DEVELOPMENT AND IMPLEMENTATION OF A METHODOLOGY FOR FUZZY LOGIC CONTROLLER DESIGN

By

Omar Hussien Ghanayem

A Thesis Submitted for the Degree of Doctor of Philosophy

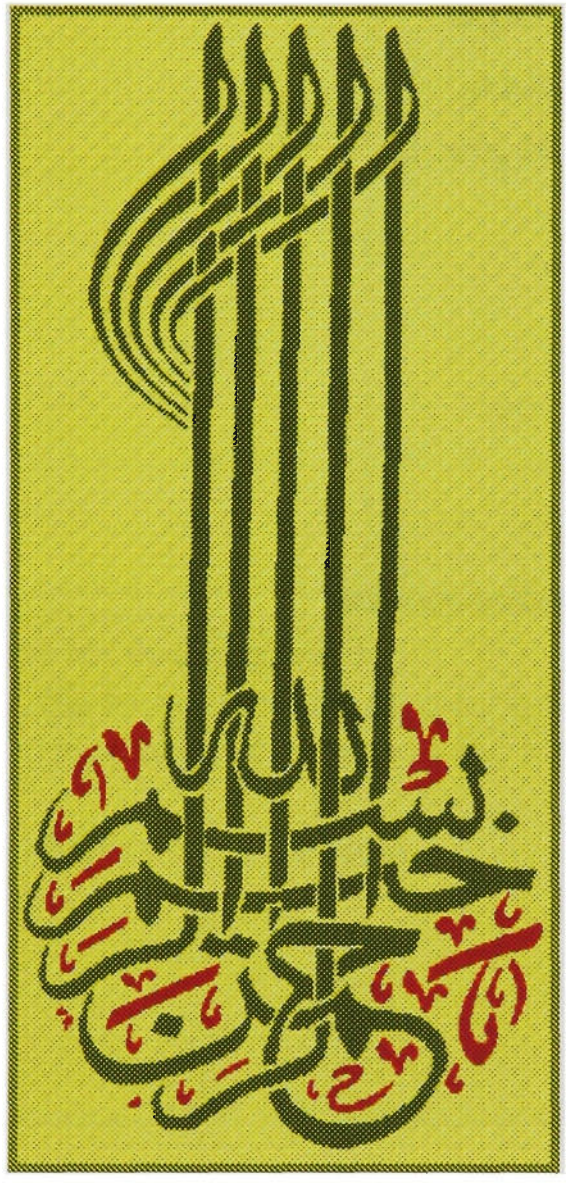
in the

Department of Electrical and Electronic Engineering

Victoria University of Technology

Melbourne, Australia

March 1997



إلى من منحاني معنى الحياة

أمي لمياء، نبع المحبة والحنان

وأبي حسين، معلم الرجولة والالتزام

هذا نتاج زرعكم فاحصدوه

ولتقر أعينكم

ابنكم

عمر غنايم

To the two who gave me meaning to life
My mother, Lamia, the dew of love and compassion
My father, Hussein, the master of manliness and morality

Here is the harvest of what you have sown
To delight your eyes

Your son,

Omar Ghanayem

Acknowledgment

After expressing my thankfulness to Allah "god" for the power, will and inspiration I had to complete this work I would like to thank:

My supervisor, **Dr.** ***Leonid Reznik*** for his support during the three years of my PhD. research. His technical and personal involvement with my work and my future plans were of great support to me.

The postgraduate students in the Department of Electrical and Electronic Engineering who have been very helpful to me in the last three years. They gave me the encouragement to complete this work. I would like to thank them all for the good environment we managed to have during our research. The memories I shared with ***Mahmood, Reza, Nasser, Zahid, Rushan, Mahvir, Valli, Rajan*** and our young friend ***Farhad Zonoozi*** will be always in my mind. My acknowledgment goes to those of my colleagues who have finished their research some time ago, my friends ***Dr. Adrian Stoica***, ***Dr. Iqbal Gondal*** and ***Chanaka Kannangara*** for their advice and support.

Technical and academic staff of the Electrical and Electronic Engineering Department for the technical and professional help they gave me during my research.

My friends and their families for all the support I had during my stay in Australia. Special thanks to ***Imad, Amer, Mohammed*** and ***Mustafa Ashkar*** and their families for being **my** family during the hard times.

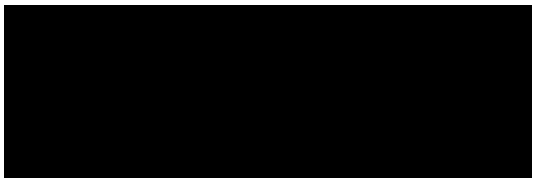
Mr. Wajeeh Khudruj and his family whom treated me as his son and who gave me great help which had and will always have a positive effect in my life.

My friend ***Thuy Trinh*** and her family for the kind help and support.

Last but never least, I would like to give my great gratitude to my mother, ***Lamia Ghanayem*** and my father ***Dr. Hussien Ghanayem*** for the life which they offered me. I would not have been able to reach this point without their financial and moral support. Many thanks "**Mum & Dad**". I would like to thank my sister ***Suha***, and my brothers ***Hisham, Mohammed*** and ***Bassel*** who were of great support to me. Special thanks to my youngest sister, ***Summar***, for all the good jokes she used to cheer me up with on the phone.

STATEMENT OF ORIGINALITY

I hereby declare that the thesis entitled **Development and Implementation of A Methodology for Fuzzy Logic Controller Design** is my own work and has not been submitted previously, in whole or in part in respect of any other academic award.



Omar Ghanayem

Synopsis

The thesis is devoted to development of the methodology of fuzzy logic controller design and implementation.

The thesis introduces a universal fuzzy logic controller structure. The structure proposed combines three main functions and two support functions. The support functions are implementing novel fuzzy logic concepts developed and used in this work. These concepts represent the plant and the controller states during the operation time. The main functions consist of a main fuzzy logic controller refered to as Main-FLC where the initial knowledge about the system is stored. The second main function considers the phase and amplitude of the final control signal. It is called Shay-PA and it implements an implicit on-line tuning and adaptation of the Main-FLC membership functions and rules. The third main function is called Shay-Tune and is concerned with the input and output ranges of the Main-FLC. An adaptation mechanism is implemented for on-line updating of the Main-FLC ranges. Both the support and the main functions operate in conjunction with each other. The decision mechanism implemented in the overall structure is designed with the goals of robustness and reusability in mind.

The thesis presents a practical implementation of the proposed FLC structure in the excitation control of a synchronous generator connected to an infinite bus through a transmission line. A novel fuzzy excitation control system was developed and implemented in a laboratory prototype generator using a digital signal processing board. The reported test results show the efficiency and robustness of the proposed FLC structure.

LIST OF FIGURES

LIST OF TABLES

List of Symbols

WWW	World Wide Web
AI	Artificial Intelligence
NN	Neural Networks
GA	Genitic Algorithms
δ	Rotor angle
ΔT_s	Change in the synchronising torque
ΔT_D	Change in the damping torque
AVR	Automatic Voltage Regulator
PSS	Power System Stabiliser
$\Delta\omega$	Rotor angle speed deviation
ΔP_e	Active power deviation
$\Delta\varpi$	First derivative of $\Delta\omega$
ΔQ	Actual reactive power deviation
FLC	Fuzzy Logic Controller
COG	Centre Of Gravity
FM	Fuzzy Mean defuzzification method
WFM	Weighted Fuzzy Mean defuzzification method
L/R COG	Left/Right COG
PI	Proportional plus Integral controller
PID	Proportional plus Integral plus Differential controller
FIDE	Fuzzy Inference Development Environment
Shay	Stability Handling Algorithm Transient and Steady-State
ref	Reference signal
μ_p	Maximum positive overshoot
μ_n	Maximum negative overshoot
sct	Operational sector vector
fz	Fuzzy input matrix (fuzzification output)
rl	Fuzzy output matrix
F_f	Fired input classes flags
F_r	Fired output classes flag
μ_f	Degree of membership to input classes
S_f	Side of fired input class
μ_r	Degree of membership to output class
S_r	Side for output class
IOS	Input operational status vector
I_R_1	Input under operation mode fuzzy value
I_R_2	Input normal operation mode fuzzy value
I_R_3	Input over operation mode fuzzy value
OOS	Output operational status vector
O_R_1	Output under operation mode fuzzy value
O_R_2	Output normal operation mode fuzzy value
O_R_3	Output over operation mode fuzzy value
G_{in}	Input gain
U_{main}	Output of the Main-FLC
G_{out}	Output gain
E_k	Normalised performance monitor

$\overline{E_k}$	Non-normalised E_k
U_{in}	Input updating signal
U_{out}	Output updating signal
U	Final output of Shay
SISO	Single input single output FLC
U_{PA}	Shay-PA output
θ_{opr}	Operational angle
θ_{sw}	The switching angle
θ_{opr_min}	Minimum range of θ_{opr}
θ_{opr_max}	Maximum range of θ_{opr}
L	Learning time
SFLG	Sector flag
M	Total number of samples
md1	Strength of mode 1
md2	Strength of mode 2
md3	Strength of mode 3
Fix	Fixing the domain range
Dec	Decrementing the domain range
Inc	Increasing the domain range
S	Sectors scale
T	Learning threshold
A	Operational angle scale
SALT	Shay parameters
t_r	Rise time
$\|e_s\|_{av}$	Absolute steady-state error
e_i	Integral of error
U_{fd}	Excitation control signal, field excitation
τ_{do}	Transient open loop time constant
T_m	Mechanical torque
T_J	Developed inertia torque
J	Moment of inertia
B	Coefficient of friction (N.m/rad/sec)
T_B	Torque developed due to B
T_e	Electrical torque
T_D	Damping torque
T_a	Accelerating torque
ω_r	Rotor speed (*rad/sec*)
ω_m	Mechanical speed (*mech. rad/sec*)
v_{ref}	Reference voltage
v_t	Terminal voltage
P-C	Pre-Control stage
M	Angular momentum
H	Normalised inertia constant
P_D	Damping power
$P_{e_{max}}$	Maximum generated electrical power
DSP	Digital signal processing
x_d	Direct-axis synchronous reactance

$x_d^{`}$	Direct-axis transient reactance
x_q	Quadrature-axis synchronous reactance
MIPS	Million Instructions Per Second
MOPS	Million Operations Per Second
SRAM	Static Random Access Memory
DPRAM	Dual-Port Random Access Memory
DM	Daughter Module
A/D	Analogue to Digital
D/A	Digital to Analog
δ_{mech}	Rotor angle, mechanical degrees
$v_{o_{dc}}$	Voltage output from the D/A converter
PC-μp	Host PC driver
DSP-μp	DSP driver

TABLE OF CONTENTS

THESIS STATEMENT

Fuzzy logic control is a fact in modern control applications. Since the early days of its use the technology has attracted a tremendous amount of attention and review. It brought to reality some old expectations in automatic control. Fuzzy logic provided the practical means to construct smart and intelligent controllers. These controllers are able to implement the already available knowledge on how to operate a system in addition to learning and developing their knowledge throughout the process. The controller knowledge is developed via means of dynamic data storage of control rules, objectives and parameters. Moreover fuzzy logic brought to life the possibility of applying modern technology ie. computers and communication systems in a more practical way. Fuzzy logic provides the mechanism for manipulating uncertainty and vagueness impeded in most of real time applications.

However, the success story behind fuzzy logic was shadowed with many problems. Fuzzy logic is well structured by the fuzzy set theory proposed by Zadeh in 1964 [1]. However, practical implementation of fuzzy logic especially in automatic control is not a well structured field as yet. The design procedure of a fuzzy logic controller is in most cases an add-hoc procedure. The designer's experience and trial and error attempts are the main keys to have a working controller. These design procedures made it very hard to predict and forecast the future behaviour of the controller. Evaluation of the designed controller prior to implementation is not possible under this design procedure.

The two contrasts which were the main motivation for this thesis are as follows: 1) The applicability of fuzzy logic in almost every aspect of our lives and what it brings of future anticipation and 2) the lack of structured methodology, design procedures and evaluation tools resulting in uncertainty and arguments against fuzzy logic.

To be able to enjoy the fruits of fuzzy logic a structured design methodology is required. This methodology should give the designer the upper hand in setting the controller's objectives and not to leave the controller performance subject to random choices of parameters.

THESIS GOALS

The main goal of this thesis is to ***develop a universal fuzzy logic controller design methodology and to test its applicability in real life applications***.

To achieve the main goal of the thesis the following sub-goals and objectives were defined:

1) **Develop Methods of Improving the Knowledge Representation in FLC**

   This goal deals with the development of ways to utilise and refine the real time inputs to the controller. The objective is to provide a better representation of the system state that enhances further processing and control.

2) **Develop and Improve Fuzzy Reasoning**

   This goal is concerned with the processing of fuzzy rules which serves as a base for any FLC operation. It aims to improve the performance of the reasoning mechanism in the FLC. The optimum use of the already available knowledge to the FLC is the objective of this goal.

3) **Develop a Structured On-Line FLC Parameter Tuning Procedure**

   This is one of this thesis's main goals. The main objective here is to develop a method of an automatic adaptation to plant and/or environment changes. The designer should not need to rebuild the FLC due to changes in its operational environment. The tuning mechanism should be simple in use and fast in operation.

4) **Develop a Universal FLC Structure**

   A universal FLC structure which can be utilised in different applications is to be produced. The output of this thesis is not supposed to be a single case system. It is supposed to be a universal and reusable FLC structure

for different applications and it should encapsulate all the objectives of the thesis. The developed universal FLC structure should have the minimum possible number of variables that require alterations when the structure is used in different applications.

5) **Develop the Tools of Parameter Tuning and Evaluation of their Influence on the Performance Indicator**

The universal FLC structure should be equipped with tools and parameters that are accessible by the designer. These tools should allow the designer to introduce changes in the FLC design. The change should have a clear effect on the controller performance according to some predefined criteria. This goal is set in order to give the designer a better view of the proposed FLC performance prior to the implementation and an ability to evaluate different design ideas.

6) **Develop an Evaluation Criteria of the Controller Performance**

This goal emerged as a major goal while investigating goal 3. The questions of when, what and how to tune were raised and an answer was set as the goal of defining measurable values used to enhance the tuning process.

7) **Develop Practical Guidelines for FLC Design**

This goal aims at providing practical guidelines for using the universal FLC developed in the thesis (see goal 4) in different control applications. There should be clear directions of how and where to use it as well as what the designer should do to use it in different cases.

8) **Prove the Applicability and Effectiveness of the Proposed Methodology in Real Time Applications**

The objective of this goal is to prove the applicability of the intended universal FLC structure in real industrial applications. To achieve this the proposed methodology should be comprehensively tested by simulation and phototyping and by the design of a real control system implementing

this methodology.

10) Develop an FLC System for the Stability Control of Electrical Power Generator

This goal was derived from goal 9. A classical and very important application chosen was the excitation control of a synchronous generator connected to an infinite bus through a transmission line. A fully fuzzy logic based excitation system is to be built based on the universal FLC structure in goal 4. This goal was set to prove that the intended universal FLC structure is able to replace classical FLCs and conventional controllers in standard and advanced applications. This is to be proven by replacing the power system stabiliser and the automatic voltage regulator in the excitation control loop with one single controller.

11) Develop Methods and Means for Practical Implementation of the Universal FLC Structure Proposed

General purpose software and control protocols are needed to achieve this goal. This software and firmware system will allow implementing the FLC developed in the thesis. The practical implementation should be performed via generalised software and control protocols. These programs and protocols should be applicable to different environments.

THESIS OVERVIEW

The thesis consists of four main sections with a total of 14 chapters and includes 10 appendices. The division of the sections and chapters is based on a logical flow starting with a general analysis of the current situation in FLC design and extending towards the development of a methodology for FLC design, description of a universal FLC structure, its analysis and research, then an implementation and design of the FLC system for the excitation control of electric power generation is presented.

Section 1: Analytical Review of Fuzzy Logic Methodologies and Applications

An analytical review of fuzzy logic control methods based on the literature survey is provided in this section. The early days as well as the fundamentals of fuzzy logic are presented in **chapter 1**. The chapter gives a brief description of the current status of fuzzy logic and its areas of application.

Fuzzy logic in process control is introduced in **chapter 2**. A survey on fuzzy processing, simple and advanced FLC structures, FLC stability and fuzzy hardware is included in the chapter. **Chapter 3** reviews the application of fuzzy logic in power systems with the emphasis on its applications in the excitation control.

Section 2: Development of the Fuzzy Logic Controller Design Methodology Based on Universal Adaptive FLC Structure

This section introduces the proposed universal FLC structure. **Chapter 4** gives an overview of the structure which is called **S**tability **h**andling **A**lgorithm,

transient and steady-state (Shay). The structure comprises three main functions and two support functions.

Chapter 5 describes the support functions where the innovative concepts of measuring the plant and the FLC status are introduced. Shay main functions are introduced in **chapters 6**,7 and **8**. **Chapter 6** introduces the Main-FLC which is one of the main Shay functions. Shay-PA which is the second main function in Shay is presented in **chapter** 7. Shay-PA is responsible for the implicit updating and tuning of the Main-FLC classes and rules. The operational angle concept and its analogy to the proportional plus differential controller algorithm is presented in **chapter** 7. **Chapter 8** introduces Shay-Tune which is the third main function in Shay. Shay-Tune is responsible for the updating and tuning of the input and output ranges of the Main-FLC. The relationship between the main and support functions is clearly defined and explained along with the description of the main functions operations.

Chapter 9 provides the complete evaluation and design considerations based on pre-defined performance criteria. The chapter explains with the aid of examples obtained by computer simulations the effect of each of the designer free variables included in Shay on the system performance.

Section 3: Implementation of A Universal Fuzzy Controller in the Excitation Control

The practical implementation of Shay-FLC is explained in this section. **Chapter 10** explains, in detail, the knowledge engineering techniques used to build up the information required to construct an FLC. The chapter also introduces a novel fuzzy logic based excitation system. The Pre-Control stage is also introduced in this chapter. The Pre-Control stage is where a single parameter, that represents both the rotor angle speed deviation and the terminal voltage, is made available.

Chapter 11 describes in detail the hardware developed and used for the practical implementation of the Shay-FLC in the excitation control. The software developed and applied for the practical implementation is also introduced in **chapter 11**. **Chapter 12** presents the laboratory tests and cases studied along with the real time implementation results.

Section 4: Main Results and Conclusion

The conclusion, project achievements, publications and future work directions are presented in this section.

SECTION 1 ANALYTICAL REVIEW OF FUZZY LOGIC METHODOLOGIES AND APPLICATIONS

Chapter 1 *Review of Fuzzy Logic Historical Development and Different Current Applications*

1.0 INTRODUCTION

Fuzzy logic is becoming one of the main streams in today's research bibliography. There is an ocean of references and resources explaining or dealing with the topic. The IEEE published over 2,266 conference and journal papers related to fuzzy logic in one way or another between the years 1990-1996. The keyword fuzzy logic results in over 13,000 sites[1] in the Internet, which can be easily seen as the only competitor to fuzzy logic in the speed of popularity amongst technical issues this decade. This alone is an enough reason for research to commence in this area. However, one has to ask why? And how this term "fuzzy" became of such great importance and popularity so quickly?.

Fuzzy logic existed along time ago, even before Zadeh's[2] first paper in 1965 [1] when he defined the famous "***fuzzy set***". This is not meant to dishonour Professor Lotfi Zadeh from his novelty. It is just surprising, that it took the human mind so long to realise the existence of such an idea, which is one of the main essence of human life on this planet. On the contrary, around 300 B.C. Aristotles, the famous Greek philosopher, stated that " ***every thing either to be or not to be***". It was this very basic idea where the binary system (the 0 and 1 logic) came from. Things are either true or false. It is very hard to draw a clear line between elderly and youth. Is it true that a man at 60 year of age is old? Can we draw a clear border line here? And if so, what classification does a 59 year old man have? Or do we need to consider the age of retirement to be fuzzy? Considering health and ability to work as a measure? If this is true, we may classify a healthy 60 year old man as young and an ill 25 year old man as old.

A general overview of fuzzy logic is provided in this chapter. The chapter describes the very basic concepts of fuzzy logic. Followed by the early days of opposition to the fuzzy logic idea and the current status of the technology. Description of different domains where fuzzy logic is currently being used concludes the chapter.

---

[1] Infoseek: 13761 sites, AltaVista: 20000 sites and Excite: 20840 sites. The search was done on 1/3/97

[2] **Lotfi A. Zadeh** received the B.S. degree, in 1942, from the University of Teheran, Iran. The M.S. from the Massachusetts institute of Technology (MIT) Cambridge, in 1946, and the Ph.D. from Columbia University, NY, in 1949. He is currently the Emiritus Professor of Electrical Engineering and Computer Science at the University of California, Berkeley.

1.1 DEFINITION OF FUZZY LOGIC BY *Lotfi Zadeh* AS A TOOL FOR REPRESENTING IMPRECISE

Lotfi Zadeh, the father of fuzzy logic, came up with the idea of graded sets and labelled it as the fuzzy sets in 1965 [1]. In this paper, Zadeh wrote

> ***" A fuzzy set is a class of objects with a continuum of grades of membership. Such a set is characterised by a membership (characteristic) function which assigns to each object a grade of membership ranging between zero and one".***

The paper defined and formulated the basis of fuzzy logic. The most important definition was of the fuzzy set in the form of

> ***A fuzzy set (class) A in X is characterised by a membership function ($f_A(x)$) which associates with each point in X a real number in the interval [0,1] with the value ($f_A(x)$) at x representing the "grade of membership" of X in A.***

The paper went into deeper definitions and operations related to the fuzzy sets, among which one can mention are:

Empty fuzzy set

A fuzzy set is empty if and only if its membership function is identically zero.

Equality between two fuzzy sets

Two fuzzy sets A and B are equal, $A=B$, if and only if $f_A(x)=f_B(x)$ for all x in X.

The complement of a fuzzy set

The complement of a fuzzy set A (denoted A') is $f_{A'}=1-f_A$.

Containment of a set into another set

A is contained in B (or equivalently, A is a subset of B or A is smaller than or equal to B) if and only if $f_A \leq f_B$ ($A \subset B \Leftrightarrow f_A \leq f_B$).

Union of two fuzzy sets

The union of two fuzzy sets A and B with membership functions $f_A(x)$ and $f_B(x)$ respectively is a fuzzy set C (written as $C=A \cup B$) whose membership function is related to those of A and B by $f_C(x)=max[f_A(x), f_B(x)]$, $x \in X$.

Intersection of two fuzzy sets

The intersection of two fuzzy sets A and B with respective membership functions $f_A(x)$ and $f_B(x)$ is a fuzzy set C written as $C=A \cap B$ whose membership function is related to those of A and B by $f_C(x)=min[f_A(x), f_B(x)]$, $x \in X$.

Zadeh described the use of some algebraic operations on fuzzy sets. These operations can be derived as the corresponding algebraic operations on the corresponding membership functions. Zadeh explained the fuzzy interpolation of these algebraic operations;

Algebraic product of *A* and *B (AB)*: $f_{AB} = f_A f_B$.

Algebraic sum of *A* and *B (A+B)*: $f_{A+B} = f_A + f_B - f_A \cdot f_B$

Absolute difference between *A* and *B* ($|A-B|$): $f_{|a-b|} = |f_A - f_B|$.

Some other basic concepts and definitions where included in the paper. This paper laid down the foundations for all the fuzzy logic oriented research. Later on in 1973, Zadeh refined these concepts in his other paper [2] in which a vision of where this newly defined prior existing concept is applicable.

1.2 PROBLEMS IN THE EARLY DAYS OF FUZZY LOGIC DEVELOPMENT

The idea proposed by Zadeh did not meet the scientific circles requirements for a new idea at that time. It lacked solid analytical proof and seemed to be so simple in such that it could not be a great hit, in classical terminology. This was the kind of arguments fuzzy logic was faced with in its infancy. This strong opposition forced some of those who had

successfully implemented fuzzy logic towards hiding this fact for commercial reasons. Two good examples in this sense are given by Elkan [3] when he described the talk by Takeo Kanade in IJCAI'91 about Matsushita's camcorder image stabiliser system without mentioning its use of fuzzy logic. The second example was the 1994 Honda Accord automatic transmission with an embedded fuzzy logic controller and how it was advertised as a " grade logic" controller.

1.3 CURRENT STATE OF FUZZY LOGIC

We believe that this kind of tough fuzzy logic childhood was useful for the idea to get stronger as it reaches its mature years. The opposition the idea had proved to be true motivating factor for the true believers in the fuzzy principle. They successfully devoted a tremendous amount of effort in explaining and exploring the superiority of this idea, and to demonstrate its applicability in many aspects of our life.

Zadeh, as expected, played a great role in this contest. He carefully tried to prove that fuzzy logic is closer to the human way of thinking and reasoning than Boolean logic. In his paper titled *"Fuzzy Logic = Computing With Words"* published in 1996 [4] he stated that: "***the main contribution of fuzzy logic is a methodology for computing with words***". Others tried to follow a similar path in proving that it is more 'human like' to reason in the form of IF-THEN rules that have more states than the Aristotles two states logic. Kosko in his book "*Fuzzy Thinking*" [5] compared the eastern culture with its spiritual values and people's ways of thinking to the western one. He came up with the conclusion that the cultural barriers, especially in the scientific circles, were the main reasons why the western circles rejected the idea at first and why the eastern people were those who promoted it. Mamdani [6] reached a similar conclusion.

The culture point of view is really surprising. However, it might be of a great influence. Especially knowing that the real practical success of fuzzy logic was first reported in Japan and some other Asian countries. In spite of the fact that the idea was formulated in the western world capital and center of most research activities, the United Sates of America.

In addition to that the first evidence of its applicability in real life was established in England by Mamdani in 1975 [7].

The picture is much different in the late nineties. Fuzzy logic became a legacy. Western institutes and researchers are doing their best to pick up some of what they have already lost in the field. The west is strongly promoting fuzzy logic. In fact, some people seems to be marketing it in such a way that facts are mixed with expectations and limitations or problems are ignored. This is creating a "fuzzy" idea about fuzzy logic in peoples minds. Fuzzy logic is shown as the ultimate solution and as a replacement for other techniques. One of those fuzzy logic promoters is Earl Cox. In his World Wide Web (WWW) article titled *"The Seven Noble Truths of Fuzzy Logic"* [8] he highlighted some very important issues about fuzzy logic, some of them are true and others fall in the new exaggerating trend in fuzzy literature. Cox "Noble Truths" are presented and discussed in this section because they represent most of what is being propagated about fuzzy logic. These truths can be considered as a good frame for a brief description of the current fuzzy logic state.

Truth One: "*There is nothing fuzzy about fuzzy logic*" [8]

This is a very solid truth and it is important to emphasise this fact. The name "fuzzy" was unfortunately misunderstood by many people. The claims that fuzzy logic violates the solid proven laws or that fuzzy logic produces a vague, or uncertain result are, until now, appearing here or there. Another WWW article [9] gave good support for this truth by stating that " ***fuzzy logic is not about thinking in a fuzzy, ie. Imprecise way, as its opponents would have us to believe, instead, fuzzy logic is about precise thinking about imprecise things***".

Truth Two: "*Fuzzy logic is different from probability*" [8]

The difference between fuzzy logic and probability is clear when we consider the underlying concepts that each attempts to model. Probability is concerned with the unpredictability in the outcome of clearly defined and randomly occurring events. While fuzzy logic is concerned with the ambiguity or undecidability inherent in the description of the event itself. This truth is very

important to publicise, as the relationship between fuzzy logic and the probability theorem is still not very clear for many people.

Truth Three: "*Designing the fuzzy set is very easy*" [8]

This is not fully accurate in practical terms. It is true that the realisation of the problem is much easier using fuzzy sets as it is closer to the human way of thinking. However when it comes to having a working system based on fuzzy logic it needs more than just simple sketching of the problem. The defined fuzzy domains and subdomains have to be tuned to match the problem status and the required solution.

Truth Four: " *Fuzzy systems are stable, easy tuned and can be conventionally validated* " [8]

The stability, tuning and validation of a fuzzy system is not a settled matter yet. However, in process control, people tend to argue the stability as industrial and behaviour requirements and not on means of mapping with conventional techniques. Sugeno and his colleagues [10] provided a good overview for this issue. This fourth truth by Cox is one of marketing statements fuzzy logic promoters are using. However, such claims when not supported with evaluation and validation tools can have negative impact on fuzzy logic progress as a branch of artificial intelligence (AI) and control.

Truth Five: " *Fuzzy systems are different from and complementary to neural networks*" [8]

This is a true fact. The relation between fuzzy logic and neural Networks (NN) is often misunderstood. Both are branches of AI. But, the differences in the processing and data manipulation are too obvious to ignore. One of the reasons of this mix might be due to the joint titles in most of intelligent control conferences and publications, where both fuzzy logic and NNs always appear as if they were the same idea with two different names.

Truth six: "*Fuzzy logic "ain't" just process control any more*" [8]

This can be one of the most powerful points in favour of fuzzy logic. The technique emerged as the solution in many different fields that were almost impossible to be computerised based on the very old Aristotles principle. Refer to section 1.4 in this chapter for various applications of fuzzy logic.

Truth seven: *"Fuzzy logic is a representation and reasoning process"* [8]

Fuzzy logic provided the means to represent and model different environments in a non precise manner, while still conserving the characteristics of the particular environment. It provides this representation so that non-deterministic parameters can be processed in a fuzzy reasoning process that mimics the human reasoning.

1.4 BRIEF REVIEW OF FUZZY LOGIC APPLICATIONS IN DIFFERENT AREAS

Apart from these theoretical and philosophical arguments, fuzzy logic reached its adulthood in practical terms. It is very hard to find any new concept that was widely adopted in many fields of our modern life as quickly as fuzzy logic did. In fact the concept reached much beyond the expectations, and it became a legacy and a good marketing tool to label consumer's goods with *"fuzzy logic inside"*. Sugeno published a book [11] on fuzzy applications and the literature is full of references to many other applications that appear every day.

The most popular field of fuzzy logic applications is in process control, where new dimensions became feasible for control engineers in such a way that the crowned analytical linear control theory is in danger of loosing its crown.

However, the superiority of the concept is in its applicability to some other fields, ie. medicine and business, where they traditionally seem to be very conservative towards new

technologies. Different fields of fuzzy logic applications are briefly presented in this section.

1.4.1 In Business and Finance

Brabazon [12] in his 1996 paper, discussed the advantages of using fuzzy logic in business computer systems. Another description of fuzzy logic emerging in the financial and business area is presented by Attaran [13] where he described the importance of fuzzy logic in removing vagueness by recognising infinite graduation between clear cut extremes. More exploration of the power of fuzzy logic in the field is presented by Roger [14].

Business and financial applications of fuzzy logic involve estimation [15-17], marketing [18], scheduling [19-22], diagnosis [23] and analysis [24,25].

Other financial and business applications of fuzzy logic include insurance rate changes [26], credit card costumes forecasting [27], manufacturing, planning and decision making [28-31], risk assessment [32] and production activity control [33].

1.4.2 In the Medical Field

The very conservative medical field applied fuzzy logic in many areas. Cabello, et al. [34] used fuzzy clustering algorithms for a statistical characterisation of ECG (electrocardiogram) records. The membership function matrix of the prototype of each of the clusters, and the matrices that induce the norms in their environment give useful description over the training set for constructing a feasible classifier for detecting ventricular arrhythmia's. Bounds et al. [35] compared the performance of a multilayer perceptron (MLP) networks to a fuzzy logic based system in the diagnosis of low back pain and sciatica. The comparison resulted in the fuzzy logic based diagnosis system out-performed the MLP and a group of three doctors. Fuzzy logic was also used in a more sophisticated medical application where an adaptive fuzzy controller was used to regulate a blood pressure [36].

1.4.3 In Consumer Products

In his 1996 paper, Berardinis [37] gave a vision of how consumer products will behave in a few years time. He also described how computer and communication technologies will migrate from the domain of high standard applications to simple home appliances allowing gas valves to communicate with gas companies and service centres. This might be very optimistic, but looking at the currently available smart dishwashers, video cameras, VCR's, microwave ovens, there is the feeling that this might happen one day, "soon". Nowadays there is even a fuzzy cooktop controller on the market [38].

Fuzzy logic is also used in some heavy duty applications. For example the automatic train operation system in Sendi municipal subway in Japan. This system is based on a predictive fuzzy control [39]. Also the rule-based fuzzy logic controller used to control the 5 MW Massachusetts Institute of Technology research reactor [40]. A fuzzy logic approach was also used in Brazil for oil recovery [41].

1.4.4 In Diagnosis Systems

The capability of the fuzzy logic principle to manipulate and deal with imprecise data in a similar fashion as a human expert gave it the boost as a great tool used in diagnosis systems. Fuzzy fault diagnosis systems have been used in [42-44]. Different fuzzy diagnosis algorithms have been described in [45,46].

1.4.5 In Scheduling and Process Planning

A fuzzy based inference using quantitative calculations and qualitative knowledge is described in [47] for scheduling an optimum speed-up and load-up operation of fossil power plants. Another fuzzy machine scheduling approach has been introduced by Adampoulos and Papas in [48].

1.4.6 In Expert Systems

Fuzzy expert systems are very popular and they cover a wide range of applications. Hall et al. [49] gave a review of the use of fuzzy expert system and described a reusable fuzzy expert system called *Fess* that was used in several applications. Leung and Lam [50] presented the expert system *Z-II* that can use either Boolean or fuzzy processing by allowing fuzzy "imprecise" and normal terms to be processed. A more complicated fuzzy expert system that performs multi-stage decision-making process was presented by Kurano et al. in [51]. The system uses the Markov-type fuzzy decision processing. In Carajas, Brazil, a practical implementation of a new expert system that employs fuzzy logic techniques to analyse train movement has been reported [52]. Other approaches and applications of fuzzy expert systems can be seen in [53-58].

Chapter 2 *Review of Fuzzy Logic Applications In Process Control*

2.0 INTRODUCTION

> ***"It was Zadeh's paper [1] published at that time which persuaded us to use a fuzzy rule based approach. Between reading and understanding Zadeh's paper and having a working controller took a mere week and it was "surprising" how easy it was to design a rule-based controller"*** **[6].**

These were the words of the first man who ever applied fuzzy logic in real time control applications. Mamdani published a paper about the first known fuzzy control application in 1975 [7]. He described his intelligent controller or as he recently has preferred calling it "*non-analytical*" controller, on a laboratory prototype steam engine during his doctoral research at Queen Mary college in the UK.

The surprising simplicity, as mentioned above, in having a working fuzzy logic controller (FLC) had a negative impact on the technology in its early days. Fuzzy logic controllers (FLCs) did not seem to be a realistic solution in the predominant analytical control theory circles. Questions have been raised on the FLC stability and evaluation methods.

A review of FLCs is given in this chapter with more emphasis on what is still required to be investigated. It is not scientific to go into exaggerated arguments in favour or against FLCs. The fact is that ***FLCs are there and are being used in many applications***.

2.1 FUZZY LOGIC CONTROLLER STRUCTURE AND OPERATION

A very general definition which encompasses the majority of FLC systems may be formulated as follows:

> ***"A FLC is a system which enhances the performance, reliability, and robustness of control by incorporating knowledge which cannot be accommodated in the analytical model upon which the design of a control algorithm is based, and that is usually taken care of by manual modes of operation, or by other safety and ancillary logic mechanisms"*** [59]

The general architecture of an FLC usually consists of three main parts as shown in figure 2.1, ie. fuzzification, fuzzy processing, and defuzzification.

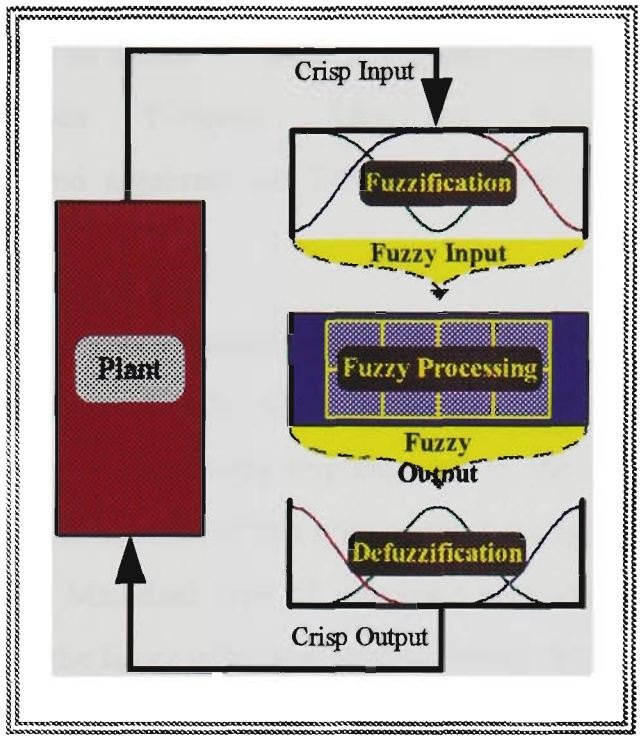


Figure 2.1 FLC layout

2.1.1 Fuzzification

In this phase, the crisp input to the controller is converted into a fuzzy value, symbolic representation. Generally, inputs to the FLC are non-fuzzy in nature, but the data manipulation in a FLC is based on the fuzzy sets theory. Hence, fuzzification of the input is necessary. To transform non-fuzzy (crisp) inputs into fuzzy inputs, membership functions must first be determined. Once membership functions are assigned, fuzzification takes a real input value and compares it with the stored membership function information to produce a fuzzy input value.

2.1.2 Fuzzy Processing

Fuzzy inputs are processed according to the rule base, which is the a set of rules representing the available knowledge in some domain. The inference engine takes the fuzzy value that resulted from the fuzzification process, and produces a fuzzy output through symbolic reasoning based on the former

knowledge stored as a set of rules. To express knowledge by means of fuzzy rules one needs logical connectives. The most used logical connectives in standard fuzzy controllers are : **AND** and **THEN** [60,61]. For implementation of the operators, the so called **T-Norm** is applied. Some examples of the T-Norm are: Zadeh T-Norm; $Min(a,b)$, Lukasiewicz T-Norm $Max(a+b-1,0)$, and algebraic ab T-Norm (derived from the probability theory) [62-64].

Although many inference methods and approaches are reported in the literature, the most frequently used inference methods are: Mamdani (symbolic) type of rules that were implemented in the first applications of fuzzy control. The consequence of this type of rules are symbolic, ie. controller output is large. The Mamdani type of inference produces a fuzzy controller output as a result of the fuzzy inference process, which has to be defuzzified to obtain a numerical controller output. The other type of fuzzy rules is the Sugeno type rule [65] where the consequence of a fuzzy rule is a function (usually linear) of the controller inputs. Successful use of this type of rules in the control of a model car was reported by Sugeno [66,67].

2.1.3 Defuzzification

This last step is the reverse of the fuzzification operation. The fuzzy output from the rule base is transformed into a crisp value, realisable by the plant or system under control. This is done by dividing the output universe of discourse into several intersecting areas (membership functions).

A closer look at the influence of this specific part of the fuzzy controller is worthwhile. The best known defuzzification methods are: Center of Area or Center of Gravity (COG), Fuzzy Mean (FM), Weighted Fuzzy Mean (WFM) and Mean of Maxima defuzzification methods [64,68-71]. Adaptive or adjustable defuzzification methods have become more and more popular in recent applications [72].

This research resulted (in addition to the main results in sections 2 and 3) in the development of a new reasoning method referred to as L/R COG (Left/Right COG) see [73] and appendix *A*.

Choosing the wrong defuzzification method can adversely affect the results achieved by the inference of fuzzy rules. It appears that applying a specific defuzzification method can also affect the characteristics of a fuzzy controller. For example, using the Weighted Fuzzy Mean (WFM) method transforms a Mamdani type controller to a Sugeno type [68,69,74]. Speed and accuracy are the most important criteria for validating any defuzzification technique applied.

2.2 ADVANCED FLC STRUCTURES

Recently, the designers of FLCs started using more complicated structures of FLC to enhance and improve the performance of the controllers. One of these advanced structures is the adaptive FLC structure, where some of the controller parameters are altered and modified throughout the process. The parameters that can be altered are:

1) scaling factors for both inputs and outputs of the FLC,

2) membership functions representing the meaning of linguistic values,

3) the rules set.

When adaptation is applied to the scaling factors or the definition of the membership functions, the FLC is usually called an adaptive self-tuning FLC [75,76]. When altering the rules set, the FLC is called, by some authors, an adaptive self-organising FLC [77-81]. Some adaptive FLC structures are based on a parameter estimation as in [82] and model reference [83]. Some designers of adaptive FLCs started to use NN and Genetic algorithms for on-line tuning of their designs [84,85]. A good overview of adaptive FLCs is given in [86].

Hierarchical schemes are also used in recent FLCs [87,88], where the controller has a structure which changes continuously according to the input signal levels. A hierarchal structure can have fast and strong control strategy under large performance degrading, and a slow and smooth control strategy under steady-state conditions. This scheme improves the system's robustness, and makes it less sensitive to external or internal disturbances. A comparison between adaptive and hierarchical schemes is given in [89].

2.3 ANALYSIS AND DESIGN OF FLCs

The design of a FLC is a very "fuzzy" issue. It is hard to find a well structured design methodology that the designer can follow and still meet the requirements of the design, ***one of this thesis commitments and contributions is the development of a FLC design methodology*** (see section 2). There are already many attempts in this path, however, most of them are either recommendations or insufficient enough to be considered as a general FLC design methodology.

In [90] Kim and Yun, described the use of Genetic Algorithms (GA) in the design of proportional plus integral (PI-like) FLC, where the controller design space is coded in base-7 strings (chromosomes). Each gene matches one of 7 discrete fuzzy values. Another GA based design has been presented [91]. Input and output mapping factors are also used in the design [92].

The big-bang design, is another style in FLC design and the most popular one. As presented in many papers, this approach is based on the designer's experience and own observations. An overview of this style is given in [93]. Some recommendations on the choice of the FLC parameters are given in [94].

The stability of the FLC was discussed in many references. In [95] Sukhbir Singh tried to provide an analytical stability analysis of a discrete FLC by characterising the nonlinearity of the FLC in linear gain sectors. On the other hand, Ghassan et al., used a steady-state analysis approach to evaluate a three-term FLC [96]. Kang [97] provided a generalised design tool for a stable FLC based on Cell-State transitions.

However, the best stability requirement and security is the "*performance stability*" defined by Mamdani in [6] as the "*industrial requirements and the set of rules that encapsulate a stability requirement following the logical approach*". ***This thesis presents a stability tool that is based not only on a philosophical approach but on realising continuous monitoring of the controller and the plant performance***, (see section 2 for details).

2.4 FLC APPLICATIONS

From a practical point of view, one can distinguish between two major classes of FLCs for closed loop control:

1) a class where the FLC is involved in the supervision of the closed loop operation, thus completing and extending the conventional control algorithm,

2) a class where the FLC directly realises the closed loop operation, thus completely replacing the conventional control algorithm [59].

Both types of FLCs were implemented in the control of linear and non-linear systems. The FLC, with a particular choice of membership functions, logical operators and inputs/outputs scales, can emulate a linear controller. From this point of view *linear control can be seen as a subset of fuzzy control*. The best known linear controllers are the proportional plus integral plus differential (PID) controllers. PID-like FLCs were implemented and demonstrated a significant performance improvement over conventional PID controllers [98-100]. The control signal generated from a PID-like FLC is not an optimal average for the whole input space as in the case with conventional PID controllers, but an interpolation between more local optimal averages.

FLCs were implemented in the control process for non-linear systems, such as induction generators, aerospace controllers, target tracking systems, nuclear reactors, power system stabiliser, induction motor control and robot controllers [101-109].

2.5 FUZZY LOGIC HARDWARE

WITH fast growth of fuzzy logic applications in real time control there emerged the need for fast fuzzy processing tools. Two types of fuzzy hardware are available nowadays in the market. The first is the digital hardware with the first processors introduced by Togai Infralogic. The second type of fuzzy hardware is the analog type. Digital processors are performing very well in many fields. However, the digitisation process and the conversion on the other side of the output are consuming large amounts of processor time and power, this is why the analog type is becoming more feasible for future growth. Analog CMOS fuzzy hardware is presented in [110], fuzzy implementations on PLAs, FPGAs, VHDL and VLSI are reported [111-115].

Fuzzy logic based applications are often implemented in higher level languages downloaded to digital signal processing (DSP) boards. Implementing a fuzzy logic controller in higher level programming language is very useful in the prototype stage of the controller development cycle. It is very easy to modify and change the controller's parameters via modifications on the source code.

Chapter 3 *Review of FLC Applications in Power System Engineering*

3.0 INTRODUCTION

Highly integrated power networks were the solution to resolve cost efficiency and natural resources distribution problems faced by electrical power utilities since the early days of power system generation for domestic consumption[1]. However, as these large networks evolved, new problems and considerations for control engineers to maintain the operational requirements of the system emerged.

The electrical power system is highly nonlinear in nature, moreover the system parameters are largely influenced by the operational conditions which combines both generation and distribution. Load variations, faults and other disturbances (can alter the generation unit mode severely) are likely to happen quite often.

Voltage stability and synchronism are the two main problems that should be attended to when dealing with the problem of power system control. Voltage stability is the ability of the system to maintain the terminal voltage of the generator and different busses in the system within certain limits during steady-state conditions. Plus the ability to regain these margins after subjecting the system to any perturbation. Synchronism or rotor angle stability is the ability of the system to maintain synchronism between its rotating components. Both phenomena are linked in one way or another, which adds more complications to the control problem.

Power generation is a mean of energy conversion from mechanical power provided by the prime mover into electrical power by the generator. Altering the mechanical parameters in order to meet different load demands is not the optimal solution in the short term consideration. This is due to the large time constants of the mechanical components of the system. Excitation control is seen as the solution for the short term stability problem. It is known that large control effect can be implemented in a very short time period using small amount of control action through controlling the field of the synchronous generator. The excitation system generally consists of an automatic voltage regulator (AVR) and power system stabiliser (PSS).

[1] The first complete power system was built in 1882 by Thomas Edison. This was a power system consist of a generator, cable, fuse meter and loads in the Pearl Street station in New York City, USA.

Conventional control techniques were implemented in the design of the excitation system. Linear control theory depends on the availability of an accurate plant model. This dependency made it very hard for a conventional controller to be able to achieve the control requirements for a power system under variable operational conditions, which always occur through the operation time. Sophisticated adaptive control schemes were implemented in an attempt to overcome this parameter variation problem. But they faced some difficulties when practically implemented, as such techniques require excessive computational power which is likely to consume significant amounts of time that might decrease the efficiency of the controller. Modelling of the power system is a problem by itself, as steady-state and transient conditions yield totally different operational parameters.

Thus adaptiveness, robustness and simplicity are the main requirements for the electric power system stability problem. These are the characteristics of fuzzy logic controllers (FLC).

This chapter starts with a brief discussion about the power system stability and control objectives. The problem of power system modeling is discussed later. Conventional excitation systems are also discussed. FLC applications in power systems in both the control and non-control areas are described. Detailed descriptions of some of the most popular FLC structures used in electric power system control are presented.

3.1 POWER SYSTEM STABILITY CONTROL PROBLEM DESCRIPTION

Traditionally, power system stability is defined as " ***the ability of the power system to maintain operational equilibrium between different forces under normal operation and to regain this equilibrium following any perturbation or disturbance***" [116]. According to this definition instability means ***loss of equilibrium***.

The equilibrium is maintained by keeping a set of operational factors within certain ranges such that, when these forces operate opposing each other, equilibrium is reached with satisfactory power operation conditions of generated power and load angle.

The analogy of this equilibrium point can be easily realised when considering the power network as an equilibrium between mechanical forces developed by several springs connected to a ball from different angles (figure 3.1). Each spring represents one of the kinds of forces that the power system is subjected to. Any change in each spring "force" effect can result in loss of equilibrium in the system, so, the ball in figure 3.1 will start moving forward and backward. If after this movement the ball regained its equilibrium while still in the domain of the stability circle (the shaded area in the figure), then it can be claimed that stability was regained. Otherwise stability is said to be lost.

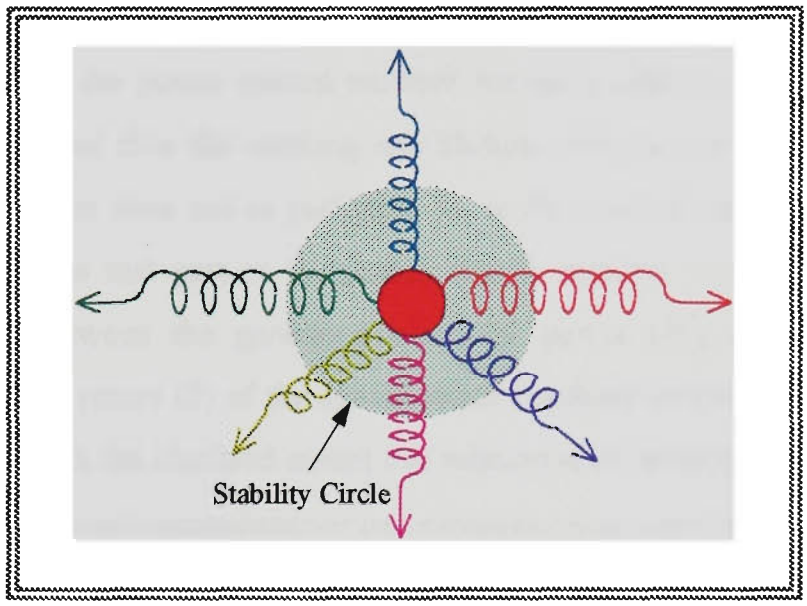


Figure 3.1 power system stability analogy

In 1920, power system stability was recognised as an important problem. Early laboratory stability tests were performed in 1924 and the first field test was conducted in 1925. During these days the stability problem was seen as simple by keeping the delivered power from the generator within certain limits.

In recent years, and with the trend of moving towards higher integrated networks, new problems and considerations arose in power system operation. The stability problem has expanded to new horizons which are related to global networks and subsystems ending with a generating unit. The main consideration in power system stability studies, is to keep the synchronism and to maintain acceptable voltage levels at all busses. Synchronism is maintained via the stability branch called rotor angle stability and the voltage stability is concerned with maintaining the bus voltage levels.

3.1.1 Rotor Angle Stability

Rotor angle stability refers to the *ability of the synchronous machines in the system to maintain synchronism or to stay "in step"*. The system is subject to many forces (torques) under all operational modes. The rotor angle stability requires maintaining equilibrium between these forces and torques within the satisfactory conditions under the steady-state operational conditions. Plus the ability to remain within these acceptable margins after subjecting the system to any type of disturbance.

Disturbances in the power system network normally alter or eliminate one of these torques and thus the existing equilibrium point is lost. The loss of an equilibrium point does not in particular mean the loss of stability. The rotor angle stability is that part of the power system stability concerned with the relationship between the generated electrical power (P_e) and the angular positions of the rotors (δ) of the synchronous machines in the system. For the simple case, with the idealised model this relation is sinusoidal as in figure 3.2.

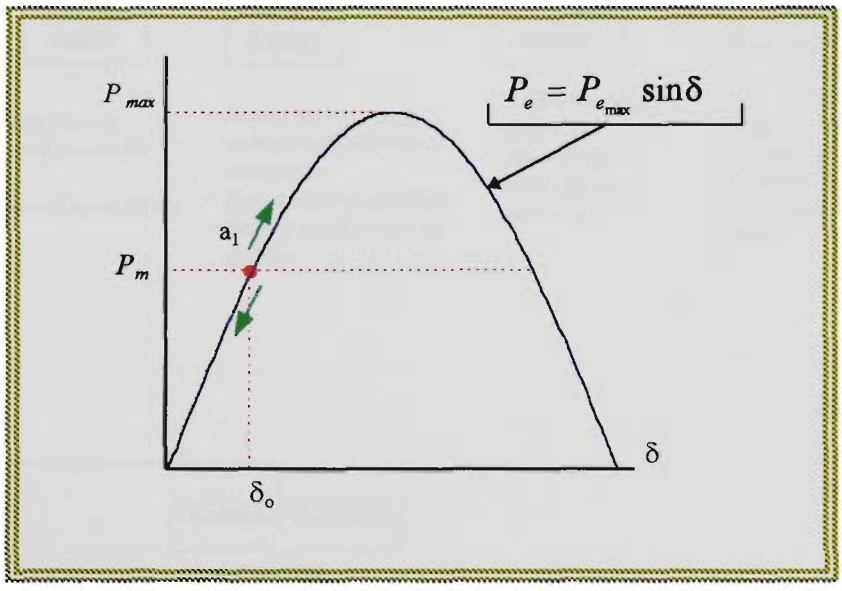


Figure 3.2 power-angle curve

This is a highly nonlinear relationship, with more accurate system components modeling the power angle curve can vary significantly from the sinusoidal wave form shown in figure 3.2.

The deviation in the electrical torque generated by the synchronous generator after any disturbance is a function of two torques, ΔT_s, the synchronous torque component of torque change which is in phase with the perturbation in δ and the damping torque change (ΔT_D). Stability is achieved by the ability of the system to provide sufficient amounts of both torques after the disturbance. Lack or missing of any of these torques will result in an unstable system.

The stability categories and causes are shown in the diagram in figure 3.3 [116].

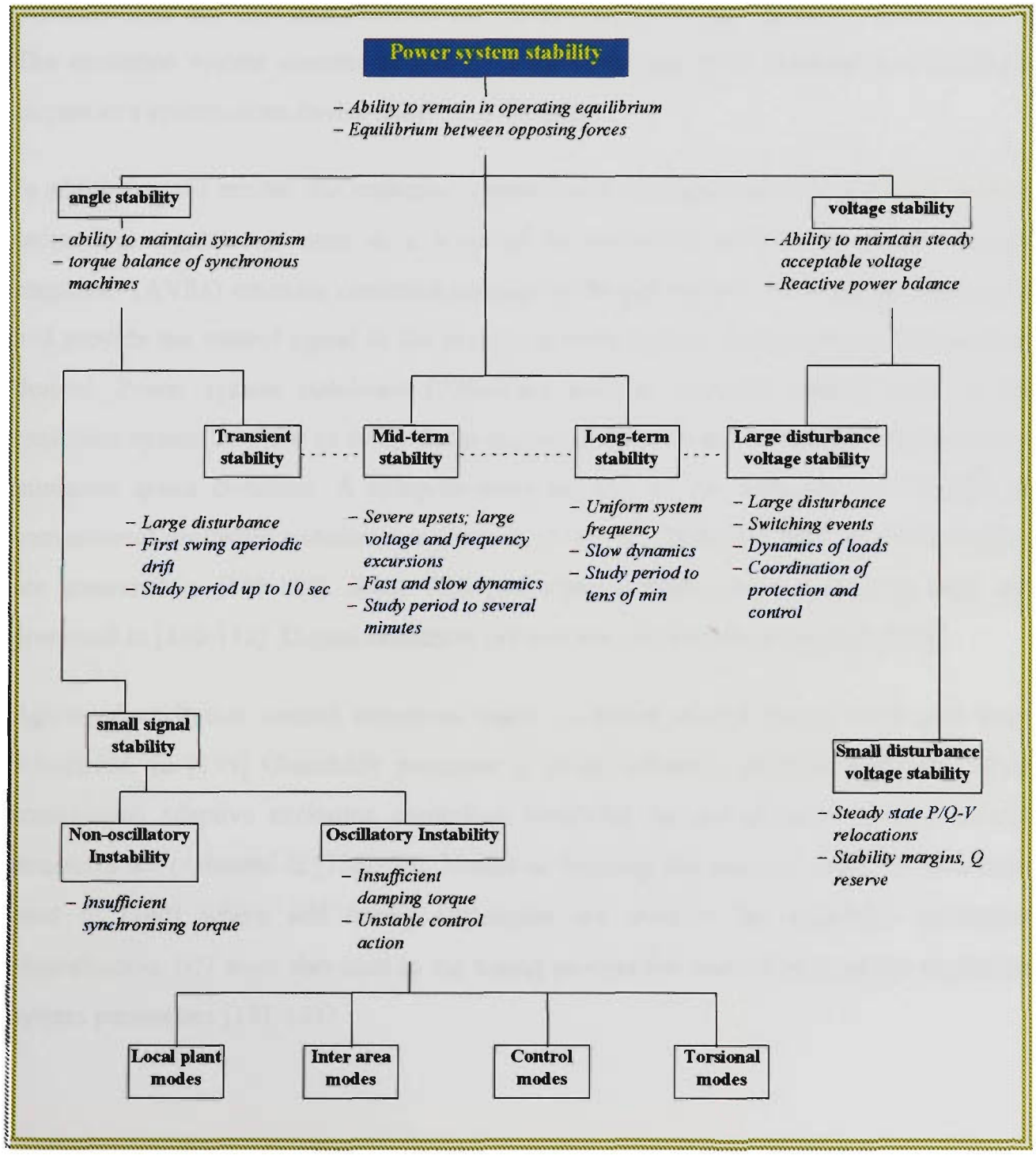


Figure 3.3 electrical power system stability classification

3.2 CONVENTIONAL METHODS FOR STABILITY CONTROL OF POWER SYSTEMS

Most of the control applications in power system generation are concerned with the excitation control, which is seen by power engineers as the best form of short term generation control due to the relatively small time response of the excitation system compared to other components of the power system.

The excitation system is defined as "***the source of field current for the excitation of asynchronous machine and includes the excitor, regulator and manual control***" [117]. The excitation system consists of one or more generating units designed to provide dc current to a synchronous device under consideration.

In addition to the exciter, the excitation system has a regulator and other auxiliary control units. The regulator is seen as a brain of an excitation system. Automatic voltage regulators (AVRs) maintain continuos sensing on the generation unit terminal voltage (v_t) and provide the control signal to the exciter in order to keep the system performance as desired. Power system stabilisers (PSSs) are used as auxiliary control units in the excitation system in order to damp down any oscillations in the rotor angle and to keep a minimum speed deviation. A comprehensive analysis of the advances and benefits of conventional excitation techniques is given in [118-122]. Different AVR and PSS designs are presented in [123-129]. Some PSS performance evaluation and analysis tools are presented in [130-132]. Digital excitation systems are also introduced in [133,134].

Advanced excitation control structures based on linear control theory have also been introduced. In [135] Ghandakly presented a model reference adaptive stabiliser. More complicated adaptive excitation controllers involving the use of multi leveled control structures are presented in [136-139]. Nonlinear learning and adaptive methods have been used in [140] where self tuning techniques are used in the excitation parameter identification. NN were also used in the tuning process (on and off line) of the excitation system parameters [141-144].

3.3 POWER SYSTEM MODELING

Modeling a power system is not an easy task to do, in fact it is almost impossible to have a single sufficient model that can represent the system states due to:

1) the power system is highly nonlinear in nature,

2) the large changes to the systems parameters when the system is perturbed,

3) The huge size of the power system network,

4) The large number of factors that effect the power plant.

Some serious attempts have been proposed in the power systems modeling domain, see [145-147] for more details. Conventional controllers rely on sufficient representation (modeling) of the plant under control. This is why they suffer critical sensitivity when operating under varying plant characteristics. This motivated the trend towards applying fuzzy logic techniques to the field of power system generation and control. FLCs have proved their robustness and do not totally rely on an accurate plant model in the design and testing stages. The following section represents an overview of the already published applications of FLCs in power systems.

3.4 FLCs APPLICATIONS IN POWER SYSTEMS

The application of fuzzy logic in power systems has reached new boundaries in the field. It did not just replace some of the old linear controllers in the control loop of the generation unit. It took a leading role in other areas of a power distribution and management, areas that have been exclusive to human experts and some limited statistical methods.

This section describes the use of fuzzy logic in power systems under two main categories: 1) non-control and 2) control.

3.4.1 Non-Control Applications

The superiority of fuzzy logic is clear in the theory ability to mimic human expert's ways of thinking and reasoning. The competition among large power providers in the huge electric energy market made resources management a major concern of electric power utilities. Forward planning is the best solution for optimal resources usage which enables electric companies can deliver the service at minimum cost. Load forecasting is the backbone in this domain. Fuzzy logic plays an important role in managing the large sums of vague data collected on daily basis in electric utilities in order to provide a future forecast of the load demands as in [148-152]. Moreover, future planning and decision systems based on fuzzy logic have been used in [153,154]. A revolutionary fuzzy logic approach for load dispatch that takes into account environmental concerns as well as economical ones is presented in [155], another fuzzy system that optimises cost with consumer satisfaction and desires is presented in [156].

Operation monitoring and security systems based on fuzzy logic have also been introduced in [157-159] where emergency alert or warning signals are produced according to accessible simple data collection. Fuzzy logic was also used in diagnosing power network faults [160-162].

3.4.2 Control Application

Electric power generation, control and stability enhancement using fuzzy logic algorithms have been a very active research and application area in the last few years. The literature, in most cases, present the comparison between FLCs and the already existing conventional controllers. The dominant area of FLC application (control ones) in power systems is in the stability enhancement using FLC power system stabilisers (PSSs). Different trends in stability and excitation control are presented in sections 3.5 and 3.6.

3.5 DIFFERENT APPROACHES TO FLCs DESIGN AS POWER SYSTEM STABILISERS (PSSs)

It is very well known that providing supplementary control signals in the excitation control loop will improve the stability margins in the power network. Conventional PSS systems, with their model dependency, suffer great degrading in performance under variations in the operation points of the power system. This section gives an overview of three of the most popular FLC structures used for the PSS design.

3.5.1 Rule-Based PSS

In their paper in 1995 [163] El-Metwally and Malik presented a rule-based fuzzy logic power system stabiliser. El-Metwally and Malik used the speed deviation ($\Delta\omega$) and the active power deviation (ΔP_e) as inputs and the stabilising signal (U) as the output in 7x7 two inputs one output rule table. The input/output domains classes used in [163] are shown in figure 3.4.

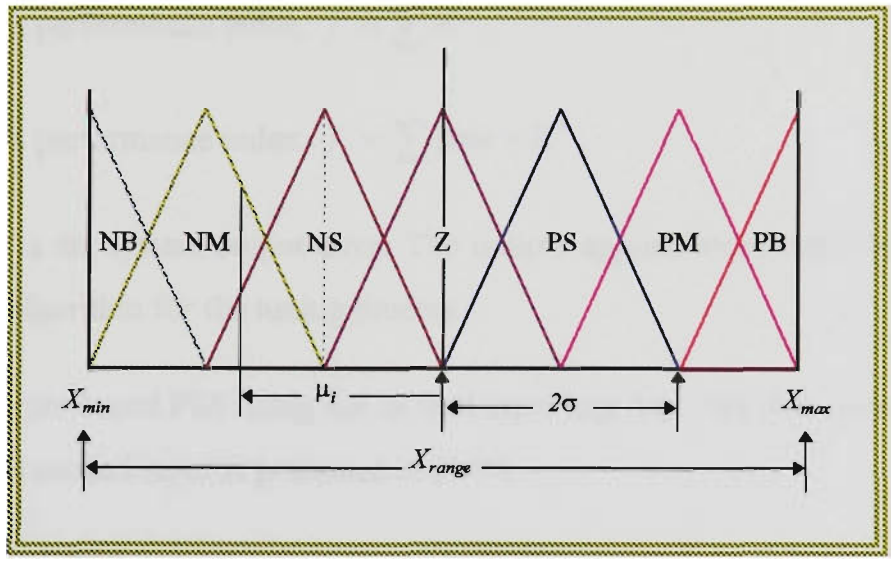


Figure 3.4 input/output domain for the rule-based PSS

The input tuning was represented by two factors (k_1 and k_2) in this PSS where, $k_i = 2/(x_{\max_i} - x_{\min_i})$. The designers used a symmetrical rule table, table 3.1, with *min-max* inference method and COG defuzzification method.

