FUSION OF SOFT COMPUTING TECHNIQUES
IN
CONTROL APPLICATIONS

by

A. Buğra KOKU

BS. in M.E., Boğaziçi University, 1994

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Systems and Control Engineering

Boğaziçi University

1997

ACKNOWLEDGMENT

The person, who does not rest satisfied with concocted answers and who does not get tired in competing with the utmost speed of technology, has always inspired us the clues of staying up-to-date and personally proved how problems will wear out with determination and patience. Many sincere thanks to my advisor Prof. Dr. M. Okyay Kaynak.

Professors in my jury, who read through this thesis with patience. Many thanks to Professors Mehmed Özkan, Feza Kerestecioğlu and Mehmet Çamurdan for their contributing comments.

Success is the fruit of a tree, seeded with determination, fed with patience and cared with love.

The man who wisely directed me in life with determination and supported with patience. The woman who cared me with the purest love and affection. To the docile couple to whom I try to be a fruit of success. Many sincere thanks to my parents, to whom I admire and want to be with forever.

The girl whose support helped even from long distances. Many thanks to my sister with the hope that she will do better in academia, as she always did up to now.

The woman who conquered my heart, who fed my joy of life with her existence in my uttermost pessimistic days. Certainly she deserves more...much more...forever...my wife...

Many thanks to the hand with the candle, to the one who shares and who teaches, who lights the candle and to the one who shows the way.

All thanks, cordially, to the one with the knowledge.

August, 1997

A. Buğra KOKU

ABSTRACT

The main objective of this thesis work is the formation of an experimental garden, suitable for the demonstration of classical and advanced control concepts, by an integration of the computational capabilities and the electromechanical equipment that exist in the Mechatronics Laboratory of Boğaziçi University. An equally important and more scholarly objective has been to construct an experimental platform on which the use of soft computing methodologies in the control of nonlinear systems can be studied and experimented with and to actually carry out some experimental investigations in this respect.

The thesis has both theoretical and experimental constituents, on the practical side it details the electromechanical construction of the X-Garden (stands for eXperimental Garden), the development of the software and the experimental designs, whereas the control algorithms designed and implemented using Soft Computing methodologies make up the theoretical side.

The software support developed for the X-Garden as a part of this thesis work can be grouped into two. The first group constitutes the C codes written and the second one is on the experimental designs. Both adaptive and non-adaptive fuzzy controllers are studied both by simulations and by real-time implementations.

The thesis carries a didactic objective too. It is intended to provide a first reading on fuzzy and fuzzy-neuro control. The fundamental concepts of fuzzy logic are and fuzzy-neuro control are therefore explained in detail, in a clear and tractable manner.

The thesis, in some parts, intentionally carries the style of a user's manual, this is to ensure that further users of the X-Garden will have no difficulty in experimenting on the platform constructed, being able to find clear explanations and instructions in this thesis and its appendices on the use of both the mechanical and the software environments.

ÖZET

Bu tez çalışmasının temel amacı, Boğaziçi Üniversitesi'ne ait Mekatronik Laboratuvarının bünyesinde yer alan elektromekanik aksamın, laboratuvarın işlemsel kapasitesi ile bütünleştirilerek, gerek klasik gerekse gelişmiş denetim yordamlarının uygulanabilmesine müsait deneysel bir düzeneğin kurulmasıdır. Çalışmanın en az bu kadar önemli ve daha fazla akademik değeri olan diğer bir hedefini de doğrusal olmayan sistem denetiminde esnek bilgi işlem metodlarının incelenip, uygulanabileceği bir deney sisteminin kurulması ve bu sistem üzerinde deneyler yapılması oluşturmaktadır.

Hem teorik hem de pratik uygulamaları içeren bu tezin pratiğe yönelik kısmını, bahsi geçen ve Deney Bahçesi adı verilen ortamda deney tasarımı ve buna uygun yazılımın geliştirmesi oluştururken, esnek bilgi işleme metodları kullanılarak tasarlanıp, Deney Bahçesinde uygulanan denetim yordamları da çalışmanın teorik yönünü oluşturmaktadır.

Tez çalışmasının bir parçası olarak Deney Bahçesi için geliştirilen yazılımlar iki ana gruba ayrılabilir. Birinci grupta uygulamaya yönelik deneysel tasarımlar yer alırken, ikinci grupta ise gelişmiş denetim yordamlarını gerçeklemeye yönelik C dosyaları bulunmaktadır. Bu sayede, gerek benzetimler gerekse gerçek zamanda uygulamalar ile uyarlamalı ve uyarlamasız bulanık denetleyiciler üzerinde çalışmalar yapılmıştır.

Yazılımı esnasında tezin bulanık ve bulanık-sinirsel denetleyiciler için bir giriş teşkil edecek şekilde derlenmiş olması, tez çalışmasına öğretici bir vasıf da kazandırmaktadır. Bu sebepten ötürü, bulanık mantık ve bulanık-sinirsel denetleyicilerin, temel kavramları mümkün olduğunca detaylı fakat bir o kadar da berrak ve takip edilebilir bir tarzda bölümler halinde işlenmiştir.

Bu tezin bazı kısımları, Deney Bahçesini daha sonra kullanacak olanların düzeneği kullanırken zorluklarla karşılaşmasını engellemek amacı ile bir kullanım kılavuzu niteliği taşımaktadır. Tezin sonundaki ekler vasıtası ile de, elektromekanik aksam, yazılım ve donanım hakkında kullanıcıya açıklayıcı bilgiler sunulmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A, B, C, ...	Classical crisp sets
a_i, b_i, c_i, ...	Set elements of crisp sets
$\tilde{A}$, $\tilde{B}$, $\tilde{C}$, ...	Fuzzy sets
X	Universal set
Ø	Empty set
$\mu_{\tilde{A}}(x)$	Membership value of x to the fuzzy set $\tilde{A}$
α	level cut for fuzzy sets, beam angle for ball and beam applications
T	T-norm
S	S-norm
N	N-norm
f_i	fuzzy rule centers
f^*	defuzzification result
(x,y)	Cartesian Coordinates
Z	Rule outputs in an adaptive fuzzy system
$\bar{y}$	Weight vector in an adaptive fuzzy system
f	Output of an adaptive fuzzy system
e	Error that comes out of an adaptive fuzzy system
(x_G,y_G)	The Cartesian coordinate of the center of mass of the pendulum pole
M	Mass of the inverted pendulum cart
m	Mass of the inverted pendulum pole
θ	Pole angle for inverted pendulum, rotary disk angle for ball and beam
L	Pole length for inverted pendulum and beam length for ball and beam

F	Force exerted to the inverted pendulum cart
V	Voltage
I_m	Motor armature current
R_m	Motor armature resistance
K_g	Internal gear ratio of the cart motor
w	Angular velocity of motor armature
T	Torque
r	Radius of cart motor gear

1. INTRODUCTION

Soft Computing is a term recently coined by Prof. L. Zadeh as an extension to his fuzzy logic concepts presented in [1] by merging it mainly with neural networks and evolutionary computing. It points out the fact that there is a radical difference between perception and expression of humans and the formalism of mathematics and can concisely be defined as a collection of techniques, capable of dealing with imprecise, uncertain or vague information. This concept now has many followers throughout the world, having found itself many application areas, ranging from industrial process control, fault diagnosis and smart appliances to speech and pattern recognition and planning under uncertainty.

Following the trend described above, the Mechatronics Laboratory of EE Department in Boğaziçi University has also directed a part of its research efforts towards the area of Soft Computing and its applications. This laboratory has appreciable computational power and a number of experimental electromechanical systems, the appropriate integration of which has been the main objective of this thesis.

The thesis has both theoretical and practical aspects. Although the main objective is set as the formation of an experimental garden, suitable for the demonstration of classical control concepts, an equally important objective has been to prepare an experimental environment on which novel concepts such as the fusion of soft computing methodologies can be implemented. The practical side of the thesis includes electromechanical construction, software development and the experimental designs, whereas the control algorithms designed and implemented using Soft Computing methodologies constitute the theoretical side.

The experimental garden established within the framework of this thesis has three main members; two pendulums and a ball and the beam setup. When it is noted that an inverted pendulum can also be used as a linear position servo, the number of members increase.

One of the inverted pendulums is a home-made one. It was originally built in Mechanical Engineering Department in 1994 and reconstructed in a more useful manner in Electrical and Electronics Department in 1996. In the early stages of this thesis work, a

considerable amount of effort was spent on it for improving its mechanical structure and its integration into the software environment. However with the arrival of the second pendulum (procured from Quanser) into the garden, which is mechanically more reliable and has a better performance, the home-made one was taken out of the garden for future reconstruction as a rotary pendulum.

The equipment outlined above are connected to a PC, equipped with a real-time DSP card and the associated software. An experimental medium, named X-Garden (stands for eXperimental Garden) decorated with different plants and control structures has thus been formed.

The software support developed for the X-Garden as a part of this thesis work can be grouped into two. The first group constitutes the C codes written and the second one is on the experimental designs. The real-time DSP is capable of compiling Simulink files. Therefore, all the control algorithms considered in this thesis are designed under Simulink. However, when the Simulink blocks do not fulfill the requirements, C codes can be written and embedded into Simulink. In this thesis work, an appreciable amount of C codes has been written for the realization of fuzzy and adaptive fuzzy controllers, in such a manner that they can easily be followed and adapted as necessary by the future users of the garden. When a simulink file is compiled by the real-time workshop and downloaded onto the DSP (the details of which will be explained in the thesis) the Simulink tools become ineffective and the experiment proceeds under two software packages, called Cockpit and Trace by Matlab. Some very basic files are also provided in this environment to reduce time losses.

This thesis intentionally goes into the very basics of some concepts because, besides being a scholarly work, it is intended to provide a first reading on fuzzy and neuro-fuzzy control . The theoretical concepts and the experimental design and implementation are tried to be presented in a clear, tractable and repeatable manner. As a first step, in Chapter 2, the fundamental concepts of fuzzy logic are introduced, namely basics of fuzzy sets, most common terminology and membership functions, set theoretic operations and operations on fuzzy sets. Secondly, in Chapter 3, the applications of fuzzy logic in controls area are considered and the main blocks of a classical fuzzy controller are described, an example is presented. In Chapter 4, the fuzzy-neuro control architecture is introduced. This concept is a blend of basic fuzzy control structure, improved with the learning capability of artificial neural networks.

The thesis continues in Chapter 5, with the derivation of the mathematical models of inverted pendulum, linear position servo and ball and beam setups. In the next chapter, detailed explanations on the components that constitute the experimental garden constructed are presented. This part is divided into two, as the experimental and the computational mediums. Within the experimental medium, the mechanical setup and the power amplifier are explained. The interaction of the user and the mechanical setup by the use of the software available is explained in the sections on the computational medium.

Chapter 7 is dedicated to the operating principles of the mechanical systems using the DSP interface. This completes the background for the applications which are presented in the next chapter. The simulations and the real-time applications form the two main topics in this part. Both adaptive and non-adaptive applications are presented by simulations and real-time implementations. Finally, in six appendices, physical parameters of the experimental setup are tabulated, DSP equipment specifications, C and Matlab code templates, commands and file locations, C codes written and a paper written during this study are presented respectively.

It should here finally be remarked that by reading through this thesis only a basic but narrow vision can be obtained on soft computing methodologies and their fusion in control applications. The subject area is very wide and includes, in addition to fuzzy logic, artificial neural networks and evolutionary algorithms. In this thesis, only an introduction to the fuzzy logic and its applications to control are introduced, although both adaptive (fuzzy-neuro) and non-adaptive systems are explained and reinforced with examples.

2. AN INTRODUCTION TO FUZZY LOGIC

In this chapter, an introduction to fuzzy logic is on a comparisonal basis with respect to the crisp logic i.e. the logic that is generally named as the classical logic. It is later seen that crisp logic can be considered to be a special case or more specifically the boundary conditions of fuzzy logic.

The dictionary meanings of the words fuzzy and crisp are:

Fuzzy : *(of something seen)* not clear in shape, esp., at the edges, misty. [2]

Crisp : showing no doubt or slowness, clear. [2]

Being conditioned to think in terms of rights or wrongs, it may therefore appear to us that fuzzy logic is not or does not sound to be a reliable tool for use in science. However, a closer look to the concepts proposed within the body of fuzzy logic reveals the fact that, it is an intuitive approach to deal with real world situations.

When real world situations are considered, it is realized that:

> *"Real situations are very often not crisp and deterministic, and they can not be described precisely. The complete description of a real system often would require far more detailed data than a human being could ever recognize simultaneously, process and understand"* [3].

These properties of life generally makes it very difficult to have precise models of facts which are to be dealt with the conventional mathematical methods. The real world situations are intuitively nonlinear, complex and uncertain, on top of which, our information about them are imprecise.

One can imagine two different spaces, in one, the problem exists and in the other the solution methods and mathematical tools exist. If these two space overlap, even though it is not a frequent case and realistic, it is very desirable, since the solution will be reached if it is in the span of the tools available. However, if such an overlap does not exist, two alternatives arise: One can either transform the problem to the solution space or seek for a solution in the problem space. The former attempt ends up in linearization techniques, simplification assumptions (lumped behavior etc.) and similar methods which transform the problem to a form which can be handled with the tools available. In the latter approach, one either transforms the tools likely to lead to a solution from solution space to the problem space or tries to formulate new tools in the problem space. Fuzzy logic may be perceived as a tool in the problem space with an image on the solution space. B. Russell's following statement is very helpful at this point:

> *"All traditional logic habitually assumes that precise symbols are being employed. It is therefore, not applicable to this terrestrial life but only to an imagined celestial existence."* [4]

Also what A. Einstein stated about reality and mathematics is very interesting:

> *"So far as the laws of mathematics refer to reality, they are not certain. And so far as they are certain, they do not refer to reality."* [5]

Fuzzy logic is not a very well developed theory like linear systems theory, but is more promising for the future, in dealing with problems of real life. However, the way it is presented sounds very natural and the way it handles the problems makes fuzzy logic very applicable in many areas such as, medicine [6], meteorology [7], control [8], and many others. [9], [10]

The basics of fuzzy logic is presented in the literature, generally with the very classical example of the set of "tall men" and it is most generally presented on a comparisonal basis with what crisp logic says and with what fuzzy logic proposes. The basics of fuzzy logic will be explained below following a similar trend with slightly different examples. It is to be realized that covering the whole fuzzy logic theory is impossible within the confinements of a masters thesis. Nevertheless, the introductory material that is going to be presented below

will be sufficient as a basis for fuzzy logic applications in controls. For more detailed information for on both fuzzy logic and its applications in different areas, the reader is referred to [3], [11], [12], [13].

2.1. The Main Idea of Fuzzy Logic

Fuzzy logic is defined to be much closer in spirit to human thinking and natural language than traditional logic [9]. The example of a "large house" may be helpful in clarifying the concept. Consider a crisp (classical) set which defines a large house. The classical approach is to put a threshold value, above which a house is called "large".

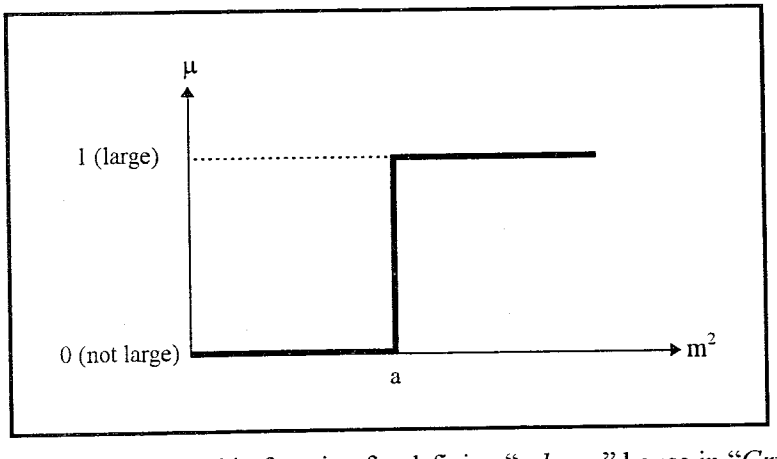


FIGURE 2.1. Membership function for defining "*a large*" house in "*Crisp Logic*".

As is seen in Figure 2.1. the houses above "a" m^2 are defined to be large whereas the others are defined not to be large. The weakness of this definition relies at the neighborhood of the transition of membership values from 0 to 1.Two houses with areas of "a+0.01" and "a-0.01" are classified to be large and not large respectively. Since crisp logic does not allow any values other than 0 or 1 for an element as a membership value to a set, the above mentioned illogical result is inevitable. Fuzzy logic, however, offers grades of membership to a set.

Fuzzy logic allows membership values in the range of [0,1] as shown in Figure 2.2. By the help of this flexible measure of membership definition, the above mentioned problem of abrupt transition in a crisp set is solved.

Fuzzy logic offers continuous membership values in the range of [0,1] (this range is conventional and has universal acceptance, but the range does not have to be within these limits). The membership value for an element indicates the degree of belonging to a set. As the membership value gets closer to 1, the degree with which an element belongs to a set increases and when it is 1, that element belongs to that set. Inversely, as the membership value gets closer to 0, the degree with which an element belongs to a set decreases and when it is 0, that element does not belong to that set at all.

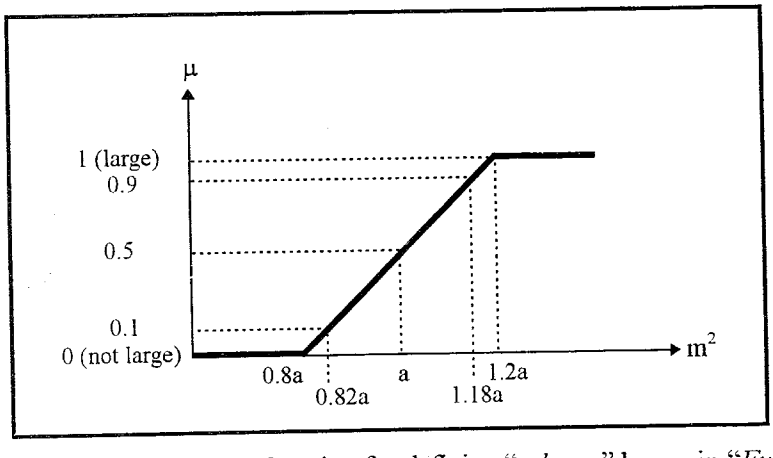


FIGURE 2.2. Membership function for defining "*a large*" house in "*Fuzzy Logic*".

This graded membership value assignment can also be perceived as a measure of fitness. If an element fits the definitions of a set, its membership value is 1, and if it does not fit, it is 0. The values in between 0 and 1 indicate how much an element fits to a set.

For example, a house of 1.18a m^2 fits to the definition of large house, with a degree (membership value) of 0.9. Similarly, a house of a m^2 may be accepted as large, and not large at the same time with equal degree.

What is stated here, indeed, violates the two laws of crisp sets: law of *excluded middle* and *law of contradiction*, since an element may somewhat belong to a set and not belong it at the same time.

$$A \cap \overline{A} = \varnothing \qquad \text{Law of Contradiction}$$

$$A \cup \overline{A} = X \qquad \text{Law of Excluded Middle}$$

where X is the universal, $\varnothing$ is the empty, A is a crisp set and $\overline{A}$ is the complement of the set A. Later, after intersection and union are defined on fuzzy sets, this violation will become clearer.

One possible way to perceive these membership values in the range [0,1] is from the probabilistic point of view. When the membership value is 1, the element is surely a member of a set, when it is 0, it is certainly not a member and the values in between indicate probability of the member belonging to that set. This way of perception is not totally correct, and should be avoided. The information content of the fuzzy set and the above mentioned probabilistic comment is totally different.

To visualize this difference, let's consider the following example. Assume that a number is close to five with a probability of 0.9. Since probabilistic statements are a priori, the mentioned number may be -100 or +1000 with a probability of 0.1. However, an element with a membership value of 0.9 is surely in the vicinity of the number 5 depending on the shape of the membership function.

Therefore, "fuzziness describes the ambiguity of an event, whereas randomness describes the uncertainty in the occurrence of an event." [11]

Before proceeding to the basic definitions of fuzzy logic, the importance of symbolism will be pointed out.

As it is the case for nearly every novel idea, fuzzy logic is also attacked by many scientists with different claims, such as the assertion that what is proposed by fuzzy logic is nothing new and it could be governed by using probability.

One has to be aware of the importance of the way things are perceived and implemented. The basic idea of fuzzy logic is very intuitive, and it makes things easier in many cases. Even if it was nothing but just another way of representing mathematical facts, the way it makes things easier should be appreciated, because sometimes complicated problems can be solved only by the use simple representational conventions. For example, polar coordinate representation is very suitable when dealing with circular functions. However, fuzzy logic is not only an easy way of representation but also it puts forward

some novel ideas. The use of linguistic variables, rather than quantitative variables in representing imprecise concepts, underlies the power of fuzzy logic.

> *"Fuzzy Logic seems to be most successful in two kinds of situations:*
>
> *i) very complex models where understanding is strictly limited, or, in fact, quite judgmental, and*
>
> *ii) processes where human reasoning, human perception, or human decision making are inextricably involved."* [11]

Finally, at the completion of this section, the statement of Aristotle (384-322 BC) is very meaningful:

> *"It is the mark of an instructed mind to rest satisfied with that degree of precision which the nature of the subject admits, and not to seek exactness where only an approximation of the truth is possible."*

2.2. The Basics of Fuzzy Sets

The basics of fuzzy sets, as mentioned before, will be explained in connection with crisp sets on a comparative basis. Only an introductory material will be covered below to enable the reader to get used to the basic terminology and to the concepts of fuzzy sets and fuzzy logic. The references [3], [11], [14] are suggested as further readings.

A classical (crisp) set is defined as a collection of elements or objects. These objects can be finite, countable or uncountable and they either belong to a set or not. A crisp set can be described in two ways: either by listing all of its members (the list method) which is preferable when the number of elements are few in number or they cannot be defined by their properties, or by stating the distinguishing properties of the elements (the rule method).

Let A and B two finite sets. By the list method the set A can be defined as,

$$A = \{a_1, a_2, a_3, \ldots, a_n\}.$$

and by the rule method the set B can be defined as,

$$B = \{b \mid b \text{ has the properties } P_1, P_2, P_3, \ldots, P_n\}.$$

An element may belong to a set or not, however, in the case of fuzzy sets, an element has a degree of membership. An element may belong to a set to a degree and not belong to the same set to some other degree. Crisp sets can be perceived as an extension to or the boundary conditions of the fuzzy sets.

From now onwards, " ~ " will be used to differentiate fuzzy sets from crisp sets. Let $\tilde{A}$ be a fuzzy set,

$$\tilde{A} = \{x, \mu_{\tilde{A}}(x) | x \in X\}.$$

The fuzzy set $\tilde{A}$ is defined in the universe of discourse X (the universal set) and x is an element of X. The degree of membership of x to the fuzzy set $\tilde{A}$ is given by the membership function $\mu_{\tilde{A}}(x)$.

Fuzzy sets are defined in different ways:

By ordered set of pairs. The first element representing the element of the set, the second, representing its degree of membership, i.e., [(3, 0.1), (4, 0.4), (5, 1.0), (6, 0.5), (7, 0.1)].

By assigning a function to calculate the membership values, i.e.,

$$\tilde{A} = \{x, \mu_{\tilde{A}}(x) | \mu_{\tilde{A}}(x) = \left(1 + \left(\frac{x-5}{2}\right)^6\right)^{-1}\} \qquad (2.1)$$

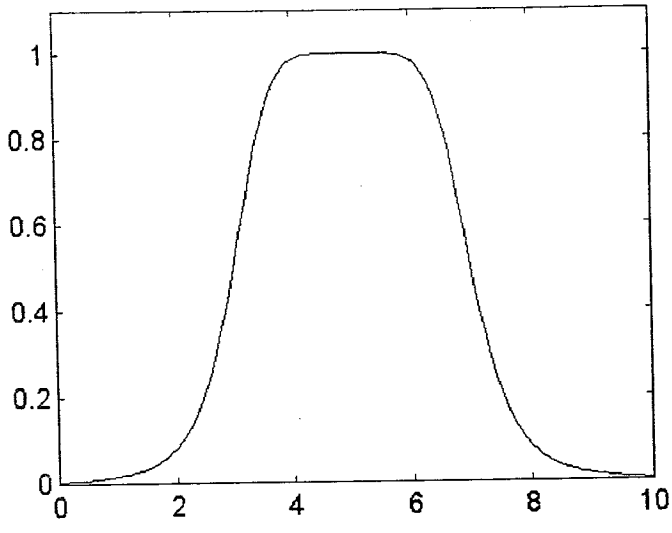


FIGURE 2.3. The Graph of the membership function in (2.1.)

By a summation or integral, i.e.,

$$\tilde{A} = \mu_{\tilde{A}}(x_1)/x_1 + \mu_{\tilde{A}}(x_2)/x_2 + \ldots \mu_{\tilde{A}}(x_n)/x_n$$

$$= \sum_{i=1}^{n} \mu_{\tilde{A}}(x_i)/x_i$$

or

$$\tilde{B} = \int_x \mu_{\tilde{A}}(x)/x$$

more specifically,

$$\tilde{A} = 0.1/1 + 0.4/2 + 0.8/3 + 1/4 + 0.5/5 + 0.2/6$$

and

$$\tilde{B} = \int_{x} \frac{1}{1+\left(\frac{x-5}{2}\right)^{6}} / x .$$

It is to be noted that the above mentioned summation and integration only implies that fuzzy set $\tilde{A}$ or $\tilde{B}$ is a **collection** of the elements inside the summation or integration, depending on the discourse of the set being discrete or continuous. They do not act as summation and integration in the classical sense, but they only serve as a specific notation and define .a fuzzy set as a collection of its elements with their degrees of membership values.

2.3. Terminology

The most common terminology used in defining fuzzy sets will be defined below.

Support: The support of $\tilde{A}$ is the set of x where $\mu_{\tilde{A}}(x)$ is greater than 0.

$$\text{support}(\tilde{A}) = \{x | \mu_{\tilde{A}}(x) > 0, x \in X\}$$

Core: The core of $\tilde{A}$ is the set of x where $\mu_{\tilde{A}}(x)$ is equal to unity.

$$\text{core}(\tilde{A}) = \{x | \mu_{\tilde{A}}(x) = 1, x \in X\}$$

Boundary: The boundaries of $\tilde{A}$ are the sets of x where $\mu_{\tilde{A}}(x)$ is between 0 and 1.

$$\text{boundary}(\tilde{A}) = \{x | 0 < \mu_{\tilde{A}}(x) < 1, x \in X\}$$

Crossover Point: The crossover point is the point x where $\mu_{\tilde{A}}(x) = 0.5$.

$$\text{crossover}(\tilde{A}) = \{x | \mu_{\tilde{A}}(x) = 0.5, x \in X\}$$

Bandwidth (or simply *width*): The distance between the two unique crossover points of a normal and convex fuzzy set $\tilde{A}$ is called the bandwidth or simply the width, which can alternatively be written as $A_{0.5}$ in terms of α-cuts (which are explained below).

Height: The height of a fuzzy set $\tilde{A}$ is the maximum value of the membership values.

$$\text{height}(\tilde{A}) = \max(\mu_{\tilde{A}}(x))$$

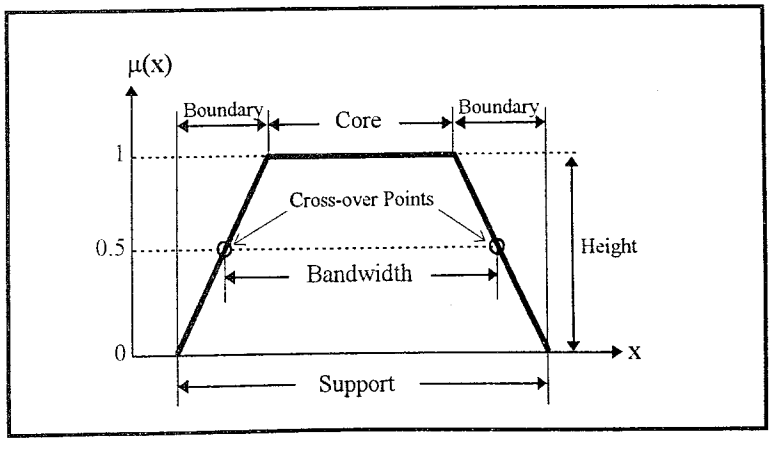


FIGURE 2.4. The terminology used in describing fuzzy sets.

Normal and Subnormal Fuzzy Sets: A fuzzy set which has at least one point x, where $\mu_{\tilde{A}}(x) = 1$, is called normal. If there is a unique point x' where $\mu_{\tilde{A}}(x') = 1$, the point x' is called the prototype of the set. A fuzzy set is called subnormal if it is not normal. In other words, if the height of a fuzzy set is 1, it is normal, otherwise, and it is subnormal.

Fuzzy Singleton: A fuzzy set $\tilde{A}$ with a single point support at x' where $\mu_{\tilde{A}}(x') = 1$.

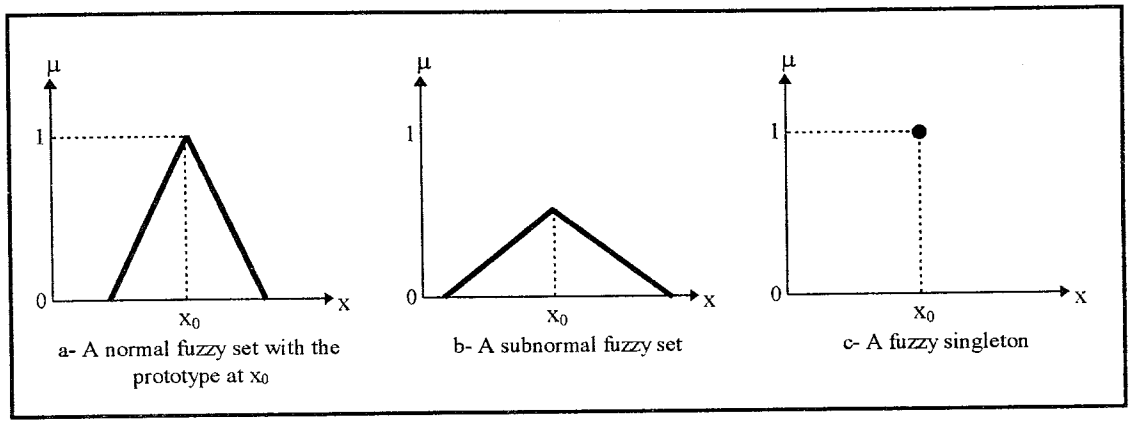


FIGURE 2.5. Normal and subnormal fuzzy sets and fuzzy singleton.

α-cut, strong α-cut: α-cut or α-level set is defined by the crisp set of elements defined by,

$$A_\alpha = \{x | \mu_{\tilde{A}}(x) \geq \alpha\}$$

similarly, strong α-cut or strong α-level set is defined by the crisp set of elements defined by,

$$A'_\alpha = \{x | \mu_{\tilde{A}}(x) > \alpha\}.$$

Alternatively, the support and the core of the fuzzy set $\tilde{A}$ can be represented as A_0' and A_1 respectively.

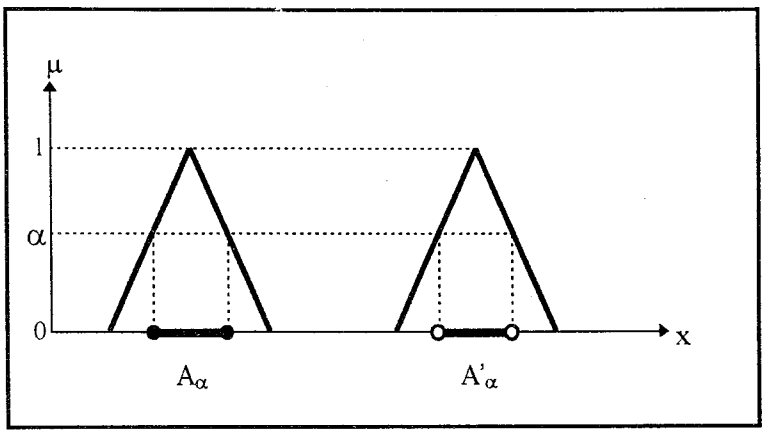


FIGURE 2.6. α-cut or strong α-cut.

Convexity: A fuzzy set $\tilde{A}$ is defined to be convex if,

$$\mu_{\tilde{A}}(\lambda x_1 + (1-\lambda)x_2) \geq \min\{\mu_{\tilde{A}}(x_1), \mu_{\tilde{A}}(x_2)\}$$

for all $x_1, x_2 \in X$, $\lambda \in [0,1]$.

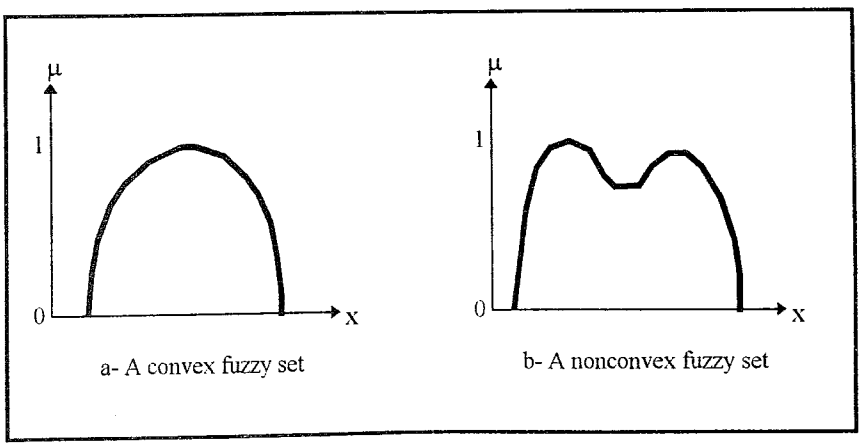


FIGURE 2.7. Convex and non-convex fuzzy sets.

In other words, a fuzzy set is convex if its membership values are strictly monotonically increasing, strictly monotonically decreasing or first strictly monotonically increasing then decreasing in the same manner, which means that, any α-cut in a fuzzy set is a continuous line or a single point. In the classical sense, convexity means that, if two elements are members of a set, then the elements in between are also elements of the same set. In fuzzy sets, this may be interpreted as, if two elements have the same degree of membership value α, the elements in between have degrees of membership values greater than or equal to α.

Fuzzy Numbers: A fuzzy number A is a normal and convex fuzzy set.

Symmetry: A fuzzy set $\tilde{A}$ is symmetric if its membership function is symmetric with respect to a point x_0.

$$\mu_{\tilde{A}}(x_0 + x) = \mu_{\tilde{A}}(x_0 - x), \ \forall x \in X$$

Open left, open right, closed fuzzy sets:

A fuzzy set $\tilde{A}$ is open left if,

$$\lim_{x \to -\infty} (\mu_{\tilde{A}}(x)) = 1 \text{ and } \lim_{x \to \infty} (\mu_{\tilde{A}}(x)) = 0 .$$

A fuzzy set $\tilde{A}$ is open right if,

$$\lim_{x \to -\infty} (\mu_{\tilde{A}}(x)) = 0 \text{ and } \lim_{x \to \infty} (\mu_{\tilde{A}}(x)) = 1 .$$

A fuzzy set $\tilde{A}$ is closed if,

$$\lim_{x \to -\infty} (\mu_{\tilde{A}}(x)) = \lim_{x \to \infty} (\mu_{\tilde{A}}(x)) = 0 .$$

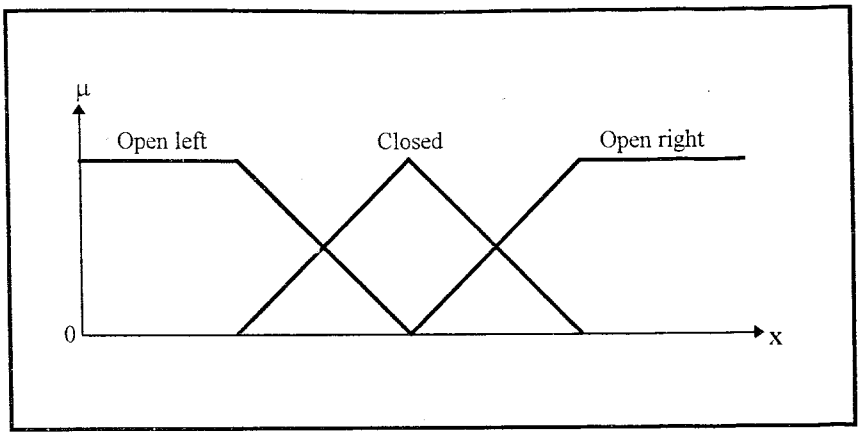


FIGURE 2.8. Open left, closed and open right fuzzy sets.

2.4. Most Common Membership Functions

In this section, the most common membership functions used in literature are introduced.

2.4.1. Triangular Membership Functions

A triangular membership function is defined by three parameters; a, b and c as shown in Figure 2.9.

Formally, the triangle can be defined as,

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

However, to make this formulation more suitable for computation, it can be rewritten in the form:

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

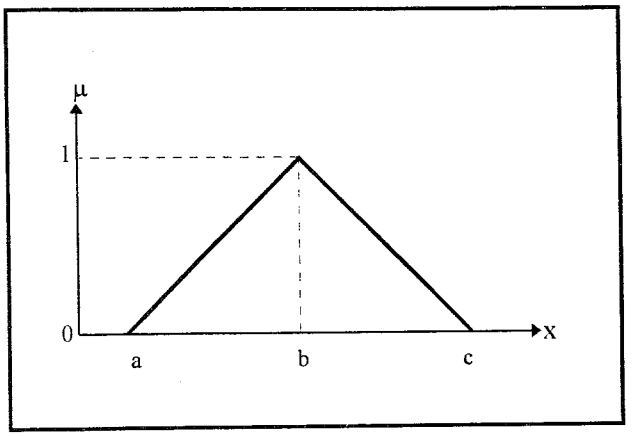


FIGURE 2.9. Triangular membership functions.

