figure 3.2, shows the general structure of a typical fuzzy controller [11]. Components shown are briefly described next.

Figure 3.2. General structure of a fuzzy controller.


Fuzzy Rule Base
Input
Signal Coding *fuzzification*
Fuzzy Inference Engine
Signal Decoding *defuzzification*
Output

□□ *Input Signal Coding*: Inputs of any FLC should be converted into fuzzy language so that the Fuzzy Inference Engine can interpret the information. Figure 3.3 illustrates this.

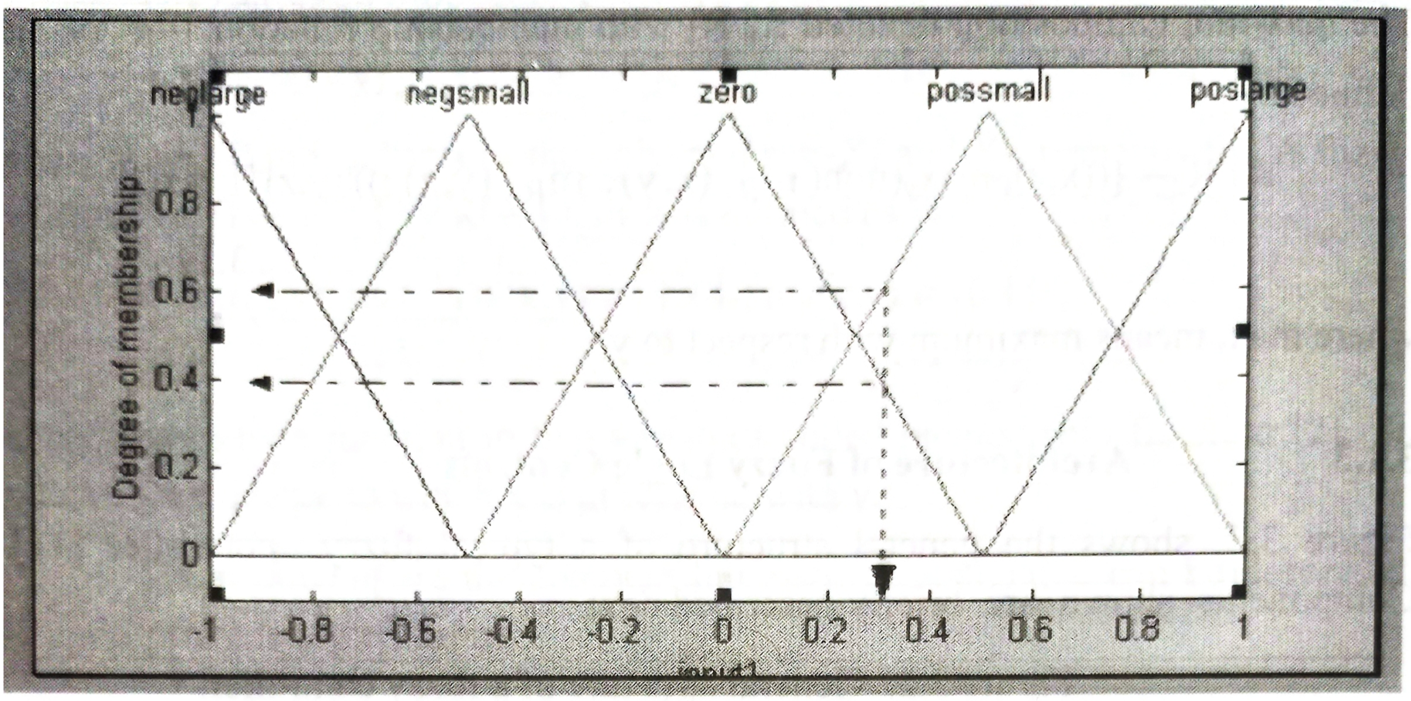


Figure 3.3. An example of an input membership function with an illustration of fuzzification of non-fuzzy input signal 0.3.

A numerical value 0.3, for example, is converted into its relevant fuzzy points.

- *Fuzzy Rules Base*: It contains the knowledge of the expert in a form of rules that could be utilized by the Fuzzy Inference Engine (FIE). The following are example of rules:

  Rule

  1. If error is *neglarge* then output is *neglarge*
  2. If error is *negsmall* then output is *negsmall*
  3. If error is *zero* then output is *zero*
  4. If error is *possmall* then output is *possmall*
  5. If error is *poslarge* then output is *poslarge*

  where neglarge is an abbreviation of negative large, negsmall of negative small, and so on.

Fuzzy Inference Engine: The task of fuzzy inference is to make decision. However, it can only provide a fuzzy output. Figure 3.4 illustrates this. This is suppose Rules 3 and 4 above fired and the input and output membership functions used are triangle. The graph in the bottom-right is the aggregated fuzzy output of our FIE.

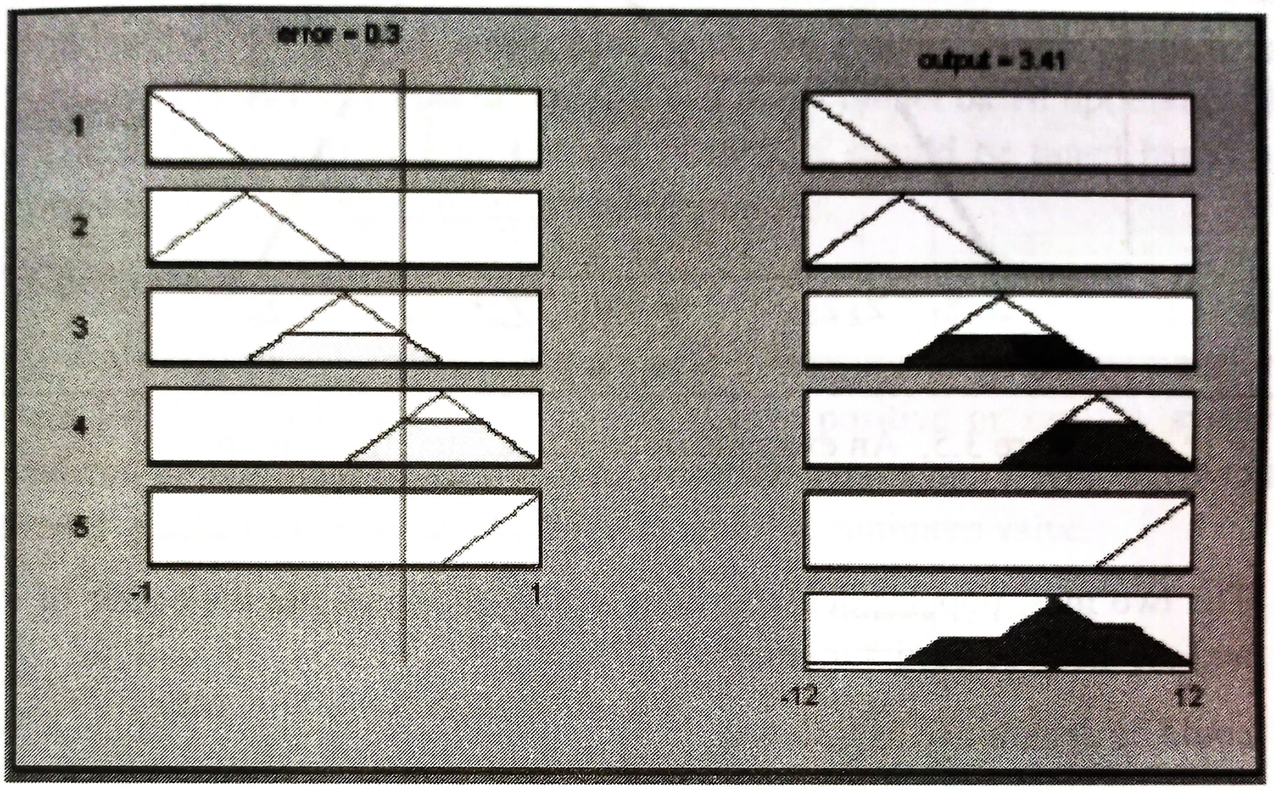


Figure 3.4. Illustration of mechanism of fuzzy inference engine.