	ΔP_e						
$\Delta\omega$	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NB	NM	NS	Z
NM	NB	NB	NM	NM	NS	Z	PS
NS	NB	NM	NM	NS	Z	PS	PM
Z	NM	NM	NS	Z	PS	PM	PM
PS	NM	NS	Z	PS	PM	PM	PB
PM	NS	Z	PS	PM	PM	PB	PB
PB	Z	PS	PM	PB	PB	PB	PB

Table 3.1 symmetrical rules used in the rule-based PSS

The two input parameters, k_1 and k_2, were tuned off-line based on three criteria:

1) the maximum overshoot, μ_p,

2) a performance index, $j_1 = \sum E^2$,

3) a performance index, $j_2 = \sum time \times E^2$,

where E is the system output error. The indices applied along with the guided training algorithm for the tuning process.

Another rule-based PSS using $\Delta\omega$ as first input and $\Delta\varpi$ (the first derivative of $\Delta\omega$) as its second input is presented in [164].

3.5.2 Fuzzy logic PSS

Hiyama [165] proposed a fuzzy logic PSS based on dividing the phase plane between $\Delta\omega$ and $\Delta\varpi$ into six control sectors shown in figure 3.5.

In this scheme the generator state at time $t=k\Delta T$, $(p(k))$ in the phase plane, is calculated as $p(k)=[\Delta\omega,(\Delta\omega(k)-\Delta\omega(k-1)/\Delta T]$.

The control action objective is to move the point $p(k)$ to the origin as soon as possible while applying six different control rules:

Rule 1: if $p(k) \in$ sector A then $u(k)=G(k)U_{max}$

Rule 2: if $p(k) \in$ sector B then $u(k)=G(k)U_{min}$

Rule 3: if $p(k) \in$ sector C then $u(k)=G(k)U_{min}$

Rule 4: if $p(k) \in$ sector D then $u(k)=g(k)U_{max}$

Rule 5: if $p(k) \in$ sector E then $u(k)=g(k)U_{min}$

Rule 6: if $p(k) \in$ sector F then $u(k)=g(k)U_{min}$

where

$G(k)=D(k)/D_r$ $\quad D(k)<D_r$

$G(k)=1$ $\quad D(k)\geq D_r$

$D(k)=|p(k)|=\sqrt{\Delta\omega^2+\Delta\varpi^2}$.

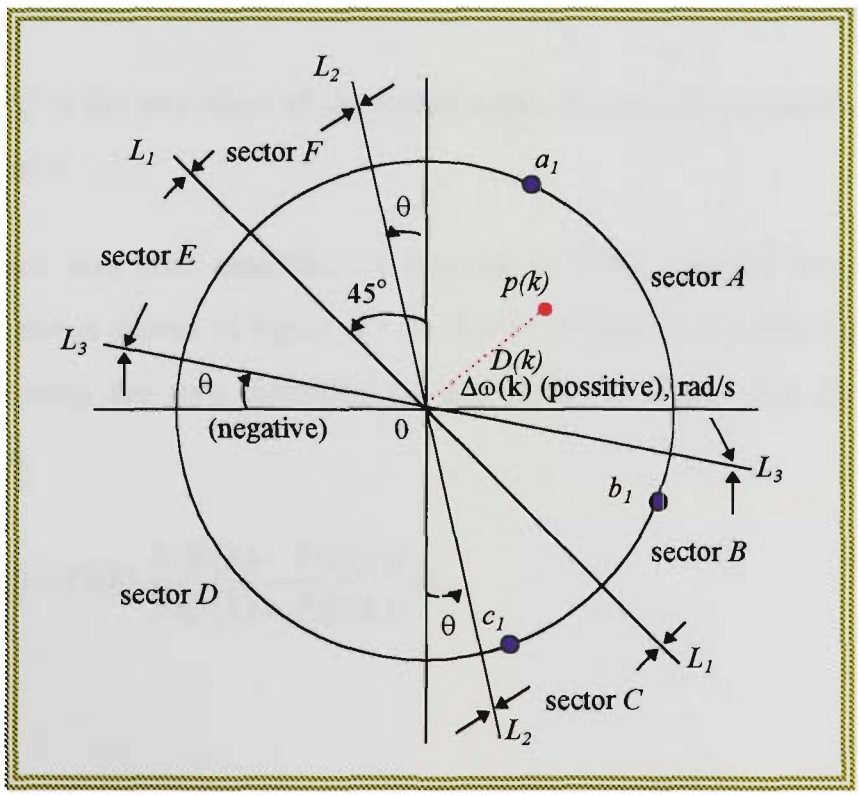


Figure 3.5 Hiyama's sectors in the phase plane

The gain factor dependence is shown in figure 3.6.

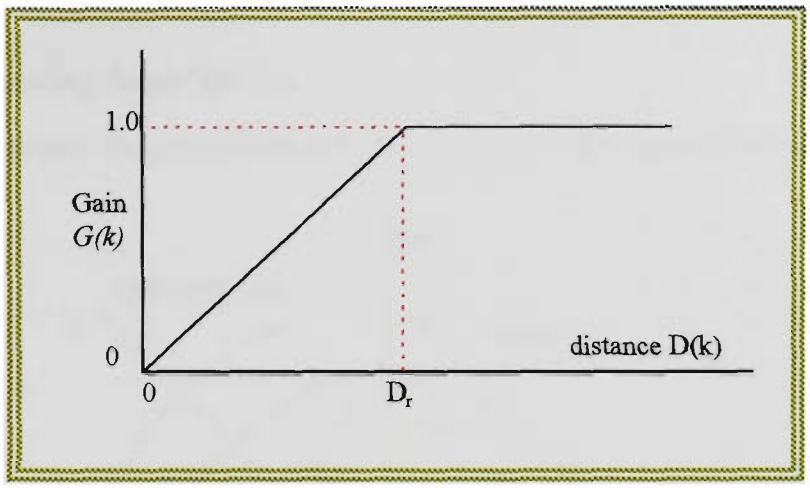


Figure 3.6 gain factor dependence

This scheme requires the parameter optimisation for L_1, L_2 and L_3 in the phase plane in figure 3.5, D_r in figure 3.6, U_{min} and U_{max}. Hiyama used two discrete quadratic performance indices for this purpose, they are:

$$j_1 = \sum_{k=0}^{M} \Delta\delta(k)^2 = \sum_{k=0}^{M} \{\delta(k) - \delta_o\}^2$$

$$j_2 = \sum_{k=0}^{M} \{t\Delta\omega(k)\}^2$$

where $\Delta\delta(t)$ is the deviation of the phase angle between the generator and the infinite busbar.

This scheme was later modified by Hiyama in [166] and the $\Delta\omega$-$\Delta\varpi$ phase plane became as shown in figure 3.7. In this modification the stabilising signal ($U(k)$) is using the two functions (P and N) as in figure 3.8 for its final calculation,

$$U(k) = G(k)\frac{N(\theta(k) - P(\theta(k))}{N(\theta(k) + P(\theta(k))}U_{\max}$$

where

$$\theta(k) = \tan^{-1} \frac{A_s \times \Delta\varpi(k)}{\Delta\omega(k)}$$

and A_s is a scaling factor for $\Delta\varpi$.

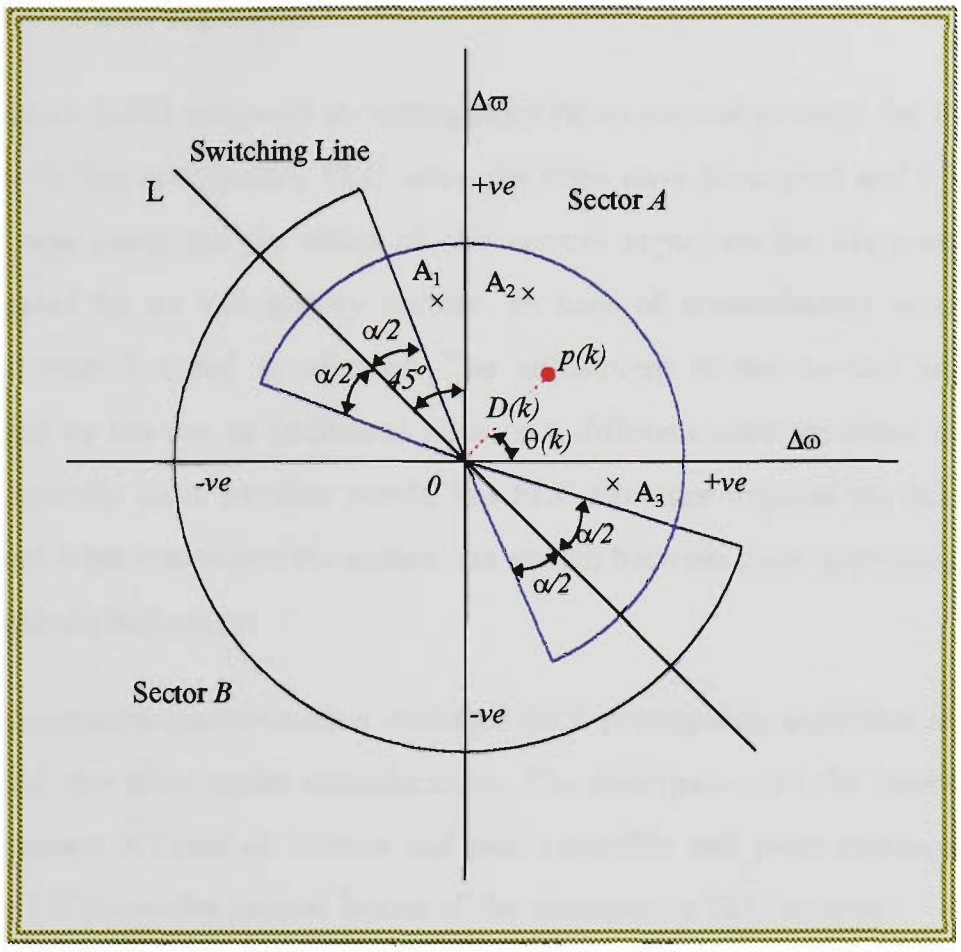


Figure 3.7 modified Δω-Δϖ phase plane

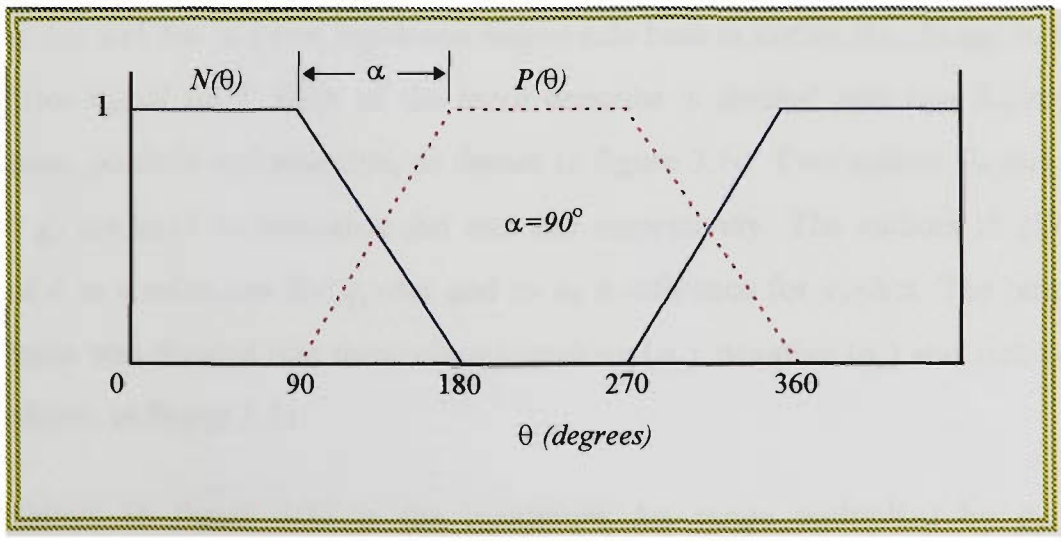


Figure 3.8 linear *P* and *N* functions

This scheme resulted in reducing the parameter number to be optimised to 3 parameters, A_s, D_r and α (shown in figure 3.8). The performance index

$j=\sum_{k=0}^{M}\Delta\omega(k)^2$ was used in the tuning and optimisation process. An evaluation and optimisation of this scheme is presented in references [167-171].

3.5.3 Adaptive Fuzzy logic PSS

Dash et al. [172] proposed an anticipatory fuzzy control strategy for the PSS design. In this anticipatory FLC, once the rules have been used and a control signal was generated the effect of this control signal on the plant output is anticipated by an anticipatory routine. In case of unsatisfactory results the output control signal is adjusted. The adjustment of the control signal is achieved by the use of additional rules or a different rules set other than the one originally used. In other words, this FLC structure requires the storage of different rules sets where the system can switch between them according to the anticipated plant output.

The anticipation routine uses a modified back propagation algorithm in order to model the plant under consideration. The anticipation of the future plant performance is based on current and past controller and plant measurements. Figure 3.9 shows the general layout of the anticipatory FLC structure.

The plant output *y(t)* shown in figure 3.9 represents $\Delta\omega$. The fuzzy controller uses $\Delta\omega$ and $\Delta\varpi$ in a two input-one output rule base to derive the change in the control signal (*dU*). Each of the input domains is divided into two different classes, positive and negative, as shown in figure 3.10. Two scaling factors g_e and g_r are used to normalise $\Delta\omega$ and $\Delta\varpi$ respectively. The authors in [172] used *e* as a reference for $g_e\times\Delta\omega$ and *ce* as a reference for $g_r\times\Delta\varpi$. The output domain was divided into three classes positive (o_p), negative (o_n) and zero (o_z) as shown in figure 3.11.

L shown in figure 3.10 is the maximum $\Delta\omega$ range multiplied by g_e or maximum $\Delta\varpi$ range multiplied by g_r according to the input domain under consideration. L_1 in figure 3.11 represents the maximum possible value of *dU*

multiplied by g_u, where g_u is a scaling factor for dU. The degrees of membership functions for the inputs were calculated as

$$\mu_{ep}(e)=\{L+g_e e\}/2L$$

$$\mu_{en}(e)=\{L+g_e e\}/2L$$

$$\mu_{rp}(e)=\{L+grce\}/2L$$

$$\mu_{rp}(e)=\{L+g_r ce\}/2L$$

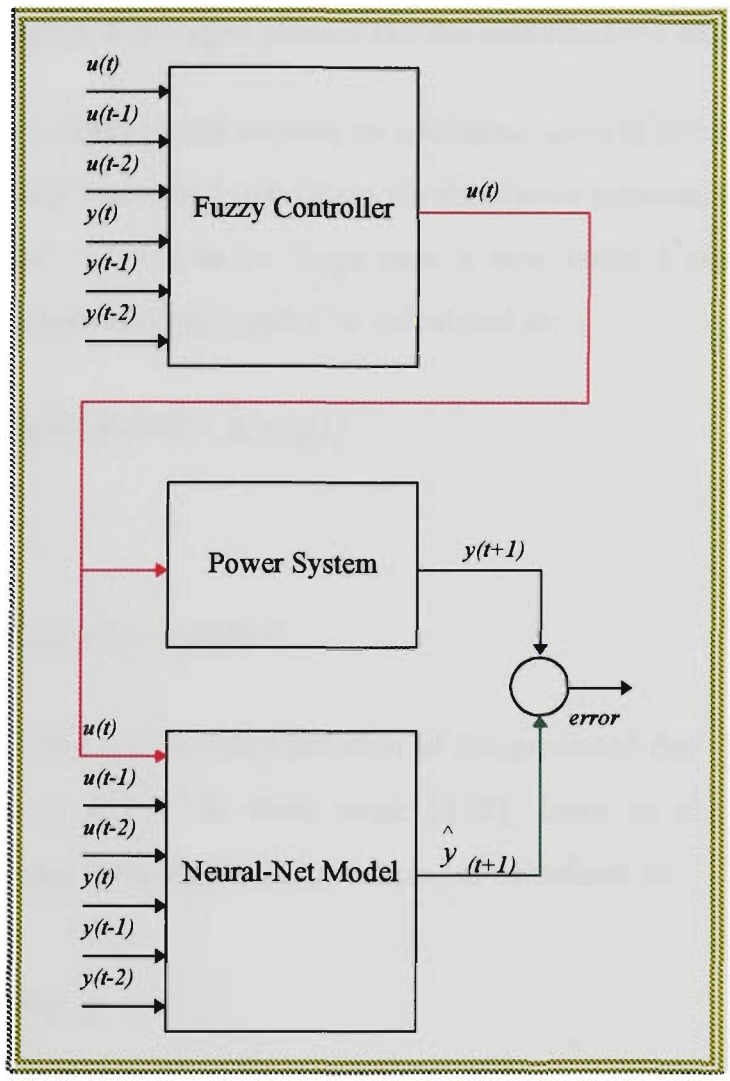


Figure 3.9 anticipatory FLC for PSS design

The output membership functions were calculated as

$$\mu_{op}(dU)= dU/2L$$

$$\mu_{on}(dU)= -dU/2L$$

$$\mu_{oz}(dU)=0$$

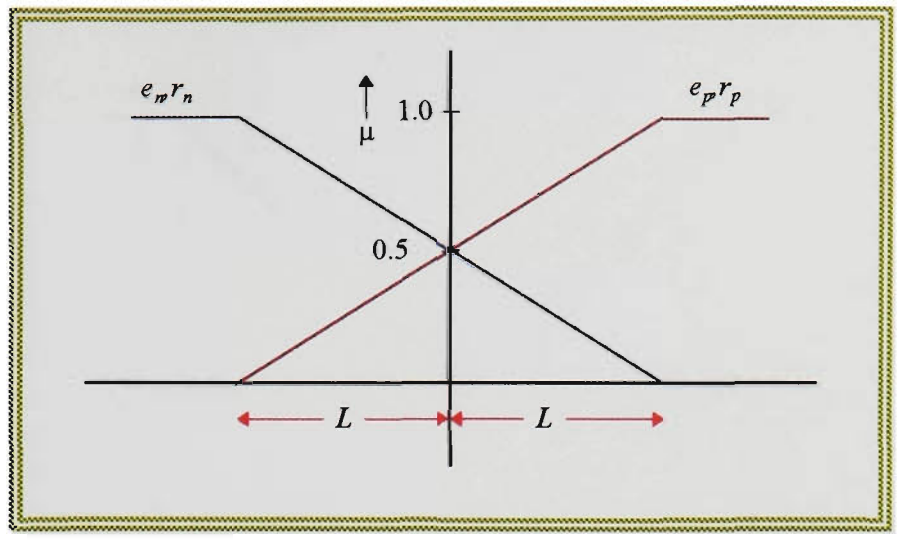


Figure 3.10 input classes for the anticipatory FLC

The anticipatory control used implies an additional control law of the form "if the current control signal (u_c) will cause the difference between the current and anticipated values of $\Delta\varpi$ to be large then a new control signal should be produced". The new control signal U is calculated as:

$$U=u_c[1-\beta\mu\{\hat{\Delta}\,\varpi(t+T)-\hat{\Delta}\,\varpi(t)\}]$$

and

$$\hat{\Delta}\,\varpi=\{\hat{\Delta}\,\omega(t+1)-\hat{\Delta}\,\omega(t)\}/T$$

where $\mu(\hat{\Delta}\,\varpi)$ is the membership function of the predicted $\Delta\varpi$. The value of β varies between 0 and 1. In their work [172], Dash et al. determined β according to the performance index j, where j is calculated as

$$J=\int_0^t (t\Delta\omega)^2 dt$$

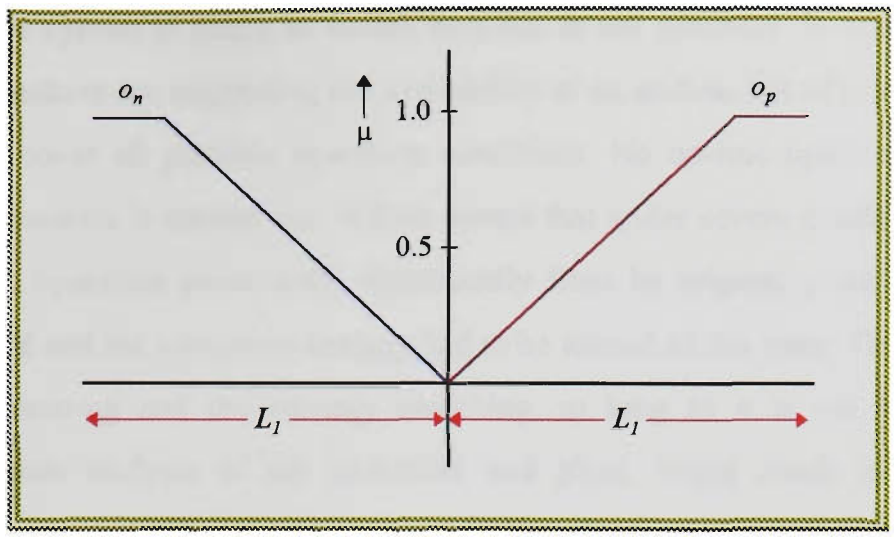


Figure 3.11 output classes for the anticipatory FLC

The implementation of the anticipatory fuzzy controller necessitates the prediction of $\Delta\omega(nT+T)$ one-step-ahead of the input signal. This was achieved using NNs for system identification and one-step-ahead prediction. The authors modelled the input-output relationship of the power system in the form,

$$y(t-T)=f\{y(t),y(t-T),\ldots,y(t-nT),u(t),\ldots,u(t-nT)\}$$

where the function f represents a memoryless nonlinear function. A modified back propagation NN was used to predict the plant output one-step-ahead. The algorithm minimises the mean-square error between the desired output and the actual output with respect to summation output.

Figure 3.9 shows that the output from the fuzzy controller $u(t)$ goes to the NN along with the current and past inputs and outputs to predict the one-step-ahead output. If the NN predicted one-step-ahead output deviates significantly from the actual system output, the weight adjustment of the previously learned neural-net model is carried out.

Although this system produced satisfactory simulation results as mentioned in the reference [172], it is very hard to practically implement or generalise it. The structure contains many parameters that require off-line adjustment and optimisation, ie. g_e, g_r, g_u and β in addition to the need of trained NN models to model the power system. More over, the requirement of unlimited rules set

where the system is going to switch between is not practical. It would seem that the authors are suggesting the availability of an endless list of rules sets in order to cover all possible operation conditions. No on-line updating of the FLC parameters is carried out. Which means that under severe conditions and when the operation point drifts significantly from its original point both the NN model and the switching strategy had to be altered all the time. This is very time consuming and the strategy switching, as long as it is not based on performance analysis of the controller and plant, might result in loss of stability.

Another adaptive scheme was proposed by Hiyama in [173] where he used NNs for real time tuning of a fuzzy logic PSS.

3.6 FLCs DESIGN FOR THE EXCITATION SYSTEM

All the previously mentioned PSSs have a conventional AVR in the control loop, and this limits the overall performance of the power unit under parameter variations while operating. Moreover, the problem such integration encounters, in most cases, is the coordination between the operation of the two logics, fuzzy and linear. Some authors proposed a fully fuzzy excitation system that employs fuzzy logic in the entire excitation loop. In [174] Draper and King proposed a rule-based excitation system with v_t and ΔQ (the difference in the actual reactive power and the scheduled one at generator terminals) as inputs and one output control signal ΔXCTTR as the change in the excitation setting. The domains of the three variables were normalised between -1 and +1 and a 5×5 rule table was used.

A more sophisticated excitation scheme was proposed by Handschin et al. in [175] where three inputs v_t, P_e and ΔP_e were used as inputs to a multi level rule table, this scheme applies the so called hierarchical or multi strategy fuzzy logic control concept with three sets of rules:

1) **terminal voltage rules**; where v_t is the only parameter in the *IF* part,

2) **damping control rules**, where P_e and ΔP_e are the only parameters in the *IF* part,

3) **compensation part**, to compensate part 1 and 2.

Hiyama in [176] proposed a fuzzy excitation system consisting of a fuzzy AVR and a fuzzy PSS.

3.7 COMMENTS AND COMMITMENTS

The previously mentioned systems highlighted the fact that FLCs are applicable in the power system generation and control. However the application of two control ideologies as mentioned in the case of fuzzy PSS and conventional AVR is not the solution FLC can provide to the industry. If there is a linear controller in the control loop then you are still limited with the linear control theory restrictions. It will be a deviation of effort to struggle with two methodologies on-line.

This is clear from the large sum of off-line tuning for the PSS parameters via extensive simulations. This by itself puts fuzzy control under the conditions of plant modeling, parameters adjustment and settings, as if those schemes are trying to map fuzzy control into linear control, by imposing the same sort of restrictions "headaches" and requirements in the design.

The commitment in this thesis is to ***provide a fully fuzzy logic based excitation system in one control block, with adaptive tools operating on line and with one single input to the controller educed by manipulating both v_t and $\Delta\omega$ in a Pre-Control stage, all done using a novel adaptive fuzzy logic controller structure***. This will be presented in section 3.

SECTION 2 DEVELOPMENT OF THE FUZZY LOGIC CONTROLLER DESIGN METHODOLOGY BASED ON A UNIVERSAL ADAPTIVE FLC STRUCTURE

Chapter 4 *Stability Handling Algorithm, Transient and Steady-State (Shay)*

4.0 Introduction

4.1 Brief Overview of the Proposed Shay Structure

4.1.1 Main Functions

4.1.2 Support Functions

4.0 INTRODUCTION

FLCs were very successful when implemented in many different applications. They proved effectiveness in missile cruise control, automobile control, robotics and in the medical field as mentioned in section 1. It was clear from the early days of FLC implementation, that this control technique is superior to a conventional one, when used in a non static environment with wide variation of operational conditions, and where nondeterministic data are the main source of performance monitors. That is the reason why the pioneers in FLC design started to introduce FLCs as the solution for controlling difficult systems where conventional controllers failed or could not perform efficiently. The failure of conventional controllers in an unstable environment can be explained by the fact that they have design methodologies that rely on the availability of a model for the plant under control, based on some predefined performance specifications. It is not always possible to have an accurate plant model. In many cases it is almost impossible to have this accurate model especially when dealing with complex real time systems. The design of a FLC does not require this model restriction, to the same extent. However, in many cases, it is possible to construct the FLC without an existing model. The fact is, that a model of the plant under consideration is required for the initial design and fine tuning of the FLC, specially if the design is to be implemented on expensive and hard to get equipment or in some very critical fields such as in medical applications. It is totally unscientific to test a controller on line during a heart surgery, for example. Even when a model is required, the difference between this requirement for a FLC and that for a conventional controller is:

1) the reduced restrictions on the modeling accuracy needed for a FLC,

2) a single model could be sufficient to design a FLC that can cover a wide range of operation conditions of the plant.

The lack of mature FLC design methodology emerges the need for a plant model which has reasonable accuracy levels. The setting of the controller parameters, ie. the input and output ranges, the membership functions and the rules, relies heavily on the designer experience. A trial and error procedure based on some nondeterministic information taken from a domain expert is followed in most cases. This section introduces a novel FLC

design algorithm that can further reduce the accuracy restrictions for the plant model. The algorithm proposed is adaptive in nature. The algorithm developed throughout this research relies on some new concepts in automatic control. Some of these concepts were derived and elicited from the linear control theory based on field and practical experiments, refer to chapter 5, section 2 for more details on these concepts. The algorithm proposed here handles the main obstacles of FLC implementation and design:

1) the interface between the FLC and the plant which involves mapping of input and output universes of discourse,

2) input and output ranges tuning,

3) rule base,

4) robustness of the controller.

These issues are being carefully encountered and a working solution is proposed and implemented for them. The algorithm proposed here deals with the problem of FLC design from the early design and development stage, to the real time implementation stage. The proposed logic provides the FLC designer with robust, effective and universally adaptive controller that can be simply used for different types of applications having some knowledge of the general facts, ie. expert knowledge about the system behaviour and requirements. This was proven through practical implementation of an excitation system for a synchronous generator presented in section 3.

General layout and brief description of the proposed algorithm is given at the beginning of this chapter followed by description of the algorithm and its supporting functions and features.

4.1 BRIEF OVERVIEW OF THE PROPOSED Shay STRUCTURE

The algorithm developed in this work is capable of handling both steady-state and transient states of the operating plant. This is why it is called **Stability Handling Algorithm, transient and steady-state (*Shay*)**. this is how it will be referred to throughout the rest of the thesis. An overview of the Shay algorithm and its implementation structure is shown in figure 4.1.

The figure shows that the Shay structure consists of ***main*** and ***support*** functions, both working in conjunction with each other to improve the performance of the plant under control. The functionality of both parts is based on some input signals from the plant environment as well as intermediate states and signals derived by the structure throughout the operation time. This figure indicates that the input communication channel from the plant environment and the controller is connected to both the *support* and the *main* functions, while the output channel is connected to the *main* functions. This is due to the fact that the *support* functions section, is a set of operations and procedures that are supporting the operation of the *main* functions in order to enhance the final control signal generated by the *main* functions. In other words, the Shay *main* functions perform the operation of inputting a performance monitor from the plant environment, processing it and then producing a control signal that is to be sent back to the plant. The *main* functions use some extra supporting decisions from the support functions that clarifies the status of the overall controlled system. These two main parts are explained in detail in later chapters of this section.

4.1.1 Main Functions

Shay main functions are a set of three functions that work in conjunction with each other in order to produce a final control signal (U). Each one of these functions is responsible for a different task. These functions operate according to the input from the plant and other derived parameters representing the plant and FLC status, as explained in later chapters of this section. The main functions are:

1) Main-FLC

A single input single output (SISO) FLC used as a reference for the Shay functions.

2) Shay-PA

A function that provides the control signal with the desired phase and fine amplitude tuning.

3) Shay-Tune

A function that updates the input and output ranges of the Main-FLC.

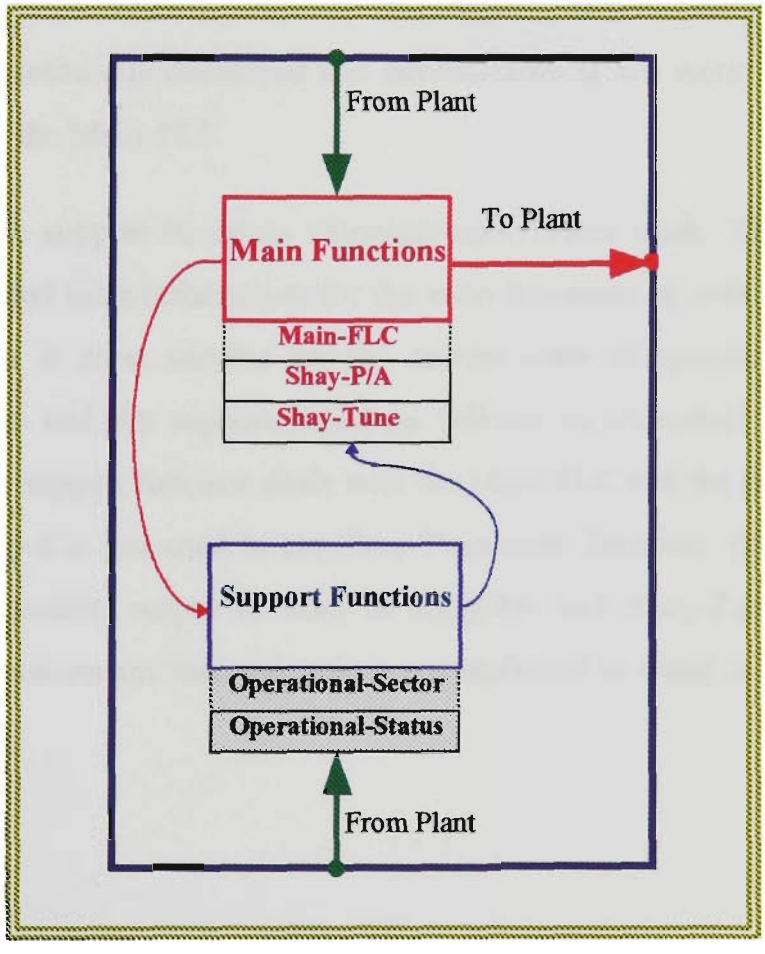


Figure 4.1 Shay algorithm

4.1.2 Support Functions

These are two supporting functions that provide the *main* functions with more precise description of the plant operation, and the main controller status. The support functions use some performance monitors from the plant environment and the Main-FLC. These functions are:

1) Operational Sector

A new concept developed and implemented in this work to give more visualisation of the current state plant of operation.

2) Operational Status

A novel technique developed and implemented in this work to evaluate the status of the Main-FLC.

The main and the support functions complete each others work. The support functions provide the required extra information for the main functions, in order to help the latter in producing U that is most suitable for the current state of operation. The relationship between the main and the support functions follows an individual function basis. The operational status support function deals with the Main-FLC and the plant environment for the input. Its output is just used in the Shay-Tune main function. While the operational-sector support function output is used in Shay-PA and Shay-Tune. The relationship between these functions and their operations are explained in detail in later chapters of this section.

Chapter 5 *Sbay Support Functions*

5.0 INTRODUCTION

This chapter explains in detail the logic and processing involved in Shay support functions. These functions realise the system under control dynamics as well as the FLC structure performance using several new concepts as explained as follows.

5.1 OPERATIONAL SECTOR CONCEPT

The basic idea of this new concept is realised from inspection of the unit step response of a classical feedback system shown in the block diagram in figure 5.1, where *y* is the plant output, *R*: unit step, used as a reference and the error signal (*e*), where $e = R-y$. The unit step response of this classical second order system is shown in figure 5.2.

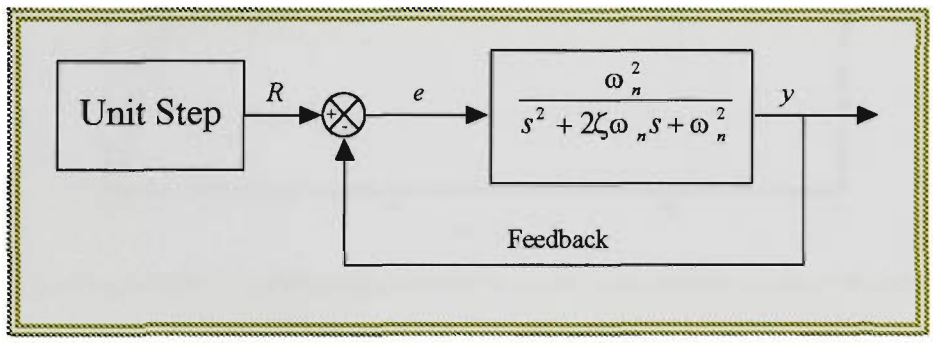


Figure 5.1 second order plant

By inspection of the unit step response shown in figure 5.2, an observer can find six different states or sectors. These sectors are identified in figure 5.2 as:

1) **Picking up:** the range where the error is large and the system is still building up towards the desired steady-state condition,

2) **Crossing 1:** the system response is crossing the reference line and switching the sign from positive to negative (considering an error signal as *Ref-Output*),

3) **After Crossing 1:** the system error is increasing and the plant is heading towards its maximum positive overshoot (μ_p),

4) **After 1st μ_p:** the error is decreasing and the system is going down towards the reference line,

5) **Crossing 2:** the error signal sign is switching from negative to positive,

6) **After Crossing 2:** when the system is approaching its maximum negative overshoot (μ_n).

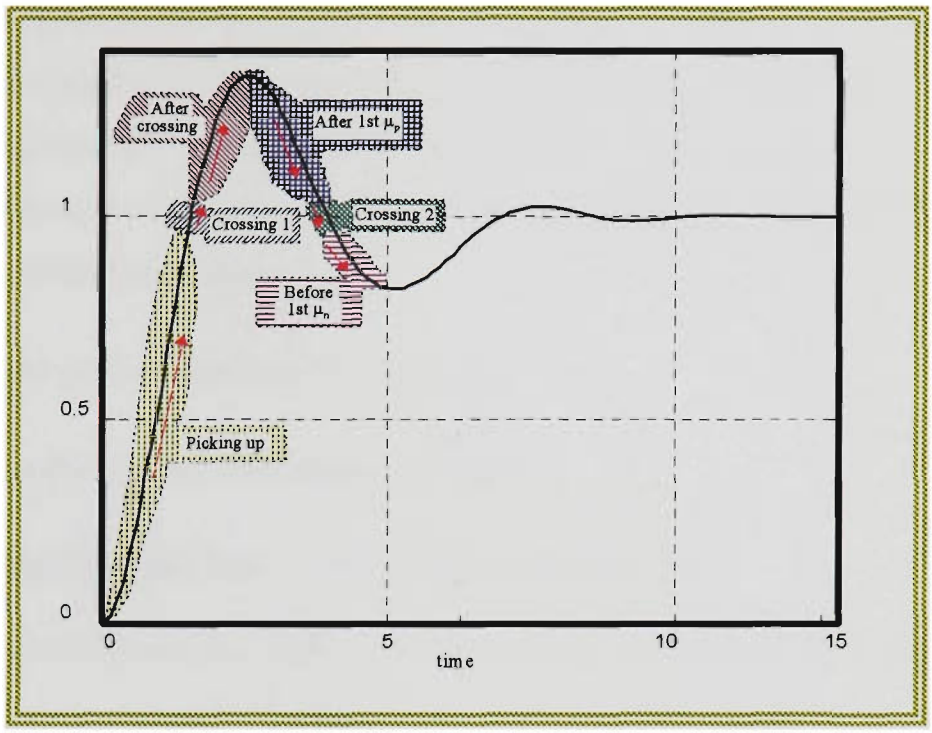


Figure 5.2 six different sectors in the unit step response of a second order system

The sectors are being identified by processing a performance monitor taken from the plant environment, ie. the error signal (*e(k)*). Actually, two successive samples of the performance monitor, *e(k)* and *e(k-1)*, are required to fully represent the plant performance for the sake of the sectors processing. The operational sectors are determined by the joint effect *of e(k)*, *e(k-1)* and $\Delta e(k)$ which is the difference between *e(k)*and *e(k-1)*, referred to as an acceleration, $\Delta e(k)=e(k)-e(k-1)$. A similar procedure is followed in [177] where four different sectors called regions were identified.

The efficiency of Shay depends upon the right choice of the performance monitor, which is the input parameter to the controller from the plant domain. Most of the Shay processing stages apply in one way or another the basic ideas of the proportional plus differential plus integral (PID) controller. A reference signal (*Ref*) is required. A classical measurement

that gives an indication of the plant performance with respect to this reference is used by some parts. Other parts use some successive samples of the performance monitor. The performance monitor used in this work (E_k) was an error signal calculated as E_k=*Ref-plant output.*

Two successive samples of the performance monitor are applied to divide the operational space of the system into six operational sectors. The three dimensional representation of the sectors is shown in figure 5.3. Where the x-axis represents E_k, the y-axis E_{k-1} and the z-axis ΔE_k. A scaling factor (S) is used to scale E_{k-1} in this function. This factor is introduced to give more freedom to the designer in controlling the Shay algorithm. For more details on this factor and its effect on the Shay performance see chapter 9. The operational sectors at time k are defined according to:

1) E_k: the performance monitor at time k,

2) SE_{k-1}: the performance monitor at time k-1,

3) E_k: the difference between E_k and SE_{k-1}, $\Delta E_k = E_k - SE_{k-1}$.

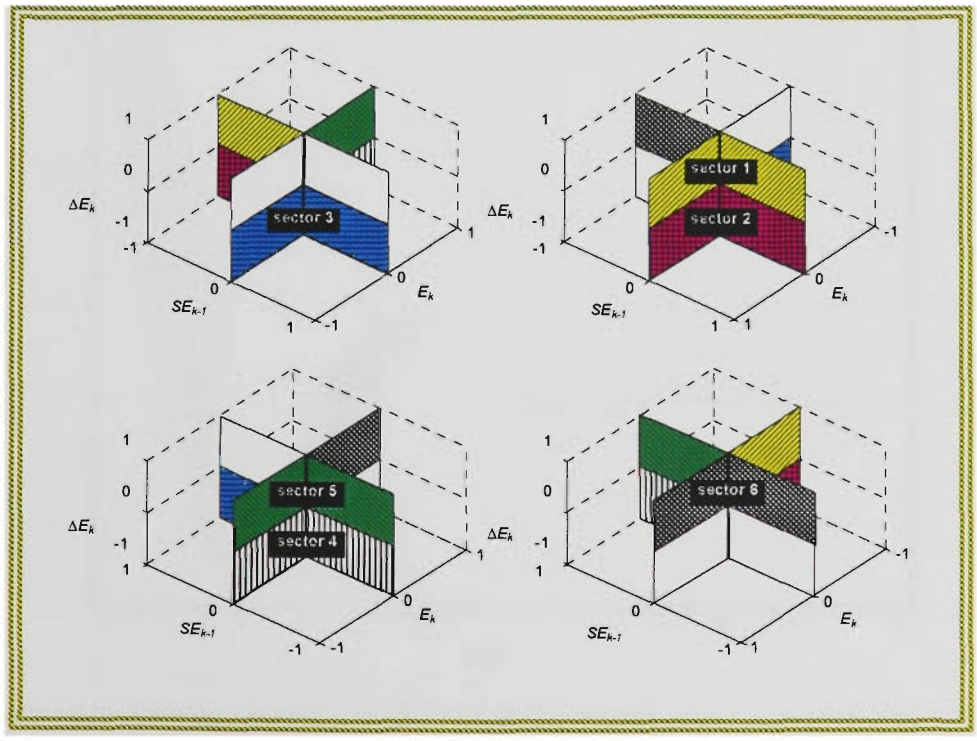


Figure 5.3 operational sectors, three dimensional view

Table 5.1 shows the rules used to derive the operational sectors. Figure 5.4 shows the error signal derived from the unit step response of the second order system. The operational sectors are identified by the numbers 1 to 6 in the figure.

Sector	E_k	SE_{k-1}	ΔE_k
1	+ve	+ve	+ve
2	+ve	+ve	-ve
3	-ve	+ve	-ve
4	-ve	-ve	+ve
5	-ve	-ve	-ve
6	+ve	-ve	+ve

Table 5.1 sectors division

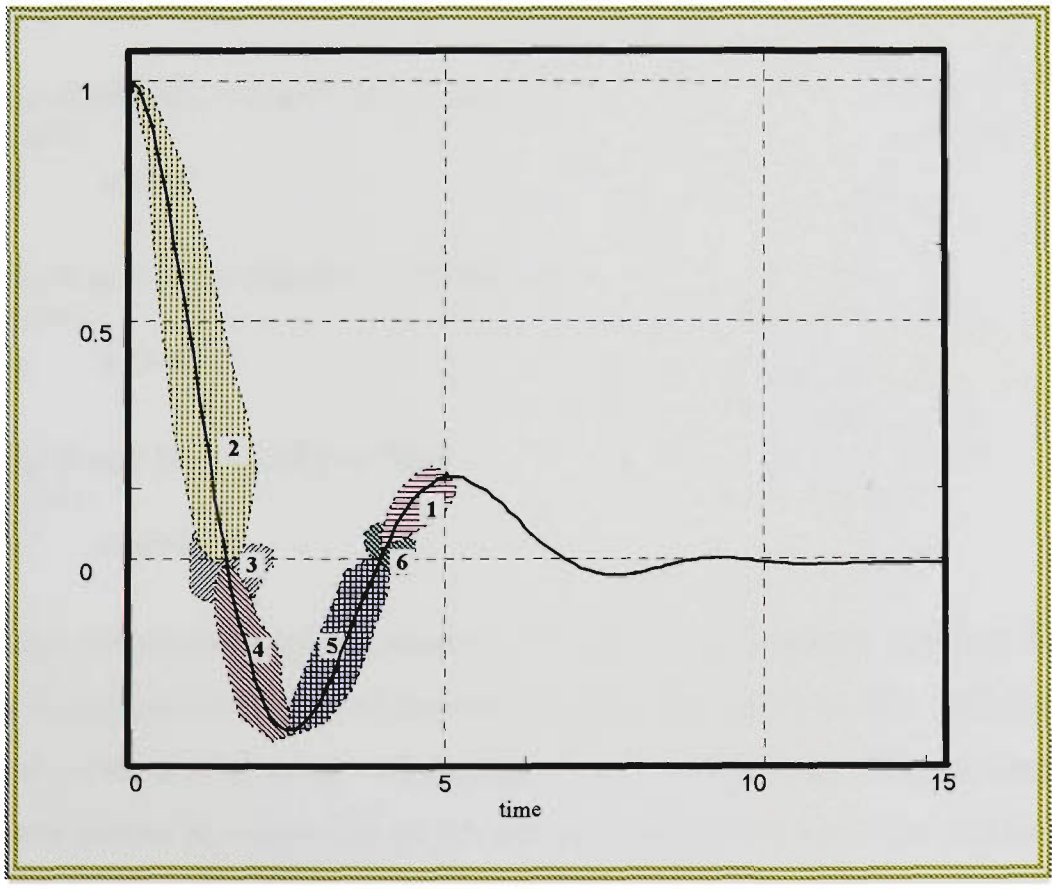


Figure 5.4 operational sectors from the error signal as performance monitor

The operational sector information is stored in a six element vector called ***sct***. Each element of *sct* represents a sector flag that will have a value of 1 if the operation was in that particular sector otherwise the flag's value is zero.

$sct=[sct1\ \ sct2\ \ sct3\ \ sct4\ \ sct5\ \ sct6]$

These flags are assigned their values according to the rules shown in table 1 as:

If $E_k>0$ **and** $SE_{k-1}>0$ **and** $\Delta E_k>0$ **Then**
$sct1=1$
else $sct1=0$

If $E_k>0$ **and** $SE_{k-1}>0$ **and** $\Delta E_k<0$ **Then**
$sct2=1$
else $sct2=0$

If $E_k<0$ **and** $SE_{k-1}>0$ $\Delta E_k<0$ **Then**
$sct3=1$
else $sct3=0$

If $E_k<0$ **and** $SE_{k-1}<0$ **and** $\Delta E_k<0$ **Then**
$sct4=1$
else $sct4=0$

If $E_k<0$ **and** $SE_{k-1}<0$ **and** $\Delta E_k>0$ **Then**
$sct5=1$
else $sct5=0$

If $E_k>0$ **and** $SE_{k-1}<0$ $\Delta E_k>0$ **Then**
$sct6=1$
else $sct6=0$

A simplified two dimensional representation of the sectors is shown in figure 5.5. Table 5.2 gives numerical illustration of the operational sector function. The definition of the operational sector proved to be very helpful at later stages of the Shay processing as it gave a more precise representation of the plant performance status. This will be clarified through the explanation of the main functions and their use of the operational sector information.

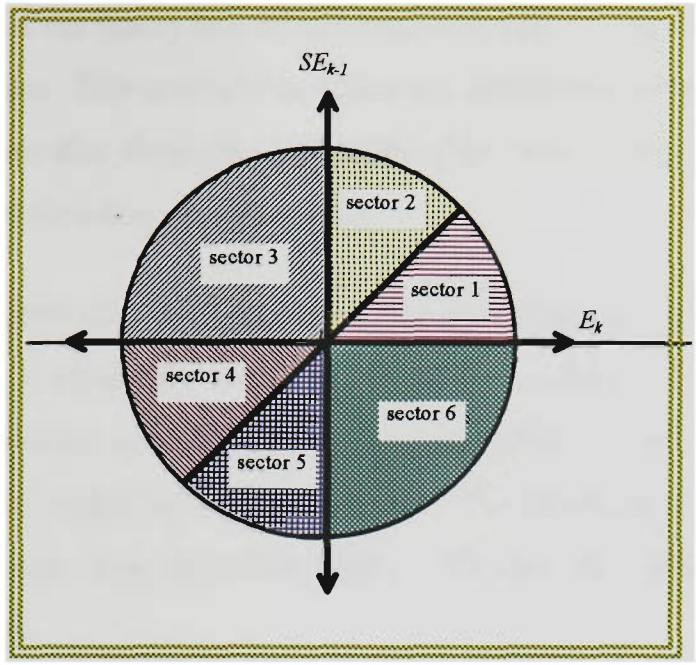


Figure 5.5 operational sectors, two dimensional view

E_k	SE_{k-1}	ΔE_k	*sct*	Operational Sector
0.1	0.05	0.05	[1 0 0 0 0 0]	Sector 1
0.03	0.2	-0.17	[0 1 0 0 0 0]	Sector 2
-0.1	0.1	-0.2	[0 0 1 0 0 0]	Sector 3
-0.3	-0.1	-0.2	[0 0 0 1 0 0]	Sector 4
-0.02	-0.12	0.1	[0 0 0 0 1 0]	Sector 5
0.1	-0.1	0.2	[0 0 0 0 0 1]	Sector 6

Table 5.2 numerical examples of operational sectors

5.2 OPERATIONAL STATUS CONCEPT

This part of Shay is responsible for generating an indicator that assists in the adaptive process of updating the input and output ranges of the Main-FLC. New concepts of evaluating the structure of the FLC based on its operation have been developed and implemented in this part.

Detailed explanation of the theory behind this indicator and its implementation procedures is given in later sections. This indicator is called the operational status of a FLC and it is defined as: ***an indicator that shows the suitability of the current input and output ranges of a FLC in respect to the plant under control.***

The definition of the operational status shows that the objective of this part of the Shay system is to produce a suitability measurement of the current ranges of the FLC. The operational status of an FLC is actually a weighted reasoning of whether the current Main-FLC ranges require any updating or not, qualitative. The reasoning is performed based on some predefined criterion. This procedure supports the operation of the Shay-Tune in the main functions of Shay.

5.2.1 Preliminary Discussion

Any general evaluation of a FLC input and output ranges would result in one of three situations:

1) the ranges need to be increased,

2) the ranges need to be decreased,

3) the ranges are suitable.

Any criterion or methodology which adaptively update and tune the FLC ranges, should consider these three cases.

It is known that the overall ranging of the FLC is initially determined in a non procedural fashion according to the designer's experience and judgment[1]. This work provides a systematic approach to assist this judgment, evaluate it and gives it a standardised measurable form that can be used when attempting to improve the ranges setting. In other words, *moving the art of FLC design from being an experience based art, towards becoming a procedural algorithm, with a structured methodology, that can give the desired results when correctly implemented.*

[1] This is the most common practice. NN and GA can help in this. Still the designer's experience is the dominant factor.

A classical FLC should have a current input/output universes of discourse with predefined upper and lower limits where the membership functions are sharing this domain. The universe of discourse shown in figure 5.6 has clear upper and lower limits bounded by the range of the discourse from 0 (zero) to 1 (one).

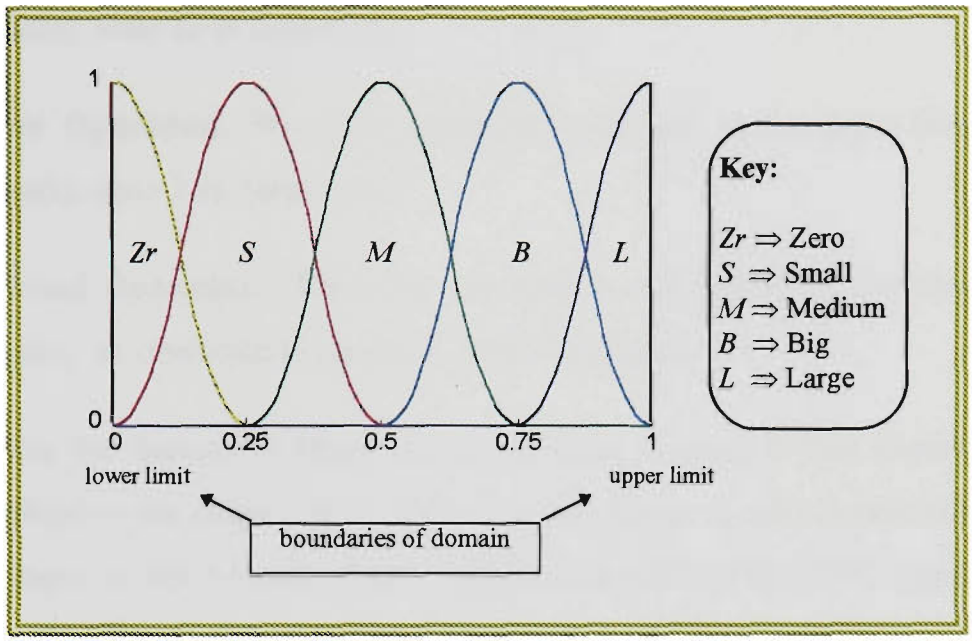


Figure 5.6 fuzzy domain

Classes *Zr* (zero) and *L* (large) in figure 5.6 are the boundary classes. If the FLC operation was always in the *Zr* or *L* classes with the continuity of a steady-state error, then, it is most likely that the current range of the FLC falls either in category 1 if it was always in the *Zr* class or category 2 if it was in class *L*. This is stated in general form in ***Remark 5.1*** which is the mid-stone of the operational status concept.

Remark 5.1

If the operation of the FLC with respect to this domain (whether input or output domain) was centred in the boundary classes and the controller shows unsatisfactory performance then we can claim that the current ranges require some modifications.

Three modes of operation are defined in this section, they are relatively measured based on the initial assumptions implemented when designing a FLC, these modes are:

1) **Under Operation:** When the operation is centred in the lower boundary class(s), class *Zr* in figure 5.6,

2) **Over Operation:** When the operation is centred in the upper boundary class(s), class *L* in figure 5.6,

3) **Normal Operation:** When the operation is not centred in the boundary class(s), ie. operation in classes *S*, *M* or *B* in figure 5.6.

Consider the domain in figure 5.6 as an input domain. If this domain was normalised to the range 0 to 1. Assume a gain factor is used to normalise the crisp input to the domain. If the fuzzification stage of the FLC resulted in having the class *Zr* fired most of the time while still having undesired steady-state error, then the FLC is in an **under operation** mode and the input gain is less than the required for the FLC to interactively respond to the input. On the other hand, if the fuzzification stage resulted in firing the input class *L* most of the time while still having unsatisfactory results, then the FLC is said to be in **over operation** mode, and this indicates that the current input gain factor is higher than what is required for the FLC to respond interactively to the input signals. Otherwise, if the operation was in other classes of the domain, and satisfactory results were achieved, then the FLC is said to be under **normal operation** mode and no adjustments are required for the gain factor. The same treatment and three modes are used for the output range of the FLC. The procedure to determine the modes and the ranges updating procedures are explained in full detail in later sections of this chapter.

The result of the fuzzification stage in the Main-FLC (*fz*) is required when evaluating the current input range while the result of the rule evaluation (*rl*) is used for the evaluation of the current output range, refer to chapter 2 for details of the FLC operation steps.

5.2.2 FLC Ranges Evaluation Criterion

In order to ease the processing in later stages of the operational status derivation, we assign a number for each class in the domain. Thus the labels are used for description while the numbers are used for the implementation and processing. The numbering is in order, from left to right. So the domain shown in figure 5.6 is being assigned the numbers as in figure 5.7.

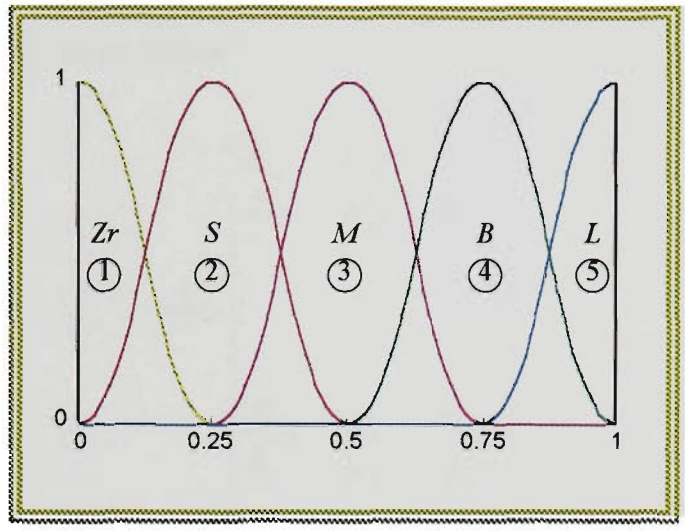


Figure 5.7 numbered fuzzy domain

The fuzzification stage for n number of input classes will result in an $nx3$ *fz* matrix, with each row (i),of *fz* representing the fuzzification information regarding class i, *fz* has the following syntax:

$$fz = \begin{bmatrix} F_{f_1} & \mu_{f_1} & S_{f_1} \\ F_{f_2} & \mu_{f_2} & S_{f_2} \\ . & . & . \\ . & . & . \\ . & . & . \\ F_{f_{n-1}} & \mu_{f_{n-1}} & S_{f_{n-1}} \\ F_{f_n} & \mu_{f_n} & S_{f_n} \end{bmatrix}$$

where

F_{f_i} : is an indicator that will have a value of 1 if the crisp input belongs to class i, or 0 if it does not.

μ_{f_i} : is the degree of membership of the crisp input to the class *i*.

S_{f_i} : the side of the fired class *i* which is the location of the crisp input with respect of the class *i COG*, refer to appendix A for *L/R COG* method.

The same syntax is used to represent the fired output classes from the rule evaluation stage. An *n* number of output classes will result in an *n*x*3 rl* matrix having the format shown below

$$rl = \begin{bmatrix} F_{r_1} & \mu_{r_1} & S_{r_1} \\ F_{r_2} & \mu_{r_2} & S_{r_2} \\ . & . & . \\ . & . & . \\ . & . & . \\ F_{r_{n-1}} & \mu_{r_{n-1}} & S_{r_{n-1}} \\ F_{r_n} & \mu_{r_n} & S_{r_n} \end{bmatrix}$$

where

F_{r_i} : is an indicator that takes a value of 1 or 0 to show that output class *i* was fired or not.

μ_{r_i} : is the degree of membership of the output to the class *i*.

S_{r_i} : the side of the fired output class *i*, with respect to the *COG* of output class *i*.

A rule-based fuzzy logic processing is implemented in this stage in order to derive the domains operational states. This part of the Shay system can be seen as two rule-based fuzzy logic blocks operating in parallel, one is concerned with the input domain and the other with the output domain. Processing in both of them is quite similar, so, what is explained for the input operational status can be directly mapped for the output domain.

Two successive samples of *fz* are required to map the input domain to the three modes. fz_k, *fz* at time *k*, is used to define three temporary input mode indicators, I_U_k, I_N_k and I_O_k where

$I\_U_k = \mu_{f_{1_k}}$, input under operation mode at time *k*.

$I\_N_k = max(\mu_{f_{2_k}}, \ldots, \mu_{f_{n-1_k}})$, input normal operation mode at time *k*.

$I\_O_k = \mu_{f_{n_k}}$, input over operation mode at time *k*.

fz_{k-1}, *fz* at time *k-1*, is used to define three temporary input mode indicators, I_U_{k-1}, I_N_{k-1} and I_O_{k-1} where

$I\_U_{k-1} = \mu_{f_{1_{k-1}}}$, input under operation mode at time *k-1*.

$I\_N_{k-1} = max(\mu_{f_{2_{k-1}}}, \ldots, \mu_{f_{n-1_{k-1}}})$, input normal operation mode at time *k-1*.

$I\_O_{k-1} = \mu_{f_{n_{k-1}}}$, input over operation mode at time *k-1*.

A graphical representation of the input domain modes is shown in figure 5.8.

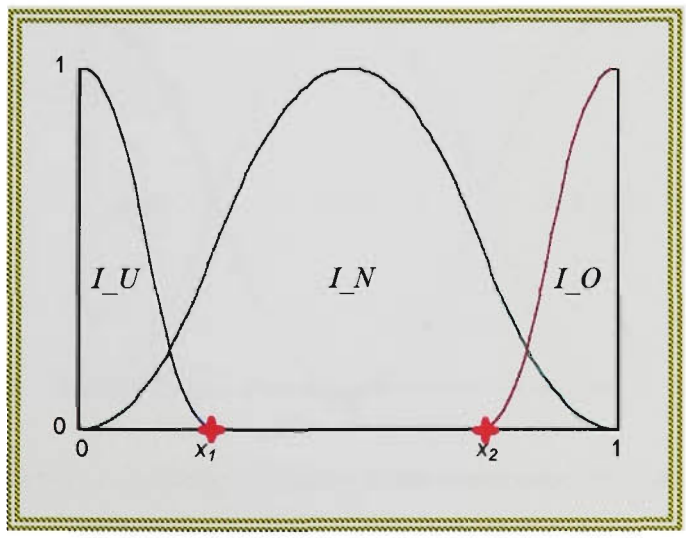


Figure 5.8 input domain modes

The point x_1 represents the upper limit of the lower class of the FLC, this is class *Zr* in figure 5.7. x_2 represents the lower limit of the upper class, this is

class L in figure 5.7. These two points are dependent upon the Main-FLC under consideration, and vary from one system to another.

The two mapped values of fz_k and fz_{k-1} are processed in a two input single output rule-based fuzzy block to derive the final indication of the input operational status. The output of this step is a three element vector called *IOS* (input operational status),

$$IOS=[I\_R_1 \quad I\_R_2 \quad I\_R_3]$$

where

$I\_R_1 \Rightarrow$ input **under operation** mode.

$I\_R_2 \Rightarrow$ input **normal operation** mode.

$I\_R_3 \Rightarrow$ input **over operation** mode.

These modes are represented graphically in figure 5.9. This representation is fixed and does not change from one system to another.

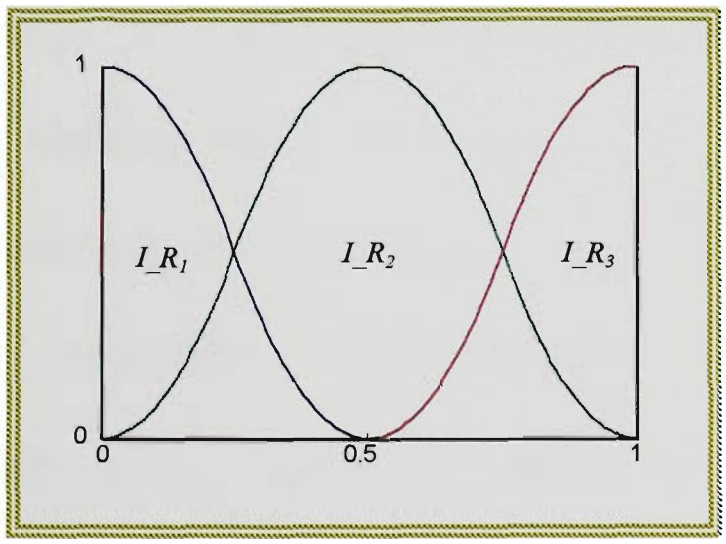


Figure 5.9 input operational modes

A *3x3* rule table is required to derive the final *IOS*. A very conservative rule base (a rule base that implements a strategy which prevents applying large changes) was used in this work. This rule base is shown in table 5.3.

		fz_k		
		I_U_k	I_N_k	I_O_k
fz_{k-1}	I_U_{k-1}	I_R_1	I_R_2	I_R_2
	I_N_{k-1}	I_R_2	I_R_2	I_R_2
	I_O_{k-1}	I_R_2	I_R_2	I_R_3

Table 5.3 rule base to derive *IOS*

The *min* operator is used in the rule evaluation and I_R_1, I_R_2 and I_R_3 are assigned the *min* strength of the set of rules that fires each of them. ie. $I\_R_1=min(I\_U_{k-1},I\_U_k)$, the first cell in table 5.3.