2.4.2. Trapezoidal Membership Function

A trapezoidal membership function is defined by four parameters; a, b, c and d as shown in Figure 2.10.

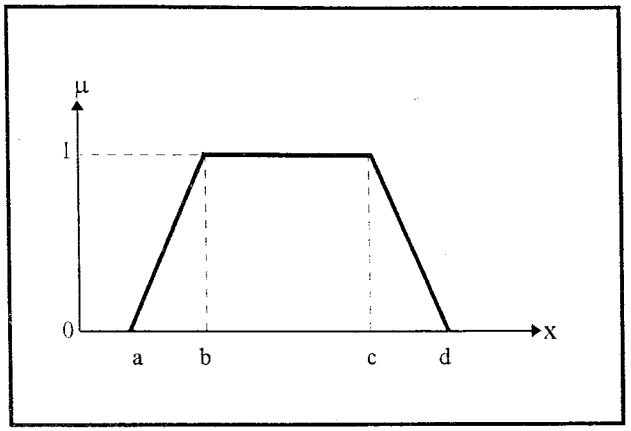


FIGURE 2.10. Trapezoidal membership functions.

Therefore,

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

Similarly, in a more computationally suitable way,

$$\text{triangle}(x,a,b,c,d)=\max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right),0\right).$$

It is obvious that the trapezoidal membership function reduces to a triangular membership function in case **b** is equal to **c**.

2.4.3. Gaussian Membership Function

A Gaussian membership function is defined by two parameters; c and σ where c determines the center and σ determines the width of the membership function.

$$\text{gaussian}(x,c,\sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

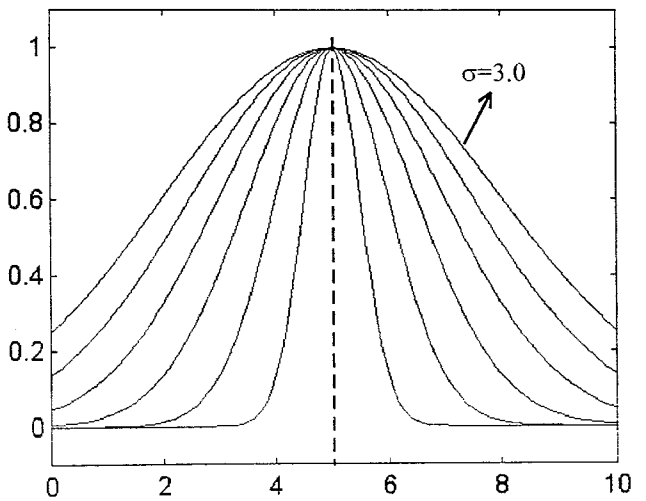


FIGURE 2.11. The Gaussian membership functions with c=5 and σ = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0.

2.4.4. Bell Membership Function

A Bell membership function is defined by three parameters; a, b and c.

$$\text{bell}(x,a,b,c)=\frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}}$$

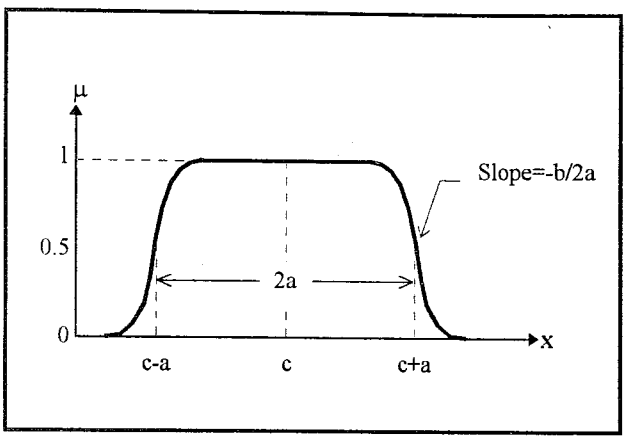


FIGURE 2.12. The meaning of the parameters defining a Bell shaped membership function.

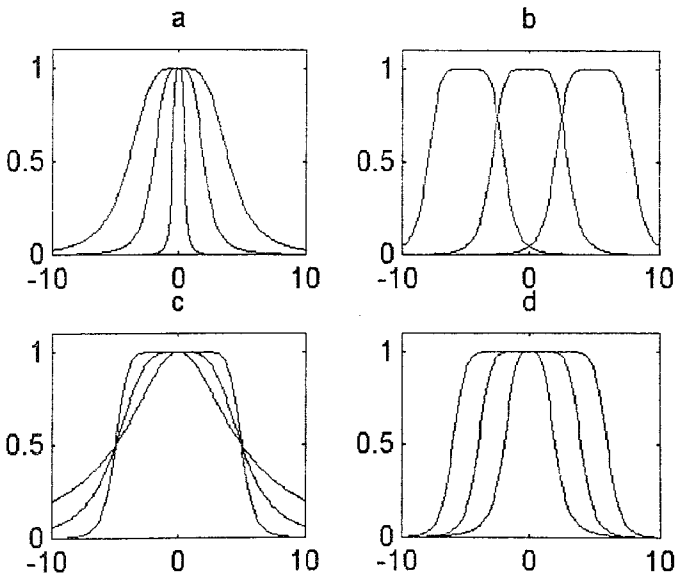


FIGURE 2.13. The effect of parameters in Bell membership function.

a- Effect of a, a=[0.5 2 5], b=2, c=0.

b- Effect of c, a=3, b=3, c=[-5 0 5].

c- Effect of b, a=5, b=[0.5 1 1.5], c=0.

d- Effect of a&b, a=[3 4 5], b=[3 4 5], c=0.

2.4.5. Sigmoid Membership Function

A sigmoid membership function is defined by two parameters; a and c, where, a determines the slope at the crossover point x=c.

$$\text{sigmoid}(x,a,c)= \frac{1}{1+e^{\left[-a(x-c)\right]}}$$

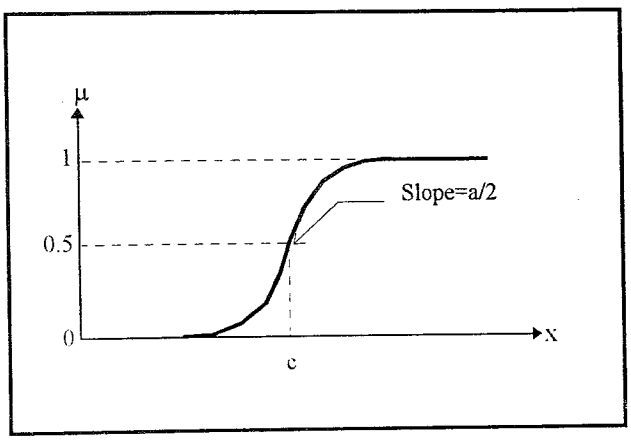


FIGURE 2.14. The meaning of the parameters defining a Sigmoid membership function.

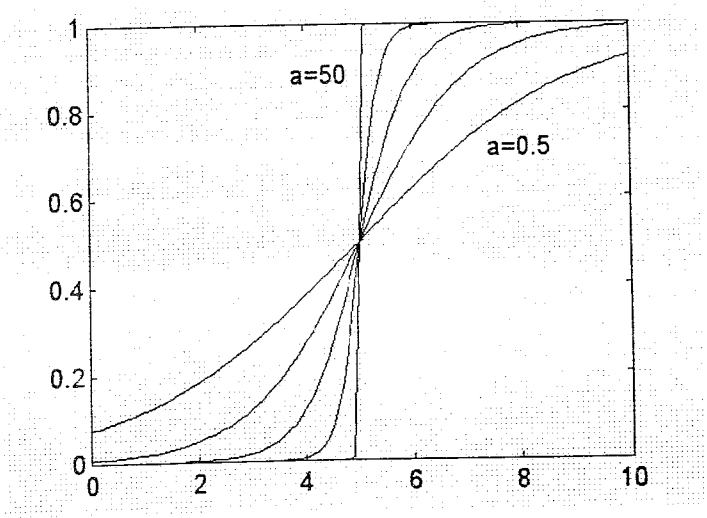


FIGURE 2.15. The Sigmoid membership function with c=5 and a=0.5, 1, 2, 5, 50.

Even though many additional membership functions can be described for use in fuzzy logic applications, the above mentioned are the most common ones.

For fuzzy control applications, where adaptation is necessary, the membership function used should have a continuous derivative. Therefore, triangular and trapezoidal membership functions are not suitable for such applications. However, they are simple and easy to use.

There is also the possibility to generate new membership functions based on the above mentioned membership functions by multiplication and subtraction to fulfill the need for non-symmetric, continuous and differentiable membership functions.

TABLE 2.1. Fuzzy membership functions defined in terms of a look-up table.

	Negative	Near Zero	Positive
$x \leq -12$	0.0	0.0	0.0
$-12 < x \leq -9$	0.3	0.0	0.0
$-9 < x \leq -6$	0.8	0.0	0.0
$-6 < x \leq -3$	0.7	0.2	0.0
$-3 < x \leq 0$	0.2	0.7	0.0
$0 < x \leq 3$	0.0	0.8	0.1
$3 < x \leq 6$	0.0	0.2	0.7
$6 < x \leq 9$	0.0	0.0	0.7
$9 < x < 12$	0.0	0.0	0.3
$12 \leq x$	0.0	0.0	0.0

Besides the definitions of membership functions that have been introduced so far, "quantized" membership functions in terms of look up tables are also used. Since this way of defining membership functions is not intensively employed throughout this study, it will not be given in detail. If the Figure 2.16. and Table 2.1. are analyzed, the concept will be clear. In the figure, a set of membership functions are given. The table gives a look up table which is very similar to the figure. Therefore, it is to be noted that membership functions can also be represented by tables.

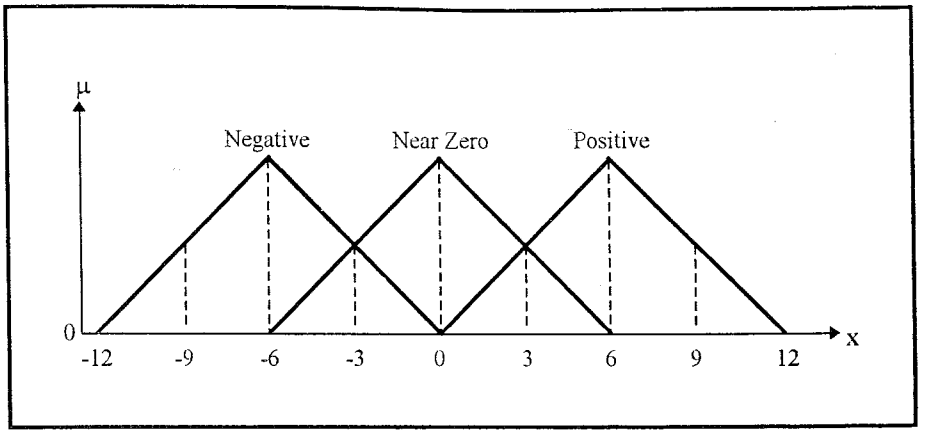


FIGURE 2.16. A set of membership functions.

2.5. Set-Theoretic Operations

Some operations on fuzzy sets are similar to operations on crisp sets and some are not, but they obey to most of the properties of crisp sets. Therefore, a brief summary of crisp set operations and properties will be given to form a basis for the introduction of fuzzy set operations and properties. The basic set-theoretic operations on crisp sets are the well known operations; union, intersection and complement.

Let A and B two crisp sets on the discourse X. Then basic set operations can be defined as,

$$A \cup B = \{x | x \in A \text{ or } x \in B\} \qquad \text{Union}$$

$$A \cap B = \{x | x \in A \text{ and } x \in B\} \qquad \text{Intersection}$$

$$\overline{A} = \{x | x \notin A \text{ and } x \in X\} \qquad \text{Complement}$$

The properties of these operations on crisp sets are defined as follows.

$\overline{\overline{A}} = A$	Involution
$A \cup B = B \cup A$	Commutativity
$A \cap B = B \cap A$	
$A \cup (B \cup C) = (A \cup B) \cup C$	Associativity
$A \cap (B \cap C) = (A \cap B) \cap C$	
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	Distributivity
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	
$A \cup A = A$	Idempotency
$A \cap A = A$	
$A \cup \varnothing = A$	Identity
$A \cap X = A$	
$A \cap \varnothing = \varnothing$	
$A \cup X = X$	
$A \cap \overline{A} = \varnothing$	Law of Contradiction
$A \cup \overline{A} = X$	Law of Excluded Middle
$\overline{A \cap B} = \overline{A} \cup \overline{B}$ $\overline{A \cup B} = \overline{A} \cap \overline{B}$	De Morgan's Laws

2.6. Operations on Fuzzy Sets

The basic set theoretic operations are defined by L. A. Zadeh's seminal paper [1]. However, besides these many other set theoretic operations are defined on fuzzy sets.

Especially, to allow differentiation for reasons such as adaptation, parameterized operations are defined on fuzzy sets which will also be given briefly.

The basic definitions of union and complement are given by L. A. Zadeh as:

Let $\tilde{A}$ and $\tilde{B}$ be two fuzzy sets defined on the discourse X.

$$\mu_{\tilde{A}\cup\tilde{B}}(x) = \max\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} \qquad \text{Union (disjunction)}$$

$$\mu_{\tilde{A}\cap\tilde{B}}(x) = \min\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\} \qquad \text{Intersection (conjunction)}$$

$$\mu_{\bar{\tilde{A}}}(x) = 1 - \mu_{\tilde{A}}(x) \qquad \text{Complement (negation)}$$

Indeed, these definitions are intuitive and seem similar to the crisp operations. It is natural to have "max" operator for union since the bigger number somewhat covers the smaller one, and similarly, the smaller number have the part that is common to both of them which results in the "min" operation for intersection.

By these definitions, it can be shown that, they have the properties which are defined for the crisp sets except the laws of contradiction and excluded middle.

Unless the membership values are either zero or one, fuzzy operations do not obey the laws of contradiction and excluded middle. For the fuzzy case, assume that $\tilde{A}$ is a fuzzy set and x is a point on the boundary of the set $\tilde{A}$. Then,

$$\mu_{\tilde{A}\cap\bar{\tilde{A}}}(x) = \min\{\mu_{\tilde{A}}(x), \mu_{\bar{\tilde{A}}}(x)\} \neq 0$$

$$\mu_{\tilde{A}\cup\bar{\tilde{A}}}(x) = \max\{\mu_{\tilde{A}}(x), \mu_{\bar{\tilde{A}}}(x)\} \neq 1$$

It is obvious that these two laws will not hold on the boundaries of fuzzy sets.

Besides the L. A. Zadeh's definitions, there are other approaches to define these set operations. Since there are several operators proposed for intersection and union, the fuzzy

intersection and union operators are referred as "T-norms" (triangular norms) and "T-conorms" (S norms) respectively.

Let $\tilde{A}$ and $\tilde{B}$ be two fuzzy sets defined on the discourse X.

Then, T and S operators can be shown as,

$$\mu_{\tilde{A}\cap\tilde{B}}(x) = T(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \mu_{\tilde{A}}(x) \tilde{*} \mu_{\tilde{B}}(x)$$

$$\mu_{\tilde{A}\cup\tilde{B}}(x) = S(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \mu_{\tilde{A}}(x) \tilde{+} \mu_{\tilde{B}}(x)$$

where, $\tilde{*}$ and $\tilde{+}$ are binary operators for the function T and S respectively.

T and S are two, two-place functions "T(.,.)" and "S(.,.)" mapping from [0,1]x[0,1] to [0,1], having the properties of monotonicity, commutativity, associativity and satisfying the boundary conditions given in the following section.

2.6.1. The Basic Properties of T and S Norms

Boundary Conditions:

$$T(0,0)=0, \quad T(a,1)=T(1,a)=a$$

$$S(1,1)=1, \quad S(0,a)=S(a,0)=a$$

Monotonicity:

$$T(a,b) \leq T(c,d)$$

$$S(a,b) \leq S(c,d) \text{ if } a \leq c \text{ and } b \leq d$$

Commutativity:

$$T(a,b)=T(b,a)$$

$$S(a,b)=S(b,a)$$

Associativity:

$$T(a,T(b,c))=T(T(a,b),c)$$

$$S(a,S(b,c))=S(S(a,b),c)$$

There are several T and S norms satisfying these conditions and here the most commonly cited four will be given in the Table 2.2.

TABLE 2.2. The basic T-norms and corresponding S-norms.

T NORMS	S NORMS
1- Minimum $T(a,b)=\min(a,b)$	Maximum $S(a,b)=\max(a,b)$
2- Algebraic Product $T(a,b)= axb$	Algebraic Sum $S(a,b)=a+b-axb$
3- Bounded Product $T(a,b)=\max(0, a+b-1)$	Bounded Sum $S(a,b)=\min(1, a+b)$
4- Drastic Product $T(a,b)=\begin{cases} a, & \text{if } b=1 \\ b, & \text{if } a=1 \\ 0, & \text{if } a,b<1 \end{cases}$	Drastic Sum $S(a,b)=\begin{cases} a, & \text{if } b=0 \\ b, & \text{if } a=0 \\ 1, & \text{if } a,b>0 \end{cases}$

Indeed, these two norms are **duals** which support the generalization of De Morgan's Law. However, first consider the complement operator N as the third operator. As it is the case for union and intersection the definition of the complement operator is not unique.

N is a one-place function N(.) mapping from [0,1] to [0,1] having the monotonicity property and similarly, satisfying some boundary conditions.

2.6.2. The Basic Properties of the Operator N

Boundary Conditions:

$$N(0)=1, \quad N(1)=0$$

Monotonicity:

$$N(a)=N(b) \qquad \text{if } a \leq b$$

The well known and the most frequently used complement operator is the one that was proposed by L. A. Zadeh in his seminal paper [1], but other complement operators are also proposed such as Sugeno's Complement [15] or Yager's Complement [16].

$$N(a)=1-a \qquad \text{Zadeh's Complement}$$

$$N(a)=\frac{1-a}{1-s.a}, \; s>-1 \qquad \text{Sugeno's Complement}$$

$$N(a)=(1-a^{w})^{1/w}, \; w>0 \qquad \text{Yager's Complement}$$

The parameters s and w used by Sugeno and Yager provides the possibility of controlling the behavior of the operator. The parameterization of operators is very helpful if the structure of the operator is to be controlled. For the sake of completeness, some parameterized T and S norms will also be presented below.

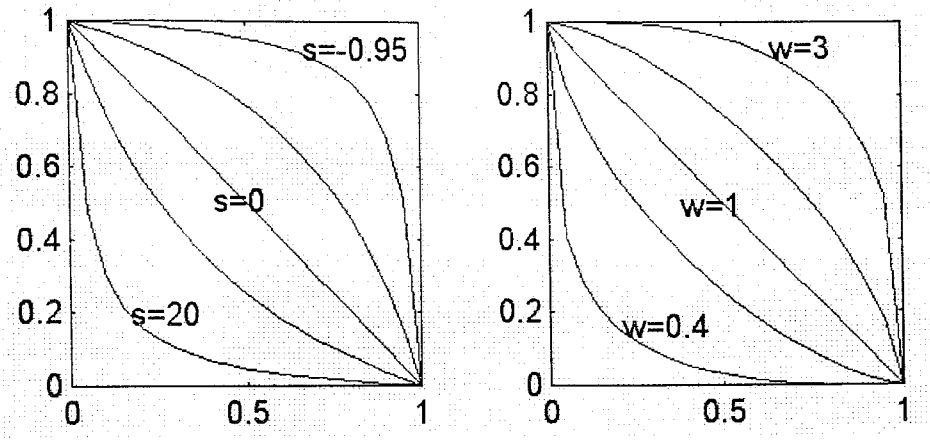


FIGURE 2.17. Sugeno's and Yager's Complement Operators with s=[20, 2, 0, -0.7, -0.95] and w=[0.4, 0.7, 1, 1.5, 3] respectively.

From Figure 2.17. it is observed that for the values of s=0 and w=1, the complements of both Sugeno and Yager are equal to the Zadeh's complement.

Remembering that T and S norms were stated as duals and supported the generalization of the De Morgan's Law, the duality can be written as,

$$T(a,b)=N(\ S(\ N(a),\ N(b)\ )\ )$$

$$S(a,b)=N(\ T(\ N(a),\ N(b)\ )\ ).$$

Therefore, for a T norm, there is always a corresponding S norm and vice versa. As an example, consider the algebraic product.

$$T(a,b) = a \times b$$

The dual operator algebraic sum can be obtained as,

$$S(a,b) = N(\ T(\ N(a),\ N(b)\ )\ )$$

where,

$$T(a,b) = a \text{x} b$$

$$T(\,N(a),\, N(b)\,) = (1\text{-}a)(1\text{-}b) = 1\text{-}a\text{-}b + a \text{x} b$$

$$N(T(\,N(a),\, N(b)\,)\,) = a + b - a \text{x} b$$

$$S(a,b) = a + b - a \text{x} b$$

TABLE 2.3. Parameterized T-norms and corresponding S-norms. [12]

Proposer	T and S norms
Yager for q>0	$T_Y(a,b,q) = 1 - \min\left\{1, \left[(1-a)^q + (1-b)^q\right]^{1/q}\right\}$ $S_Y(a,b,q) = \min\left\{1, (a^q + b^q)^{1/q}\right\}$
Dubois and Prade for $\alpha \in [0,1]$	$T_{DP}(a,b,\alpha) = ab / \max\{a,b,\alpha\}$ $S_{DP}(a,b,\alpha) = \dfrac{a+b-ab-\min\{a,b,(1-\alpha)\}}{\max\{1-a,1-b,\alpha\}}$
Hamacher for $\gamma > 0$	$T_H(a,b,\gamma) = \dfrac{ab}{\gamma + (1-\gamma)(a+b-ab)}$ $S_H(a,b,\gamma) = \dfrac{a+b+(\gamma-2)ab}{1+(\gamma-1)ab}$
Frank for s>0	$T_F(a,b,s) = \log_s\left[1 + (s^a - 1)(s^b - 1)/(s-1)\right]$ $S_F(a,b,s) = 1 - \log_s\left[1 + (s^{1-a} - 1)(s^{1-b} - 1)/(s-1)\right]$
Sugeno for $\lambda \geq -1$	$T_S(a,b,\lambda) = \max\{0, (\lambda+1)(a+b-1) - \lambda ab\}$ $S_S(a,b,\lambda) = \min\{1, a+b-\lambda ab\}$
Dombi for $\lambda > 0$	$T_D(a,b,\lambda) = \dfrac{1}{1+\left[(\frac{1}{a}-1)^{\lambda} + (\frac{1}{b}-1)^{\lambda}\right]^{1/\lambda}}$ $S_D(a,b,\lambda) = \dfrac{1}{1+\left[(\frac{1}{a}-1)^{-\lambda} + (\frac{1}{b}-1)^{-\lambda}\right]^{-1/\lambda}}$

As stated before, the parameterized T and S norms will not be analyzed in detail, but are introduced in Table. 2.3 to relay an idea to their forms.

3. INTRODUCTION TO FUZZY CONTROL

Over the years since its first introduction, fuzzy logic has found a very wide range of application areas. In this section, the use of fuzzy logic in control applications will be briefly introduced. Since, this thesis is intended to serve as an introductory material on soft computing techniques, and since the concepts of "fuzzy logic" and "control" are wide areas of research, the material will be introduced in a way simple enough to serve as an introduction, but, in detail enough to explain what has been done within the framework of this thesis.

Fuzzy logic provides us the power of using words in place of numbers, which enables us to embed our qualitative experience about a control system, to the quantitative controller.

The main idea relying at the back of fuzzy logic control (FLC) is very well explained by Kickert and Mamdani as:

> *"The basic idea behind this approach was to incorporate the "experience" of a human process operator in the design of controller. From the set of linguistic rules which describe the operator's control strategy a control algorithm is constructed where the words are defined as fuzzy sets. The main advantage of this approach seem to be the possibility of implementing "rule of thumb" experience, intuition, heuristics and the fact that it does not need a model of the process."* [17]

Very intensive studies on fuzzy logic with the above mentioned purpose in mind have given rise to numerous engineering oriented approaches where the goal is to tune a controller until a sufficient behavior is achieved, regardless of whether there is any human like operation or not. [3]

Consider a very simple example shown in Figure 3.1., i.e. the problem of positioning the car near the "stop" sign. In other words consider a linear position servo problem. Since

every driver has some experience on how to accomplish such a task, one can easily put forward some linguistic rules similar to the following ones:

Rule Base 01:

- if **sign** is *far ahead* then **continue with the same velocity**
- if **sign** is *close ahead* then **slow down**
- if **sign** is *nearby* then **stop the car**
- if **sign** is *close behind* then **slow down**
- if **sign** is *far behind* then **continue with the same velocity**

It is needless to say that the motion is assumed to be always towards the sign.

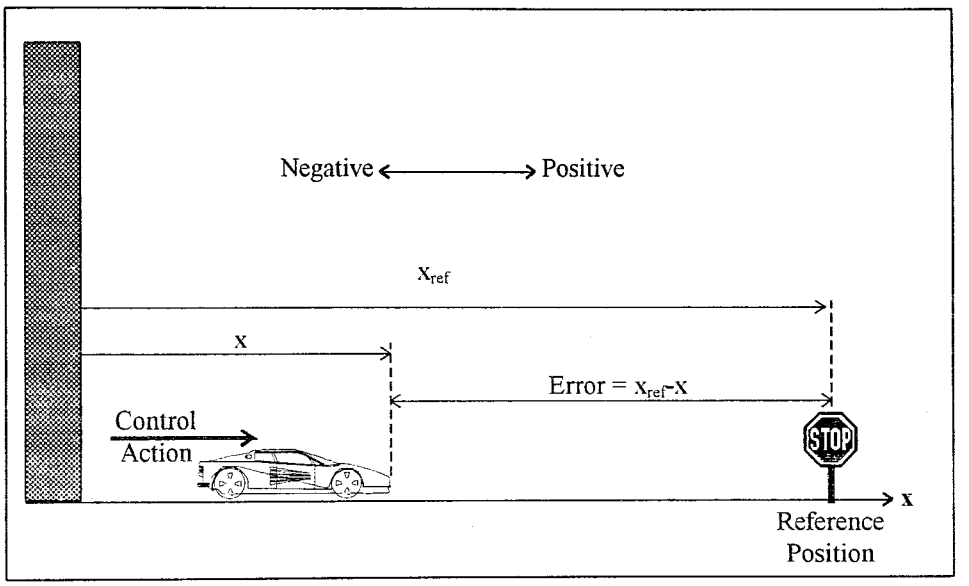


FIGURE 3.1. Linear position servo problem.

These rules in the Rule Base 01 can be converted to the following rules which governs the position servo.

Rule Base 02:

- if **error** is *positive large* then **control action** is *positive large*
- if **error** is *positive small* then **control action** is *positive small*
- if **error** is *near zero* then **control action** is *near zero*
- if **error** is *negative small* then **control action** is *negative small*
- if **error** is *negative large* then **control action** is *negative large*

These five rules are sufficient to position a linear servo system and fuzzy logic provides us the tools to convert these rules to quantitative control actions. Without naming it to be a fuzzy logic control, we perform very precise and accurate controls in our daily life. What fuzzy logic does is, to enable the person to convert this experience to a set of control rules which then can be used in control systems. However, it is to be noted that converting rules to numbers is not a very easy process. Some very basic methods have been proposed (intuition, inference, experience etc.), but, they are generally subjective and therefore, they are all subject to error. The intelligence of the human allows her/him to put forward some rules to control a system and if s/he is experienced enough on that system the content of the qualitative control may be suitable. However, in converting these qualitative information to quantitative control actions, one has to be careful. What is produced in this way may somehow work, but if some strict performance criteria has to be achieved or if the stability has to be guaranteed, then a finer tuned rule base is required. Therefore, adaptive techniques and learning methods are introduced to fuzzy logic control systems. Both approaches will be introduced below by means of simple examples. In the first example, a classical fuzzy control problem namely, the inverted pendulum problem will be stated and the solution will be presented by using classical fuzzy logic approach which incorporates no adaptation. Then, in the second example, another very classical problem of sine wave learning will be introduced to demonstrate what adaptation means. Finally, a linear position servo problem will be introduced where an adaptive fuzzy (*fuzzy-neuro*) system will is used as a controller and further readings will be recommended to reinforce the concepts. However, before proceeding to these examples, some basic information will be given about fuzzy control.

3.1. Basics of Fuzzy Logic Control

A fuzzy control system can be divided into the main sub groups that are shown in the Figure 3.2. Even though, different scientists give different definitions to explain the structure of a fuzzy logic controller, the one given below is extracted from many similar definitions.

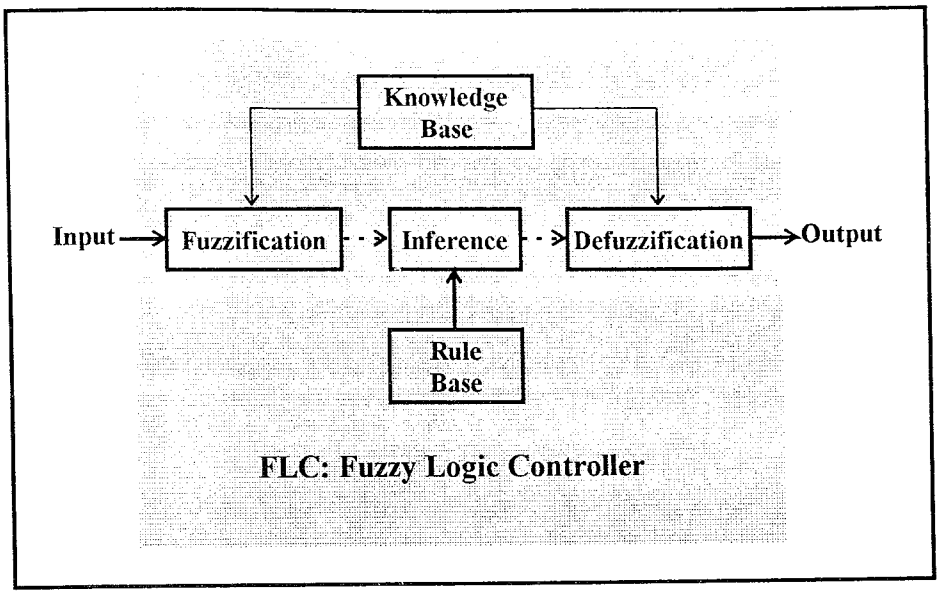


FIGURE 3.2. Schematic drawing of fuzzy logic controller. Dotted lines refer to the parts where the data flow is fuzzy.

The main parts that build a fuzzy logic control system are:

- Fuzzification
- Knowledge Base
- Inference
- Rule Base
- Defuzzification

In finding the fuzzy control output corresponding to an input to the fuzzy logic controller, the input passes through the stages of fuzzification, inference and defuzzification. This is a crisp mapping where fuzziness is present in between. Knowledge base and rule base are used in parallel to fuzzification, defuzzification and inference stages respectively. Each of these five stages are explained below briefly.

3.1.1. Fuzzification

Consider the first rule in the Rule Base 01 about stopping a car near the stop sign.

- if **sign** is *far ahead* then **continue with the same velocity** *(towards the sign)*

The term, *far ahead* is a **fuzzy term**. However, the distance to the sign is a crisp number with an exact value. Beyond that, *far ahead* is a **subjective measure**. Therefore, the first thing that should be done is to convert this crisp distance value into a fuzzy number which will be used in the next step of inference. This process is called **fuzzification** and the fuzzification is performed by relying on the information that is available in the knowledge base.

3.1.2. Knowledge Base

This is the part that provides the necessary information on how fuzzification and defuzzification processes are to be performed. In other words the membership function definitions are stored at the knowledge base.

3.1.3. Rule Base

The rule base contains the rules which form the basis of decisions to be taken in a fuzzy system. The rules may be based on personal experience, intuition, inference, neural networks, genetic algorithms, some empirical results or on any information that will be helpful in defining the desired behavior. [11]

A rule is composed of two main parts: antecedent (*premise*) and consequent (*consequence*).

If (antecedent) then (consequent)

Again consider the above mentioned rule.

- if **sign** is *far ahead* then **continue with the same velocity** *(towards the sign)*
  antecedent (*premise*) consequent (*consequence*)

Both the antecedent and the consequent parts have single arguments in this simple example, but, they do not have to be so. A rule may be written with multiple arguments in its antecedent and consequent. These types of rules are employed in dealing with MIMO systems and the rules as given in the previous example are used in SISO systems.

Now, consider the rule given below for a MISO system.

- if **sign** is *far ahead* and **the car** is *slow* then **increase velocity** *(towards the sign)*

Here, two arguments are present in the antecedent and they are associated by the connective "and". How each word within a rule is evaluated will be presented in the below example. It is to be noted here that, the connective "and" corresponds to a T-norm which have been explained in the previous chapter.

3.1.4. Inference

This part is the core of decision where the rules in the rule base are evaluated to produce fuzzy decisions. During this process each rule is evaluated separately and then a decision is made for each individual rule. The result is a set of fuzzy decisions. For example, again consider the above example. The inference may result in decisions such as: Increase the velocity slightly, keep the same velocity, etc.

3.1.5. Defuzzification

In this part, the meanings of the linguistic decisions, such as "increase the velocity slightly" are evaluated and these fuzzy decisions are aggregated to a single crisp value

(which is a necessity in most real time applications). More specifically, one cannot ask for a slight increase in the control voltage, but one can demand a 5 volt increase.

Summarizing what has already been stated, a fuzzy logic system receives crisp inputs, these crisp inputs are then fuzzified and, based on the rule base, they are evaluated. A resulting decision depending on the input variables are inferred after which the fuzzy decision is converted to a crisp value. This procedure is clarified in the following examples.

3.1.6. Example: Fuzzy Logic Control of Inverted Pendulum

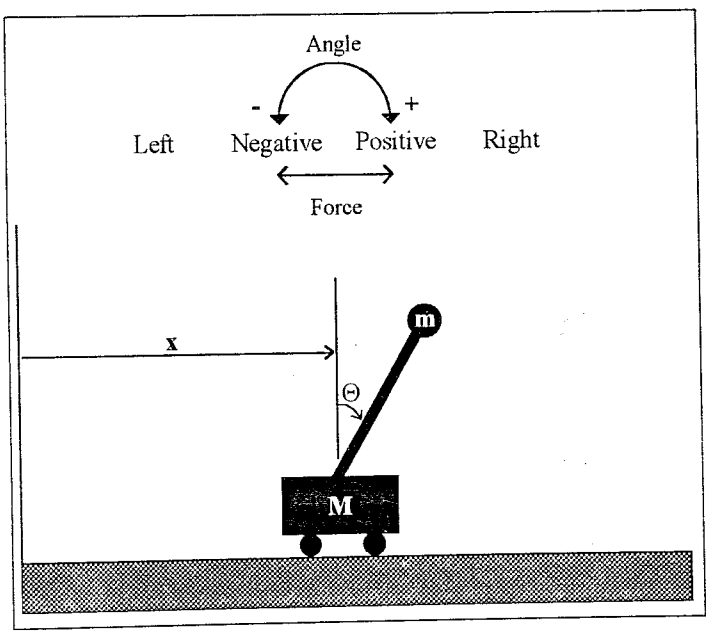


FIGURE 3.3. The inverted pendulum.

The mathematical model of an inverted pendulum is given in Section 5. It is unstable and is not easy to control. However, one can propose some rules on how an inverted pendulum can be balanced similar to the way a child would try to balance a stick on a fingertip.. The following are the rules proposed for this purpose:

Rule Base 03:

1. if **pole angle** is *positive* and **pole** *leans to right* then strongly push **the cart** *right*

2.	if	**pole angle** is	*positive*	and	**pole**	*does not move*	then	push **the cart** *right*
3.	if	**pole angle** is	*positive*	and	**pole**	*leans to left*	then	do not push **the cart**
4.	if	**pole angle** is	*near zero*	and	**pole**	*leans to right*	then	slightly push **the cart** *right*
5.	if	**pole angle** is	*near zero*	and	**pole**	*does not move*	then	do not push **the cart**
6.	if	**pole angle** is	*near zero*	and	**pole**	*leans to left*	then	slightly push **the cart** *left*
7.	if	**pole angle** is	*negative*	and	**pole**	*leans to right*	then	do not push **the cart**
8.	if	**pole angle** is	*negative*	and	**pole**	*does not move*	then	push **the cart** *left*
9.	if	**pole angle** is	*negative*	and	**pole**	*leans to left*	then	strongly push **the cart** *left*

Note: *lean to right* means that the pendulum has a positive angular velocity whereas similarly *lean to left* means that the pendulum has a negative angular velocity.

TABLE 3.1. Tabulation of the Rule Base 03.

Angular Velocity \ Angle	n	nz	p
n	hn	sn	nz
nz	n	nz	p
p	nz	sp	hp

1. hn = **strongly push** the cart *left*
2. n = **push** the cart *left*
3. sn = **slightly push** the cart *left*
4. nz = **do not push** the cart
5. sp = **slightly push** the cart *right*
6. p = **push** the cart *right*
7. hp = **strongly push** the cart *right*

refer to Rule Base 03

This rule base definition is enough to balance a pole provided that the knowledge base is appropriate. Since the rule base is generally constructed depending on intuition or experience, its construction is somehow easy, however, as mentioned before, the knowledge base assignment is the most difficult part of the process. In this thesis the knowledge bases are generally formed based on the data collected from the system. The free response and some critical forced responses of the system are observed and the meanings of adjectives (i.e., large, small, etc.) are then fixed. This approach has revealed satisfactory results. The word "satisfactory" should be emphasized here, since some post tuning have been necessary for better performance, however, the rule base extracted from experience or intuition and the knowledge base constructed from the observed data have generally resulted in very satisfactory controls.

The set of rules given above can be stated in the form of a table. This is generally the preferred way of stating a rule base since it is easier to tabulate and it is more compact as shown in Table 3.1. Also, it is easier to write the software code by looking at the table rather than reading many rules.