- *Defuzzification of Fuzzy Output*: The fuzzy output provided by the FIE should be decoded into non-fuzzy output to make it a useful control manipulator or plant's input. Suppose Figure 3.5 is the fuzzy output.

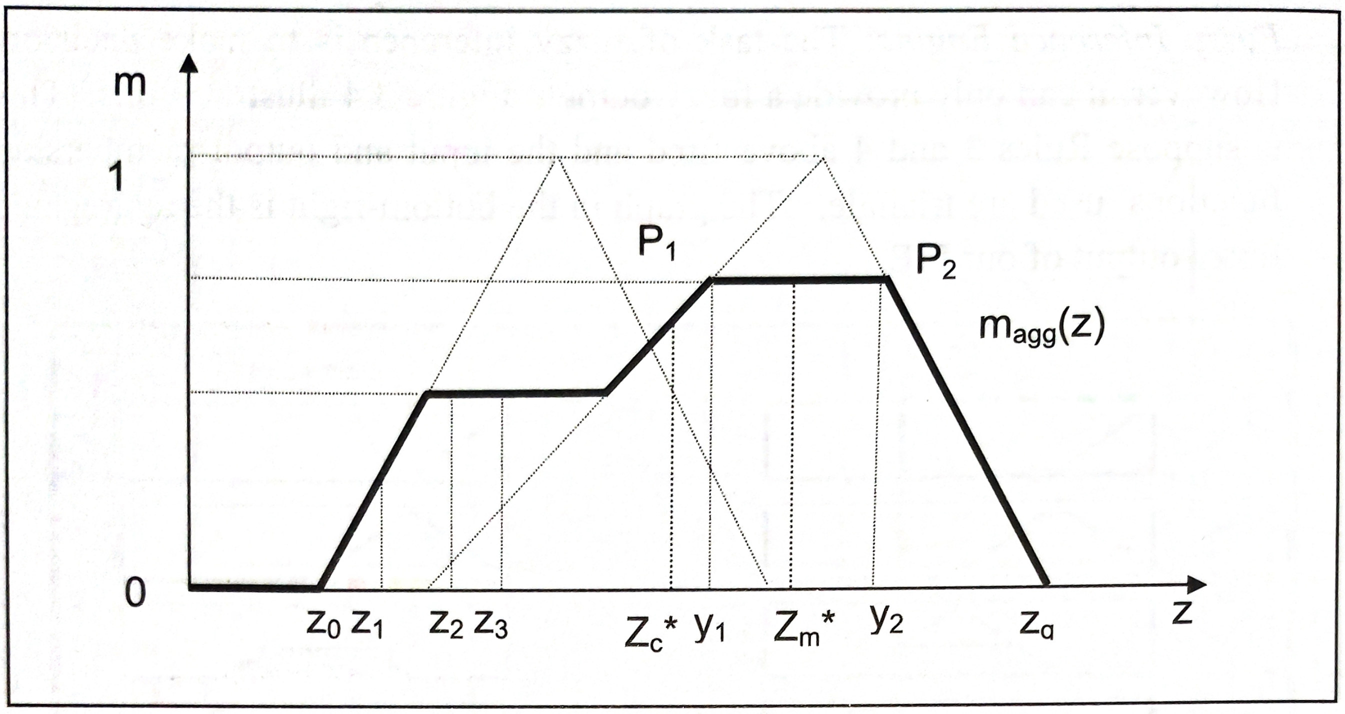


Figure 3.5. An example of an aggregated fuzzy output.

The two most popular defuzzification methods which are the center of area and the mean of maximum methods [2]

could be determined as follows:

Center of Area Method

The crisp value z_c* is computed as follows:

$$z_c{}^* = \frac{\sum_{k=1}^{q-1} z_k m_{agg}(z_k)}{\sum_{k=1}^{q-1} m_{agg}(z_k)} \qquad 3.15$$

where $m_{agg}(z_k)$ is the membership function of the aggregated control rules.

Mean of maximum method

Let the projection of the flat segment P_1P_2 with maximum height on z axis be the interval $[y_1, y_2]$ (See Fig. 3.5.). Then z_m* is determined by the formula

$$z_m^* = \frac{y_1 + y_2}{2} \qquad 3.16$$

3.2 Fuzzy Optimal Control

An optimal control system is a system that usually yields an optimal trajectory of control variables which optimize the given dynamic performance criteria subject to desired constraints, such as system dynamics and other constraints [11]. A fuzzy optimal control is a fuzzy model based optimal control. For a fuzzy system to give an optimal control, it should be tuned based on a performance index using an optimization technique.

3.2.1 Performance Indices

A performance index is a quantitative measure of the performance of a system [6]. It must be a number that is always positive or zero. A system is considered an optimum control system when the parameters are adjusted so that the index reaches an extremum value, commonly a minimum value.

The following are the known performance indices:

- ISE (Integral of the square of the error)
- IAE (Integral of the absolute magnitude of the error)
- ITAE (Integral of time multiplied by absolute error)
- ITSE (Integral of time multiplied by the squared error)

Among the four enumerated indices, ISE is the most common criterion in optimizing a control system. It is minimized by minimizing Areas 1, 2 and 3, as shown in Figure 3.6.

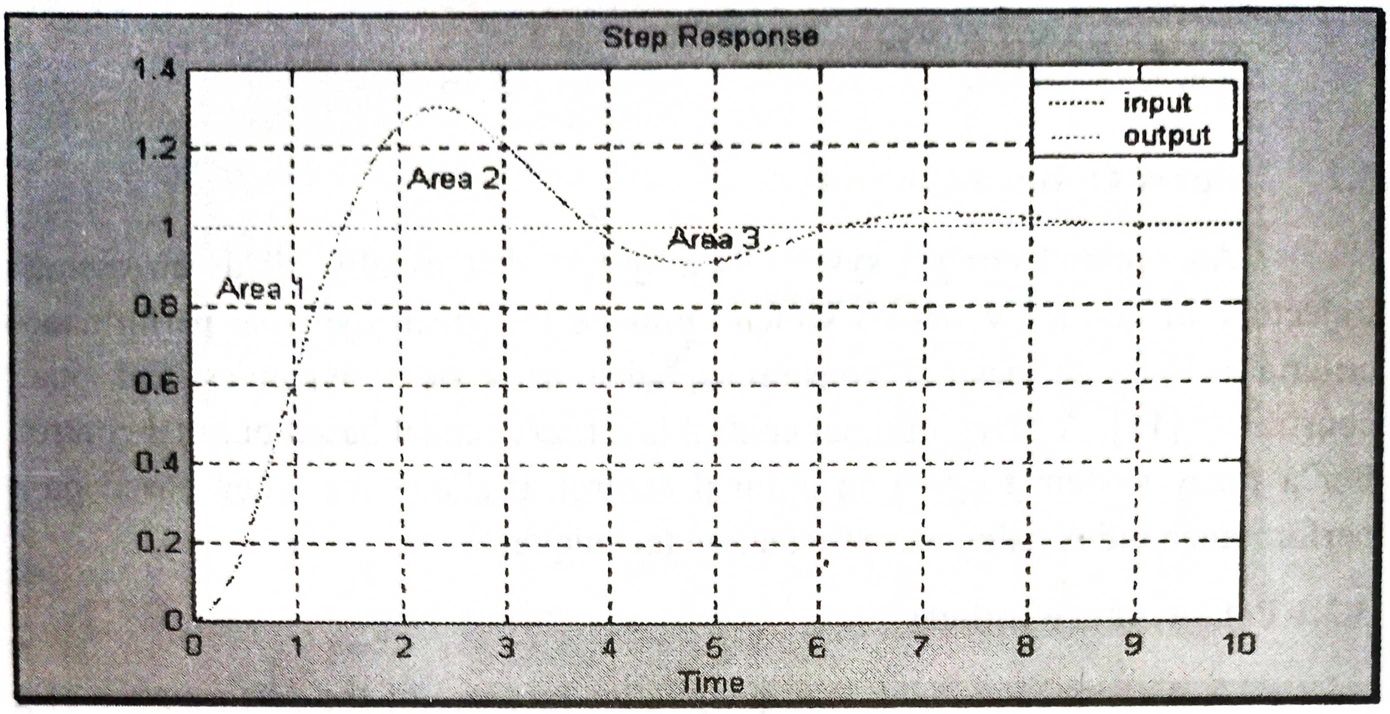


Figure 3.6. The graph of ideal step response (input) and actual step response.