Processing the output domain in order to determine the output operational status vector *OOS* is performed in the same manner, where O_U_k, O_N_k and O_O_k are calculated from rl_k as:

$O\_U_k=\mu_{r_{1_k}}$, output under operation mode at time *k*.

$O\_N_k=max(\mu_{r_{2_k}},\ldots,\mu_{r_{n-1_k}})$, output normal operation mode at time *k*.

$O\_O_k=\mu_{r_{n_k}}$, output over operation mode at time *k*.

rl_{k-1} is mapped into O_U_{k-1}, O_N_{k-1} and O_O_{k-1} as

$O\_U_{k-1}=\mu_{r_{1_{k-1}}}$, output under operation mode at time *k-1*.

$O\_N_{k-1}=max(\mu_{r_{2_{k-1}}},\ldots,\mu_{r_{n-1_{k-1}}})$, output normal operation mode at time *k-1*.

$O\_O_{k-1}=\mu_{r_{n_{k-1}}}$, output over operation mode at time *k-1*.

These two mapped values are processed in a two input single output rule-based fuzzy block to derive the final indication of the output operational status. The output of this step is a vector called *OOS* with three members as shown in the following,

$$OOS=[O\_R_1 \quad O\_R_2 \quad O\_R_3]$$

where

$O\_R_1 \Rightarrow$ output **under operation** mode.

$O\_R_2 \Rightarrow$ output **normal operation** mode.

$O\_R_3 \Rightarrow$ output **over operation** mode.

Table 5.3 is used in this work for the input and output operational status. This table is reintroduced in table 5.4 with reference to the output domain.

		rl_k		
		O_U_k	O_N_k	O_O_k
rl_{k-1}	O_U_{k-1}	O_R_1	O_R_2	O_R_2
	O_N_{k-1}	O_R_2	O_R_2	O_R_2
	O_O_{k-1}	O_R_2	O_R_2	O_R_3

Table 5.4 rule base to derive *OOS*

5.2.3 Generalised Form of the Operational Status

The explanation provided before was for the domain shown in figure 5.7 which ranges from 0 to 1. This domain was used to simplify the description of the operational status concept. However, the same procedures mentioned earlier in section 5.2 stands for the fuzzy domain shown in figure 5.10 which ranges from -1 to 1.

The same modes are used as well as the same rules in tables 5.3 and 5.4. The only difference is in determining the temporary modes indicators.

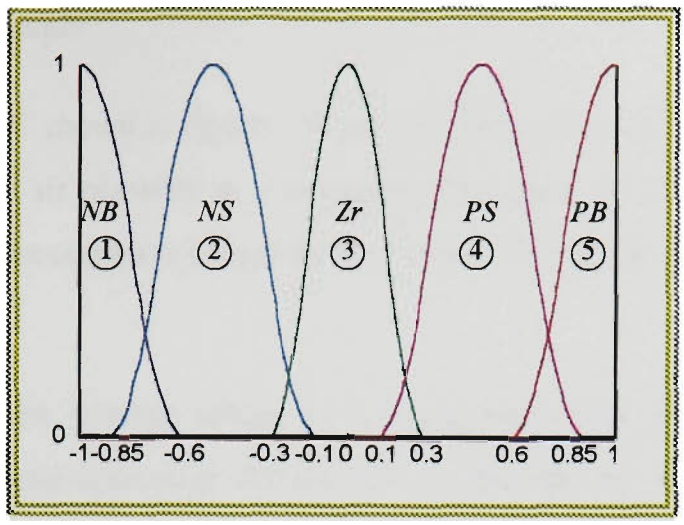


Figure 5.10 fuzzy domain of the range -1 to 1

For the domain in figure 5.10 the input temporary modes indicators are calculated as in equation 5.1.

$$\left.\begin{aligned} I\_U_k &= \mu_{f_{3_k}} \\ I\_N_k &= max(\mu_{f_{2_k}}, \mu_{f_{4_k}}) \\ I\_O_k &= max(\mu_{f_{1_k}}, \mu_{f_{5_k}}) \end{aligned}\right\} \quad (5.1)$$

fz_{k-1} is used to define three temporary input mode indicators, I_U_{k-1}, I_N_{k-1} and I_O_{k-1} as shown in equation (5.2).

$$\left.\begin{aligned} I\_U_{k-1} &= \mu_{f_{3_{k-1}}} \\ I\_N_{k-1} &= max(\mu_{f_{2_{k-1}}}, \mu_{f_{4_{k-1}}}) \\ I\_O_{k-1} &= max(\mu_{f_{1_{k-1}}}, \mu_{f_{5_{k-1}}}) \end{aligned}\right\} \quad (5.2)$$

The same technique is followed in case of an output domain. This form mentioned in equations (5.1) and (5.2) is a general form of deriving the modes temporary indicators.

5.2.4 Illustrative Example

Consider the FLC shown in figure 5.11 which represents an air flow controller that controls the air pressure in a container. The input to the FLC is the air pressure inside the container measured via a sensor that gives values from -5 to 5 volts.

The output is the voltage setting of a compressor that moves clockwise (forward) and anti-clockwise (backward) to increase or decrease the air pressure.

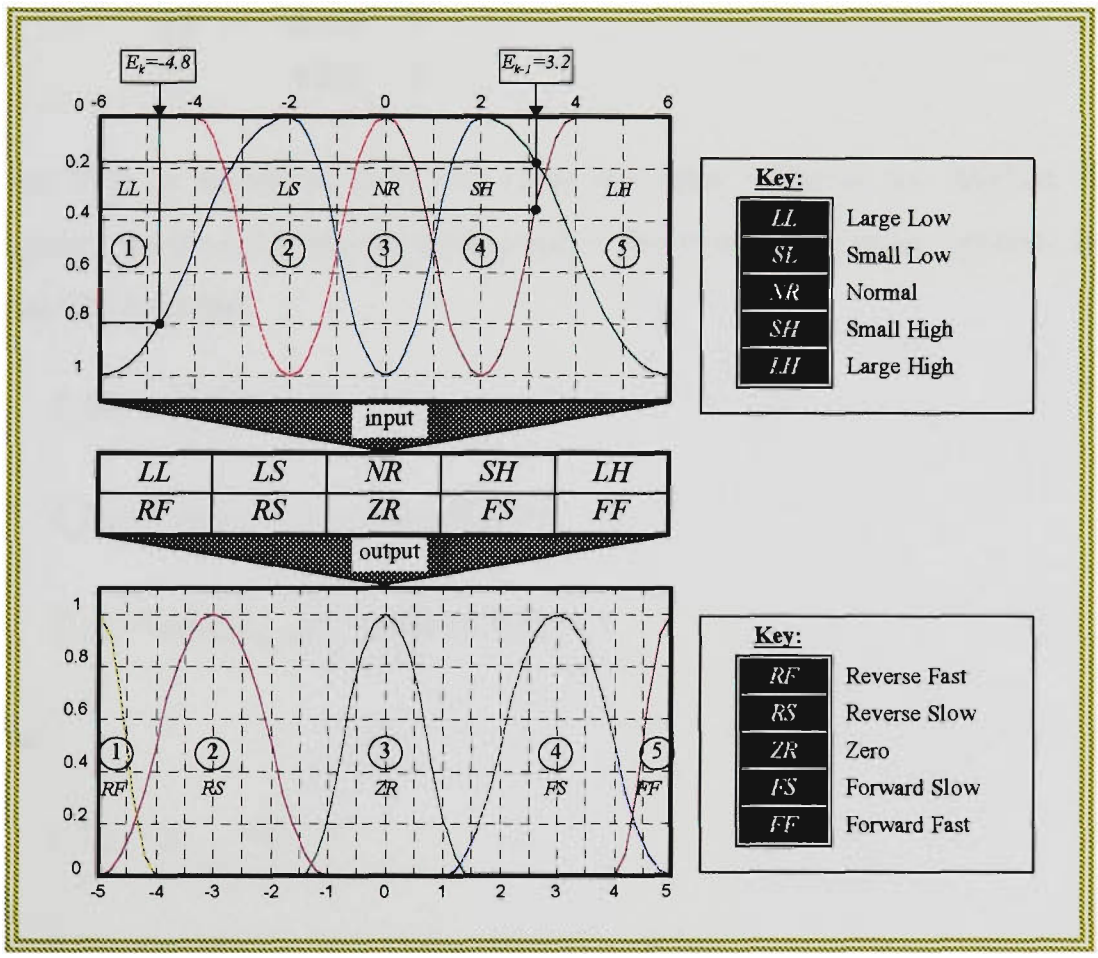


Figure 5.11 FLC for air flow controller example

If the sensor's reading was -4.8 *volt* at time k (point E_k in figure 5.11), E_k belongs to the input class *LL* with $\mu_{LL}=0.8$. If the reading was 3.2 volt at time *k-1* (point E_{k-1} in figure 5.11), E_{k-1} belongs to the input classes *SH* and *LH* with $\mu_{SH}=0.167$ and $\mu_{LH}=0.367$. Equations (5.3) and (5.4) show fz_k and fz_{k-1} for these two samples.

$$fz_k = \begin{bmatrix} 1 & 0.8 & X \\ 0 & 0 & X \\ 0 & 0 & X \\ 0 & 0 & X \\ 0 & 0 & X \end{bmatrix} \tag{5.3}$$

and

$$fz_{k-1} = \begin{bmatrix} 0 & 0. & X \\ 0 & 0 & X \\ 0 & 0 & X \\ 1 & 0.367 & X \\ 1 & 0.167 & X \end{bmatrix} \tag{5.4}$$

Note that in equations (5.3) and (5.4) the sides columns are labelled X: (ignore), as the sides are not considered in the operational status analysis for simplicity reasons.

$$I\_U_k = \mu_{f_{3_k}} = 0$$

$$I\_N_k = max(\mu_{f_{2_k}}, \mu_{f_{4_k}}) = max(0,0) = 0$$

$$I\_O_k = max(\mu_{f_{1_k}}, \mu_{f_{5_k}}) = max(0.8,0) = 0.8$$

and

$$I\_U_{k-1} = \mu_{f_{3_{k-1}}} = 0$$

$$I\_N_{k-1} = max(\mu_{f_{2_{k-1}}}, \mu_{f_{4_{k-1}}}) = max(0,0.367) = 0.367$$

$$I\_O_{k-1} = max(\mu_{f_{1_{k-1}}}, \mu_{f_{5_{k-1}}}) = max(0,0.167) = 0.167$$

thus applying table 5.3 for the input operational status yields

$$I\_R_1 = min(I\_U_k, I\_U_{k-1}) = min(0,0) = 0$$

$$I\_R_2 = max(min(I\_U_k I\_N_{k-1}),\ min(I\_U_k I\_O_{k-1}),\ min(I\_N_k I\_U_{k-1}),$$
$$min(I\_N_k I\_N_{k-1}),\ min(I\_N_k I\_O_{k-1}),\ min(I\_O_K, I\_U_{k-1}),$$
$$min(I\_O_k I\_N_{k-1}))$$

$$= max(min(0, 0.367),\ min(0, 0.167),\ min(0, 0.367),\ min(0, 0.367),$$
$$min(0, 0.167),\ min(0.8, 0),\ min(0.8, 0.367))$$

$$= max(0,0,0,0,0,0,0.367) = 0.367$$

$$I\_R_3 = min(I\text{-}O_k I\_O_{k-1}) = min(0.8, 0.167) = 0.167$$

$$\Rightarrow IOS = [0 \quad 0.367 \quad 0.167]$$

Applying the SISO rule base shown in figure 5.11 gives rl_k as

$$rl_k = \begin{bmatrix} 1 & 0.8 & X \\ 0 & 0 & X \\ 0 & 0 & X \\ 0 & 0 & X \\ 0 & 0 & X \end{bmatrix}$$

and

$$rl_{k-1} = \begin{bmatrix} 0 & 0 & X \\ 0 & 0 & X \\ 0 & 0 & X \\ 1 & 0.367 & X \\ 1 & 0.167 & X \end{bmatrix}$$

$$O\_U_k = \mu_{r_{3_k}} = 0$$

$$O\_N_k = max(\mu_{r_{2_k}}, \mu_{r_{4_k}}) = max(0,0) = 0$$

$$O\_O_k = max(\mu_{r_{1_k}}, \mu_{r_{5_k}}) = max(0.8,0) = 0.8$$

and

$O\_U_{k-1} = \mu_{r_{3_{k-1}}} = 0$

$O\_N_{k-1} = max(\mu_{r_{2_{k-1}}}, \mu_{r_{4_{k-1}}}) = max(0, 0.367) = 0.367$

$O\_O_{k-1} = max(\mu_{r_{1_{k-1}}}, \mu_{r_{5_{k-1}}}) = max(0, 0.167) = 0.167$

using table 5.4 to derive *OOS* results in

$O\_R_1 = min(O\_U_k, O\_U_{k-1}) = min(0,0) = 0$

$O\_R_2 = max(min(O\_U_k, O\_N_{k-1}), min(O\_U_k, O\_O_{k-1}), min(O\_N_k, O\_U_{k-1}),$
$min(O\_N_k, O\_N_{k-1}), min(O\_N_k, O\_O_{k-1}), min(O\_O_k, O\_U_{k-1}),$
$min(O\_O_k, O\_N_{k-1}))$

$= max(min(0, 0.367), min(0, 0.167), min(0, 0.367), min(0, 0.367),$
$min(0, 0.167), min(0.8, 0.167), min(0.8, 0.367))$

$= max(0,0,0,0,0,0.167,0.367) = 0.367$

$O\_R_3 = min(O\_O_k, O\_O_{k-1}) = min(0.8, 0.167) = 0.167$

$\Rightarrow OOS = [0 \quad 0.367 \quad 0.167]$

Chapter 6 *Shay Main Functions Overview and Main-FLC*

6.0 INTRODUCTION

This chapter and the following two chapters explain in detail the operation as well as the logic implemented in the Shay main functions. Processing in Shay structure is performed via a joint effort of the main functions, that are operating in a network of fuzzy nodes. Plus the support functions which supply the main functions with more processed data regarding the Main-FLC and the plant under control states.

This chapter starts with a general overview of the Shay structure and the communication links used in it, then the Main-FLC (the first of Shay main functions) is introduced.

6.1 GENERAL OVERVIEW

A general layout of the Shay structure is shown in figure 6.1. The figure shows both the main and the support functions as well as the communication channels between Shay and the plant and the internal communication within Shay itself.

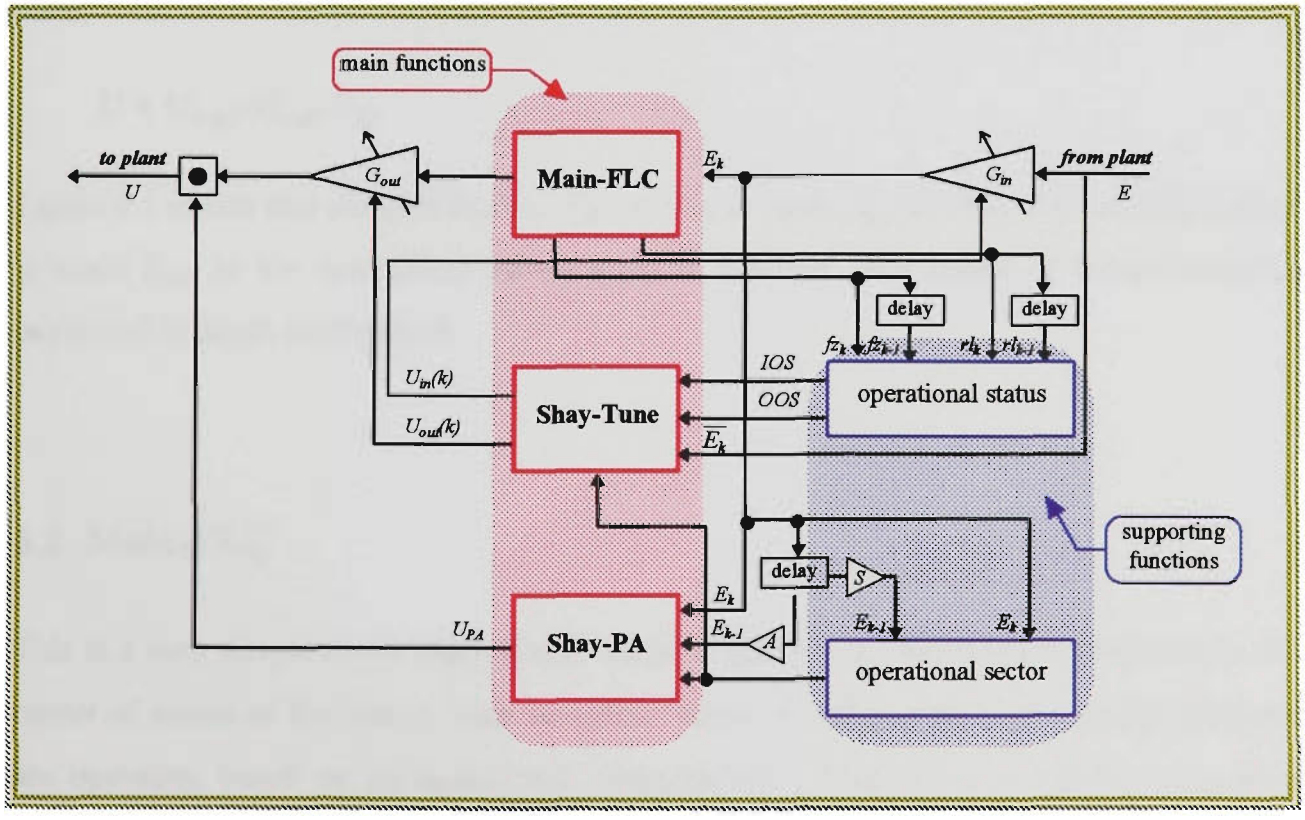


Figure 6.1 Shay general layout

Shay receives one input from the plant, E_k. E_k is then normalised by the input gain G_{in}. The normalised E_k is the input to the Main-FLC which in turn produces U_{main}. The output gain, G_{out} is used to normalise U_{main}. Note that in the following parts of the thesis, the normalised E_k is referred to by E_k and the unnormalised E_k is referred to by $\overline{E_k}$.

The operational sector uses two successive samples of E_k and produces the vector *sct* that describes the operational sector. Shay-PA uses *sct*, E_k and E_{k-1} to produce the enhancement signal U_{PA}.

The operational status support function uses fz_k, fz_{k-1}, rl_k and rl_{k-1} (two successive samples of the fuzzy input and output of the Main-FLC respectively) and produces the vectors *IOS* and *OOS* describing the operational status of the Main-FLC from input and output perspective.

Shay-Tune uses four inputs, the *sct* vector from the operational sector support function, *IOS* and *OOS* from the operational status support function and E_k in order to produce the two updating signals for the input and output gains, G_{in} and G_{out}. These updating signals are U_{in} and U_{out}. The final control signal, U, is produced at the last product node in the form,

$$U = U_{main} \times G_{out} \times U_{PA}$$

Figure 6.1 shows that the gain factors: *(A)* is used to scale E_{k-1} in Shay-PA, and *(S)* is used to scale E_{k-1} in the operational sector support function. The effect of these factors is explained in detail in chapter 9.

6.2 Main-FLC

This is a very simple single input single output (SISO) FLC. However it functions as the center of action of the whole Shay structure, where all other main and support functions are operating based on its operational characteristics. This SISO is effecting the final control signal directly and indirectly. The direct effect is in its proportionality nature, as it largely influences the amplitude of U. The indirect effect is clear when knowing that the

status and performance of the Main-FLC is directly influencing the other main and support functions.

The *IOS* and *OOS* vectors are representatives of the status of this SISO. Thus the operation of both Shay-Tune and Shay-PA is focused on the Main-FLC status. From this dependency, the importance of Main-FLC is very obvious. This will become clearer in chapters 7 and 8, as the operation of the other main functions are explained.

The initial design of the Main-FLC is not subjected to many restrictions. The designer has enough margins in accuracy in choosing the initial parameters of the FLC. This is because these parameters are going to be adaptively adjusted and tuned throughout the operation time by the other two main functions:

1) Shay-Tune, which will continuously update the input and output ranges of the Main-FLC utilising the operational sector and the operational status information,

2) Shay-PA which applies an implicit membership functions adaptive tuning from amplitude and phase points of view with the assistance of the operational sector support function.

The Main-FLC produces the signal U_{main} proportional to the absolute value of the performance monitor. Whereas Shay-PA provides the required phase in the final control signal, U. A sample of Main-FLC is shown in figure 6.2. The figure shows the actual Main-FLC used in the practical implementation of Shay in this thesis.

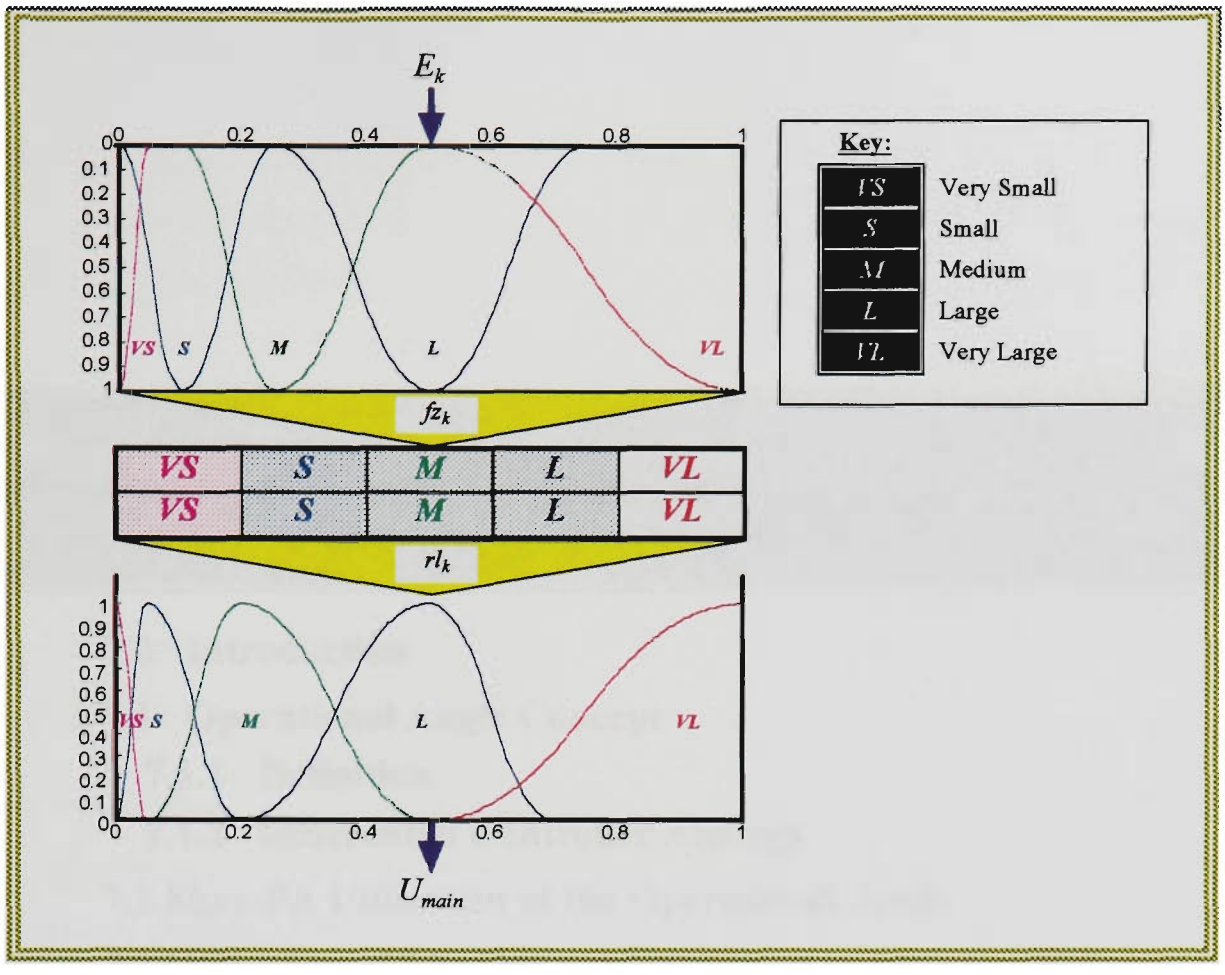


Figure 6.2 Main-FLC example

Chapter 7 *Shay Phase and Amplitude Tuner (Shay-PA)*

7.0 INTRODUCTION

Shay-PA performs the differential controller action, it is concerned with the dynamics of the plant under control. Shay-PA operates in conjunction with the operational sector support function in order to produce the control signal U_{PA} that has two objectives:

1) to provide U_{main} with the required phase,

2) to fine tune the amplitude of U_{main} in accordance to the current dynamics of the plant under control.

The two letters PA at the end of the string "Shay-PA" are abbreviations for **P**hase and **A**mplitude, respectively.

Shay-PA applies a novel concept to evaluate the dynamics of the plant using two successive samples of the performance monitor, E_k and E_{k-1}, this concept is called the ***operational angle concept***.

This chapter starts with the description of the operational angle concept and the way it represents the plant dynamics, the utilisation of the operational angle in Shay-PA is shown in later sections.

7.1 OPERATIONAL ANGLE CONCEPT

7.1.1 Definition

The operational angle concept was developed in this work as an accurate means of evaluating the dynamics of the plant under control. This measurement is used to enhance the adaptive fine tuning control signal. The plant dynamics are captured in a parameter called the ***operational angle*** (θ_{opr}). The operational angle is defined as: ***the angle between two successive samples of the performance monitor, E_k and the scaled E_{k-1} (AE_{k-1}), with E_k represents the horizontal axis and AE_{k-1} the vertical axis***. θ_{opr} is calculated at

time k in the form of: $\theta_{opr_k} = \tan^{-1}(AE_{k-1}/E_k)$ and presented graphically as shown in figure 7.1.

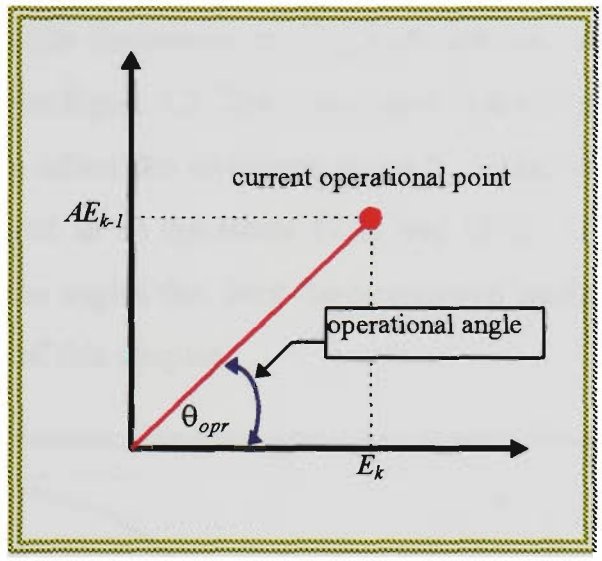


Figure 7.1 operational angle graphical representation

7.1.2 Differential Controller Analogy

The FLC based on the operational angle concept has a great analogy to the classical differential controller. The differential controller uses the input that represents the time differential of two successive samples, in discrete time domain, and thus, this difference is considered as an input used mainly for the transient state of the plant. θ_{opr} is doing the same, the quarter in the x-y plane and the angle measurement can give a clear representation of the plant dynamics at steady and transient states. Moreover, the analogy becomes clear if one considers the procedure of producing θ_{opr} which is performed by processing two successive samples of the input. The only difference is that the differential controller relies on the sampling time to perform the differentiation, and θ_{opr} does not require this. To realise a differential controller based on the operational angle concept, one has to set the sampling time according to the required time and set the A gain to 1. Based on this, it can be seen that the operational angle concept is much broader than the differential one, and that the differential principle is a subdivision of the operational angle concept.

7.2 Shay-PA UTILISATION OF THE OPERATIONAL ANGLE

Shay-PA utilises θ_{opr} in the derivation of U_{PA} with the use of two nonlinear Gausian functions P and N shown in figure 7.2. The x-axis used in the figure is θ_{opr} and the angle in the x-axis where P=N is called the switching angle θ_{sw}. The two functions P and N are mathematically represented as in equations (7.1) and (7.2). The two boundary angles, θ_{opr_min} and θ_{opr_max} are the angles that limit the operational angle range used. This is to be clarified in later sections of this chapter.

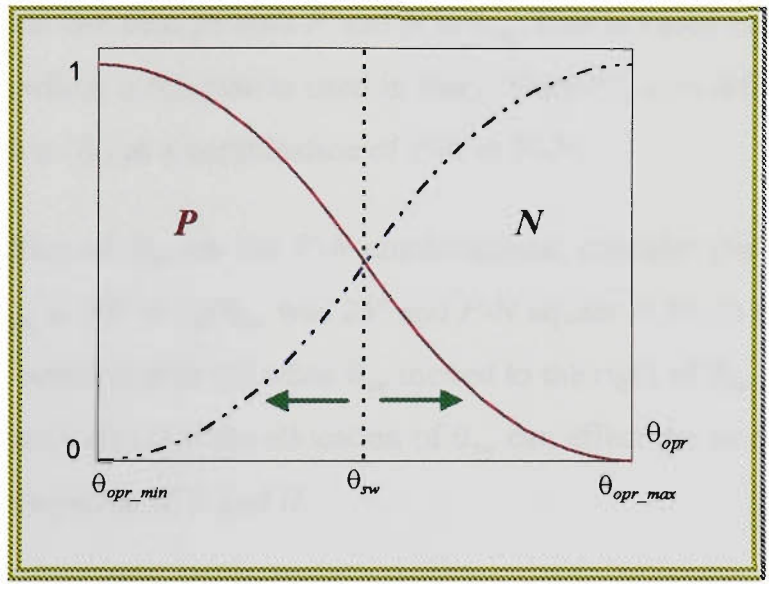


Figure 7.2 two Guassian functions, *P* and *N*

$$P=\begin{cases}0 & \theta_{opr}<\theta_{opr\_min}\\ 1 & \theta_{opr}=\theta_{opr\_min}\\ 1-2^{\left(\frac{\theta_{opr}+\theta_{opr\_min}-2\theta_{sw}}{2(\theta_{sw}-\theta_{opr\_min})}\right)^2} & \theta_{opr\_min}<\theta_{opr}\le\theta_{sw}\\ 2^{\left(\frac{\theta_{opr}+\theta_{opr\_min}-2\theta_{sw}}{2(\theta_{sw}-\theta_{opr\_min})}\right)^2} & \theta_{sw}<\theta_{opr}\le(2\theta_{sw}-\theta_{opr\_min})\\ 0 & \theta_{opr}>(2\theta_{sw}-\theta_{opr\_min})\end{cases} \quad (7.1)$$

and

$$N = \begin{cases} 0 & \theta_{opr} < \left(2\theta_{sw} - \theta_{opr\_max}\right) \\ 2^{\left(\frac{\theta_{opr}+\theta_{opr\_max}-2\theta_{sw}}{2\left(\theta_{opr\_max}-\theta_{sw}\right)}\right)^2} & \left(2\theta_{sw} - \theta_{opr\_max}\right) < \theta_{opr} \le \theta_{sw} \\ 1-2^{\left(\frac{\theta_{opr}+\theta_{opr\_max}-2\theta_{sw}}{2\left(\theta_{opr\_max}-\theta_{sw}\right)}\right)^2} & \theta_{sw} < \theta_{opr} \le \theta_{opr\_max} \\ 1 & \theta_{opr} = \theta_{opr\_max} \\ 0 & \theta_{opr} > \theta_{opr\_max} \end{cases} \quad (7.2)$$

Equations (7.1) and (7.2) show that P and N are dependent on θ_{opr} and θ_{sw}. Knowing that θ_{opr} is dependent on the plant dynamics, and there is no on-line control on its value makes the only variable that can change both P and N is θ_{sw}. This is based on the assumption of having constant A, which is the case as used in Shay. Shay-PA uses different allocations of θ_{opr} in order to derive U_{PA} as a combination of *P-N* or *N-P*.

To illustrate the effect of θ_{sw} on the *P-N* combinations, consider the example shown in figure 7.3 where θ_{opr} is 50° in (a) θ_{sw} was 25° and *P-N* equals -0.84, in (b) θ_{sw} was 45° and *P-N* became -0.21, notice that in (c) when θ_{sw} moved to the right of θ_{opr}, *P-N* became 0.72. This example shows clearly that the allocation of θ_{sw} can effect the amplitude and the sign of the resultant combination of P and N.

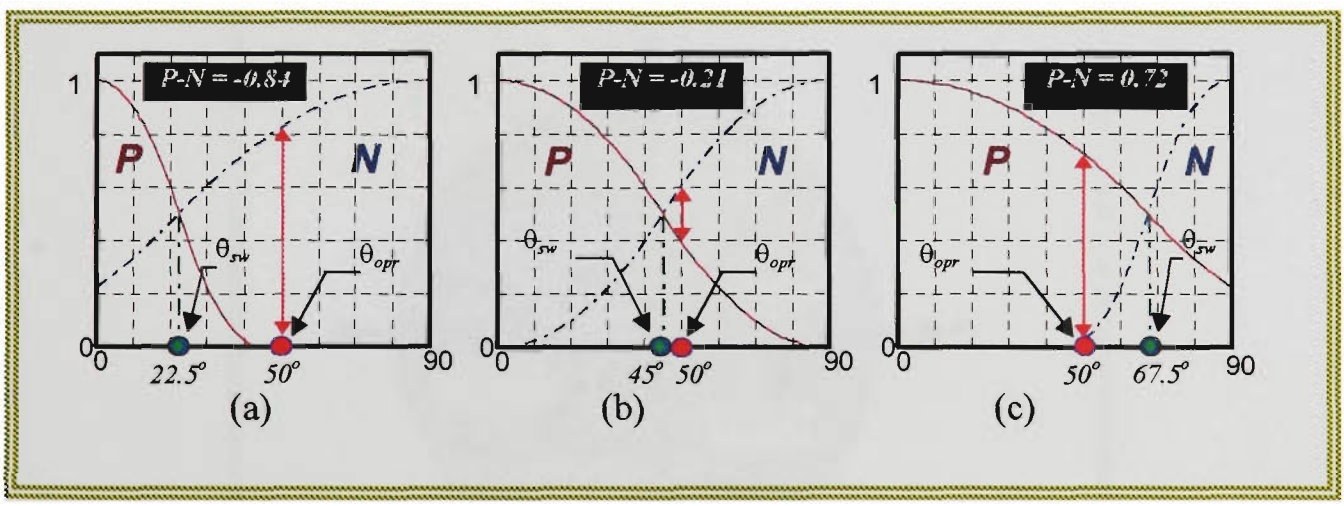


Figure 7.3 effect of different θ_{sw} on *P* and *N*

(a) θ_{sw}=22.5°,θ_{opr}=50°
(b) θ_{sw}=45°, θ_{opr}=50°
(c) θ_{sw}=67.5°, θ_{opr}=50°

The allocation of θ_{sw} in Shay-PA is achieved via fuzzy processing. The methodology implemented in the allocation utilises both the operational sector and the operational angle concepts. This procedure is explained in the following section.

7.3 UTILISATION OF THE OPERATIONAL SECTOR AND THE OPERATIONAL ANGLE CONCEPTS IN SHAY-PA

The main focus in this stage is the dynamics of the plant under control. The goal is to provide the tuning signal U_{PA} that enhances the plant operation under transient and steady-state conditions. The solution is to utilise both the operational sector vector (*sct*) and the operational angle information in the derivation of the switching angle θ_{sw}.

Referring back to figure 5.5 which shows the six sectors in a two dimensional plane. The sectors are defined in a Boolean manner with clear angles limiting their angular boundaries as shown in figure 7.4.

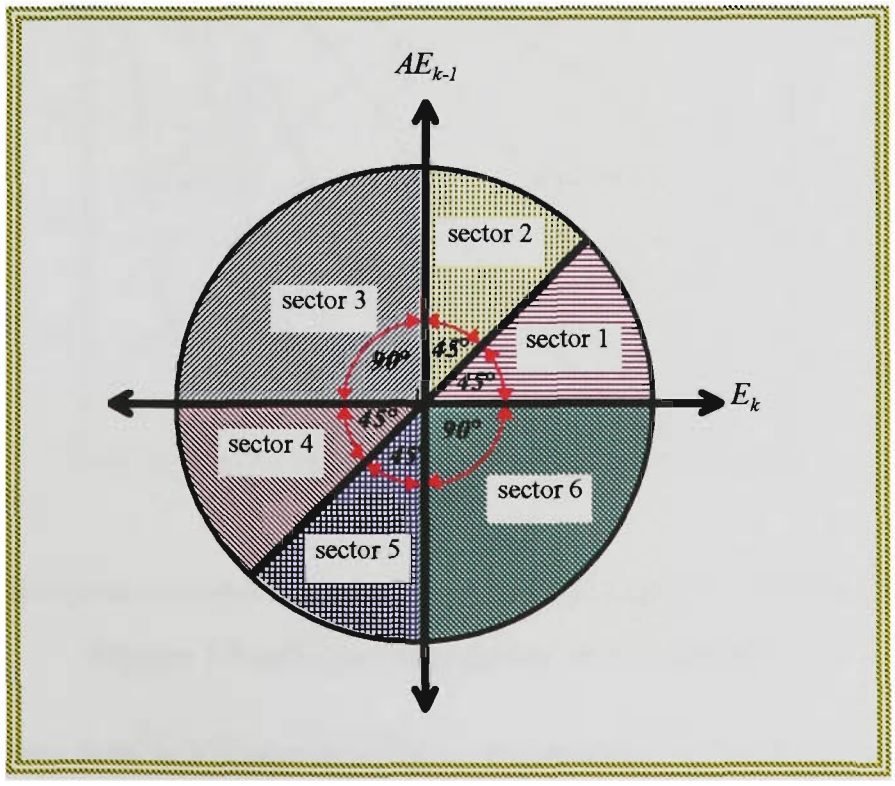


Figure 7.4 the angular boundaries of the sectors

In Shay-PA every sector is represented with a set of *P* and *N* functions with θ_{opr_min} *and* θ_{opr_max} defined according to table 7.1 for each sector. A different rule table is used for each sector to derive θ_{sw}. An example is shown in figure 7.5 which represents different allocations of θ_{sw} in sector 1. Notice that θ_{opr_min} and θ_{opr_max} are set to 0 and 45 degrees respectively for sector 1.

sector	θ_{opr_min}	θ_{opr_max}
1	0°	45°
2	45°	90°
3	90°	180°
4	180°	225°
5	225°	270°
6	270°	360°

Table 7.1 range of θ_{opr} for each sector

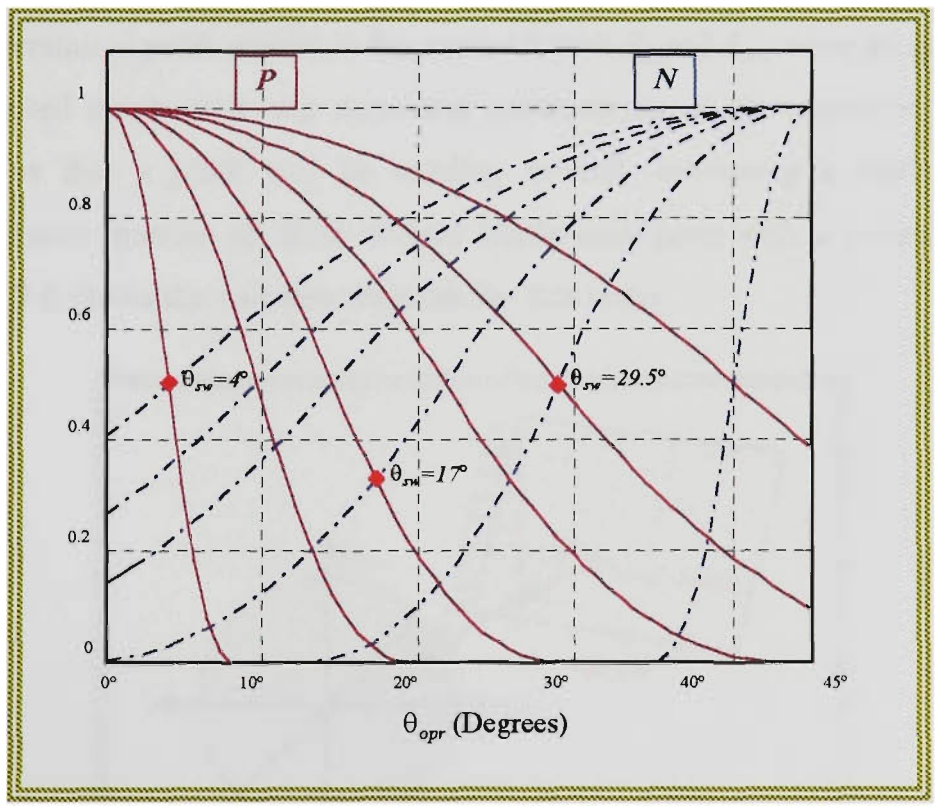


Figure 7.5 different allocations of θ_{sw} in sector 1

Shay-PA can been seen as a hierarchy of fuzzy nodes where *sct* parameters are used as the *Enable* signal of the fuzzy node, according to the current operational sector (see section 7.5).

7.4 CALCULATION OF THE SWITCHING ANGLE

The derivation of θ_{sw} in Shay-PA is performed by fuzzy processing of two inputs in every sector. The inputs to the fuzzy rule base are θ_{sw} and $|E_k|$. $|E_k|$ was used in order to give a better realisation of the plant's current state. At steady-state and while having very minor deviations of the performance monitor from the desired value one can still have a large operational angle that might mislead fuzzy processing of θ_{opr}, ie. if the performance monitor used was the error signal, θ_{opr} will be 63.43°, if E_{k-1} was 10 and E_k was 5, and it will have the same value if E_{k-1} was 10^{-4} and E_k was 5^{-4}, with $A=1$.

This section describes the fuzzy nodes (rule-tables) in detail with considerations applied for constructing the nodes for each sector. The description provided next, assumes an error signal as a performance monitor (E_k), where E_k=*Ref-plant output. Ref=1* and $A=1$.

7.4.1 Sector 1

The operational point will be in this sector if both E_k and E_{k-1} were greater than the desired steady-state. An important characteristic of the operation in this sector is that a plant will be heading towards increasing a drift in the performance monitor off from desired steady-state point with a positive ΔE_k. Figure 7.6 shows the unit step response for this sector.

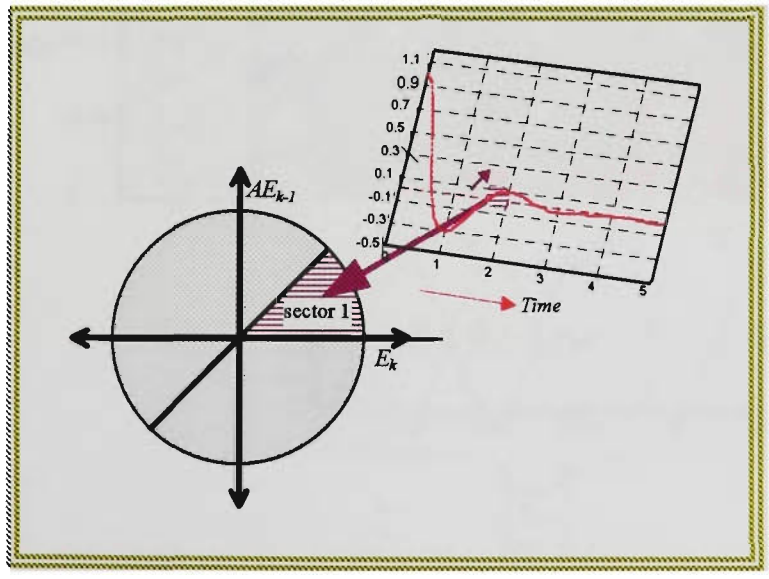


Figure 7.6 operation in sector 1

The control objective in this sector is devoted towards preventing extra building up of the system acceleration towards an overshoot in E_k. U_{PA} is calculated as *P-N* combination in this sector with θ_{sw} tending to be greater or equal to θ_{opr}. The FLC used in sector 1 to derive θ_{sw} is shown in figure 7.7. The figure shows that the range of θ_{sw} (the output universe of discourse) is from 0° to 56.25° while θ_{opr} ranges from 0° to 45°. This is done to ensure the possibility of having θ_{sw} greater or equal to θ_{opr} for all conditions. Figure 7.8 shows the control surface of sector 1 FLC. Note that this is a unidirectional (the rules are in one direction, acceleration) FLC and that U_{PA} is directly proportional to $|E_k|$ and inversely proportional to θ_{opr}. This inverse proportionality is there, because ΔE_k will have lower values at the upper boundaries of θ_{opr} in sector 1, which indicates that the plant speed towards increasing the error is very low and vice versa.

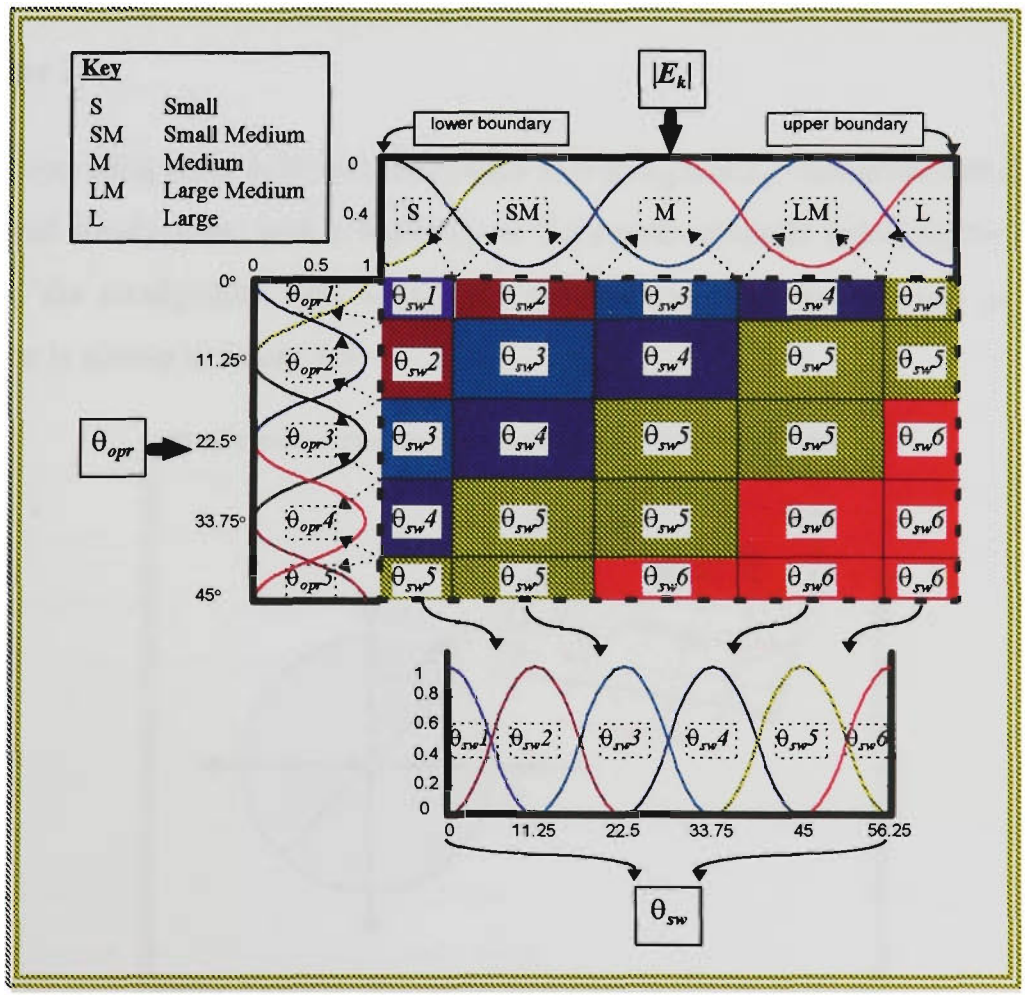


Figure 7.7 FLC used for sector 1

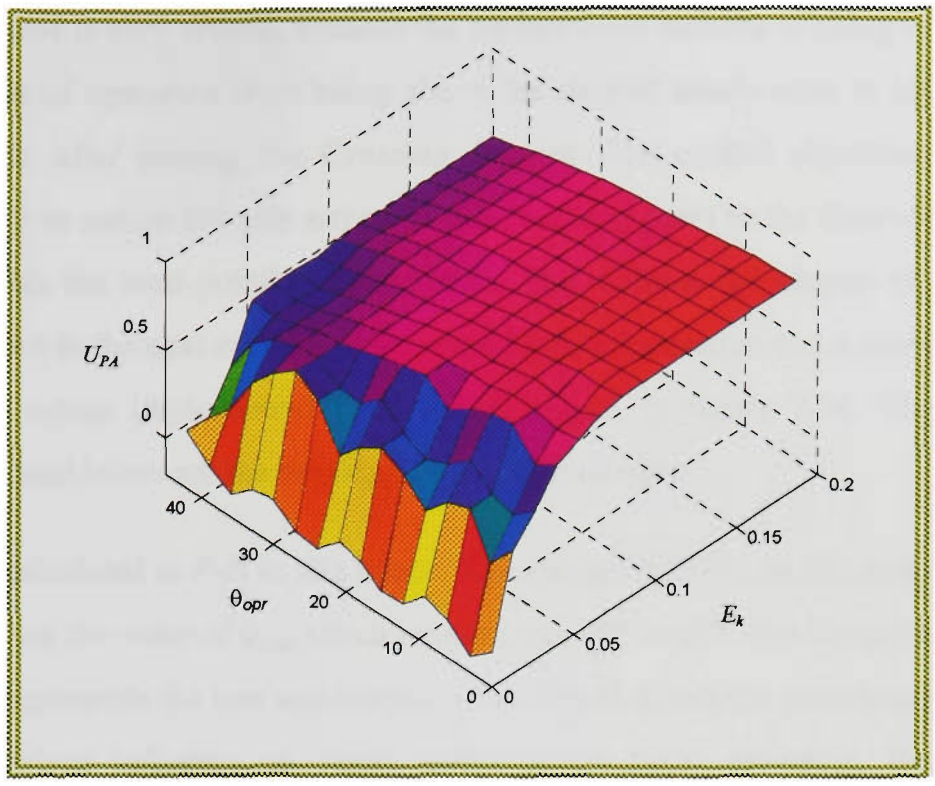


Figure 7.8 control surface for sector 1 FLC

7.4.2 Sector 2

The operating point will be in this sector if both E_k and E_{k-1} are greater than the desired steady-state, with a tendency of the system towards reducing the drift from the steady-state with a negative acceleration. The plant's state in this sector is shown in figure 7.9

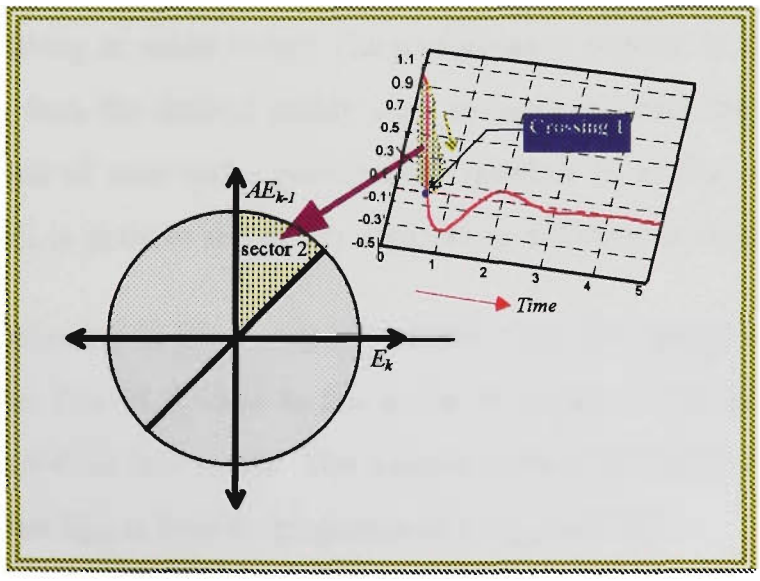


Figure 7.9 operation in sector 2

This sector is very critical, because the performance monitor is going to switch its mode of operation from being above the desired steady-state to becoming below it, after passing the **Crossing 1** point. The control objective in this sector, is to secure the safe arrival of the operating point to the desired steady-state with the least possible acceleration. This reduces the chance of greater overshoot in the next stage. It is achieved by an acceleration and a deceleration control action implemented by the FLC shown in figure 7.10. The black background boxes are the breaking (deceleration) rules.

U_{PA} is calculated as *P-N* in this sector. The allocation of θ_{sw} in this sector relies heavily on the value of θ_{opr} which ranges from 45° to 90°. The lower boundary of θ_{opr} represents the low acceleration area. This is where the plant acceleration has a minor influence on future states of the plant. However, the upper boundaries of θ_{opr} range are in the high acceleration range, where the plant dynamics are of great influence on future states of the plant. This is why the breaking rules were implemented at high angles with lower $|E_k|$ values. The control surface of sector 2 FLC is shown in figure 7.11. The figure shows the inverse proportionality between U_{PA} and θ_{opr} and the direct proportionality between U_{PA} and $|E_k|$.

7.4.3 Sector 3

This is a switching in mode sector. The performance monitor is changing from being greater than the desired steady state to become lower than it. So for a reference signal of zero and a performance monitor E_k as the error, sector 3 occurs when E_k is positive and E_{k-1} is negative as indicated in figure 7.12.

The control objective in this sector is to dump down the system speed towards the new mode. The FLC used in this sector is shown in figure 7.13. U_{PA} is calculated as *N-P* in this sector. The control surface of sector 3 FLC, figure 7.14, shows that U_{PA} is directly proportional to θ_{opr} and $|E_k|$.

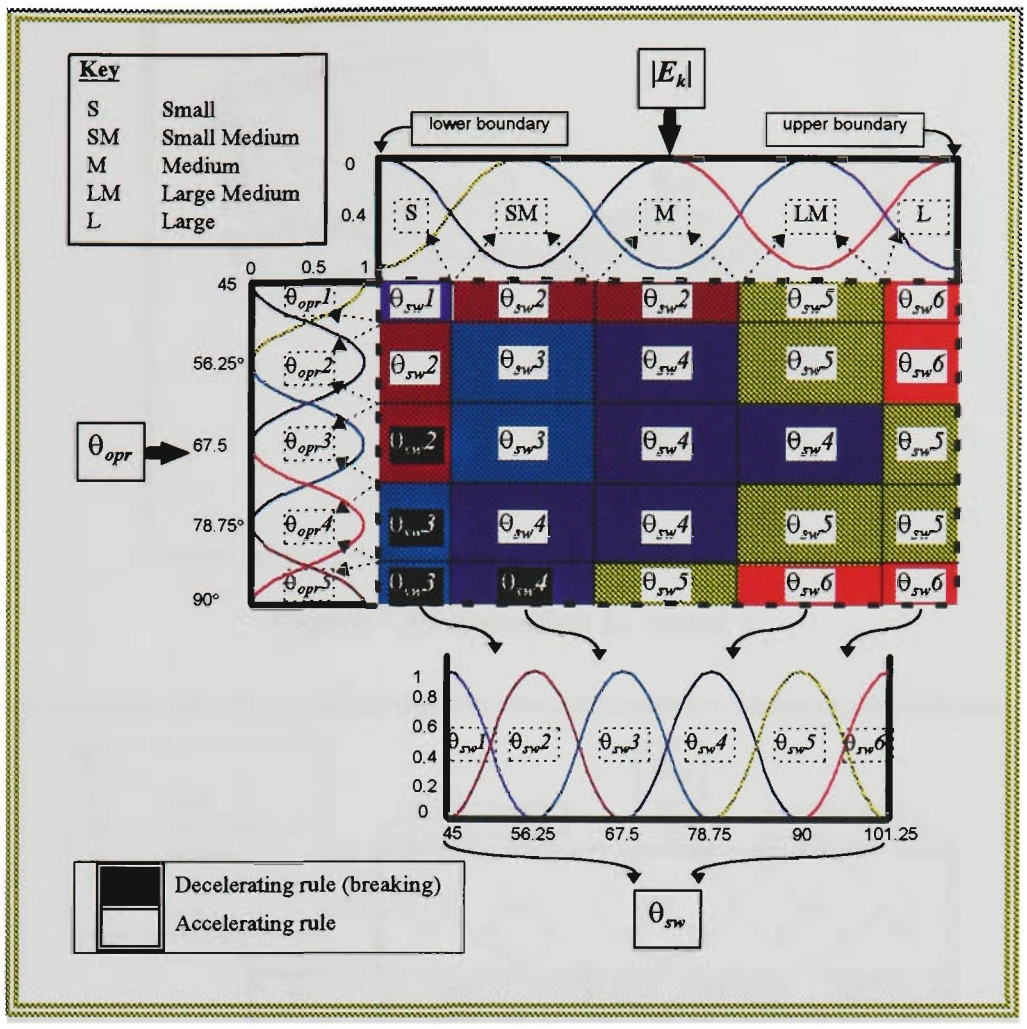


Figure 7.10 FLC used for sector 2

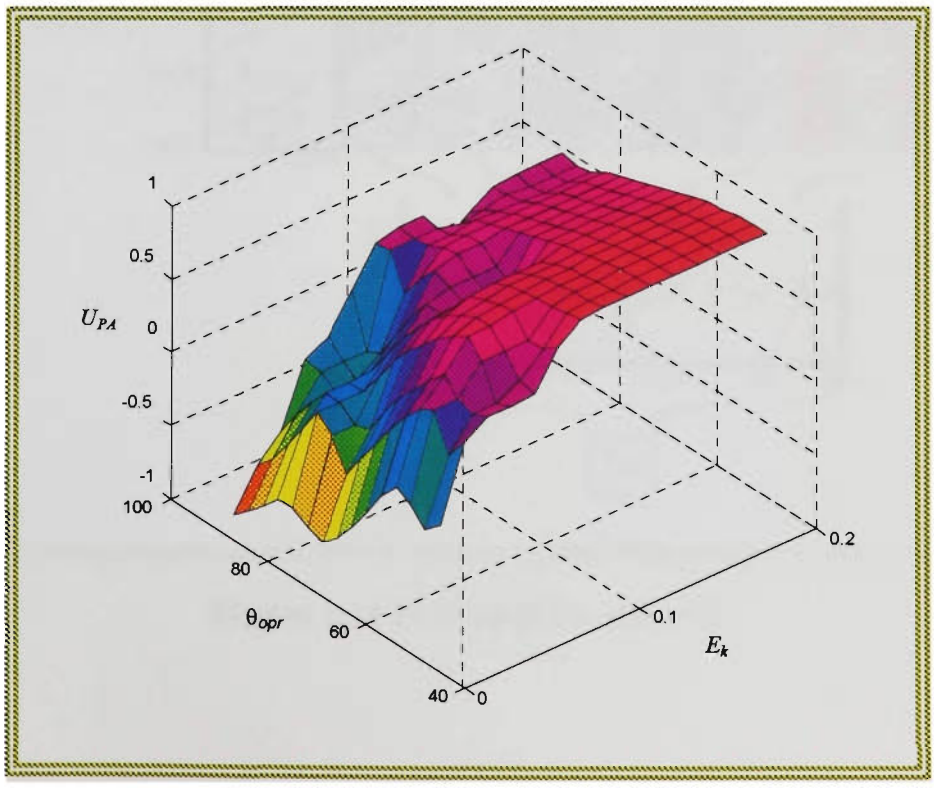


Figure 7.11 control surface for sector 2 FLC

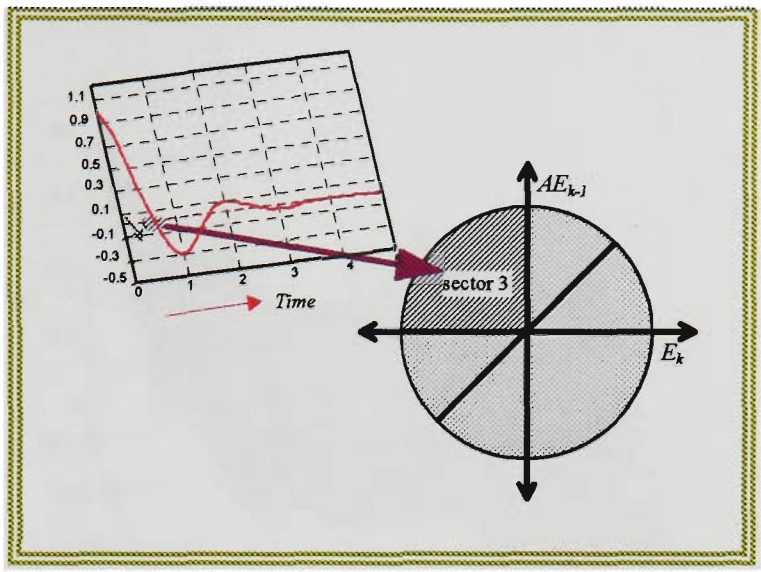


Figure 7.12 operation in sector 3

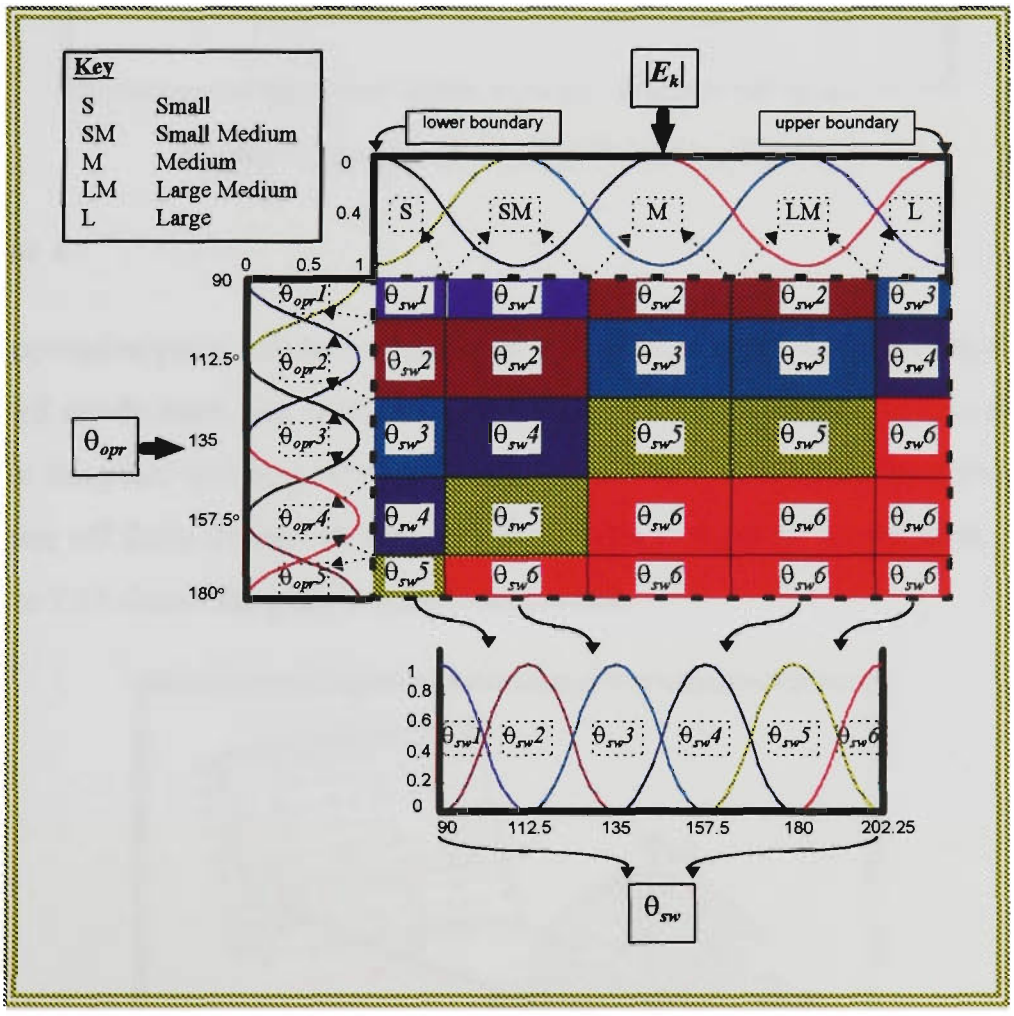


Figure 7.13 FLC used for sector 3

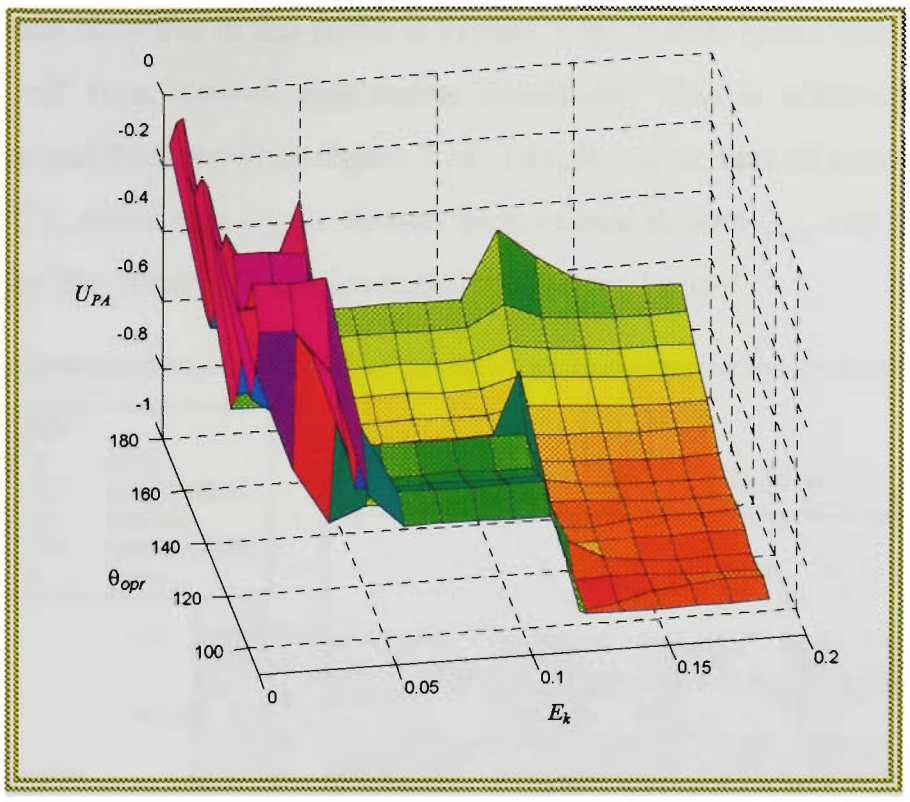


Figure 7.14 control surface for sector 3 FLC

7.4.4 Sector 4

The operation point will be in this sector if both E_k and E_{k-1} are lower than the desired steady-state. An important characteristic of the operation in this sector is that the plant will be heading towards increasing a drift of the performance monitor off from desired steady-state point with a negative acceleration, ΔE_k. Figure 7.15 shows the plant's state in this sector.