Here, a basic concept called "completeness" should be stated. Completeness provides the fuzzy logic controller to infer a proper control action for any state of the plant. More specifically, at any instant, the fuzzy logic controller should activate some rules and in no way fuzzy logic controller will face a situation where the current states corresponds to no rules.

At this stage, the problem is defined, which is the balancing of the pole on a cart; the strategy to fulfill this task is intuitively put forward, i.e. the rule base. The remaining steps will be traced numerically for better understanding. However, to avoid becoming highly case dependent, general information will also be provided while progressing through the steps of this specific example.

The control system can schematically be shown in Figure 3.4.

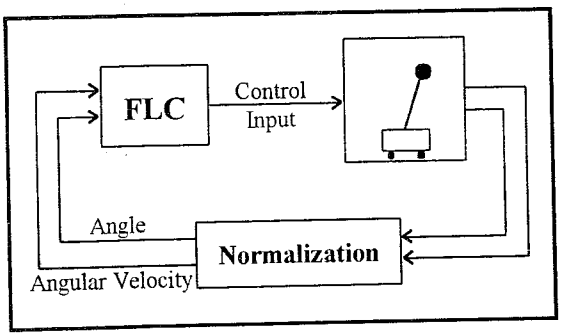


FIGURE 3.4. The inverted pendulum control system.

In many cases, normalization of the measured values are essential. This is in most cases helpful, since the measurements are in terms of volts and the variables that are stated in the rule base are distance, angle, velocity etc. Therefore, normalization provides sensible membership value assignments to the rules. However, if the knowledge base will be determined by observation as mentioned before, normalization is not so important although it will help to produce data in a desired interval.

Since balancing the pole means zero angle and zero angular velocity, the plant output is not compared with a reference and is directly fed to the controller.

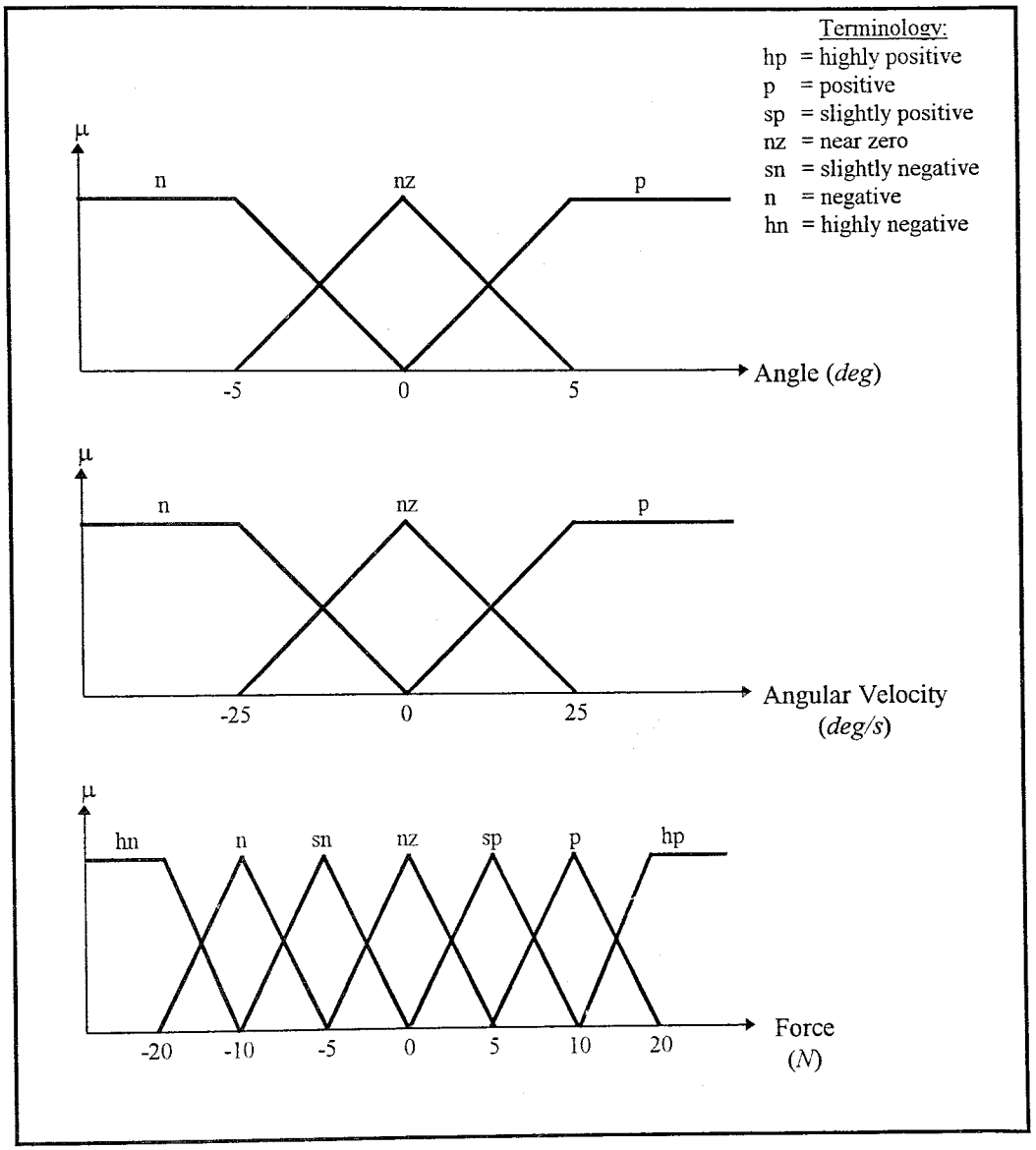


FIGURE 3.5. Input output membership functions that forms the knowledge base.

The next step is the definition of the knowledge base upon which fuzzification and defuzzification processes will be based. The membership functions defined for the two system outputs are shown in Figure 3.5. They are constructed based on the data gathered from the free response of the pendulum.

Suppose that, at the instant k, the values read as the angle and angular velocity are θ= 1.5 deg. and ω= -20 deg/s respectively.

It is here assumed that the voltage readings are somehow converted to angle and angular velocity and we will use these values which are physically more sensible, during the decision process.

The next step is to go through the fuzzy logic controller with the two states and see what happens in detail.

The membership values corresponding to θ= 1.5 deg. and ω=-20 are shown in Figure 3.6. As seen, and as it have been expected, not all of the membership functions have some values other than zero. This will cause only the firing of some rules in the rule base. As a result of the "**fuzzification**" process it is revealed in the fuzzy logic controller that the angle is near zero by 0.7, positive by 0.3, negative by 0.0 and the angular velocity is negative by 0.8, near zero by 0.2 and positive by 0.0. This is the end of fuzzification stage.

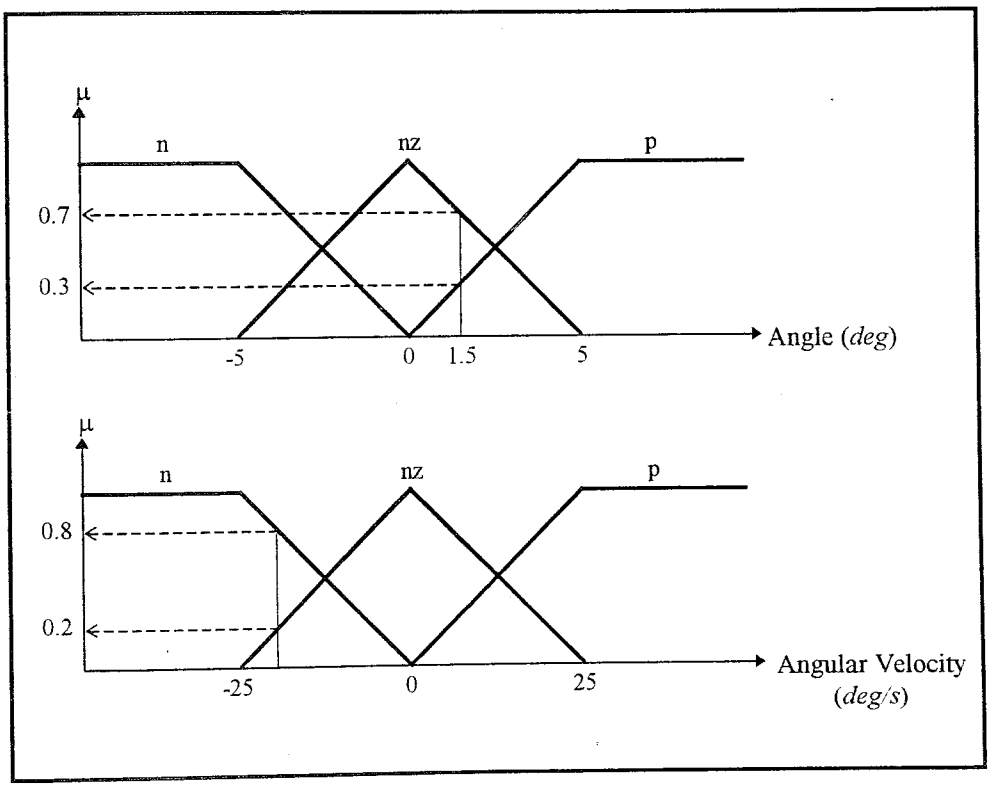


FIGURE 3.6. Fuzzification of pendulum angle and angular velocity.

At the stage of "**inference**" output of each rule will be evaluated. Since the antecedents have two arguments first of all the connective "and" should be evaluated. "and" is perceived as a T-norm and any of the T-norms mentioned in the previous chapter can be used. The T-

norms are given below. Again it is to be remembered that these are some very basic T-norms and various other T-norms are suggested in the literature.

TABLE 3.2. Basic non-parametric T-norms.

T NORMS	
1- Minimum	$T(a,b)=\min(a,b)$
2- Algebraic Product	$T(a,b)= axb$
3- Bounded Product	$T(a,b)=\max(0, a+b-1)$
4- Drastic Product	$T(a,b)=\begin{cases} a, & \text{if } b=1 \\ b, & \text{if } a=1 \\ 0, & \text{if } a,b<1 \end{cases}$

When these T-norms are considered, T(a,0)=0 is obvious.

Consider the first rule for example:

1. if **pole angle** is *positive* and **pole** *leans to right* then strongly push **the cart** *right*
(0.3) (0.0)

It is trivial that the antecedent of the first rule will result in zero (T(0.3, 0.0)=0.0) and this rule will automatically become irrelevant for the current measurement.

Therefore, among the nine rules stated in the rule base five of them drop out and the remaining four are the ones that will determine the fuzzy logic controller output at the instant ***k***. Below are these four rules:

2. if **pole angle** is *positive* and **pole** *does not move* then push **the cart** *right*
(0.3) (0.2) = 0.06

3. if **pole angle** is *positive* and **pole** *leans to left* then do not push **the cart**
(0.3) (0.8) = 0.24

5. if **pole angle** is *near zero* and **pole** *does not move* then do not push **the cart**
(0.7) (0.2) = 0.14

6. if **pole angle** is *near zero* and **pole** *leans to left* then slightly push **the cart** *left*
(0.7) (0.8) = 0.56

In this example, "Algebraic Product" is selected as the T-norm. Selection of the norm is not based on any rule or heuristic, any of the other norms could have been selected as well. The results of the T-norm operation is also shown above. These are the results of each rule. These values in a way represent *how much each rule will be effective* in the final decision that will be done by the fuzzy logic controller at the defuzzification stage. They are called the "**firing strengths**" of the rules. The reflections of each of these fuzzy decisions on the output variable "Force" are also shown in Figure 3.7. As can be seen, each decision covers an area proportional to its membership value and these areas very well illustrates how much each rule will effect the final decision. This is the end of "**inference**" stage. Therefore, it is seen that the dominant rule at the output will be the sixth one.

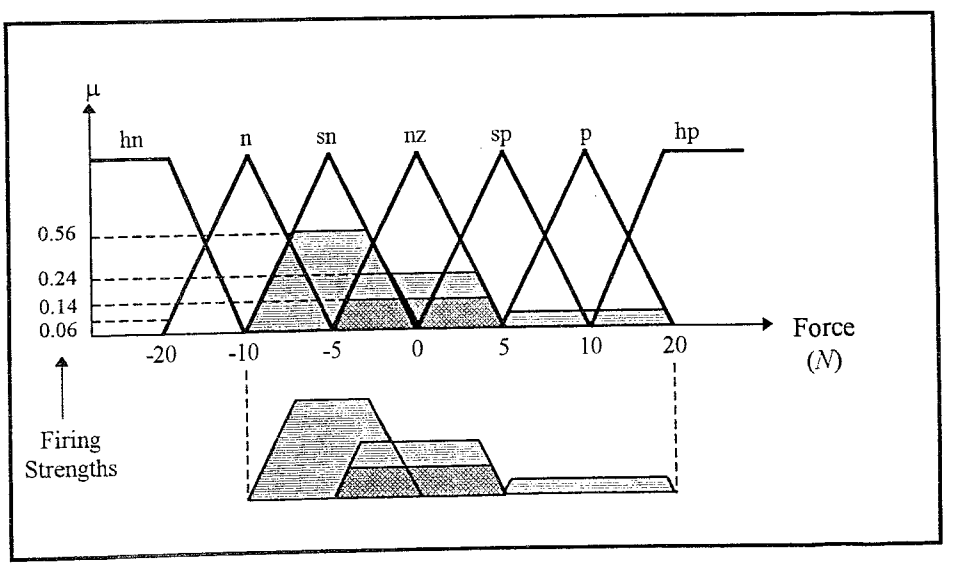


FIGURE 3.7. The inference stage.

The areas representing the firing strengths of each rule do not have to be reflected as trapezoidal as shown in Figure 3.7. They can very well be represented with triangles as shown Figure 3.8. or by any other method which will reflect the firing strengths to the defuzzification stage. An alternative is to use the firing strength itself. This is very useful when the computational burden is considered. In this example, the trapezoidal representation is preferred for illustration.

These preferences cause slight changes in the value of the crisp decision, however, since the aim of this thesis is different, their effects will not be discussed. However, it is to be noted that, every stage (fuzzification, inference and defuzzification) has many alternative routes which may lead to different results.

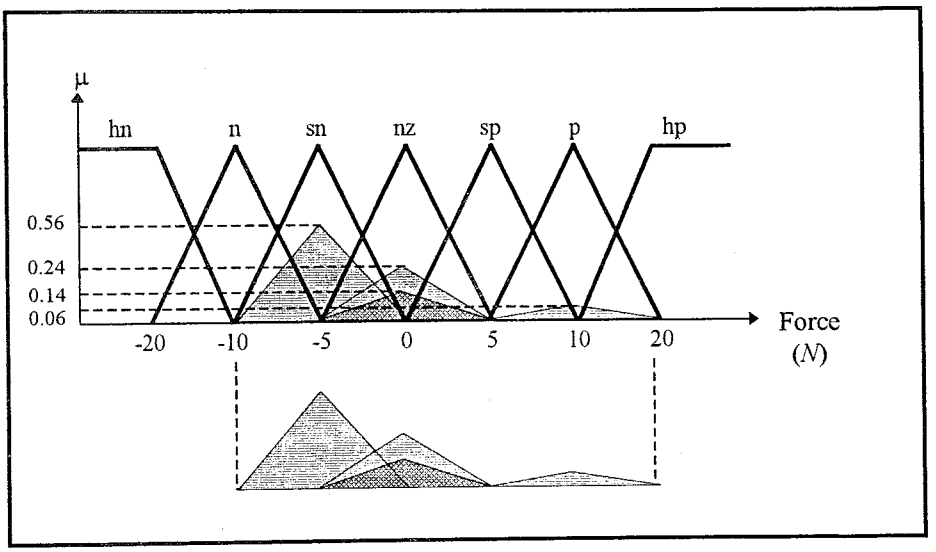


FIGURE 3.8. An alternative inference mechanism.

The following are the very basic and the most frequent methods used in aggregating the individual rule outputs produced at the inference stage.

Consider the trapezoids obtained as the fuzzy results of the rules shown in Figure 3.7. These decisions are in the form of "do not push the cart" or "push the cart left". These partial decisions have to be aggregated into a single crisp value in order it to be applicable to the control system as a control input.

Some possible ways of aggregating the rule outputs to a single value are stated below. It is once again to be remarked that there are many other ways suggested in the literature.

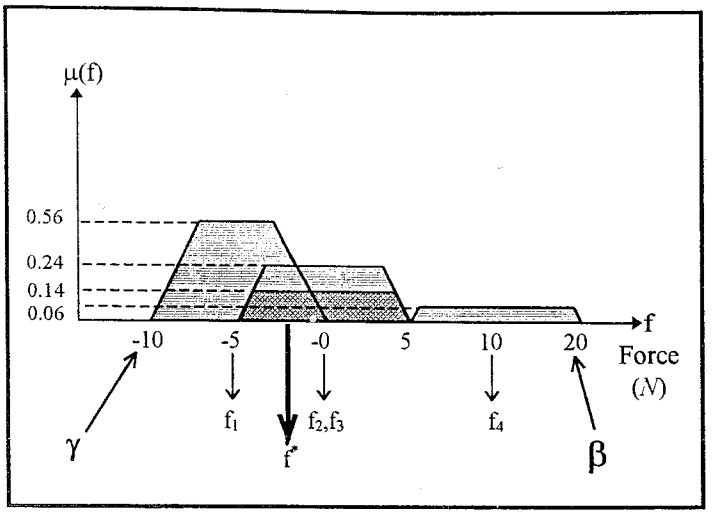


FIGURE 3.9. Individual decisions produced at the inference stage (f_1, f_2, f_3, f_4) are aggregated to f^*.

Centroid Method: The main idea behind this is to aggregate the decision at the center of gravity of the resulting shapes produced by inference. Formally, this can be stated as,

$$f^* = \frac{\int_{\gamma}^{\beta} \mu(f).f.df}{\int_{\gamma}^{\beta} \mu(f).df}$$

where γ is the lower and β is the upper bound of the shape in Figure 3.9..

It is to be noted that calculating these integrals are not very easy since the function defining the resulting shape is not given. This is computationally cumbersome and this way of calculating the center of gravity gives the center of the overall shape. However, it is obvious that there may be shapes that overlap and the actual center of gravity of the overlapping pieces is somewhere else. This is also the case in our example and Figure 3.10. illustrates this fact.

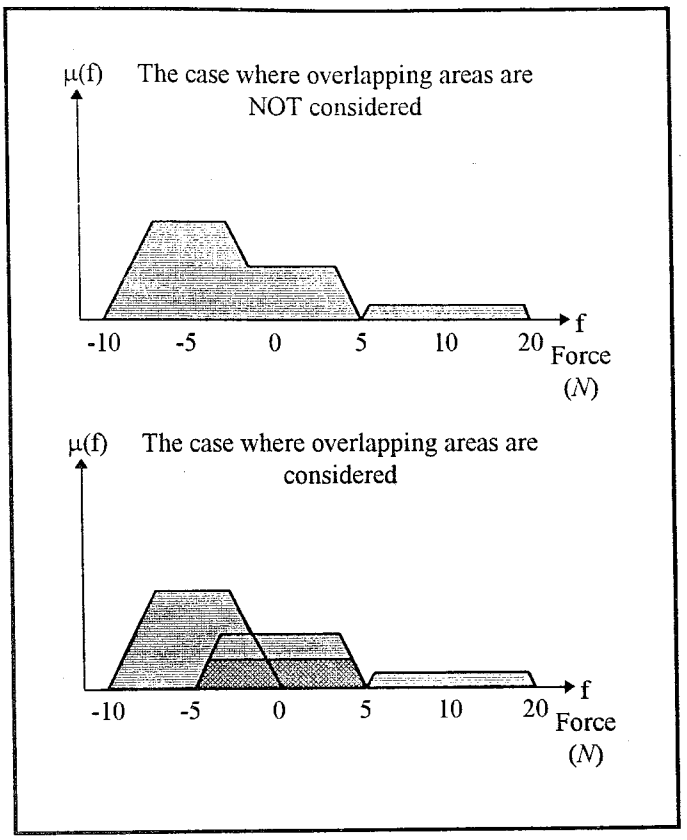


FIGURE 3.10. Two different approaches in centroid method.

A consideration of these figures reveals that computing the centroid of the first shape is much harder than the second one. The second case is indeed very easy. The area A of a trapezoid can be calculated by using its base "b" and the height "h" of it as follows:

$$A = (1 - \frac{h}{2y}).h.b\,.$$

The Figure 3.11. shows the details on how the area is calculated. Generally the membership functions used are "normal" and therefore, y=1 and the above equation reduces to:

$$A = (1 - \frac{h}{2}).h.b\,.$$

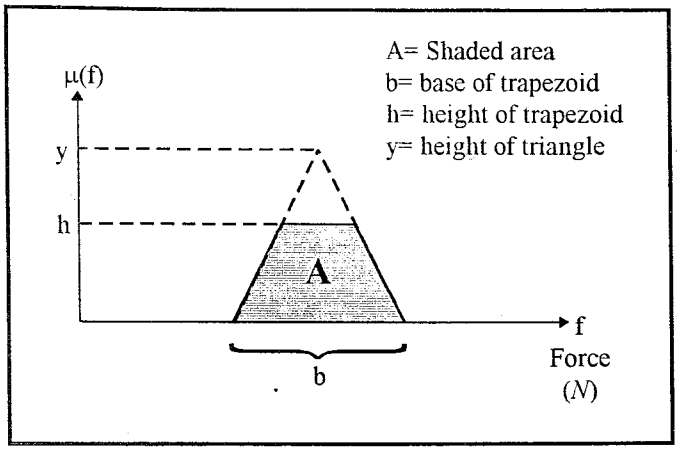


FIGURE 3.11. Parameters used in calculating the area of a trapezoid.

Suppose that A_i represent the area values for the fired rule I and f_i represents the center of this trapezoid, then the center of mass can be calculated as:

$$f^* = \frac{\sum_{i=1}^{R} A_i . f_i}{\sum_{i=1}^{R} A_i}$$

where R is the number of fired rules.

Bisector Method: This method divides the area into two equal areas and the decision is the point of separation. In other words f^* is chosen in so as to satisfy,

$$\int_{\gamma}^{f^*} \mu(f).df = \int_{f^*}^{\beta} \mu(f).df .$$

Smallest Of Maximum: The smallest number corresponding to the maximum value of the resultant membership function is selected as the crisp output.

Largest Of Maximum: Similarly, this method aggregates the decision to the largest value corresponding to the largest membership value.

Mean Of Maximum: Here the crisp value is calculated as the mean of the above two methods.

Weighted Average: This is the easiest and therefore, most applicable method. In this method, the firing strengths are used as the areas of individual trapezoids (or whatever shape have been produced i.e. triangles) and the resulting equilibrium point is calculated with respect to the center of output membership values.

The results of these methods are shown in Figure 3.12.

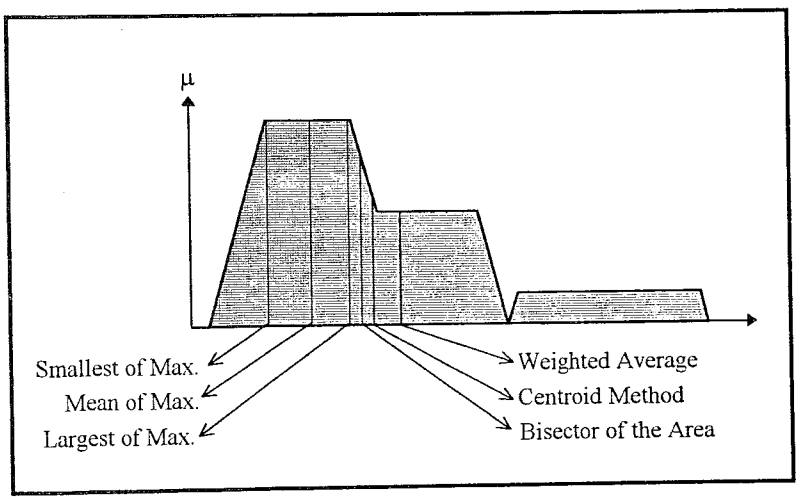


FIGURE 3.12. The results of different defuzzification methods.

In our specific example, the weighted average method have been used and therefore, the result can be calculated as:

$$f^* = \frac{(-5*0.56)+(10*0.06)}{0.06+0.14+0.24+0.56} = -2.2$$

In summary, a crisp set of input going into the fuzzy logic controller, is first converted to fuzzy values (fuzzification) and then these fuzzified values are fed to the rules to produce fuzzy results (inference). The individual results are then aggregated into a single crisp value (defuzzification) which completes a single pass through the fuzzy logic controller.

In our specific example, the input of (angle, angular velocity)=[1.5, -20] is fed the controller and the result is produced as [-3.4]. This control input will be fed to the plant and in the next sampling a new couple of inputs will pass through the same procedure to produce the corresponding control input.

4. ADAPTIVE FUZZY (FUZZY-NEURO) CONTROL

4.1. Introduction to Adaptive Fuzzy Systems

It has previously been stated a number of times that adapting the parameters (i.e. membership functions, T-norms) of a fuzzy system is very helpful if what can be done by intuition or by similar conventions is not satisfactory.

Many different methods for adaptive fuzzy systems have been proposed in literature. In the content of this text only one of them will be explained in detail.

Before going into the details of adaptation methodology, the parameters that are suitable for adaptation should be examined. If the general structure of a fuzzy system that have been explained in the previous sections is considered, these parameters become apparent. Let us once again consider the Figure 3.2.

The knowledge base provides the information about the membership functions and therefore, contains some material subject to adaptation. The rule base is similar in this respect. Since if the performance is poor, new rules may well improve it. However, this is a complicated task and not adopted frequently in control applications. The other candidate is the inference procedure. In this stage the antecedents of the rules are evaluated. The particular T-norms selected for this purpose is important since different choices produce different results. Therefore, adaptation can be incorporated at this stage too. Another set of adaptive parameters beyond these ones have been introduced by Sugeno [18].

In his approach Sugeno defines a fuzzy rule as,

$$\text{If } x_1 \text{ is } A_1 \text{ and } \ldots \text{ and } x_k \text{ is } A_k \text{ then } y=p_0+p_1x_1+ \ldots +p_kx_k.$$

The output of the rules are polynomials. The parameters of polynomials are updated by using least squares. Since the problem of complex adaptation is turned into a linear least square estimation this method appears to be very attractive but the complexity of the solution increases exponentially, since inversion of matrices are involved in the solution. The details of the method will not be given but [13], [18] and [19] are recommended for further reading.

The adaptive fuzzy control methodology is given in Figure 4.1. The input vector to the fuzzy network is $\mathbf{x}=(x_1, x_2, \ldots x_n)$. The rules are shown with dotted boxes and their outputs are represented with Z^i. At the defuzzification stage the rule outputs are multiplied with the rule weights $\bar{y}^i$ where i=1,2, ..., M.

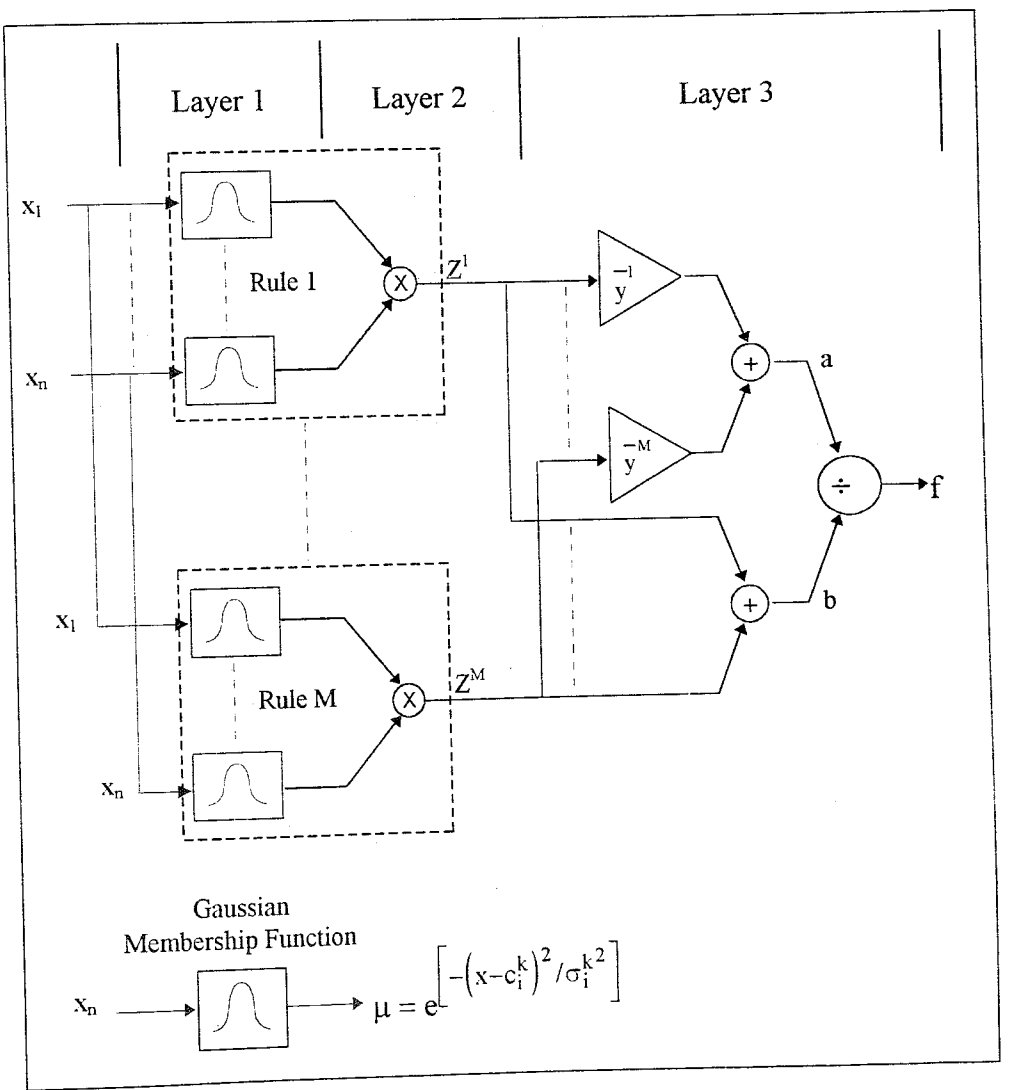


FIGURE 4.1. Three layer adaptive fuzzy network.

This system may be called a three layered feed-forward fuzzy network. This is indeed in no way different than the fuzzy logic controllers defined in the previous section. The membership functions are chosen to be Gaussians, multiplication is selected as the T-norm and the weighted average is used at the defuzzification stage. This representation is divided into three layers as shown in the figure. At the first layer the input vector is fuzzified, namely this is the fuzzification level. The consequences of individual rules are calculated at the second layer which is the inference stage. At the last layer the rule outputs (firing strengths) and the weights corresponding to these rules are multiplied and divided by the sum of firing strengths to yield the decision "f" shown in the figure. This completes the feed-forward pass through the fuzzy network and if the parameters of the system is not to be changed after this stage, the system will be similar to the ones explained before. Therefore, any fuzzy system can well be represented as shown above. The following is a formal expression of the above defined system.

$$f(\underline{x}) = \frac{\sum_{k=1}^{M} \bar{y}^k \left(\prod_{i=1}^{n} \mu_{F_i^k}(x_i) \right)}{\sum_{k=1}^{M} \left(\prod_{i=1}^{n} \mu_{F_i^k}(x_i) \right)}$$

The adaptation part is based on the very widely used concept of "gradient descent". After the feed forward pass of input data over the network is completed, the output is compared with a reference value which indeed is the expected output of this network. The difference between these two are treated as an error term and the parameters of the network are adjusted to minimize this error.

Let us now identify the parameters that can be subject to adaptation. At the first layer of fuzzification, the two parameters $[c,\sigma]$ can be adapted. The second layer has only the T-norm and since it is selected as "multiplication" nothing is subject to adaptation. If a parametric T-norm have been selected then adaptation could also have been employed here. The third and the last layer contains the centers of output membership functions, namely the vector **y**, and the value of this vector can also be adapted in time.

If the structure of the above given system is considered, each rule has "n" membership functions where n is the number of inputs. Since each membership function has two parameters [c,σ], there are 2n parameters for each rule. Assume that there are M rules in the rule base then there are 2nM parameters and if you add the M parameters of the **y** vector to this value there are (2n+1)M parameters exist in such a network structure.

Let k=1,2,…,M and i=1,2,…,n. Then, the parameter set corresponding to the i^{th} input of the k^{th} rule can be represented as $\left[c_i^k, \sigma_i^k\right]$, and y^k represents the weight corresponding to the k^{th} rule.

As mentioned let **x** represent an input vector of length n and f^d be the desired output corresponding to this input vector. When the vector $\mathbf{x}=(x_1, x_2, \ldots x_n)$ is propagated through the fuzzy network as explained before, let the output be f(**x**). then the error term which is a measure of how much the output of this network is beyond our expectations can be given as follows:

$$e = \frac{1}{2}\left[f(x) - f^d\right]^2$$

However, it is to be noted that, when error is smaller than one this way of calculation reduces the value of actual error. If there are any stopping criteria, it should be appropriately chosen.

The gradient descent method proposes the following update rule for a parameter p(t) to be adapted at the next sampling time p(t+1)as follows:

$$p(t+1) = p(t) - \alpha \frac{de}{dp}$$

where the term α is called the learning rate. This parameter determines how fast the parameter should be updated. Excessive learning rates may cause unstability, whereas, learning rates that are admissible but large, may cause oscillation about the stable parameter

and finally, small learning rates degrades the reaching time to the solution. Therefore, an optimized learning rate is important for better response.

This shows us that adaptation will employ many derivatives over the parameters that change in time. This clarifies the reason for choosing a membership function with continuous derivative (the Gaussian in this case) rather than a function whose derivative is not smooth and continuous like a triangle.

Similarly, the update rules of the system parameters can be given as follows:

$$\bar{y}^k(t+1) = \bar{y}^k(t) - \alpha \left.\frac{\partial e}{\partial \bar{y}^k}\right|_k = \bar{y}^k(t) - \alpha \frac{f - f^d}{b} z^k$$

$$\underline{c}_i^k(t+1) = \underline{c}_i^k(t) - \alpha \left.\frac{\partial e}{\partial \underline{c}_i^k}\right|_k = \underline{c}_i^k(t) - \alpha \frac{f - f^d}{b}(\bar{y}^k - f) z^k \frac{2(x_i - \bar{c}_i^k(t))}{\sigma_i^{k2}(t)}$$

$$\sigma_i^k(t+1) = \sigma_i^k(t) - \alpha \left.\frac{\partial e}{\partial \underline{x}_i^k}\right|_k = \sigma_i^k(t) - \alpha \frac{f - f^d}{b}(\bar{y}^k - f) z^k \frac{2(x_i - \bar{c}_i^k(t))^2}{\sigma_i^{k3}(t)}$$

where, i=1,2,...,n, k=1,2,...,M, x_i is the i^{th} element of the input vector **x**, f is simply f(**x**) and t denotes the discrete sampling time. The proofs of these update rules will not be given but can be found in [19 pg. 31].

So far an adaptive fuzzy logic system have been introduced. Many more such systems can be found in [19]. Now this concept will be consolidated with two examples.

4.2. Example: Sine Wave Learning

This is also a very classical problem and it will be presented here just to show the way the algorithm works.

The fuzzy network will try to adaptively generate a sine function in the interval $[0,\pi]$. For this purpose six rules are determined in the rule base. The network structure is as

explained previously in this chapter. The structure of the system is a very simple SISO system. A number goes in and the sine of it is expected to come out. The system structure and the rules are shown in Figure 4.2.

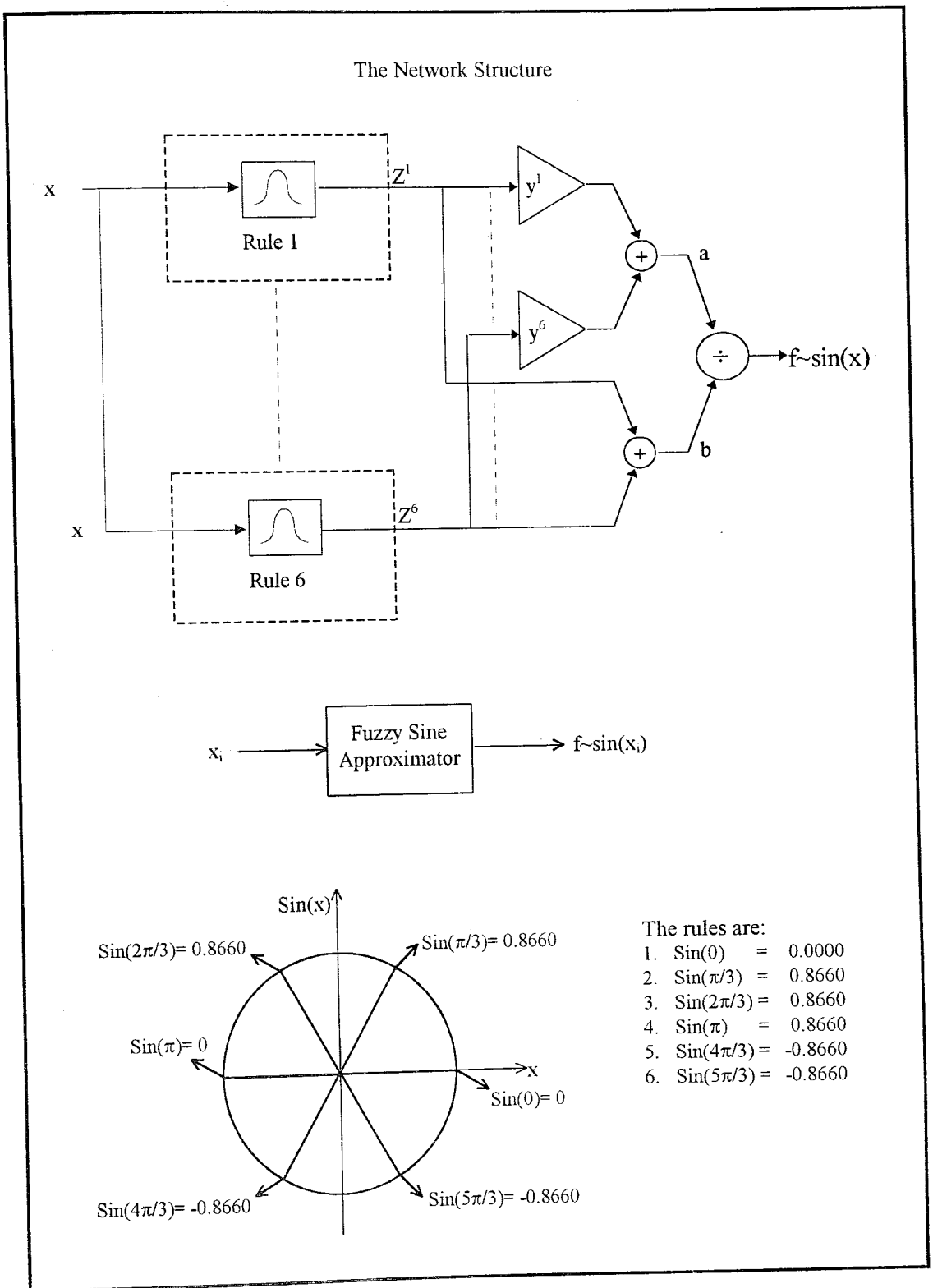


FIGURE 4.2. The network structure and the rule base used in sine wave learning.