3.2.2 Optimization Techniques

In optimizing the above performance indices there is a correspondingly appropriate technique or techniques for each of them which would give a fast or at least guarantee convergence. There are two categories of optimization techniques, namely; the derivative-based and derivative-free optimizations [10]. The former is capable of determining search directions according to an objective function's derivative information, while the latter is stochastic, which means that it uses random number generators in determining subsequent search directions.

There are two methods which form the foundation of many derivative-based algorithms, namely, the steepest descent and the Newton-Raphson. Though the former is simple, the latter proves to be more robust. However, due to heavy computational requirements of the Hessian Matrix, concepts like adaptive step length and Levenberg-Marquardt concepts were integrated into Newton-Raphson Method. There is also a method that approximates the Hessian Matrix which is called Quasi-Newton Method. For very large linear problems, Conjugate Gradient Methods claim to be significantly less expensive than the Quasi-Newton Method. On the other hand, for nonlinear least-squares problems, the Gauss-

Newton Method provides a better solution. As expected, to make this method more attractive, the step size was introduced creating a new method called Hartley's Method (or dumped Gauss-Newton Method). But, the integration of Levenberg-Marquardt concepts into the Gauss-Newton Method, which was eventually called the Levenberg-Marquardt Methods, proves to be more robust than Hartley's Method.

The most popular derivative-free optimization methods are the following: genetic algorithms (GA), simulated annealing (SA), random search method, and downhill simplex search. They share these common characteristics; derivative freeness, intuitive guidelines, flexibility, randomness and analytic capacity. The random search method and downhill simplex search concepts and implementations are simple but GAs and SA are regarded better for all problems all the time [10]. Due to the parallel-search procedures of GA, it is expected to converge faster and less likely to get trapped in local minima than SA. GAs provide a stochastic optimization method where if they get stuck at a local optimum, they try to simultaneously find other parts of the search space and jump out of the local optimum to a global one [17].

3.3 Genetic Algorithms (GAs)

GAs are derivative-free optimization methods based on the mechanics of natural selection and natural genetics [7]. They adapted the natural survival of the fittest principle with the integration of structured yet randomized information exchange. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

3.3.1 Differences of GAs from Traditional Methods

GAs are different from more traditional methods in the following ways [7]:

- GAs work with a coding of the parameter set, not the parameter themselves.
- GAs search from a population of points, not a single point.

- GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.
- GAs use probabilistic transition rules, not deterministic rules.

Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet. To demonstrate, consider the function $f(x) = x^2$. Suppose, we are to maximize this function on the integer interval, (0, 31). In more traditional methods we would be searching the parameter x directly over the specified space interval that would give the highest objective function value. In GAs, however, the first step of our optimization process is to code the parameter x as a finite-length string, say, 0000 (binary code) for x equals to zero.

In many traditional search procedures, we move arduously from a single point in the decision space to the next point using a transition rule. This point-to-point method is dangerous because it is a perfect prescription for locating false peaks in multi-modal (many peaks) search spaces. While in GAs, we work from a rich database of points simultaneously (a population of strings), climbing many peaks in parallel; thus, the probability of finding a false peak is reduced.

GAs are blind. They do not need auxiliary information such as derivatives of the objective function and complete tabulation of parameters which the gradient-based and combinatorial optimizations need respectively. All they need is the payoff values associated with individual strings to perform an effective search for better and better structures.

Unlike most traditional methods which are deterministic, GAs use probabilistic transition rules to guide their search toward regions of the search space with likely improvement.

3.3.2 Characteristics of GAs

Aside from being free from dependence on functional derivatives, GAs are popular due to the following characteristics [10]:

- GAs are parallel-search procedures that can be implemented on parallel-processing machines for massively speeding up their operations.

- GAs are applicable to both continuous and discrete (combinatorial) optimization problems.
- GAs are stochastic and less likely to get trapped in local minima, which inevitably are present in any practical optimization application.
- GAs' flexibility facilitates both structure and parameter identification in complex models such as neural networks and fuzzy inference systems.

3.3.3 Components of GAs

The major components of GAs are namely: encoding schemes, fitness evaluation, selection, crossover, and mutation [10].

Encoding schemes. These transform points in parameter space into bit string representation. For instance, a point (11,6,9) in a three-dimensional parameter space can be represented as a concatenated binary string:

$$\underset{11}{\underline{1\,0\,1\,1}}\ \underset{6}{\underline{0\,1\,1\,0}}\ \underset{9}{\underline{1\,0\,0\,1}}$$

in which each coordinate value is encoded as a gene composed of four binary bits using binary coding. Other encoding schemes, such as the Gray coding, can also be used and when necessary, arrangements can be made for encoding negative, floating-point, or discrete-valued numbers. Genetic operators, such as crossover and mutation, can and should be designed along with the encoding scheme used for a specific application.

Fitness evaluation. After creating a generation, the fitness value f_i of each member in the population is calculated. It is the value of the objective function evaluated at the ith member. Usually positive fitness values are needed, so scaling and/or translation may be necessary if the objective function is not strictly positive. For minimization problem, the negative of the objective function values are taken since GAs can only do maximization. Minimizing f(x) is equivalent to maximizing -f(x).

Selection. The selection operation determines which parents participate in producing offspring for the next generation, and it is analogous to survival of the fittest in natural selection. Members are selected for mating with a selection probability usually equal to $f_i / \sum_{k=1}^{k=n} f_k$ where *n* is the population size. In this way, members with above-average fitness values will reproduce and replace members with below-average fitness values.

Crossover. Crossover is applied to selected pairs of parents with a probability equal to a given *crossover rate.* It is hoped then that the new generated chromosomes retain the good features of the previous generation. The two common crossover operators are the one-point and the two-point; these are illustrated in Figure 3.7. One can define the n-point crossover similarly.

```
      crossover

        point

  1 0 0 |1 1 1 1 0        1 0 0 | 1 0 0 1 0

        |            ⇒         |

  1 0 1 |1 0 0 1 0        1 0 1 | 1 1 1 1 0

                   (a)


  1| 0 0 1 1|1 1 0        1|0 1   1 0|1 1 0

   |        |        ⇒    |          |

  1|0 1 1 0 |0 1 0        1| 0 0 1 1 |0 1 0

                   (b)
```

Figure 3.7. Crossover operators: (a) one-point crossover; (b) two-point crossover.

Mutation. If gene mixing, i.e., crossover, cannot produce a satisfactory solution, mutation is applied by flipping a bit, i.e., form 1 to 0 or vice versa, with a probability equal to a very low given *mutation rate.* It can prevent the population from converging and stagnating at any local optima. The mutation rate is usually kept low so that good chromosomes obtained from crossover are not lost. Because if it is high (above 0.1), GA performance will approach that of a primitive random search.

3.3.4 A Simple GA

Based on the concepts mentioned above, a simple GA could be done as follows:

Step 1. Initialize a population with randomly generated individuals and evaluate the fitness value of each individual.

Step 2. Select two members from the population with probabilities proportional to their fitness values.

Step 3. Apply crossover with a probability equal to the crossover rate.

Step 4. Apply mutation with a probability equal to the mutation rate.

Step 5. Repeat Steps 2 to 5 until enough members are generated to form the new generation.

Step 6. Repeat Steps 2 to 6 until a stopping criterion is met.