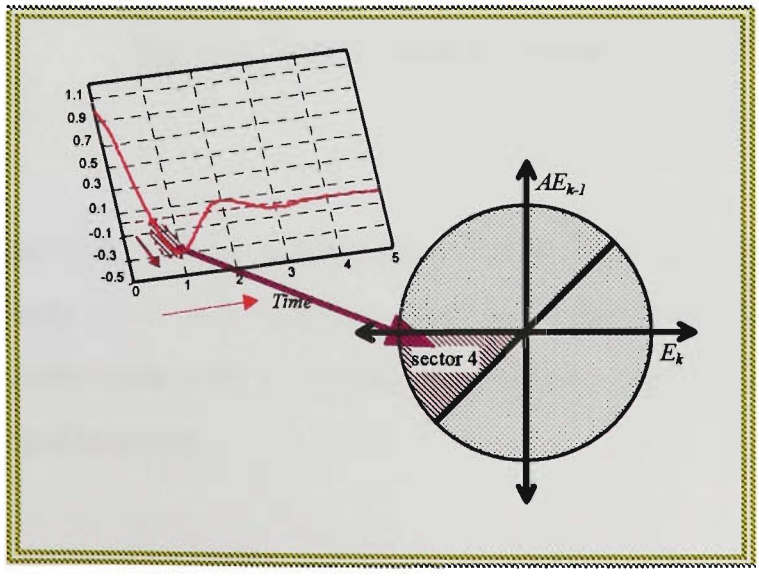


Figure 7.15 operation in sector 4

The control objective in this sector is to reduce the system speed towards extra drifting off from desired steady-state conditions. This is achieved by the unidirectional FLC shown in figure 7.16. The control surface of sector 4 FLC, figure 7.17, shows that U_{PA} is directly proportional to both θ_{opr} and $|E_k|$. Note the similarities between the characteristics of sector 1 and 4.

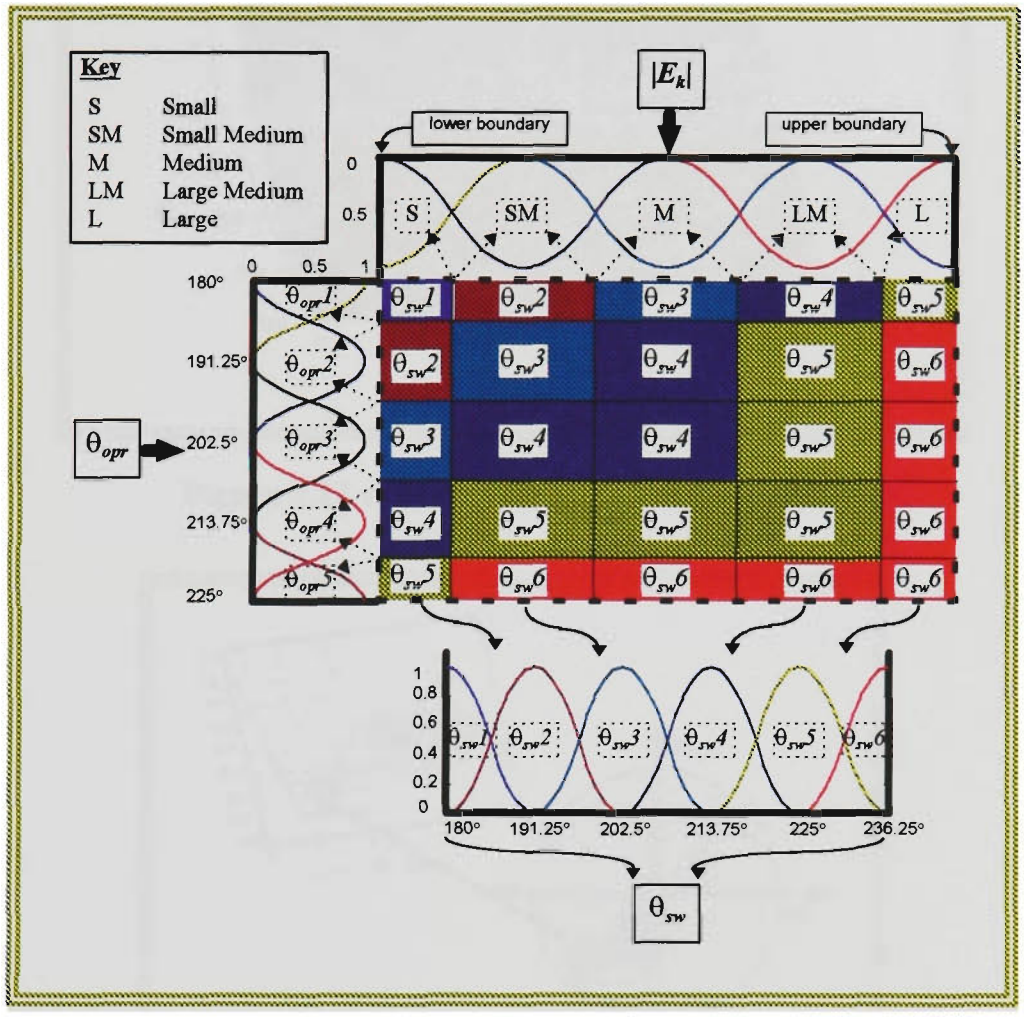


Figure 7.16 FLC used for sector 4

7.4.5 Sector 5

The operational point will be in this sector if both E_k and E_{k-1} are lower than the desired steady-state, with a tendency of the system towards reducing the drift in the steady-state with a positive acceleration. Figure 7.18 shows the plant's status in this sector.

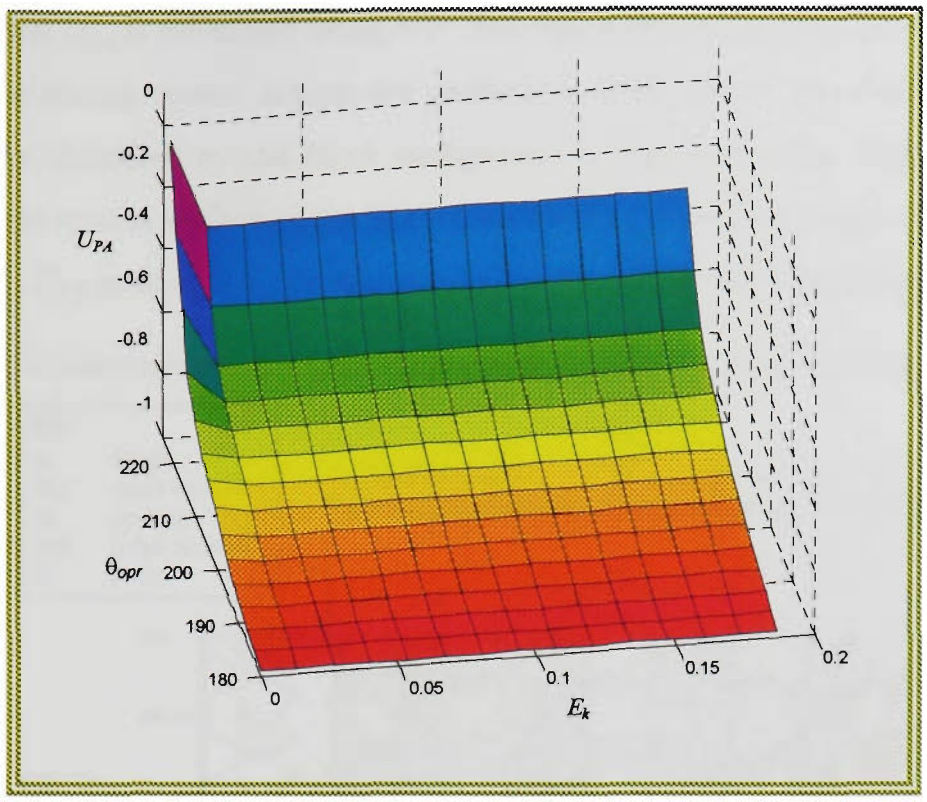


Figure 7.17 control surface for sector 4 FLC

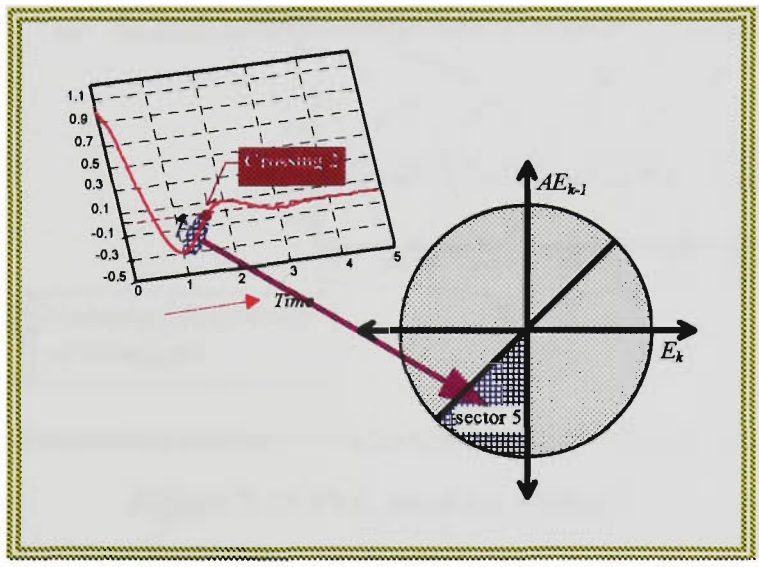


Figure 7.18 operation in sector 5

This a critical sector because the plant dynamics are of great influence on the current and future states of the operation. There is a great number of similarities between this sector and sector 2. The plant status tends to cross the **Crossing 2** point. The control action required in this sector should consider providing breaking signals to reduce the chances of larger overshoots in the state after **Crossing 2**. The FLC used in this sector is shown in figure 7.19.

The signal U_{PA} is calculated using *N-P*. The figure shows that both accelerating and decelerating control actions are performed in this sector. The deceleration rules are identified by the black background in the rules cells. Figure 7.20 shows the control surface of the FLC of sector 5 as an inverse image of that of sector 2. U_{PA} is directly proportional to $|E_k|$ and inversely proportional to θ_{opr}.

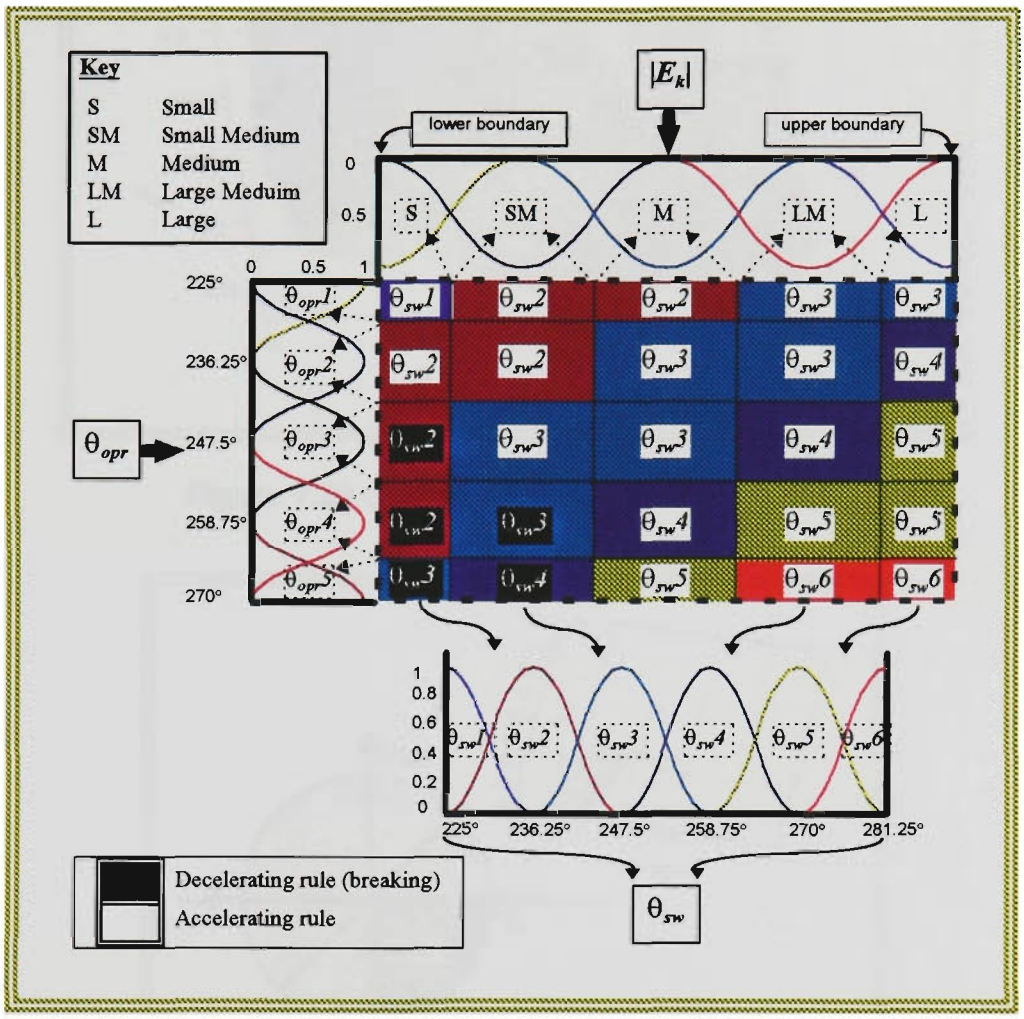


Figure 7.19 FLC used for sector 5

7.4.6 Sector 6

This is a switching mode sector, where E_k is becoming greater than the desired steady-state, E_k is negative and E_{k-1} is positive, as shown in figure 7.21.

The control objective in this sector is to dump down the system speed towards a new mode. The FLC used in this sector is shown in figure 7.22. U_{PA} is calculated as *N-P* in this sector. The control surface of sector 6 FLC, shown in figure 7.23, shows that U_{PA} is directly proportional to θ_{opr} and $|E_k|$.

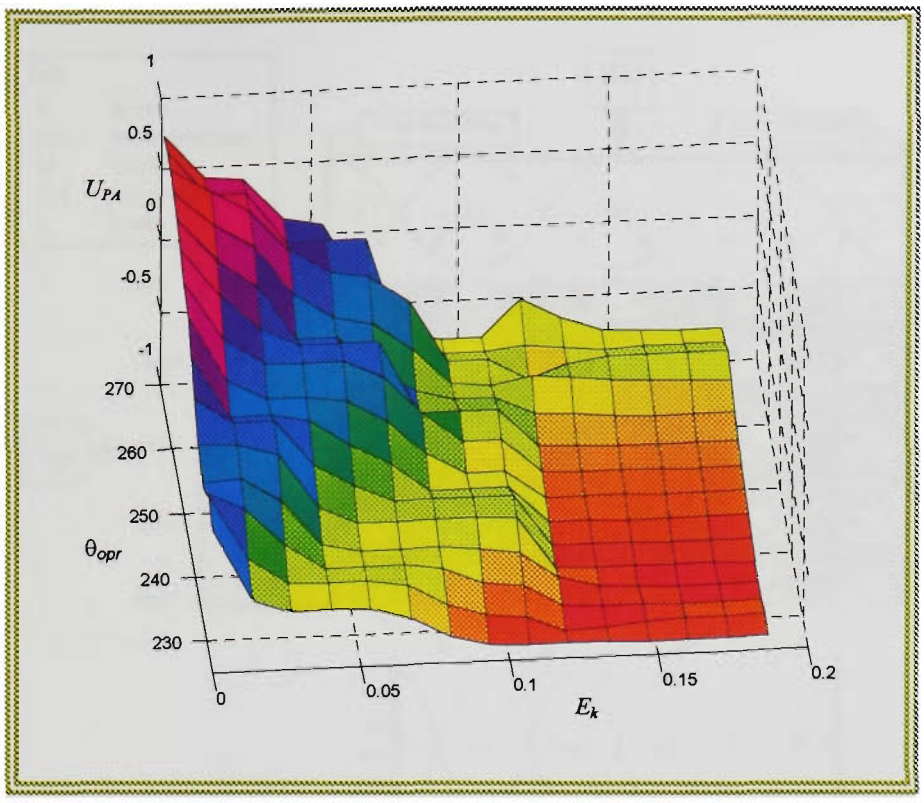


Figure 7.20 control surface for sector 5 FLC

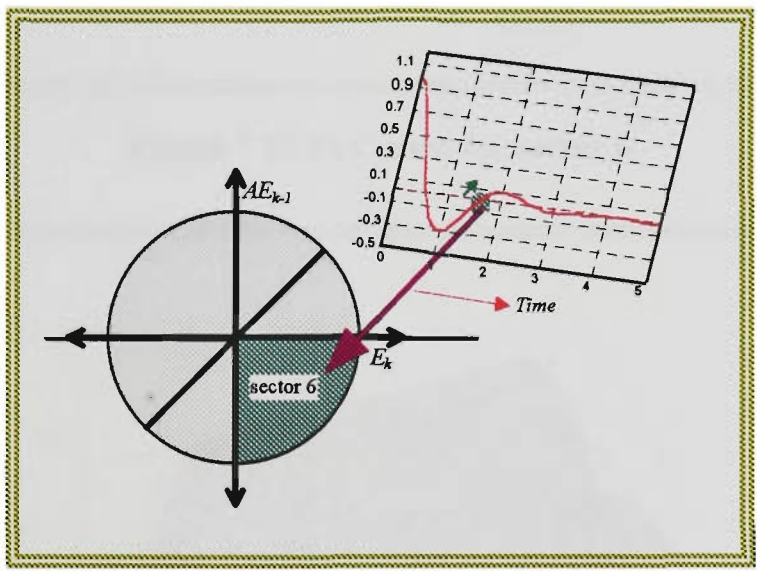


Figure 7.21 operation in sector 6

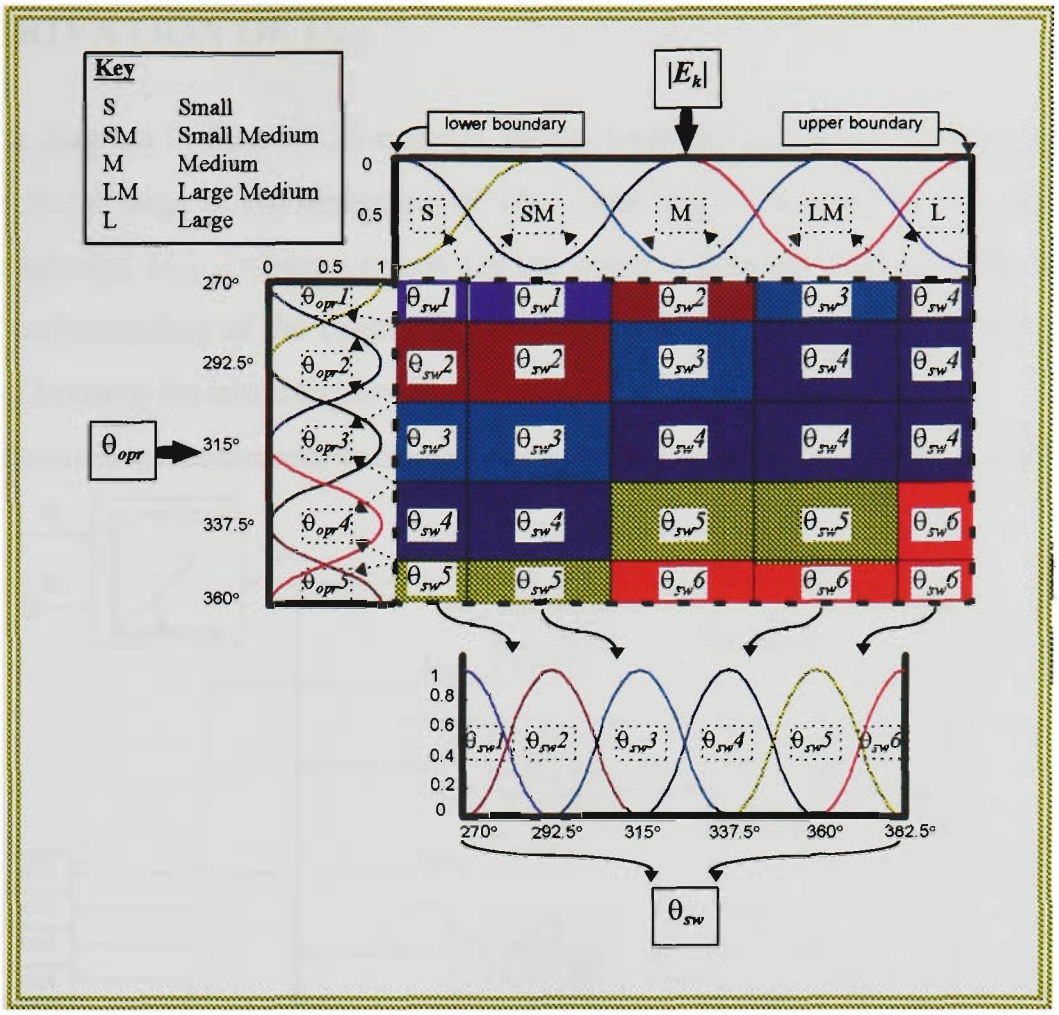


Figure 7.22 FLC used for sector 6

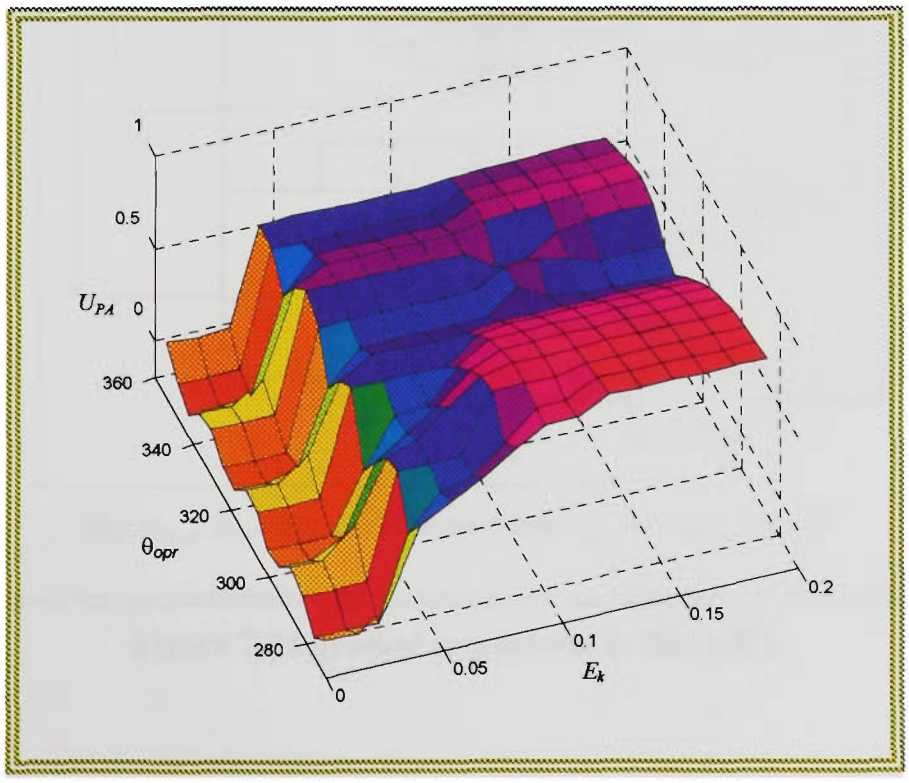


Figure 7.23 control surface for sector 6 FLC

7.5 DERIVATION OF U_{PA}

The block diagram in figure 7.24 explains in full detail the logical operations in Shay-PA from the initial stage to the derivation of U_{PA}. Note that in figure 7.24, U_{PA} out of each sector is labelled U_{PA_si} where i stands for the sector's number. This notation is used to ease the understanding of the figure. The Enable signal shown in figure 7.24 triggers the switching between the hierarchy levels.

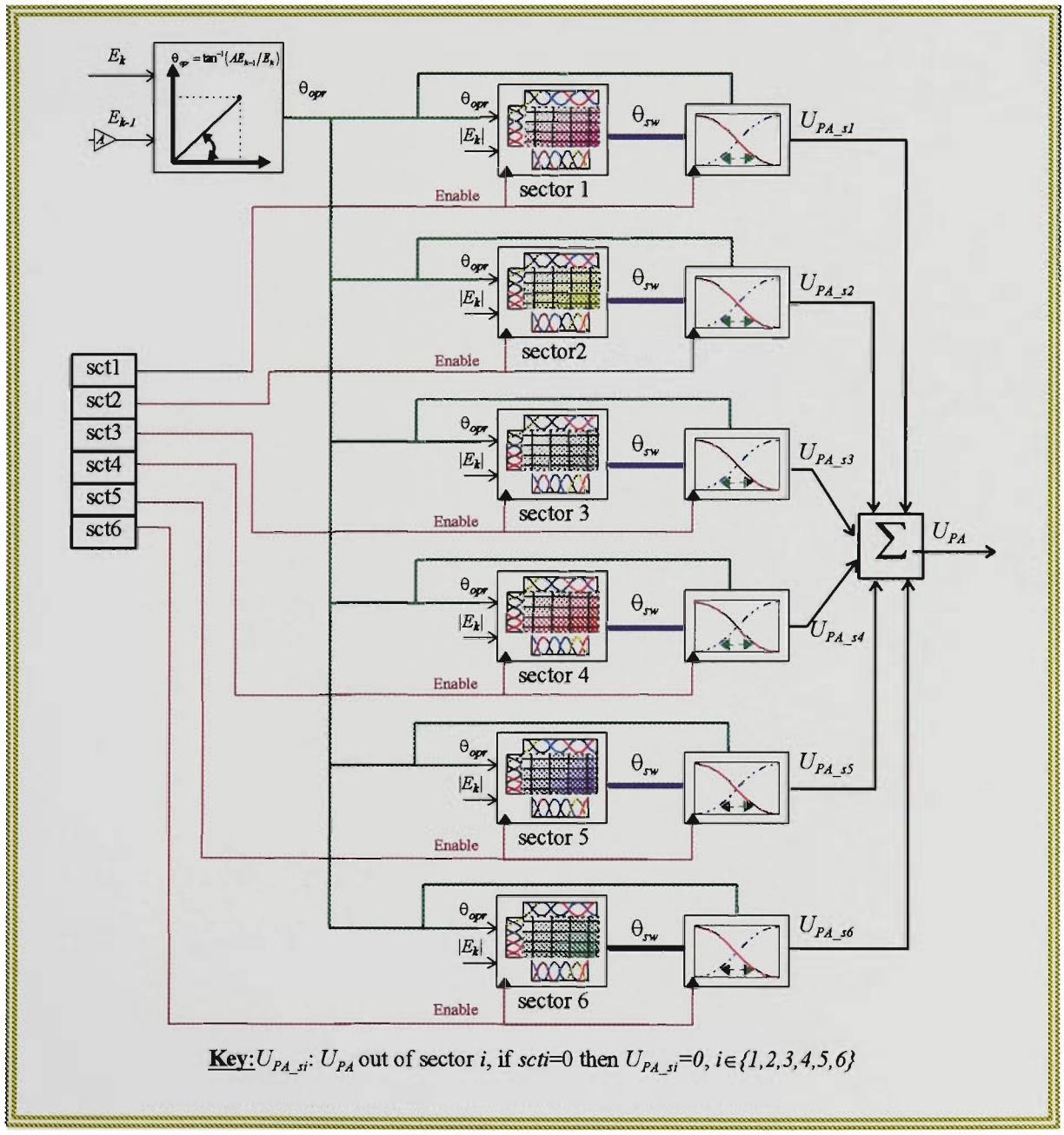


Figure 7.24 general procedures in Shay-PA

The procedure to generate U_{PA} starts with determining the vector *sct* and the operational angle θ_{opr}. Both variables are generated from E_k and AE_{k-1}. The active sector (having the corresponding bit in *sec* to be 1) enables the FLC for that sector, which in turn produces θ_{sw} using θ_{opr} and $|E_k|$. Then the same *sct* bit will enable the *P-N* block, so the signal U_{PA_si} is calculated. Note that in figure 7.24, the last summation point is there to show the modelling of Shay-PA in the form of a block diagram. At every k time, only one level of the hierarchy is going to be enabled and thus the output of other levels is zero.

Chapter 8 *Shay Input and Output Ranges Tuner (Shay-Tune)*

8.0 INTRODUCTION

This chapter describes, in detail, a novel approach for an on-line adaptive FLC ranges updating procedure. Shay-Tune is responsible for updating the two global normalising factors G_{in} and G_{out}. The updating procedure used is based on the plant performance and the Main-FLC status. It was mentioned in chapter 5 that the Main-FLC is acting as the fuzzy reference for the Shay structure. This is to be clarified in this chapter.

A learning period called L is required by Shay-Tune to produce the necessary updates for G_{in} and G_{out}. The period L is selected by the designer prior to the system running. The guidelines for the selection of the L parameter are provided in chapter 9. The plant status is represented by the *sct* vector and $\overline{E_k}$ (the non normalised performance monitor). The Main-FLC operational status is represented by *IOS* and *OOS* vectors as explained in chapter 5, which are monitored during the L period.

The following sections of this chapter explain an integrated hierarchy of Boolean and fuzzy logic used in the updating procedure.

8.1 PLANT PERFORMANCE CONSIDERATIONS IN TUNING

The plant performance is monitored during the L time. The operational sectors represented by *sct* and $\overline{E_k}$ are the two indicators used for this purpose.

8.1.1 Operational Sector Information

The operational sector information is collected during the L time. The sectors are used to categorise the plant operation under three categories called modes. The three modes (*mode 1*, *mode 2* and *mode 3*) are used to switch between the updating hierarchy strategy levels, as shown later.

The modes are representatives of the average occurrence of particular sectors during L time. Sectors 1 and 2 are represented by *mode 1*, sectors 4 and 5 are represented by *mode 2* and sectors 3 and 6 are represented by *mode 3*.

Careful analysis of the modes and the associated sectors can explain the selection of the modes. *Mode 1* is the case where the performance monitor is greater than the desired value. This case occurs when the plant is in either sector 1 or 2. *Mode 2* is when the performance monitor is less than the desired value, which is the case in sectors 4 and 5. *Mode 3*, represents a transient mode as the plant state is moving between *mode 1* and *2*, as in the case of sectors 3 and 6. Later sections in the chapter show that *mode 3* is the complement of the algebraic sum of *mode 1* and *mode 2*.

8.1.1.1 Determining the current mode

A flag called $SFLG_i$, $i \in \{1,2,3,4,5,6\}$, is assigned for every sector, where $SFLG_i$ is the flag of sector i. The values of these flags are updated during the L period in the form

$$SFLG_i = \sum_{k=0}^{M-1} sct[i](k)$$

where M is the total number of samples collected during L period and k is the current sample.

After every L period, the strength of *mode 1* (*md1*) is calculated as

$$md1 = \frac{SFLG_1 + SFLG_2}{M}$$

and the strength of mode 2 (*md2*) is calculated as

$$md2 = \frac{SFLG_4 + SFLG_5}{M}$$

It should be noted here that *md1* and *md2* range from 0 to 1. The logic implemented in this function implies that the strength of *mode 3* (*md3*) is 1-

(*md1+md2*) or *md3=(md1+md2)`*. More over *md1+md2+md3=1* and this stands for all cases.

In order to give the designer more freedom and control on the performance of the Shay-Tune scheme, a threshold value (T) ranging from 0 to 1 is used for the mode choice. For more details on the optimisation of T and its effect on overall Shay performance refer to chapter 9.

The procedure to determine the operational mode starts with comparing *md1* to *md2*. The largest is then compared to T. The mode is decided to be either *mode 1* or *mode 2* according to the condition that the largest one between *md1* and *md2* is larger than T as well. Otherwise the mode chosen will be *mode 3*. Figure 8.1 shows the decision tree used in determining the operational modes.

Setting a mode means that the operation is in that mode and it means resetting the other modes. The modes effect on the ranges updating procedure is shown in later sections.

8.1.2 Non-Normalised Performance Monitor Information

This parameter is used to set some guidelines for the performance monitor levels that can be used to assist any improper settings of the Main-FLC ranges. The performance monitor prior to the updating stage is used to encounter any problem that might emerge due to inaccurate updating decisions.

A moderate number of three $\overline{E_k}$ levels is used. Figure 8.2 shows these levels in the three classes labelled $\overline{e1}$, $\overline{e2}$ and $\overline{e3}$, where the x-axis is $\overline{E_k}$. The levels are assigned fuzzy values according to the value of $\overline{E_k}$.

Note that, the range of $\overline{E_k}$ is set from 0 to 1 and the boundaries of $\overline{e_1}$ and $\overline{e_2}$ are set to 0.1 and 0.9. These two values were found to be reasonable and robust arrangement for the $\overline{E_k}$ range. Changes to these values can cause minor effects on the Shay-Tune performance.

$\overline{E_k}$ levels are being updated and averaged every L period, thus after L time the value of $\overline{e_i}$ equals

$$\overline{e_i} = \frac{\sum_{k=0}^{M-1} \mu_{\overline{e_i}}(\overline{E}_k)}{M}$$

where, $i \in \{1,2,3\}$ and $\mu_{\overline{e_i}}$ stands for the degree of membership of $\overline{E_k}$ to the ith class.

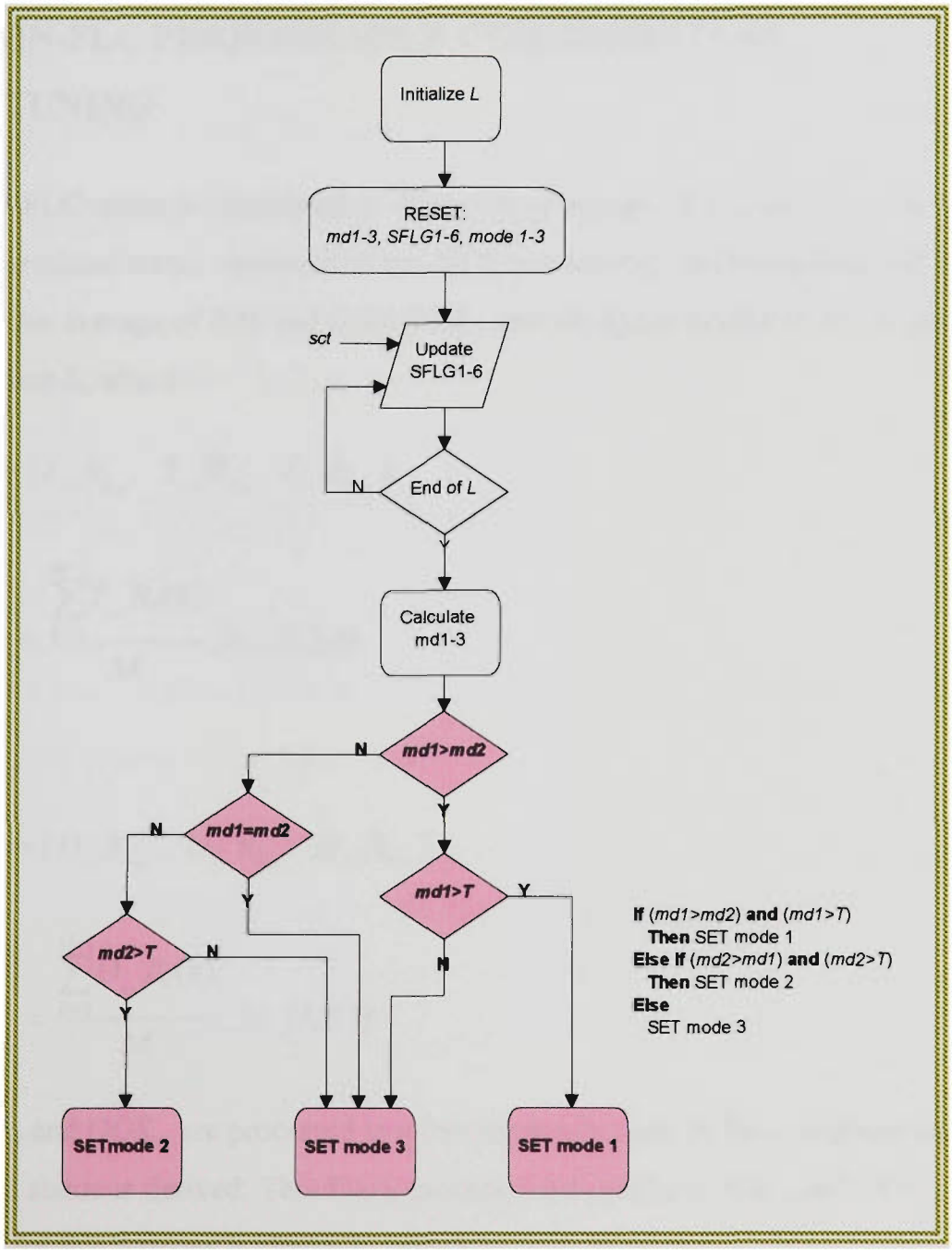


Figure 8.1 the decision tree to select the operational modes

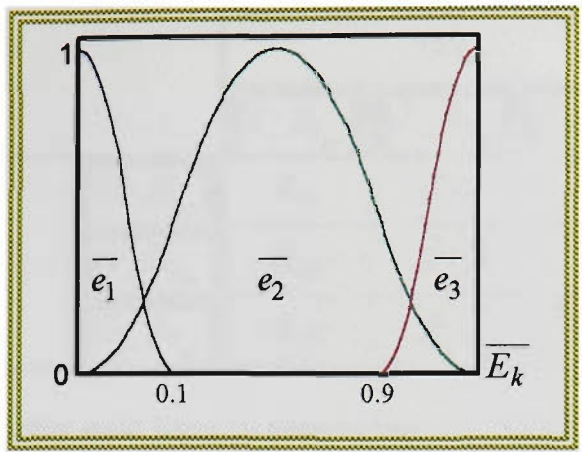


Figure 8.2 $\overline{E_k}$ levels

8.2 MAIN-FLC PERFORMANCE CONSIDERATION IN TUNING

The Main-FLC status is considered in the updating process of G_{in} and G_{out}. The input and output operational status vectors, *IOS* and *OOS* respectively, are being collected during the L period. An average of *IOS* and *OOS* (IOS_{av} and OOS_{av}) is produced at the end of every learning time L, where

$$IOS_{av} = [\,I\_R_{1_{av}} \quad I\_R_{2_{av}} \quad I\_R_{3_{av}}\,]$$

$$I\_R_{i_{av}} = \frac{\sum_{k=0}^{M-1} I\_R_i(k)}{M},\ i \in \{1,2,3\}$$

and

$$OOS_{av} = [\,O\_R_{1_{av}} \quad O\_R_{2_{av}} \quad O\_R_{3_{av}}\,]$$

$$O\_R_{i_{av}} = \frac{\sum_{k=0}^{M-1} O\_R_i(k)}{M},\ i \in \{1,2,3\}$$

Both IOS_{av} and OOS_{av} are processed in a two input rule base. A fuzzy representation of the Main-FLC status is derived. This fuzzy process combines both IOS_{av} and OOS_{av}. The rule base used for this purpose is shown in table 8.1.

		IOS_{av}		
		I_R1_{av}	I_R2_{av}	I_R3_{av}
OOS_{av}	O_R1_{av}	R_{11}	R_{21}	R_{31}
	O_R2_{av}	R_{12}	R_{22}	R_{32}
	O_R3_{av}	R_{13}	R_{23}	R_{33}

Best
Worst

Table 8.1 fuzzy rule base to determine the Main-FLC status

The *min* operator is used to generate the truth values of all R_{xy}s in table 8.1 in the form, $R_{xy} = \min(I\_R_{x_{av}}, O\_R_{y_{av}})$.

Note that the convention used in labelling the R_{xy}s refers x to IOS_{av} and y to OOS_{av},

ie. $x=2 \Rightarrow IOS_{av}[2] \Rightarrow I\_R_{2_{av}}$ *and* $y=3 \Rightarrow OOS_{av}[3] \Rightarrow O\_R_{3_{av}} \Rightarrow R_{23}$.

The R_{xy}s are of crucial importance in drawing the updating strategy implemented in Shay-Tune. This role is explained in later sections of this chapter.

8.3 MANIPULATING MAIN-FLC AND PLANT PARAMETERS

Sections 8.1 and 8.2 showed that the final parameters produced after each L time are: 1) the operational modes, 2) $\overline{e_i}$ s and 3) R_{xy}s.

The operational modes are used to switch the updating strategy in a Boolean logic, creating three virtual levels of processing. In practice two levels are used as *modes 1* and *2* are similar in nature. So what is produced for *mode 1* is the complement of what is produced for *mode 2*. The $\overline{e_i}$ s and R_{xy}s are used in every level in order to generate three updating parameters, *Fix*, *Dec* and *Inc* where:

- ***Fix*** stands for fix (keep constant) the domain range under consideration
- ***Dec*** stands for decrement the domain range under consideration

- ***Inc*** stands for increment the domain range under consideration.

Both the input and output domains are considered in the updating process in every mode. Two parallel processes are implemented in every mode. This parallel processing results in *Fix_In*, *Dec_In* and *Inc_In* for the input domain and *Fix_Out*, *Dec_Out* and *Inc_Out* for the output domain.

The $\overline{e_i}$ s and the R_{xy}s are utilised in a 9×3 rule base that results in a 27 updating situation matrix as shown in table 8.2.

	$\overline{e_1}$	$\overline{e_2}$	$\overline{e_3}$
R_{11}	$F_{11,1}$	$F_{11,1}$	$F_{11,1}$
R_{21}	$F_{21,1}$	$F_{21,2}$	$F_{21,3}$
R_{31}	$F_{31,1}$	$F_{31,2}$	$F_{31,3}$
R_{12}	$F_{12,1}$	$F_{12,2}$	$F_{12,3}$
R_{22}	$F_{22,1}$	$F_{22,2}$	$F_{22,3}$
R_{32}	$F_{32,1}$	$F_{32,2}$	$F_{32,3}$
R_{13}	$F_{13,1}$	$F_{13,2}$	$F_{13,3}$
R_{23}	$F_{23,1}$	$F_{23,2}$	$F_{23,3}$
R_{33}	$F_{33,1}$	$F_{33,2}$	$F_{33,3}$

Table 8.2 manipulating the $\overline{e_i}$ s and the R_{xy}s

The truth value of any $F_{xy,z}$ in table 8.2 is calculated as $F_{xy,z}=min(R_{xy}, \overline{e_i})$.

The $F_{xy,z}$s can be labelled either *Fix*, *Inc*, or *Dec* according to the updating strategy used for the particular domain.

8.4 UPDATING STRATEGIES FOR INPUT AND OUTPUT RANGES

The updating strategies used in Shay-Tune are based on the assumption that the ideal and desired R_{xy} state is R_{11}. The updating effort is directed towards bringing the system to this origin state.

Multilevel strategies are used for updating. This is done to ensure a smooth transition towards the desired state while avoiding large changes in the domain ranges that might have a negative effect on the system stability.

8.4.1 Modes 1 and 2

The updating strategy used for these modes is shown in figure 8.3.

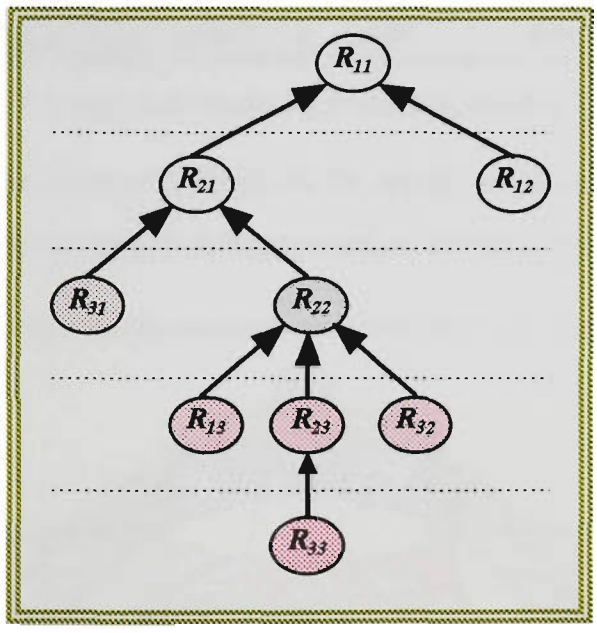


Figure 8.3 updating strategy in modes 1 and 2

8.4.1.1 Input domain

Table 8.3, shows the implementation of the updating strategy shown in figure 8.3 for the input domain in *mode 1* and *mode 2*. According to the table the *Fix_In*, *Dec_In* and *Inc_In* are calculated as

$$Fix\_In=max(F_{11,1},F_{12,1},F_{22,1},F_{23,1},F_{21,2},F_{31,2},F_{32,2},F_{13,2},F_{33,2},F_{21,3},F_{31,3},F_{22,3},F_{32,3},F_{13,3},F_{33,3})$$

$Inc\_In=max(F_{13,1},F_{11,2},F_{12,2},F_{22,2},F_{23,2},F_{11,3},F_{12,3},F_{23,3})$

$Dec\_In=max(F_{21,1},F_{31,1},F_{32,1},F_{33,1})$

	$\overline{e_1}$	$\overline{e_2}$	$\overline{e_3}$
R_{11}	Fix	Inc	Inc
R_{21}	Dec	Fix	Fix
R_{31}	Dec	Fix	Fix
R_{12}	Fix	Inc	Inc
R_{22}	Fix	Inc	Fix
R_{32}	Dec	Fix	Fix
R_{13}	Inc	Fix	Fix
R_{23}	Fix	Inc	Inc
R_{33}	Dec	Fix	Fix

Table 8.3 input domain updating in modes 1 and 2

The full updating decision procedure for the input range in *mode 1* and *mode 2* is shown in the multidimensional representation in figure 8.4.

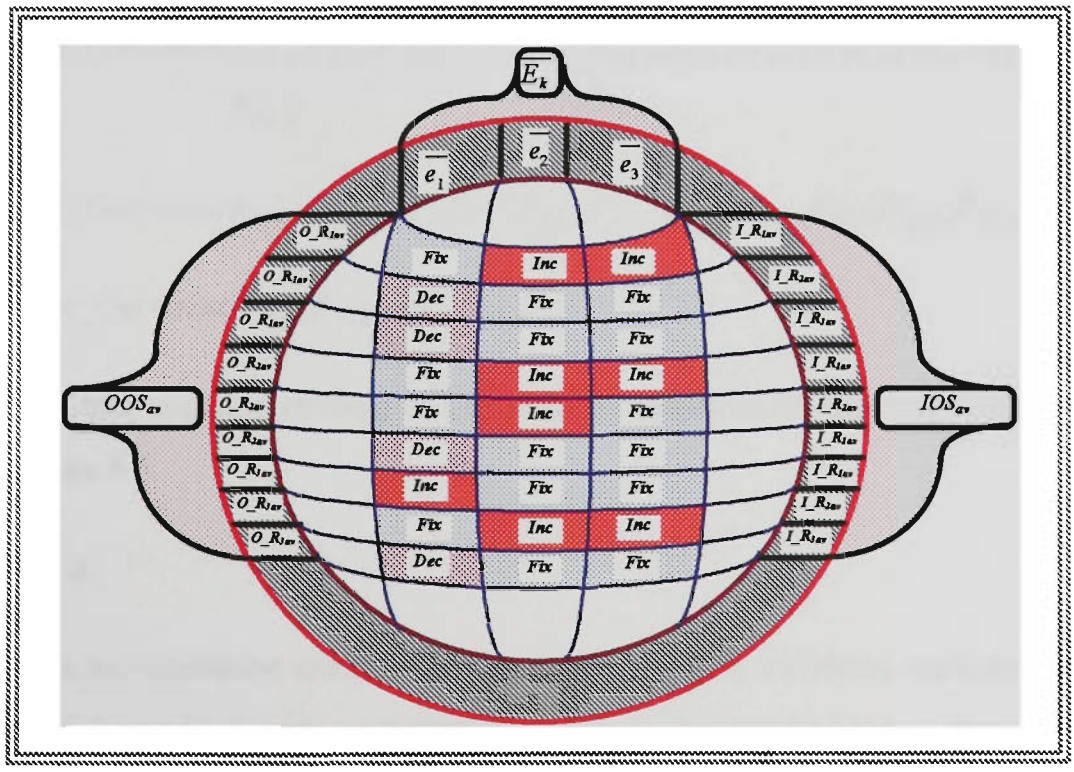


Figure 8.4 input updating in modes 1 and 2

8.4.1.2 Output domain

The output domain updating rules are shown in table 8.4 for *mode 1* and *mode 2*. These rules implement the updating strategy shown in figure 8.4.

	$\overline{e_1}$	$\overline{e_2}$	$\overline{e_3}$
R_{11}	Fix	Fix	Fix
R_{21}	Fix	Inc	Inc
R_{31}	Fix	Inc	Inc
R_{12}	Dec	Fix	Inc
R_{22}	Dec	Fix	Inc
R_{32}	Fix	Inc	Inc
R_{13}	Dec	Inc	Inc
R_{23}	Dec	Fix	Fix
R_{33}	Fix	Inc	Inc

Table 8.4 output domain updating in modes 1 and 2

Accordingly the fix-out, *Dec_Out* and *Inc_Out* are calculated as

$$Fix\_Out=max(F_{11,1},F_{21,1},F_{31,1},F_{32,1},F_{33,1},F_{11,2},F_{12,2},F_{23,2},F_{11,3},F_{12,3},F_{23,3},F_{22,2})$$

$$Inc\_Out=max(F_{21,2},F_{31,2},F_{32,2},F_{13,2},F_{33,2},F_{31,3},F_{22,3},F_{32,3},F_{13,3},F_{33,3},F_{21,3})$$

$$Dec\_Out=max(F_{12,1},F_{22,1},F_{13,1},F_{23,1})$$

The output updating strategy is shown in the multidimensional representation in figure 8.5.

8.4.2 Mode 3

This is an oscillation mode where the system is in a transition state between sectors 3 and 6. A very conservative updating strategy is used in this mode. The updating strategy is shown in figure 8.6.

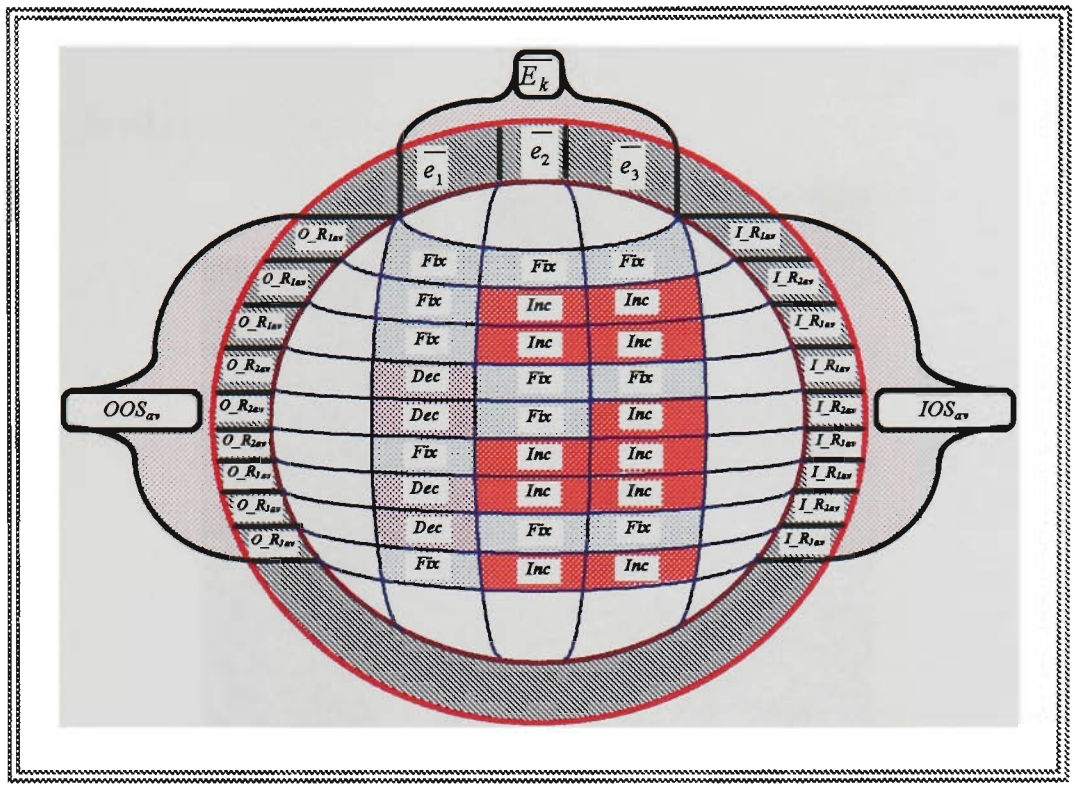


Figure 8.5 output updating in modes 1 and 2

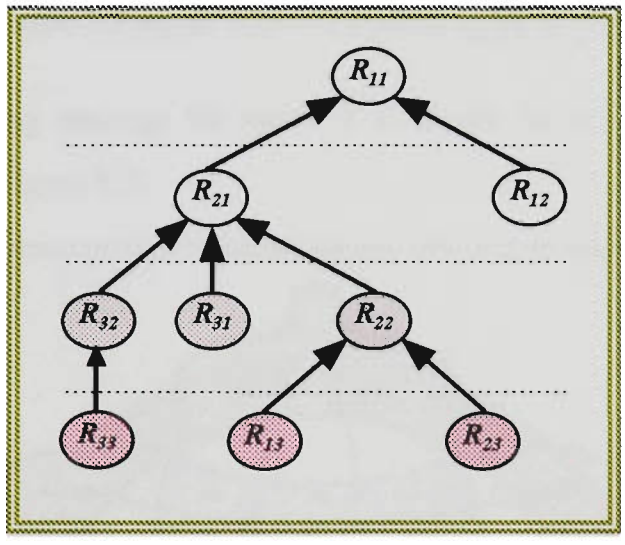


Figure 8.6 updating strategy in mode 3

8.4.2.1 Input domain

The updating strategy of *mode 3* is implemented in table 8.5 for the input domain.

The values of *Fix_In*, *Dec_In* and *Inc_In* are calculated as shown below

$$Fix\_In=max(F_{11,1},F_{12,1},F_{22,1},F_{23,1},F_{33,1},F_{11,2},F_{21,2},F_{12,2},F_{22,2},F_{13,2},F_{23,2},F_{11,3},F_{21,3},F_{12,3},F_{13,3})$$

$$Dec\_In=max(F_{21,1},F_{31,1},F_{32,1},F_{13,1},F_{31,2},F_{32,2},F_{33,2},F_{31,3},F_{22,3},F_{32,3},F_{23,3},F_{33,3})$$

$$Inc\_In=0$$

	$\overline{e_1}$	$\overline{e_2}$	$\overline{e_3}$
R_{11}	Fix	Fix	Fix
R_{21}	Dec	Fix	Fix
R_{31}	Dec	Dec	Dec
R_{12}	Fix	Fix	Fix
R_{22}	Fix	Fix	Dec
R_{32}	Dec	Dec	Dec
R_{13}	Dec	Fix	Fix
R_{23}	Fix	Fix	Dec
R_{33}	Fix	Dec	Dec

Table 8.5 input domain updating in mode 3

The input updating strategy for *mode 3* is shown in the multidimensional representation of figure 8.7.

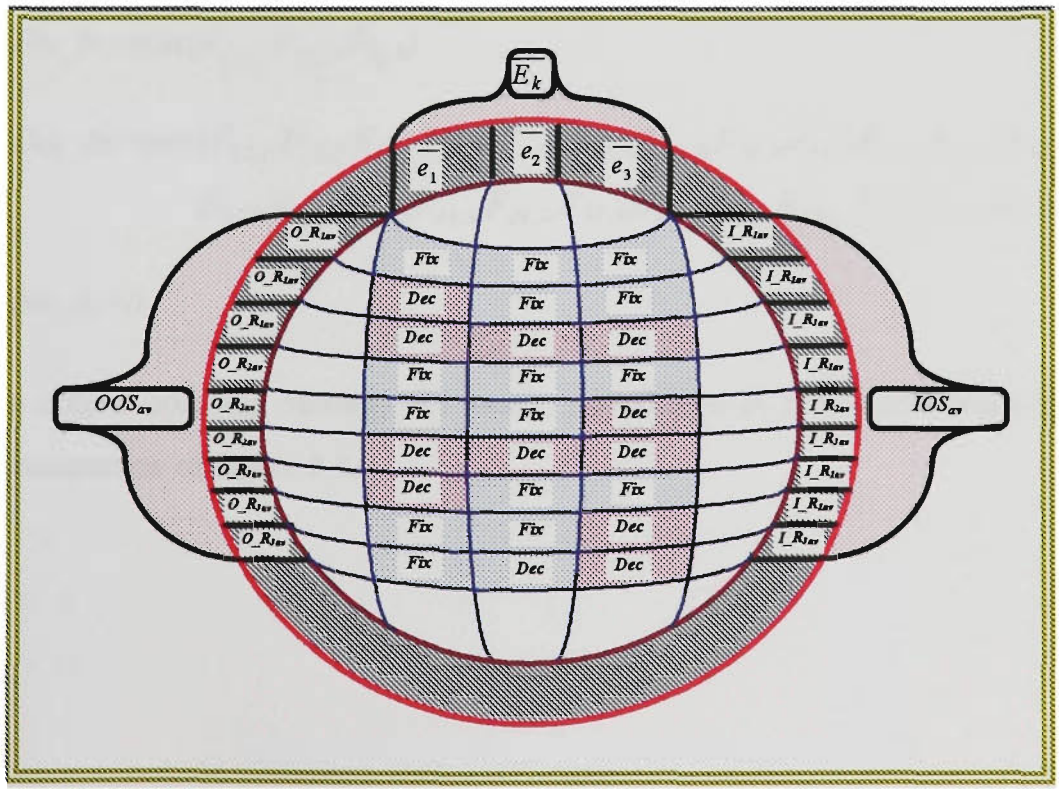


Figure 8.7 input updating in *mode 3*

8.4.2.2 Output domain

The updating strategy of *mode 3* is implemented in table 8.6 for the output domain.