A set of training parameters are determined to tune the system. This is given as the vector **x** where **x**={0.0, 0.2, 0.4, ... 5.8, 6.0, 6.2} with a length of 32.

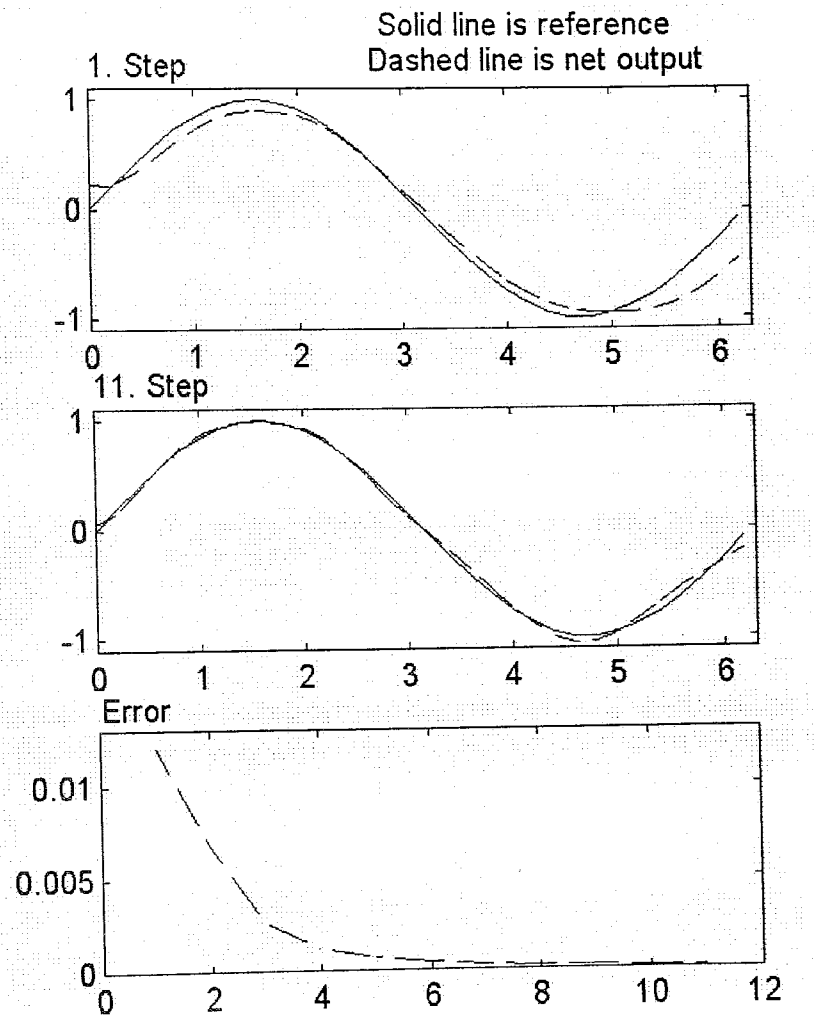


FIGURE 4.3. The performance of learning with an appropriate rule base.

The procedure is as follows: The first element of vector **x** namely 0.0 is fed to the network and the with respect to the expected value of sin(0.0) an error term is calculated. This value is back propagated and the parameters of the system is adapted in accordance. Then the same procedure is applied to the second element. This iteration continues until all 32 elements of vector **x** pass through the network. This whole pass of the training data through the network is called an "epoch". The first two parts of Figure 4.3. shows the response of the network at the beginning and after 11 epochs have been completed. Also as

a performance index the total error accumulated at each epoch is given in the last part of the same figure.

This example is given to show that the fuzzy network does not need to start from scratch. If you have some information about the system, you have the chance to impose it to the system at the very beginning, in our case these are the six rules that have been put forward. The result given above shows that even without training, the network produces reasonable results and the second part of the figure shows that, the system is adapting itself very fast even with six rules. The last part of the figure shows the change of total error.

The next example will demonstrate a control application in which adaptive fuzzy logic controller is employed to serve as an inverse plant.

4.3. Example: Control Application

In this example a simple servo plant with a second order linear model is selected. The aim is to let the output of the plant to track a reference signal. The control system is shown in Figure 4.4.

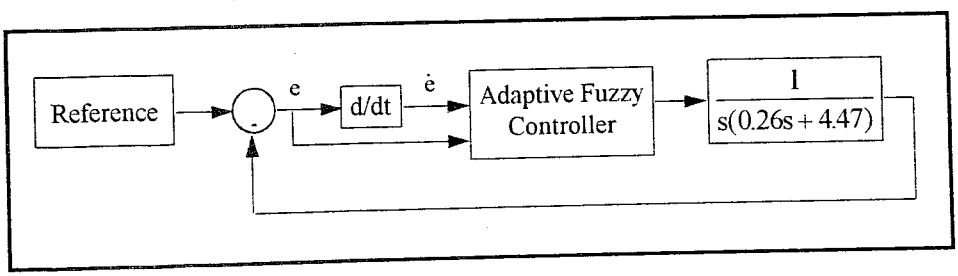


FIGURE 4.4. The control system structure.

The rule base is designed to minimize the output error and is given below. The details of the controller will not be analyzed in detail since this has been done in the previous application, but here it is aimed to present the application of this algorithm to control.

The simulation results are given below. Two classical control approaches are applied on this plant (PD and LQ controllers) and it was seen that the performance of the adaptive

fuzzy controller is much superior [20]. This study was carried out for presentation during Power Conversion Conference; PCC'97 that was held in Nagaoka, Japan during (August 3-6, 1997). The paper was accepted but later withdrawn from the program due to the difficulties met in finding support for participating in the conference. The whole paper is presented in Appendix F.

TABLE 4.1. The rule base used in linear position servo problem.

Derivative of Error \ Error	n	nz	p
n	n	sn	nz
nz	sn	nz	sp
p	nz	sp	p

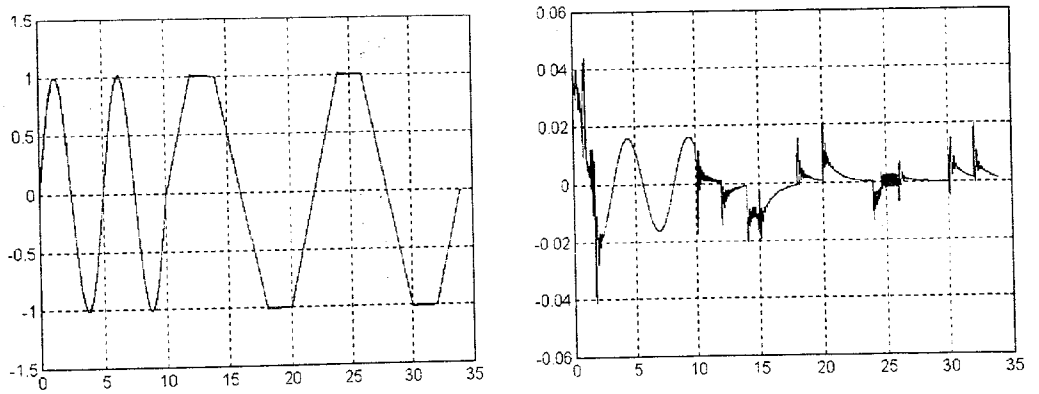


FIGURE 4.5. Tracking performance and the resulting error profile in linear position servo problem.

This completes the section on adaptive fuzzy logic controller systems. Further details can be found in the references [12], [13] and [19].

5. MODELING

In this section, the mathematical models of the plants that are in the scope of this study, namely the inverted pendulum, position servo and ball and beam modules are presented.

5.1. Modeling of Inverted Pendulum

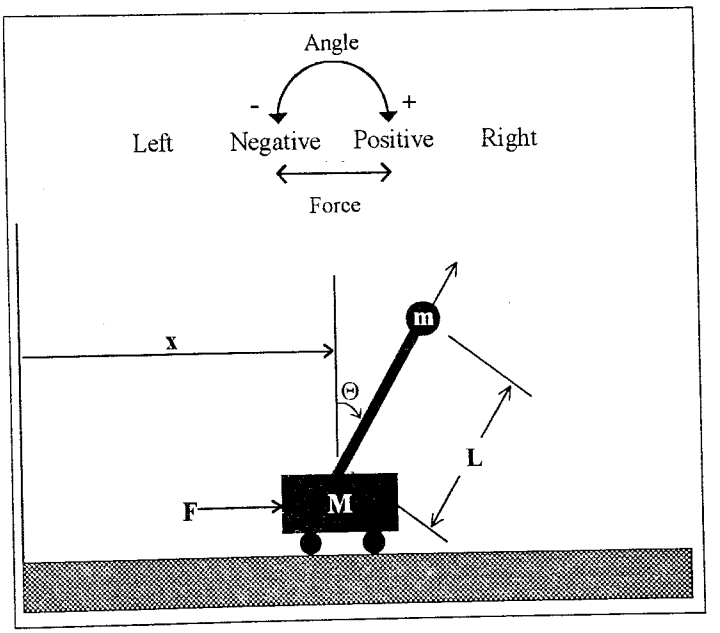


FIGURE 5.1. Inverted Pendulum.

From [21, pg. 51] the modeling of inverted pendulum shown in Figure 5.1. can be obtained as follows:

$$x_G = x + L.\sin\theta$$

$$y_G = L.\cos\theta .$$

where (x_G, y_G) is the Cartesian coordinate of the mass **m** at the end of the pole with a length of **L**. The inclination of the pole is θ.

Applying Newton's second law to the x direction of the motion,

$$M\frac{d^2x}{dt^2} + m\frac{d^2x_G}{dt^2} = F$$

or

$$M\frac{d^2x}{dt^2} + m\frac{d^2}{dt^2}(x + L.\sin\theta) = F \qquad (5.1)$$

Noting that,

$$\frac{d}{dt}\sin\theta = (\cos\theta)\dot{\theta}, \qquad \frac{d^2}{dt^2}\sin\theta = -(\sin\theta)\dot{\theta}^2 + (\cos\theta)\ddot{\theta},$$

$$\frac{d}{dt}\cos\theta = -(\sin\theta)\dot{\theta}, \qquad \frac{d^2}{dt^2}\cos\theta = -(\cos\theta)\dot{\theta}^2 - (\sin\theta)\ddot{\theta}.$$

Eqn. (5.1) can be written as

$$(M+m)\ddot{x} - mL(\sin\theta)\dot{\theta}^2 + mL(\cos\theta)\ddot{\theta} = F$$

or,

$$F = (M+m)\ddot{x} - mL[(\sin\theta)\dot{\theta}^2 - (\cos\theta)\ddot{\theta}] \qquad (5.2)$$

Applying Newton's second law to rotational motion,

$$m\frac{d^2x_G}{dt^2}L\cos\theta - m\frac{d^2y_G}{dt^2}L\sin\theta = mgL\sin\theta$$

or

$$\left[m\frac{d^2}{dt^2}(x + L\sin\theta)\right]L\cos\theta - \left[m\frac{d^2}{dt^2}(L\cos\theta)\right]L\sin\theta = mgL\sin\theta$$

$$m\left[\ddot{x} - L(\sin\theta)\dot{\theta}^2 + L(\cos\theta)\ddot{\theta}\right]L\cos\theta - m\left[-L(\cos\theta)\dot{\theta}^2 - L(\sin\theta)\ddot{\theta}\right]L\sin\theta = mgL\sin\theta$$

further simplification results in,

$$m\ddot{x}\cos\theta + mL\ddot{\theta} = mg\sin\theta \qquad (5.3)$$

Eqn.s (5.2) and (5.3) define the nonlinear motion of the pendulum. This model is an ideal model and additional modifications are case dependent. If specifically the inverted pendulum of Quanser is considered, such modifications can be done to the model.

The cart is driven by a DC motor and the equations governing dynamics of which are given below.

The relation between the angular velocity and the applied voltage is given as,

$$V = I_m R_m + K_m K_g \omega = I_m R_m + K_m K_g \frac{\dot{x}}{r}$$

which can be expressed as,

$$I_m = \frac{V}{R_m} - K_m K_g \frac{\dot{x}}{r}$$

where, I_m is motor armature current, R_m is motor armature resistance, K_m is motor constant, K_g is gear ratio, r is the radius of the gear mounted on the motor shaft and ω is angular velocity.

The torque generated at the motor shaft is governed by the equation,

$$T = K_m K_g I_m .$$

Hence,

$$F = \frac{T}{r} = \frac{K_m K_g I_m}{r} = \frac{K_m K_g}{R_m r} V - \frac{K_m^2 K_g^2}{R_m r^2} \dot{x}$$

The result is obtained when the values of the system parameters that are given by Quanser Consulting [22] in Appendix A are substituted to this equation.

$$F = 1.72V - 7.68\dot{x}$$

When this result and the actual values of the system parameters are substituted to equations 2 and 3 the following equations turn out to be the final governing equations:

$$V = 0.39\ddot{x} + 4.47\dot{x} - 0.075[(\sin\theta)\dot{\theta}^2 - (\cos\theta)\ddot{\theta}] \qquad (5.4)$$

$$\ddot{\theta} = -1.64\ddot{x}\cos\theta + 16.08\sin\theta \qquad (5.5)$$

For the sake of completeness, the linearized version of the pendulum model will also be given. The nonlinear model given by (5.2) and (5.3) can be linearized by the angular approximation,

$$\sin\theta \cong \theta,\ \cos\theta \cong 1 \text{ and } \theta.\dot{\theta} \cong 0.$$

$$(M+m)\ddot{x} + mL\ddot{\theta} = F$$

$$m\ddot{x} + mL\ddot{\theta} = mg\theta$$

The organization of these equations result in the following set of equations.

$$ML\ddot{\theta} = (M+m)g\theta - F \qquad (5.6)$$

$$M\ddot{x} = F - mg\theta \qquad (5.7)$$

The results obtained above at (5.2) and (5.3) can also be obtained by using the energy approach which incorporates Lagrangians to model the dynamics.

When the pole of the inverted pendulum is removed, the rest is a linear position servo. This system is also used in simulations and real time applications therefore, the model of it will also be presented.

5.2. Modeling of Linear Position Servo

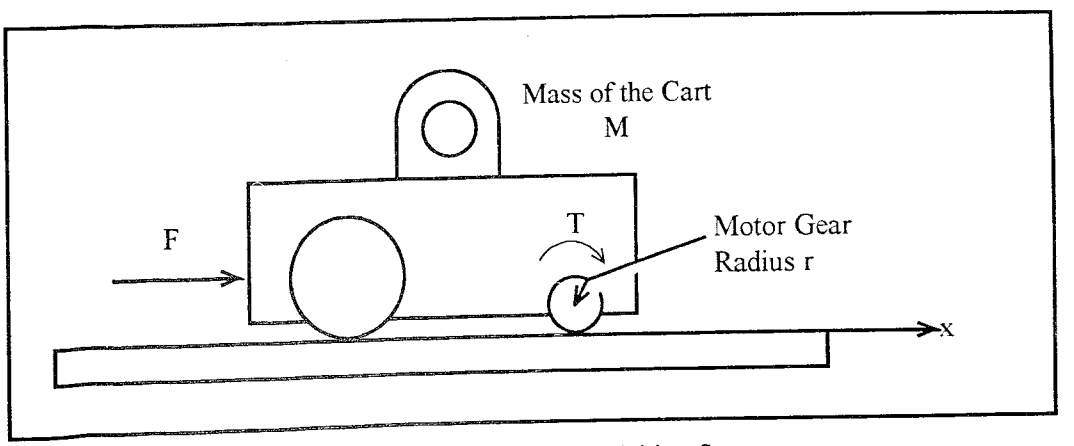


FIGURE 5.2. Linear Position Servo.

The following are the modeling of the position servo given by [22]. The schematic drawing of the position servo is shown in Figure 5.2.

Obviously,

$$F = M\ddot{x} \qquad (5.8)$$

where the system is driven by a DC motor and the equations governing the DC motor dynamics have been given above.

Since,

$$F = \frac{T}{r} = \frac{K_m K_g I_m}{r} = \frac{K_m K_g}{R_m r} V - \frac{K_m^2 K_g^2}{R_m r^2}\dot{x} \qquad (5.9)$$

using (5.8) and (5.9) the resulting dynamics can be given as follows,

$$M\ddot{x} = \frac{K_m K_g}{R_m r} V - \frac{K_m^2 K_g^2}{R_m r^2}\dot{x}.$$

This can be written in the form of a transfer function as,

$$\frac{X(s)}{V(s)} = \frac{1}{s(\frac{MR_m r}{K_m K_g}s + \frac{K_m K_g}{r})}$$

When the systems parameters are substituted the model of the position servo is obtained as,

$$\frac{X(s)}{V(s)} = \frac{1}{s(0.26s + 4.47)} \qquad (5.10)$$

5.3. Modeling of Ball and Beam

Finally, the model of the ball and the beam is given for completeness. The schematic drawing of the ball and the beam setup is illustrated in Figure 5.3.

It is obvious that,

$$F = -Mg\sin\alpha = M\ddot{x}$$

which results in an equation of motion which is independent of the ball mass M.

$$\ddot{x} = -g\sin\alpha$$

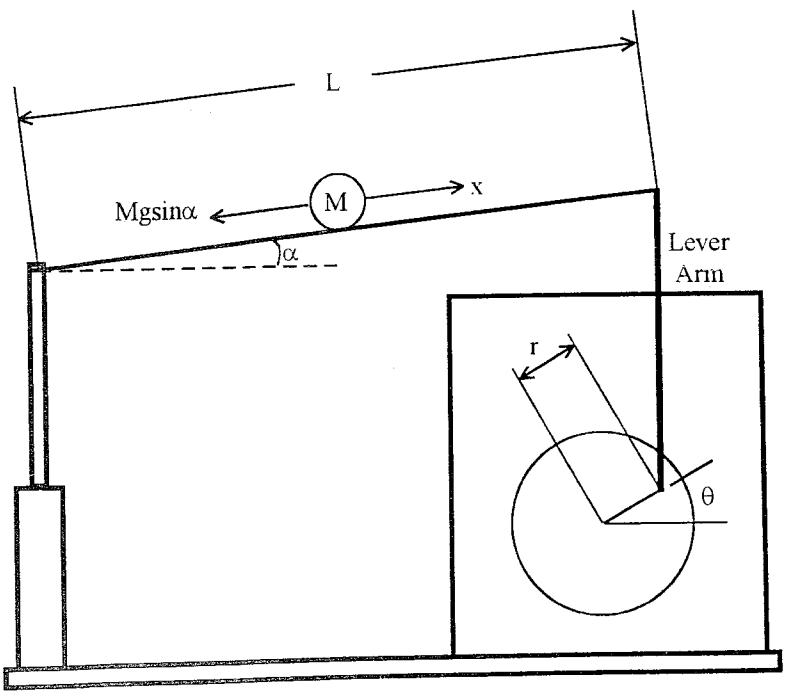


FIGURE 5.3. The ball and the beam.

The nonlinear relationship between α and θ can be linearized for r«L as,

$$\alpha = \frac{r}{L}\theta .$$

Therefore,

$$\ddot{x} = -g\sin(\frac{r}{L}\theta)$$

The ball and the beam model is used only in real-time applications, therefore, this short introduction is presumed to be sufficient..

6. THE EXPERIMENTAL ENVIRONMENT

The experimental environment that has been constructed in the scope of this project is positioned on a table under the name of "X-Garden" which stands for the "eXperimental Garden" It is intended to serve as a test bed in the control lectures to demonstrate some basic concepts in controls. This thesis is primarily devoted to the organization of this garden. It may be analyzed in two main categories; the mechanical experimental medium and the computational medium.

The experimental medium is mainly composed of the equipment bought from Quanser Consulting and an inverted pendulum which has been built in our university. Since the pendulum by Quanser is much reliable it will be kept as a permanent member of the garden and the home-made one will be taken off. Therefore, the details of this homemade setup is not explained in detail. It is to be noted however that it has a very transparent structure with a power amplifier and a connection board. The power amplifier has a booklet, which is satisfactory, and the cabling structure of the connection board is very clear and easily understood. The equipments procured from Quanser are an inverted pendulum setup, which can also serve as a linear position servo when the pole is removed, a ball and the beam setup and a power amplifier and a connection unit, which amplifies the control signal that is to be applied to the plant and also conditions the voltages that are read from the setup to make it ready for computer manipulation. The equipment has been procured without any controller cards, therefore, the first thing that has had to be done has been to establish the connection with these setups and the DSP currently present in our laboratory.

The DSP card and the peripherals form the computational medium. The computational power becomes very important if real time applications is to be performed. The computational medium is simply composed of hardware and software. The hardware part is the computer itself and the DSP card available, and the software are Matlab, Simulink, Trace and Cockpit modules and the Real-time Workshop.

The elements of the X-Garden mentioned above are explained in detail below.

6.1. The Experimental Medium

6.1.1. The Linear Position Servo

The details of the linear position servo is shown in Figure 6.1. The main parts that build the position servo are; the motor, rack and gear and the slider that provide the motion, and the potentiometer that provides the position information through the gear on the rack.

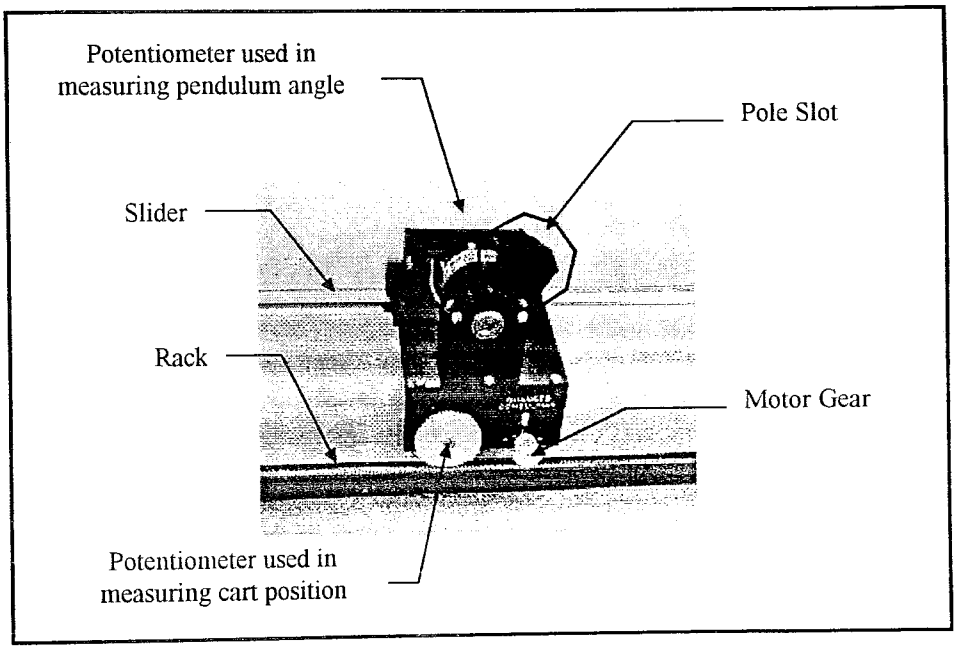


FIGURE 6.1. The Linear Position Servo and its components.

The slot to which the pendulum is inserted to convert the linear position servo to an inverted pendulum setup is indicated with a circle. The potentiometer that senses the position of the pendulum angle is also shown in the same figure. To maintain usage without problems, an initialization of the position potentiometer is vital at the beginning of each experiment, since the potentiometer gear can easily loose contact with the rack. Therefore, it does not measure an absolute value, but it is relative and has to be adjusted each and every time a new experiment starts. The details of how this adjustment is made is given in the section below together with operating principles.

6.1.2. The Inverted Pendulum

The inverted pendulum setup is simply the linear position servo with the pole placed over the cart as shown in Figure 6.2.

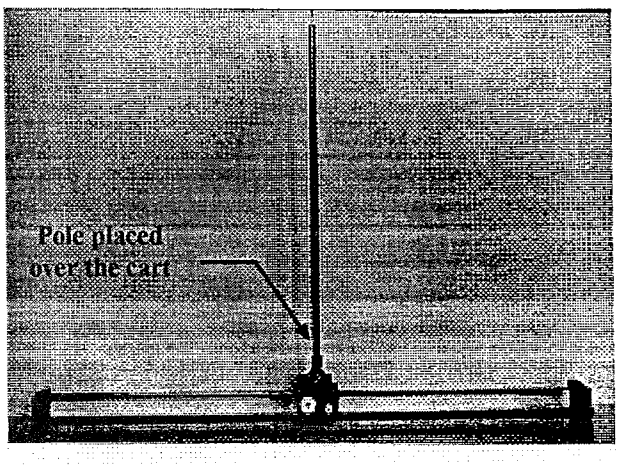


FIGURE 6.2. The inverted pendulum setup.

It is to be noted that, whenever an algorithm is to be applied to the inverted pendulum setup, the pole should be balanced and the experiment should start with a slight disturbance. Quick response of the cart to balance the pole can easily damage the gear mounted to the motor shaft, therefore, instant changes to the pole angle in terms of disturbance should be avoided.

6.1.3. The Ball and The Beam

This system, beyond being a nonlinear system, also has mechanical problems which makes experiments on this system more difficult. The first mechanical problem is a fixture problem and the second is a wind up or loosening problem. The Figure 6.3. shows the ball and the beam setup.

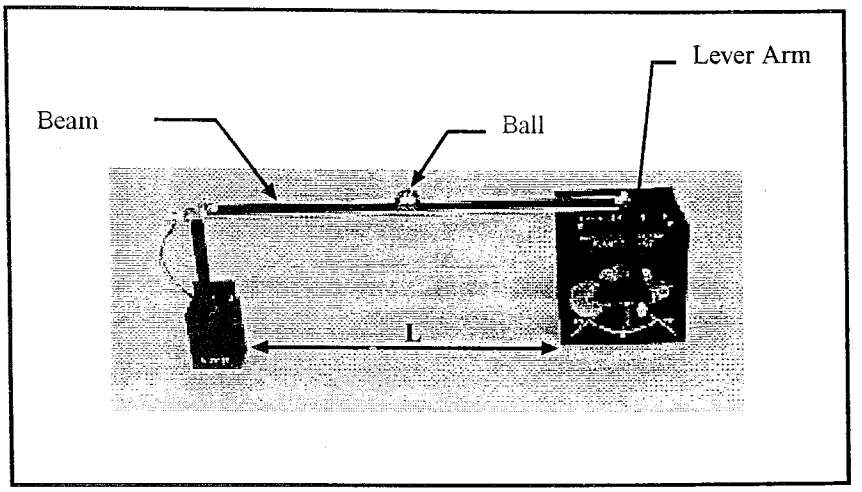


FIGURE 6.3. The ball and the beam setup.

The distance L shown in the figure is to be at an optimum value for the setup to work with satisfactory performance. In order to avoid changes in this value, it is fixed on the table.

The second problem which can be summarized as wind up or loosening problem stems from the fact that, when the disk, on which the lever arm is fixed with a bolt, makes full rotations, the bolt either winds up or gets loose. Either case is troublesome. If it winds up, the system gets locked, and if it is loosened, the bolt comes out of the pivot and the ball and the beam falls down. To avoid this problem, the disk to which the lever arm is attached should be moved around within a bounded angular region. This is provided by software and the details of this procedure will be given in the further sections.

6.1.4. The Power Amplifier Module

The power amplifier module is used both for amplifying the control signal produced at the computer and also used for transmitting the sensed values of system parameters such as ball position or pendulum angle to the computer by conditioning the signal through a Quick Connect module. In Figure 6.4. the power amplifier is shown.

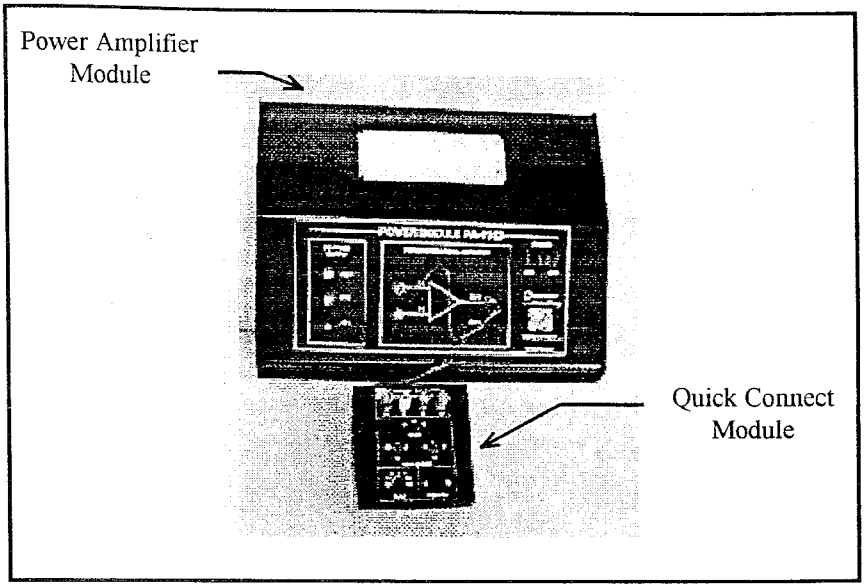


FIGURE 6.4. The power amplifier and Quick Connect Module.

6.2. The Computational Medium

6.2.1. The Hardware

The hardware is composed of the PC which is the host for the DSP that provides the interaction of the PC with its environment. The technical specifications of the hardware are given in Appendix B.

6.2.2. The Software

The software environment in developing a controller for real-time applications is somewhat complicated. The basic environment is Matlab. Simulink is a symbolic tool that

provides easy control system design. The control systems are designed with Simulink using the available blocks. The important thing to realize is that the design of a controller and application of this design to the real system are two totally separate stages.

The design of the controller is done with Simulink. When the design phase is over, this file is to be downloaded onto the DSP to run it. In this way, the algorithm runs not on the processor of the PC but on the processor of the DSP card. This is the reason why the design and application stages are said to be totally apart. While downloading the control algorithm onto the DSP, the Real Time Workshop converts the Simulink file to C code and then compiles the overall C code. This conversion process does not only produce an application file for the design but also produces many child files which provide support to the system when DSP is running. After the application file is downloaded onto the DSP, Matlab and Simulink become useless. Therefore, the blocks that are used for graphic outputs, reading the time etc., become useless. In Figure 6.5., the blocks that have no function within the real time applications are shown.

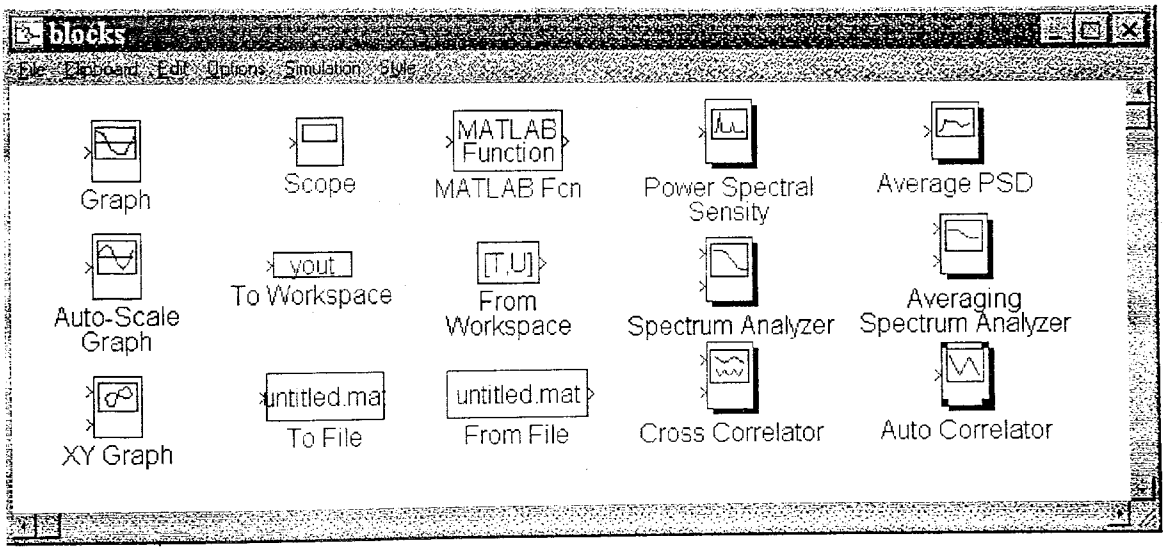


FIGURE 6.5. The non-functional blocks of Simulink when working with real-time interface.

The structure of the software hierarchy is better understood by considering Figure 6.6.

It is indicated in Figure 6.6 that Matlab and Simulink are used at the very base of the application. The design is then converted totally to C code through the real time interface. The finalized and compiled files are downloaded onto the DSP ready to run.

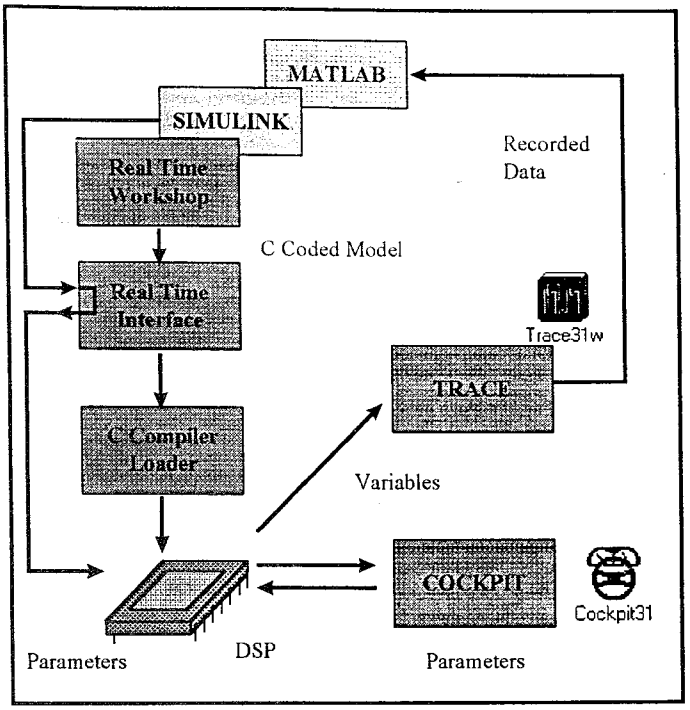


FIGURE 6.6. The schematic drawing of the combined development environment.

After this is completed successfully, the two software packages Cockpit and Trace are used in both manipulating the real time applications and observing and collecting data out of the plant while the application is running. As mentioned above, the real time environment produces many related files referred above as "child files". These files provide the information about the system parameters and other details. An important note that should be made is that, when the Simulink blocks are compiled and downloaded onto the DSP with "Generate and Built Real Time" command, all the files related to this process will be produced in the current directory of Matlab. Therefore, caution should be taken before downloading an application to avoid misplaced file accumulation. A directory is better to be dedicated for the generation of these files.In our applications the directory **"c:\x_garden\members\pendulum\dspace"** is chosen for this purpose. The reason why such a specific place is reserved can easily be understood if it is considered that each download produces more than a dozen files.

6.2.2.1. Cockpit:

Cockpit is a very easy to use program which allows both the observation and the manipulation of the parameters of the real time application even when the simulation is running. The usage of cockpit is explained in the web site of X-Garden:

http://mecha.ee.boun.edu.tr/~lab

6.2.2.2. Trace:

Trace is the module that can both plot and save the specified system parameter values in a given time interval. This tool cannot change any system parameter unlike the Cockpit module, but is very helpful, especially if the resulting plots will be exported to Matlab and other programs. When any data are saved as a ***.mat** file and loaded into the Matlab workspace by "load *.mat", the "**graphs**" command will help to plot the saved data in alternative forms. The details of using these programs are also given at the above mentioned URL.

For a better explanation of the software environment a very simple simulation in real time is described below.

6.2.3. Example: Sinusoidal Excitation of a Linear Second Order Plant

Figure 6.7. shows the simple block diagram designed in Simulink. It is intentionally chosen to be very simple. It is simply the excitation of a second order linear system with a sinusoidal input. This may be an electric circuit or a mass spring system. After the design is complete, as it is the case in the figure, then the "Real-time options" under the menu "Code" is selected. The window that appears on screen at the end of this selection is shown in Figure 6.8.