CHAPTER 4

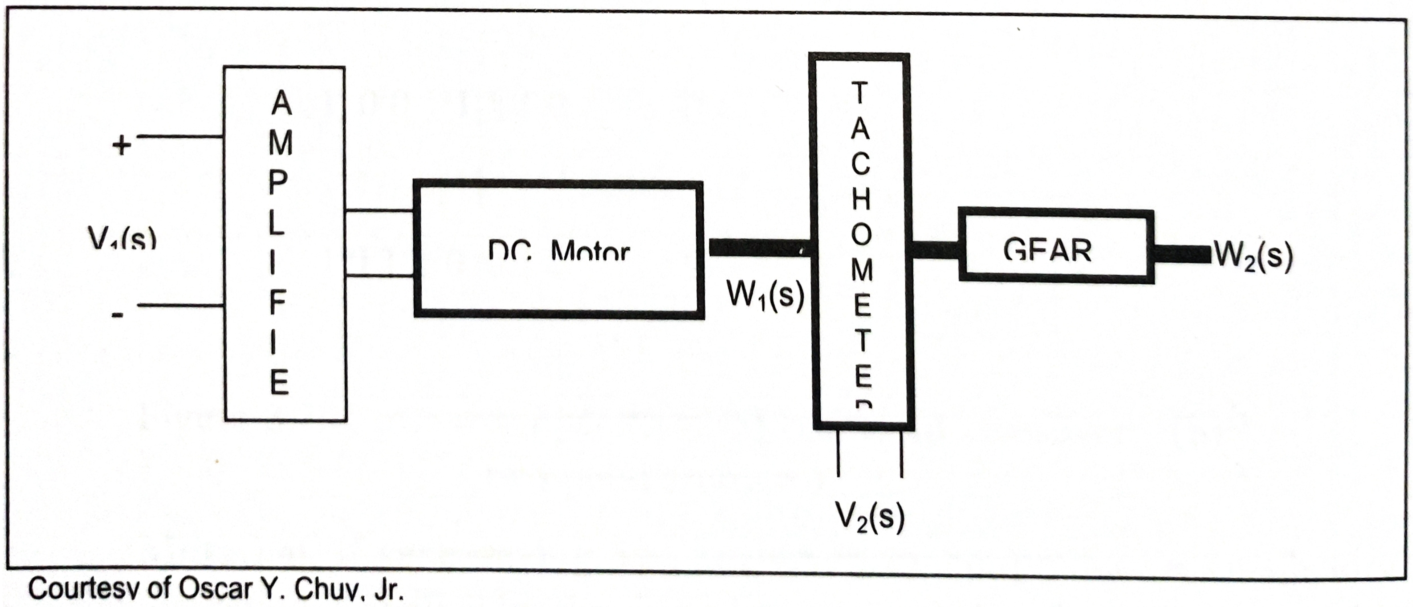
METHODOLOGY

This chapter presents the methodology of the study. First, it discusses the system design, followed by fuzzy controller design. Section 4.3 discusses the optimization program coding succeeded by the operator combination search methodology. Methods of the simulated genetic optimization are presented in Section 4.5. The last section of this chapter describes the evaluation process of the controller.

4.1 System Design

A basic speed control system will be composed of an amplifier, a DC motor, a tachometer, and a gear as shown in Figure 4.1. This system will ensure non-linearity in which fuzzy controllers are advantageously applicable. It will be the testing set-up of this study.

Using the *Simulink*, a simulation diagram will be developed. Due to the unavailability of control system analyzer, transfer functions of the plant will be modeled after Chuy's [4] work.



Courtesy of Oscar Y. Chuy, Jr.

Figure 4.1. Schematic diagram of a speed control system.

4.2 Fuzzy Controller Design

To limit the degree of complexity of the study, the fuzzy controller will be designed to allow variation in the width of the input membership functions only. The rules and output membership functions will be made static. To allow variation in the width at least of the input membership functions is to accommodate optimization processes. To have an average speed in the execution, seven membership functions will be chosen.

The controller design described in this section will be facilitated by the *Fuzzy Logic Toolbox* for MATLAB.

4.3 Optimization Program Coding

Since this study intends to apply GA in the optimization of a fuzzy controller, a genetic optimization program will be developed. This program will be coded using MATLAB. It will utilize functions from the Genetic Algorithm Optimization Toolbox (GAOT). The mechanics of this program is described and illustrated in Section 1.4 and Figure 1.1, respectively. (See page 6.)

4.4 Operator Combination Search

This study will make a cursory exploration on the combination of genetic operators that would yield an excellent convergence. Using the developed program, several trials will be run for every combination using default options. Fewer *generations* will be allowed for each trial to shorten runtime. ISE will be recorded for each trial. Average ISE and standard deviation for each combination will be computed. The lowest average ISE returned would be the basis for the selection of a suitable genetic operator combination.

4.5 Simulated Genetic Optimization

With the selected GA operator combination, the developed simulator will be allowed to autonomously optimize the parameter of the controller. Further tuning will be done to reach a satisfactory optimum ISE. Several trials will be run for every adjustment to have a conclusive results. ISE and *generations* will be tabulated for analysis.

4.6 Evaluation

To be able to determine that the optimization process has reached its goal, a performance index will be necessary. ISE will be used in this study to measure the performance of the system under fuzzy control since it is found suitable for many control optimization problems [6]. (See Section 3.2.1 on page 28 for the description.) It should be kept at a level of 0.052 or less for one to be able to say that GA is applicable for this type of system.

Generation will be used to quantitatively describe its convergence rate. It is the evolutionary equivalence of iteration used in most derivative-based optimization methods. To describe *generation*, it is the single application of the three operators after which is the stopping condition testing. This genetic repeat-until process will go for another generation until stopping condition is met.

CHAPTER 5
DESIGN AND IMPLEMENTATION

This chapter presents the implementation of the study. Section 5.1 discusses the implementation of the system design. It is followed by the implementation of the fuzzy logic controller design. Section 5.3 presents optimization program coding implementation followed by the operator combination search implementation. The simulated genetic optimization implementation follows next. The last section of this chapter discusses the research implementation.

5.1 System Design Implementation

The system block diagram in Figure 5.1 was generated with the aid of *Simulink*. This represents a speed control system. As seen from the figure, the diagram consists the following blocks:

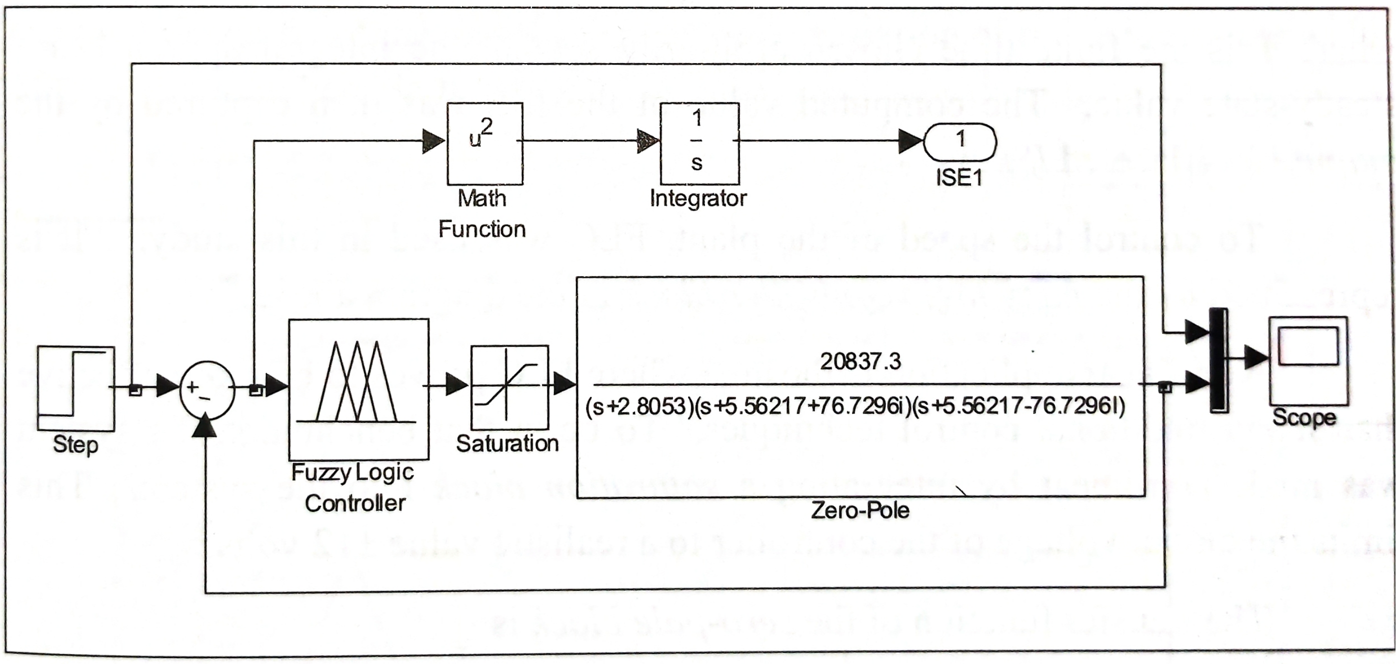


Figure 5.1. Speed control system block diagram.