	$\overline{e_1}$	$\overline{e_2}$	$\overline{e_3}$
R_{11}	Fix	Dec	Dec
R_{21}	Fix	Dec	Dec
R_{31}	Fix	Dec	Dec
R_{12}	Dec	Dec	Dec
R_{22}	Dec	Dec	Dec
R_{32}	Dec	Dec	Dec
R_{13}	Dec	Dec	Dec
R_{23}	Dec	Dec	Dec
R_{33}	Dec	Dec	Dec

Table 8.6 output domain updating in *mode 3*

The values of *Fix_In*, *Dec_In* and *Inc_In* are calculated as shown below

$$Fix\_In=max(F_{11,1},F_{21,1},F_{31,1})$$

$$Dec\_In=max(F_{12,1},F_{22,1},F_{32,1},F_{13,1},F_{23,1},F_{33,1},F_{11,2},F_{21,2},F_{31,2},F_{12,2},F_{22,2},F_{32,2},F_{13,2},F_{23,2},F_{33,2},F_{11,3},F_{21,3},F_{31,3},F_{12,3},F_{22,3},F_{32,3},F_{13,3},F_{23,3},F_{33,3})$$

$$Inc\_In=0$$

The output updating strategy for *mode 3* is shown in the multidimensional representation of figure 8.8.

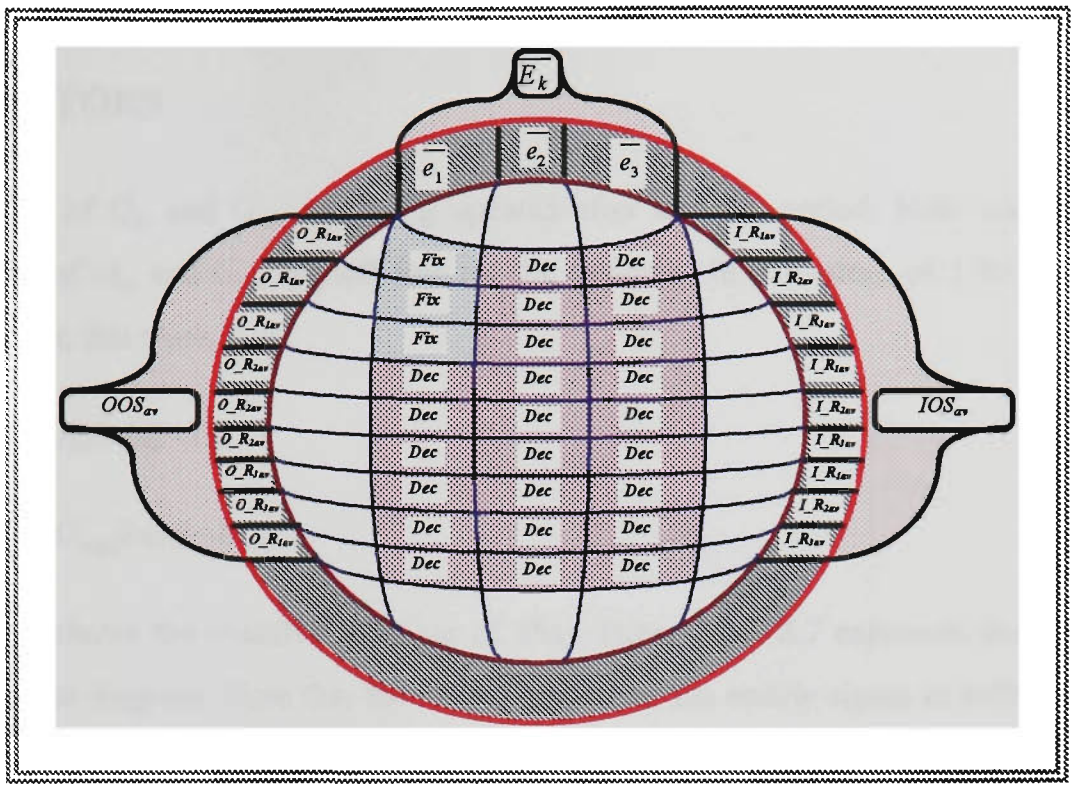


Figure 8.8 output updating in *mode 3*

8.5 CALCULATION OF INPUT AND OUTPUT TUNING SIGNALS

The *Fix*, *Inc* and *Dec* values derived in the previous sections are used to calculate U_{in} and U_{out} as:

$$U_{in}=(1-Fix\_In)\times(Inc\_In-Dec\_In)$$

and the output updating signal U_{out} is calculated as

$$U_{out}=(1-Fix\_Out)\times(Inc\_Out-Dec\_Out)$$

Note that the procedure used to derive the updating signal ensures that *Fix*, *Dec* and *Inc* range from 0 to 1 and that $Fix+Inc+Dec=1$ and $Fix=1-(Dec+Inc)$ for inputs and output domains in all modes.

8.6 UPDATING INPUT AND OUTPUT SCALING FACTORS

The values of G_{in} and G_{out} are being updated after every L period. Note that the initial conditions of G_{in} and G_{out} are left free for the designer. Initial values of 1 for both gains were used in this work.

$$G_{in}(k)=G_{in}(k-M)+U_{in}$$

$$G_{out}(k)=G_{out}(k-M)+U_{out}$$

Figure 8.9 shows the overall operations of Shay-Tune. Table 8.7 expresses the operations shown in the diagram. Note that the modes operate as the enable signal to switch between the two parallel levels in figure 8.9.

	Response	
Block	*High L (set)*	*Low L (reset)*
1	*0*	*md1,md2,md3*
2	*0*	$R_{xy}s$
3	*0*	$\overline{e_i}\ s$
4	*Fix, Dec, Inc*	*0*
5	*Fix, Dec, Inc*	*0*
6	U_{in}	*0*
7	U_{out}	*0*
8	U_{in}	*0*
9	U_{out}	*0*
10	$G_{in}+U_{in}$	G_{in}
11	$G_{out}+U_{out}$	G_{out}

Table 8.7 timing and sequence in figure 8.9

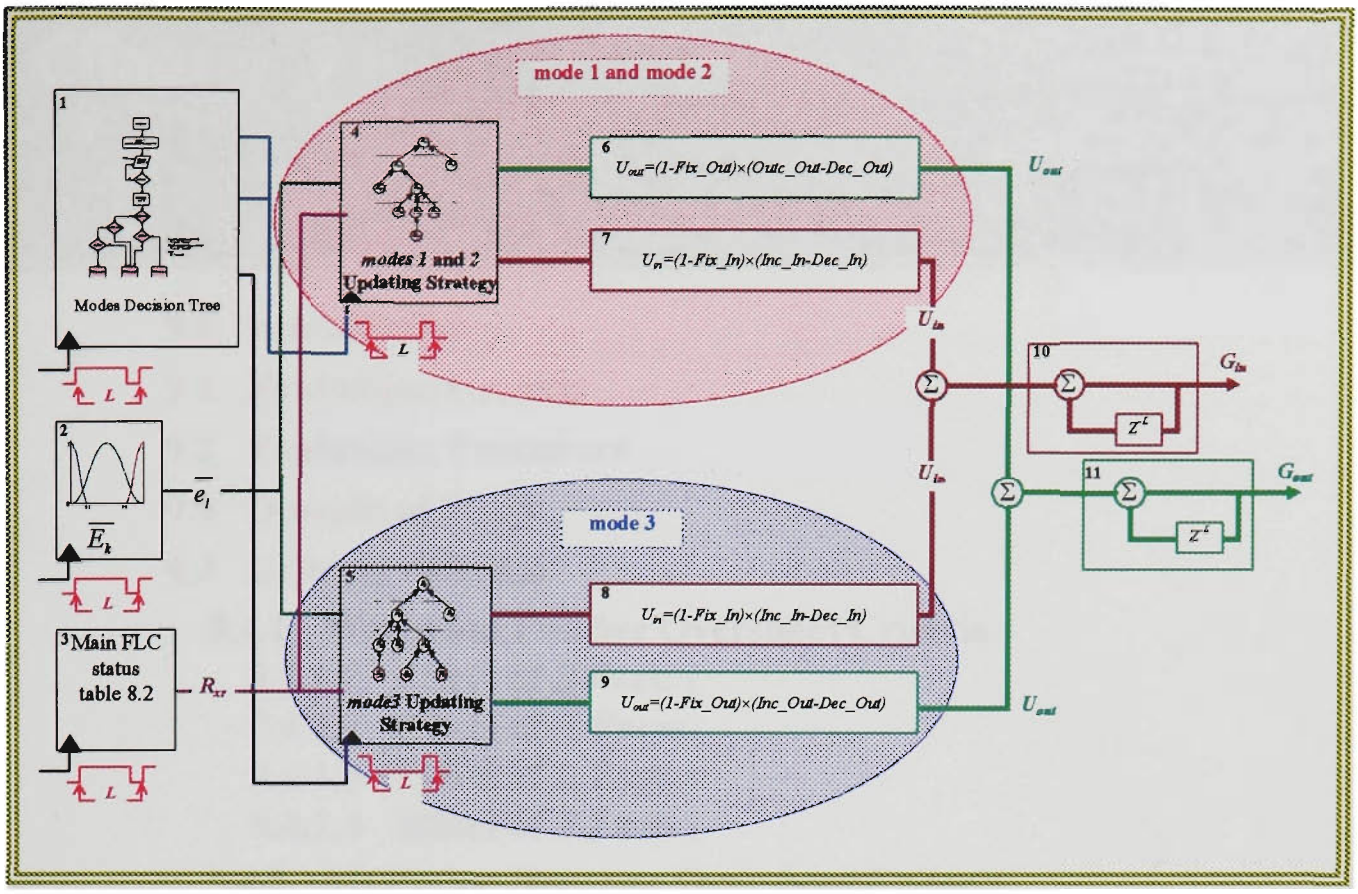


Figure 8.9 overall Shay-PA operations

Chapter 9 *Analysis of the Shay Parameters Influence on the Main Indicators Performance*

9.0 INTRODUCTION

This chapter considers the performance and design of a Shay-FLC. The structure mentioned in the previous chapters of this section is equipped with very powerful adaptation tools. These tools can gradually alter the controller parameters, on-line, towards better performance. Although the structure of Shay seems to be complicated and involves many procedures, its use and implementation are very simple. The designer is not to be concerned about the internal components of Shay. The only parameters in Shay that might be altered to suit a particular system are the four factors:

1) *S*, the sectors scale,

2) *A*, the operational angle scale,

3) *L*, the learning period,

4) *T*, the learning threshold.

These factors are referred to as the *SALT* parameters for simplicity throughout the chapter. The introduction of the *SALT* parameters should not, in any way, lead to understanding that the Shay-FLC requires some off-line optimisation prior to any implementation. The four factors were developed based on the learning algorithm used in Shay. They are provided here in order to give the designer the advantage of having the best start for his controller and to reduce the required on-line tuning by the algorithm.

This chapter analyses the independent influence of each of the *SALT* parameters on the system performance under five of the standard controllers evaluation criteria. The chapter also investigates the effect of the *SALT* parameters on each other and provides some guidelines in utilising them.

9.1 EVALUATION CRITERIA

The effects of the *SALT* parameters on a Shay-FLC performance have been investigated. A heuristic approach was taken in an attempt to develop general patterns of how each factor

effects the system behaviour with respect to some standard control criteria. These criteria are:

1) the maximum positive overshoot, μ_{ρ},

2) the minimum negative overshoot, μ_{η},

3) the rise time, t_r,

4) the absolute steady state error, $|e_s|_{av}$,

5) the integral of the error, e_i.

These measures are shown in figure 9.1 which represents a standard unit step response.

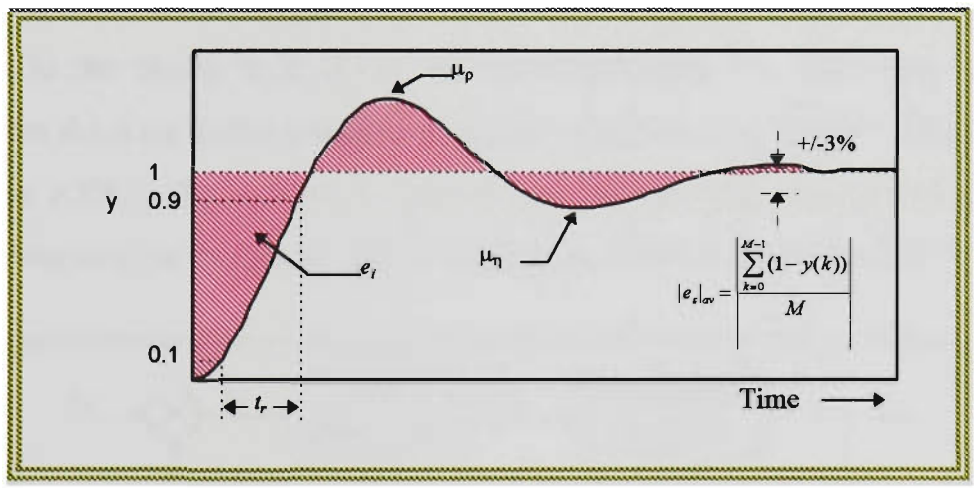


Figure 9.1 ***SALT*** **parameters evaluation criteria**

9.2 EVALUATION PROCEDURE

The analysis and evaluation of the *SALT* parameters effect on the Shay-FLC was performed via computer simulations. The domain of application was the voltage control of a synchronous generator connected to an infinite bus. A linearized third order model of the synchronous generator was used [178]. The generator parameters are shown in section 3 chapter 12. Figure 9.2 shows the block diagram of the generator model.

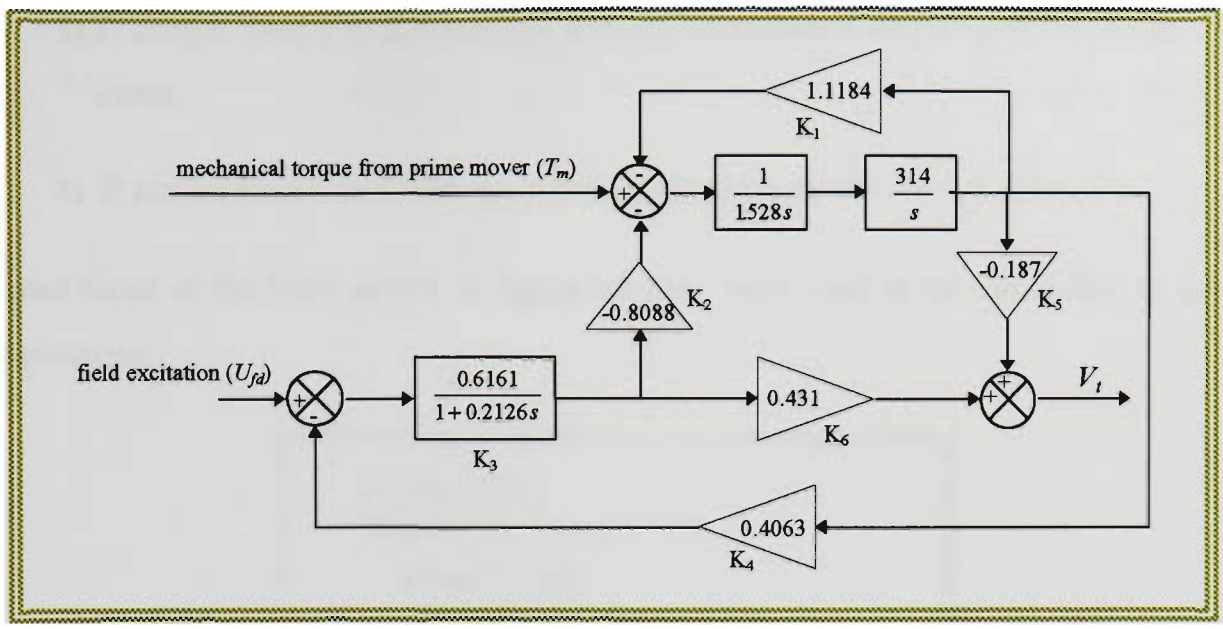


Figure 9.2 linear third order model of a synchronous generator

Shay-FLC has been designed with the same parameters mentioned in section 2, and used as an AVR in the closed loop control as shown in figure 9.3. The input to the Shay controller was the error in the generator terminal voltage (e_{vt}) as shown in the figure. The output of the AVR is the excitation signal (U_{fd}). The unit step response of the terminal voltage (v_t) was used to analyse the *SALT* parameters effect on the Shay-FLC AVR.

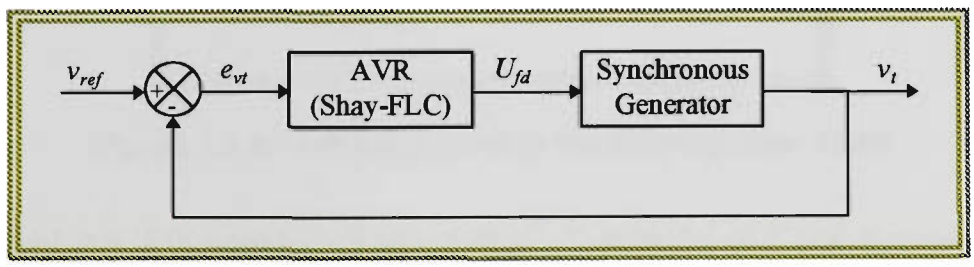


Figure 9.3 closed loop control used in the investigation study

9.3 DOMAIN OF STUDY

The simulations were performed for all the possible combinations of the *SALT* parameters according to the following:

1) *S*: ranges from 0 to 2 with the incremental step size of 0.1, 20 cases,

2) *A*: ranges from 0 to 2 with the incremental step size of 0.1, 20 cases,

3) L: ranges from 0 to 2.5 seconds with the incremental step size of 0.125, 20 cases,

4) T: ranges from 0 to 1 with the incremental step size of 0.05, 20 cases.

Nested loops of the form shown in figure 9.4 have been used in the simulation to cover these ranges.

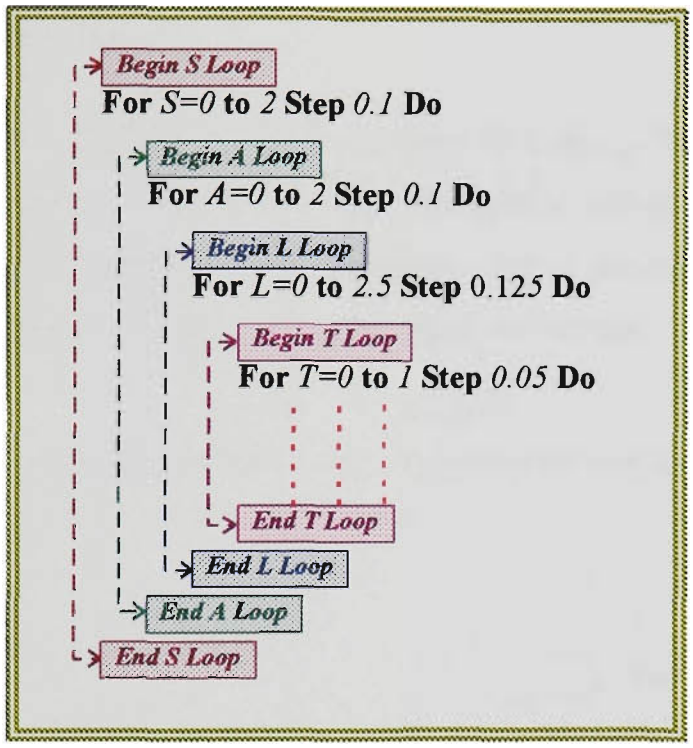


Figure 9.4 nested loops used in the investigation study

Thus, a total of 160,000 cases have been studied[1]. The ranges of S and A were chosen such that they cover all conditions of greater, equal and less than in the amplitude difference between E_k and E_{k-1} (the two successive samples of the performance monitor). The range of L was chosen according to the synchronous generator time constants used. The transient open loop time constant (τ_{do}`) of the generator was 0.34 seconds and the simulation sampling time was 25 milliseconds. The number of samples collected during the learning period ranges from 0 to 100. T values were chosen to cover all the ranges of the possibility of an occurrence of each sector during the learning period L, 0% to 100%.

[1] Each case requires 56 seconds of simulation time using ALPHA Station $200^{4/233}$. The simulation required 2489 hours in total, 17 work stations were used each for 146.4 hours, and the simulations were running in parallel for 6.1 days (excluding system crashes).

The simulation results showed some great dependency of the influence of L and T on each other. While both S and A influences can be seen as independent from each other and from L and T.

9.4 CRITERION PATTERNS

9.4.1 Maximum Positive Overshoot Criteria

9.4.1.1 Effect of S factor

Figure 9.5 shows the effect of different values of S on μ_ρ. The figure clearly shows that increasing S will increase μ_ρ. Although the pattern shown in figure 9.5 has some local minimums and maximums, but a general approximation yields an average positive slope. Thus one can conclude that;

$$\mu_\rho \propto S \qquad \forall\, S \in [0,2]$$

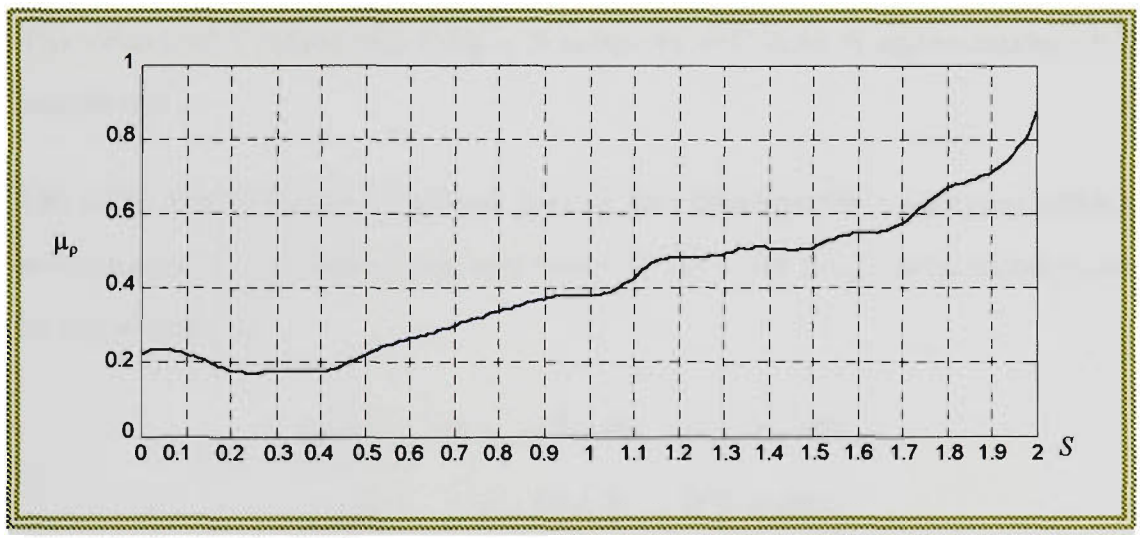


Figure 9.5 μ_ρ pattern with respect to S

9.4.1.2 Effect of A factor

The effect of A on μ_ρ is shown in figure 9.6. A general decaying pattern can be noticed in the μ_ρ curve with an increase in A. It is also clear from the pattern that the decaying slope is larger in the region where A is less than 1. Therefore, the relationship between μ_ρ and A can be expressed as:

$$\mu_\rho \propto \frac{1}{A} \qquad \forall\, A \in [0,2]$$

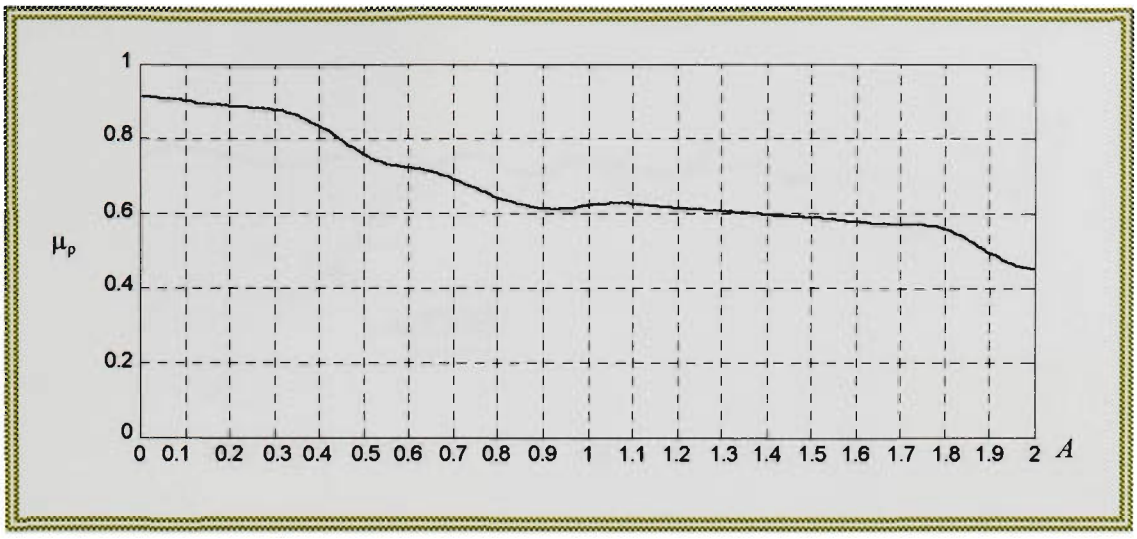


Figure 9.6 μ_ρ pattern with respect to *A*

9.4.1.3 Effect of *L* factor

L effect on μ_ρ is largely influenced by *T*. Figure 9.7 shows that μ_ρ change has a positive slope with the increase of *L* providing that *T* is large (the dashed line). The values of *T* where this *L*-μ_ρ characteristic will hold is approximately 0.8 and above.

The solid line in figure 9.7 shows that μ_ρ will demonstrate a decaying pattern with increasing *L* in case *T* was less than 0.8. Thus the *L*-μ_ρ characteristics can be represented as

$$\mu_\rho \propto L \qquad \forall L \in [0,2.5] \qquad \textit{If T is large}$$

$$\mu_\rho \propto \frac{1}{L} \qquad \forall L \in [0,2.5] \qquad \textit{If T is small}$$

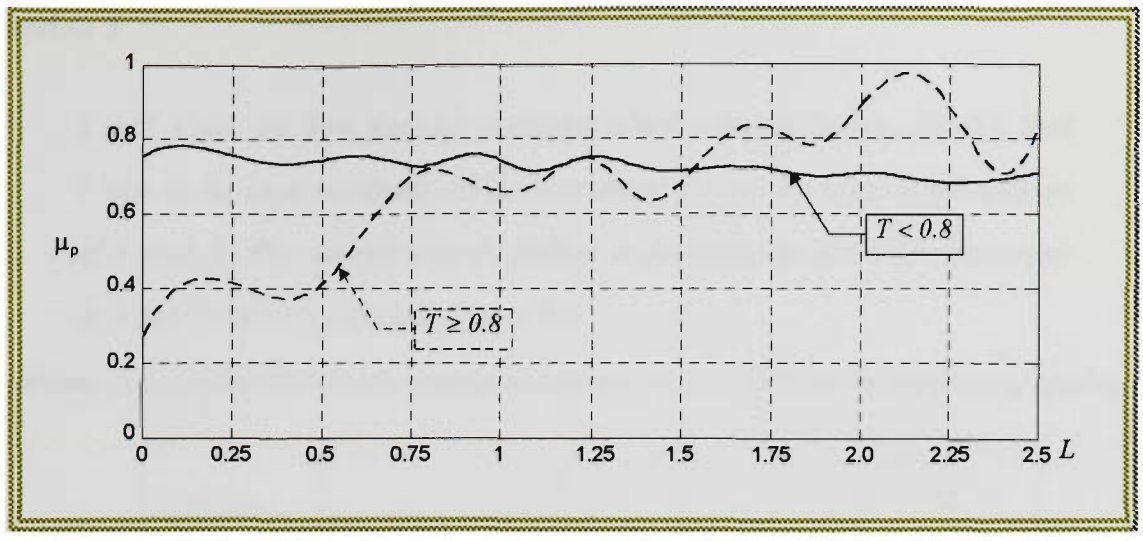


Figure 9.7 μ_ρ pattern with respect to L

9.4.1.4 Effect of T factor

Figure 9.8 shows that the effect of T on μ_ρ results in a decaying pattern as long as T is increasing. However, the pattern shows that smaller values of T (T<0.5), will have minor effect on μ_ρ. While larger values of T will result in more negative slope. Thus the T-μ_ρ characteristics can be expressed as follows

$$\mu_\rho \cong constant \qquad \forall\ T \in [0, 0.5]$$

$$\mu_\rho \propto \frac{1}{T} \qquad \forall\ T \in [0.5, 1]$$

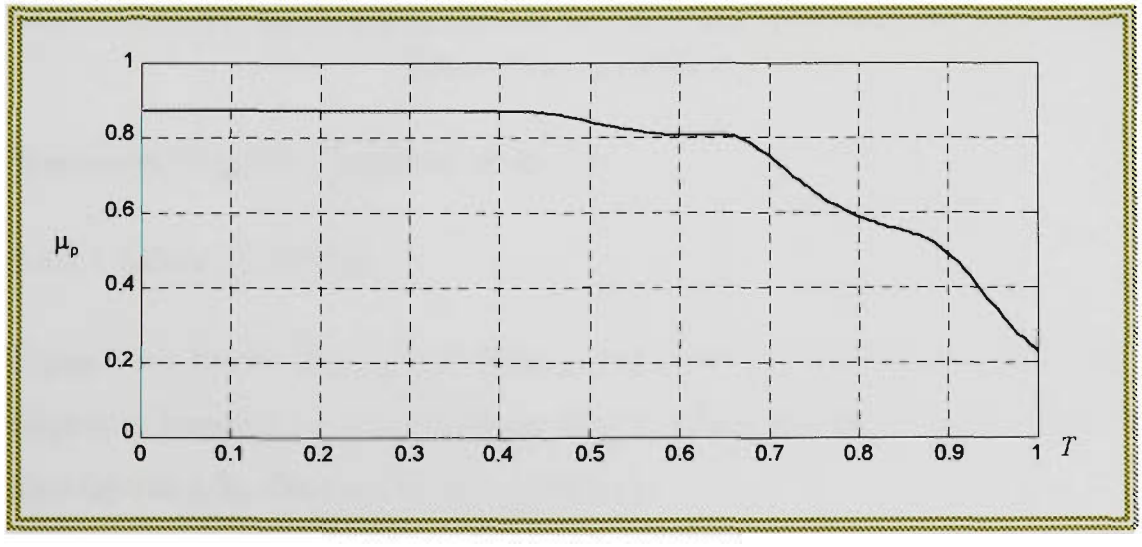


Figure 9.8 μ_ρ pattern with respect to T

Conclusion 1

S and A are the best factors to adjust when considering μ_p. Both L and T are to be kept constant while searching for an optimal combination of S and A. The search should follow a decrease in A and an increase in S for better results, see figure 9.9.

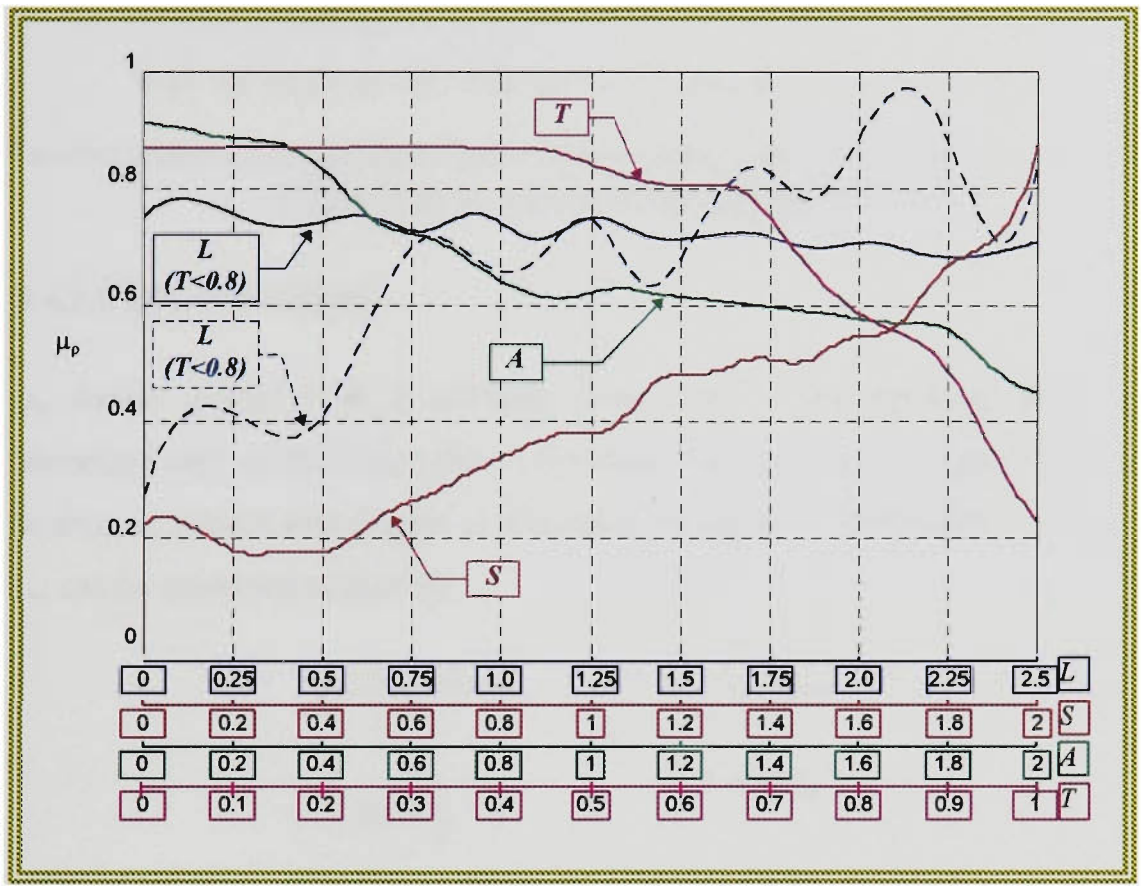


Figure 9.9 μ_p patterns

9.4.2 Maximum Negative Overshoot Criteria

9.4.2.1 Effect of *S* factor

Figure 9.10 shows that μ_η will have a step increase when *S* is close to one (higher or lower) μ_η is approximately constant when *S* is small (below 0.85) or high (above 1.2). Thus μ_η can be expressed as:

$$\mu_\eta \cong constant \qquad \forall\, S \in [0,0.85] \cup [1.2,2]$$

$$\mu_\eta \propto S \qquad \forall\, S \in]0.85,1.2[$$

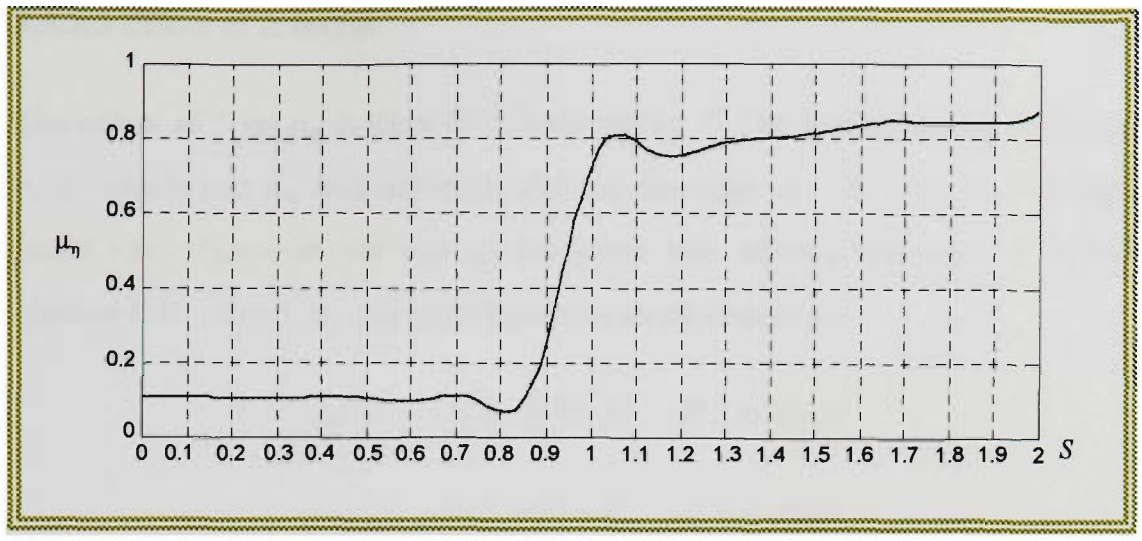


Figure 9.10 μ_η pattern with respect to *S*

9.4.2.2 Effect of *A* factor

μ_η decays gradually as A increases from 0 to 1. The decaying slope is becoming very small at the values of A higher than 1, it can be approximated to zero. μ_η pattern with respect to A is shown in figure 9.11. The effect of A on μ_η can be expressed as follows:

$$\mu_\eta \cong constant \qquad \forall A \in [0, 0.5]$$

$$\mu_\eta \propto \frac{1}{A} \qquad \forall A \in [0, 1]$$

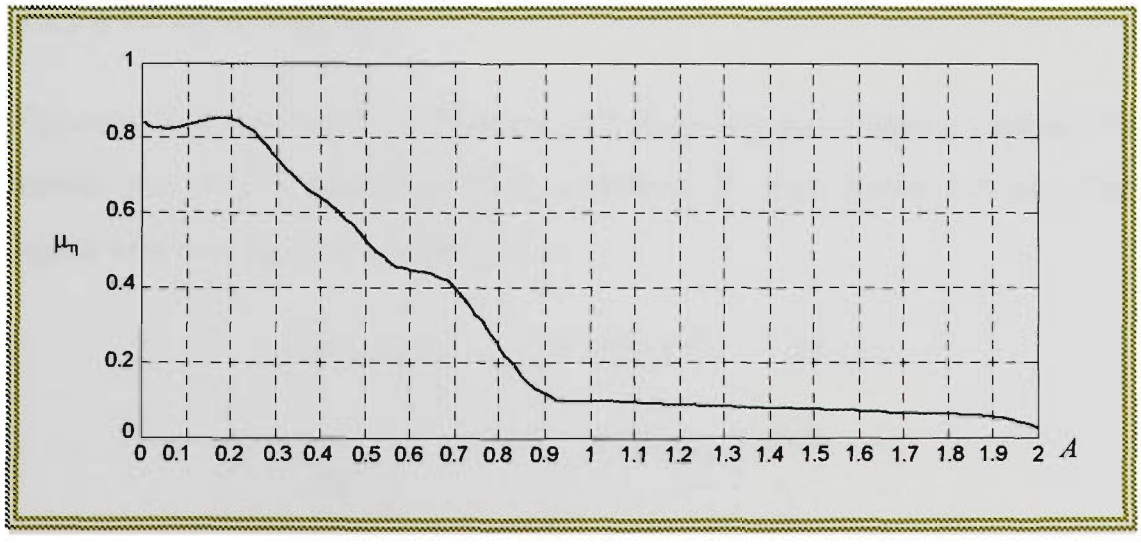


Figure 9.11 μ_η pattern with respect to *A*

9.4.2.3 Effect of L factor

The effect of L on μ_η is directly influenced by T. The pattern shown in figure 9.12 reveals that μ_η will increase with the increase of L as long as T is high (solid line). However the sign of the slope will be reversed when T is low (dashed line). The L-μ_η characteristics can be expressed as:

$$\mu_\eta \propto L \qquad \forall\, L \in [0,2.5] \qquad \textit{If } T \textit{ is large}$$

$$\mu_\eta \propto \frac{1}{L} \qquad \forall\, L \in [0,2.5] \qquad \textit{If } T \textit{ is small}$$

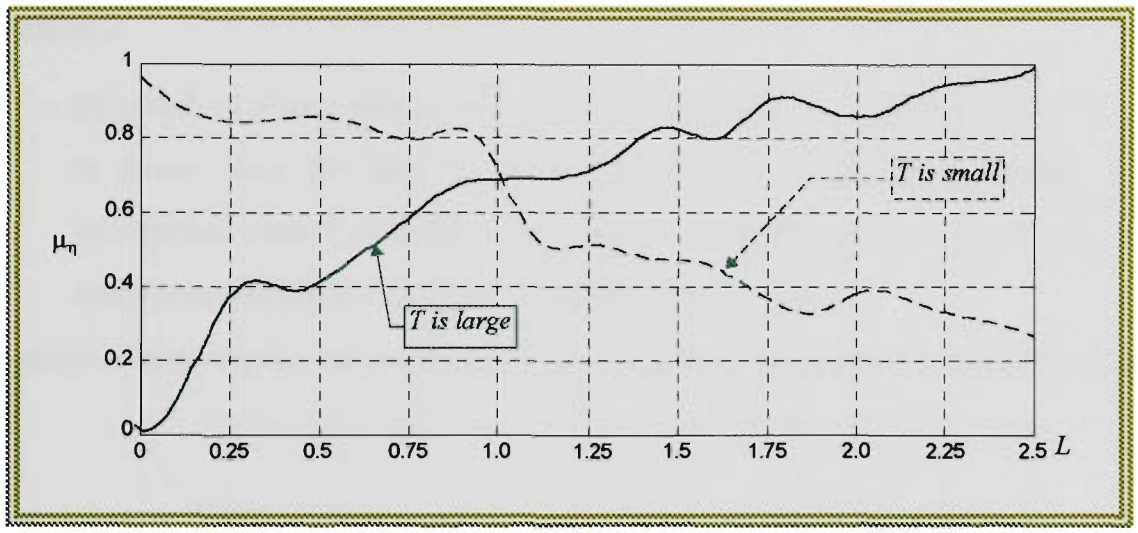


Figure 9.12 μ_η pattern with respect to L

9.4.2.4 Effect of T factor

Figure 9.13 shows that μ_η will have a very small negative slope as long as T is smaller than 0.35 or larger than 0.68, while having a large negative slope in the region between. μ_η can be expressed as:

$$\mu_\eta \cong constant \qquad \forall\, T \in [0,0.35[\, \cup\,]0.68,1]$$

$$\mu_\eta \propto \frac{1}{T} \qquad \forall\, T \in [0.35,0.68]$$

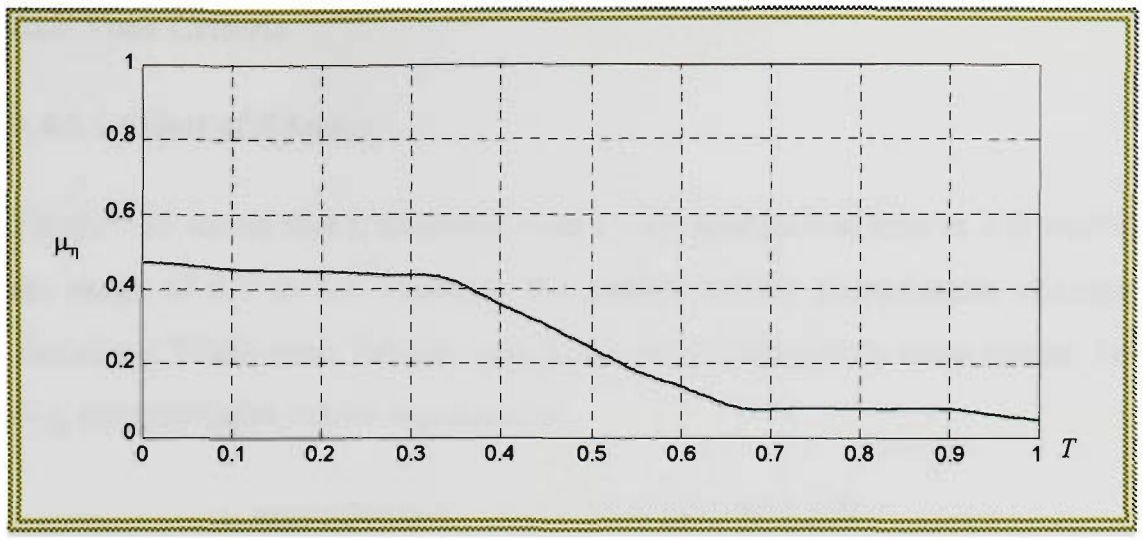


Figure 9.13 μ_η pattern with respect to *T*

Conclusion 2

Figure 9.14 shows that μ_n is robust to any changes in S and A while S is lower than 0.8 and A larger than 1. Thus the optimisation mechanism should consider decreasing L while fixing S, A and T at values around 0.8 for S, 1 for A and less than 0.5 for T.

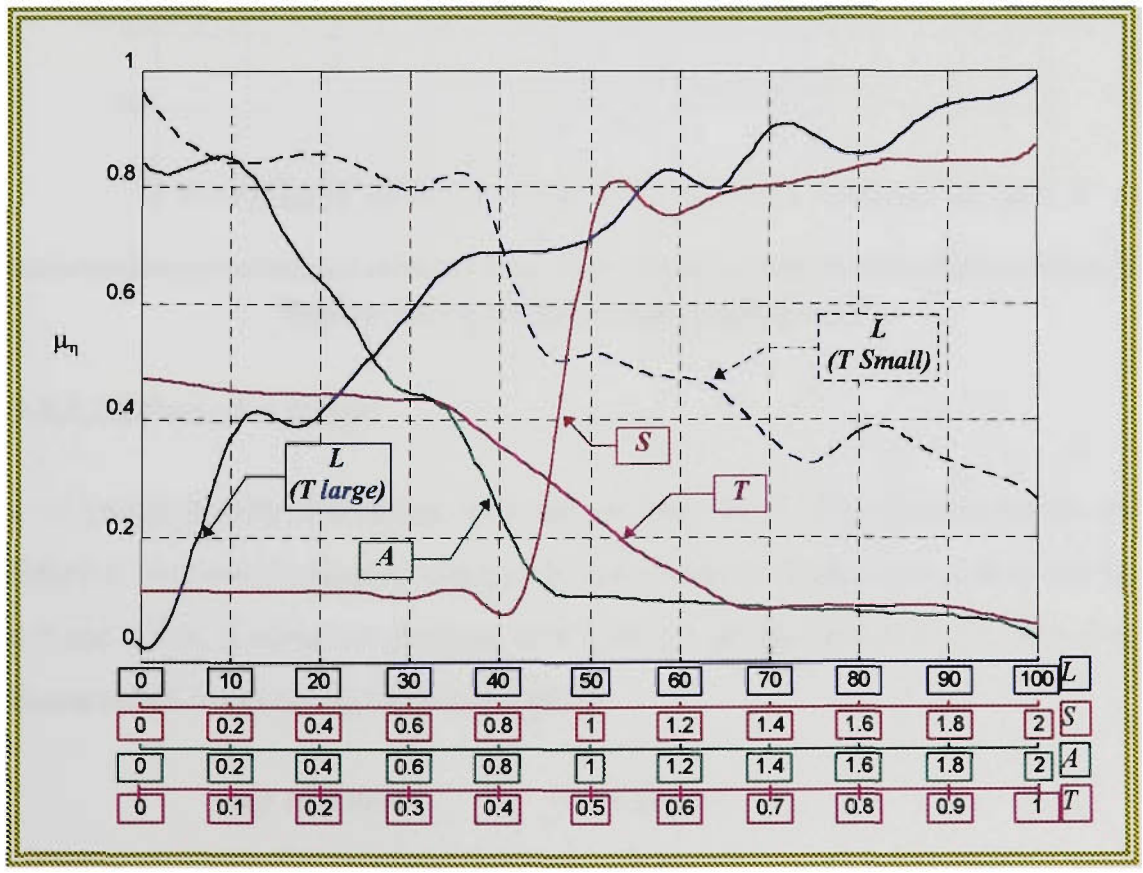


Figure 9.14 μ_η patterns

9.4.3 Rise Time Criteria

9.4.3.1 Effect of *S* factor

Figure 9.15 shows that t_r decreases with an increase in S as long as S is within the range of 0.5 to 1.1. However the pattern suffers unpredictable changes elsewhere. While other factors seem to be more dominant in these ranges. So S- t_r characteristics can be expressed as:

$$\mu_\eta \; unpredictable \qquad \forall S \in [0,0.5[\cup]0.1.1,2]$$

$$\mu_\eta \propto \frac{1}{S} \qquad \forall S \in [0.5,1.1]$$

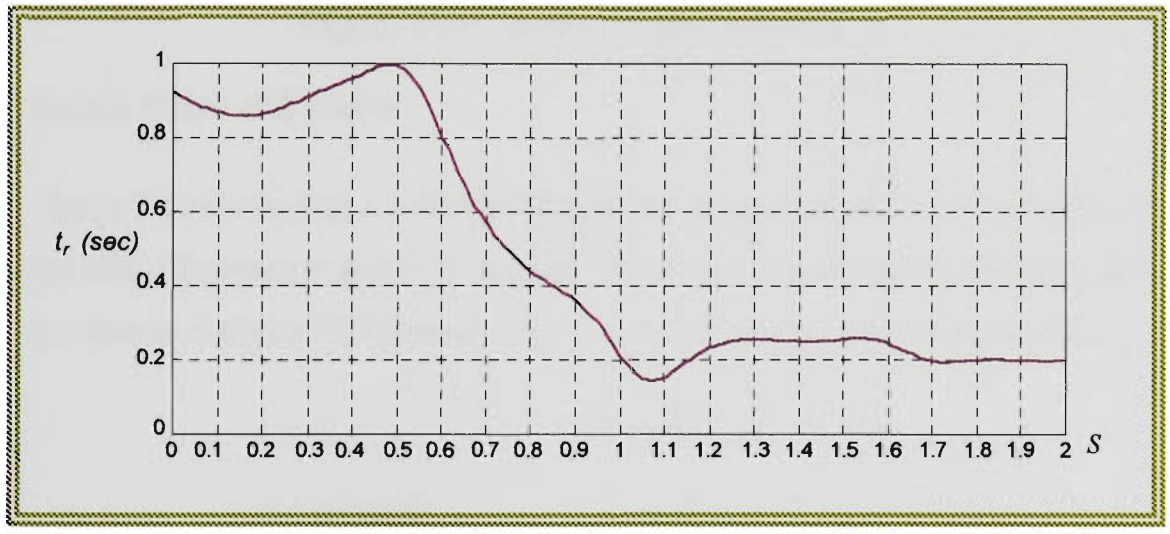


Figure 9.15 t_r pattern with respect to *S*

9.4.3.2 Effect of *A* factor

t_r is proportionally increasing with an increase of A. The pattern shown in figure 9.16 shows some step changes in the climbing slope when A is closer to 0.9 and 1.9. t_r is robust to changes in A when A ranges from 0 to 0.7. The A-t_r characteristics can be expressed as follows:

$$tr \cong constant \qquad \forall A \in [0,0.7[$$

$$t_r \propto a_1 A \qquad \forall A \in [0.7,1[$$

$$t_r \propto a_2 A \qquad \forall A \in [1,1.65[$$

$$t_r \propto a_3 A \qquad \forall A \in [1.65, 2]$$

where $a_3 > a_1 > a_2$

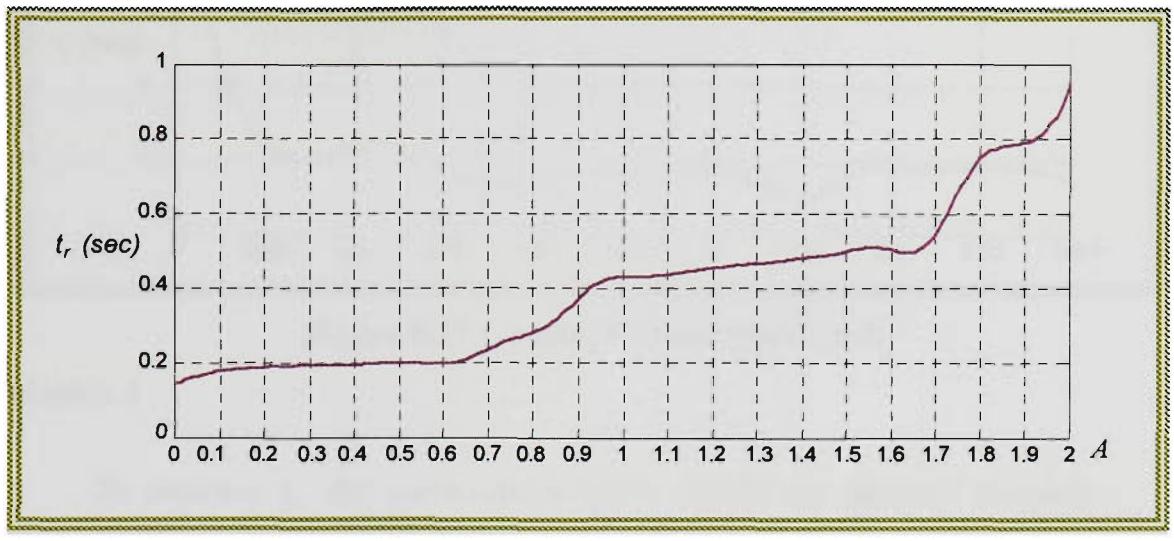


Figure 9.16 t_r pattern with respect to A

9.4.3.3 Effect of L factor

Figure 9.17 shows that t_r will significantly be reduced when L is very small, in the range between 0 and 0.25 seconds. However, t_r tends to be robust to any increase in L above 0.25 seconds. The L-t_r characteristics can be expressed as:

$$t_r \propto L \qquad \forall L \in [0, 0.25]$$

$$tr \cong constant \qquad \forall L > 0.25$$

9.4.3.4 Effect of T factor

No clear pattern was obtained for the effect of T on t_r, as other factors seemed to be more dominant.

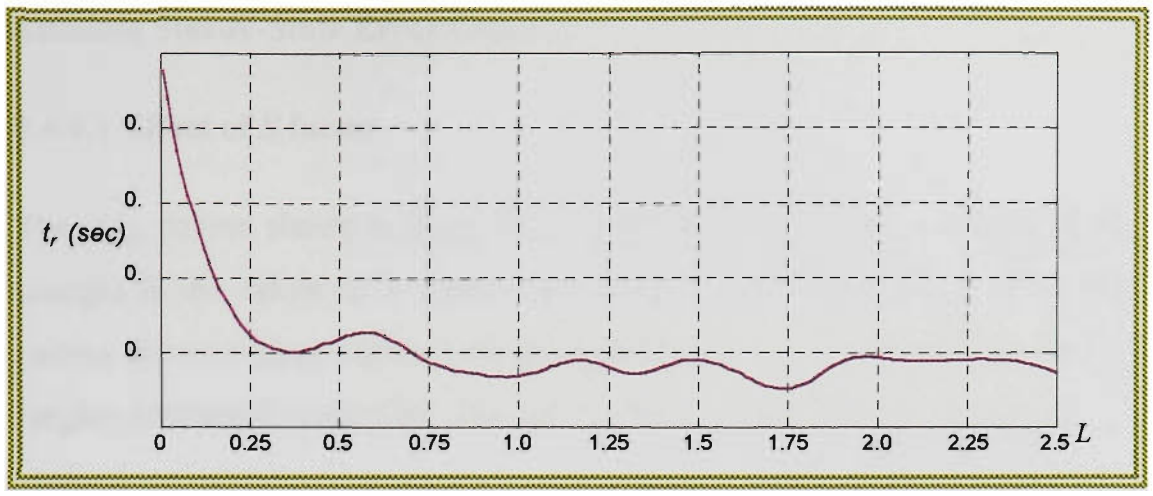


Figure 9.17 t_r pattern with respect to L

Conclusion 3

To improve t_r, the optimisation effort should be directed towards reducing S and L while keeping A at values less than 0.5. The patterns shown in Figure 9.18 show that T does not have a clear pattern against t_r.

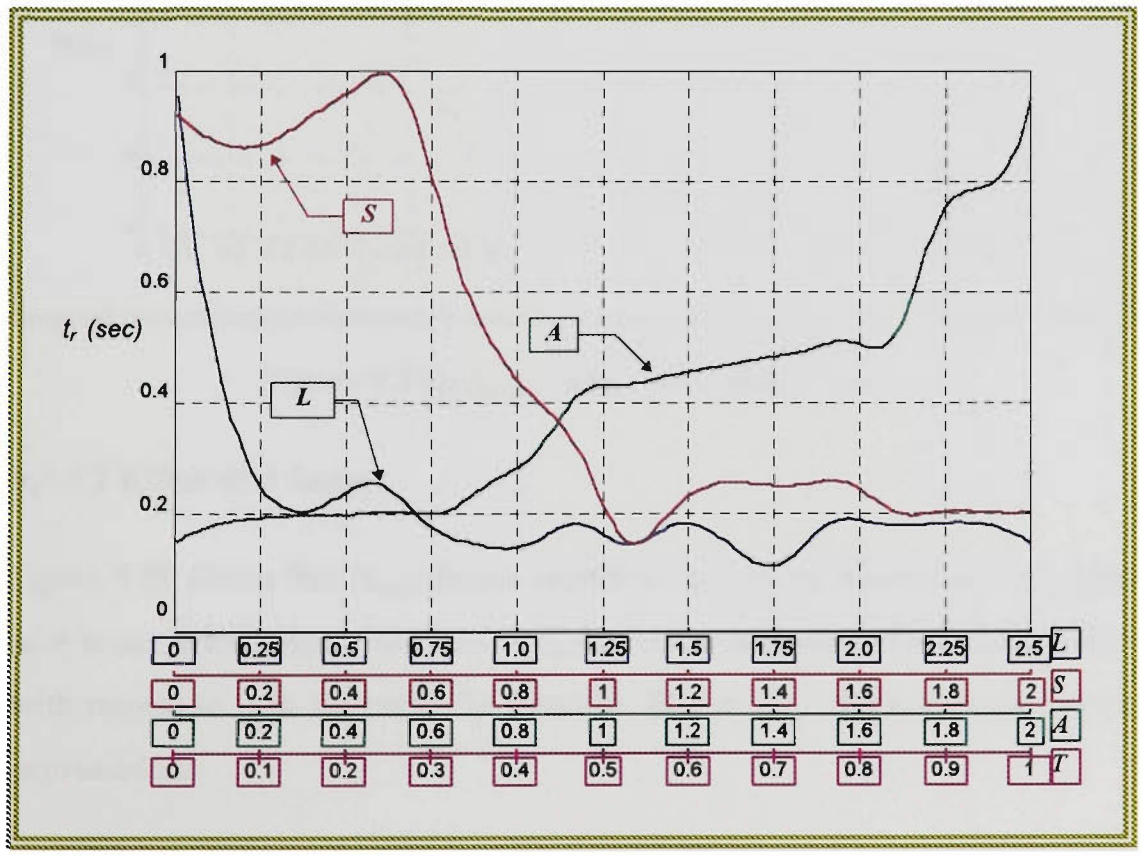


Figure 9.18 t_r patterns

9.4.4 Absolute Steady-State Error Criteria

9.4.4.1 Effect of S factor

The $|e_s|_{av}$ pattern shown in figure 9.19 demonstrates that $|e_s|_{av}$ is robust to any changes in the values of S, when S is less than 0.5 or greater than 0.85. The pattern shows a large negative decaying slope in $|e_s|_{av}$ in the area where S is ranging between 0.5 and 0.85. The S-$|e_s|_{av}$ characteristics are expressed as:

$$|e_s|_{av} \cong constant \qquad \forall\, S \in [0, 0.5[\cup [0.85, 2]$$

$$|e_s|_{av} \propto \frac{1}{S} \qquad \forall\, S \in [0.5, 0.85[$$

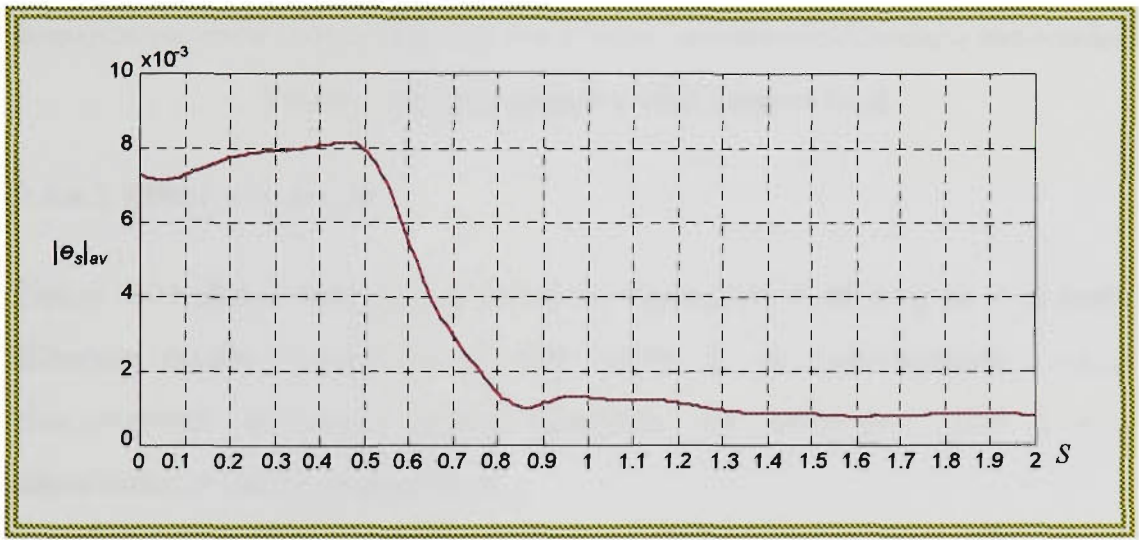


Figure 9.19 $|e_s|_{av}$ pattern with respect to S

9.4.4.2 Effect of A factor

Figure 9.20 shows that $|e_s|_{av}$ decays proportionally to an increase in A as long as A is not in the range of 0.6 to 1. $|e_s|_{av}$ suffers an unpredictable characteristics with respect to A in the region in between. The A-$|e_s|_{av}$ characteristics can be expressed as:

$$|e_s|_{av} \propto -a_1 \frac{1}{A} \qquad \forall\, A \in [0, 0.6[$$

$$|e_s|_{av} \propto a_2 \frac{1}{A} \qquad \forall\, A \in [0.6, 1[$$

$$|e_s|_{av} \propto -a_3 \frac{1}{A} \qquad \forall A \in]1,2]$$

Where, $|a_1| > |a_3|$ *and* a_2 *is dependent on other parameters*

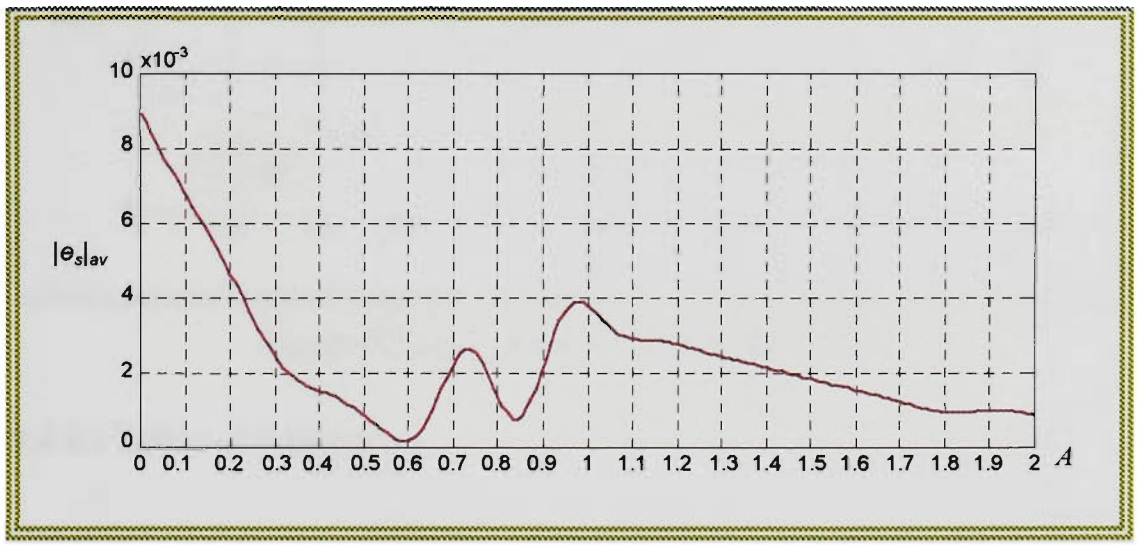


Figure 9.20 $|e_s|_{av}$ **pattern with respect to** A

9.4.4.3 Effect of L factor

Figure 9.21 shows that $|e_s|_{av}$ is robust to changes in L as long as T is high. Whereas smaller values of T will result in an unpredictable L-$|e_s|_{av}$ characteristics as other factors dominate the behaviour. The L-$|e_s|_{av}$ characteristics can be expressed as:

$$|e_s|_{av} \cong constant \qquad \forall L \in [0,2] \qquad if\ T\ is\ large$$

$$|e_s|_{av} \propto L_{|e_s|_{av}} L \qquad \forall L \in [0,2] \qquad if\ T\ is\ small$$

Where $L_{|e_s|_{av}}$ *is random (depends on other factors)*

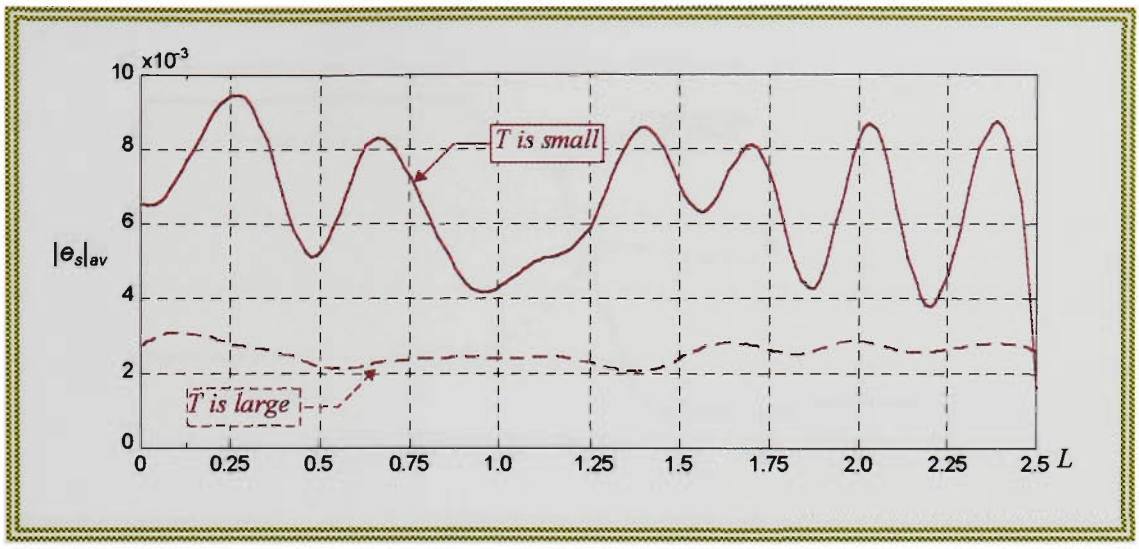


Figure 9.21 $|e_s|_{av}$ pattern with respect to L

9.4.4.4 Effect of T factor

The effect of T on $|e_s|_{av}$ is heavily influenced by L values. The patterns shown in figure 9.22 demonstrate that $|e_s|_{av}$ is robust to any variations in T while T is in the ranges of 0 to 0.4 and 0.6 to 1. This T-$|e_s|_{av}$ characteristic is valid for both cases when L is large or small. However smaller values of L will result in a positive climbing slope in the $|e_s|_{av}$ pattern with an increase in T in the region between 0.4 and 0.6. Larger values of L will reverse the sign of the slope in the same region of T. The $|e_s|_{av}$ characteristics can be expressed as:

$$|e_s|_{av} \cong constant \qquad \forall\ T \in [0,0.4] \cup [0.6,1]$$

$$|e_s|_{av} \propto T \qquad \forall\ T \in [0.4,0.6] \qquad If\ L\ is\ small$$

$$|e_s|_{av} \propto \frac{1}{T} \qquad \forall\ T \in [0.4,0.6] \qquad If\ L\ is\ large$$

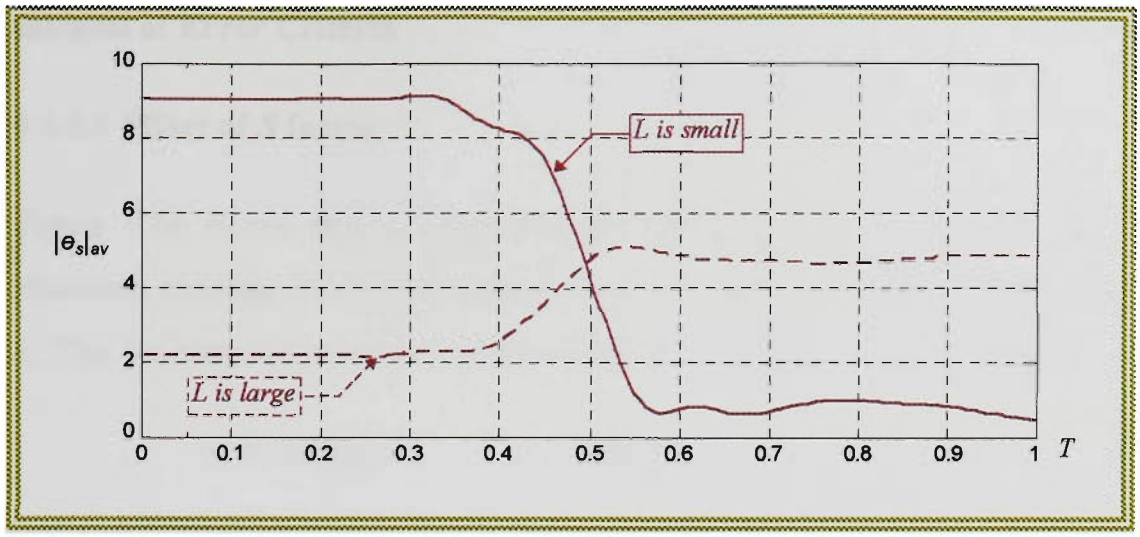


Figure 9.22 $|e_s|_{av}$ pattern with respect to T

Conclusion 4

When trying to optimise the SALT parameters in the search for the best $|e_s|_{av}$, it is recommended to fix T between 0.6 and 0.7, fix S at values larger than 1.2, fix L and search for an optimal A between 1.2 and 1.8. See figure 9.23 for $|e_s|_{av}$ patterns.