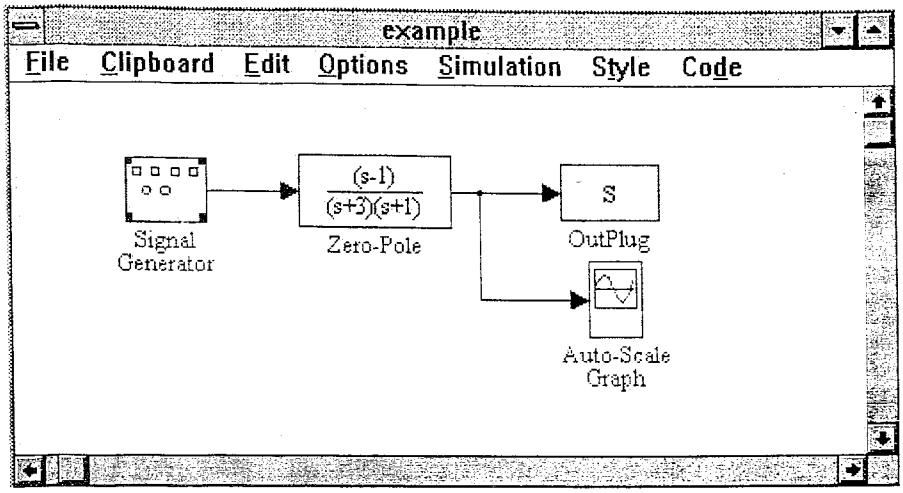


FIGURE 6.7. The design made in Simulink.

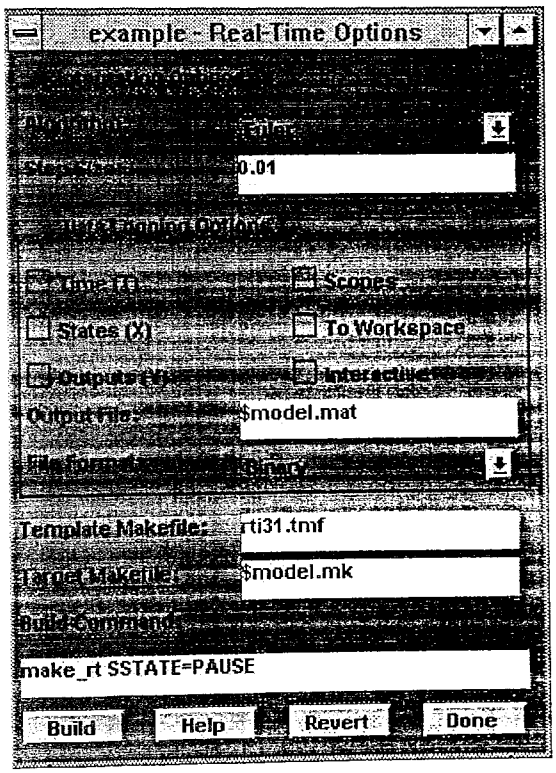


FIGURE 6.8. The real-time options window.

The parameters shown on the figure are all to be kept as they are except the value of the step size and algorithm type. The "Algorithm" is selected as Euler but other options are also available such as Runge Kutta. The sampling time is selected in the "Step Size". Down

to 0.0001 sec. have been selected before, and if the algorithm is not very complex and time consuming, such very low sampling rates can be achieved. Check boxes are non functional. "Output File", "Template Makefile" and "Target Makefile" should be as given in the Figure 6.8. The "Build Command" may also simply be written as "make_rt" but this is not very convenient. This is not preferable, since when the files are compiled and the real time application file is generated, it is immediately downloaded onto the DSP and it starts to run. The procedure of compilation can be observed on an MS DOS window, which is shown in the Figure 6.9. After downloading is completed, the "Cockpit" should be opened, which is time consuming . This gives rise to not only missing the transient behavior of the system, but also if the system to be controlled is an unstable system such as the inverted pendulum, the algorithm runs and very undesired actions may result. Therefore, "SSTATE=PAUSE" is appended to "make_rt" command which causes the simulation to start at **pause** mode. It is to be noted that before each download application the Cockpit and Trace programs should be closed.The "build" command button should then be pressed to start the download process and the steps of the operation will be printed on an MS DOS window.

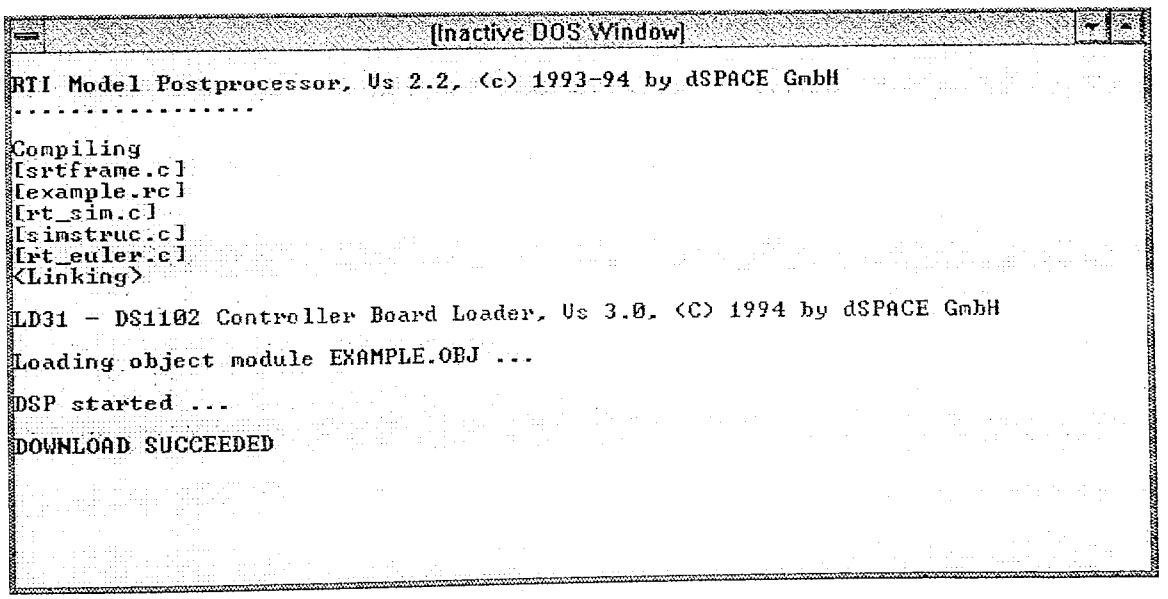


FIGURE 6.9. The MS DOS screen that shows the steps of downloading process.

The prompt "DOWNLOAD SUCCEEDED" shows that the algorithm is downloaded onto the DSP. It is the time to run Cockpit and Trace.

The name of the Simulink file that has been downloaded is **example.m**. As mentioned before, while downloading takes place, a child file called **example.trc** is produced. This file contains the data about the system parameters and is used in tracing and manipulating these parameter values. Therefore, when Cockpit or Trace is opened, the related trace (*.trc) file should be loaded. The Figure 6.10. shows the interface designed specifically for this example.

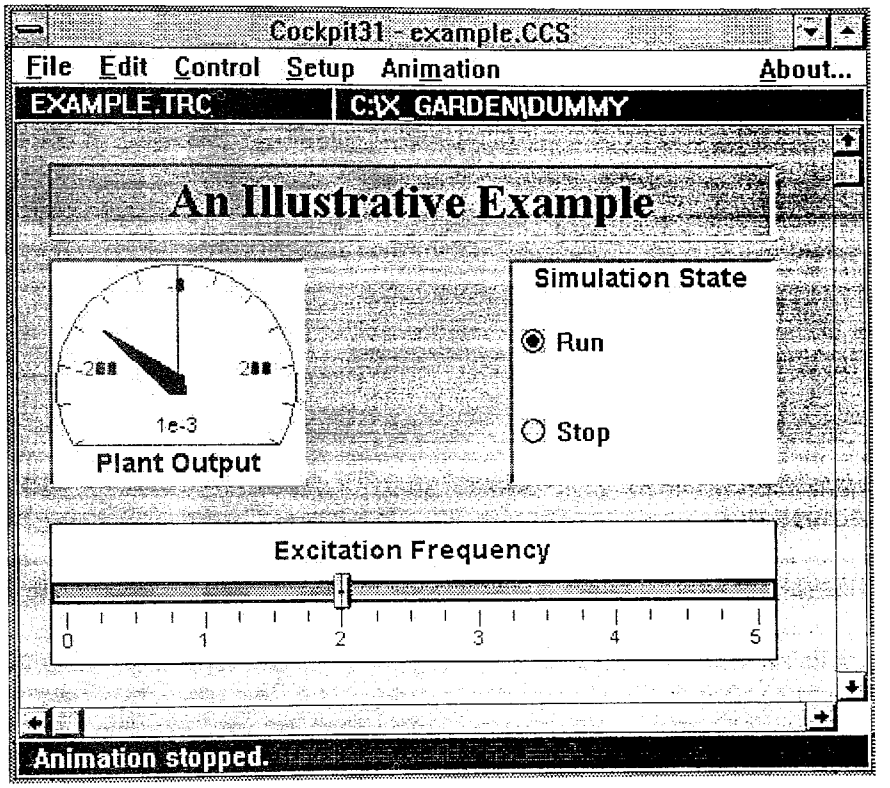


FIGURE 6.10. The Cockpit file prepared for illustration.

How such a file is designed will not be explained within the context of this thesis. When an application is downloaded, the corresponding trace file has to be loaded at first, then a control interface similar to the above given one may be designed. It is observed in this figure that both observation and manipulation can be done by using this program. The plant output is observed on a tachometer and the frequency of excitation can be changed by the use of the slider. Cockpit is very powerful in manipulating the system parameters. Any parameter that can be changed in Simulink, can also be accessed under Cockpit and changed during a real-time application.

Up to now, a brief introduction is given about the system. This much is enough to make simple designs by using available Simulink blocks. However, when complex algorithms are to be written, the lack of blocks necessitates writing down the code of that algorithm. Matlab codes are not functional under real-time. Only C codes are acceptable and they can be compiled with the rest of the blocks. The next section will explain how C codes can be embedded in design.

6.2.4. Embedding C Codes

The C codes are embedded in the S-function block that is under the Nonlinear block of Simulink.

Embedding a C code has some restrictions and these points have to be considered carefully to obtain a smooth working design. First of all the C files that will be called by the S-functions has to be under a specific directory. A group of search directories, unfortunately cannot be defined as it is done in Matlab. A definite directory is selected to locate the C source codes. This directory is:

c:\x_garden\members\s

For example, suppose that a C code named **fuzcon.c** is written and referred from an S-function. An m-file with the name **fuzcon.m** has also to be present under the same directory "c:\x_garden\members\s" with a specific format. In the absence of the accompanying m-file the download will not be completed. The C file also has to fulfill some format requirements. The formats of these files are given in the Appendix C.

This completes the explanation of the hardware and the software details of the X-Garden. In the next section, the operating principles of the X-Garden will be described in detail.

7. OPERATING PRINCIPLES

In this chapter detailed information is given on how to operate the experimental modules of the X-Garden, how to carry out simulation and real-time studies, what to do in case of difficulties and the precautions that should be taken in the operation of the equipment.

Operating the systems in the X-Garden is indeed not very complicated. If some very basic precautions are taken, no serious problems will be encountered, although this may not mean that a satisfactory performance will be obtained with the particular control algorithm that is being implemented!

The cable connections are nearly foolproof, therefore, the cabling details will not be explained. In case of any difficulties, one can refer to the URL, http://mecha.ee.boun.edu.tr/~lab.

First of all, some previous effort must have been spent on Matlab and Simulink and it is assumed that a Simulink file is ready for downloading, similar to the one illustrated in the previous section. As it have been explained in the section X-Garden, the real time application will be built by first opening the "Real-time options" and then pressing the "Build" button or by directly choosing the "Generate and Built Real-time" option from the "Code" menu.

This will open an MS DOS window on which the steps of the process will be observed. When there is a memory problem the following screens may well show up, the only valid action at this stage will be **turn**ing **off** the **computer** and then on.

It is to be noted and emphasized once again that, before starting the downloading (model post-processing) the directory of Matlab has to be changed to:

c:\x_garden\members\pendulum\dspace

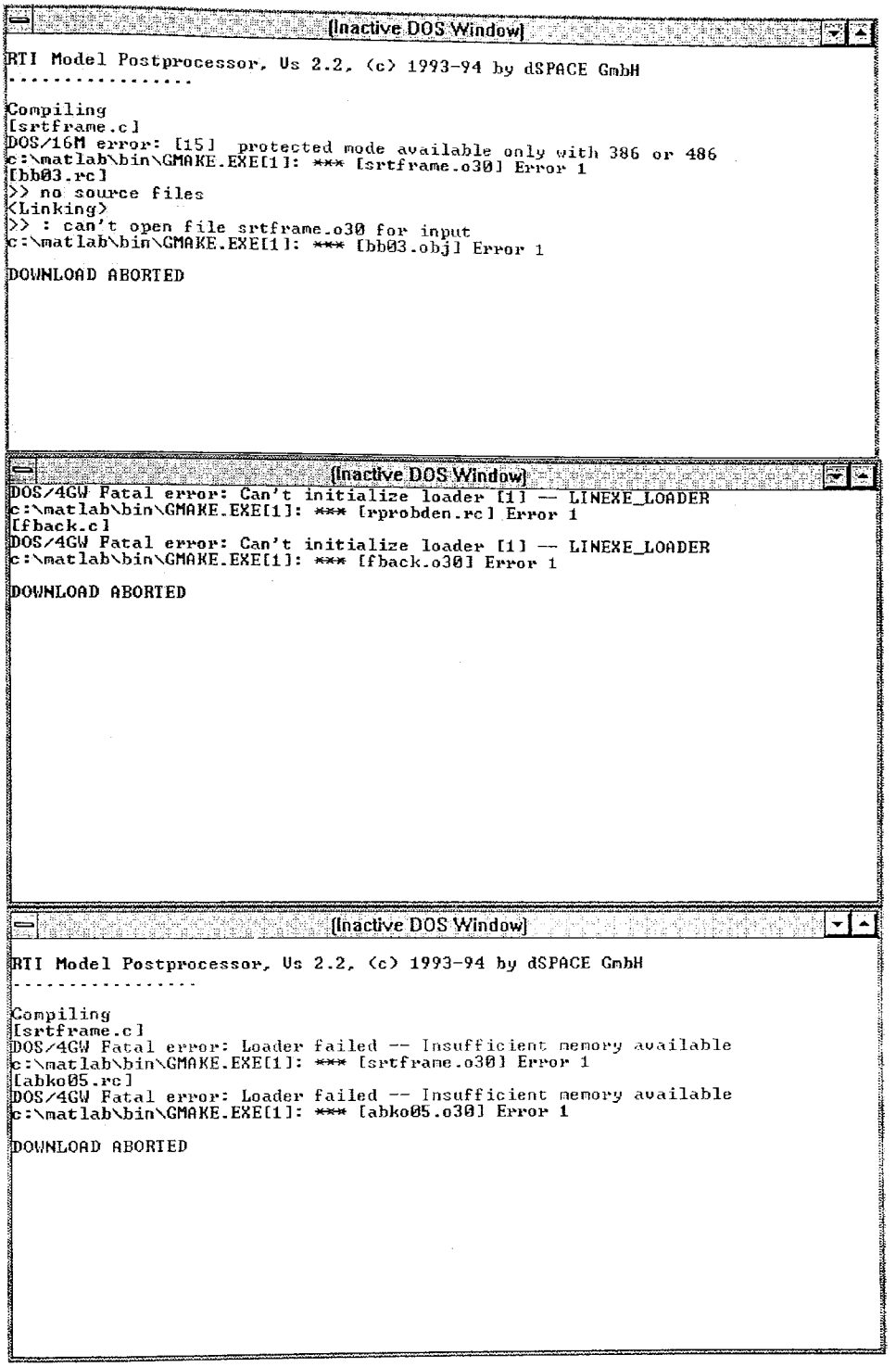


FIGURE 7.1. The possible messages of failure which will be cured by resetting the computer.

This is for avoiding the gathering of files scattered at random places since a download process produces more than 12 files per download. This change of directory will automatically be made when the present experiments are initialized by clicking twice to the

box in which the name of the experiment is written. The details of initialization is be explained in the sections defining the experiments.

Another very important point is that, if this is the **first download after the computer is powered up**, the power must not be connected to the motor. Just unplug the motor connection and then turn the power of the amplifier unit on. After successful completion of the download, the algorithm has to be run once and then stopped. At the first download after the computer is opened or reset for some reason, the DSP DA output has no value, which causes the saturation of the op-amp of the. Application of such an abrupt high voltage to the motor and therefore to thr mechanical structure may cause a mechanical breakdown. Therefore, precaution has to be taken and a voltage signal has to be sent to the DA output before the motor power cable is connected. As explained above, this is done by simply running the algorithm for a short while and stopping it afterwards.

Before connecting the motor power cable, it is advisable to calibrate the process. This is the insertion of the soft-fuses in the Simulink blocks to avoid unwanted displacements. How the calibration should be carried out is explained n the later sections. It is a step that can also be done here because it is easier to be carried out with no load on (no voltage to) the motor.

In Figure 7.1. some messages are shown for unsuccessful downloads. In all these cases, the computer is best to be resetted.

Another reason why downloading will be aborted is the absence of the m-files that has to associate C codes specified in an S-function. In this case the download operation will not even start and the message shown in Figure 7.2 will appear.

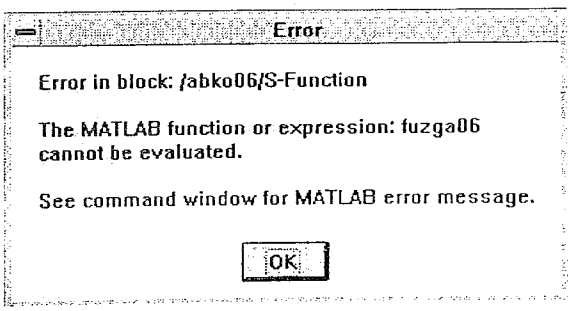


FIGURE 7.2. The error message displayed when the m-file that has to associate a C code is missing.

Wrong coding may also give rise to an error message. In such a case the C code should be corrected and then the real time should be rebuilt.

The computer of the X-Garden has a hard protect at the back on the parallel (printer) port. In the absence of this plug, the real-time will not work at all and a message of the following from will appear.

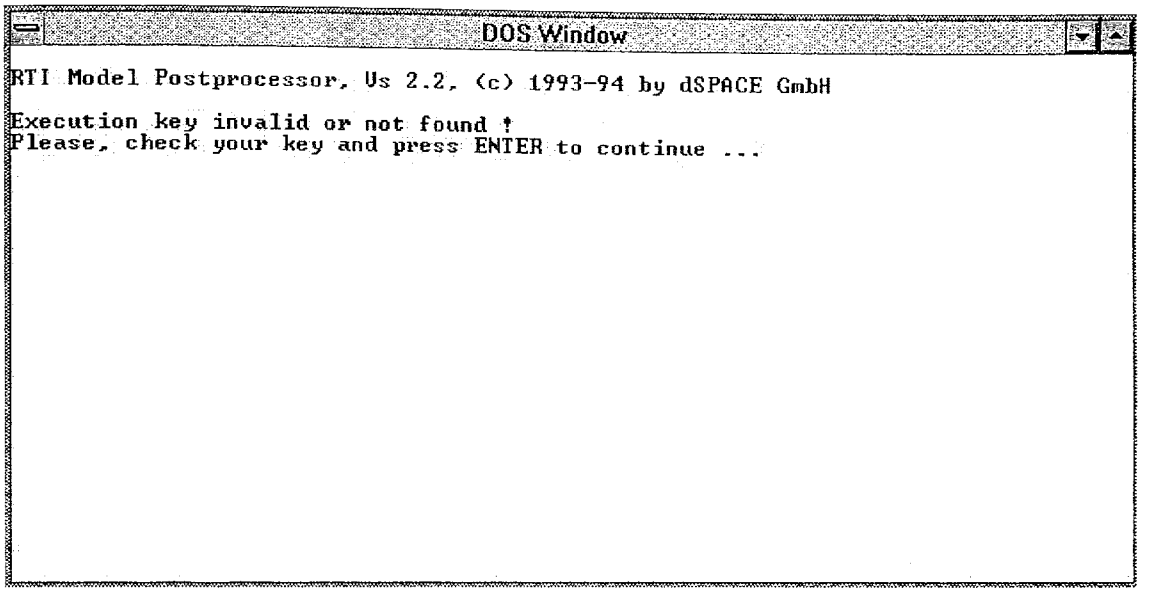


FIGURE 7.3. The error message displayed in the absence of hard protect.

If such a message is displayed then the hard protect at the serial pot of the computer should be checked. Sometimes, it may get loose and cause such an error. Tightening the socket will work.

These have been the most frequent errors that have occurred throughout the study.

When the model post-processing is over and the message "Download Succeeded" is seen on screen, this means that the algorithm is now on the DSP. It is either running or waiting for you to run it, depending on your choice that you must have specified in the "Build command".

If a present experiment is being run, then there should be a ready Cockpit interface ready for that experiment, and the only thing that has to be done is to open that file which is located under the directory c:\x_garden\members\...\cockpit. For each element of the garden there are separate directories, therefore, there are different cockpit directories for

each element. For example, consider the directory for the ball and the beam: c:\x_garden\members\ballbeam\cockpit.

However, if a ready Cockpit design is not present, the URL http://mecha.ee.boun.edu.tr/~lab should be visited for the details of how Cockpit is used. On top of this, the programs should be used in the laboratory to gain experience.

Assuming that one of the ready made designs is to be used and a successful download is completed, the Cockpit file should be opened.

It is to be noted that the opened Cockpit file is designed for the downloaded file and the corresponding trace file is loaded. If any error message is displayed regarding the absence of the trace file, then it can be loaded by choosing the "Load TRC file" from the "File" menu. The trace file must have been generated during the post-processing and be present at the current directory of Matlab.

When Cockpit is first opened, the screen that appears is the design environment. The interface is designed here. To run an application, choose the "Start" command from the "Animation" menu. This will open the session of application.

As it has been mentioned before, the animations are preferred to start at the pause mode, otherwise there would be a time lag between the start of the execution and the time that the user can open the corresponding Cockpit file. This time gap may cause instability or loss of transient response, therefore, if the simulation starts in pause mode, the user can have full control on the system.

The calibration is the first operation that should be done. While dealing with calibration, it is recommended not to turn the power of the motor on. In the calibration stage, the limits and states of the physical system are introduced to the software. The main idea is to bring the system to a state and to enter the corresponding voltage value to the appropriate block. Generally, the experiments are not designed to measure the position in terms of meters or the angle in terms of degrees but they are normalized within the workspace. The details of how each member of the garden will be calibrated is explained in the section where experimental designs are introduced.

As an additional precaution, when the simulations starts, the motor power is off. This is a software action, and in every design made, there is a separate section for the motor control, to run it on and off. When the calibration is done, op-amp saturation is avoided and

every thing seems to work properly, then the motor should be connected to the power amplifier so that the experiment could start.

It is to be noted that starting the animation session, starting the animation and starting the motor are three different things.

A final note on terminating the work is that, the motor should be disconnected from the power amplifier again by simply unplugging its power cable before turning the computer off.

The applications section will introduce sample applications both in the form of simulations and real-time applications and will give information on each setup individually.

8. APPLICATIONS

The applications that have been made can be introduced in two subsections as **simulations** and **real-time applications**. In Matlab if the command "**xgarden**" is used, a window will open which will help the user to reach each and every simulation or real-time application prepared. Simply write **xgarden** to the command line of Matlab. Among many such applications only a few samples will be demonstrated below which will help the user to understand what has been done and also these examples will provide a basis for new applications. The software that is written in C code and embedded in the applications will be introduced in more detail in the software support section. These codes can be easily adopted or improved and used in other fuzzy applications.

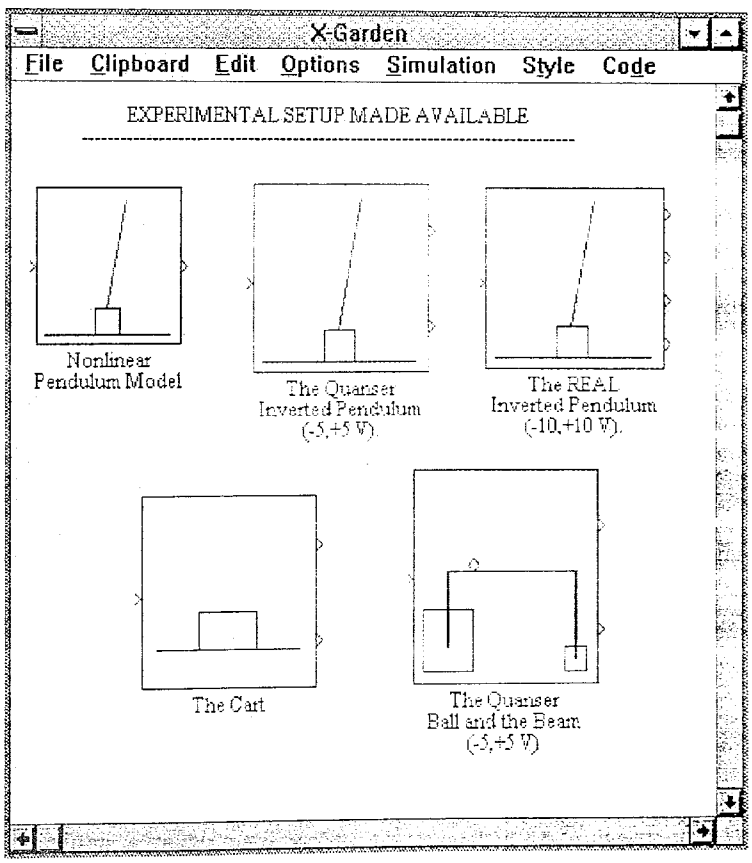


FIGURE 8.1. The blocks that will be used in simulations and real-time applications.

The main blocks that has to be used in simulations and real-time applications are prepared in Simulink and grouped under the X-Garden icon.

The blocks introduced in Figure 8.1. are recommended to be used, since they are specifically designed for this purpose. The first one is an inverted pendulum model to be used in simulations. The parameters of the pendulum can be specified by double clicking on the icon. The other block are designed for real-time applications and their internal structure will be explained in the following sections.

8.1. Simulations

The simulations are generally concentrated on inverted pendulum, but there are other simulations a explained in previous sections such as sine wave learning or linear position servo. The simulations can also be divided into two categories as adaptive and non-adaptive fuzzy applications. The inverted pendulum simulations are non adaptive and are aimed to show the similarity in the performance of PD and Fuzzy PD controllers. The adaptive simulations have been introduced in the previous sections, therefore, will not be repeated here.

When the fuzzy rules are based on the error and the rate of change of error, the resulting fuzzy controller can be perceived as a "Fuzzy PD". In the simulations, different P, PD, PID, Fuzzy P and Fuzzy PD examples are provided. Running these simulations are not even necessary, since the results are saved and can also be reached and plotted by using the **xgarden** or **sim** command.

8.1.1. Non Adaptive Fuzzy Control Simulations

Before illustrating the simulation results, how non adaptive fuzzy controllers for simulation are designed are explained.

Examine Figure 8.2. which is designed as a Fuzzy PD control of the inverted pendulum model.

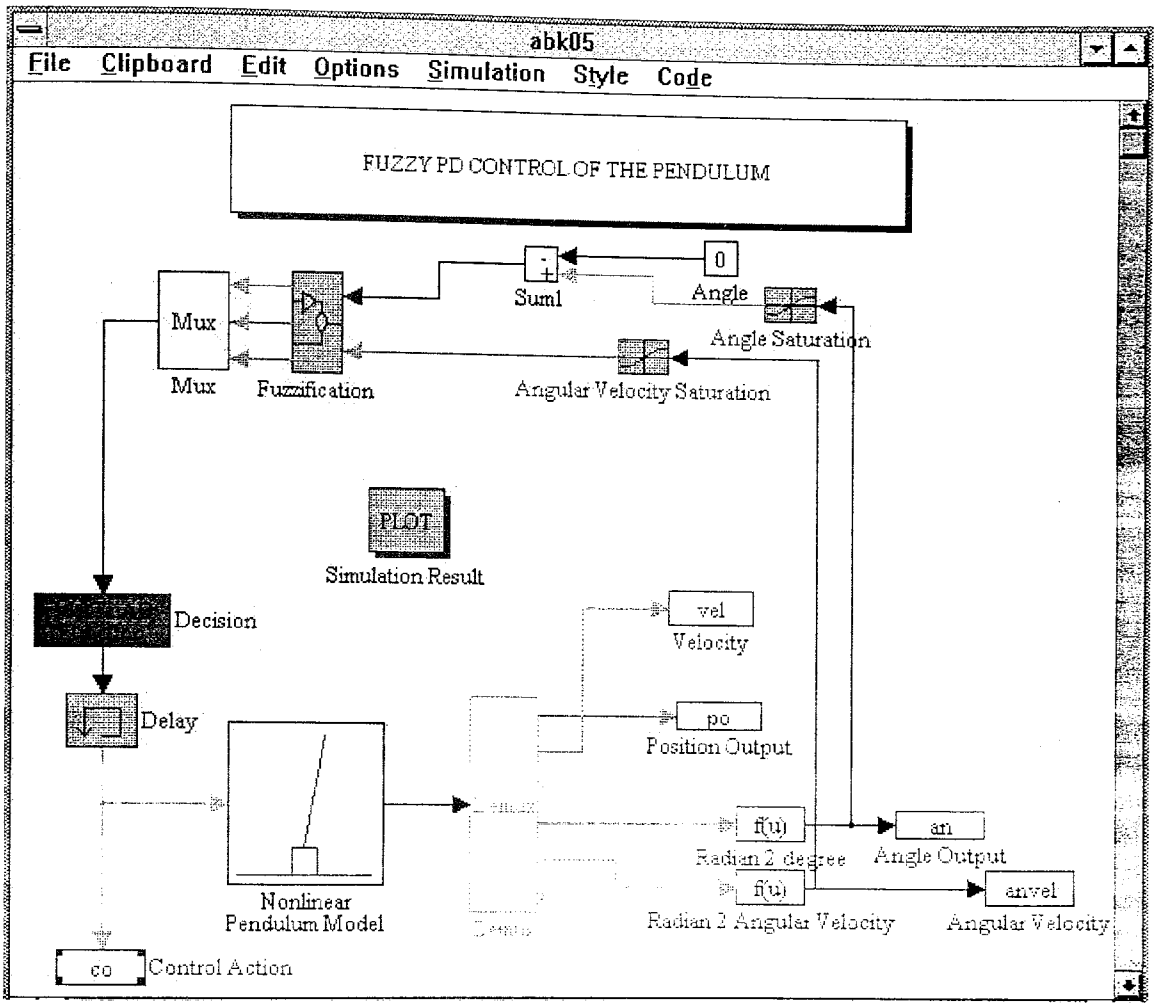


FIGURE 8.2. The Fuzzy PD control structure proposed for inverted pendulum.

The first and the most important thing to start either a simulation or a real-time application is to initialize the necessary variables in Matlab workspace. This is simply done by double clicking on the text box containing the name of the simulation. In this case, just double click on the name box "FUZZY PD CONTROL OF THE PENDULUM". The parameters will be initialized and a message in the Matlab screen will indicate that simulation is ready to start.

In this simulation the two important part are the "**Fuzzification**" and the "**Decision**" blocks. The fuzzification block contains the individual membership functions and the

variables **angle** and **angular velocity** are fuzzified within this block. The output of the block is a vector containing the fuzzified values of these two variables.

There are Simulink blocks for different membership functions under the icon "Fuzzy Membership Functions". The Figure 8.3. shows the membership function blocks available.

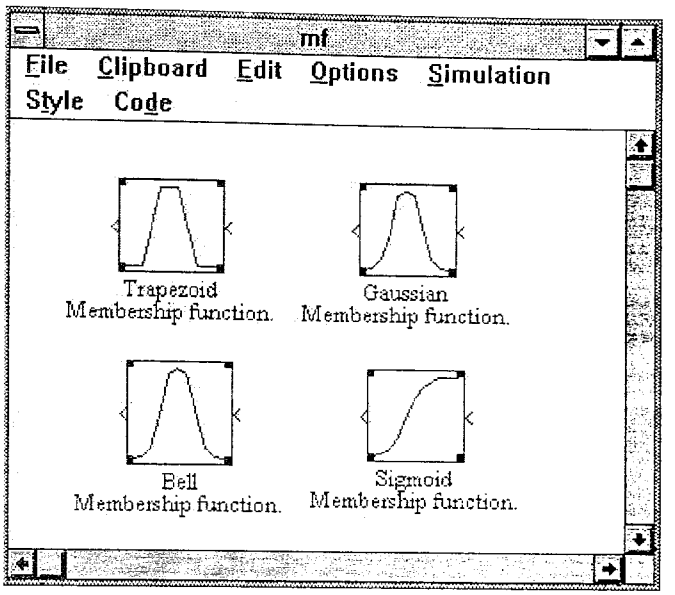


FIGURE 8.3. The membership function blocks available in Simulink.

The parameters of the membership functions can be changed simply by double clicking on the icon.

The **Fuzzification** block employs many such blocks and returns the fuzzified vector corresponding to the input vector. This vector is fed to the **Decision** block.

The **Decision** block contains the rule base and performs inference and defuzzification and finally produces a control input for the control system. The decision block is specifically designed for inverted pendulum and its code will be given in the following section.

The Fuzzy PD control of the inverted pendulum, starts with an initial angle of nine deg. The pendulum cart is assumed to have a mass of 0.5 kg and the pole is assigned a mass of 0.1 kg and a length of 0.6 m.

The membership functions used in balancing the pole are shown in Figure 8.4.

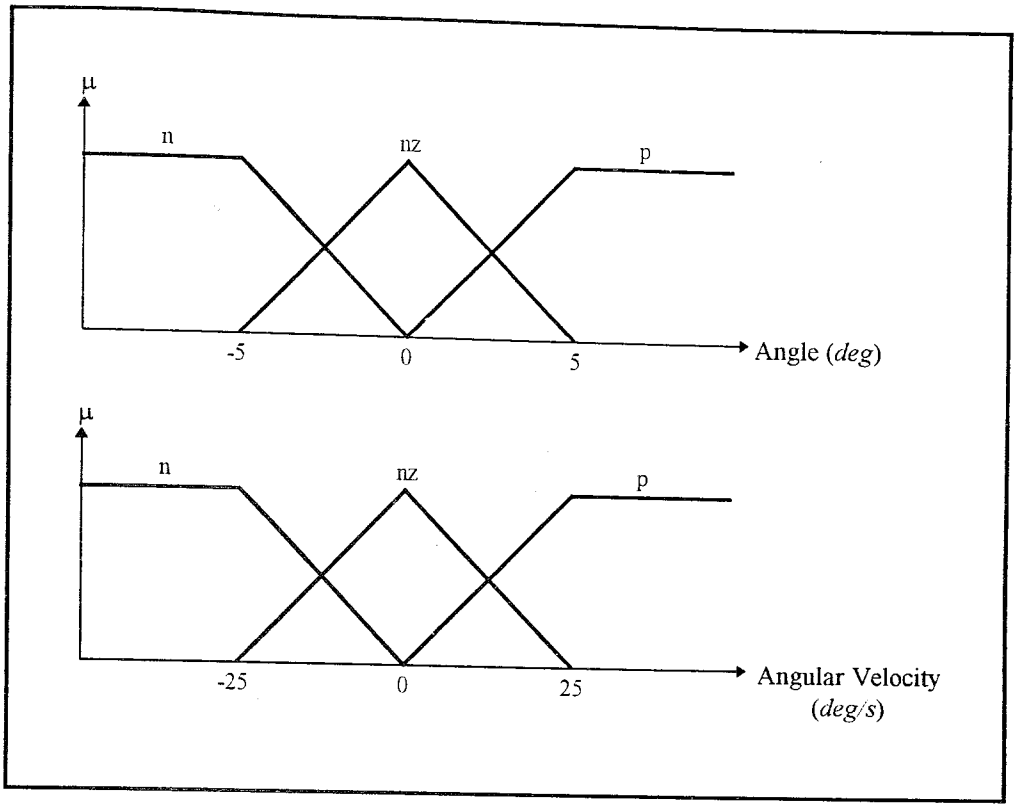


FIGURE 8.4. The membership functions corresponding to angle and angular velocity used in pole balancing.

The rules governing the fuzzy controller and the corresponding weights are tabulated in the Table 8.1.

TABLE 8.1. The rule base and the rule weights used in pole balancing simulations.

Angular Velocity \ Angle	n	nz	p
n	hn	sn	nz
nz	n	nz	p
p	nz	sp	hp

where,
hn=-120, n=-110, sn=-15,
nz=0,
hp=120, p=110, sp=15

The response of the pendulum is given below. Since there are several simulations on the pendulum, here only one selected simulation result will be presented, however, every simulation can be run by following the **sim** command. This command will provide necessary guidance to reach every simulation, stored data and will help to plot their results.