- Step Source
- Math Function

- Integrator
- Output
- Fuzzy Logic Controller
- Saturation
- Zero-Pole
- Scope

The *step source block* simulated the voltage source to be supplied to the plant under control which is a *dc motor* and is depicted by the *zero-pole block-set*.

As the chosen performance index criterion of this study, the ISE was calculated by the *math function* and *integrator block-sets*. These two block-sets implemented the ISE formula,

$$ISE = \int_0^T e^2 \, dt \qquad 5.1$$

where T is the finite time chosen arbitrarily so that the integral approaches a steady-state value. The computed value of the ISE was then captured by the *output block* labeled *ISE*.

To control the speed of the plant, FLC was used in this study. It is represented by the *fuzzy logic controller block* in the diagram above.

Non-linear application is the area where FLC proves to be more effective than many traditional control techniques. To be in that benchmark, the system was made non-linear by integrating a *saturation block* into the system. This limits the output voltage of the controller to a realistic value ±12 volts.

The transfer function of the *zero-pole block* is

$$T(s) = \frac{20837.3}{(s + 2.80538)(s^2 + 11.124S + 5918.4)} \qquad 5.2$$

It includes the transfer functions of the dc motor and its coupled amplifier used by Chuy in his research [4]. This was so selected that the system can model a practical speed control system.

The last block in the diagram is the *scope*. This is optional but it was deliberately included to vitally monitor the optimization progress.

5.2 Fuzzy Logic Controller Design Implementation

Gaussian membership functions were chosen for the input membership functions and triangular membership functions for the output membership functions.

The calculation of the value of a Gaussian membership function is given by the equation below,

$$\text{Membership Value} = e^{\frac{\left(x - x_{center}\right)^2}{2*\sigma^2}} \qquad 5.3$$

where x is the input, x_{center} is the mean of the Gaussian distribution, and σ is the standard deviation of the distribution. It is the variation of σ that varies the width of the input membership function.

Figure 5.2 below shows the initial form of the input membership functions.

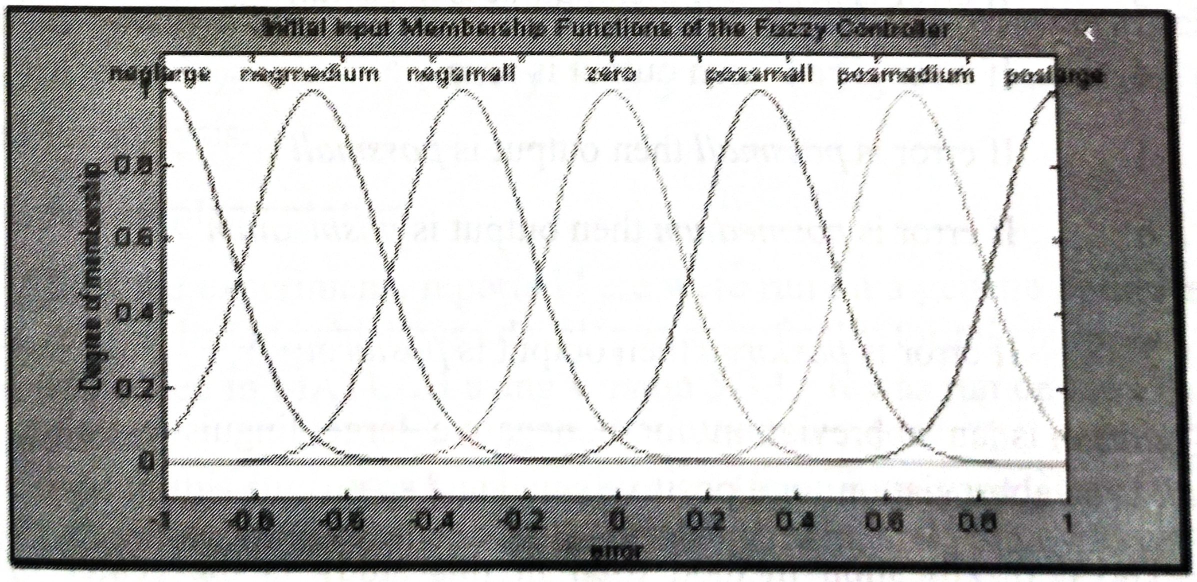


Figure 5.2. Initial Input Membership Functions.

And the output membership functions are shown in Figure 5.3 below.

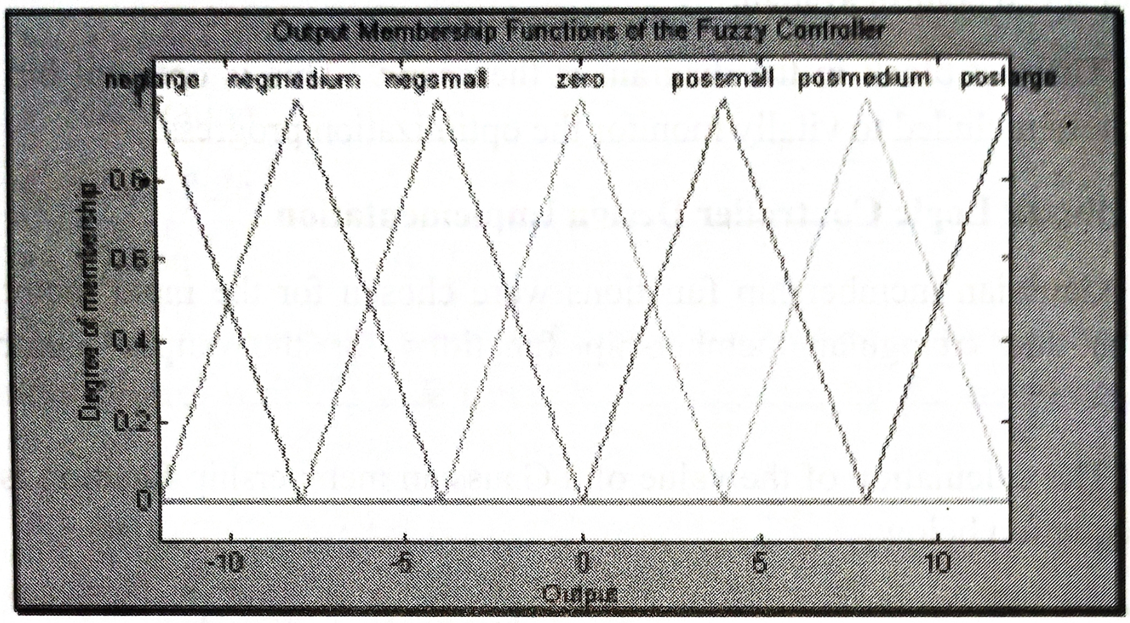


Figure 5.3. Output Membership Functions.

Below were the rules used by the FLC:

Rule

1 If error is *neglarge* then output is *neglarge*

2 If error is *negmedium* then output is *negmedium*

3 If error is *negsmall* then output is *negsmall*

4 If error is *zero* then output is *zero*

5 If error is *possmall* then output is *possmall*

6 If error is *posmedium* then output is *posmedium*

7 If error is *poslarge* then output is *poslarge*

The *neglarge* is an abbreviation for a negative large linguistic value. The *possmall* is an abbreviation for a positive small and so on.

The defuzzification method used in this study is the *center of area method* described in Section 3.1.3 on page 27.

5.3 Optimization Program Coding Implementation

A program had been coded under a filename, *run_gafuz7.m*. Refer to Appendix A for the source code of this routine. This program utilized the *ga* function from the GAOT. This optimization function used an evaluation function that would update the FLC parameters and calculate the ISE every time it is called. *DC_GAFUZ7.m* was developed for this purpose. It was designed to return a negative value of ISE since the *ga* function can only do maximization task. Maximizing negative ISE is just another way of minimizing ISE. See Appendix B for the source code of this evaluation function.

5.4 Operator Combination Search Implementation

This study made a cursory exploration on the combination of genetic operators that would yield an excellent convergence. *run_gafuz7.m* was run for thirty (30) trials for every combination using default options. Forty (40) *generations* were allowed for each trial to shorten runtime. ISE was recorded for each trial. Average ISE and standard deviation for each combination were computed. The lowest average ISE returned was the basis for the selection of a suitable genetic operator combination.