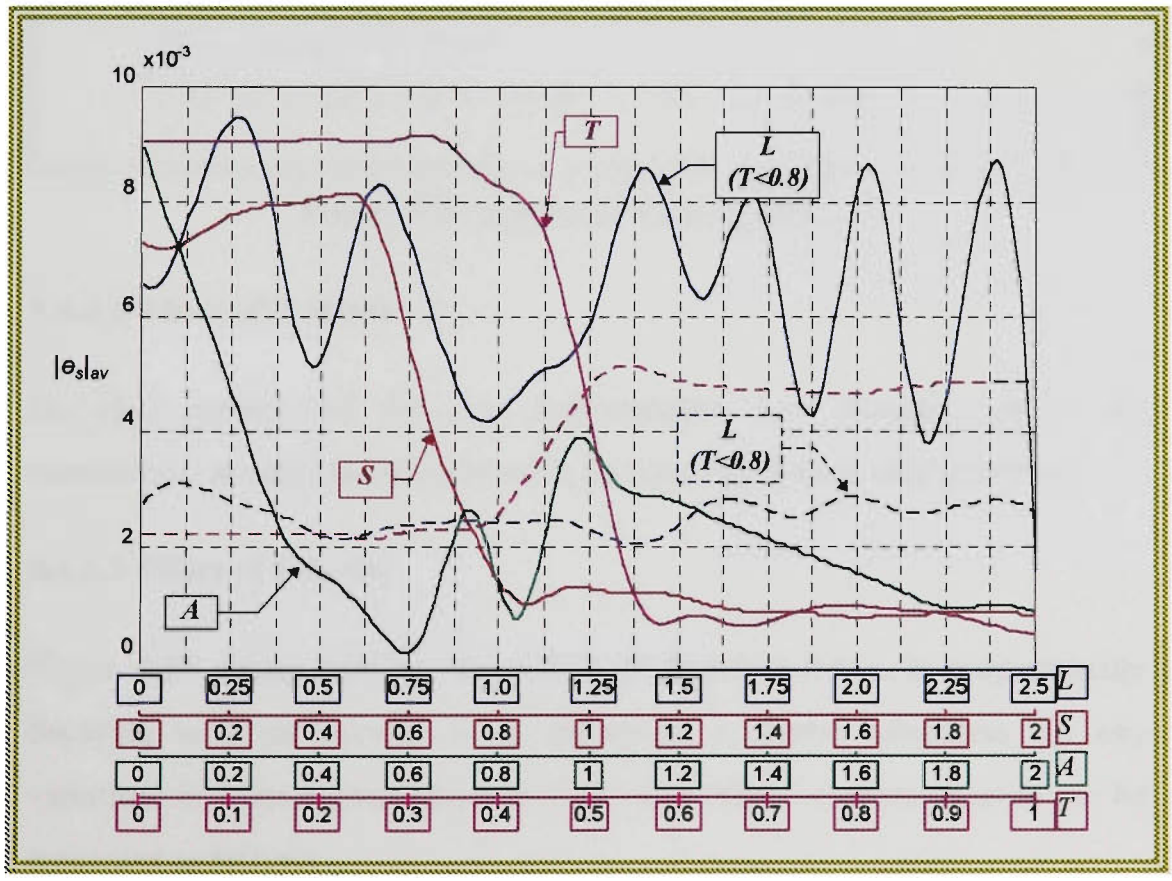


Figure 9.23 $|e_s|_{av}$ patterns

9.4.5 Integral of Error Criteria

9.4.5.1 Effect of *S* factor

Figure 9.24 shows that e_i proportionally increases with an increase in S. However, changes in S in the range from 0 to 1 seem to have minor effect on e_i. The S-e_i characteristics can be expressed as:

$$e_i \cong constant \qquad \forall\, S \in [0,1]$$

$$e_i \propto S \qquad \forall\, S \in [1,1.5]$$

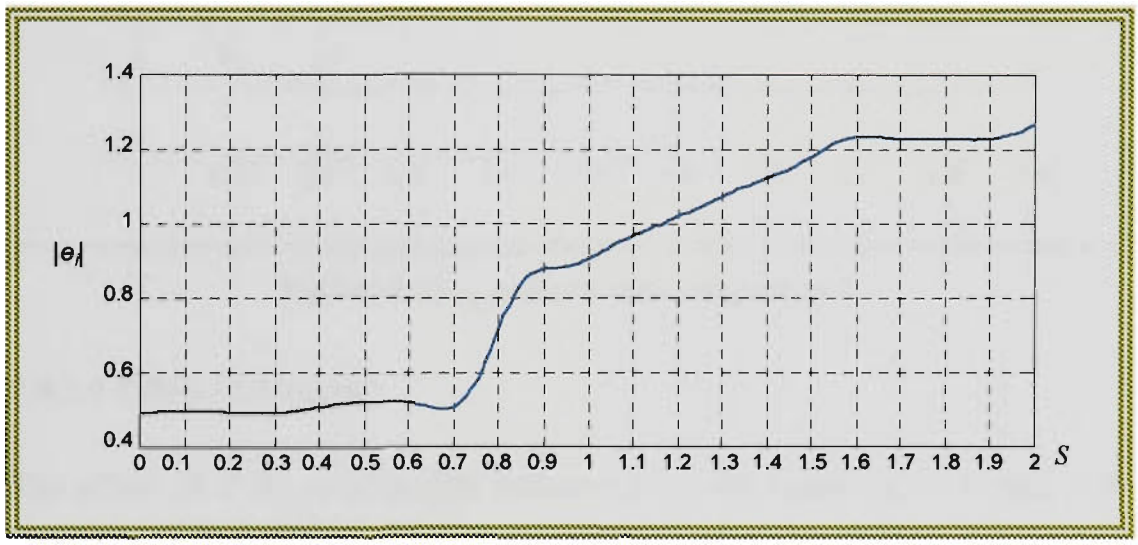


Figure 9.24 e_i pattern with respect to S

9.4.5.2 Effect of *A* factor

No clear pattern for the A-e_i characteristics was obtained out of the simulations, as other factors seemed to be dominating these characteristics.

9.4.5.3 Effect of *L* factor

Figure 9.25 shows that for the values of L below 0.5 e_i is proportionally decaying with an increase in L. However, e_i shows robustness for any variations in L for higher values of L ($L>0.5$). The L-e_i characteristics can be expressed as follows:

$$e_i \propto \frac{1}{L} \qquad \forall\, L \in [0,0.5]$$

$$e_i \cong constant \qquad \forall\, L \in [0.5,2.5]$$

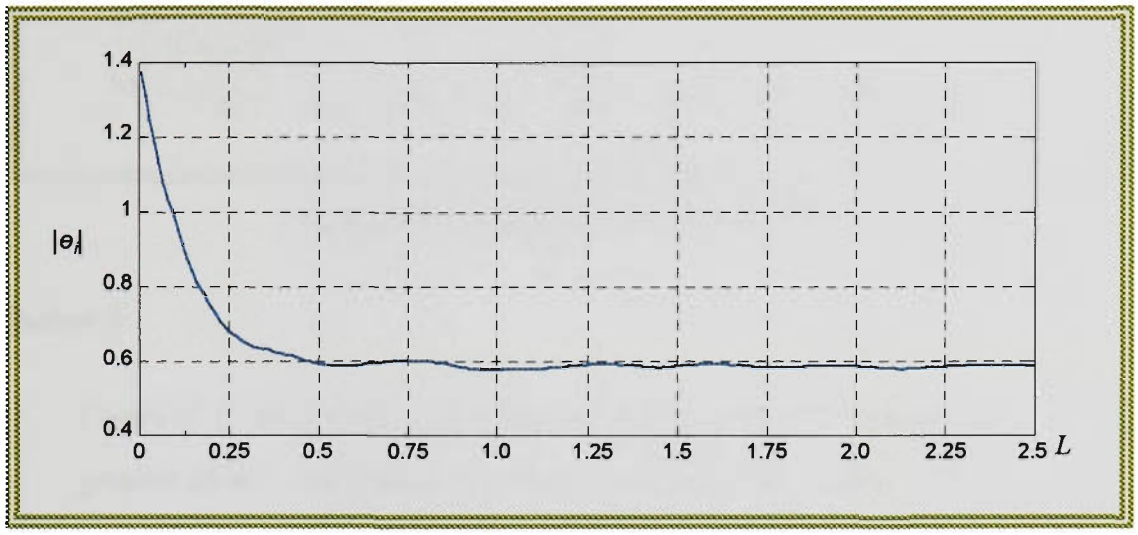


Figure 9.25 e_i pattern with respect to L

9.4.5.4 Effect of T factor

The effect of T on e_i is largely influenced by the values of L. Figure 9.26 shows that e_i will increase proportionally to any increase in T as long as L is large. However, the pattern's slope sign will be reversed in the case of smaller values of L. The T-e_i characteristics are expressed as:

$$Ei \cong constant \qquad \forall\, T \in [0,0.4] \cup [0.6,1]$$

$$Ei \propto T \qquad \forall\, T \in [0.4,0.6] \qquad If\ L\ is\ small$$

$$Ei \propto \frac{1}{T} \qquad \forall\, T \in [0.4,0.6] \qquad If\ L\ is\ large$$

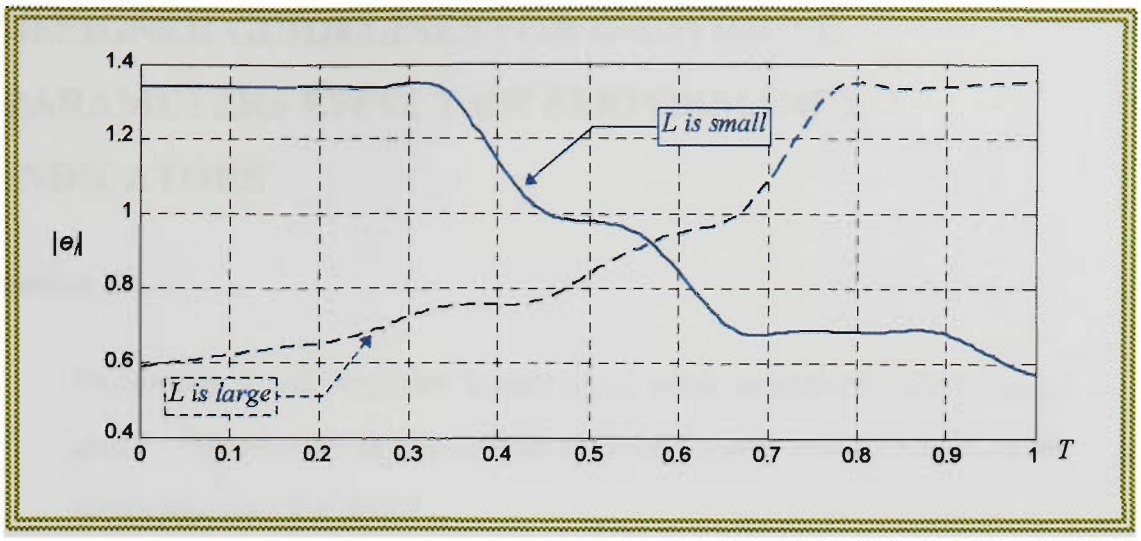


Figure 9.26 e_i pattern with respect to T

Conclusion 5

Figure 9.27 shows that e_i is robust to any changes in L when L is greater than 1, and that decreasing T will improve e_i when L is greater than 1. S is preferred to be below 0.7.

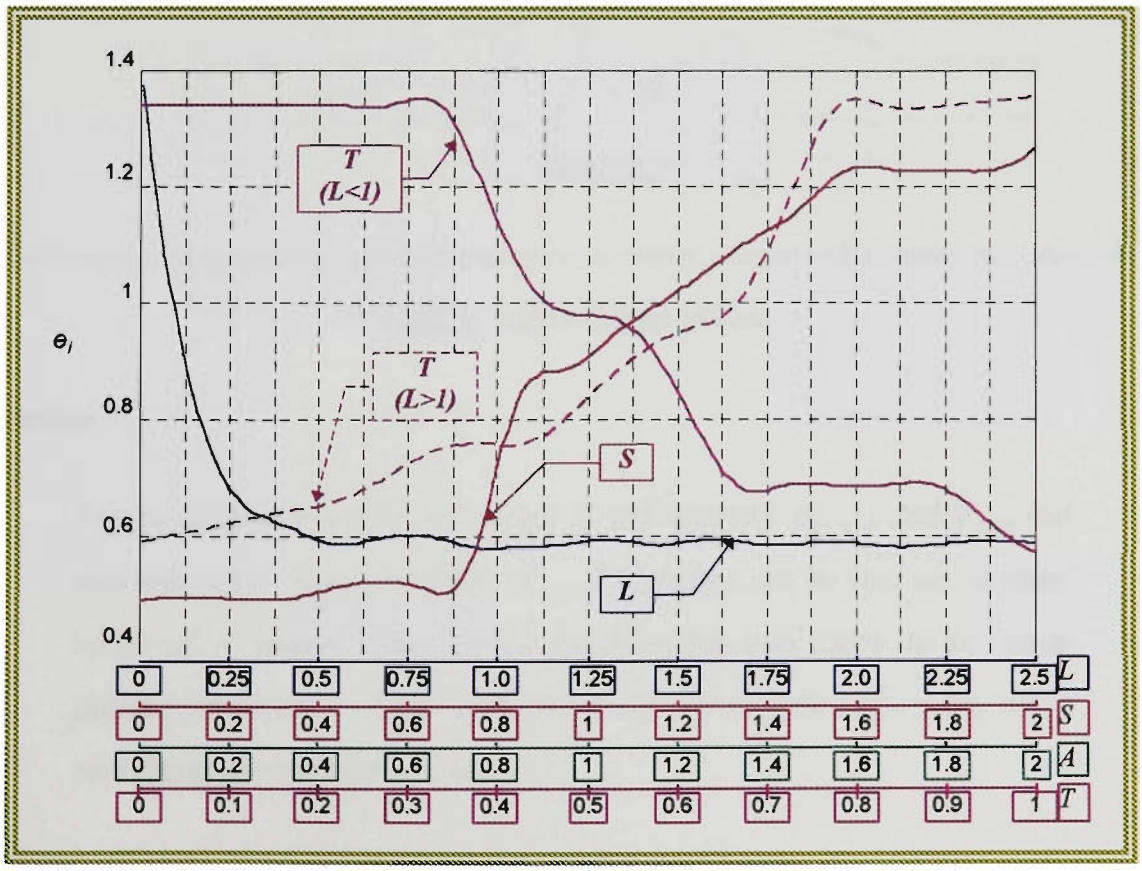


Figure 9.27 e_i patterns

9.5 DESIGNER GUIDELINES FOR INDIVIDUAL PARAMETERS EFFECT ON PERFORMANCE INDICATORS

Conclusion 6

Increasing S will improve t_r and $|e_s|_{av}$ while negatively effect μ_n, μ_p and e_t. Figure 9.28 shows that an optimal S value can be found in the range between 0.8 and 1.

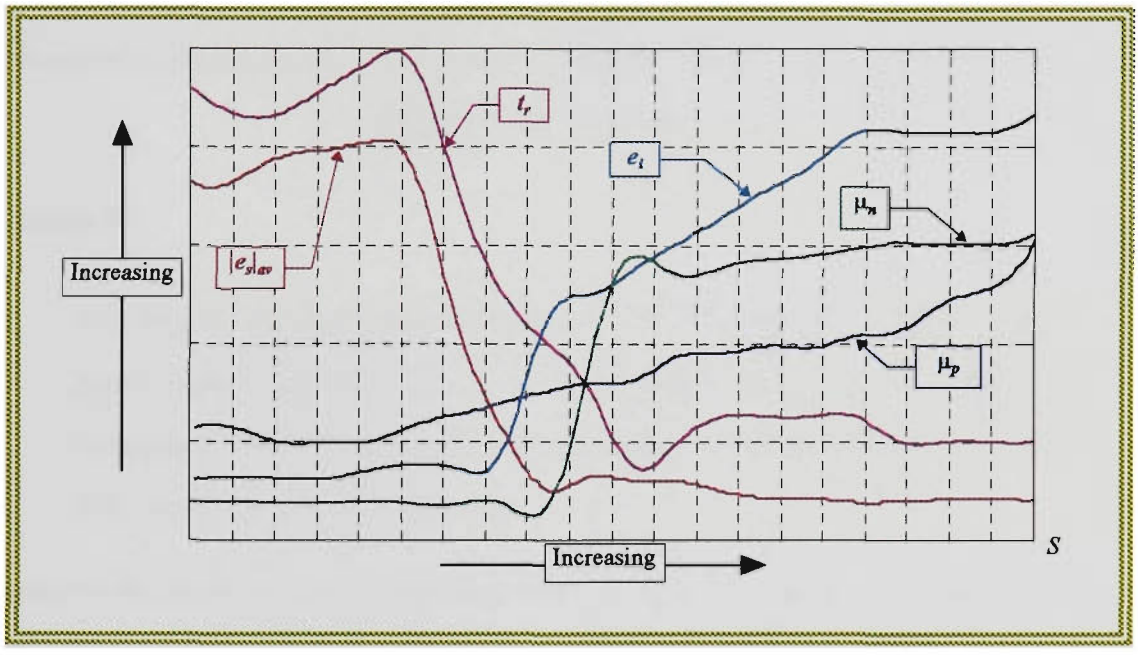


Figure 9.28 *S* factor effect

Conclusion 7

Figure 9.29 shows that increasing A will improve μ_n, μ_p and $|e_s|_{av}$ but this will have negative effect on t_r. The figure shows that an optimal value of A is not clear especially knowing that there is no clear pattern for A on e_t. Thus, it is recommended to take extra care when searching for a global optimal A.

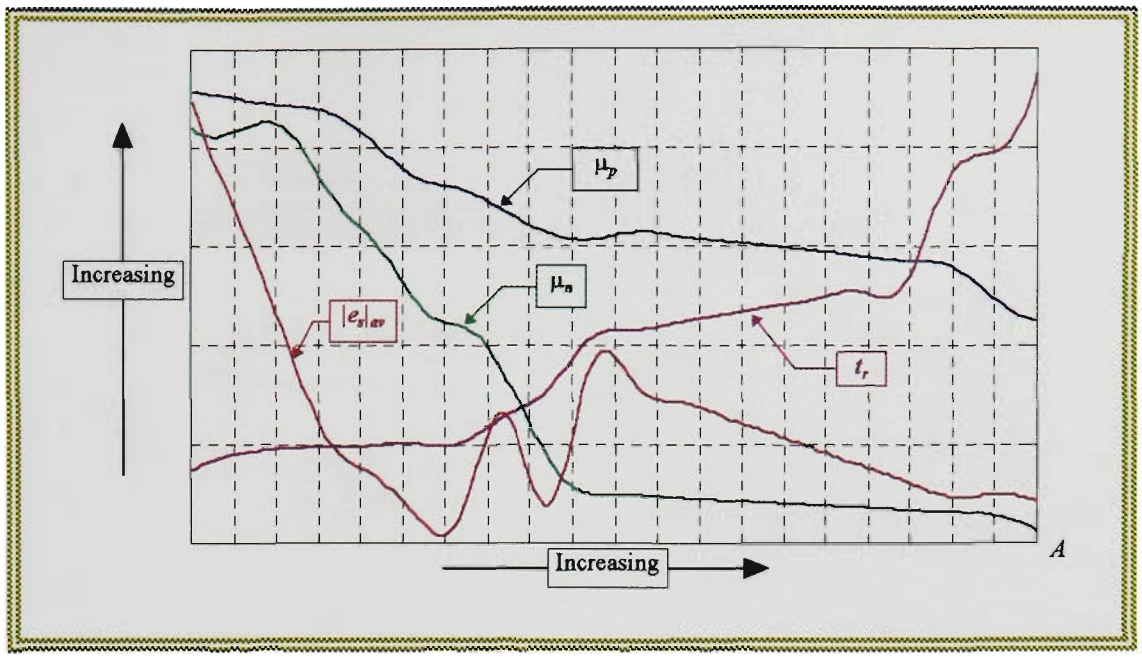


Figure 9.29 *A* factor effect

Conclusion 8

L is largely effected with the value of T. For better performance and a good global optimisation of both T and L, it is better to keep T between 0.7 and 1 and to keep L between 1 and 50. Figures 9.30 and 9.31 show the L and T patterns.

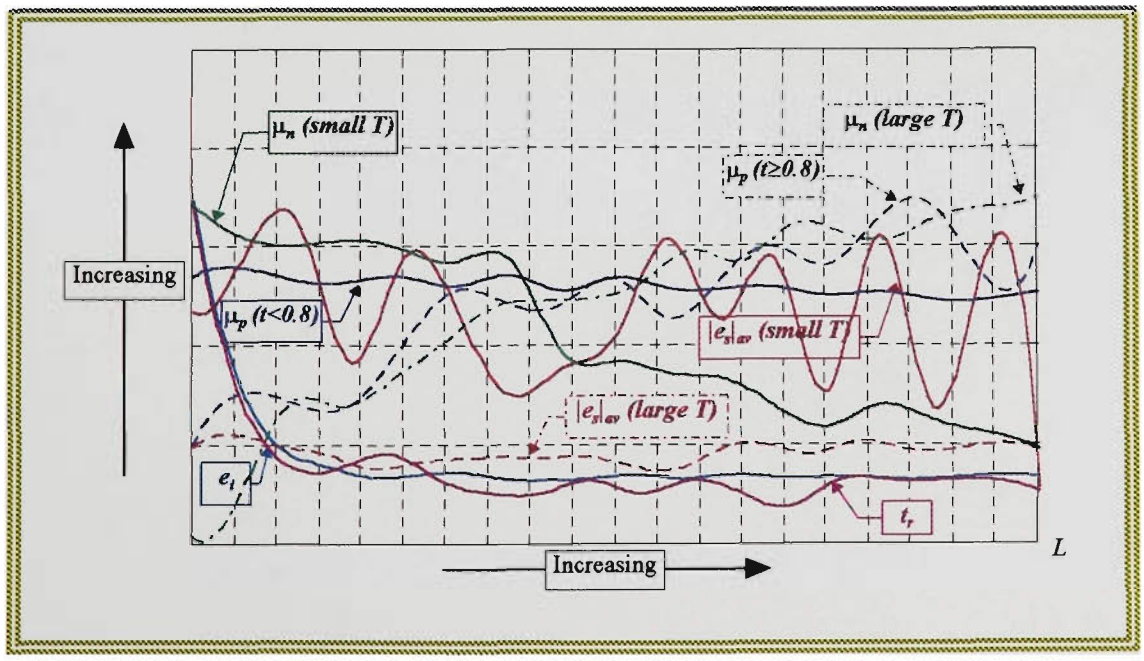


Figure 9.30 *L* factor effect

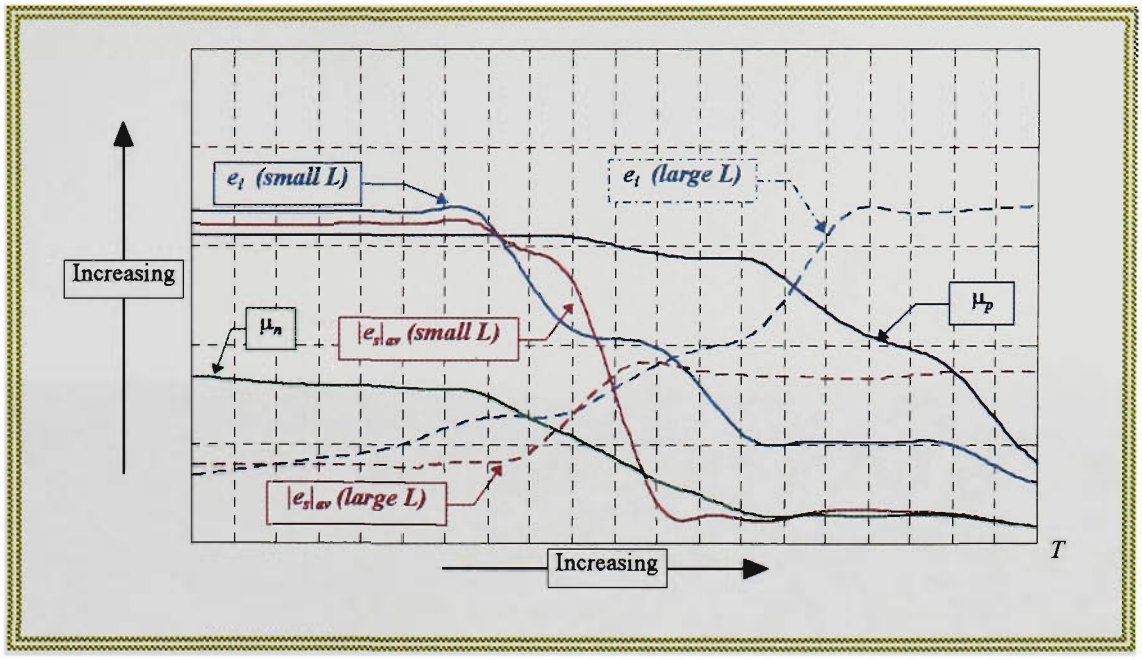


Figure 9.31 *T* factor effect

SECTION 3 IMPLEMENTATION OF A UNIVERSAL FUZZY CONTROLLER IN THE EXCITATION CONTROL

Chapter 10 *Shay in the Excitation Control*

10.0 INTRODUCTION

The proposed Shay-FLC architecture was used for the excitation control of a synchronous generator connected to an infinite bus through a transmission line. In order to cater for both voltage and rotor angle stability, the performance monitor used in the system considers the rotor angle speed deviation ($\Delta\omega$) and the error in the generator terminal voltage (v_t) compared to a reference signal (v_{ref}). This was achieved by having the input to the Main-FLC as well as the inputs to Shay-PA and Shay-Tune derived from the manipulation process of both signals in a Pre-Control stage (P-C). This manipulation stage is explained in detail in this chapter.

This chapter starts with a preliminary discussion about the forces and torques that effect the synchronous machine in operation. A control strategy is derived based on the discussion. The description of the P-C stage is presented next. The full Shay-Excitation system is shown at the end of the chapter.

10.1 PRELIMINARY DISCUSSION

This section discusses the main factors that effect the synchronous generator while in operation. The forces and torques the machine is subjected to are explained with their influence on both the stability and performance of the generator. The effect of these forces is encapsulated in the swing equation as shown in section 10.1.1. The graphical solution of the swing equation, refered to as the equal area criteria method, is described in section 10.1.2.

The following discussion is provided in order to build up the knowledge required and used in constructing the Shay-Exciter.

10.1.1 The Swing Equation

The synchronous machine is subject to torques and forces of different sources while operating. Some of these torques are caused by the machine itself and are due to its characteristics and others are developed and generated due to a

rotation of the rotating masses of the machine. Figure 10.1 shows these mechanical torques and their relative direction. These torques are:

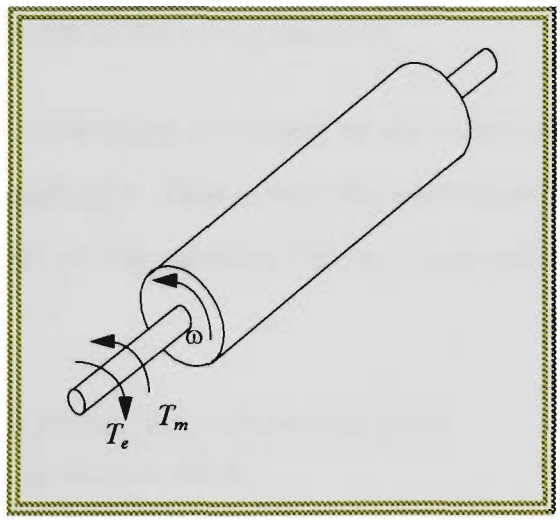
1) T_m, the mechanical torque supplied by the prime mover through the shaft,

2) T_J, the developed torque due to the moment of inertia J ($kg.m^2$) of the rotating masses in the machine,

3) T_B, the torque developed due to the coefficient of friction B ($N.m/rads/sec$),

4) T_e, the electrical torque developed by the machine,

5) T_D, the damping torque caused by the currents circulating in the windings of the rotor.



Figures 10.1 torques influencing the synchronous machine

Newton' s law relates these torques for the synchronous machine as

$$T_J+T_B+T_D=T_m-T_e=T_a \qquad (10.1)$$

Where T_a is the accelerating torque and

$$T_J = \frac{d\theta^2}{dt^2}$$

$$T_B = B\frac{d\theta_R}{dt}$$

Multiplying (10.1) by the rotor speed (ω_R) gives the torque equation in terms of power as

$$P_J + P_B + P_D = P_m - P_e = P_a \tag{10.2}$$

P_B in equation (10.2) can be neglected because of its small value compared to other parameters in the equation. The machine design considerations recommend B to be almost zero. P_J in equation (10.2) is mostly presented in terms of the angular momentum (M) where $M=J\omega_m$. Thus (10.2) can be rewritten as shown in (10.3).

$$M\frac{d\delta^2}{dt^2}\delta(t) + P_D = P_m - P_e = P_a \tag{10.3}$$

Equation (10.3) is known as the swing equation.

M can vary within a wide range according to the value of J, where J depends on the machine size and type. This is why the normalised inertia constant (H) is more preferred in the swing equation (10.3), H normally varies between 1 to 10 pu.

$$H = \frac{\text{stored kinetic energy at synchronouse speed}}{\text{generator rating}}$$

$$H = \frac{\frac{1}{2}J\omega_{synch}^2}{S_{rated}} \tag{10.4}$$

substituting H with (10.4) in the swing equation results in

$$\frac{2H}{\omega_{synch.}}\omega_{pu}(t)\frac{d\omega(t)}{dt} + P_D = P_m(t) - P_e(t) = P_a(t) \tag{10.5}$$

where

$$\omega_{pu} = \frac{\omega_r}{\omega_s}$$

$\omega_r \equiv$ rotor mechanical speed

$\omega_s \equiv$ synchronous speed

Note that the second parameter at the left hand side of equation (10.5) (P_D) will have a value not equal to zero only when the mechanical speed of the rotor (ω_r) is not equal to the synchronous speed of the machine (ω_s), ie. $\omega_r \neq \omega_s$. In other words, when $\frac{d\delta(t)}{dt} \neq 0$. Based on this, P_D can be approximated in a linear proportional relation with $\frac{d\delta(t)}{dt}$ as

$$P_D(t) = P_D \frac{d\delta(t)}{dt} \tag{10.6}$$

where P_D is the damping power per unit speed (*watt.sec/rad*).The parameter $P_e(t)$ in the right hand side of equation (10.5) is the developed electrical power by the synchronous generator. In the case of a single machine connected to an infinite bus, the case studied in this work, P_e is expressed as shown in equation (10.7).

$$P_e(t) = \frac{v_\infty v_t}{x} \sin\delta(t) \tag{10.7}$$

where, v_∞ is the infinite bus voltage and x is the transmission line reactance.

Solving the swing equation and plotting the resulting angle $\delta(t)$ versus time would give an idea of whether the system can keep or lose synchronism under severe disturbances. A straight numerical solution of the swing equation (second order differential equation) is not easy to realise and a computer simulation is needed to get the results. Simple linear approximations are available for the swing equation. However, in most cases of power systems disturbances, ie. short circuit, extensive loading, ... etc., linearisation is not

valid at all because of the large variations in the system parameters under severe disturbances. The non linear form of the swing equation is the only valid form for large disturbances stability analysis.

10.1.2 The Equal Area Criteria

The equal area criteria gives a graphical analysis of the power system stability for the case of a single machine (it is also applicable for double machines) connected to an infinite bus. This simple case of a single machine is not the real practice in power system networks. Large sized networks consist of much more than one or two generating units. However, the introduction of the equal area criteria in this section is aimed at gradually building the knowledge required to construct an excitation control system.

Under steady-state conditions x, v_∞ and v_t in equation (10.7) are constants, and equation (10.7) can be simplified by:

$$P_e(t) = P_{e_{max}} \sin\delta(t) \qquad (10.8)$$

Where $P_{e_{max}}$ is the maximum generated electrical power calculated as $\frac{v_\infty - v_t}{x}$.

Equation 10.8 is shown in the power-angle curve in figure 10.2.

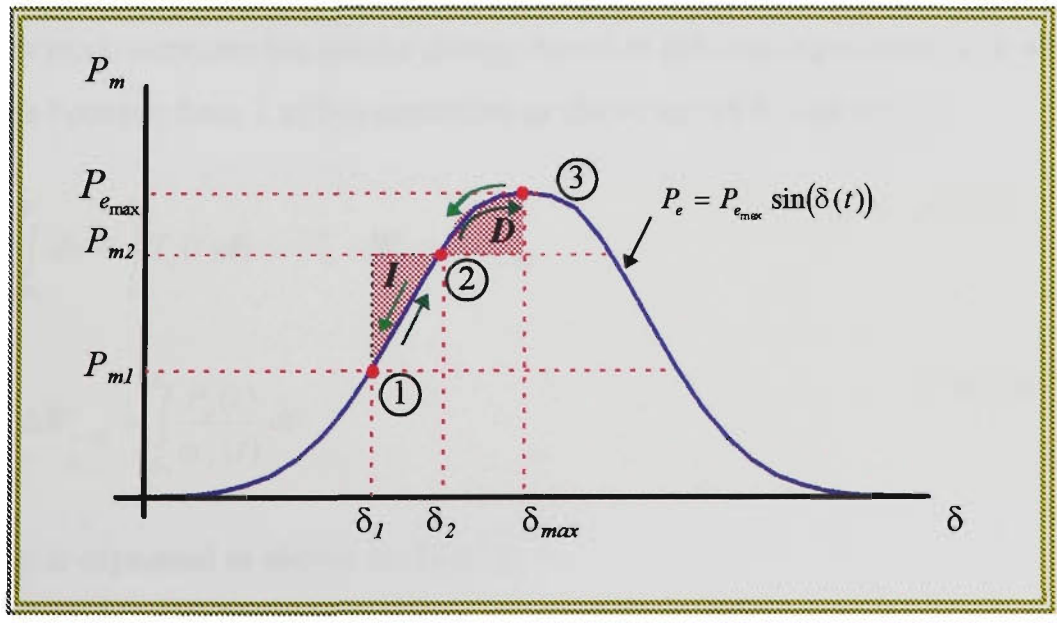


Figure 10.2 P_m and P_e versus δ

An equilibrium in power should occur when the machine is in the steady-state. This equilibrium occurs when the right hand side of swing equation, in (10.5), equals zero. In other words, when the generated electrical power (P_e) equals the imputed mechanical power (P_m), neglecting the losses. This situation will result in a zero acceleration power (P_a) or $\left(\frac{d\delta}{dt}=0\right)$. This equilibrium is shown at point 1 in figure 10.2. A sudden step change in the input mechanical power, ie. moving it up to P_{m2}, shall change the electromechanical equilibrium in equation (10.5) making $P_a>0$ as P_{m2} becomes greater than P_e. This extra mechanical power will be absolved by the system in the form of increasing rotational speed in the rotating elements of the synchronous machine. This extra speed will pull up the generated electrical power along the power angle curve to point 2, where $P_e=P_m$. But still, at this stage, ω_r is greater than ω_s, this will move the operational point towards δ_{max} where P_e reaches its maximum ($P_{e_{max}}$) and where $P_a=0$. At point 3, $P_e>P_m$ this will drive the system back to point 1 passing through point 2. If the damping effect of P_D in equation (10.5) is neglected, the operational point will keep swinging between points 1 and 2. However, due to the natural damping the operational point will settle at point 2 and equilibrium will be established again after some transients.

The rotor acceleration from point 1 to 2 is due to a work done on the rotor. This work increases the kinetic energy stored in the rotating masses. The work done between from 1 to 2 is expressed as shown in (10.9) and (10.10).

$$\int_{W_1}^{W_2} dw = \int_{\delta_1}^{\delta_2} T_a(t)d\delta = W_2 - W_1 = \Delta W_{1\to 2} \tag{10.9}$$

$$\Delta W_{1\to 2} = \int_{\delta_1}^{\delta_2} \frac{P_a(t)}{\omega_r(t)} d\delta \tag{10.10}$$

$P_a(t)$ is expressed as shown in (10.11),

$$P_a(t) = T_a(t)\omega_r(t) = P_m(t) - P_e(t) \tag{10.11}$$

substituting (10.11) into (10.10) yields

$$W_{1\to 2} = \frac{1}{\omega_s}\left[\int_{\delta_1}^{\delta_2}\left(P_m - P_{\max}\sin\delta\right)d\delta\right] \tag{10.12}$$

The integration result, parameter between brackets, in (10.12) is the shaded area (I) in figure 10.2.

At δ values greater than δ_2, P_e is greater than P_m, and this generates negative P_a. The rotor gradually loses the kinetic energy it has from 1 to 2, until it completely loses it at δ_{max} and it swings back a gain to point 1. From point 2 to 3 where $\omega_r=\omega_s$, the rotor is losing energy to the infinite bus. The work done by the rotor in this area is expressed as:

$$\int_{W2}^{W_3} dw = \int_{\delta 2}^{\delta_{\max}} T_a(t)d\delta = W_3 - W_3 = \Delta W_{2\to 3} \tag{10.13}$$

$$\Delta W_{2\to 3} = \int_{\delta_2}^{\delta_3}\frac{P_a(t)}{\omega_r(t)}d\delta \tag{10.14}$$

substituting equation (10.10) into (10.14)

$$\Rightarrow W_{2\to 3} = \frac{1}{\omega_s}\left[\int_{\delta_2}^{\delta_3}\left(P_m - P_{\max}\sin\delta\right)d\delta\right] \tag{10.15}$$

The parameter between brackets in (10.15) is the shaded area (D) in figure 10.2.

The two works, $W_{1\to 2}$ and $W_{2\to 3}$ in (10.12) and (10.15) represent the increase and decrease in the kinetic energy of the rotational masses in the generator rotor due to the variation in the rotor speed ω_r. Dynamics suggest that $W_{1\to 2}=W_{2\to 3}$ whenever $\omega_r=\omega_s$, the case at points 1 and 3, which means

$$\frac{1}{\omega_s}I = \frac{1}{\omega_s}D$$

so

$$I=D$$

Thus we can conclude that to satisfy the equal area criteria, areas I (increase) and D (decrease) should be equal.

The knowledge elicited from the equal area criteria method is used in formulating the control strategy implemented in the P-C stage to handle the rotor angle stability problem. The general control objective is to:

1) assure that both D and I areas are equal,

2) make D and I as small as possible.

10.2 CONTROL STRATEGY AND OBJECTIVES

The equal area criteria analysis in section 10.1.2 showed that any distraction in the electromechanical equilibrium between the supplied mechanical power and the developed electrical one will result in variations in the rotor angle position. The system's physical characteristics of the machine size and type, the transmission system in addition to the excitation control type and efficiency as well as the disturbance type and severity all are very important factors. These factors need to be considered when it comes to the power system's ability to sustain acceptable operation conditions under different situations. The excitation control is the only handy tool for the power engineer to alter from this combination. Alterations and changes in the controller response are required to help the generator under transient as well as steady-state conditions.

Considering P_e which is presented in equation (10.8) as $P_e = P_{e_{max}} sin(\delta(t))$. $P_{e_{max}}$ is a function of three variables, v_∞, v_t and x. It was mentioned in section 10.1.1 that these are constants under steady-state and thus $P_{e_{max}}$ is assumed to be constant. This fact will not be valid any more when disturbances occur. Transmission line short circuits can significantly effect x, and large sudden load variations alter v_t. v_∞ might be constant in most conditions

though it might suffer some changes as well. This means that $P_{e_{max}}$ cannot be considered as a constant under transient conditions. $P_{e_{max}}$ varies when disturbances occur giving the operational point a different curve than the original power-angle curve it started with before the disturbance. To measure the variations in P_e one can measure x or v_t, assuming constant v_∞. The transmission line reactance is not easy to measure accurately, so the only easy measurable parameter in equation (10.8) that effect $P_{e_{max}}$ is v_t. In the control strategy followed in this work v_t is under continuous monitoring with the objective to maintain certain levels of v_t under all circumstances.

In section 10.1.2 it was mentioned that when the electromechanical equilibrium in the power system is distracted, P_a will no longer be zero and so $d\delta/dt$ which is the speed deviation $\Delta\omega$. P_a might be an acceleration ($P_a>0$) or deceleration ($P_a<0$) power according to P_m and P_e.

Knowing that $P_a=P_m-P_e$, if P_m was larger than P_e then P_a is an accelerating power and $\Delta\omega$ is greater than zero. The control objective in this accelerating condition is to increase P_e by increasing the excitation signal. On the other hand if P_e was greater than P_m, P_a will be a deceleration power and $\Delta\omega$ will be negative. The control objective under this condition is to reduce P_e by reducing the excitation control signal. This is all based on the assumption of having constant mechanical power, which is the case considered in this work.

One other mile-stone in the strategy was the dynamics of the system. From the practical point of view it is not realistic to assume that P_e will settle in a new point instantly and without any intermediate transient modes. In fact when the system is subject to any disturbance that can distract the balance between P_e and P_m, resulting in either acceleration or deceleration operation, the excitation system will not be able to regain the balance immediately, some intermediate transients should occur. Figure 10.3 shows an example of a synchronous generator which was subject to a short period of load change[1]. Certainly after the sudden loss of the load and before the governor action took place, P_e was greater than P_m, the operational point moved to point 1 and then due to the deceleration control

[1] The synchronous generator parameters are included in chapter 11 section 11.1.1.2. The disturbance considered is a sudden inductive load change of 70% of the rated load sustained for 5 seconds and then removed.

action and the natural damping of the system the operational point swings back to point 2. The swings continue with decaying amplitudes until P_e settles back in its original position. These swings are identified by the red crosses in the curve in figure 10.3. Note that figure 10.3 shows the power-angle curve after the sudden removal of the load.

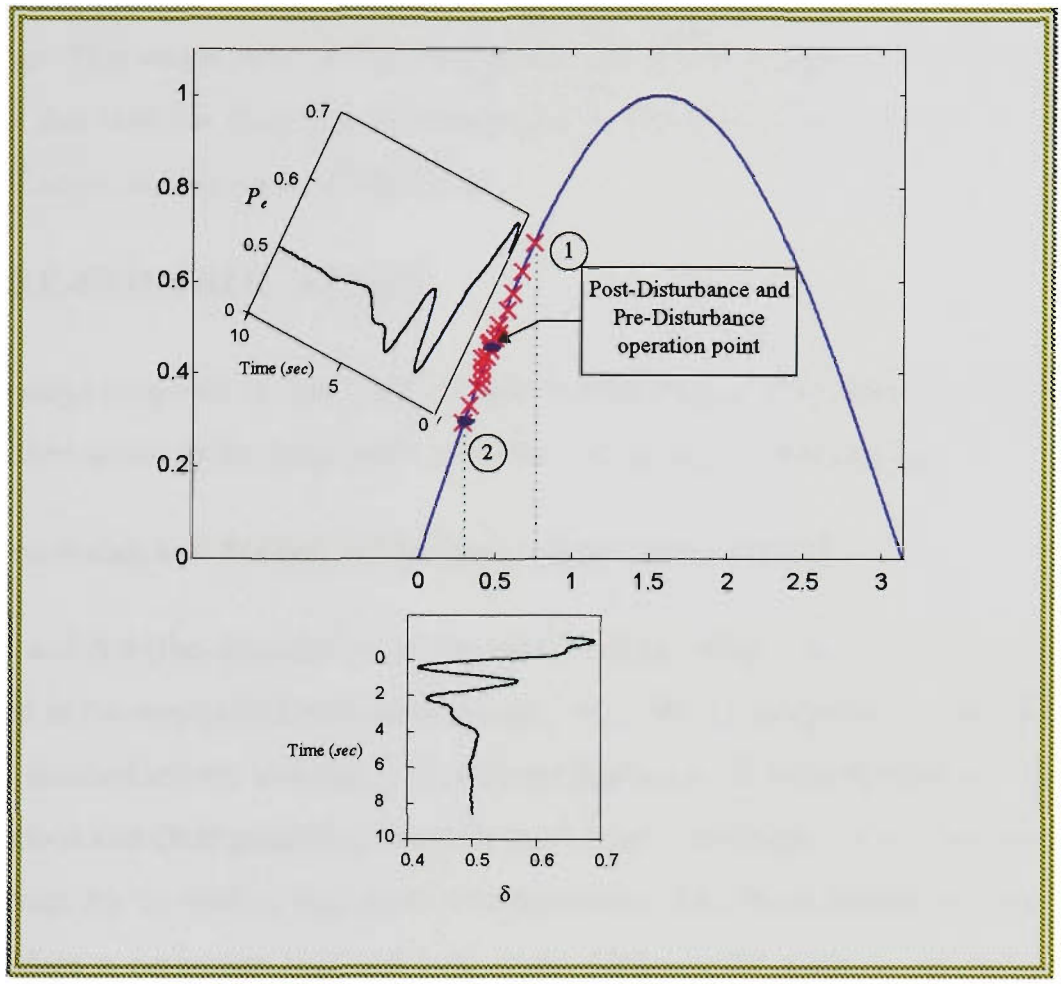


Figure 10.3 variations in P_e and δ after sudden load change

The control objectives from the excitation control in this work was to: " *maintain the terminal voltage and the speed deviation to the minimum under normal and abnormal operation conditions, within very small margins of an error, and to return to these margins after any disturbance within a short time, and with the minimal oscillations*".

To achieve these objectives, $\Delta\omega$ and v_t are considered as inputs to the excitation control system. These two parameters cover the rotor angle and the terminal voltage stability problems and considerations.

It is very obvious that there is a great deal of interconnection between the two factors, as any disturbance in v_t will result in a similar one in $\Delta\omega$, especially when using fast

excitation control action, which is one objective of this work. Moreover, it was noticed by experience that in most cases Post-Disturbance stability problems are due to the effect of one factor on the other (rotor angle control and terminal voltage control). To counter this conflict, the decision was to use an integrated single input that encapsulates the required information about both factors (the rotor angle and the terminal voltage) in A Pre-Control (P-C) stage. This single input is then used as the performance monitor (E_k) to an excitation controller that uses the Shay structure explained in section 2. This controller is referred to by Shay-Exciter in later parts of the thesis.

10.3 PRE-CONTROL STAGE

The P-C stage proposed in this work considers producing a processed scaled $\Delta\omega$ ($A_s\Delta\omega$) which is then added to the error in the terminal voltage (e_{vt}) to produce E_k.

10.3.1 Processing and Scaling of the Rotor Angle Speed Deviation

$\Delta\omega$ and $\Delta\varpi$ (the acceleration in $\Delta\omega$ calculated as $\Delta\varpi(k) = \Delta\omega(k) - \Delta\omega(k-1)$) are used in the manipulation stage that scales $\Delta\omega$. The considerations in this phase are directed mostly towards the plants dynamics, $\Delta\omega$ is considered as the most obvious and clear parameter to suffer in transient conditions. In this process the current $\Delta\omega$ is used as well as its first derivative ($\Delta\varpi$). Both signals are used to produce a parameter that can help when added to the error in the terminal voltage to evaluate the current and anticipated future states of the plant.

In order to reduce the swinging of the operational point along the power-angle curve it is very important to have control signals that consider a trend in the system's behaviour. For instance smaller $\Delta\omega$ with very large $\Delta\varpi$ can indicate larger $\Delta\omega$ in future readings and thus a precautionary control signal is required to reduce the deviation. Moreover a large $\Delta\omega$ signal at time k with very large $\Delta\varpi$ indicates that stronger control actions are required. The general guideline is that when the acceleration of $\Delta\omega$ is large, more emphasis on the direction of future states should be given. So if $\Delta\omega$ is positive and $\Delta\varpi$ is negative this can indicate that the system is moving towards a reduction in $\Delta\omega$ amplitudes, and it implies that the excitation system should pay less attention to the rotor angle

stability, than it is paying to the terminal voltage one. However, a larger $\Delta\varpi$ can assist the excitation system to predict larger $\Delta\omega$ amplitudes in future steps, so a precautionary control action should take place.

The manipulation of $\Delta\omega$ and $\Delta\varpi$ is achieved by a two input single output FLC shown in figure 10.4. The stability considerations require an implementation of three types of rules in the FLC. These three types of rules are identified by different shadings in figure 10.4. The FLC in figure 10.4 implements the following control objectives:

- **Acceleration control**: a set of rules that are responsible for giving the excitation system the signal to boost up the excitation in order to increase P_e.
- **Deceleration control**: rules that are responsible to indicate to the excitation system to bring down the excitation signal in order to reduce P_e.
- **Breaking control**: rules that are responsible of producing the precautionary signals that will help the system to over come any anticipated future undesired behaviour.

10.3.2 Voltage Stability

The generator terminal voltage stability is accounted for by considering the error in the generator terminal voltage (e_{vt}) compared to a reference voltage (v_{ref}), e_{vt} is calculated as:

$$e_{vt} = v_{ref} - v_t \tag{10.16}$$

The performance monitor used as an input to Shay-FLC (E) is calculated as;

$$E = A\Delta\omega + e_{vt} \tag{10.17}$$

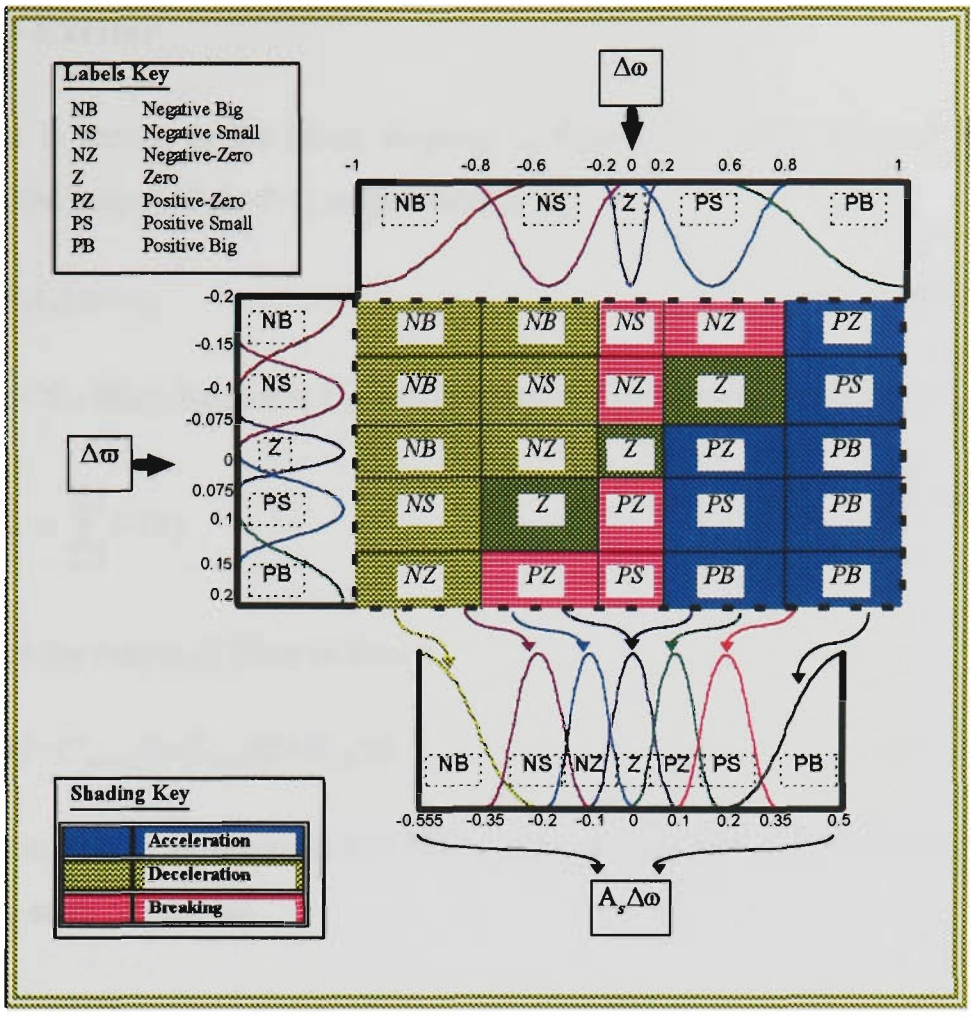


Figure 10.4 P-C FLC

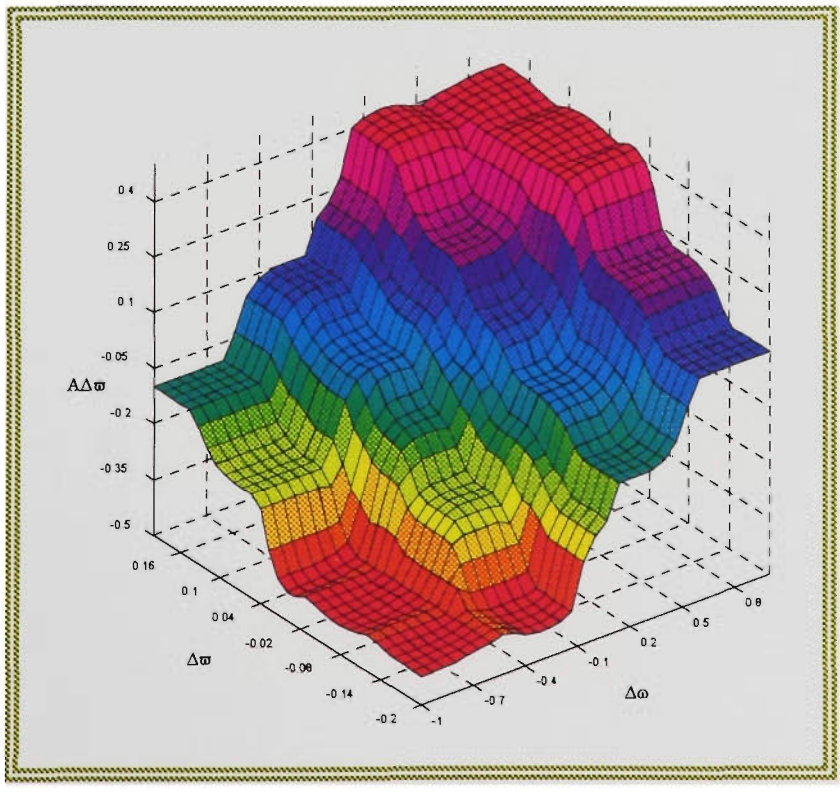


Figure 10.5 P-C FLC control surface

10.4 Shay-Exciter

Shay-Exciter is shown in the block diagram in figure 10.6. Note that the input to the controller is the output of the P-C stage which is E_k,

$$E_k = A_s \Delta\omega + e_{vt} \tag{10.17}$$

The output of the Shay-Exciter is U_{fd} which is the excitation signal calculated as:

$$U_{fd} = \sum_{k=0}^{M-1} U(k) \tag{10.18}$$

where $U(k)$ is the output of Shay at time k.

$$U(k) = U_{main}(k) \times G_{out}(k) \times U_{PA}(k) \tag{10.19}$$

The summation point in equation (10.18) is used to maintain constant excitation levels under steady-state conditions.