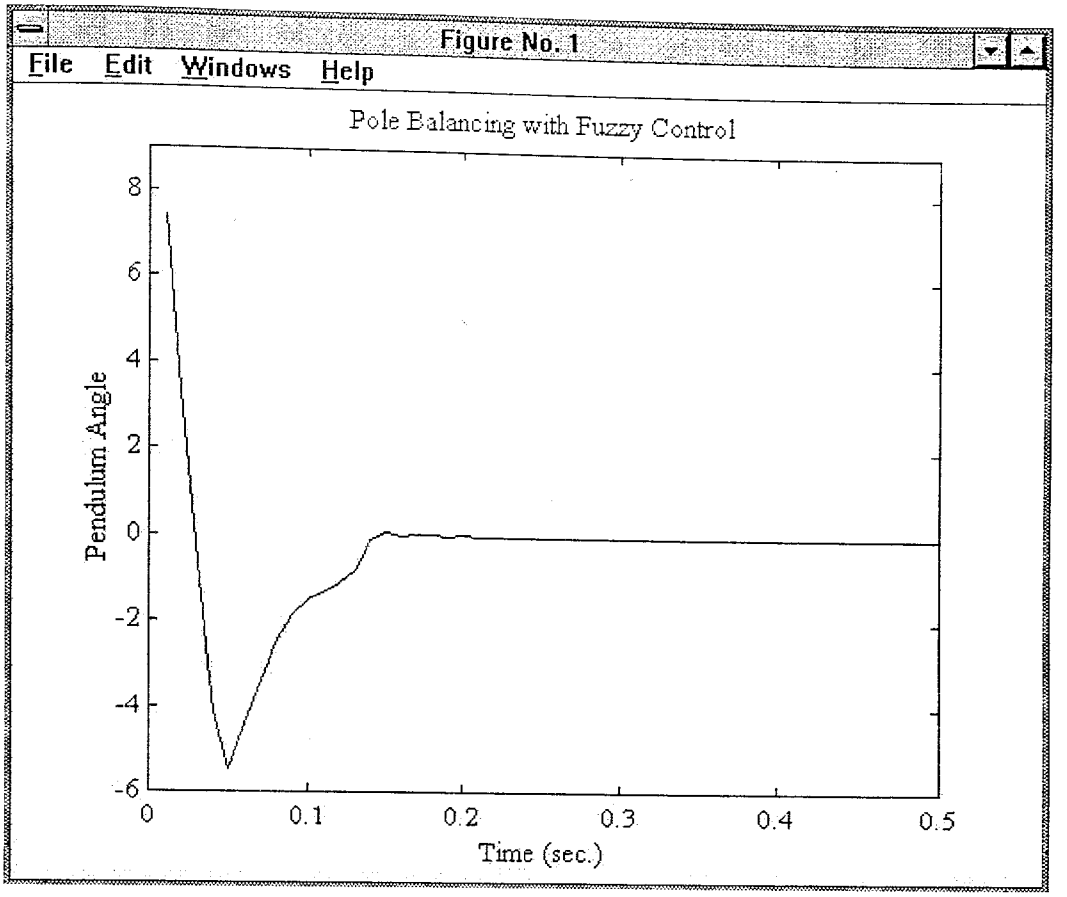


FIGURE 8.5. The response of inverted pendulum with a Fuzzy PD controller.

8.1.2. Adaptive Fuzzy Control Simulations

A number of simulation studies on adaptive fuzzy controllers are carried out to understand its potential applications in real-time control of physical systems. The simulation results have indicated the intuition that this algorithm has a very powerful adaptation capability. The most important feature of adaptive fuzzy systems is the capability of importing experiences into the controller. This means that the controller does not start adaptation with random values as it is the case in the artificial neural networks. In the sine wave learning example, this feature has already been demonstrated. The six rules which define the rule base have improved the learning with respect to a system which starts from

scratch. Since the results of some adaptive simulations have been introduced in section 4.3. and in Appendix F, no additional example will be given here.

8.2. Real-Time Applications

The real-time applications can also be divided into two categories as it has been the case in simulations, adaptive and non-adaptive fuzzy control applications.

The experimental studies on the adaptive fuzzy controllers have indicated that considerable amount of effort is required to obtain a satisfactory performance. However it is seen that a very good performance in prediction can be obtained. Therefore, an example on position prediction will be introduced below.

Among the non-adaptive fuzzy controllers one application will be introduced for each setup and it is hoped that these will complete the necessary information for a beginner to run a present application in real-time on an experimental setup.

8.2.1. Non-Adaptive Fuzzy Control Applications in Real-Time

In this section each setup will be considered and the necessary information to run these systems will be completed with an example. There are three main systems, namely, the position servo, the inverted pendulum and the ball and the beam setup that will be introduced respectively.

8.2.1.1. The Linear Position Servo

Figure 3.6 shows the Simulink screen for the fuzzy PD position control of a linear servo.

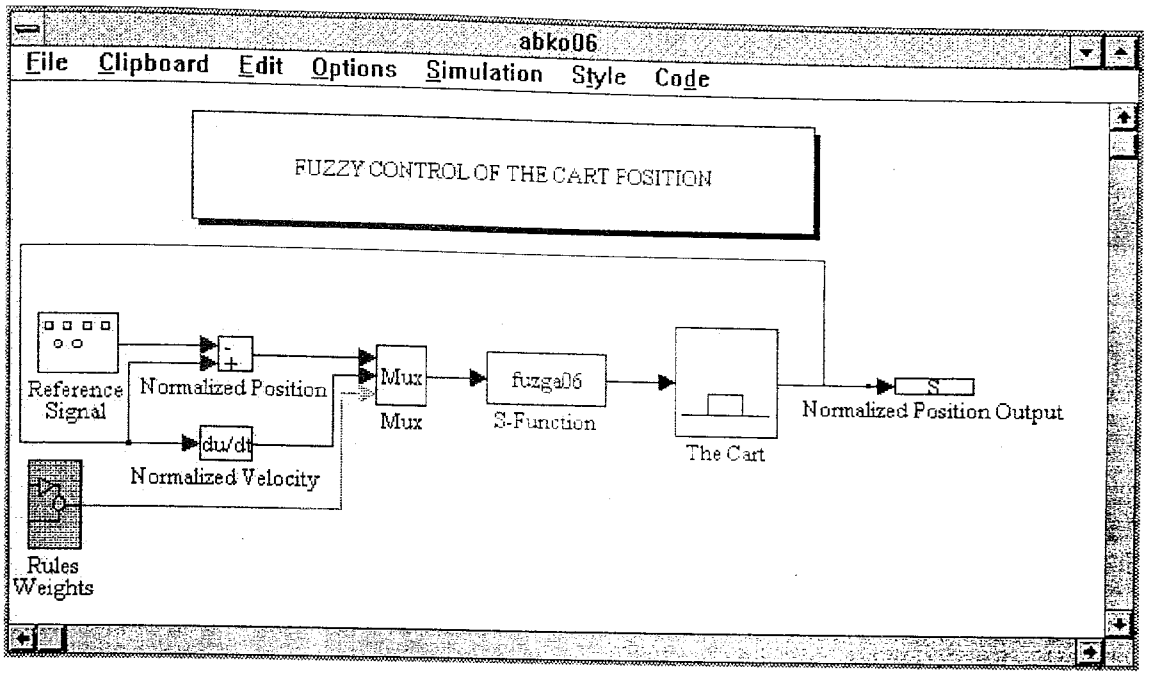


FIGURE 8.6. The Simulink block diagram used in position control of the pendulum cart.

The system is very simple here. The block "The Cart" is the block that provides the connection between the computer and its environment. This block has inside the AD DA connections an the input to the block is sent to the DA as a control signal and the output of the block is the normalized position of the cart.

First of all, before staring to download this file for real-time application, the design has to be understood in detail since the other designs are very similar in structure.

At this stage the power amplifier may be turned on but the power cable of the motor should be unplugged until the algorithm is successfully .downloaded and the op-amp saturation is avoided. This has been explained previously. Also, before proceeding, the all the cable connections between the setup, amplifier and the computer have to be done except the motor power cable.

The "**S-function**" contains the C code for the fuzzy controller, the "**Rule Weights**" block provides the rule weights. The rest is self explanatory.

The inside of the S-function is explained in detail in the software support section.

The inside of "The Cart" block has to be understood well for better use of the system. Figure 8.7. shows the inside of this block.

As mentioned in the simulation section, the **initialization** of the software environment is necessary also in real-time applications. Similarly, a double click on the name box

"FUZZY CONTROL OF THE CART POSITION" will initialize the system parameters for the application.

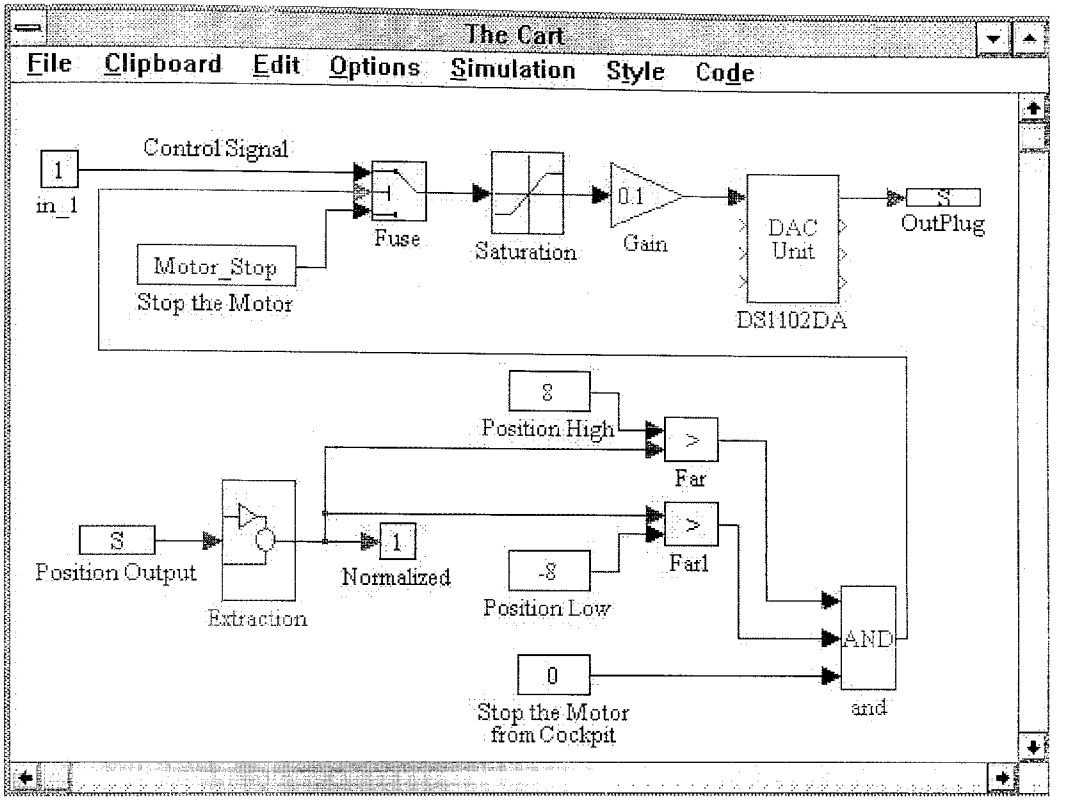


FIGURE 8.7. The inside of The Cart block.

The blocks "**Position High**" and "**Position Low**" determines the limits that the software will automatically cut the motor power off when the cart exceeds these limits. Therefore, they are referred as "soft fuses".

The value of the block "Stop Motor from the Cockpit" provides the control on the motor. It either has the value one or zero. Zero disables the motor and one allows the motor to run. This is done simply by actuating the relay block "Fuse". If the block "Stop Motor from the Cockpit" has a value of zero than this relay will pull the relay down and stop the motor. If it has a value of one, it will push it up and the control signal sent to the motor will pass through.

The "**Extraction**" block provides the normalized position information. The details of this block will not be given, but for the sake of completeness it will be noted that this block contains a four channel AD converter, a set of filters to get rid of measurement noise and a

set of normalization blocks to express the measured voltage that correspond to the cart position within a desired interval.

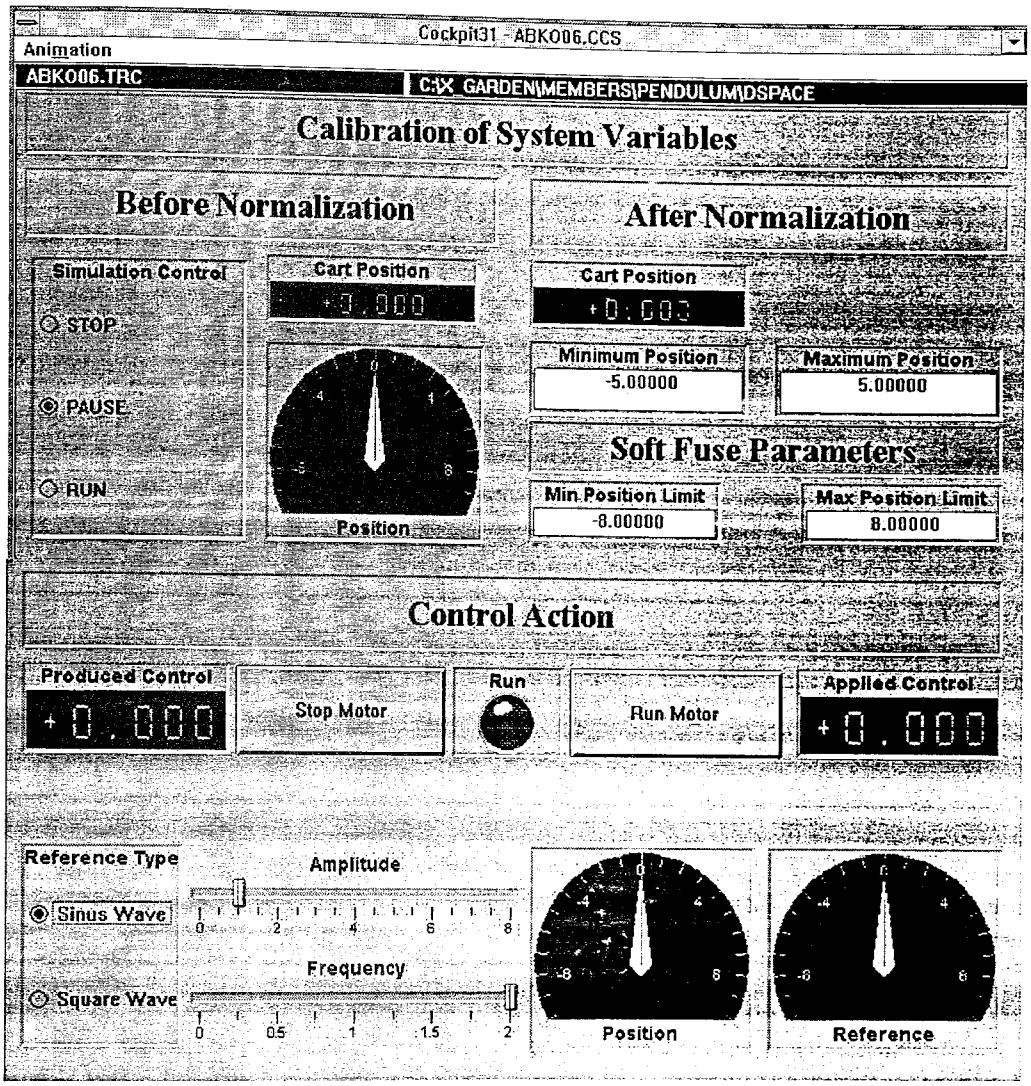


FIGURE 8.8. The Cockpit file designed for cart position control.

The normalization is a process that should be done at the beginning of each application to limit the measured position within an interval. Generally, [-10, +10] interval is preferred. Before normalization can be performed, the Simulink file has to be downloaded onto the DSP and the Cockpit should be run. It is assumed that after a successful download, the Cockpit is opened.

The cart potentiometer also has to be initialized before running the cart. If this is not done the position potentiometer may be damaged during application. Since the cart can

rotate around the slider and the potentiometer can rotate when it is not in contact with the rack, the potentiometer and the cart positions can change independently. Therefore, the potentiometer may easily get stuck while running the cart. To avoid this the cart potentiometer should be initialized. First the cart should be hold up so that the potentiometer will not touch the rack any more. Then the cart should be pulled to the left end of the path. Finally, the potentiometer gear should be rotated counter clockwise till the end and a slight clockwise rotation will complete the potentiometer initialization. The cart is then placed on to the rack. Such an initialization is not required in other sensors since their mechanical fixture does not allow independent motions.

When the Cockpit is opened with the file "**abko06.ccs**" corresponding to the downloaded Simulink design, the following screen will show up. From the animation button, choose "**Start**".

The first part of the Figure 8.8. is named "Calibration of System Parameters" and is divided into two columns as "Before Normalization" and "After Normalization". The first thing that will be done is to start the simulation, which is supposed to be at the pause mode, by pressing the "RUN" button in the "Simulation Control" box. This will start the simulation. After this point on, no op-amp saturation problem will arise. Then the calibration (of normalization limits) has to be performed. Observe that, when the cart is moved on the path, the values in the boxes "**Cart Position**" change simultaneously. There are two different "**Cart Position**" boxes. The one in the "Before Normalization" column shows the voltage value corresponding to the current cart position. The other block shows the normalized value of the position. However, the above mentioned fact of position potentiometer and cart position not being mechanically coupled, causes slight changes in the normalized values. Also, the air temperature has some slight effect on the values read. Therefore, to get rid of these effects, the normalization limits should be introduced to the software. This is done simply by moving the cart to the ends of the path. If the power amplifier is not turned on up to this stage, turn it on keeping the motor cable unplugged. First move the cart to the left end till the potentiometer gear reaches the end of the rack. Read the voltage value from the first "Cart Position" block and write this value into the "**Minimum Position**" box. Then the value read in the second "Cart Position" block will become -10.00. Similarly, move the cart to the right end till the motor gear reached the edge of the rack and this time write the read value to the "**Maximum Position**" box. And

observe that the normalized cart position will display +10.00. This will complete the normalization calibration.

The two boxes of "**Soft Fuse Parameters**" define the limits beyond which the cart power will be turned off. Some value near plus and minus eight is recommended.

The "**Control Action**" part provides the tools for controlling and observing the system variables during an animation is running. The push buttons "**Stop Motor**" and "**Run Motor**" are obvious. When a simulation is started, the motor is generally kept unpowered as an additional precaution. This is done by assigning a zero value to the "Stop the Motor from Cockpit" inside the cart block, as has been mentioned before.

After op-amp saturation is avoided and calibration is performed the motor power cable can be plugged. To run the motor press the "Run Motor" button. After this the motor will start to follow the reference signal.

Their positions will be displayed on the tachometers called "**Position**" and "**Reference**". The sliders "**Amplitude**" and "**Frequency**" changes the amplitude and the frequency of the reference signal. The reference signal type is a sine wave by default and this can be changed to a square wave easily.

If the response is desired to be plotted or saved, first the Trace program should be run. The usage of this program is simple and will not be explained here. The files saved by using the Trace program store the data in a specific format. There is a command "**graphs**" to plot the data file produced. First load the *.mat file produced by the Trace program and then simply write "**graphs**" in the Matlab command line. A bunch of alternatives will be provided for plotting the experimental results.

8.2.1.2. Example: Linear Position Servo

In our example again a rule base composed of nine rules are used. The position error and the velocity are used as the input variables of the fuzzy controller. The membership functions employed in the controller are given below.

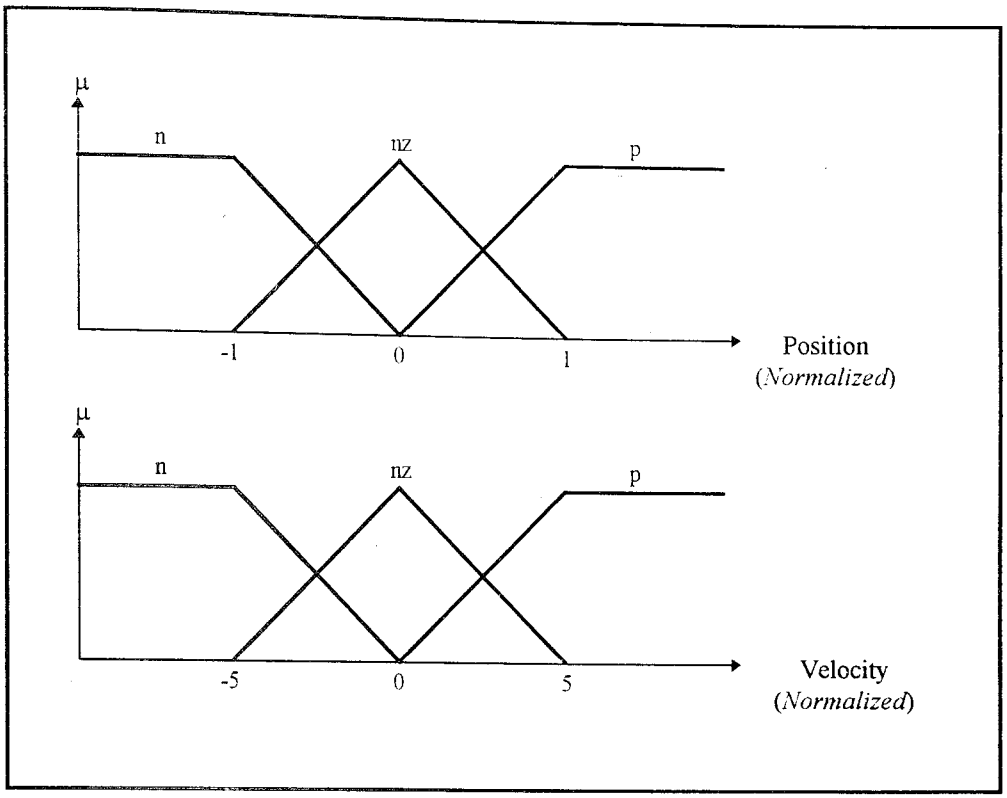


FIGURE 8.9. The membership functions corresponding to position error and velocity used in cart position control.

The rules governing the fuzzy controller and the corresponding weights are tabulated in the Table 8.2.

TABLE 8.2. The rule base and the rule weights used in cart position control.

Angular Velocity \ Angle	n	nz	p
n	p	sp	nz
nz	sp	nz	sn
p	nz	sn	n

where,
n=-2.5, sn=-1.5,
nz=0,
p=2.5, sp=1.5

The tracking performance of the cart with the above explained controller is illustrated in Figure 8.10.

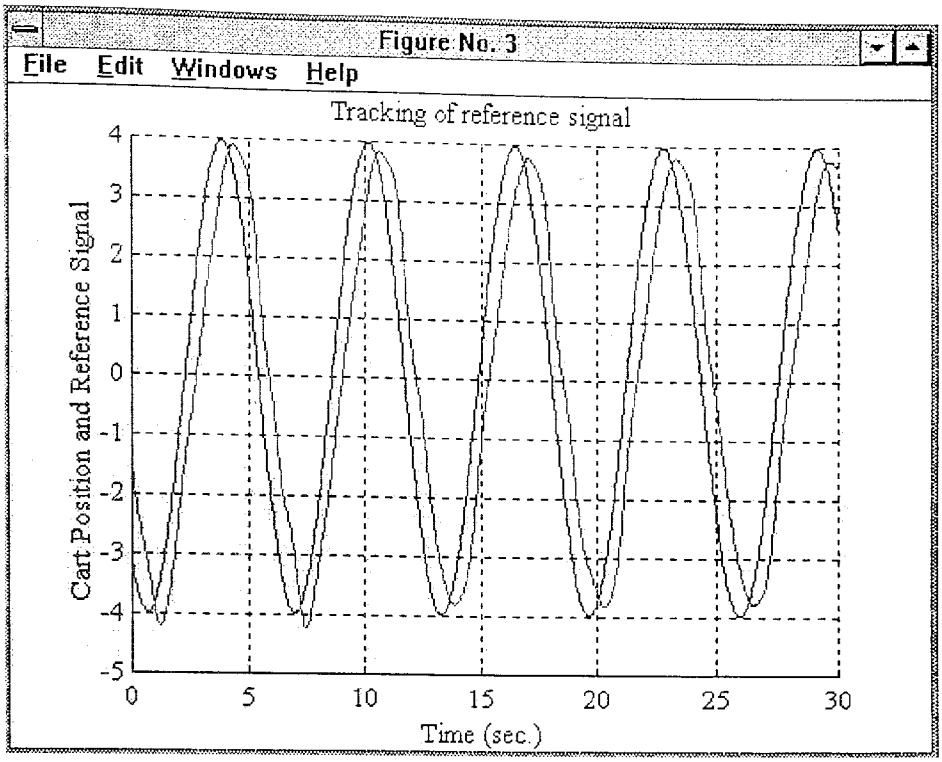


FIGURE 8.10. The reference signal and the tracking performance of the cart with a Fuzzy PD controller.

There are other fuzzy and PID applications prepared for experiments. The "**xps**" command will guide the user to these experiments.

When the experiment is finished the motor power cable should be unplugged.

8.2.1.3. The Inverted Pendulum

The inverted pendulum setup is nothing but the position servo with a pole located on the cart. Therefore, detailed information is not required for the inverted pendulum setup. To run it, every required step is similar to the position servo. The only additional thing that should be done is in the calibration stage, following a successful download. Previously, the limits of the cart track have been introduced to the software, and obviously this time the limits and the balance positions of the pendulum have to be introduced.

This process is very similar to the one that have been performed in position servo. Figure 8.11. shows the Cockpit screen used in calibration of the inverted pendulum parameters.

This time the cart position should be calibrated similar to the one that have been explained in the previous section. The pendulum angle calibration has three positions to be introduced to the system. The both sides to which the pendulums leans on and the balance position. The balance position is especially very important for a good performance.

First lean the pole to left and write the voltage read from the first "**Pendulum Angle**" display to the "**Minimum Angle**" box. Similarly lean the pendulum to right and write the corresponding voltage value to the "**Maximum Angle**" box. Finally, balance the pole and write the voltage corresponding to this state to the box "**Balance**".

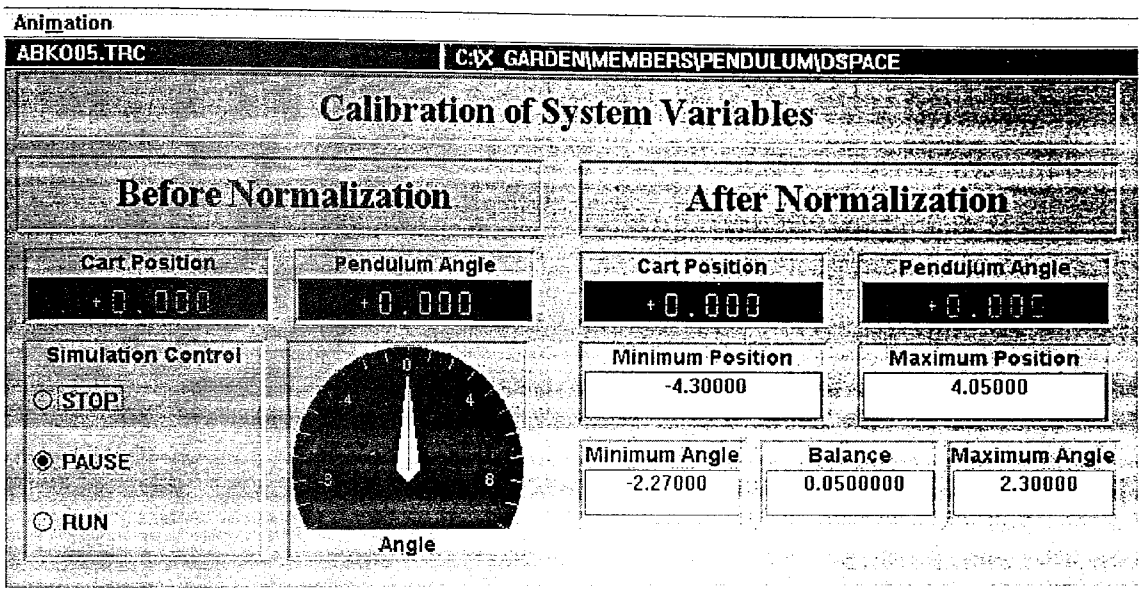


FIGURE 8.11. The Cockpit file designed for inverted pendulum control.

The inverted pendulum controller is built similar to the position servo controller, however, satisfactory results could not be obtained. Tuning the membership functions and the rule weights improved the performance, but the optimum combination could not be obtained. This outcome indeed reinforces the fact stated in the previous sections that the most difficult part of writing fuzzy controllers is achieved at the stage of assigning quantitative values to the qualitative information. The effort spent on the tuning process showed that an appropriate set of parameters can balance the system, but this is very time consuming.

8.2.1.4. The Ball and The Beam

This set up somewhat different with respect to the previous ones. As explained in the previous sections, the mechanical structure of the setup necessitates careful use of the system.

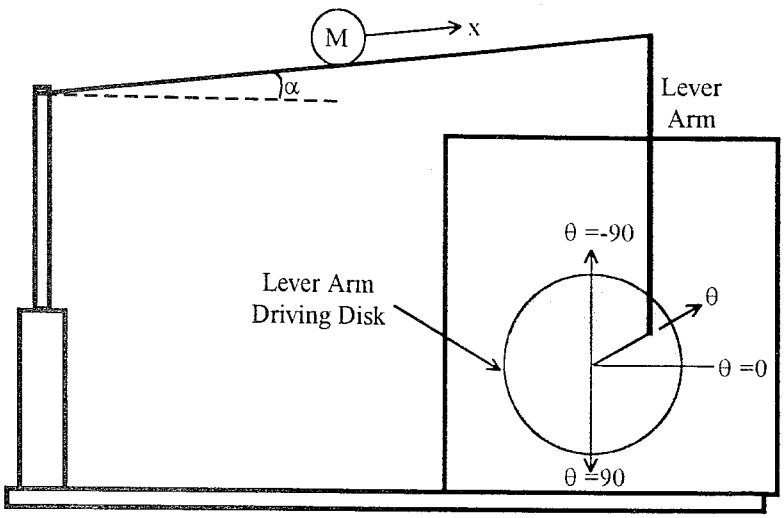


FIGURE 8.12. The ball and the beam setup.

The lever arm is fixed to the lever arm driving disk with a bolt and this bolted structure is the cause of a problem. To get rid of this problem, the setup should be operated within the [+90,-90] interval. Full rotations should be avoided. Notice that the angle is positive clockwise.

Simply, "**xps**" command will lead the user to the ball and the beam applications. The details of operation will not be re-explained on this setup since sufficient information have been presented on position servo. The only difference is again at the calibration stage. In this setup, similar to the inverted pendulum setup, five states have to be introduced to the software. These are the two ends of the rod on which the ball rolls, and the three different θ values (-90, 0, +90). The -90 and +90 determines the interval of operation and the 0 value determines the parallel rod position. This value is very important for better performance. Since it effects the stability, it should be carefully calibrated.

The controller suggested at the Quanser handbook is for this setup is a state feedback, which is worth trying, but within this thesis, a different approach is used and resulted in quite satisfactory results. Making use of the fact that the ball motion is very slow with

respect to the motor response a controller is designed. The motor is controlled by a simple PID controller. The aim is to consider the current state of the system (the ball position and the velocity) and decide the angle of the beam that will result in the target ball position. This decision forms the desired value of the beam angle and in the effort to reach this angle a PID controller is used. The decision of the angle is produced through a discrete table similar to the one mentioned in defining membership expression types.

The structure of the control system is shown in Figure 8.13.

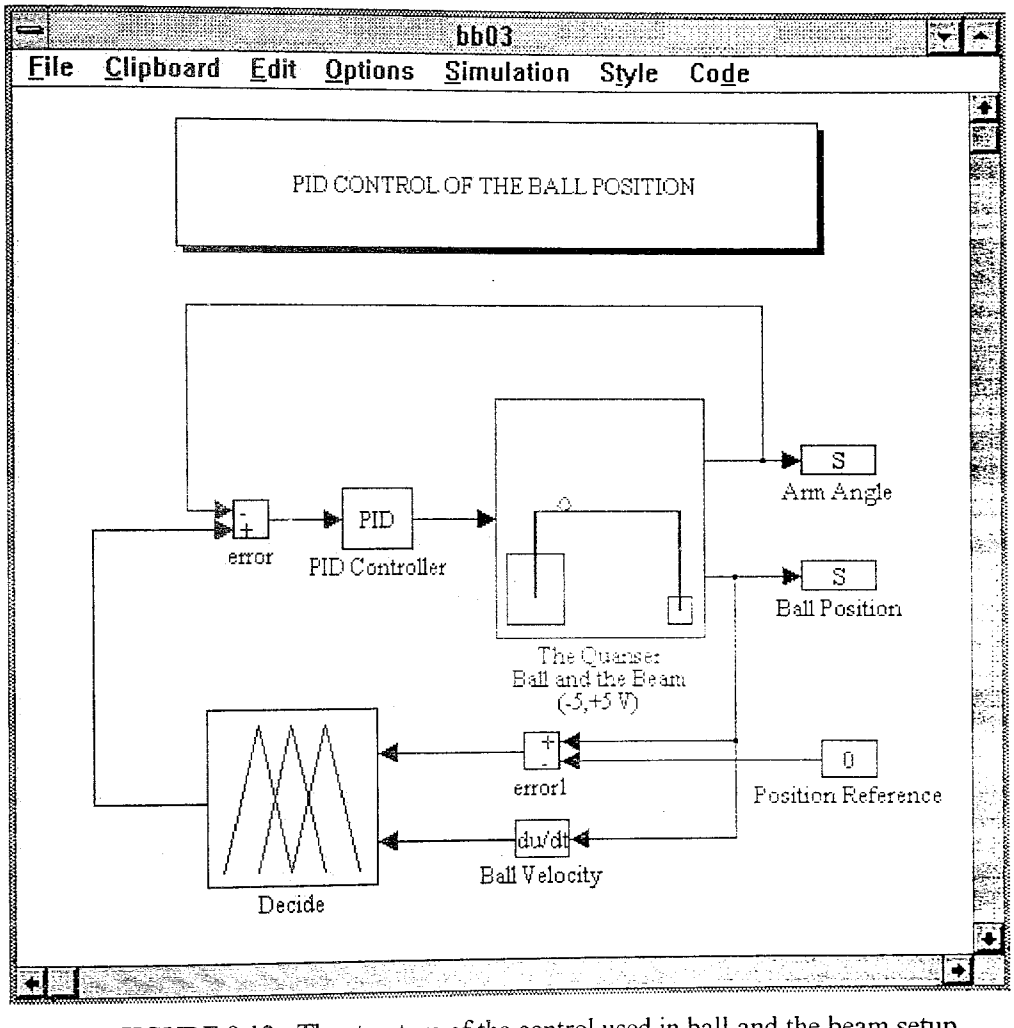


FIGURE 8.13. The structure of the control used in ball and the beam setup.

An easy calibration of the table resulted in successful applications. A sample result is shown in Figure 8.14.

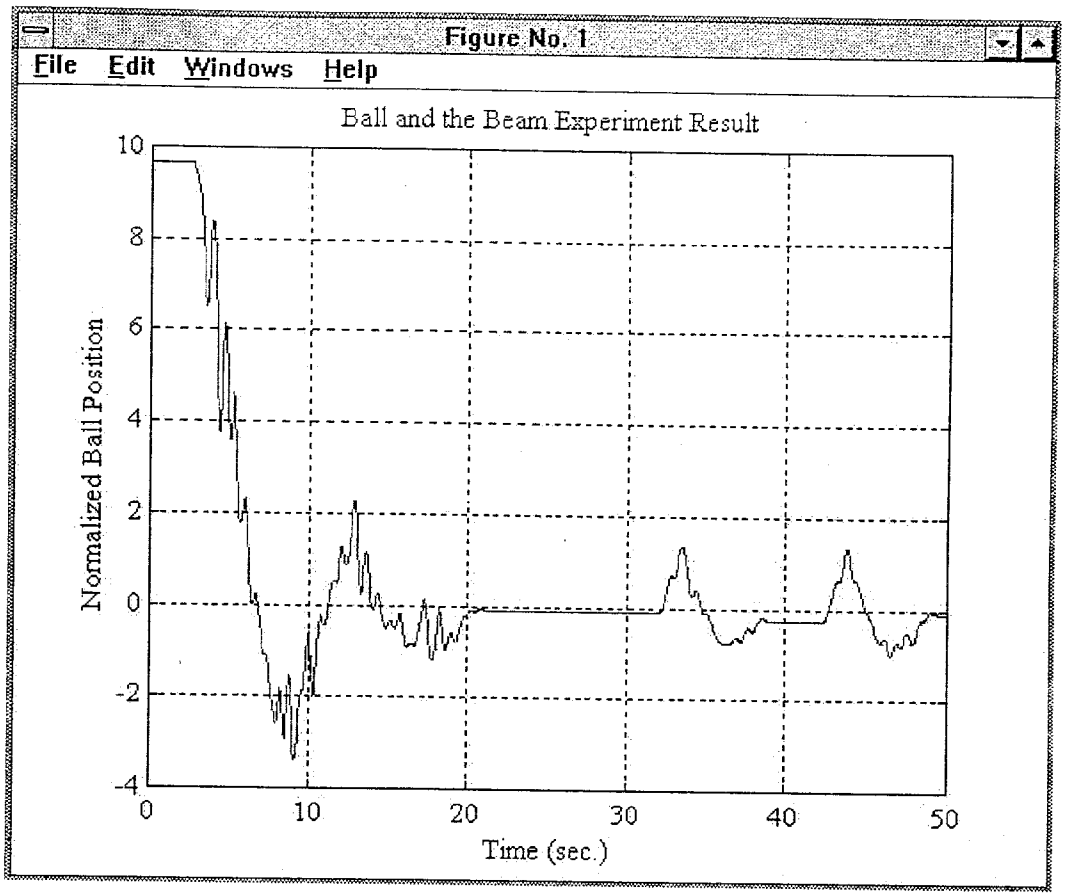


FIGURE 8.14. A sample result of ball and the beam experiment.

The control starts at the third second and restores the ball to the origin. After thirtieth and fortieth seconds two disturbances are introduced. A very important problem of the system has to be pointed out that, the ball position sensor is subject to sudden measuring error. This is most probably due to rolling, but it has a very degrading affect on the performance of the system. Cleaning the ball path will improve the performance. Take care to clean the conductive material with alcohol.

8.2.1.5. Position Estimation by Using Adaptive Fuzzy Systems

This experiment is designed to show the potential use of adaptive fuzzy systems in prediction as well. The main idea is to predict the position of the cart relying on the current state of the cart (velocity) and the control input applied at that instant. The control system is shown below.