5.5 Simulated Genetic Optimization Implementation

With the selected GA operator combination, *run_gafuz7.m* was allowed to autonomously optimize the ISE of the system. Further tuning on the termination option was done since satisfactory optimum ISE was not reached. Thirty (30) trials were run for every tuning. ISE and *generations* were then tabulated for analysis.

5.6 Research Implementation

All of the experiments reported here were run on a genetic optimization program, *run_gafuz7.m*, which was developed specifically for this purpose. This program was coded in MATLAB using version 5.3.1. It was run on the Pentium III PC workstation operated by Microsoft's Windows 98. The machine and software used in this study was provided by IRC (Instrumentation, Robotics, and Control) Laboratory of the College of Engineering, Mindanao State University - Iligan Institute Of Technology, Tibanga, Iligan City.

CHAPTER 6

RESULTS AND DISCUSSION

This chapter discusses the results of this study. It shows the outcome of the cursory exploration on the GA operator combination. It then presents the optimization results done by the selected combination.

6.1 Operator Combination Search Results

Before the operator combination search was completed, an exploration on the tournament size to be used in the tournament selection operator, *tournSelect*, was done since no default option provided by the GAOT. The exploration was done within the *tournSelect*, *simpleXover*, and *multiNonUnifMutation* combination. The sizes explored were 20, 30, 40, up to 80. Fifty (50) was found suitable for this application. Table 6.1 shows the result. Refer to Appendix C for details.

On the cursory exploration on the appropriate operator combination, the result showed that with all the

Table 6.1. Exploration on the tournament size

Tournament Size	Average ISE	Standard Deviation
20	0.0854	0.0761
30	0.0816	0.0420
40	0.0772	0.0298
50	0.0661	0.0213
60	0.0742	0.0331
70	0.0749	0.0406
80	0.0855	0.0387

combinations using boundary mutation, the optimization process.failed. This failure had occurred since this mutation operator allows rounding off some parameters to either its lower or upper bound as described in Section 2.3 on page

14. In this study, the lower bound happened to be zero. Rounding off the parameter (i.e., the sigma) of the Gaussian input membership function to zero can make it undefined. Along with this failure mutation operator, roulette selection operator failed to converge with any combination. The rest converged satisfactorily. Among the combination that converged, the *normGeomSelect*, *heuristicXover*, and *unifMutation* combination returned the smallest average ISE of 0.0645 with a standard deviation of 0.0222. Table 6.2 summarizes the results. Appendix D has the details. **(NO APPENDIX SUPPLIED)**

Table 6.2. Operator Combination Search Results

SELECTION	CROSSOVER	MUTATION	AVE. ISE	STD. DEV.
normGeomSelect	arithXover	multiNonUnifMutaion	0.0911	0.0410
"	"	nonUnifMutation	0.1053	0.0471
"	"	unifMutation	0.0870	0.0339
"	heuristicXover	multiNonUnifMutaion	0.0755	0.0303
"	"	nonUnifMutation	0.1069	0.0516
"	"	unifMutation	0.0645	0.0222
"	simpleXover	multiNonUnifMutaion	0.0751	0.0280
"	"	nonUnifMutation	0.1056	0.0443
"	"	unifMutation	0.0847	0.0700
roulette	arithXover	multiNonUnifMutaion	30.9191	36.9748
"	"	nonUnifMutation	28.2437	29.6835
"	"	unifMutation	37.0814	42.2245
"	heuristicXover	multiNonUnifMutaion	31.0225	34.9507
"	"	nonUnifMutation	31.1168	28.5604
"	"	unifMutation	27.7980	15.9218
"	simpleXover	multiNonUnifMutaion	31.2664	28.7629
"	"	nonUnifMutation	45.5697	38.6727
"	"	unifMutation	35.3215	29.9644
tournSelect	arithXover	multiNonUnifMutaion	0.0749	0.0282
"	"	nonUnifMutation	0.1288	0.0506
"	"	unifMutation	0.0774	0.0208
"	heuristicXover	multiNonUnifMutaion	0.0660	0.0267
"	"	nonUnifMutation	0.1090	0.0543
"	"	unifMutation	0.0712	0.0210
"	simpleXover	multiNonUnifMutaion	0.0661	0.0213
"	"	nonUnifMutation	0.1117	0.0481
"	"	unifMutation	0.0945	0.0360

6.2 Simulated Genetic Optimization Results

With the selected operator combination above, the termination options were further adjusted to come with a more satisfactory ISE. Using the error tolerance of 1e-6, the optimization was not able to converge lower than with that of Chuy's [4] work. It was only when the termination options were set to 80 and 1e-8 for the No Change (i.e., the allowed number of generation without change in the best solution) and error tolerance, respectively. It converged with an average ISE of 0.0506 with a standard deviation of 0.0047. Table 6.3 below shows this. See Appendix E for details.

Table 6.3. Fine Tuning Results of the Termination Options.

No Change	Error Tolerance	Ave. ISE	Std. Dn.
40	1e-6	0.0621	0.0183
60	1e-6	0.0628	0.0381
80	1e-6	0.0532	0.0083
100	1e-6	0.0536	0.0077
80	1e-8	0.0506	0.0047

Note: No Change – allowed number of *generation* without change in the best solution.

6.3 An Optimal Input Membership Function

In the last run of the optimization program using the 80 no change and 1e-8 error tolerance in the termination options, the optimal fuzzy parameters returned were 0.0160, 1.0729, 0.4244, 0.0581, 0.4050, 24.1082, and 2.3287. Figure 6.1 shows the input membership function reflecting these optimal parameters. Comparing this from Figure 5.2 (before optimization) on page 49, a noticeable variation of the width of the function is seen after optimization.

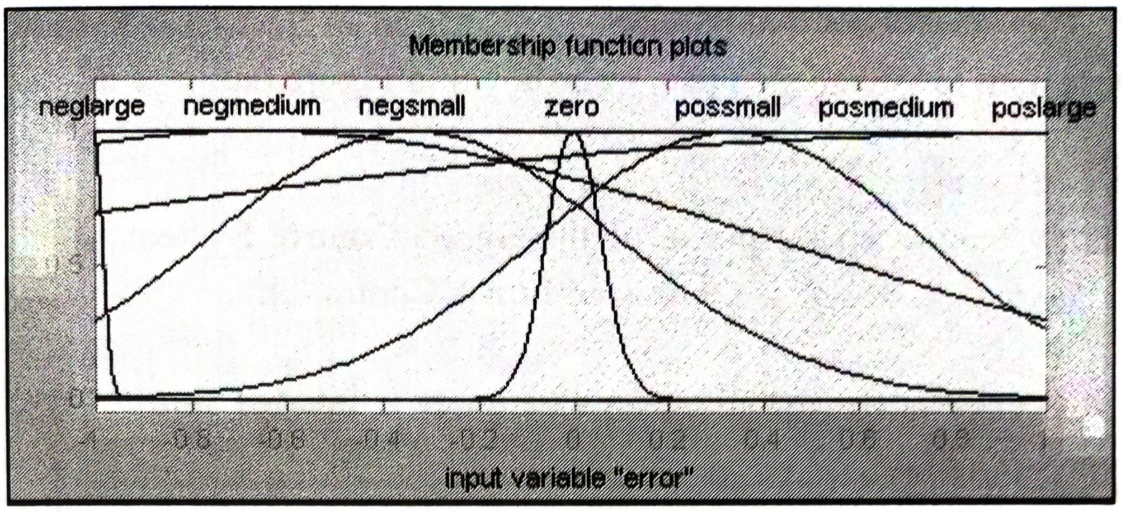
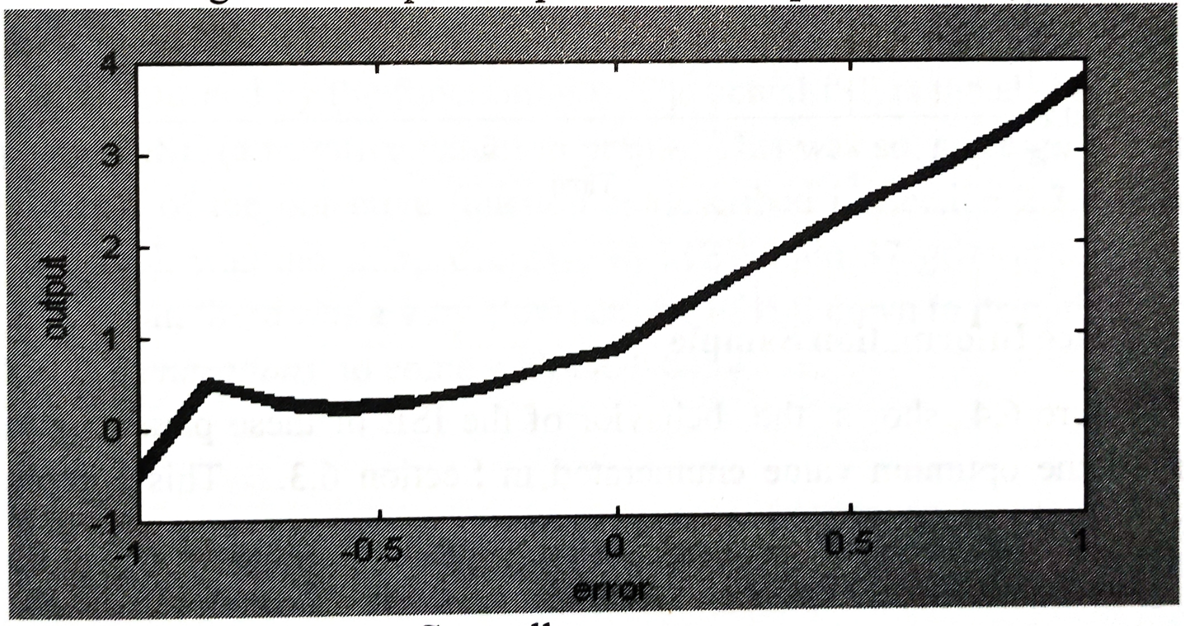