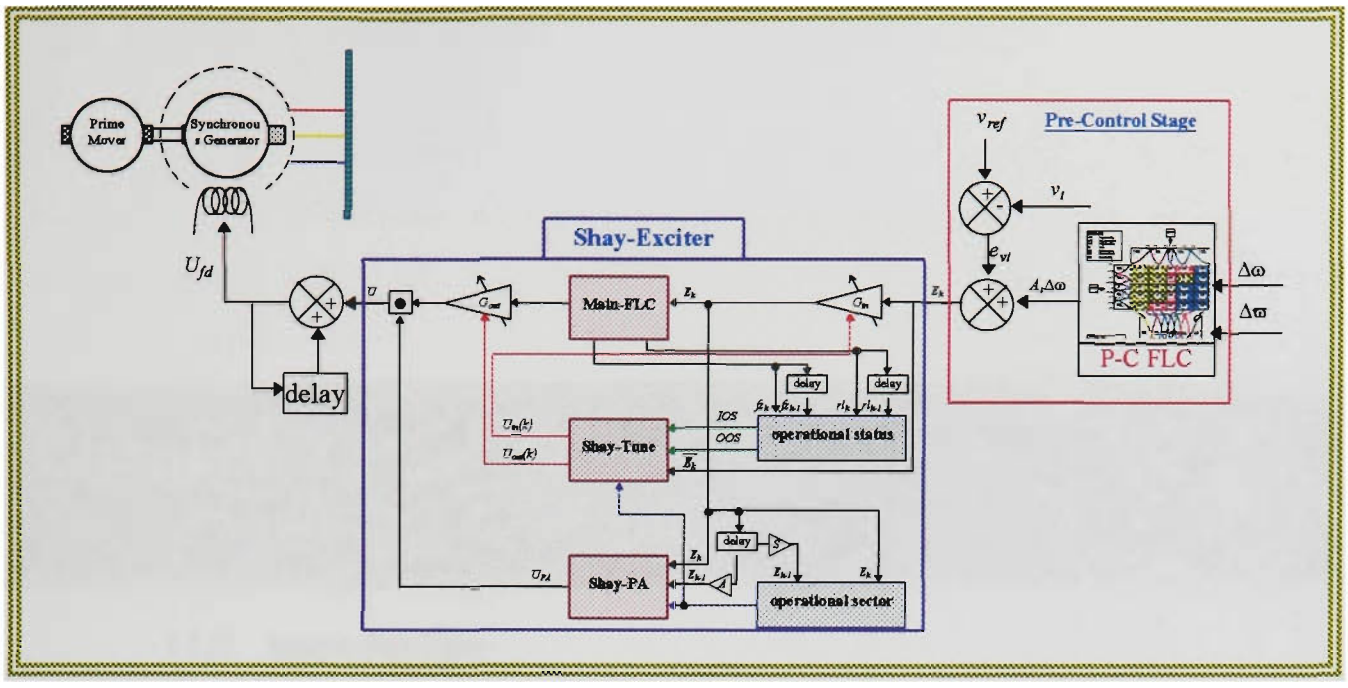


Figure 10.6 Shay-Exciter block diagram

Chapter 11 *Hardware and Software Components Developed*

11.0 INTRODUCTION

The implementation study was performed in a laboratory setup of a synchronous generator connected to an infinite bus through a transmission line. Shay-Excitation system was implemented in a C-program downloaded to digital signal processing board (DSP), DPC/C40 board mounted with two TMS320C40 processors[1]. A 5 KVA synchronous machine was used as a generator and a dc motor was used as the prime mover. The adjustable reactors as well as variable load banks were also applied in order to imitate the most of the real time components of a power system network. A general view of the laboratory setup is shown in figure 11.1.

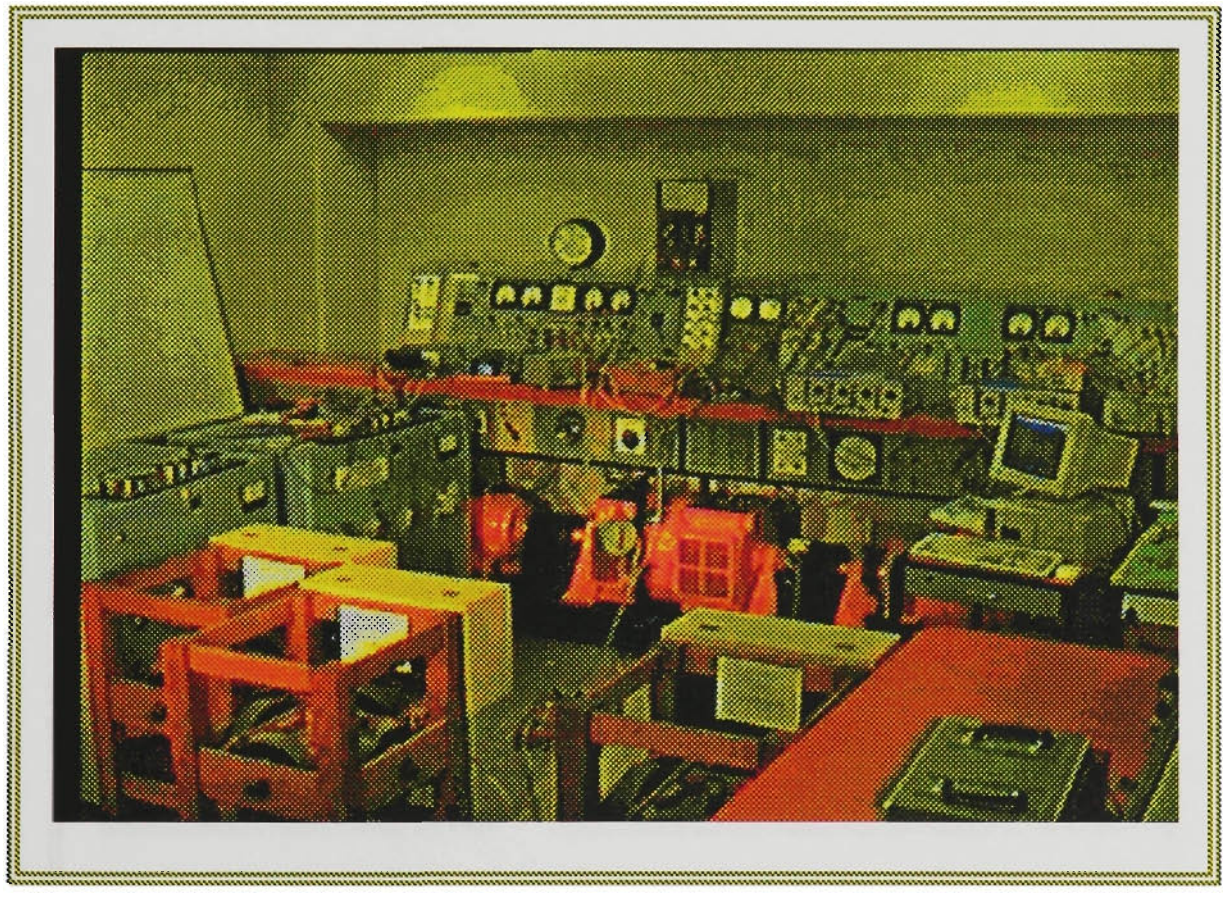


Figure 11.1 general view of laboratory setup

The implementation study required the development of both hardware and software tools to facilitate the implementation of the digital form of Shay-FLC, as well as to allow safe interfacing between the synchronous generator and the DSP board, in addition to providing a flexible man/machine interface through software. This section gives a detailed

[1] Texas Instruments C40 Processors

description of both the hardware and software tools developed and used in the practical implementation study.

11.1 HARDWARE DEVELOPED AND USED

The general view of the laboratory setup is described in a block diagram form in figure 11.2. The figure shows all the hardware circuit used to interface the synchronous generator and both the host PC and the DSP board. The interconnection between the synchronous generator and the infinite bus via the transmission line is also shown in the block diagram. These hardware components are fully detailed in this section.

11.1.1 Primary Hardware Used

11.1.1.1 Prime mover

The objective of the prime mover is to provide the required constant mechanical torque to the synchronous generator shaft. A Scott motor was used for this purpose. The motor was fed from the infinite bus, which is sufficiently stable, so a constant mechanical power output was possible from the dc motor. The motor platform and view are shown in figure 11.3.

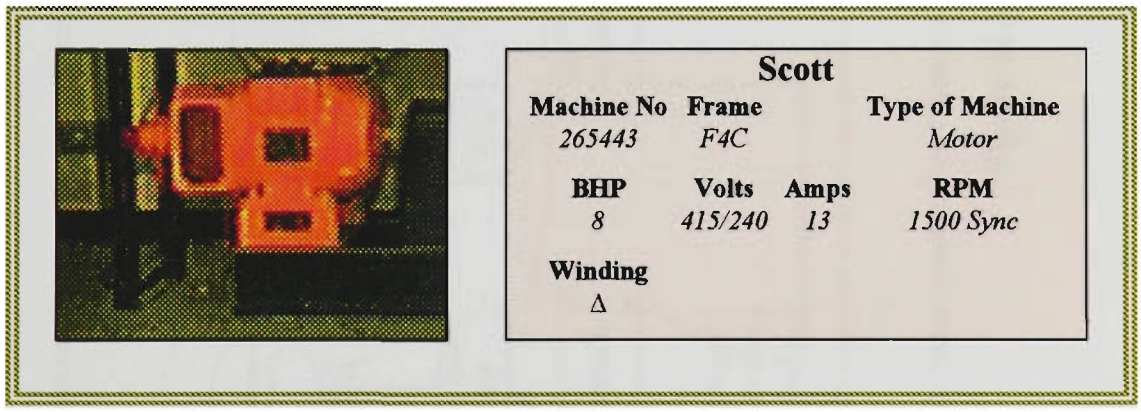


Figure 11.3 dc motor (prime mover)

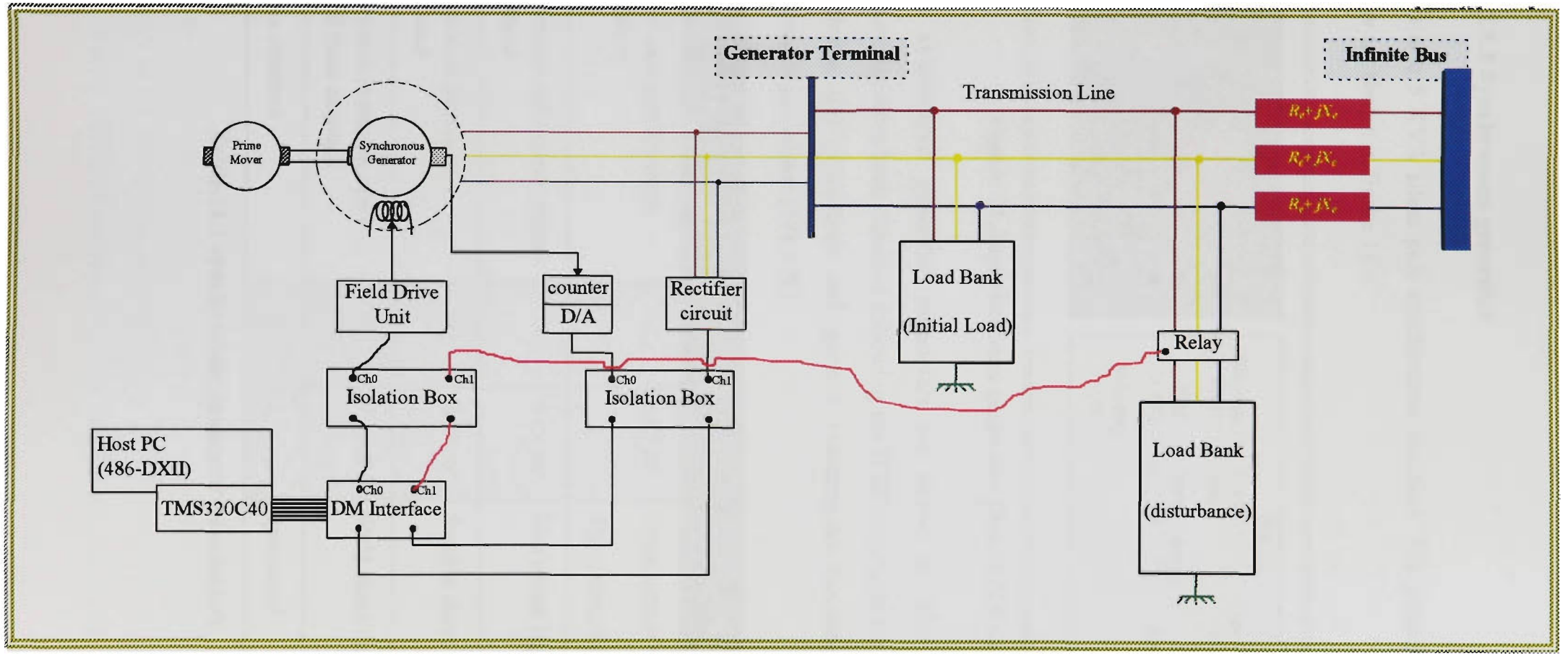


Figure 11.2 laboratory setup block diagram

11.1.1.2 Synchronous generator

This is a 5 KVA silent pole synchronous machine. The generator plate and view are shown in figure 11.4.

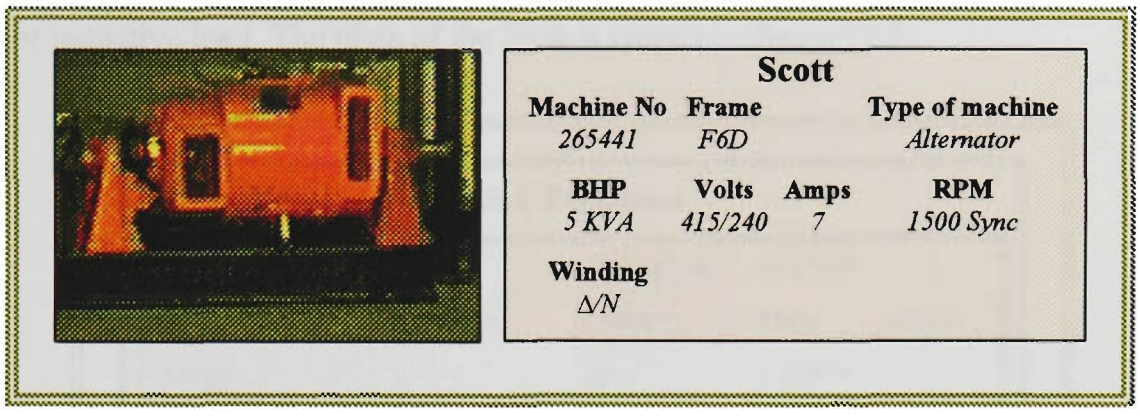


Figure 11.4 synchronous generator (Scot 5 KVA)

The synchronous generator parameters are shown in table 11.1, these parameters have been obtained following the IEEE standards for synchronous machines test procedures and general synchronous machine parameters estimation guidelines [178,179].

Parameter	Symbol	Value	Test/s
Direct-axis synchronous reactance	X_d	*1.03 pu*	Open circuit saturation test Short circuit saturation test
Quadrature-axis synchronous reactance	X_q	*0.49 pu*	Maximum lagging current
Direct-axis transient reactance	$X_d`$	*0.48 pu*	Sudden short circuit
Direct-axis transient open circuit time constant	$\tau_{do}`$	*0.34 sec*	Field short circuit
Inertia constant	H	*1*	Estimated

Table 11.1 synchronous generator parameters

11.1.1.3 Load banks

Two types of load banks were used, active/reactive and resistive load banks. The active/reactive load bank features a current range from 0 to 7 Amperes and a variable power factor from 0.1 to 0.98. The bank can be used as a capacitive or inductive load. The plate of the bank is shown in figure 11.5.

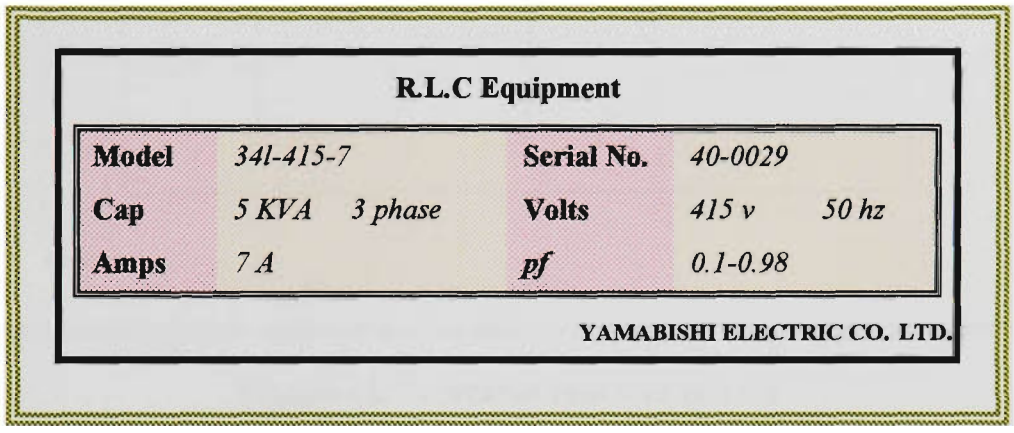


Figure 11.5 active/reactive load bank plate

The resistive load bank used was 9 KW, 230 *v* load bank with variable load settings ranging from *1×500* to *5×500 watts/phase*.

11.1.1.4 Adjustable Reactors

The adjustable reactors used in the transmission line have the plate shown in figure 11.6.

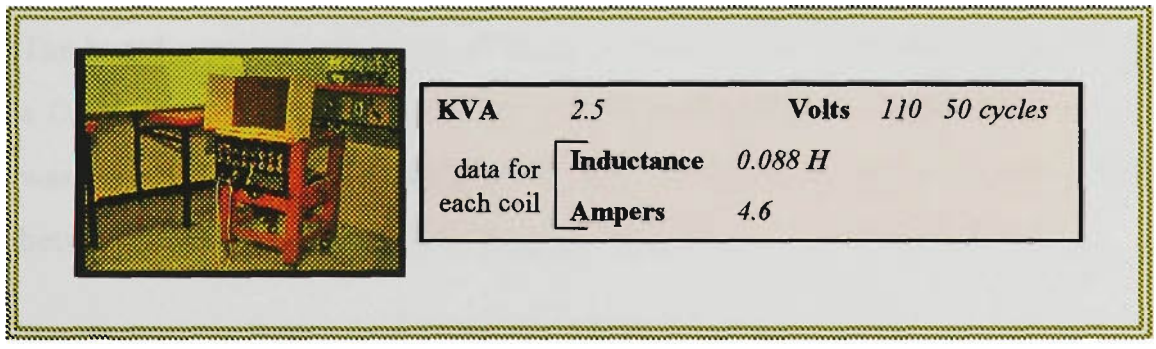


Figure 11.6 adjustable reactors

Figure 11.7 shows the circuit diagram for the adjustable reactors. Different settings of the switches have been used in the laboratory experiments in order to simulate different operating and transmission conditions, as different

settings of the transmission line will yield different contributions of the infinite bus to the local and disturbance loads.

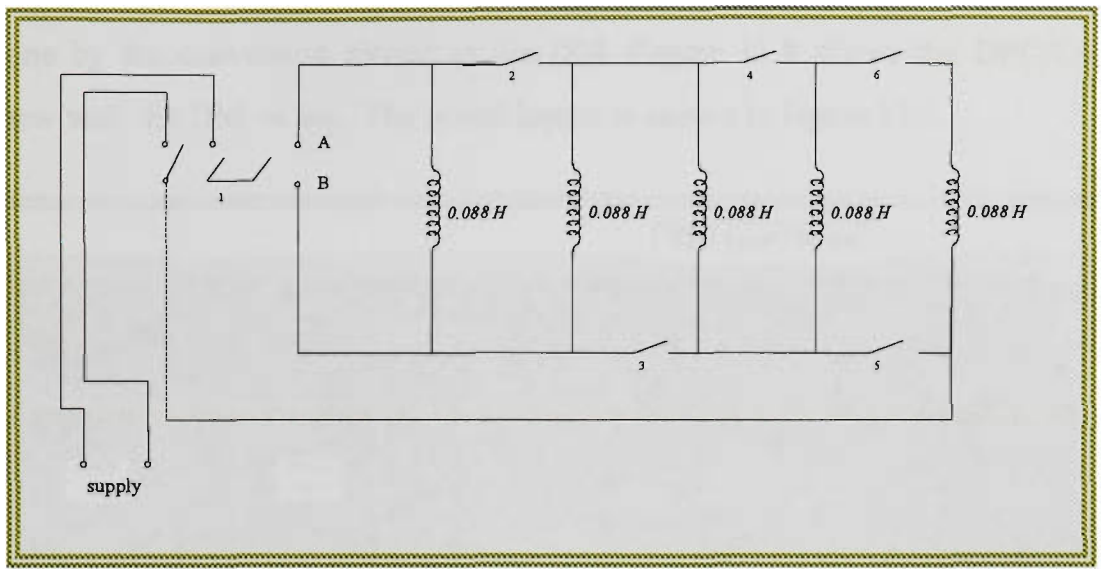


Figure 11.7 variable reactors circuit

11.1.1.5 DSP board (TMS320C40)

The DSP board used in the application was the Texas Instruments floating point TMS320C40 parallel digital signal processor (DSP). The DSP was mounted on a DPC/C40B board with two TMS320C40 processors (SITE A and SITE B). The board is clocked by a 40 *Mhz* clock, and is capable of having 2 million instructions per second (20 *MIPS*). A peak arithmetic performance of 220 million operations per second (220 *MOPS*) is possible to achieve.

The board contains two banks of *Static Random Access Memory* (SRAM)[2] and a *Dual-Port Random Access Memory* (DPRAM) block. The DPC/C40B board was hosted by a 486 DXII mother board PC. Flexible and fast communication between the DSP board and the PC was achieved through the DPRAM block.

Interfacing the DSP board with the isolation box was done via a *Daughter Module* (DM) (AM/D16 SA) mounted on the DPC/C40B board. The DM allows a 16 bit Analogue/Digital (A/D) conversions with a programmable sampling rate up to 200 *Khz*.

[2] 32K×32 in bank 0 and 512K×32 in bank 1

The DM has two analogue input channels (AIN_A, AIN_B) and two analogue output channels (AOUT_A, AOUT_B) with a voltage span of ±3 volts. All signal conversions from analogue to digital or vice versa were automatically done by the conversion circuit in the DM. Figure 11.8 shows the DPC/C40 view with the DM on top. The board layout is shown in figure 11.9.

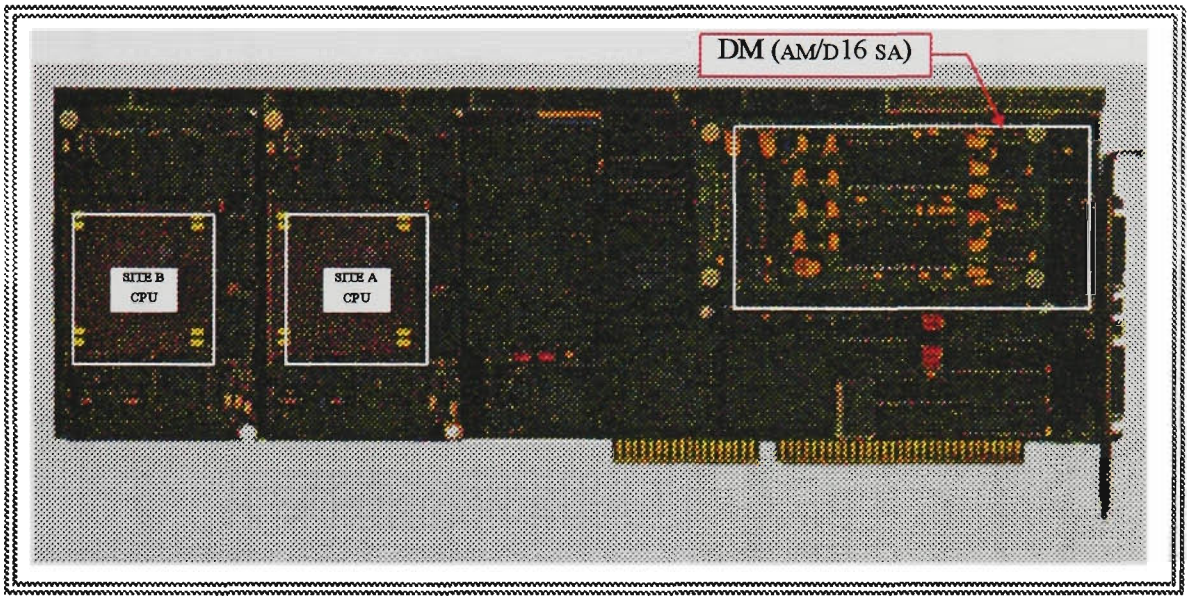


Figure 11.8 DPC/C40B board view

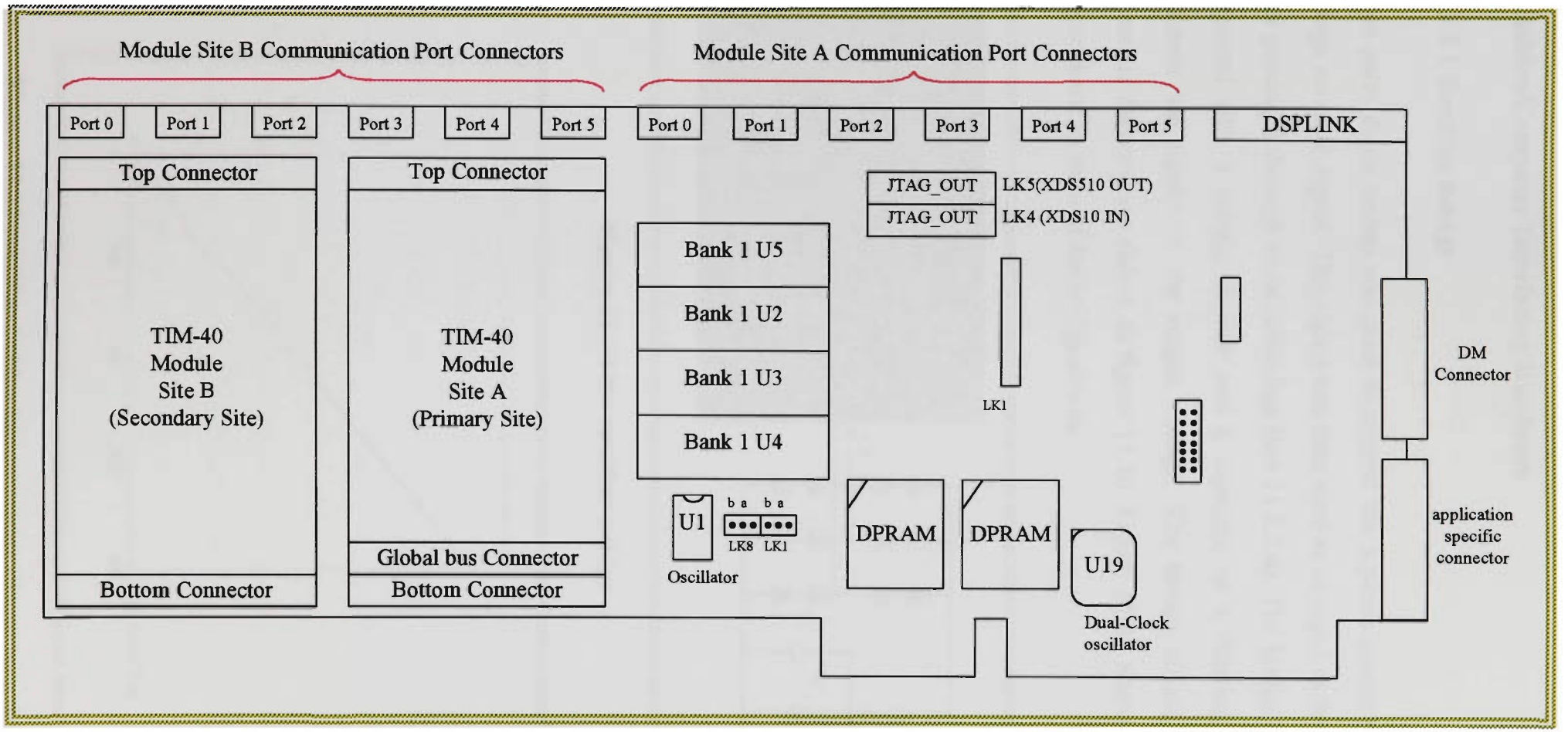


Figure 11.9 DPC/C40B board layout

11.1.2 Machine/Computer Interfacing Hardware

11.1.2.1 Rectifier bridge

A six pulse diode bridge was used to convert the 3 phase generator terminal voltage into a dc signal. This signal was then used as an input to the controller after passing it through an isolation box (see 11.1.2.4). The bridge circuit was supported with a voltage divider and a capacitor in a filtering circuit to eliminate the ripples in the output voltage. The bridge circuit view and schematic diagram are shown in figure 11.10. Figure 11.11 shows the input output characteristics of the bridge circuit.

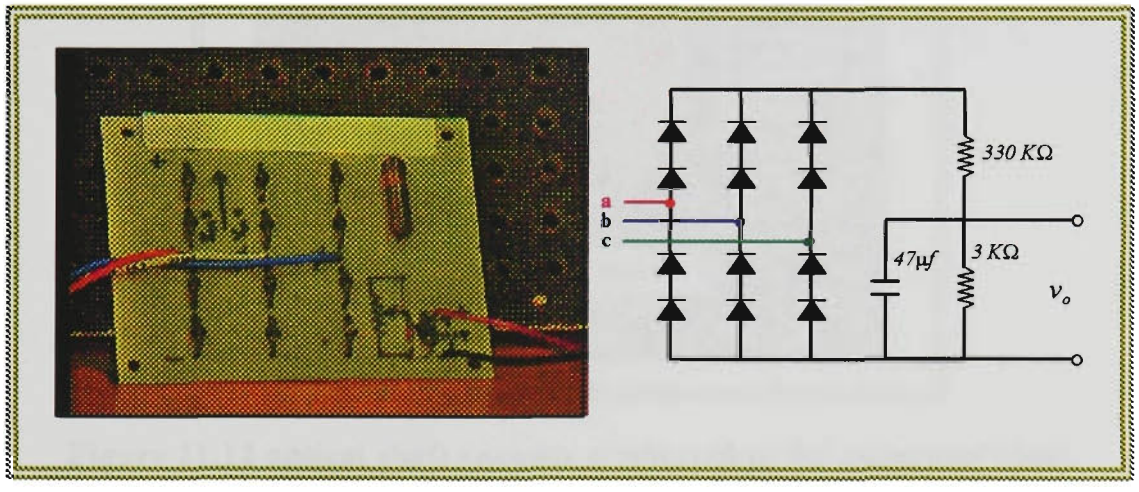


Figure 11.10 the rectifier bridge

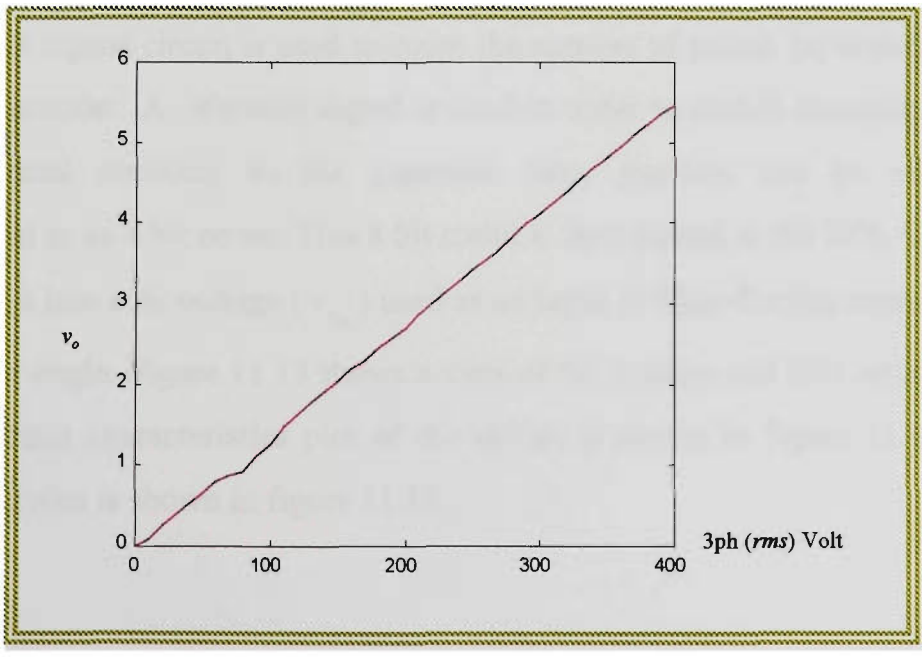


Figure 11.11 rectifier bridge input/output characteristics

11.1.2.2 Optical shaft encoder

An incremental type optical shaft encoder is connected to the synchronous generator in order to measure the rotor position (figure 11.12). The encoder produces 360 pulses every 360 electrical degrees, thus one pulse per electrical degree. The pulses as well as a reference signal are then passed to a counter and digital to analog (D/A) converter circuit. The counter circuit produces an 8 bit digital count representing the rotor position with respect to the reference signal (see 11.1.2.3).

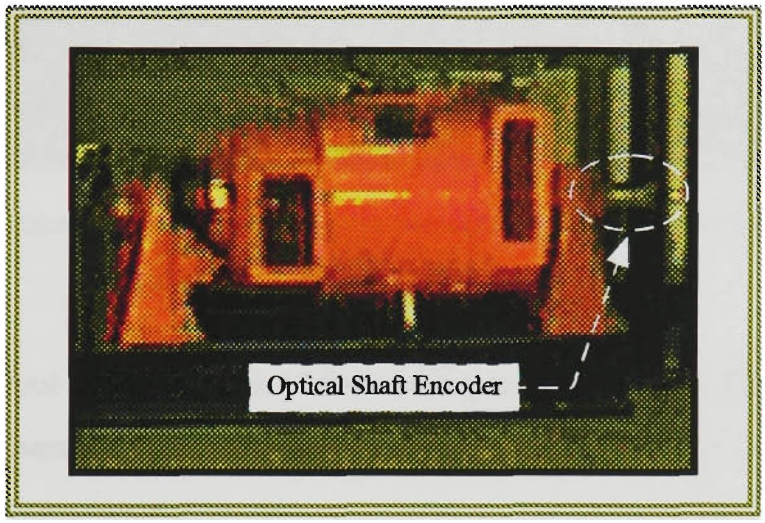


Figure 11.12 optical shaft encoder connected to the generator shaft

11.1.2.3 Counter and D/A circuit

A simple digital circuit is used to count the number of pulses generated by the optical encoder. A reference signal is used in order to enable incremental and decremental counting so the generator rotor position can be relatively measured in an 8 bit count. This 8 bit count is then passed to the D/A circuit to convert it into a dc voltage ($v_{o_{\alpha}}$) used as an input to Shay-Exciter representing the rotor angle. Figure 11.13 shows a view of the counter and D/A circuit. The input/output characteristics plot of the circuit is shown in figure 11.14. The circuit layout is shown in figure 11.15.

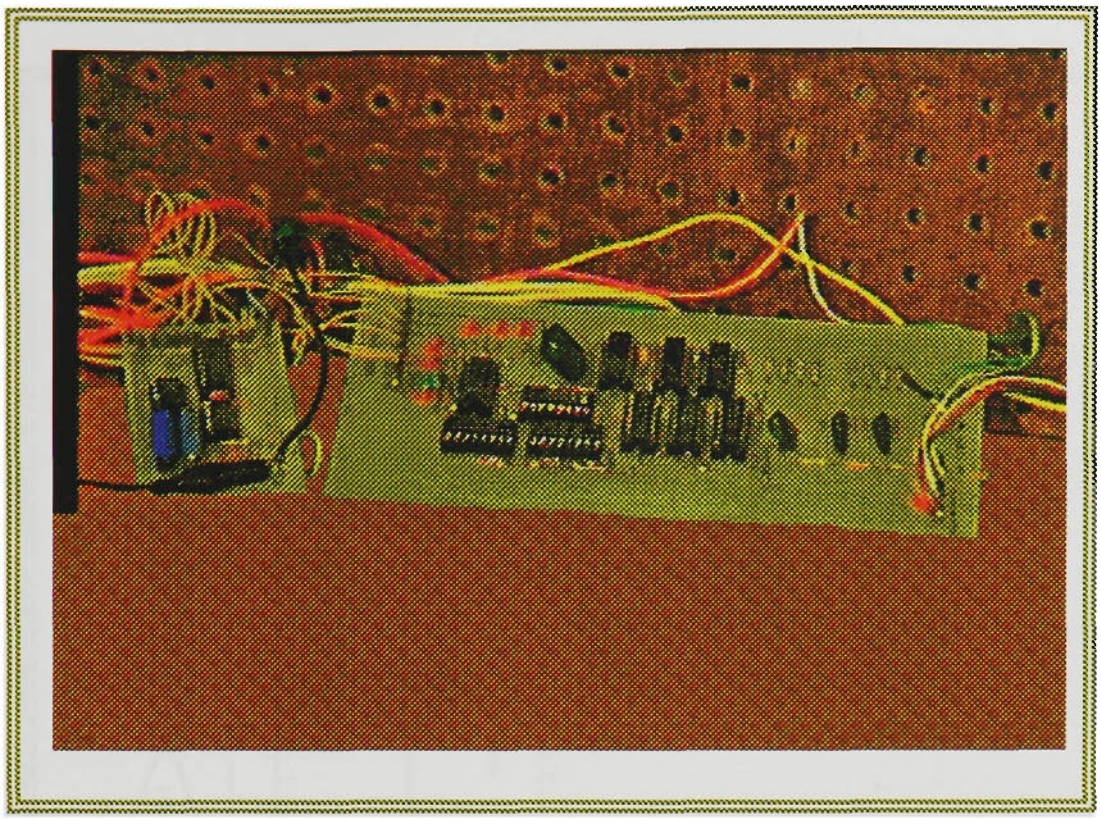


Figure 11.13 counter and D/A circuit view

The input/output characteristics of the counter and D/A circuit shown in figure 11.14 can be best approximated by the second order form as in equation (11.1)

$$v_{o_{dc}} = 0.0001\delta^2_{mech} + 0.1494\delta_{mech} + 0.192 \qquad (10.1)$$

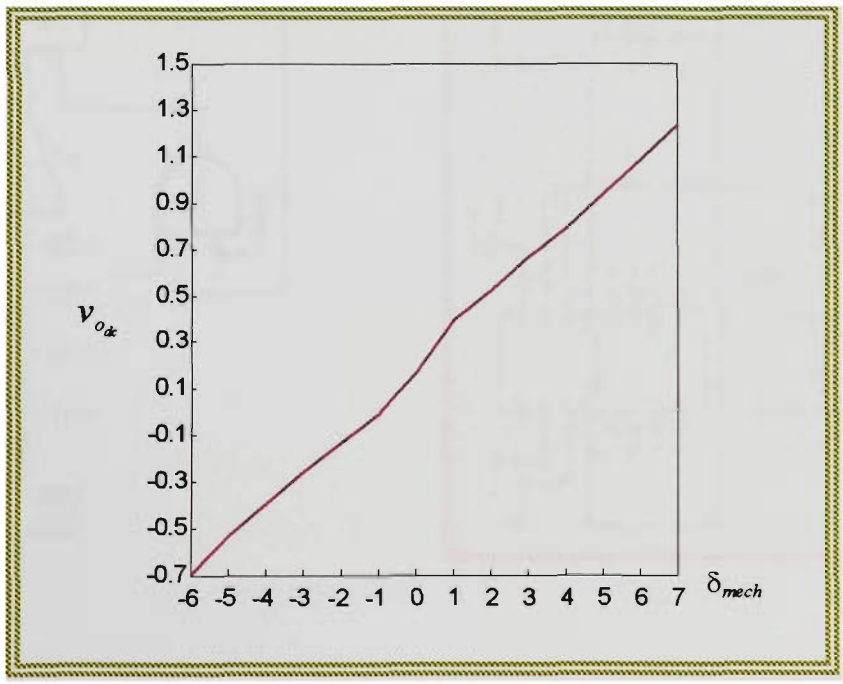


Figure 11.14 δ_{mech} versus $v_{o_{dc}}$ from the counter D/A circuit

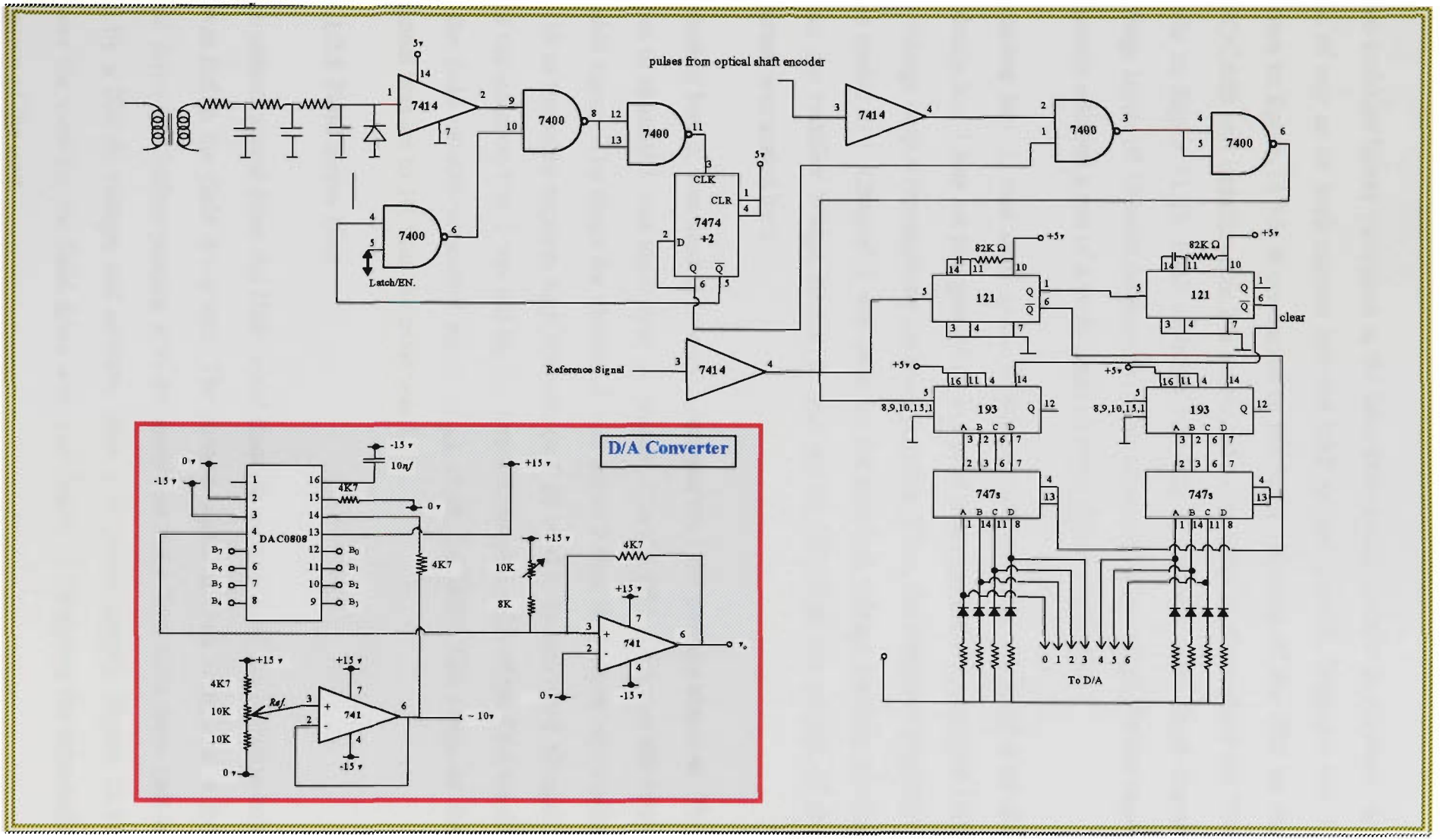


Figure 11.15 counter and D/A circuit layout

11.1.2.4 Isolation box

Two isolation boxes were used in the laboratory setup in order to eliminate the risk of any ac or back current into the DSP or the host PC. Isolation box 1, shown in figure 11.16, is connected to the input channels of the DM on the DPC/C40B and isolation box 2 is connected to the outputs channels of the DM (refer to figure 11.2). Each isolation box has 4 channels with fixed output voltage levels of 10 volts (maximum). The adjustable input range for the input channels allowed a use of a wide range of input signal amplitudes.

Isolation box 1, was used as the Machine/DSP interface. Channel 0 of the isolation box 1 was set to a gain of 10 and used as an input for the counter D/A dc voltage output representing the rotor position. Thus, the output of channel 0 was scaled to 1. Channel 1 was used for the terminal voltage reading coming from the rectifier bridge circuit, it was set to 100, thus the output of the channel was scaled by 0.1.

Isolation box 2, was used to interface between the DSP and the machine. The input to channel 1 was the output of channel 1 in the DM. This was the relay enable signal. The range for channel 1 was set to 1 thus the output was scaled by 10 as the relay requires high voltages ($\approx$ 6 dc *volt*) to be activated. Channel 0 of the isolation box 2 was fed by the output from channel 0 of the DM which is the field excitation control signal (U_{fd}), from the DSP. The range of the channel was set to 10, thus the scale was 1.

11.1.2.5 Field Drive Unit

The control signal from the DSP board through channel 0 of the isolation box 2 was fed to the field drive unit. The control signal is then amplified in the field drive unit before passing it to the generator field. The field drive unit is fed by a 200 dc voltage and operates from a 5*v* power supply. Figure 11.17 shows the view for the field drive unit, and figure 11.18 shows the schematic diagram of the unit.

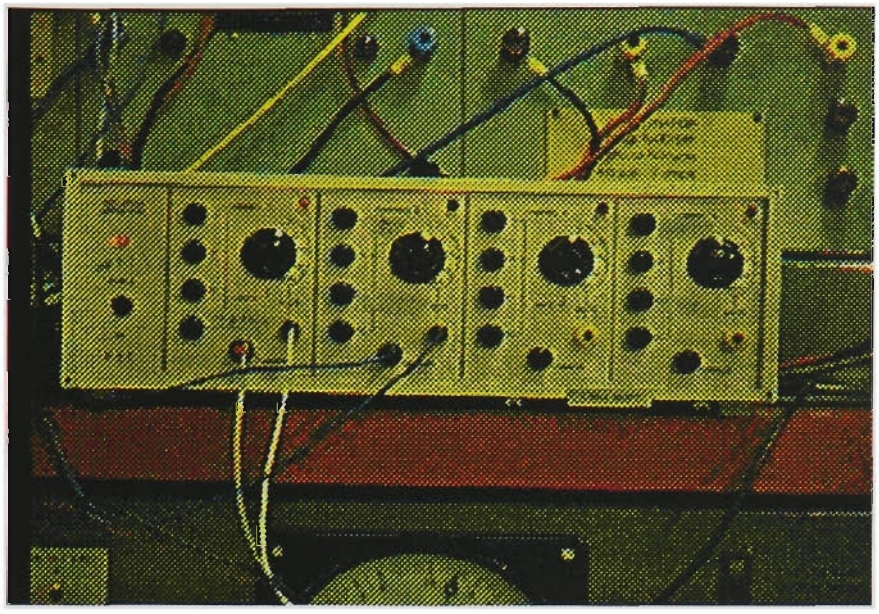


Figure 11.16 the isolation box

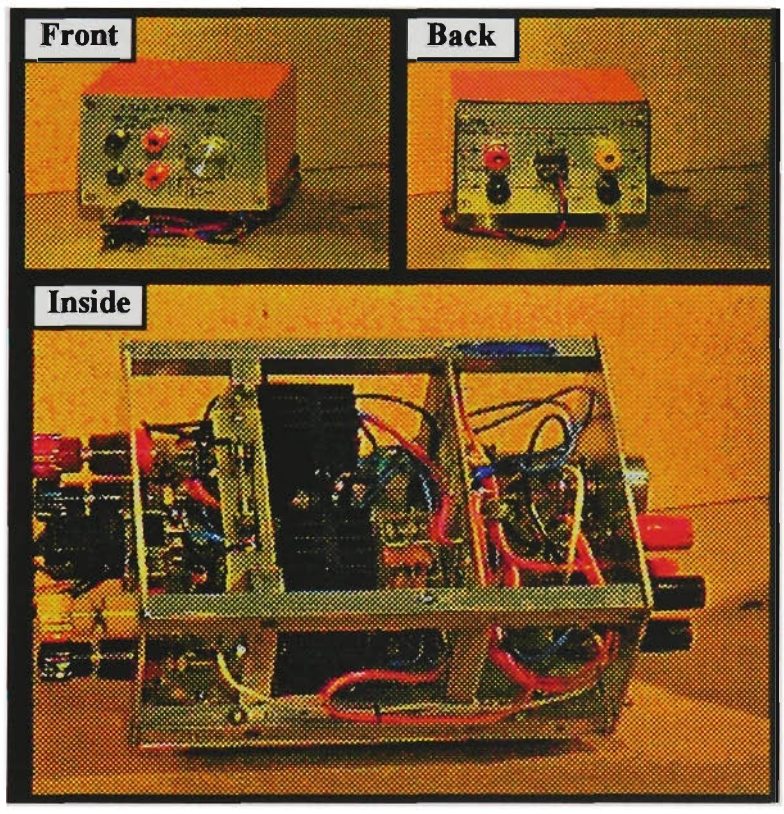


Figure 11.17 the field drive unit

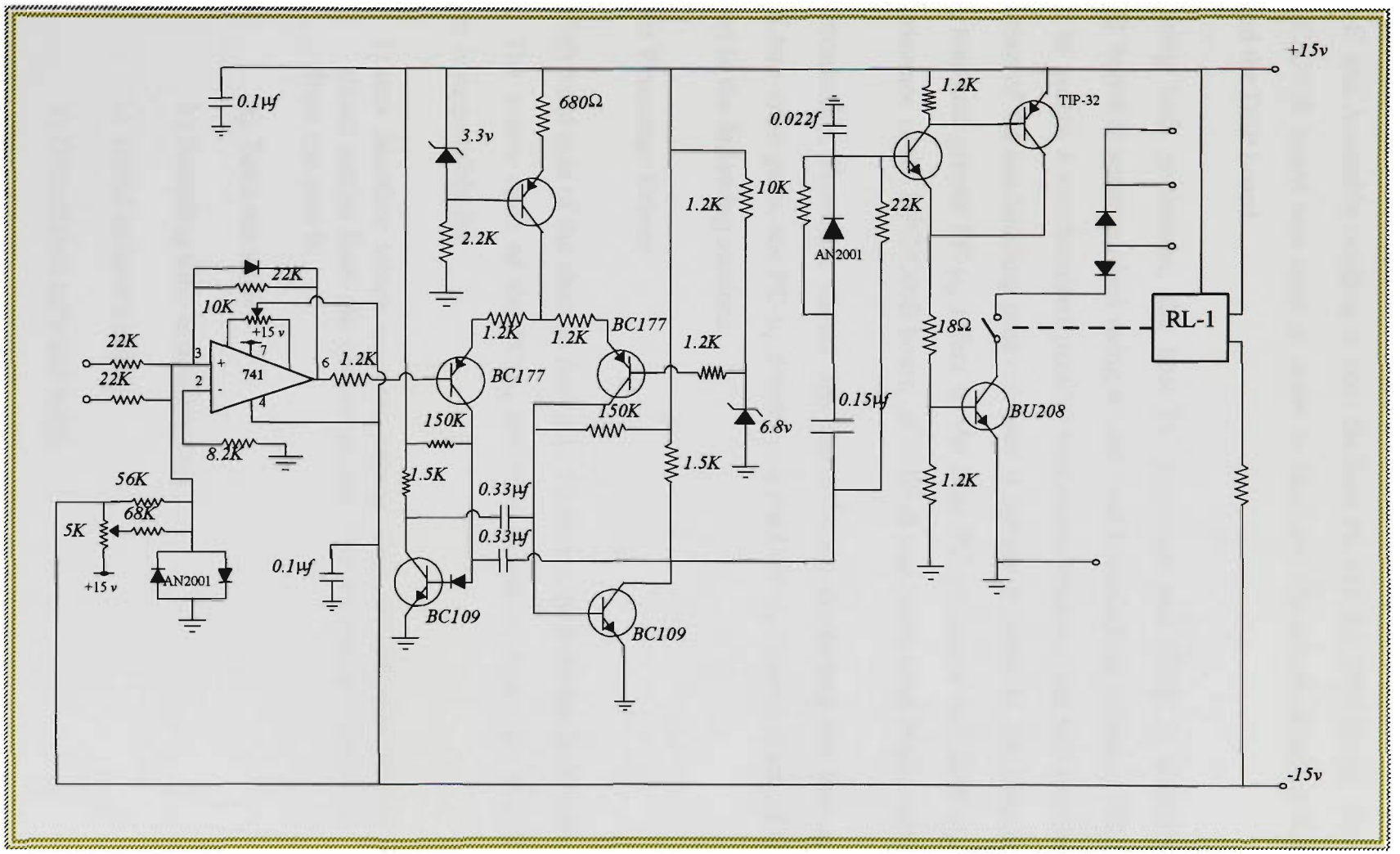


Figure 11.18 Schematic diagram of the field drive unit

11.2 SOFTWARE DEVELOPMENT

The proposed Shay-Exciter was implemented as a software system including programs written in C and Assembly residing in both the host PC and the DSP board. The DPRAM in the DPC/C40B board was used in order to facilitate the communication between the host PC and the DSP board.

Synchronising both processors, the host PC processor and SITE A processor in the DPC/C40B board is accomplished using a restricted handshaking protocol. Several flags were used to secure a synchronised parallel processing between the two processors. The general layout of the handshaking protocol used is shown in figure 11.19. Note that, in the following text and graphs PC-μ_p refers to the host PC processor and DSP-μ_p refers to SITE A processor in the DPC/C40B board, as SITE B was never used in this application.

The synchronisation flowchart shows that the software developed for this application consists of two main parts, the PC-μ_p drivers and the DSP-μ_p drivers. Each of these parts is explained in the following sections.

11.2.1 Host Processor Driver

The left hand side of the chart in figure 11.19 shows most of the details of this part. The source code of the PC-μ_p driver is shown in Appendix B.1. This driver is responsible for:

1) user interface which involves getting the user information via direct entries from the command line. The information required from the user is:

   a) Total run time.

   b) Sampling time in milliseconds.

   c) Initial counter reading.

   d) Disturbance type and value.

   e) Disturbance start and end times.

   f) Data file name for results saving. register receive sample enable reset interrupts.

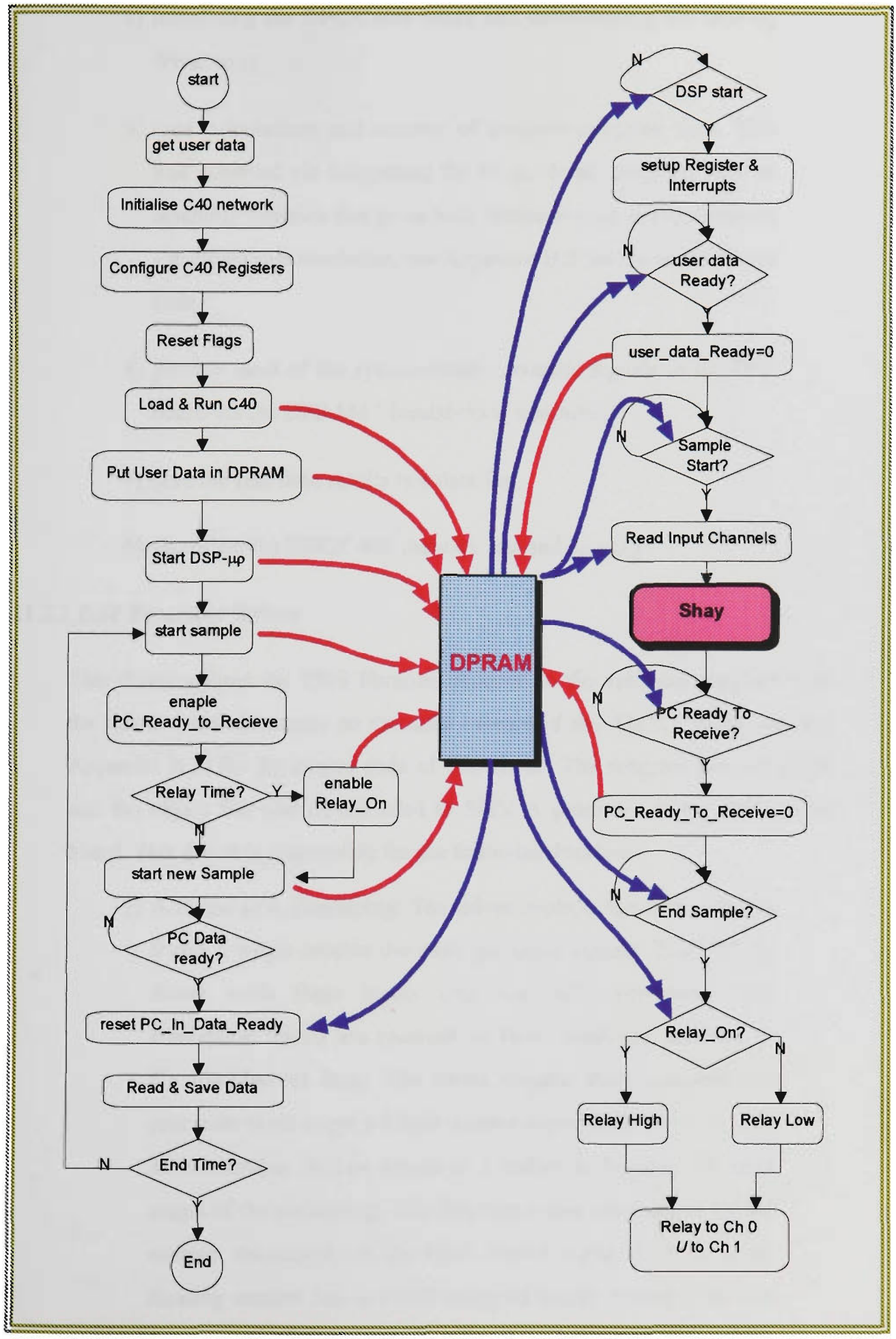


Figure 11.19 PC-μ_p and DSP-μ_p synchronisation flowchart

2) initialising the DPC/C40B board and downloading the DSP-μ_p driver to it,

3) time calculations and security of accurate sampling time. This was achieved via integrating the PC-μ_p driver program with an assembly function that gives time measures with the accuracy of 1 milliseconds resolution, see Appendix B.2 for the timer source code,

4) provide most of the synchronisation control signals to the DSP board via the DPRAM ' handshaking controller",

5) save the real time results in a data file,

6) shutdown the DPC/C40B network and end running.

11.2.2 DSP Processor Driver

This driver utilises the TMS libraries included in the software supplied with the board, which is simply an extended version of the ANSI-C language (see Appendix B.3) for the source code of this driver. The program was compiled and the object file was downloaded to SITE A processor in the DPC/C40B board. This driver is responsible for the following functions:

1) A/D and D/A interfacing: The driver controls the DM channels 0 and 1 which receive the analogue input signals. The DSP-μ_p driver reads these inputs after the A/D conversion. The conversion results are received as 16-bit unsigned numbers in 2's complement form. The driver negates these numbers and processes them to get a 32-bit number scaled from -3 to +3. The 32-bit number is then stored in a buffer to be used for later stages of the processing. This function is also responsible for the reverse conversion of the final control signal from a 32-bit floating number into a 15-bit unsigned integer scaled from 0 to 3 volts and to set or reset the most significant bit (MSB) of the

16-bit number to zero or 1 according to the sign. The output, 16-bit unsigned number, is then stored in the output buffer to be sent out to the machine and then to the relay via output channels 0 and 1 in the DM,

2) implementing the Shay-Exciter algorithm: The inputs from the analogue channels and the user inputs, are processed and used to implement the digital form of the Shay-FLC used as the exciter controller. The Shay function is explained in details in section 11.2.3,

3) passing the real time data to the PC-μ_p via the DPRAM,

4) activating the relay when required.

This driver uses the interrupt and I/O flag 1 (IIOF1). The main processing is done in the interrupt service routine This routine is merged with the main functions in the flowchart in figure 11.19 for readability reasons.

11.2.3 Shay Module

The service routine in the DSP-μ_p driver calls the Shay module (see Appendix B.4 for the source code). The Shay module consists of three main parts:

1) functions drivers,

2) general purpose fuzzy processing functions,

3) subfunctions.

This function fully implements the Shay algorithm presented earlier, in section 2. The function flowchart is shown in figure 11.20. This function and its subfunctions represent the core of the whole software system developed in this work. Both the FLC drivers and the general purpose fuzzy processing functions are fully explained in the next section. The subfunctions are the functions that perform the operational angle and the sector definition to support the Shay main functions. Refer to Appendix B.5 for the subfunctions source code.

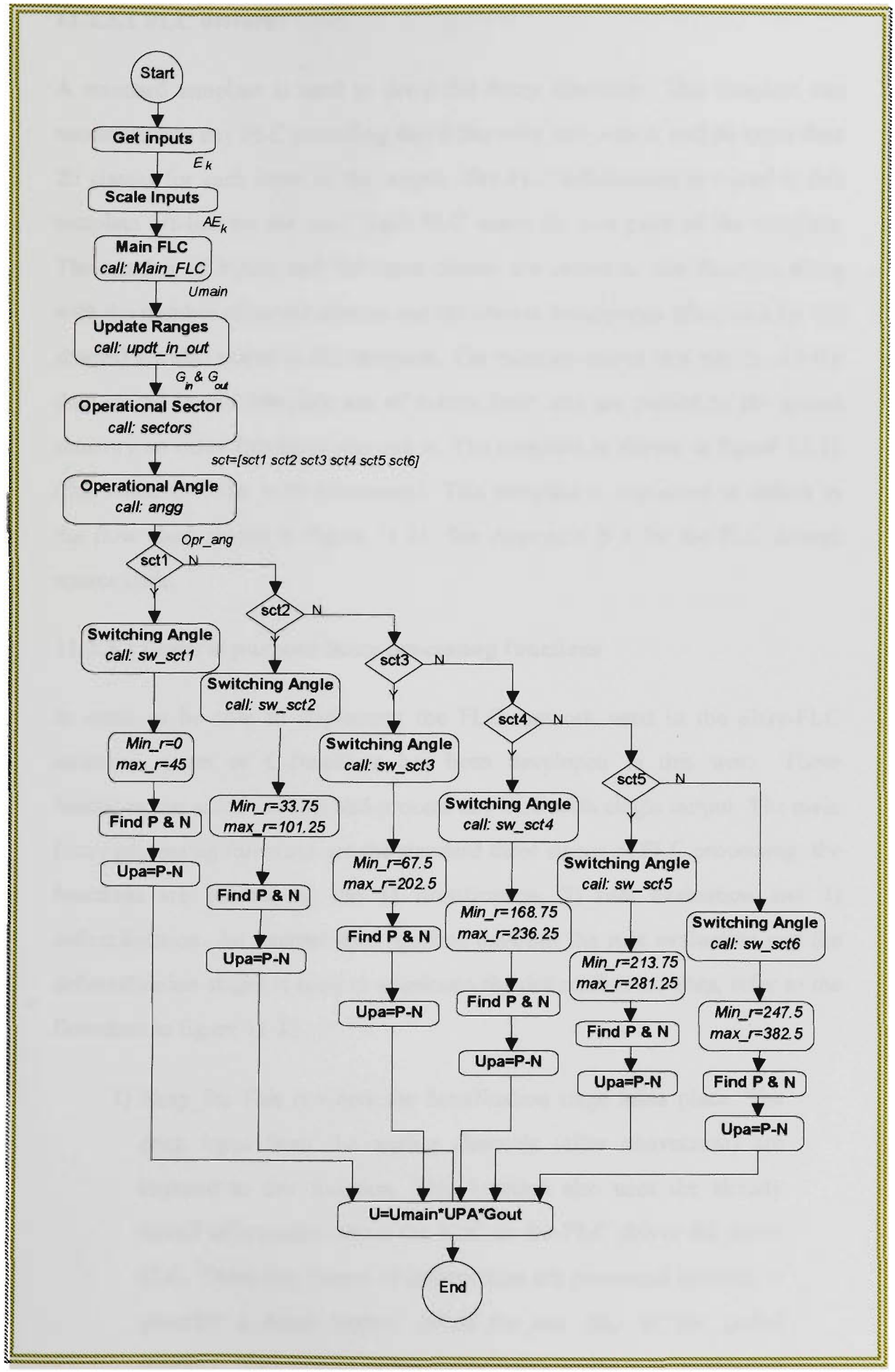


Figure 11.20 Shay function flowchart

11.2.3.1 FLC drivers

A standard template is used to drive the fuzzy functions. This template can accommodate any FLC providing that it has only one output, and no more than 20 classes for each input or the output. The FLC information is stored in this template off-line by the user. Each FLC needs its own page of the template. The number of inputs and the input classes are stored in this function along with the number of output classes and the classes descriptions (formulas for the shapes) are also stored in this template. The rules are stored in a matrix. All the data stored in this template are of matrix form and are passed to the global memory so other functions can use it. The template is shown in figure 11.21 (the actual C-Code, with comments). This template is explained in details in the flow chart shown in figure 11.21. See Appendix B.6 for the FLC drivers source code.