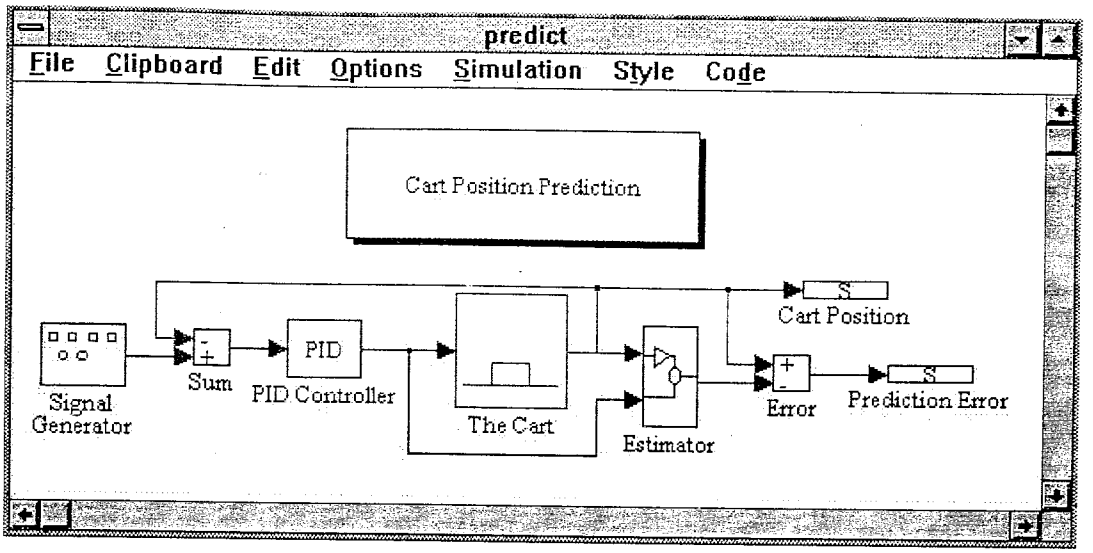


FIGURE 8.15. The block diagram of the position predictor.

The idea is very simple in this example. The estimator considers the current velocity and the control input to the system at one instant and depending on these values a fuzzy decision produces the estimated displacement that will take place in one sampling interval. This estimation is added to the current position and the result is the position prediction of the cart for the next sampling time.

TABLE 8.3. The rule base used in position estimation.

Control Input \ Cart Velocity	n	sn	nz	sp	p
n	-0.20	-0.15	-0.10	-0.05	0.00
sn	-0.15	-0.10	0.05	0.00	0.05
nz	-0.10	-0.05	0.00	0.05	0.10
sp	-0.05	0.00	0.05	0.10	0.15
p	0.00	0.05	0.10	0.15	0.20

As explained before, the membership function used in adaptive systems is Gaussian. The center of the Gaussian membership functions are taken to be [-4, -2, 0, 2, 4] for both

cart velocity and control input and [2] is selected as the standard deviation of all the membership functions.

The Table 8.3. illustrates the rule base used in the system.

This way of tabulating the rule base is an alternative. Rather than specifying the rules and the rule weights separately, they can also be tabulated in this way in a more compact manner.

The results of the position estimation is presented in Figure 8.16.

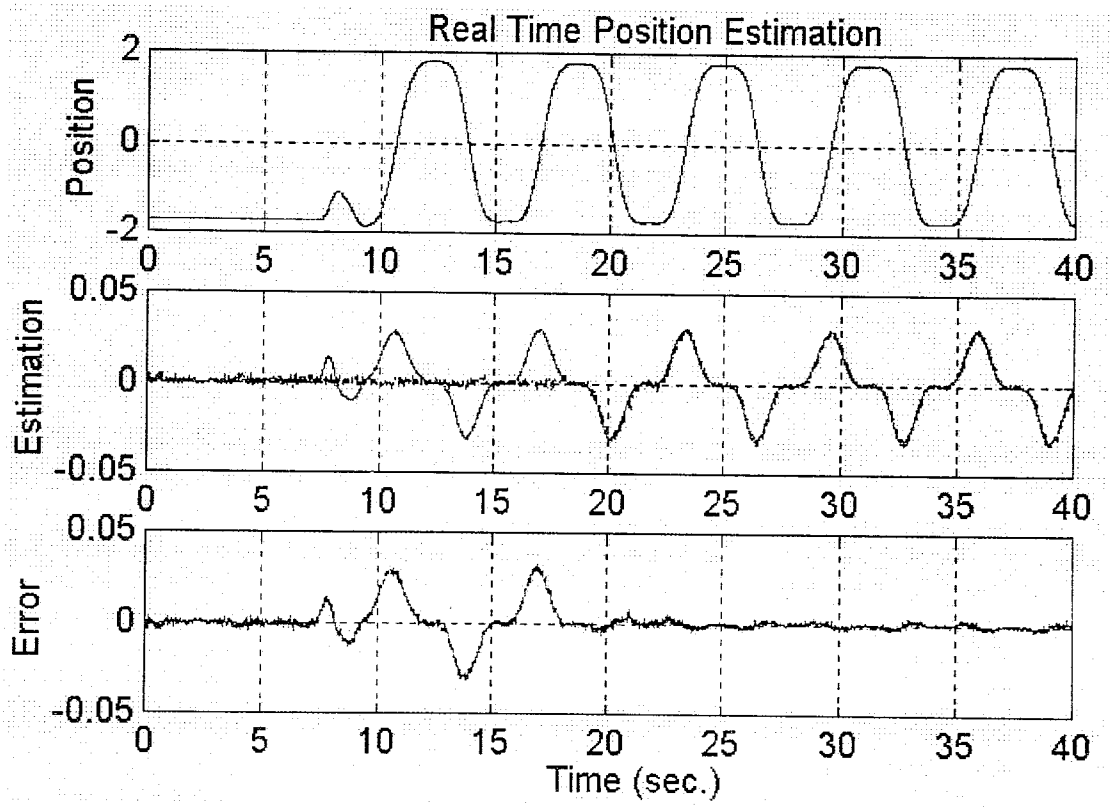


FIGURE 8.16. The result of position estimation.

The first plot shows the position tracked by the cart during the experiment. It is observed that the controller acts sometime after the fifth second. The second plot shows the estimation and the actual values of cart position. Until the 17^{th} second the algorithm runs with the above mentioned parameters and learning rate is kept zero to avoid adaptation. The plot shows that without adaptation the membership assignments provides no useful estimation. However, afterwards the learning rate is introduced and adaptation starts. The

performance immediately improves and both the estimation and the actual values overlap. The final plot shows the error profile of the experiment.

Some important remarks are to be made. First of all the position are given as normalized position of the cart and the whole track of the cart is normalized within the interval [-3, +3]. The second plot "Estimation" shows the actual and estimated values of the change in cart position in one sampling interval. Therefore, its value is very small with respect to the position. If it was illustrated on the position, then the performance of the estimator would not be observed due to small amplitude of the displacement in a sampling interval.

9. SOFTWARE SUPPORT

The software written within the scope of the thesis is mainly centered around fuzzy logic controllers. They can be gathered under four main groups: adaptive and non-adaptive m-files used in simulations, adaptive and non-adaptive c-codes used in real-time applications. Since the M files are very simple and short codes they will not be explained, however, the C codes are explained below in detail. With very little adaptations these files can be used for different applications. The real-time application codes are all located under the directory:

c:\x_garden\members\s

A convention used in these files is to be clarified before proceeding to the individual definitions. Consider the Table 9.1. This table shows the way the rules are numbered. The "**Angle**" is the first variable and the "**Angular Velocity**" is the second variable of this system and a total of nine rules define the rule base. The numbering of the rules are shown in the Table 9.1.

TABLE 9.1. The rule numbering convention used in source codes.

Angular Velocity \ Angle	1	2	3
4	1	4	7
5	2	5	8
6	3	6	9

Keeping this convention in mind a weight matrix corresponding to the below rule base can be written as.

[-5, -2, 0, -2, 0, 2, 0, 2, 5]

The rule and weight association is shown in Table 9.2.

TABLE 9.2. The rule and weight association.

Angular Velocity \ Angle	1	2	3
4	-5	-2	0
5	-2	0	2
6	0	2	5

The matrices in the following codes are all written with respect to this convention.

9.1. Non-Adaptive C Code

The non-adaptive C code written is given in Appendix E. An analysis of the code will help to understand the algorithm. It is very easy to change the membership functions, rule base and the rule weights. Also with some additional effort, different defuzzifiers can be added to the code.

Currently, the mentioned code provides four different membership functions (Trapezoid, Bell Shape, Gaussian, Sigmoid), and four different types of T-norms (Minimum, Product, Bounded Product, Drastic Product). The defuzzification is only done with weighted average.

There are some parameters that should be updated in case of any change in the system. When the below code is traced over they will be observed and understood.

There are no restriction on the number of antecedents to be used in the rules, but they have to be specified in the variable **num_of_antecedents**. The rule and weight association

is to be done in the variable **rule**. The Table 9.3. shows the weight rule assignment for the following code.

TABLE 9.3. The rule and weight association.

Variable 2 \ Variable 1	1	2	3
4	10	5	-2
5	7	0	-7
6	2	-5	-10

Finally, the membership functions defined in the variable memberships is given in Figure 9.1.

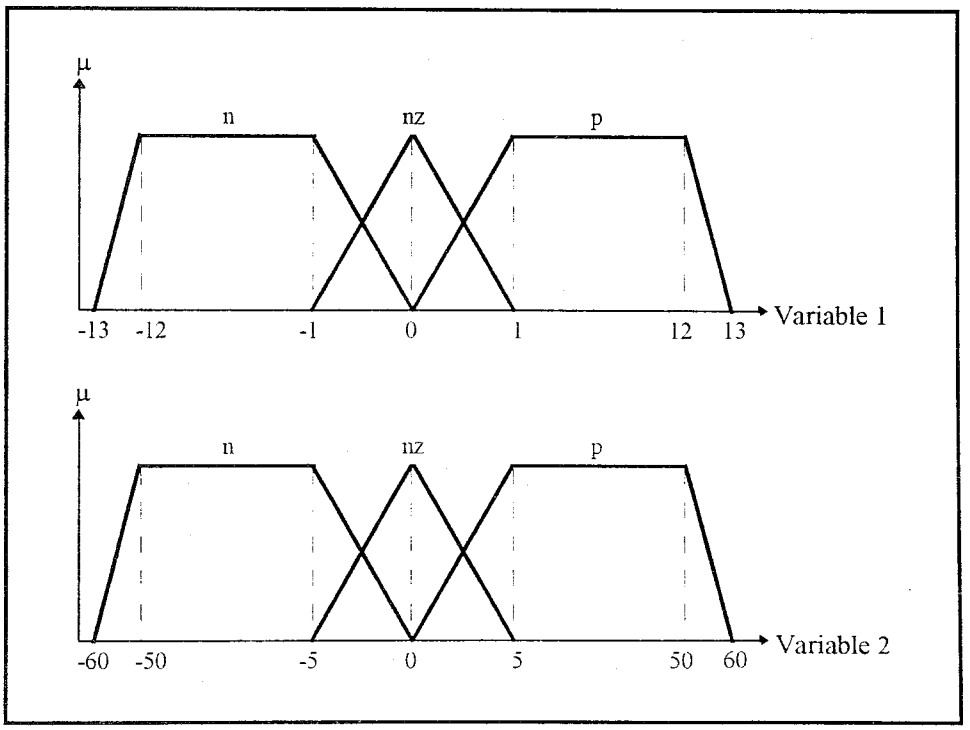


FIGURE 9.1. Membership functions used in the C code.

If another type of membership function is to be used, then the part of the code defining membership functions should be studied carefully.

9.2. Non-Adaptive C Code

This code is much simpler and shorter. There are three variables subject to adaptation in the code given in Appendix ..... The variables **XX**, **SS** and **YY** are subject to adaptation. **XX** is contains the centers of the membership functions for each rule. The n^{th} row corresponds to the n^{th} rule. The columns correspond to the input variables. In our case there are two input variables of the system (i.e. angle and angular velocity). **SS** is similar and it contains the standard deviations of the Gaussian membership functions. **YY** is the vector with rule weights and simply the first element of the vector corresponds to the first rule and the last element to the last rule. For better understanding the codes should be passed over carefully.

The meaning of the variable **loop_until** is also to be noted. In each sampling interval, as explained previously, the data are passed forward through the network resulting in an error between the actual and expected values and the network parameters are updated in accordance. However, this process may be performed more than once in a single sampling interval. The value of the variable **loop_until** determines how many times adaptation will take place in each sampling interval.

10. CONCLUSIONS

Within the scope of thesis work carried out a number of objectives have been achieved as outlined below.

Most importantly, an experimental environment, namely, X-Garden is constructed. It functions flawlessly in real-time and constitutes a very suitable test bed for the future studies. There are many experiments in the service of a beginner (i.e. open loop and PID controls). The environment is open to development both by adding new plants or by improving the present source codes.

In parallel with this study, a final year project has been completed. Within this project, a web site is built which explains the finest details of the X-Garden. The URL is:

http://mecha.ee.boun.edu.tr/~laboratory

Simulink is enriched with the blocks that are to be used in connection with X-Garden applications. The C codes written allows the user to make either adaptive or non-adaptive fuzzy logic control applications in real-time. The codes are written as transparent and informative as possible to provide easy tracing and update.

On the more academic side, a number of control algorithms are designed, simulated and some experimentally implemented. The simulation studies on the inverted pendulum have indicated that fuzzy logic is very successful in importing intuition and experience to the controller. However, inverted pendulum applications revealed the fact that, tuning fuzzy membership functions is not very easy, and is a very important problem of the fuzzy engineer which is not generally mentioned in literature. It is pointed out that this is a problem and there are no rules for membership function assignment, however, the difficulties are not emphasized sufficiently. The present inverted pendulum controller is promising but requires further tuning, making one appreciate the difficulties of tuning process.

Adaptive systems are claimed to avoid the tuning process. In simulations, the adaptive fuzzy controller performed much better than classical control techniques such as PD or LQ or even better than neural network based control. [20] However, in real-time, the adaptive

fuzzy controller has performed poorly as an inverse plant, whereas revealed very satisfactory results in estimation.

In summing up, it can be stated that the effort expended for this project has been quite fruitful. The laboratory within which this study was carried out (Mechatronics Laboratory) has profited from it because its full potential has been activated. The experience gained by simulation and experimental studies has been useful in understanding the limits of applied techniques and in understanding the difficulties involved. In this way a conceptual solidification has been reached and engineering practices of the author and the co-workers are improved.

APPENDIX A

The system parameters given by Quanser are tabulated in this section. However, an important note for the ones who are going to rely on the Quanser Consulting Handbook [22] is that sometimes metric and some other time British units are preferred.

TABLE A.1. The parameter values for linear position servo.

Parameter	Symbol	Value	Unit
Motor Torque Constant	K_t	0.00767	Nm/Amp
Back EMF Constant	K_b	0.00767	V/(rad/sec)
Efficiency	Eff	0.9	
Armature Resistance	R_m	2.6	Ω
Armature Inductance	L_m	0.18	mHenry
Maximum Voltage	Vmax	6.0	V
Internal Gear Ratio	K_g	3.7:1	
Armature Inertia	J_m	$3.87e^{-7}$	Kgm^2
Motor Gear Radius	r	0.00635	m
Cart Mass	M	0.455	Kg
Cart Position Potentiometer	R_s	10000	Ω
Bias Voltage	V_b	+/- 12	V
Bias Resistors	R_b	7150	Ω
Rotational Range		3600	Deg
Potentiometer Gear Diameter		0.0148	m
Linear Range		0.93	m
Track Length		0.99	m
Cart Travel		0.91	m
Potentiometer Sensitivity	K_s	10.7	V/m

TABLE A.2. The parameter values for inverted pendulum.

Parameter	Symbol	Value	Unit
Pendulum Length	L	0.61	m
Pendulum Mass	m	0.21	Kg
Angle Sensor Resistance	R_s	10000	Ω
Bias Resistors	R_b	0	Ω
Bias Voltage	V_b	+/- 12	V
Sensitivity	K_s	0.068	V/Deg
Range		+/- 26	Deg

TABLE A.3. The parameter values for ball and the beam [22].

Parameter	Symbol	Value	Unit
Beam Length	L	0.4318	m
Lever Radius	r	0.0254	m
Motor to Lever Gear Ratio	K_{gr}	0.2:1	
Overall Beam Inertia	J_{eq}	0.0029	Kgm^2
Sensor Range	R_s	2000	Ω
Bias Voltage	V_b	+/- 12	V
Bias Resistors	R_b	1000	Ω
Sensitivity (biased +/- 12 V)	S	23	V/m
Range		+/- 0.215	m

APPENDIX B

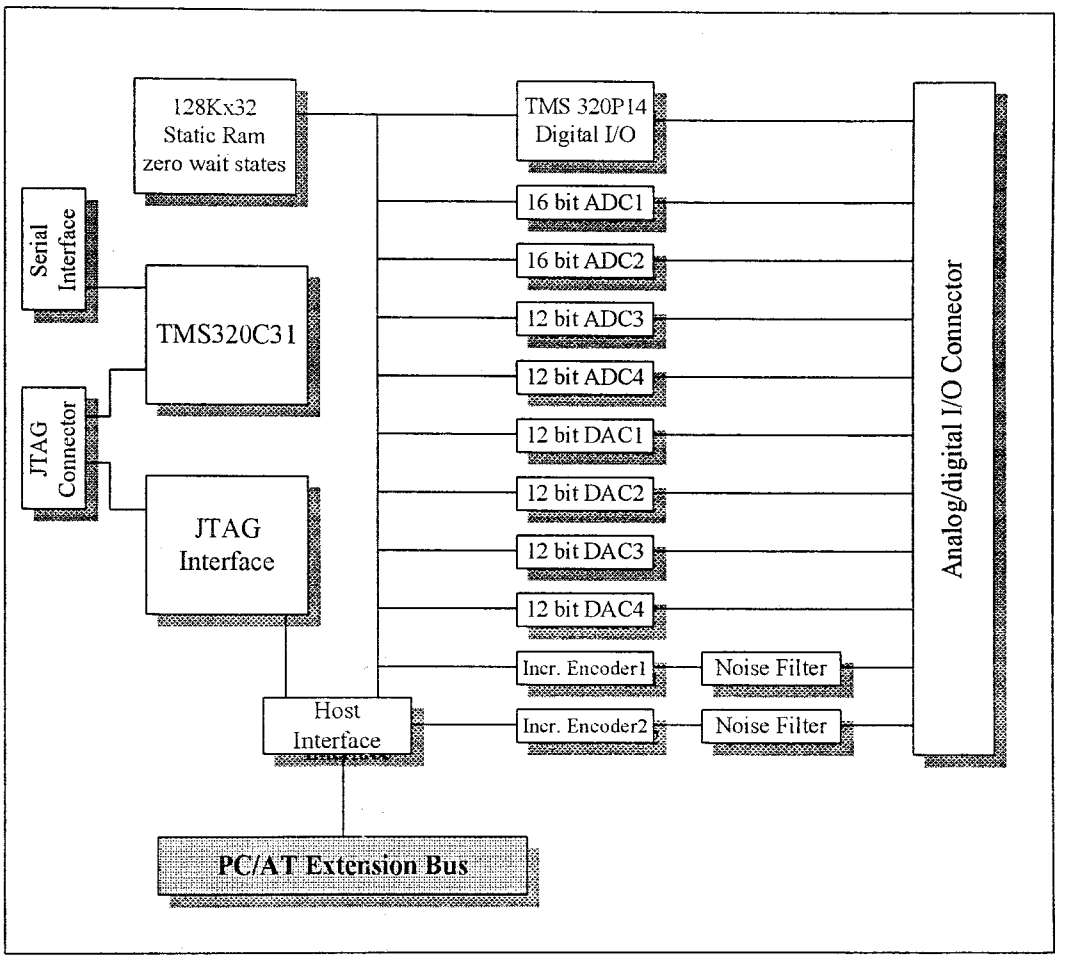


FIGURE B.1. The block diagram of DS1102. [23]

The technical specifications of the hardware used in the X-Garden.

The PC that is hosting the DSP is a 486DX/66 HP Vectra VL2 with 16 MB RAM and 325MB hard disk.

The DSP board used is DS1102 of dSPACE Company. The DSP is built around the Texas Instruments TMS320C31 floating-point Digital Signal Processor (DSP).

It contains 128K Words memory and decorated with several peripheral subsystems to support a wide range of DSP applications. Figure B.1. presents the block diagram of DS1102.

APPENDIX C

In this appendix, the templates of C and m files are presented. They have to be written with respect to this format. They can be found on the directory,

C:\X_Garden\Template.

A C code has to be associated with an m-file. The lack of either one will cause failure in the application. They both should have the same name with as *.c and *.m respectively and they should be located in the directory,

c:\x_garden\members\s.

```
----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<------

function [sys, x0] = strain(t, x, u, flag)

if abs(flag) == 0,
        % This part takes care of all initialization; it is used only once.
        % **** In this template system ****
        % The sizes vector is six elements long, and it looks like this:

        % sizes(1) = number of continuous states
        sizes(1) = 0;

        % sizes(2) = number of discrete states
        sizes(2) = 0;

        % sizes(3) = number of system outputs (length of output y)
```

```
        sizes(3) =  1;
        % sizes(4) = number of system inputs (length of input u)
        sizes(4) =  -1;

        % sizes(5) = number of discontinuous roots; unused feature, set to zero
        sizes(5) = 0;

        % sizes(6) = direct feedthrough flag; used to detect algebraic loops.
        % Set sizes(6) to 1 if the output y depends directly on the input u.
        % Otherwise, it should be set to 0.
        sizes(6) = 0;

        % Set the initial conditions on the states
        x0 = 0;
        sys = sizes';

elseif flag==3
        sys=0;

else
        % Flags not considered here are treated as unimportant.
        % Notice that since there are no discrete states in this system,
        % there is no need to deal with FLAG==2 or FLAG==4.
        % Output is set to [ ].
        sys = [ ];

end
----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<------
```

The following is the format for writing C codes that are embeddable to Simulink S-functions. Note that "//" is not accepted when dropping notes inside the code or canceling the active command lines. The older notation "/*" should be used at the start point of cancellation and "*/" should be at the end of it.

The places where the user code will be added is shown in the below code. First of all, the name of the C code should be specified at the beginning, which is marked.

The second important thing is to specify the number of inputs and outputs. If the number of inputs are not known "-1" is used.

The three main parts used are "mdlInitializeConditions", which is used if there are any variables that are to be initialized at the very beginning, "mdlUpdate", which is the part any calculation or loops are carried out for updating the system parameters, "mdlOutput", which is the final stage that returns the output values calculated out of the S-function. For a better understanding, the codes presented in the next appendix can also be studied.

```
----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<------

/*
* SFUNTMPL  Template C S-function source file.
*
*  Copyright (c) 1994 by The MathWorks, Inc.
*  All Rights Reserved
*/
/*
* The following #define is used to specify the name of your S-Function.
*
*.You should change the define to include the name of your S-function.
* Simply write the name of your C code there. For example write;
*
*                              #define S_FUNCTION_NAME example
*
* if the name of your C code is example.c
*/

#define S_FUNCTION_NAME your_sfunction_name_here

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */

#include "simstruc.h"
```

```
/* mdlInitializeSizes - initialize the sizes array
*
* The sizes array is used by SIMULINK to determine the S-function block's
* characteristics (number of inputs, outputs, states, etc.).
*/
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);    /* number of continuous states */
    ssSetNumDiscStates(    S, 0);    /* number of discrete states */
    ssSetNumInputs(        S, 0);    /* number of inputs */
    ssSetNumOutputs(       S, 0);    /* number of outputs */
    ssSetDirectFeedThrough(S, 0);    /* direct feedthrough flag */
    ssSetNumSampleTimes(   S, 1);    /* number of sample times */
    ssSetNumInputArgs(     S, 0);    /* number of input arguments */
    ssSetNumRWork(         S, 0);    /* number of real work vector elements */
    ssSetNumIWork(         S, 0);    /* number of integer work vector elements */
    ssSetNumPWork(         S, 0);    /* number of pointer work vector elements */
}

/* mdlInitializeSampleTimes - initialize the sample times array
*
* This function is used to specify the sample time(s) for your S-function.
* If your S-function is continuous, you must specify a sample time of 0.0.
* Sample times must be registered in ascending order.  If your S-function
* is to acquire the sample time of the block that is driving it, you must
* specify the sample time to be INHERITED_SAMPLE_TIME.
*/
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTimeEvent(S, 0, 0.0);
ssSetOffsetTimeEvent(S, 0, 0.0);
    /*
    * SET OTHER SAMPLE TIMES AND OFFSETS HERE
    */
}

/* mdlInitializeConditions - initialize the states
*
* In this function, you should initialize the continuous and discrete
```

```
* states for your S-function block.  The initial states are placed
* in the x0 variable.  You can also perform any other initialization
* activities that your S-function may require.
*/
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
    /* YOUR CODE GOES HERE */
}

/* mdlOutputs - compute the outputs
*
* In this function, you compute the outputs of your S-function
* block.  The outputs are placed in the y variable.
*/
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
    /* YOUR CODE GOES HERE  */
}

/* mdlUpdate - perform action at major integration time step
*
* This function is called once for every major integration time step.
* Discrete states are typically updated here, but this function is useful
* for performing any tasks that should only take place once per integration
* step.
*/
static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
    /* YOUR CODE GOES HERE  */
}

/* mdlDerivatives - compute the derivatives
*
* In this function, you compute the S-function block's derivatives.
* The derivatives are placed in the dx variable.
*/
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
    /* YOUR CODE GOES HERE  */
}
```

```
/* mdlTerminate - called when the simulation is terminated.
*
* In this function, you should perform any actions that are necessary
* at the termination of a simulation.  For example, if memory was allocated
* in mdlInitializeConditions, this is the place to free it.
*/
static void mdlTerminate(SimStruct *S)
{
    /* YOUR CODE GOES HERE  */
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<------

APPENDIX D

This appendix contains the commands made available in X-Garden and the file locations.

D.1. Commands

The commands written during the thesis are collected under the directory

c:\x_garden\commands

Some of these commands are automatically activated by the m-file in use, however, they will be explained for clarity in the following table. The most helpful commands will be **xgarden**, **xps**, **sim** and **graphs**.

TABLE D.1. The commands of X-Garden.

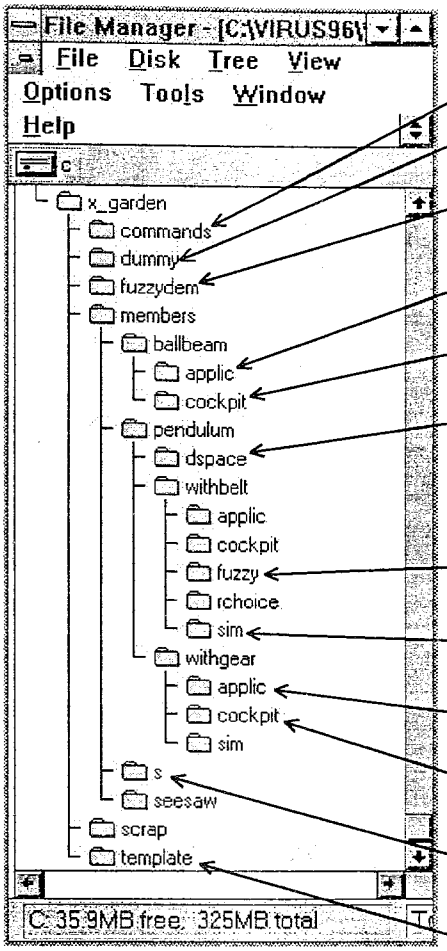
Command Name	Command Description
xgarden	This command will help you to reach the files containing simulations, experiments and their results.
xps	This command leads you to the existing experiments.
sim	This command leads you to the existing simulations.

dummy	Changes the current directory to c:\x_garden\dummy.
dspace	Changes the current directory to c:\x_garden\members\pendulum\dspace.
root	Changes the current directory to c:\matlab.
gearpen	This command initializes the Quanser Inverted Pendulum Parameters.
ballini	This command initializes the Quanser Ball and the Beam Parameters.
valuate	This command initializes the Quanser Inverted Pendulum Parameters.
plotall	This command plots the results of inverted pendulum simulations.
graphs	This command plots the results obtained from the Trace program. The Trace program creates a mat file. This file has to be loaded first, the outputs of the mat file will be plotted by using this command.

D.2. File Locations

The locations of some files are vital, therefore, their places are explained in the following table.

TABLE D.2. The file locations table.



- The X-Garden commands are located.
- The files that are safe to be cleaned are located.
- A fuzzy demo simulator is located. Try it...
- Ball and the Beam Simulink files are located.
- Ball and the Beam Cockpit files are located.
- This directory should be the current directory of Matlab before any download starts. The child files of the download are located.
- M files used in Inverted Pendulum Simulations are located.
- Inverted Pendulum Simulation files are located.
- Inverted Pendulum Application files are located.
- Inverted Pendulum Cockpit files are located.
- C codes and associating M files are located.
- Template files are located.

APPENDIX E

E.1. The Non-Adaptive C-Code Example

```
----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<------

/*                  Written
*                     by
*                 Bugra Koku
*
*         Make your Effort to Contribute
*                 Best Regards
*/

#define S_FUNCTION_NAME fuzga06 /* Write the name of the C code here */

#include "simstruc.h"

/*DEFINE YOUR VARIABLES HERE*/

/*MAXIMUM NUMBER OF PARAMETERS NEEDED TO DEFINE A MEMBERSHIP FUCTION*/
        /*FOR THE CASE OF TRAPEZOID IT IS 4 AND IS THE MAXIMUM VALUE*/
#define         max_par_num                 4

/*UPDATE IF YOU USE MORE ANTECEDENTS IN ANY RULE*/
#define         max_antecedent_num          2

/*UPDATE IF YOU USE MORE MEMBERSHIP FUNCTIONS*/
#define         max_mem_fun_num             3

/*UPDATE IF YOU CHANCE NUMBER OF RULES*/
#define         num_of_rules                9
```

```
/*UPDATE IF YOU CHANGE THE NUMBER OF INPUTS VARIABLES*/
#define         num_of_variables                2

/*UPDATE IF THE OVERALL TOTAL NUMBER OF MEMBERSHIP FUNCTIONS EXEEDS THIS
NUMBER*/
#define         max_total                       20

/*There are some membership types available:
* % shows the number of parameters needed to define that membership function
* U R ugred to add new ones...
*
*                n                %n
*          ----------------------   -----
*         1- TRAPEZOIDAL          %4
*
*         2- BELL SHAPE           %3
*
*         3- GAUSSIAN             %2
*
*         4- SIGMOID              %2
*/

/* 1 IS TRAPEZOIDAL, UPDATE WITH n IF YOU CHANGE*/
int mem_fun_type        =       1;

/* %4 IS TRAPEZOIDAL, UPDATE WITH %n IF YOU CHANGE MEMBERSHIP TYPE*/
int mem_fun_par         =       4;


/*There are some T-norms available:
*
*         #1 Minimum
*
*         #2 Product
*
*         #3 Bounded Product
*
*         #4 Drastic Product
```

```
*/
int t_norm              =       1;      /* UPDATE WITH #n IF YOU CHANGE*/

double rule[num_of_rules][max_antecedent_num+1]={  1,  4,   10,
                                                   1,  5,    7,
                                                   1,  6,    2,
                                                   2,  4,    5,
                                                   2,  5,    0,
                                                   2,  6,   -5,
                                                   3,  4,    2,
                                                   3,  5,   -7,
                                                   3,  6,  -10 };

int num_of_antecedents[num_of_rules]={2, 2, 2, 2, 2, 2, 2, 2, 2};

int num_of_mem_fun[num_of_variables]={3, 3};

double memberships[num_of_variables][max_mem_fun_num][max_par_num]={
        -13.0, -12.00, -1.00, 0.00,     -1.00, 0.00, 0.00, 1.00,        0.00, 1.00, 12.00, 13.00,

        -60.0, -50.00, -5.0, 0.00,      -5.0, 0.00, 0.00, 5.00,         0.00, 5.00, 50.0, 60.0 };

double input[max_total];

double fuzzy_variable[num_of_variables];

double consequent[num_of_rules];

double antecedent[num_of_rules];

/*BEYOND THIS POINT NO ADAPTATION IS REQUIRED*/
/*---------8<---------8<---------8<---------8<---------8<---------8<---------8<---------8<*/


/*WRITE YOUR FUNCTIONS BELOW TO AVOID EXCESSIVE FUNCTION DEFINITIONS*/

/*********************************************************************/
double MAX(double first, double second)
{
```

```
double dummy;
        if(first > second)
                dummy=first;
        else
                dummy=second;
return(dummy);
}
/*********************************************************************/

/*********************************************************************/
double MIN(double first, double second)
{
double dummy;
        if(first < second)
                dummy=first;
        else
                dummy=second;
return(dummy);
}
/*********************************************************************/

/*********************************************************************/
void fuzzify(void)
{

int i, j, k;

double a, b, c, d, sigma, x;

        switch(mem_fun_type)
        {

                /*~-~-~-~-~-~-TRAPEZOIDAL MEMBERSHIP-~-~-~-~-~-~-*/
                case 1:

                for(j=0, i=0; j<num_of_variables; j++)
                {       for(k=0; k<num_of_mem_fun[j]; k++, i++)
                        {
                                a = memberships[j][k][0];
```

```
                b = memberships[j][k][1];
                c = memberships[j][k][2];
                d = memberships[j][k][3];
                x = fuzzy_variable[j];

                if(a==b) a=a-0.01;
                if(c==d) d=d+0.01;

                input[i]= MAX( MIN( MIN( (x-a)/(b-a), 1.0), (d-x)/(d-c) ), 0.0);

        }
}break;


/*~-~-~-~-~-~-~GAUSSIAN MEMBERSHIP-~-~-~-~-~-~-*/
case 2:

for(j=0, i=0; j<num_of_variables; j++)
{       for(k=0; k<num_of_mem_fun[j]; k++, i++)
        {
                c       = memberships[j][k][0];
                sigma   = memberships[j][k][1];
                x       = fuzzy_variable[j];

                input[i]= exp( -0.5*pow( (x-c)/sigma,2) );

        }
}break;


/*~-~-~-~-~-~-~BELL MEMBERSHIP-~-~-~-~-~-~-*/
case 3:

for(j=0, i=0; j<num_of_variables; j++)
{       for(k=0; k<num_of_mem_fun[j]; k++, i++)
        {
                a       = memberships[j][k][0];
                b       = memberships[j][k][1];
                c       = memberships[j][k][2];
```

```
                x       = fuzzy_variable[j];

                input[i]= 1/( 1+ pow( abs( (x-c)/a ), 2*b ) );
        }
}break;


/*~-~-~-~-~-~ SIGMOID MEMBERSHIP-~-~-~-~-~-~-*/
case 4:

for(j=0, i=0; j<num_of_variables; j++)
{       for(k=0; k<num_of_mem_fun[j]; k++, i++)
        {
                a       = memberships[j][k][0];
                c       = memberships[j][k][1];
                x       = fuzzy_variable[j];

                input[i]= 1/( 1+ exp( -a*(x-c) ) );
        }
}break;

default:

/*~-~-~-~-~-~TRAPEZOIDAL MEMBERSHIP-~-~-~-~-~-~-*/

for(j=0, i=0; j<num_of_variables; j++)
{       for(k=0; k<num_of_mem_fun[j]; k++, i++)
        {
                a       = memberships[j][k][0];
                b       = memberships[j][k][1];
                c       = memberships[j][k][2];
                d       = memberships[j][k][3];
                x       = fuzzy_variable[j];

                if(a==b) a=a-0.01;
                if(c==d) d=d+0.01;

                input[i]= MAX( MIN( MIN( (x-a)/(b-a), 1.0), (d-x)/(d-c) ), 0.0);
        }
```

```
                }
        }

}
/*********************************************************************/

/*********************************************************************/
double decide(void)
{
        int i, j;

        double dummy, torque, weight;

/*      for(i=0; i<num_of_rules; i++)
                consequent[i]=rule[i][max_antecedent_num];
*/

        switch(t_norm)
        {

                /*~-~-~-~-~-~MINIMUM T-NORM-~-~-~-~-~-~*/
                case 1:
                for(i=0; i<num_of_rules ; i++)
                {
                        for(j=0, dummy=1.0; j<num_of_antecedents[i] ;j++)
                        {
                                dummy=(dummy < input[((int)(rule[i][j]))-1]) ? dummy :
input[((int)(rule[i][j]))-1] ;

                        }
                                antecedent[i]=dummy;

                }break;


                /*~-~-~-~-~-~PRODUCT T-NORM-~-~-~-~-~-*/
                case 2:
                for(i=0; i<num_of_rules ; i++)
                {
```

```
                for(j=0, dummy=1.0; j<num_of_antecedents[i] ;j++)
                {
                        dummy=dummy*input[((int)(rule[i][j]))-1];

                }
                        antecedent[i]=dummy;

        }break;


        /*~-~-~-~-~-~BOUNDED PRODUCT T-NORM-~-~-~-~-~-~-*/
        case 3:
        for(i=0; i<num_of_rules ; i++)
        {
                for(j=0, dummy=1.0; j<num_of_antecedents[i] ;j++)
                {
                        dummy=(0 < dummy+input[((int)(rule[i][j]))-1]-1 ) ?
dummy+input[((int)(rule[i][j]))-1]-1 : 0 ;


                }
                        antecedent[i]=dummy;

        }break;


        /*~-~-~-~-~-~DRASTIC PRODUCT T-NORM-~-~-~-~-~-~-*/
        case 4:
        for(i=0; i<num_of_rules ; i++)
        {
                for(j=0, dummy=1.0; j<num_of_antecedents[i] ;j++)
                {
                        if (dummy==1) dummy=input[((int)(rule[i][j]))-1];
                        else if (input[((int)(rule[i][j]))-1]==1) dummy=dummy;
                        else dummy=0;

                }
                        antecedent[i]=dummy;
```

```
                }break;


                /*~-~-~-~-~-~PRODUCT T-NORM-~-~-~-~-~-~-*/
                default:
                for(i=0; i<num_of_rules ; i++)
                {
                        for(j=0, dummy=1.0; j<num_of_antecedents[i] ;j++)
                        {
                                dummy=dummy*input[((int)(rule[i][j]))-1];

                        }
                                antecedent[i]=dummy;

                }


        }

        for(i=0, torque=0, weight=0; i<num_of_rules; i++)
        {
                torque=torque+antecedent[i]*consequent[i];
                weight=weight+antecedent[i];
        }

return(torque/weight);