Figure 6.1. The input membership function after optimization.

6.4 Input-Output Relationship

These optimal parameters enumerated in the preceding section leads to an input-output characteristic curve shown in Figure 6.2 below.

Figure 6.2. Input-Output Relationship of the Fuzzy

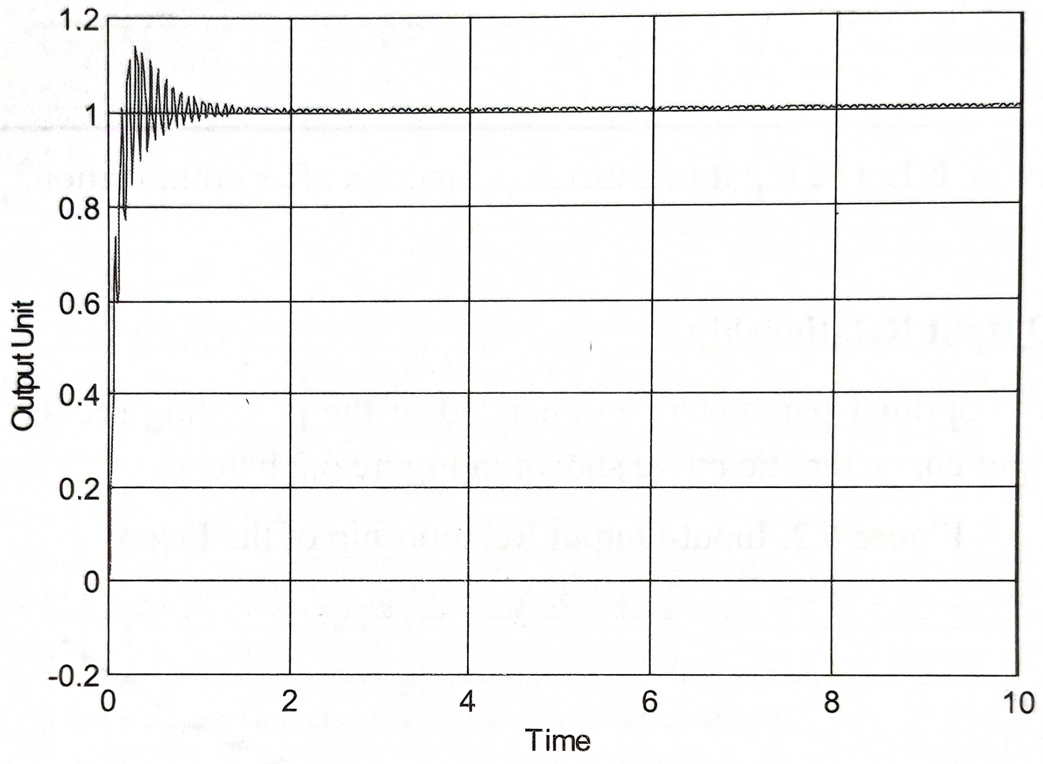


Controller.

6.5 A Step Response Sample

The sample optimal parameters discussed in Section 6.3 made the speed control system to yield a step response shown in Figure 6.3.

Figure 6.3. Step Response of the Speed Control System using an Optimized Fuzzy Controller.



6.6 ISE Trace Information Sample

Figure 6.4 shows the behavior of the ISE of these parameters in the search of the optimum value enumerated in Section 6.3. This graph was derived from the trace

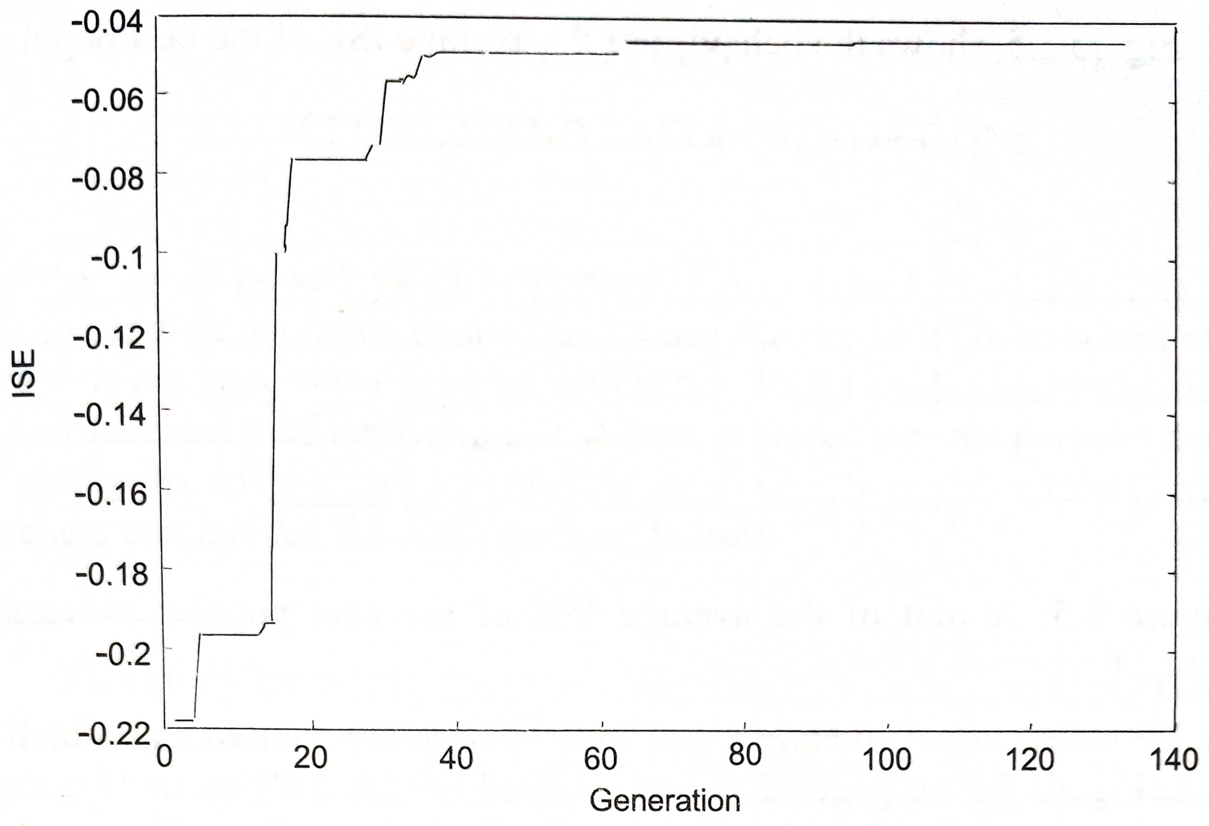


Figure 6.4. A plot of ISE of the best string versus generation.

information returned by the function, *ga*. The actual ISE is the absolute value of the indicated ISE (a negative value) in graph. This was so, since *ga* can only do maximization of the objective function as described in Section 2.3. Revealed from the graph was the sharp decrease in ISE within 37 *generations* (more or less). After that, there was a very slow settling of ISE down to its optimum value. It took 128 *generations* to come with the 0.0454 ISE.