}
/*****************************************************************/


/*
 * mdlInitializeSizes - initialize the sizes array
 *
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */

static void mdlInitializeSizes(SimStruct *S)
```

```
{
   ssSetNumContStates(    S,  0);     /* number of continuous states */
   ssSetNumDiscStates(    S,  0);     /* number of discrete states */
   ssSetNumInputs(        S, -1);     /* number of inputs */
   ssSetNumOutputs(       S, -1);     /* number of outputs */
   ssSetDirectFeedThrough(S,  0);     /* direct feedthrough flag */
   ssSetNumSampleTimes(   S,  1);     /* number of sample times */
   ssSetNumInputArgs(     S,  0);     /* number of input arguments */
   ssSetNumRWork(         S,  0);     /* number of real work vector elements */
   ssSetNumIWork(         S,  0);     /* number of integer work vector elements */
   ssSetNumPWork(         S,  0);     /* number of pointer work vector elements */
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 *
 * This function is used to specify the sample time(s) for your S-function.
 * If your S-function is continuous, you must specify a sample time of 0.0.
 * Sample times must be registered in ascending order.  If your S-function
 * is to acquire the sample time of the block that is driving it, you must
 * specify the sample time to be INHERITED_SAMPLE_TIME.
 */

static void mdlInitializeSampleTimes(SimStruct *S)
{
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);

   /*
    * SET OTHER SAMPLE TIMES AND OFFSETS HERE
    */
}

/*
 * mdlInitializeConditions - initialize the states
 *
 * In this function, you should initialize the continuous and discrete
 * states for your S-function block.  The initial states are placed
 * in the x0 variable.  You can also perform any other initialization
```

```
 * activities that your S-function may require.
 */

static void mdlInitializeConditions(double *x0, SimStruct *S)
{

}

/*
 * mdlOutputs - compute the outputs
 *
 * In this function, you compute the outputs of your S-function
 * block.  The outputs are placed in the y variable.
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{

        y[0]= decide();
}

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
        int i;

        for(i=0; i<num_of_variables; i++)
                fuzzy_variable[i]=u[i];
        for(i=0; i<num_of_rules; i++)
                consequent[i]=u[i+num_of_variables] ;
```

```
        fuzzify();

}

/*
 * mdlDerivatives - compute the derivatives
 *
 * In this function, you compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */

static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
   /*
    * YOUR CODE GOES HERE
    */
}

/*
 * mdlTerminate - called when the simulation is terminated.
 *
 * In this function, you should perform any actions that are necessary
 * at the termination of a simulation.  For example, if memory was allocated
 * in mdlInitializeConditions, this is the place to free it.
 */

static void mdlTerminate(SimStruct *S)
{
   /*
    * YOUR CODE GOES HERE
    */
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
----8<----------8<----------8<----------8<----------8<----------8<----------8<------
```

E.2. The Adaptive C-Code Example

```
----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<------

/*                  Written
*                     by
*                  Bugra Koku
*
*          Make your Effort to Contribute
*                  Best Regards
*/

#define S_FUNCTION_NAME cartide /* Write the name of the C code here */
#include "simstruc.h"

/*DEFINE YOUR VARIABLES HERE*/

#define GAUSSIAN exp( -( (x[i]-X[j][i])/S[j][i] )*( (x[i]-X[j][i])/S[j][i] ) )

#define number_of_inputs        2

#define number_of_rules         25

#define pi                      3.141592654

int loop_until          =       3;

int     MM              =       number_of_rules;

int     NN              =       number_of_inputs;

double  xx[number_of_inputs];   /*Input Vector*/

double  dd;     /*Desired Value*/
```

```
double XX[number_of_rules][number_of_inputs]={ -4,   -4,
                                               -4,   -2,
                                               -4,    0,
                                               -4,    2,
                                               -4,    4,
                                               -2,   -4,
                                               -2,   -2,
                                               -2,    0,
                                                2,    2,
                                               -2,    4,
                                                0,   -4,
                                                0,   -2,
                                                0,    0,
                                                0,    2,
                                                0,    4,
                                                2,   -4,
                                                2,   -2,
                                                2,    0,
                                                2,    2,
                                                2,    4,
                                                4,   -4,
                                                4,   -2,
                                                4,    0,
                                                4,    2,
                                                4,    4 };

double SS[number_of_rules][number_of_inputs]={  2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
                                                2,    2,
```

```
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2,
2,    2 };
```

```
double  YY[number_of_rules]={-0.2, -0.15, -0.1, -0.05, 0, -0.15, -0.1, -0.05, 0, 0.05, -0.1, -0.05, 0,
0.05, 0.1, -0.05, 0, 0.05, 0.1, 0.15, 0, 0.05, 0.1, 0.15, 0.2 };

double  alpha=0.5;

double  decision, torque, weight, z[number_of_rules];

/*BEYOND THIS POINT NO ADAPTATION IS REQUIRED*/
/*---------8<---------8<---------8<---------8<---------8<---------8<---------8<---------8<*/


/*WRITE YOUR FUNCTIONS BELOW TO AVOID EXCESSIVE FUNCTION DEFINITIONS*/

void feed_forward(void)
{

int j,i;
double dummy;

for(j=0; j<MM; j++)
{
        for(i=0, dummy=1; i<NN; i++)
                dummy= dummy*exp( -( (xx[i]-XX[j][i])/SS[j][i] )*( (xx[i]-XX[j][i])/SS[j][i] ) );
        z[j]=dummy;
}
```

```
  for(i=0, torque=0, weight=0; i<MM; i++)
  {       torque=torque+YY[i]*z[i];
          weight=weight+z[i];
  }

  decision=torque/weight;

  }


  void feed_backward(void)
  {

  int i,j;


  for(j=0; j<MM; j++)
          YY[j]=YY[j]-alpha*(decision-dd)/weight*z[j];

  for(j=0; j<MM; j++)
          for(i=0; i<NN; i++)
          {
                  XX[j][i]=XX[j][i]-alpha*(decision-dd)/weight*(YY[j]-decision)*z[j]*2*(xx[i]-
XX[j][i])/(SS[j][i]*SS[j][i]);
                  SS[j][i]=SS[j][i]-alpha*(decision-dd)/weight*(YY[j]-decision)*z[j]*2*((xx[i]-
XX[j][i])*(xx[i]-XX[j][i]))/(SS[j][i]*SS[j][i]*SS[j][i]);
          }
  }

/*********************************************************************/


/*
 * mdlInitializeSizes - initialize the sizes array
 *
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */
```

```
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S,  0);     /* number of continuous states */
    ssSetNumDiscStates(    S,  0);     /* number of discrete states */
    ssSetNumInputs(        S, -1);     /* number of inputs */
    ssSetNumOutputs(       S, -1);     /* number of outputs */
    ssSetDirectFeedThrough(S,  0);     /* direct feedthrough flag */
    ssSetNumSampleTimes(   S,  1);     /* number of sample times */
    ssSetNumInputArgs(     S,  0);     /* number of input arguments */
    ssSetNumRWork(         S,  0);     /* number of real work vector elements */
    ssSetNumIWork(         S,  0);     /* number of integer work vector elements */
    ssSetNumPWork(         S,  0);     /* number of pointer work vector elements */
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 *
 * This function is used to specify the sample time(s) for your S-function.
 * If your S-function is continuous, you must specify a sample time of 0.0.
 * Sample times must be registered in ascending order.  If your S-function
 * is to acquire the sample time of the block that is driving it, you must
 * specify the sample time to be INHERITED_SAMPLE_TIME.
 */

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTimeEvent(S, 0, 0.0);
    ssSetOffsetTimeEvent(S, 0, 0.0);

    /*
     * SET OTHER SAMPLE TIMES AND OFFSETS HERE
     */
}

/*
 * mdlInitializeConditions - initialize the states
 *
 * In this function, you should initialize the continuous and discrete
```

```
 * states for your S-function block.  The initial states are placed
 * in the x0 variable.  You can also perform any other initialization
 * activities that your S-function may require.
 */

static void mdlInitializeConditions(double *x0, SimStruct *S)
{

}

/*
 * mdlOutputs - compute the outputs
 *
 * In this function, you compute the outputs of your S-function
 * block.  The outputs are placed in the y variable.
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{

int k;

for(k=0; k<loop_until; k++)
{
                feed_forward();
                feed_backward();
}
}

y[0]=decision;

}

/*
 * mdlUpdate - perform action at major integration time step
 *
* This function is called once for every major integration time step.
* Discrete states are typically updated here, but this function is useful
* for performing any tasks that should only take place once per integration
```

```
 * step.
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{

int i;

for(i=0; i<NN ;i++)
        xx[i]=u[i];
dd=u[NN];
alpha=u[NN+1];
}

/*
 * mdlDerivatives - compute the derivatives
 *
 * In this function, you compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */

static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
   /*
    * YOUR CODE GOES HERE
    */
}

/*
 * mdlTerminate - called when the simulation is terminated.
 *
 * In this function, you should perform any actions that are necessary
 * at the termination of a simulation.  For example, if memory was allocated
 * in mdlInitializeConditions, this is the place to free it.
 */

static void mdlTerminate(SimStruct *S)
{
   /*
```

```
    * YOUR CODE GOES HERE
    */
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

----8<----------8<----------8<----------8<----------8<----------8<----------8<----------8<-----

APPENDIX F

Comparison of Soft-Computing and Conventional Methodologies in Control of Motor Drives

[1] M. Önder Efe, [2] A. Buğra Koku, [3] Okyay Kaynak
[3] UNESCO Chair on Mechatronics and Mechatronics Research and Application Center
Phone : ++90-212-287 24 75, Fax : ++90-212-287 24 65,
[1] efemond@boun.edu.tr, [2] koku@boun.edu.tr , [3] kaynak@boun.edu.tr

Abstract - **In this study, the performance of soft-computing methodologies and the conventional controllers are compared for the control of a linear servo system. A number of approaches in both domains are simulated and their performances are assessed in terms of several performance measures. It is seen that the use of soft computing methodologies result in some desirable characteristics.**

I. INTRODUCTION

This study investigates the soft-computing techniques and the learning algorithms imposed by these techniques in comparison with the conventional design methodologies from the system control point of view.

In many control applications soft-computing algorithms seem to be preferable when they are compared to the traditional design methodologies. What make them so attractive are the following: Real systems have, in general, complex and nonlinear structure. Therefore, understanding the behavior of a system and estimating the future behavior are formidable problems for traditional methods which are based on analytical techniques. In fact, the problem stems from the intricacy of the physical problem definition and the categoricity of the computation environment.

In this paper, a linear servo system is used as the test bed. In conventional methodologies, estimated plant parameters are used in the design of the controller, whereas, in the soft-computing methodologies the plant parameters are assumed to be unknown.

In the next section, the servo model used in simulations is introduced. The following section explains the details of the two conventional control techniques used, namely, PD and LQ controls. Section IV is devoted to Soft Computing Methodologies. Dynamical Neural Units (DNU) and Fuzzy Logic controller architectures are explained. Finally, the comparison of these methods are assessed with respect to several performance criteria based on the given simulation results.

II. MOTOR MODEL

As is mentioned earlier, a linear servo system is used in the simulations. The general form of the model for this type of a system is given by (1).

$$\frac{x(s)}{u(s)} = \frac{1}{s(\alpha s + \beta)} \qquad (1)$$

In (1), x(s) is the position of the cart, and u(s) is the applied control input in volts.

The plant model seems simple but in real implementations, uncertainty in plant parameters, and the effect of friction makes the design of a controller difficult.

III. CONVENTIONAL CONTROL STRATEGIES

If the plant and the performance specifications are defined well, it is relatively a simple matter to design a suitable controller. In this study, following parameters are considered as the true values of the linear servo parameters, to which the classical controllers are designed due to.

$$[\alpha \quad \beta] = [0.2600 \quad 4.4700] \qquad (2)$$

In the next subsections, a PD and an LQ control scheme for the linear servo system are developed by using the values given by (2).

A. PD Control Strategy

A proportional and derivative controller is able to obtain a critically damped closed loop behavior. If the control given by (3) is applied to the system, closed loop behavior is obtained as given by (4).

$$u(t) = K_p(x_d(t) - x(t)) - K_d\dot{x}(t) \tag{3}$$

$$\frac{x(s)}{x_d(s)} = \frac{K_p}{\alpha s^2 + (\beta + K_d)s + K_p} \tag{4}$$

where K_p, K_d are proportional and derivative gains respectively. x_d is the desired position trajectory. In order to obtain a critically damped response to a step input with a peak time of 0.5 seconds, the PD gains are set as follows:

$$\begin{bmatrix} K_p & K_d \end{bmatrix} = \begin{bmatrix} 8\pi^2\alpha & 4\pi\alpha - \beta \end{bmatrix} \tag{5}$$

The simulation results are given in Figs. 1,2 and 3.

B. Linear Quadratic Controller

This section concerns the design of a linear quadratic controller. The reason why this method is used here is the fact that the LQ regulator design methodology allows to decide on relative importance of the tracking ability and the control effort with respect to each other. This has a practical importance because the boundaries of the admissible controls are determined by physical limitations.

The motor model can be expressed in state-space form as follows;

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{6}$$

where;

$$A = \begin{bmatrix} -\beta/\alpha & 0 \\ 1 & 0 \end{bmatrix},\ B = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ C = \begin{bmatrix} 0 & 1/\alpha \end{bmatrix} \tag{7}$$

$$J = \frac{1}{2}\int_{to}^{tf} \left(\| x - r \|_Q^2 + \| u \|_R^2 \right) dt \tag{8}$$

Minimization of the cost function in (8) yields the following control which is obtained from the solution of the Riccatti equation for the steady state case.

$$u = -Kx + Sr \tag{9}$$

where;

$$K = [5.0715\ \ 100] \text{ and } S = 100 \tag{10}$$

The simulation results for LQR design are given in Figs. 4, 5 and 6.

IV. SOFT COMPUTING METHODOLOGIES

Recent studies showed that the design of a controller by utilizing neural networks and fuzzy logic is more attractive for several reasons. Firstly, the adaptive behavior of these approaches leads to the learning of dynamical properties of the system under control. Consequently, better fulfillment of the performance specifications can be achieved.

Fuzzy logic processes the data obtained from the control system verbally and operates on linguistic variables, whereas in the neural network approach, the data processing is carried out on the physical values of the system parameters. The most striking aspect of these methods is that the former reflects the experience of an expert to controller and the latter imitates the human brain activity. In both cases a mathematically tractable system model and an admissible controller are sought.

In this section, two approaches for the control of the DC motor drive introduced in the second section are presented.

A. Dynamical Neural Networks for Control

Dynamical neural networks are composed of dynamical neural units (DNU) which possess a second order discrete system and an output sigmoidal nonlinearity. The neuron model is comprised of synaptic and somatic parts and adaptation is carried out on the coefficients of this second order block (synaptic part) and the on the slope of its nonlinear activation function (somatic part).

The topology of a single dynamical neural unit consists of delay elements, feedforward and feedback synaptic weights and a nonlinear somatic operator. The architecture of the DNU model is illustrated in Fig 7. The difference equation which describes the behavior of the second order dynamical structure is given in (11) in which $v_1(k)$, $x(k) \in R^1$. Similarly, the pulse transfer function of this part can be given by (12).

$$\begin{aligned} v_1(k) = &- b_1 v_1(k-1) - b_2 v_1(k-2) + a_0 x(k) + \\ &a_1 x(k-1) + a_2 x(k-2) \end{aligned} \tag{11}$$

$$T(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \quad (12)$$

The output of the DNU is evaluated as follows;

$$v(k) = g_s v_1(k) \quad (13)$$

$$u(k) = \Psi(v(k)) = \tanh(g_s v_1(k)) \quad (14)$$

The control objective is based on the minimization of the instantaneous error evaluated at the output of the system. The cost function which is to be minimized is defined by (16) in which E denotes the expectation operator. If w denotes the parameters of a single DNU, the update rule can be given by;

$$w_{k+1} = w_k - \mu \frac{\partial J}{\partial w} \quad (15)$$

$$J = \frac{1}{2} E\left(e^2(k)\right) \quad (16)$$

$$e(k) = y_d(k) - u(k) \quad (17)$$

More compactly, the parameter update rule is given by (18) through (20) where i = 0,1,2 and j = 1,2 ;

$$\Delta a_i(k+1) = \mu_{a_i} E\left(e(k)\, g_s^2(k)\, \mathrm{sech}^2[v_1(k)]\, x(k-i)\right) \quad (18)$$

$$\Delta b_j(k+1) = -\mu_{b_j} E\left(e(k)\, g_s^2(k)\, \mathrm{sech}^2[v_1(k)]\, v_1(k-j)\right) \quad (19)$$

$$\Delta g_s(k+1) = g_s(k)\mu_{g_s} E\left(e(k)\, \mathrm{sech}^2[v_1(k)]\, v_1(k)\right) \quad (20)$$

In the parameter update equations, the coefficients μ_{ai}, μ_{bj} and μ_{gs}, denote the step size for the corresponding parameter and are chosen to be constant throughout a simulation. In Fig. 8, DNU layer includes the desired number of individual DNU blocks whose inputs are connected together and whose outputs are added to form the control u(k). Depending on the magnitude of the output error, the algorithm updates the feedforward and feedback weights and the gain of the nonlinear activation functions of each dynamical neural unit in the DNU layer. The derivation of the algorithm is given in [1] in detail. Simulation results are presented in Figs. 9, 10 and 11.

B. Fuzzy Control of Motor Drives

Fuzzy logic based systems are well known with the property of reflecting the expert knowledge to a system. In terms of control, this is done via exporting our intuitive feeling or experience to the controller.

Fuzzy logic has proved to be very convenient for designing controllers where the designer has some experience or information about the plant to be controlled. However, the major problem of parameter tuning is still there, due to the fact that, converting the qualitative information to quantitative control action is not very easy.

In this paper, a fuzzy logic identifier which is presented similar to a neural network structure is used. To overcome the above mentioned problem of parameter tuning, the well known backpropagation algorithm is used.

A fuzzy logic system with, product inference rule, singleton fuzzifier and Gaussian membership function given in (21) can be represented as a three layer feedforward network and can serve as a universal approximator [2],

$$f(\underline{x}) = \frac{\sum_{k=1}^{M} \bar{y}^k \left(\prod_{i=1}^{n} \mu_{F_i^k}(x_i) \right)}{\sum_{k=1}^{M} \left(\prod_{i=1}^{n} \mu_{F_i^k}(x_i) \right)} \quad (21)$$

where, k=1,2,...,M and i=1,2,...,n. $\underline{x}$ is the input vector of size n and $\bar{y}$ is a vector of size M containing the center values of the center average defuzzifier. There are M rules governing this fuzzy logic system. Finally, $\mu_{F_i^k}$ is the Gaussian membership function for the i^{th} input in the k^{th} rule.

The above mentioned three layer feedforward representation of the fuzzy logic system is given in Fig. 12.

This way of representing the fuzzy system enables us to use the gradient descent methodology to update the two parameters c_i^k and σ_i^k of the Gaussian and the centers of defuzzifier vector $\bar{y}$.

Let the couple $(\underline{x}^D, f^D)$ represent a desired input output pair. Then, the error can be defined as:

$$e = \frac{1}{2}\left[f(\underline{x}^D) - f^D\right]^2 \quad (22)$$

The error backpropagation results in the following update procedure

$$\bar{y}^k(t+1) = \bar{y}^k(t) - \alpha \left.\frac{\partial e}{\partial \bar{y}^k}\right|_k = \bar{y}^k(t) - \alpha \frac{f - f^D}{b} z^k \quad (23)$$

$$\underline{x}_i^k(t+1) = \underline{x}_i^k(t) - \alpha \left.\frac{\partial e}{\partial \underline{x}_i^k}\right|_k$$

$$\underline{x}_i^k(t+1) = \underline{x}_i^k(t) - \alpha \frac{f - f^D}{b}(\bar{y}^{-k} - f) z^k \frac{2(x_i^D - \bar{x}_i^{-k}(t))}{\sigma_i^{k2}(t)} \quad (24)$$

$$\sigma_i^k(t+1) = \sigma_i^k(t) - \alpha \left. \frac{\partial e}{\partial \underline{x}_i^k} \right|_k$$

$$\sigma_i^k(t+1) = \sigma_i^k(t) - \alpha \frac{f - f^D}{b}(\bar{y}^{-k} - f) z^k \frac{2(x_i^D - \bar{x}_i^{-k}(t))^2}{\sigma_i^{k3}(t)} \quad (25)$$

where t=0,1,.... and α is the learning rate.

The control system is designed to behave as an inverse plant as shown below in Fig. 13, and the simulation results are given in the Figs. 14, 15 and 16.

V. CONCLUSIONS

Based on the extensive simulations carried out on a linear servo system, the performance of soft-computing methodologies seem to have more desirable characteristics in terms of the comparison measures considered.

It is obvious that PD and LQ control strategies are strictly dependent on the system parameters and can not compansate the deficiencies caused either by poor parameter estimation or by time varying parameters. This can be observed in Figs 2. and 5. There is no tendency to reduce the tracking error in time. However, this is not the case in Soft Computing Methodologies based controller design.

It is also to be noted that, learning, which is generally introduced as a very desirable property, has a transient phase which may result in instantaneously large controls or a period of ringing in the control signal.

The comparison of the simulation results are given in Table 1.

The future aim is to improve the transient response of the controller by means of adapting the learning strategy depending on the past history of the controls and system response.

TABLE I
COMPARISON OF CONTROL STRATEGIES

	PD	LQC	DNU	FLC
Tracking performance	M	H	H	H
Applicability to	L	M	H	H
Robustness under	L	L	M	M
Noise reduction	M	L	M	H
Capability of fault	L	M	H	M

(H: High, M: Medium, L: Low)

REFERENCES

[1]. Gupta, M. M., Rao, D. H., "Dynamical Neural Units with Applications to the Control of Unknown Nonlinear Systems", *Journal of Intelligent and Fuzzy Systems*, Vol. 1, No. 1, pp. 73-92, 1993

[2]. Wang, L., Adaptive Fuzzy Systems and Control, Design and Stability Analysis, PTR Prentice Hall, 1994

[3]. Narendra, K. S., Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 4-27, March 1990.

[4]. Hagan, M. T., Menhaj, M. B., "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, pp. 989-993, November 1994.

[5]. Efe, M. Ö., "Identification and Control of Nonlinear Dynamical Systems Using Neural Networks," M.S. Thesis, Boğaziçi University, 1996

[6]. Erbatur K., Kaynak O., Rudas I., "An Inverse Dynamics Based Robot Control Method Using Fuzzy Identifiers", to be presented during 1997 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 16-20 June, 1997, Tokyo, Japan

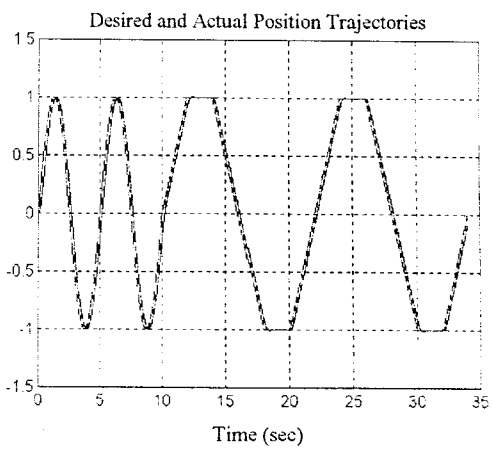


Figure 1. Plant Output and Reference Trajectory for PD Control Strategy

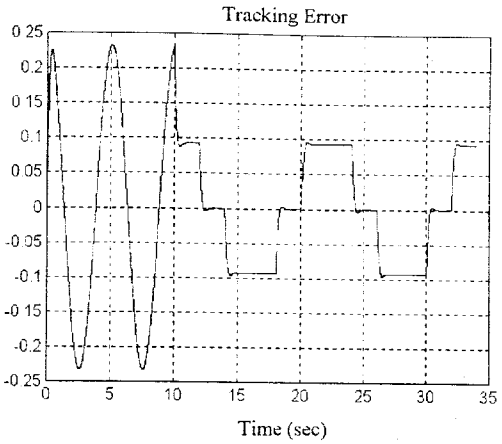


Figure 2. Tracking Error for PD Control Strategy

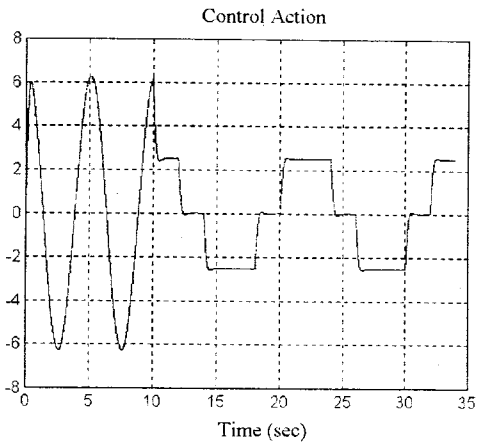


Figure 3. Control Action for PD Control Strategy

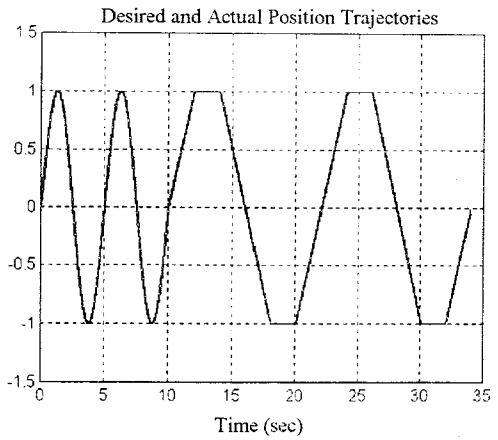


Figure 4. Plant Output and Reference Trajectory for LQR Control Strategy

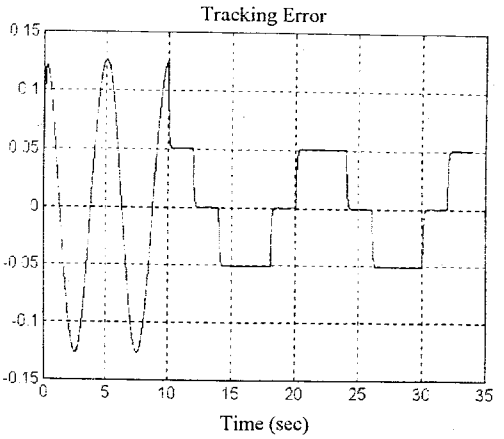


Figure 5. Tracking Error for LQ Control Strategy

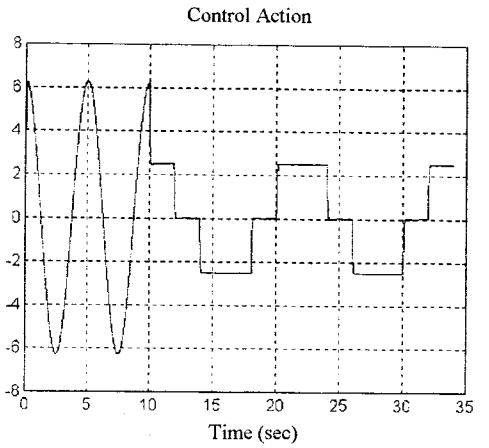


Figure 6. Control Action for LQ Control Strategy

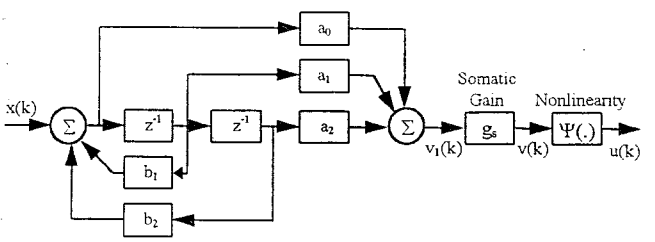


Figure 7. Structure of Dynamical Neural Unit

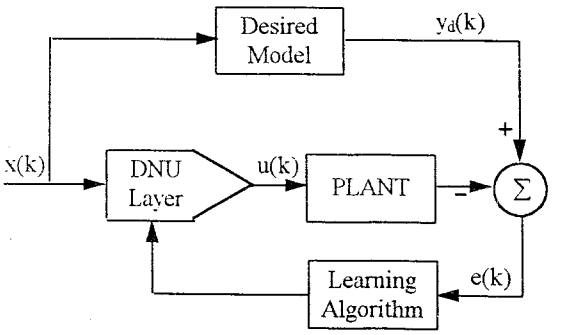


Figure 8. Control System Architecture with Dynamical Neural Unit based Controller

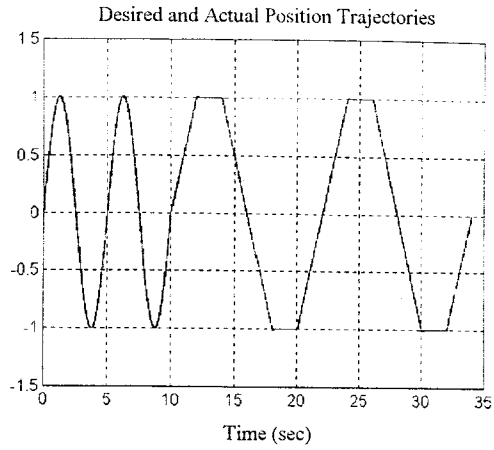


Figure 9. Plant Output and Reference Trajectory for Dynamical Neural Controller

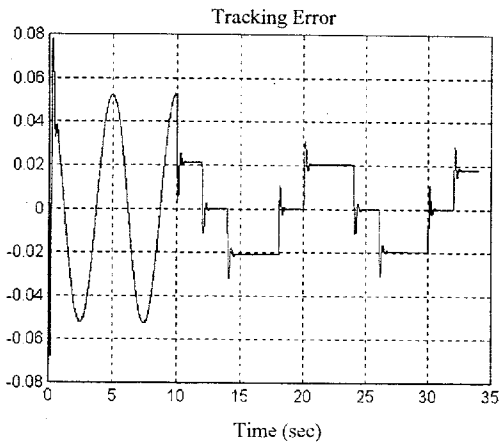


Figure 10. Tracking Error for DNU Control Strategy

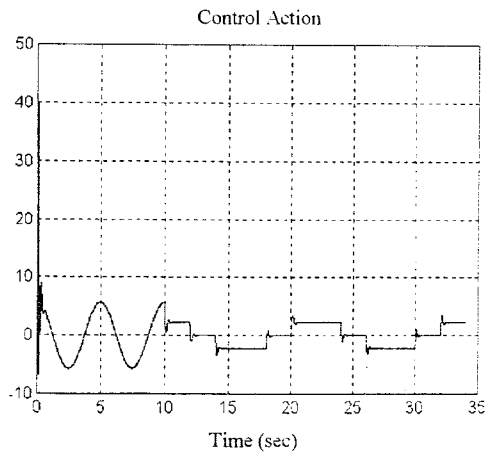


Figure 11. Control Action for DNU Control Strategy

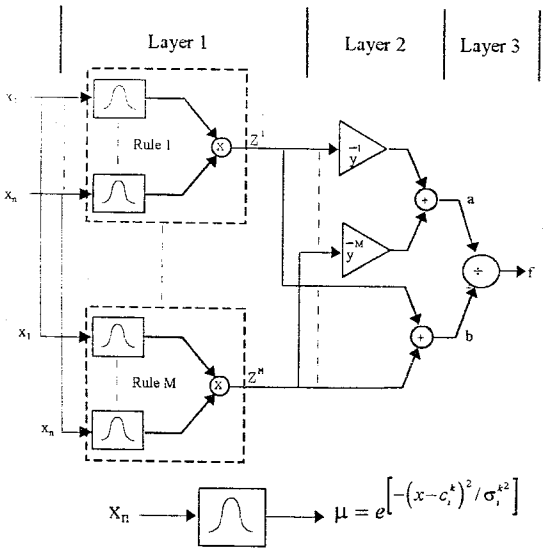


Figure 12. The 3 layer network representation of the fuzzy logic system.

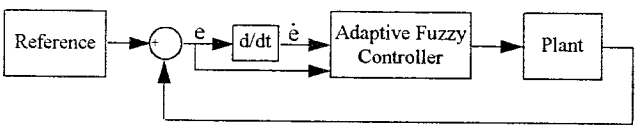


Figure 13. Architecture of Fuzzy Control Strategy

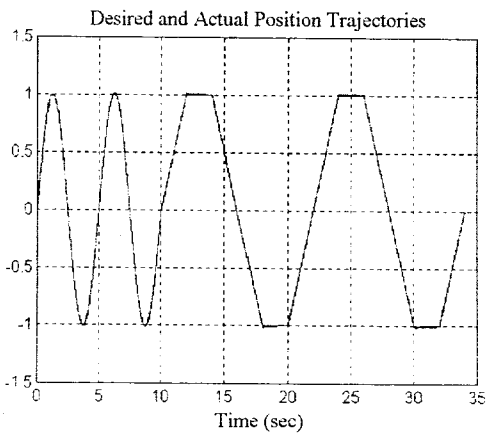


Figure 14. Plant Output and Reference Trajectory for Fuzzy Controller

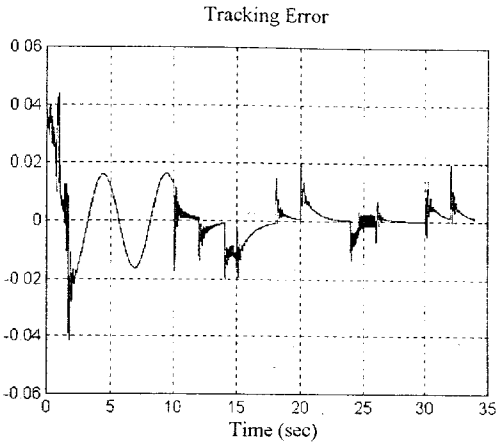


Figure 15. Tracking Error for Fuzzy Control Strategy

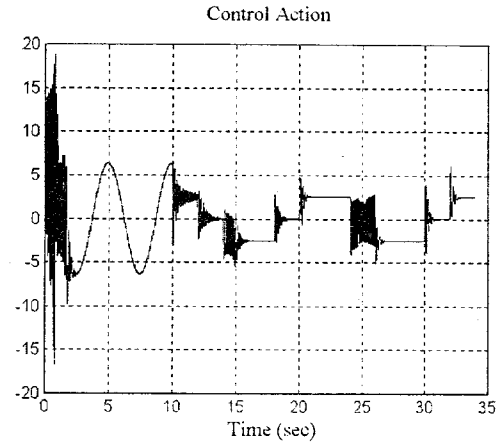


Figure 16. Control Action for Fuzzy Control Strategy

REFERENCES

1. Zadeh, L. A., "Fuzzy Sets," *Information and Control*, Vol. 8, pp. 338-353, 1965.

2. Longman Dictionary of Contemporary English, Longman, Bath, 1978.

3. Zimmerman, H. -J., *Fuzzy Set Theory and Its Applications*, Kluwer Academic Press, 1996.

4. Russell, B., "Vagueness," *Australasian Journal of Philosophy*, No. 1, pp. 84-92, 1923.

5. Einstein, A., "Geometrie und Erfahrung," *Lecture to Prussian Academy*, 1921.

6. Vila, M. A., and Delgado, M., "On Medical Diagnosis Using Possibility Measures," *Fuzzy Sets and Systems*, Vol.10, pp. 211-222, 1983.

7. Cao, H., and Chen, G., "Some Applications of Fuzzy Sets of Meteorological Forecasting," *Fuzzy Sets and Systems*, Vol.9, pp. 1-12, 1983.

8. Sugeno, M., and Nishida, M., "Fuzzy Control of Model Car", *Fuzzy Sets and Systems*, Vol.16, pp. 103-113, 1985.

9. Lee, C. C., "Fuzzy Logic in Control Systems: Fuzzy Logic Controller Part I," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20, No. 2, March/April 1990.

10. Lee, C. C., "Fuzzy Logic in Control Systems: Fuzzy Logic Controller Part II," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20, No. 2, March/April 1990.

11. Ross, T., *Fuzzy Logic with Engineering Applications*, McGraw-Hill, 1995.

12. Jang, J. -S. R., Sun, C. -T., and Mizutani, E., *Neuro-Fuzzy and Soft Computing, A Computational Approach to Learning and Machine Intelligence*, Matlab Curriculum Series, Prentice Hall,1997.

13. Terano, T., Asai, K., and Sugeno, M., *Fuzzy Systems Theory and Its Applications*, Academic Press, 1992.

14. Klir, G. J., and Folger, T. A., *Fuzzy Sets, Uncertainty and Information*, Prentice Hall, 1988.

15. Sugeno, M., "Fuzzy Measures and Fuzzy Integrals: A Survey," in Gupta, M. M., Saridis, G. N., and Gaines, B. R, editors, *Fuzzy Automata and Decision Process*, pp. 89-102, North-Holland, New York, 1977.

16. Yager, R. R., "On a General Class of Fuzzy Connectives," *Fuzzy Sets and Systems*, Vol. 4, pp. 235-242, 1979.

17. Kickert, W. J. M., and Mamdani E. H., "Analysis of a Fuzzy Logic Controller" *Fuzzy Sets and Systems*, Vol. 1, pp. 29-44, 1978.

18. Takagi, T., and Sugeno, M., "Fuzzy Identification of Systems and Its Applications to Modeling and Control," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-15, No. 1,January/February 1985.

19. Wang, L., *Adaptive Fuzzy Systems and Control, Design and Stability Analysis*, PTR Prentice Hall, 1994.

20. Efe, M. Ö., Koku, A. B., and Kaynak, M. O., "Comparison of Soft-Computing and Conventional Methodologies in Control of Motor Drives," *accepted for presentation (but later withdrawn) for PCC'97*, Nagaoka, Japan, August 3-6 1997.

21. Ogata, K., *Designing Linear Control Systems with Matlab*, Matlab Curriculum Series, Prentice Hall, New Jersey,1994.

22. *A Comprehensive and Modular Laboratory for Control Systems Design and Implementation*, Quanser Consulting, 1995.

23. *DSP-CIT eco Hardware Handbook*, Vol. 1.0, dSpace Digital Signal Processing and Control Engineering, GmbH, Paderborn, 1993.