Figure 6.5 shows the behavior of the average ISE of the best population.

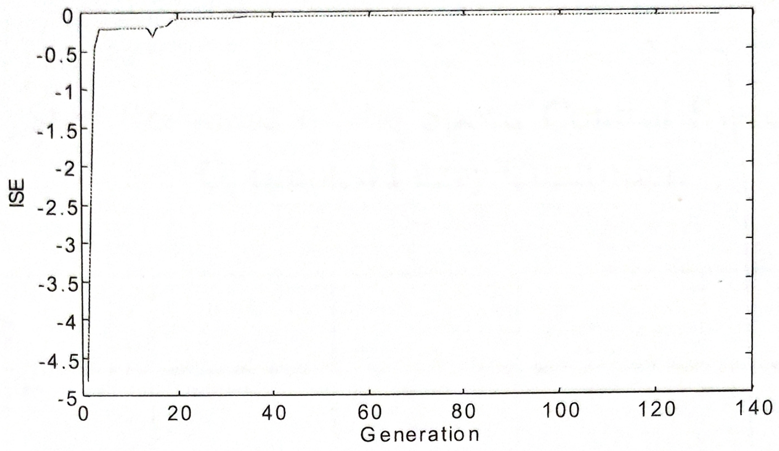


Figure 6.5. A plot of the average ISE of the best population versus generation.

With the aid of *Simulink,* the speed control system model which was shown in Figure 5.1 on page 42, was simulated for ten (10) seconds using the optimal parameters enumerated in Section 6.3. Figure 6.6 shows the result. As indicated in the graph, the optimization converged smoothly. No occasional spikes and sudden drops were observed.

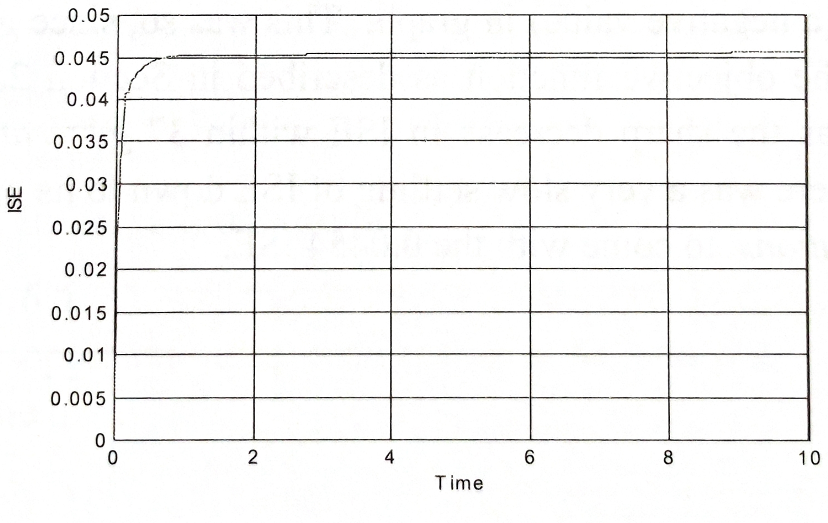


Figure 6.6. A plot of optimum ISE of the speed control system versus time (in seconds).

CHAPTER 7

CONCLUSION AND RECOMMENDATIONS

Systems nowadays are getting more complex, demanding, more intelligent and autonomous controllers. Fuzzy logic, in its determined effort to address this demand, was equipped with nature-based evolutionary algorithms in search of machine self-innovation. GA was explored for this purpose due to its seen ability to invent and re-combine new search paths. However, some researchers commented that GAs are hard to use.

7.1 Conclusion

This study intends to establish a methodology on how to apply GA in the search of a global optimal solution of a fuzzy controller in a speed control system. As a result, this population-based optimization algorithm was able to converge satisfactorily to the lowest ISE of 0.0454 in 128 generations. However, it was just able to return an average ISE of 0.0506 which is insignificantly lower than is reported in Chuy's [4] study using a derivative-based optimization technique by about 2.7 percent. Though the performance reported here did not arrive at a significantly minimal index, the trend of the optimization process encountered in this study showed a very promising result.

Weaknesses were also seen in GA based on the results. It sometimes terminated in an unsatisfactory step response. By mistakenly picking MATLAB functions to be used for selection, crossover, and mutation operations, this algorithm would even fail to converge.

7.2 Recommendations

Most real-time experimentation experience difficulty in the implementation due to some presence of noise in the actual set-up. It is therefore recommended that this study be continued on this assumption to check the consistency of the performance of GA on this type of application. To be more conclusive on the robustness and advantages of GA, an application of the algorithm on other systems and/or controllers could be carried on.

REFERENCES

[1] Akbarzadeh, M.R., Kim, Y.T., and Feerouzbakhsh, B., "*Evolutionary Fuzzy Speed Regulation for a DC Motor*," in Proceeding of the Twenty-Ninth Southeastern Symposium on System Theory, pp. 292-296, March 1997.

[2] Bojadziev, G. and Bojadziev, M., *Fuzzy Sets, Fuzzy Logic, Applications*. World Scientific Publishing, Singapore, 1995.

[3] Braunstingl, R., Mujika, J., and Uribe, J. P., "*A Wall Following Robot with a Fuzzy Logic Controller Optimized by a Genetic Algorithm*," in Proceedings of 1995 International IEEE Conference on Fuzzy Systems, pp. 77-82, July 1995.

[4] Chuy, Jr., Oscar Y., *A Quantitative Comparison of Classical and Fuzzy Controllers Using ISE Criterion*. Unpublished Thesis, UP, Diliman, November 2000.

[5] Coleman, T., Branch, M.A., and Grace, A., *Optimization Toolbox User's Guide*. The Mathworks, Natick, MA, 1990.

[6] Dorf, R.C. and Bishop, R.H., *Modern Control Systems*. Addison-Wesley, Menlo Park, CA, 1998.

[7] Goldberg, David E., *Genetic Algorithm in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[8] Gulley, N. and Jang, R., *Fuzzy Logic Toolbox for use with MATLAB*. The Mathworks, Natick, MA, 1996.

[9] Houck, C.R., Joines, J.A., and Kay, M.G., *Binary and Real-Valued Simulation Evolution for MATLAB*. The MathWorks, Natick, MA, 1996.

[10] Jang, J.-S. R., et. al., *Neuro-Fuzzy and Soft Computing*. Printice Hall, Upper Saddle River, NJ, 1997.

[11] Lu, Yong-Zai, *Industrial Intelligent Control*. John Wiley & Sons, Chichester, 1996.

[12] Mathworks, *Getting Started with MATLAB*, Version 5. The Mathworks, Natick, MA, 1999

[13] Mathworks, *Fuzzy Logic Toolbox User's Guide*. The Mathworks, Natick MA, 1999.

[14] Mathworks, *MATLAB: The Language of Technical Computing*, Version 5.3.1. The Mathworks, Natick, MA, 1999.

[15] Mathworks, *Using SIMULINK*, Version 3. The Mathworks, Natick, MA, 1999.

[16] Moler, C. and Costa, P. J., MATLAB Symbolic Math Toolbox. The Mathworks, Natick, MA, 1997.

[17] Passino, Kevin M. and Yurkovich, Stephen, *Fuzzy Control*, Addison-Wesley, Menlo Park, CA, 1998.

[18] Thornton, Chris, *Why GAs are hard to use?* http://cogslib.cogs.susx.ac.uk/csr_abs.php?csrp399.

[19] Walpole, Ronald E. and Myers, Raymund H., *Probability and Statistics for Engineers and Scientists*, 5th edition. MacMillan Publishing, New York, NY, 1993.