11.2.3.2 General purpose fuzzy processing functions

In order to be able to implement the FLC network used in the Shay-FLC structure, a set of C-functions has been developed in this work. These functions can accommodate and process any FLC with single output. The main fuzzy processing functions are the standard three stages of FLC processing, the functions are performing the 1) fuzzification, 2) rule evaluation and 3) defuzzification. An intermediate function between the rule evaluation and the defuzzification stages is used to accelerate the defuzzification step, refer to the flowchart in figure 11.22.

1) **Shay_fz:** This is where the fuzzification stage takes place. The crisp input from the analog channels (after conversion) are imputed to this function. This function also uses the already stored information about the FLC in the FLC driver for every FLC. These two pieces of information are processed in order to generate a fuzzy matrix called *fuz_mat* (fz_k) in the global memory. This matrix is the fuzzy input to the rule evaluation stage. Shay_fz uses the L/R COG fuzzification in the production

of the *fuz_mat* matrix. See Appendix B.7 for the source code of Shay_fz.

2) **Shay_rl:** The fuzzy input from **Shay_fz** as well as the rule matrix from the FLC driver function are both used in order to generate a matrix called *rules_mat, rl*. The *rules_mat* represents the fired rules, the corresponding output classes and the rules truth values in the function and is stored in the global memory. This function implements the inference for a Mamdani type rules and uses the min operator as well as the L/R COG inference. See Appendix B.8 for the source code of Shay_rl.

3) **Shay_dbr:** This is an intermediate stage between the rules evaluation and the defuzzification. It is used here in order to provide only the necessary information to the defuzzification stage. Shay_bdr does some pre calculations to help accelerating the defuzzification function. In other words, this function organises and provides the *rules_mat* matrix in a refined form to the defuzzification stage. The output of this function is the matrix *bdr_mat* in the global memory. The source code of this function is given in Appendix B.9.

4) **Shay_dff:** This function implements both the standard center of gravity (COG) and the L/R COG defuzzification methods in order to produce the final crisp output from the FLC. The inputs to this function is the *bdr_mat*. It also uses the already provided information (regarding the output classes) from the FLC driver. See Appendix B.10 for the source code of Shay_dff.

```
float FLC_Template(float Input_1,float Input_2)
        {
        float i,u;
        int No_Input= ;                                  /* number of inputs to FLC*/
/******* INPUT 1 definitions *******/
        int No_In_1=5;                                   /* number of input 1 classes*/
        float Max_In_1=56.25;                            /*maximum range of input 1*/
        float Min_In_1=11.25;                            /*minimum  range of input 1*/
        float     In_1[6][4]={                           /*1st row==>No. of input 2 classes*/
                        define the classes here
                        };
/******* INPUT 2 definitions *******/

        int No_In_2= ;                                   /* number of input 2 classes*/
        float Max_In_2= ;                                /*maximum range of input 2*/
        float Min_In_2= ;                                /*minimum range of input 2*/
        float     In_2[6][4]={                           /*1st row==>No. of input 2 classes*/
                        define the classes here
                        };
/******* OUTPUT definitions *******/
        int No_Out=14;                                   /*number of output classes*/
        float Out_mat[8][3]={                            /*1st row==>No. of output classes*/
                        define the classes here
                        };
/******* Rules definitions *******/
        int Rules_m[6][20]={                             /*Rules_m[0][0]==>No_input_1,*/
                        define the rules here            /*Rules_m[0][1]==>No_input_2,*/
                        };                               /*Rules_m[0][2]==>no of output classes*/
/******* Scaling Input 1 *******/
        if (Input_1>Max_In_1)
                Input_1=Max_In_1;
        if (Input_1<Min_In_1)
                Input_1=Min_In_1;
        for (i=1;i<No_In_1+1;i++)
                {
                In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                }
/******* Scaling Input 2 *******/
        if (Input_2>Max_In_2)
                Input_2=Max_In_2;
        if (Input_2<Min_In_2)
                Input_2=Min_In_2;
        for (i=1;i<No_In_2+1;i++)
                {
                In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                }
/******* Fuzzification *******/
        if (No_Input==1)                                 /*single input*/
                {
                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                }
        if (No_Input==2)                                 /* two inputs */
                {
                        shay_fuz(Input_1,In_1,Input_2,In_2,2);
                }
/******* Rule Evaluation *******/
                shay_rl(fuzzy_mat,Rules_m);
/******* Borders *******/
                shay_bdr(rules_mat,Out_mat);
/******* defuzzification *******/
                U=shay_dff(bdr_mat,Out_mat);

        return(U); /*return the crisp output*/

        } /* end of Template */
```

Figure 11.21 FLC drivers template

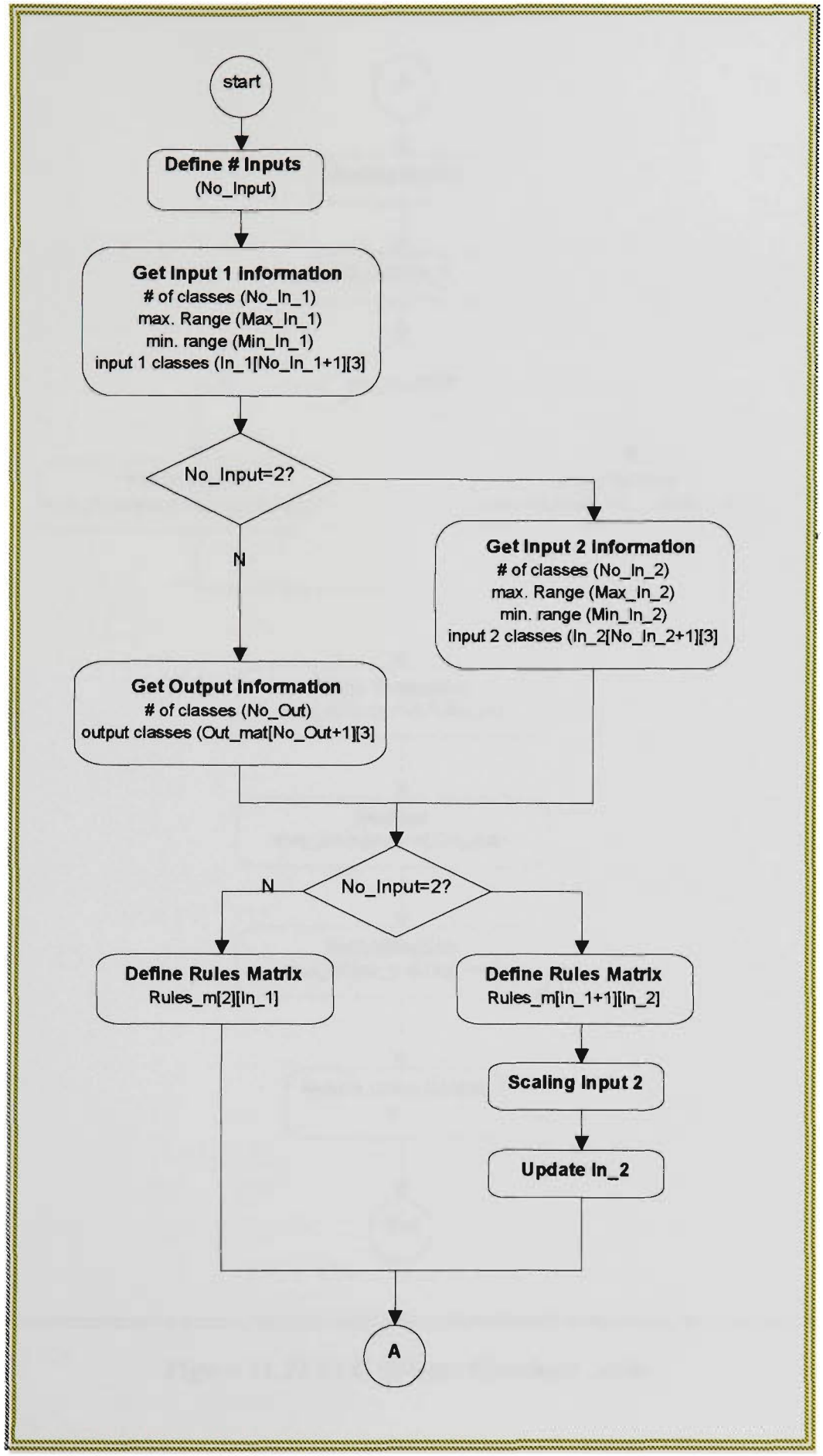


Figure 11.22 FLC drivers flowchart.....cont.

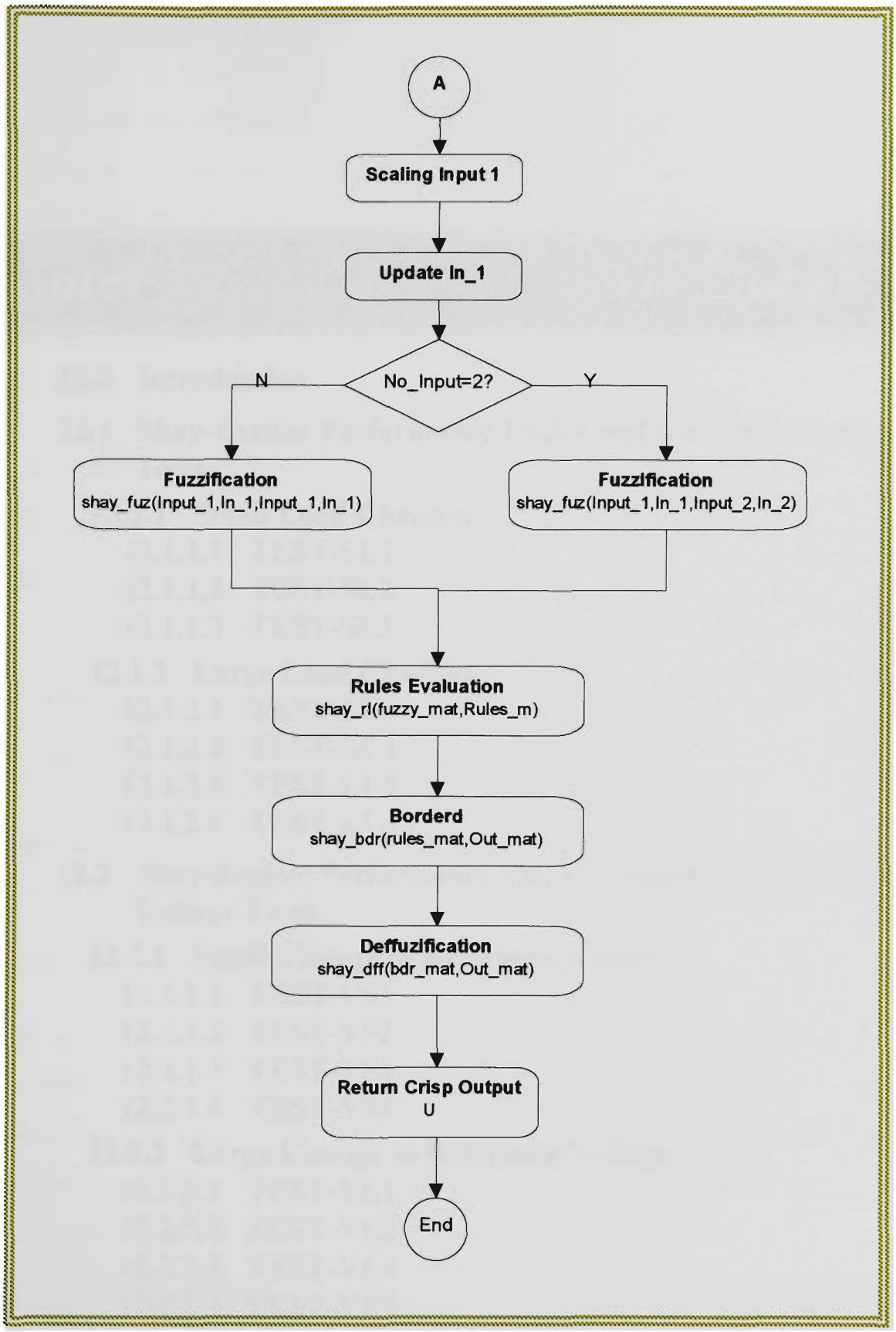


Figure 11.22 FLC drivers flowchart ..cont.

Chapter 12 *Laboratory Tests and Results*

12.0 INTRODUCTION

The robustness and efficiency of the proposed Shay adaptive FLC structure and its applicability to the excitation control is verified in this chapter. Several tests were conducted using the laboratory setup mentioned earlier in chapter 11. A wide range of operating conditions and different types of tests have been performed.

The tests are categorised under three main categories according to the perturbation case studied. These categories are:

1) sudden load changes,
2) reference voltage changes,
3) transmission line short circuits.

12.1. SHAY-EXCITER PERFORMANCE TESTS UNDER SUDDEN LOAD CHANGES TESTS

In these tests a sudden load change in the generator loading was introduced. Two types of sudden load changes have been tested, small load change and large load change.

12.1.1 Small Load Change

In these tests the synchronous generator was subject to 30% of rated load balanced resistive sudden load change. The load was sustained for five seconds and then removed. This test was conducted while the generator is operating under three different operation conditions. Table 12.1 describes the tests and the operation conditions.

Test	Operating conditions		Disturbance	Duration
	Active Power (*pu*)	Power Factor		
TEST-SL1	0.9	0.9 (*lag*)	30% resistive load	5 *sec*
TEST-SL2	0.5	0.6 (*lag*)	30% resistive load	5 *sec*
TEST-SL3	0.4	0.98	30% resistive load	5 *sec*

Table 12.1 sudden small load changes testing conditions

12.1.1.1 TEST-SL1

Figures 12.1-12.4 show the result of TEST-SL1. The terminal voltage plot in figure 12.1 shows a small overshoot (less than 2.5%) and a zero steady-state error. Figure 12.2 shows that the rotor angle was stabilised after 1.5 seconds of the disturbance. The rotor angle stability is clear from the $\Delta\omega$ plot in figure 12.3. Figure 12.4 shows the excitation control signal.

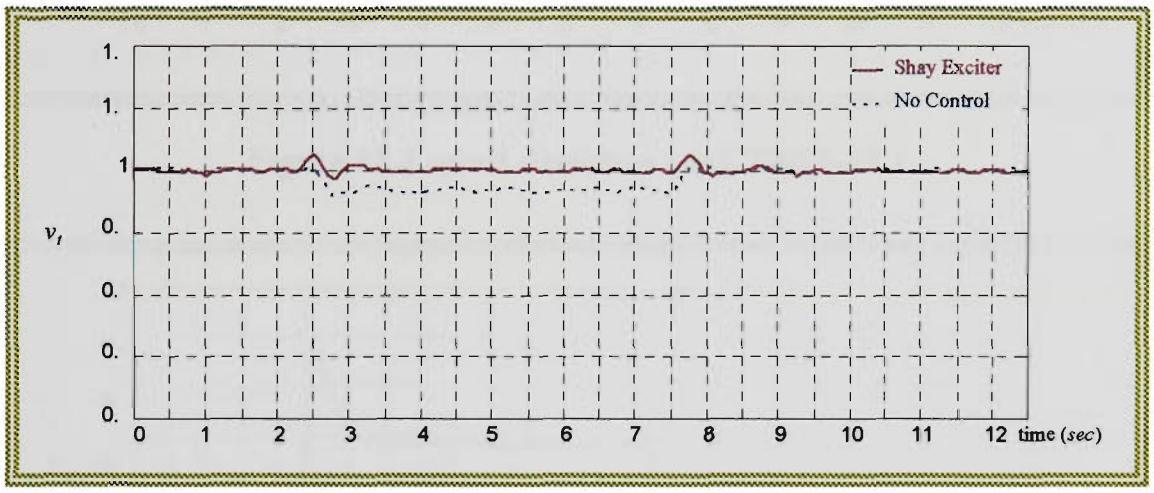


Figure 12.1 terminal voltage (*pu*), TEST-SL1

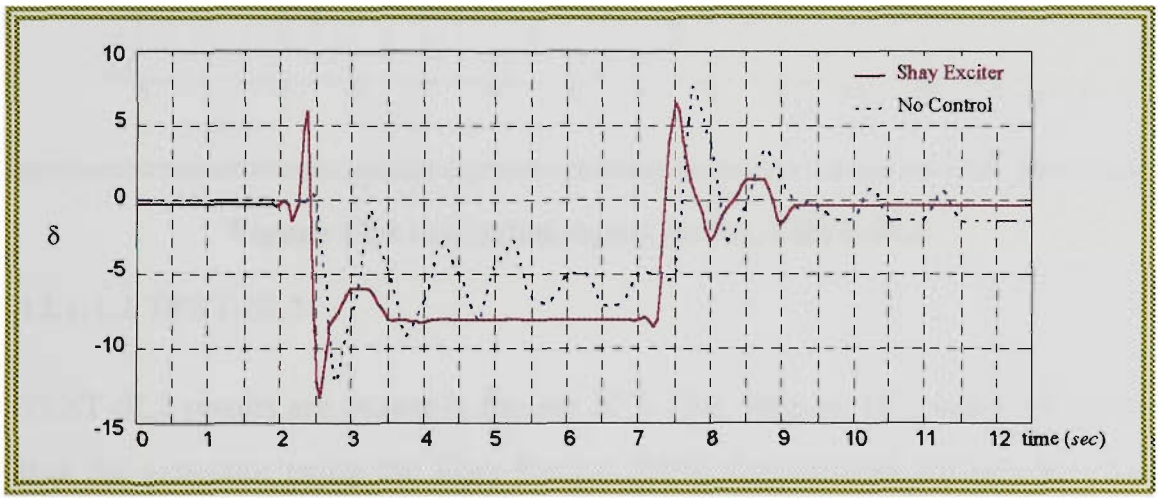


Figure 12.2 rotor angle (electrical degrees), TEST- SL1

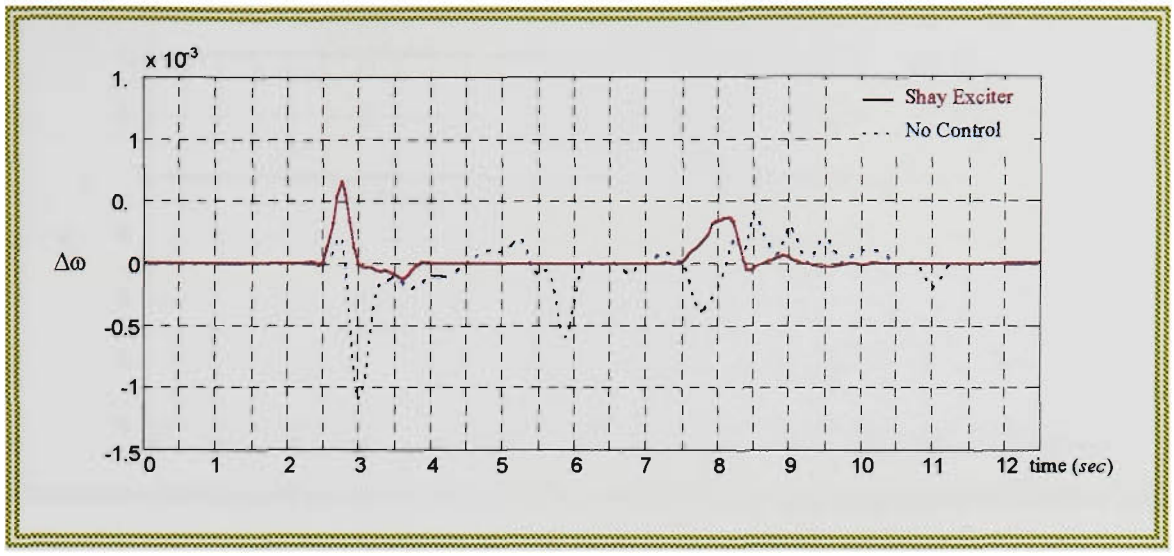


Figure 12.3 speed deviation (*pu*), TEST-SL1

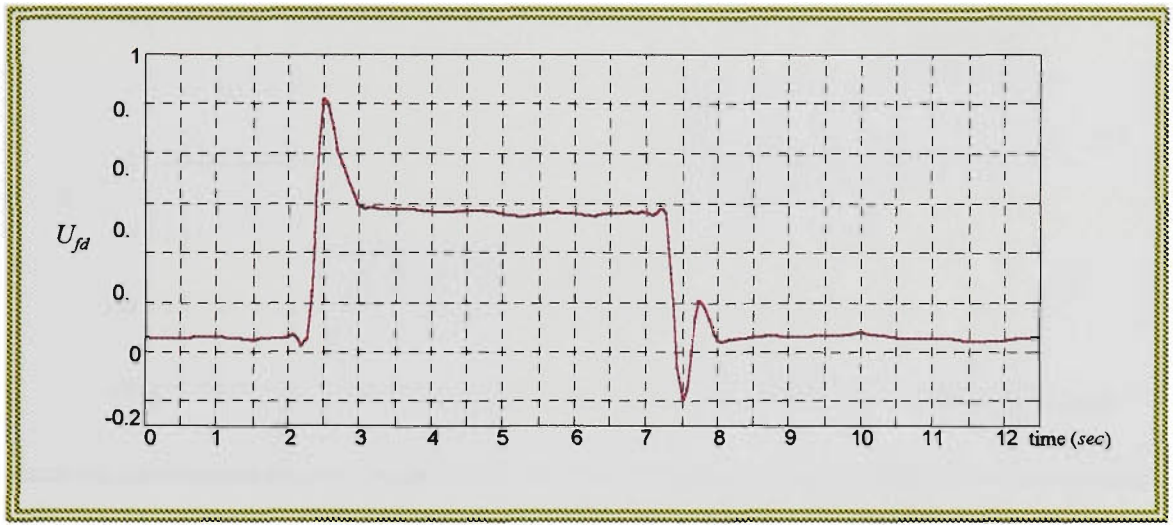


Figure 12.4 excitation signal (*volts*), TEST-SL1

12.1.1.2 TEST-SL2

TEST-SL2 results are shown in figures 12.5-12.8. Figures 12.5 and 12.6 show that the generator using the Shay-Exciter fulfils the terminal voltage and the rotor angle stability requirements in 1 and 2 seconds respectively. The speed deviation and the excitation signal are shown in figures 12.7 and 12.8.

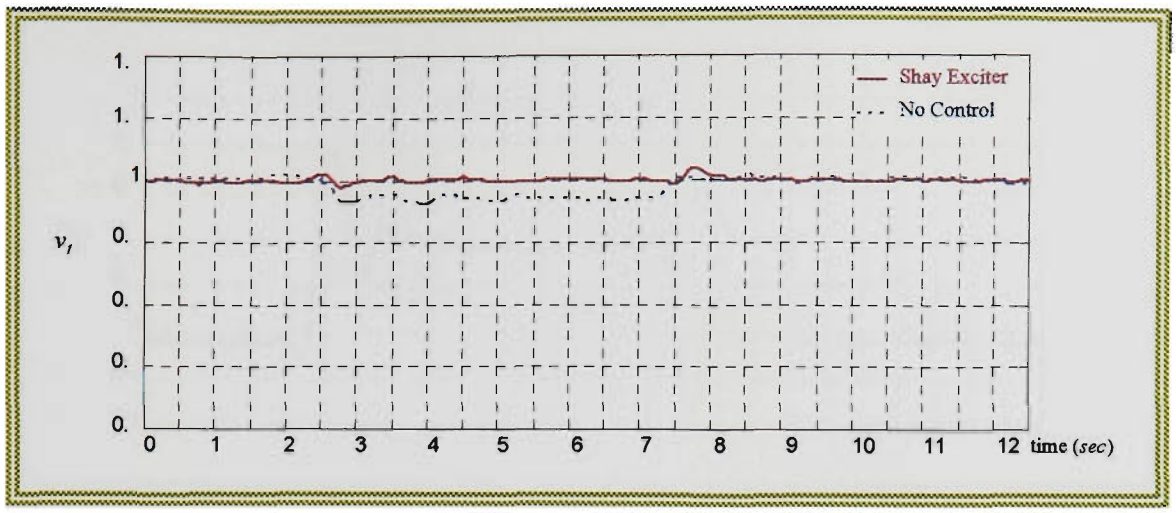


Figure 12.5 terminal voltage (*pu*), TEST-SL2

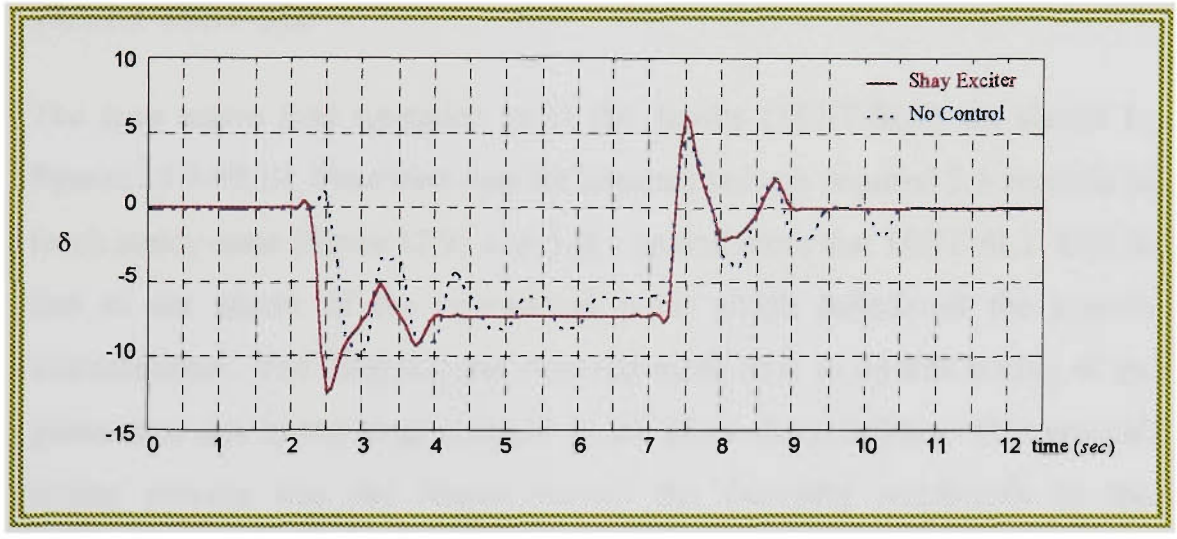


Figure 12.6 rotor angle (electrical degrees), TEST- SL2

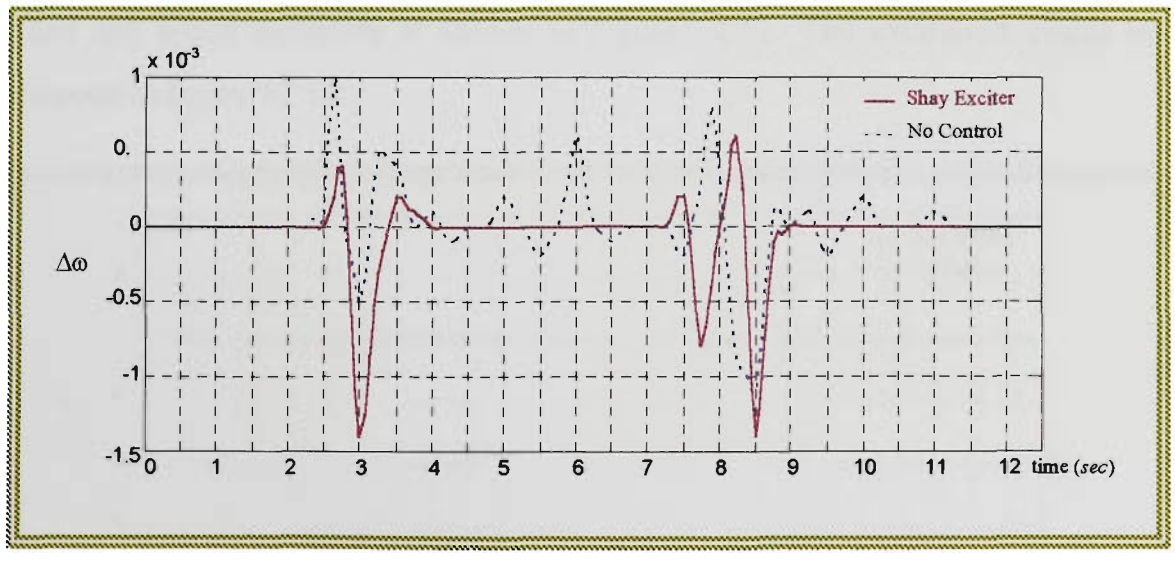


Figure 12.7 speed deviation (*pu*), TEST-SL2

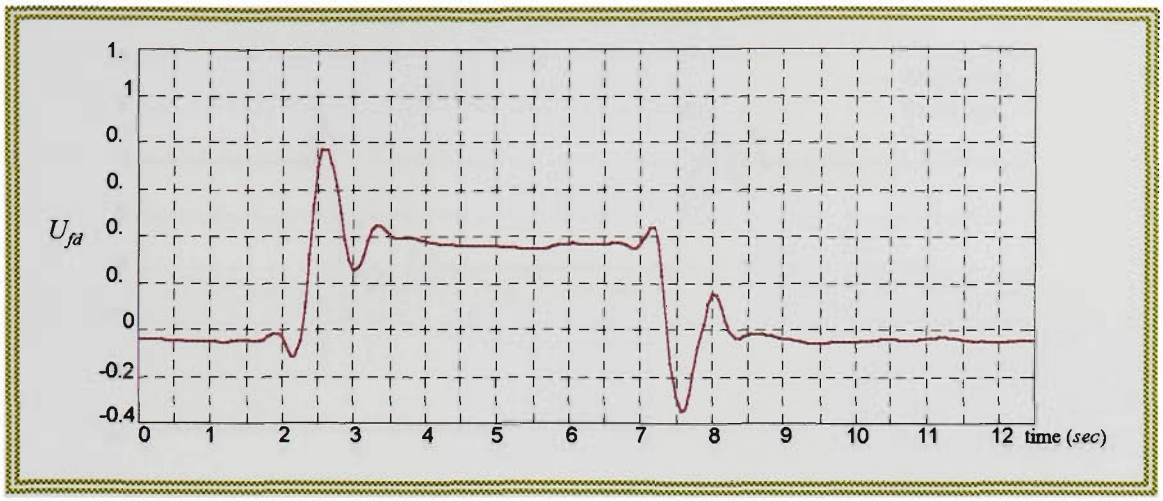


Figure 12.8 excitation signal (*volts*), TEST-SL2

12.1.1.3 TEST-SL3

The pure active load operation point test results (TEST-SL3) are shown in figures 12.9-12.12. Note here that the terminal voltage required 2.5 seconds to reach steady-state (figure 12.9) which is 1 second more that TEST-SL1. This is due to the nature of the operational point which influenced the system characteristics. The Shay-Exciter required more time in on-line tuning of its parameters due to the large changes in the plant characteristics. This gradual tuning process was the reason behind the decaying amplitudes in the oscillations shown in the figures. Note that the oscillations amplitudes became much smaller when the load was released. Figure 12.10 shows the rotor angle and the speed deviation is shown in figure 12.11. The excitation signal is shown in figure 12.12.

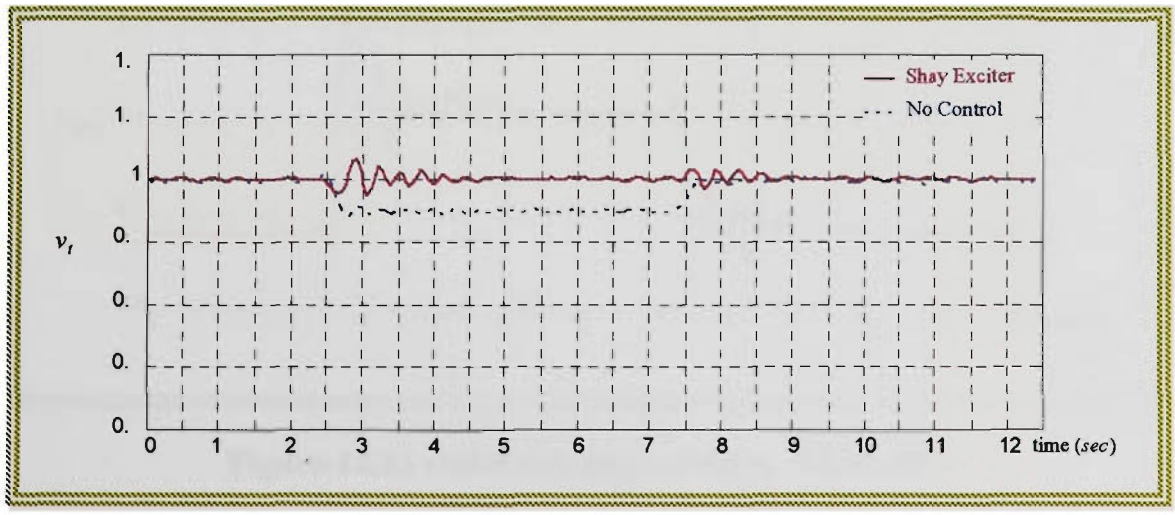


Figure 12.9 terminal voltage (*pu*), TEST-SL3

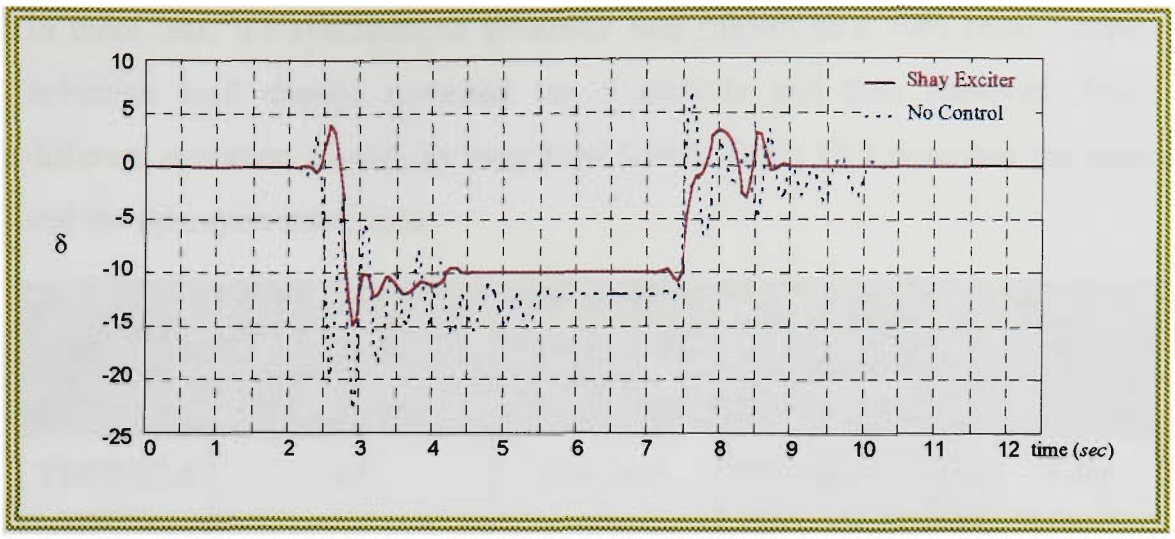


Figure 12.10 rotor angle (electrical degrees), TEST- SL3

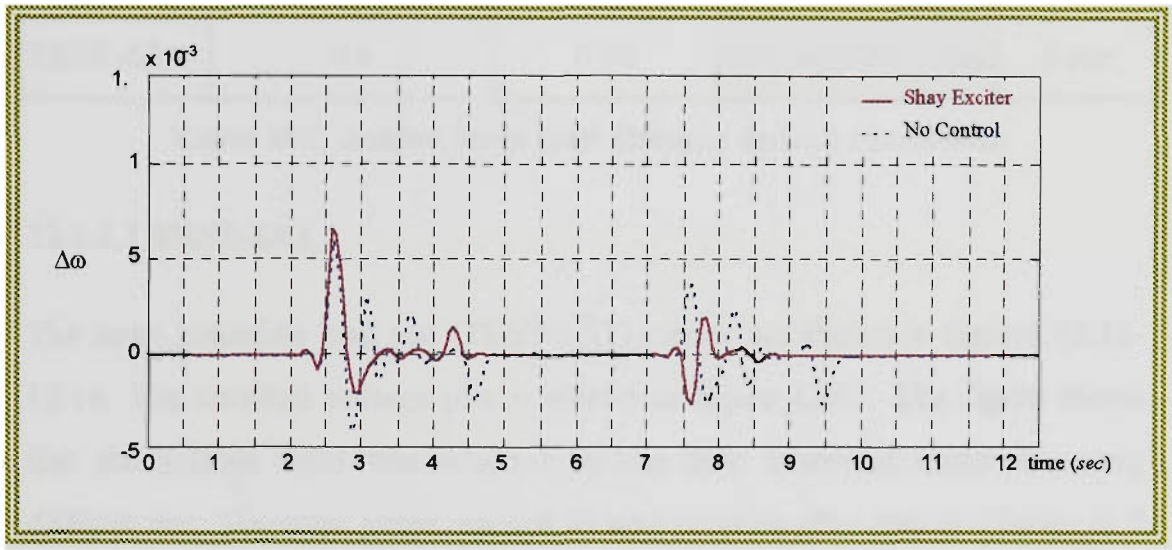


Figure 12.11 speed deviation (*pu*), TEST-SL3

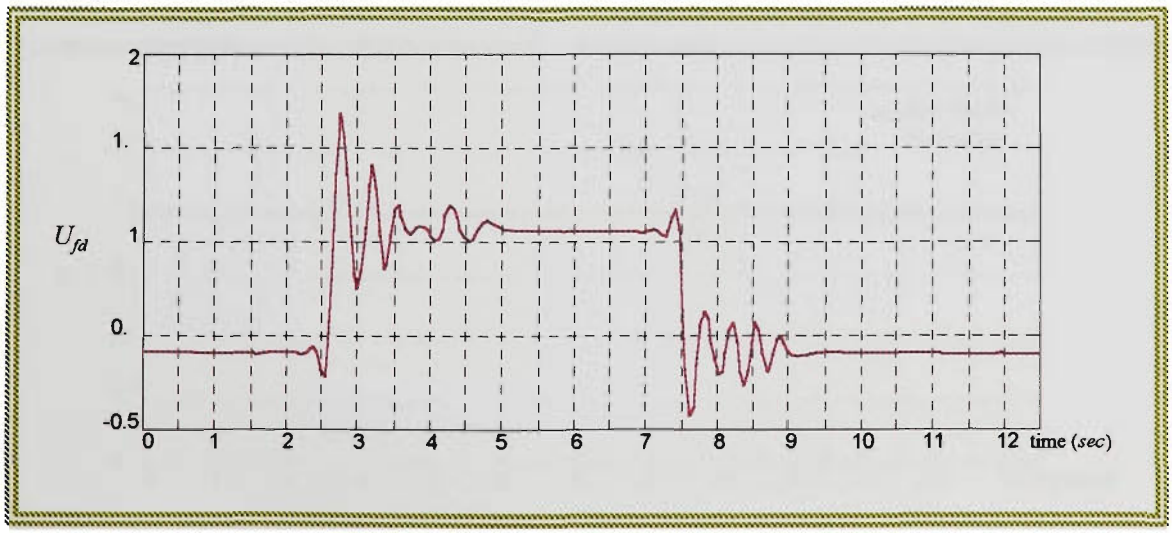


Figure 12.12 excitation signal (volts), TEST-SL3

12.1.2 Large Load Changes

In these tests the synchronous generator was subject to a 70% rated sudden inductive load change sustained for 5 seconds and then removed. Four different operation conditions have been tested. Table 12.2 describes the tests and the operation conditions.

Test	Operating conditions		Disturbance	Duration
	Active power (*pu*)	Power Factor		
TEST-LL1	0.9	0.9 (*lag*)	70% inductive load	5 *sec*
TEST-LL2	0.5	0.6 (*lag*)	70% inductive load	5 *sec*
TEST-LL3	0.4	0.98	70% inductive load	5 *sec*
TEST-LL4	0.6	0.98	70% inductive load	5 *sec*

Table 12.2 sudden large load changes testing conditions

12.1.2.1 TEST-LL1

The large inductive load test (TEST-LL1) results are shown in figures 12.13-12.16. The terminal voltage plot is shown in figure 12.13. The figure shows that the voltage level was retained in less than 1 second under this very difficult test. The rotor angle reached its new position after one oscillation in 2 seconds as shown in figure 12.14. The speed deviation is shown in figure 12.15 and the excitation signal is shown in figure 12.16.

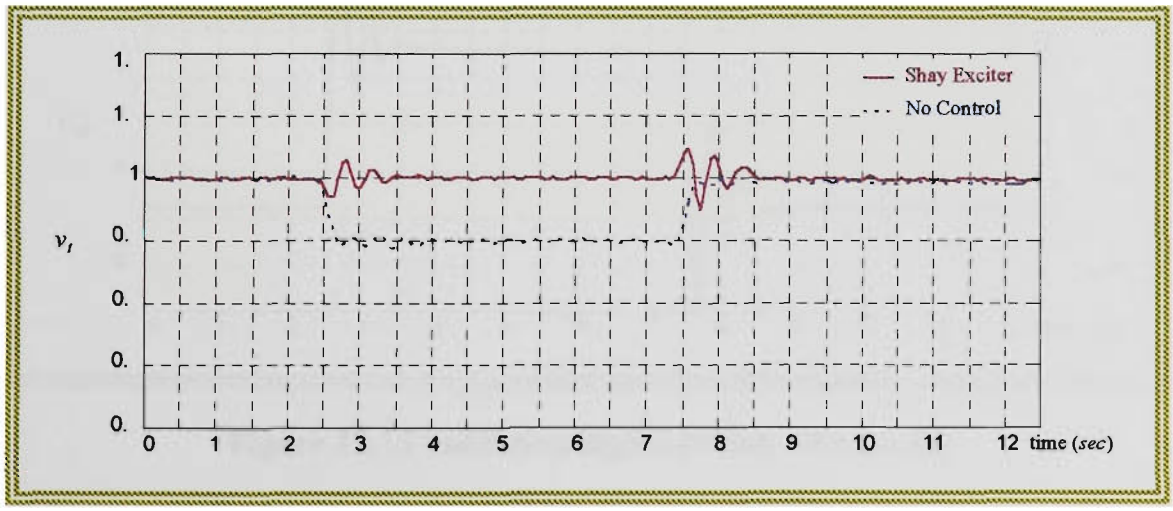


Figure 12.13 terminal voltage (*pu*), TEST-LL1

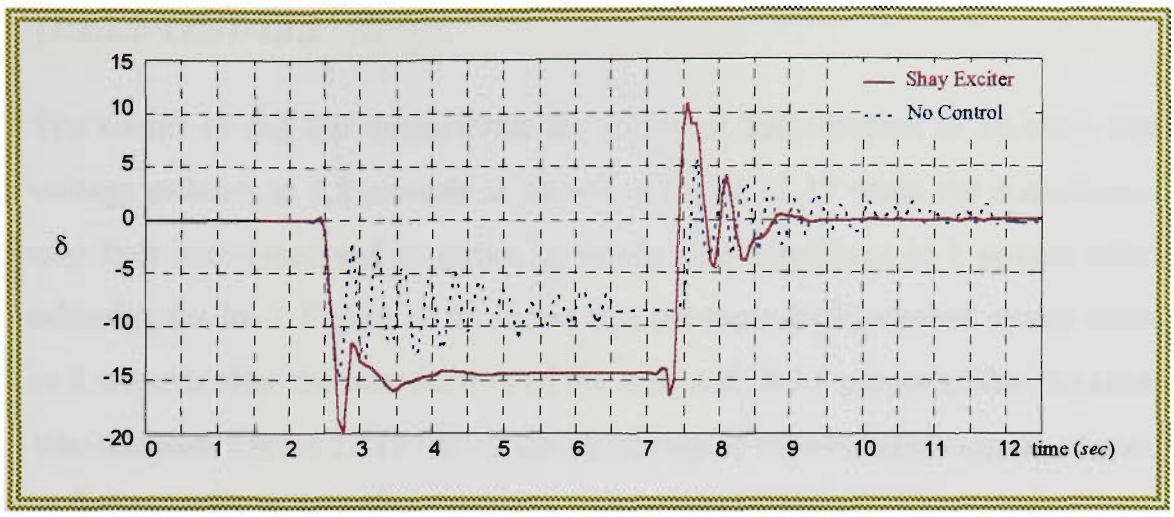


Figure 12.14 rotor angle (electrical degrees), TEST- LL1

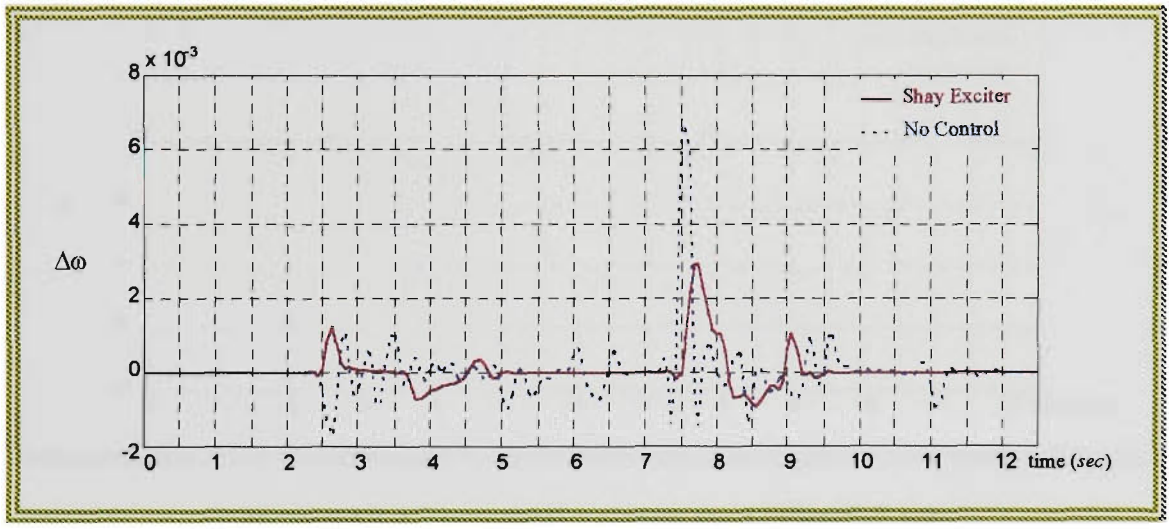


Figure 12.15 speed deviation (*pu*), TEST-LL1

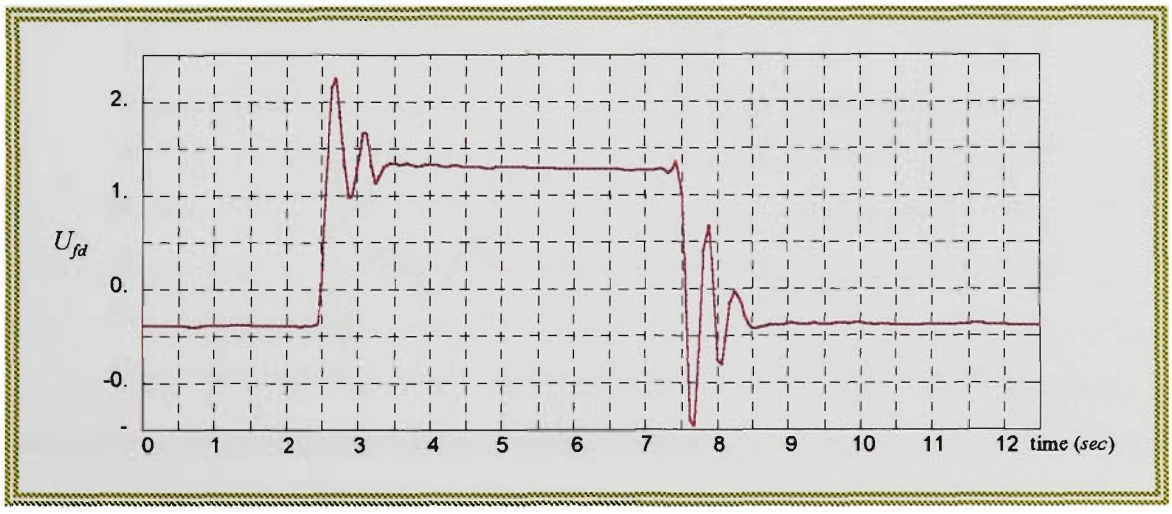


Figure 12.16 excitation signal (*volts*), TEST-LL1

12.1.2.2 TEST-LL2

The results of this test demonstrate the ability of Shay-Exciter to maintain the voltage stability in 1.5 seconds as shown in figure 12.17 when the disturbance was first introduced and to return to steady sate conditions in 1 second after releasing the load. Figure 12.18 shows that the rotor angle reached steady-state in 3 seconds after the introduction of the load and in 1.5 seconds after the load was released. Figure 12.19 shows the efficiency of the excitation signal, shown in figure 12.20, in dumping down the oscillations in $\Delta\omega$.

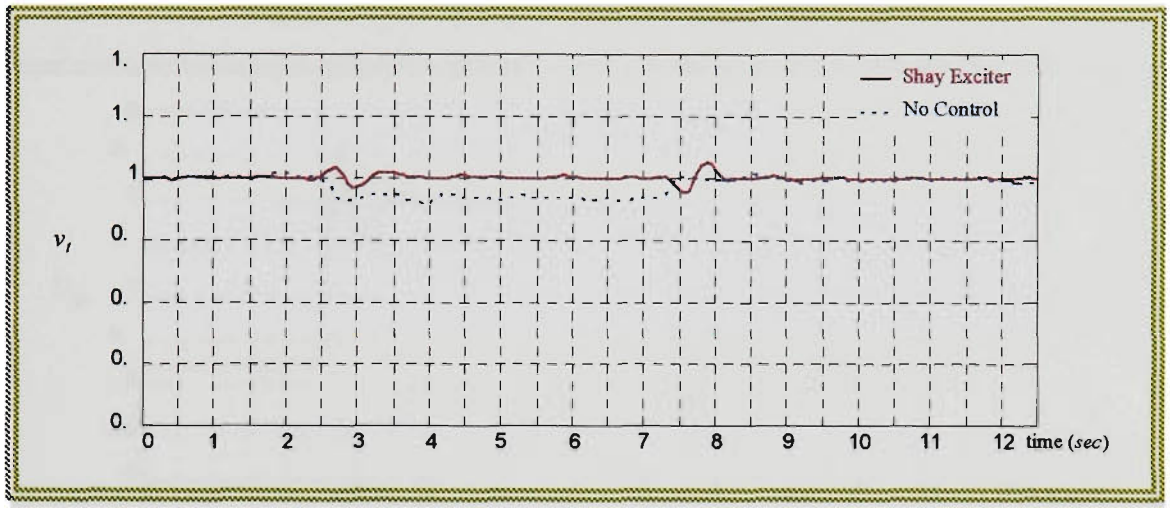


Figure 12.17 terminal voltage (*pu*), TEST-LL2

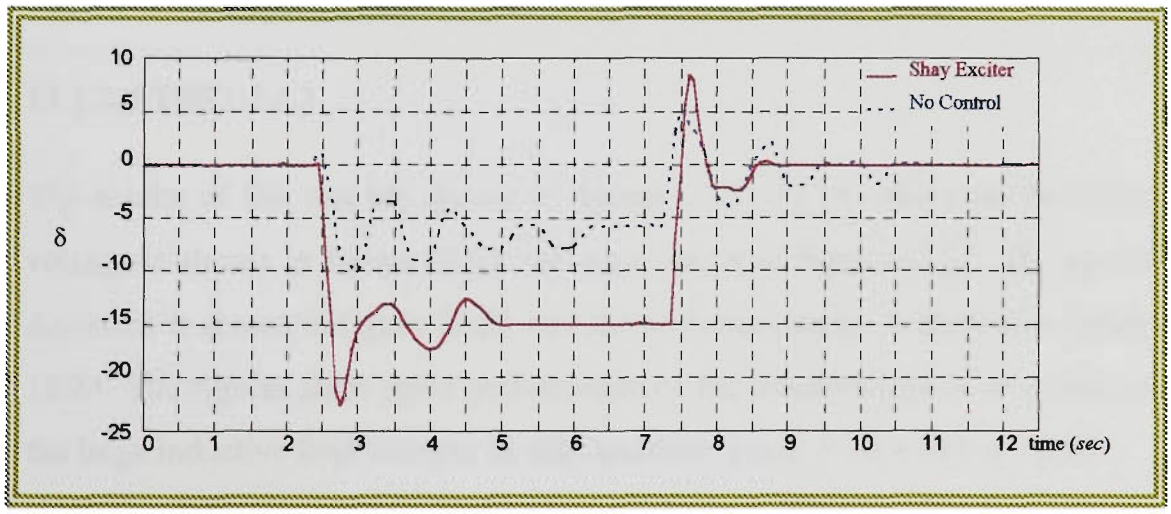


Figure 12.18 rotor angle (electrical degrees), TEST- LL2

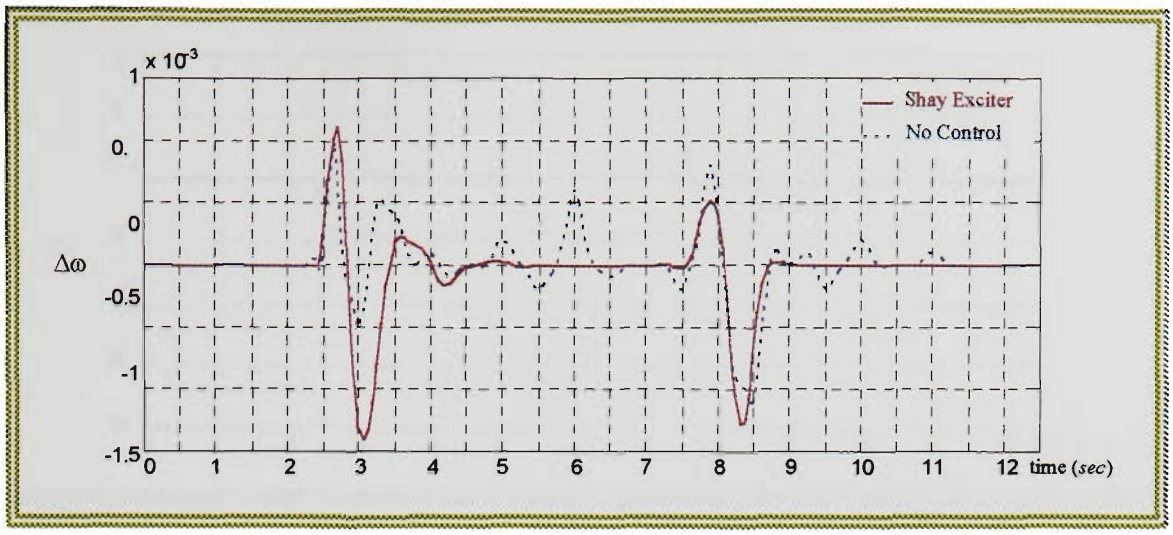


Figure 12.19 speed deviation (*pu*), TEST-LL2

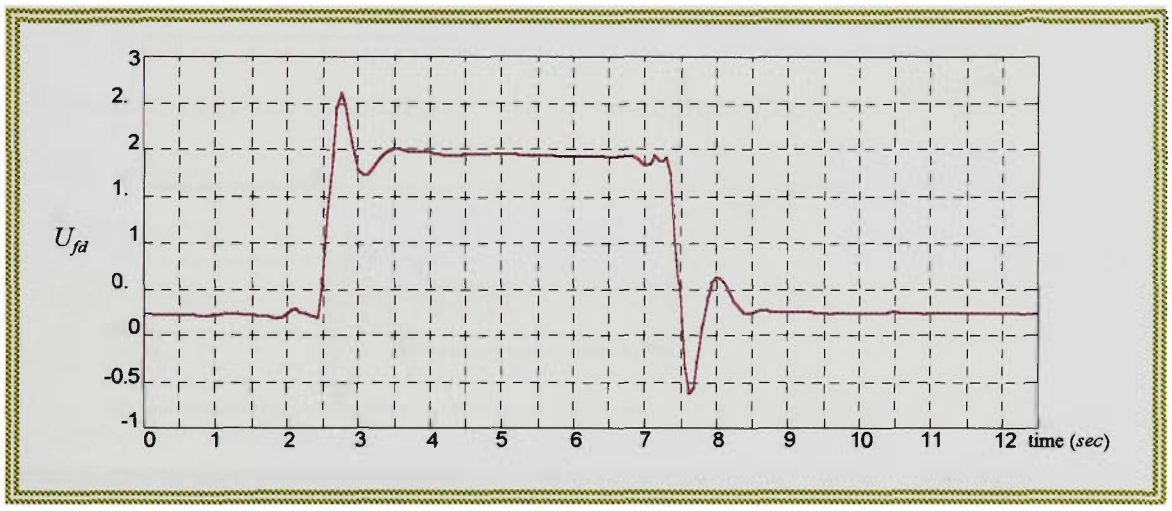


Figure 12.20 excitation signal (*volts*), TEST-LL2

12.1.2.3 TEST-LL3

The results of this test are shown in figures 12.21-12.24, where the terminal voltage is shown in figure 12.21, the rotor angle in figure 12.22. The speed deviation is shown in figure 12.23 and the excitation signal is shown in figure 12.24. The figures show good performance of the Shay-Exciter in response to the large inductive load changes in this operation point, $P=0.4$ and $pf=0.98$.

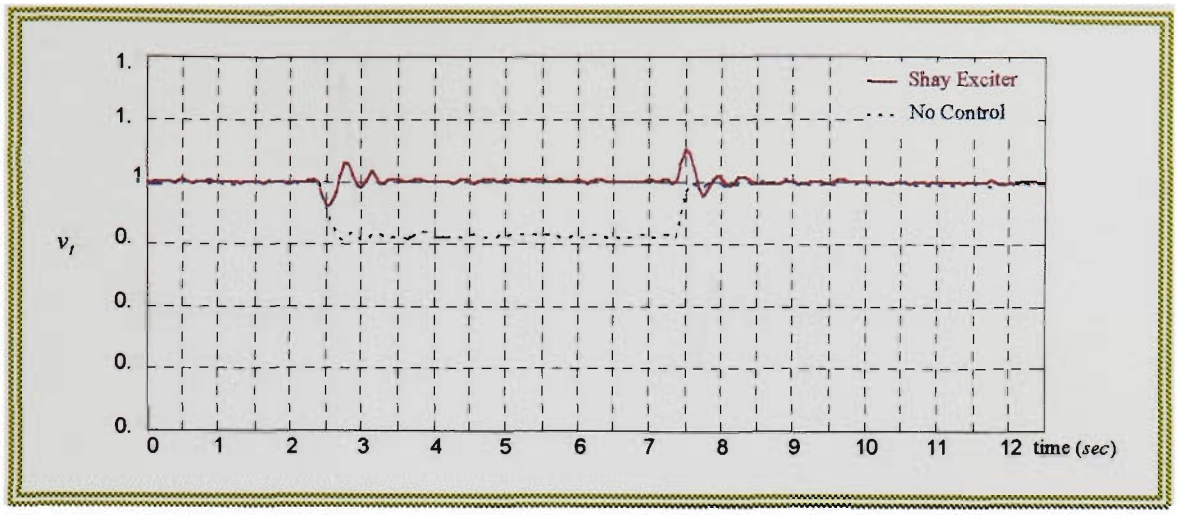


Figure 12. 21 terminal voltage (*pu*), TEST-LL3

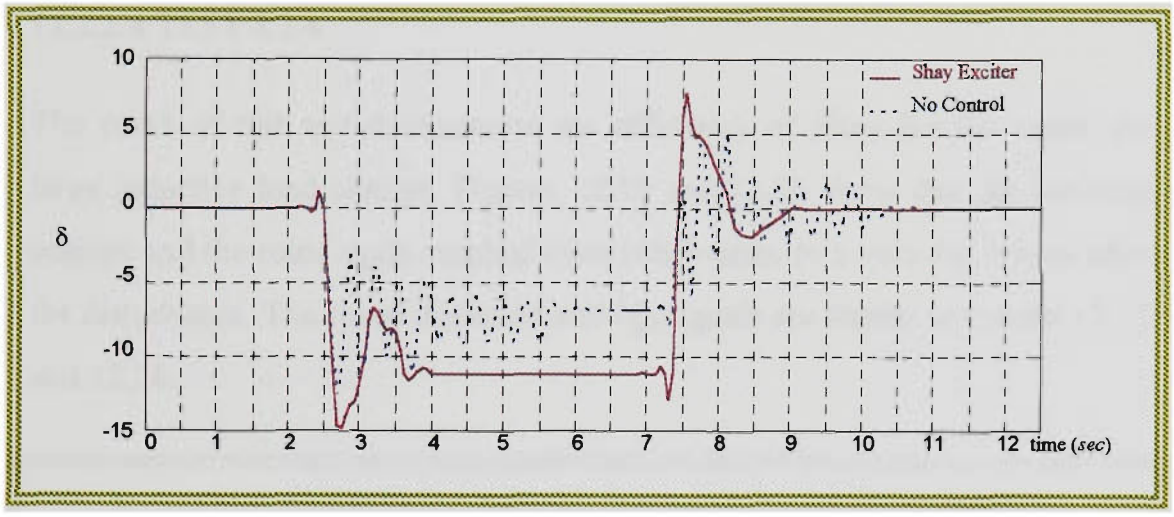


Figure 12.22 rotor angle (electrical degrees), TEST- LL3

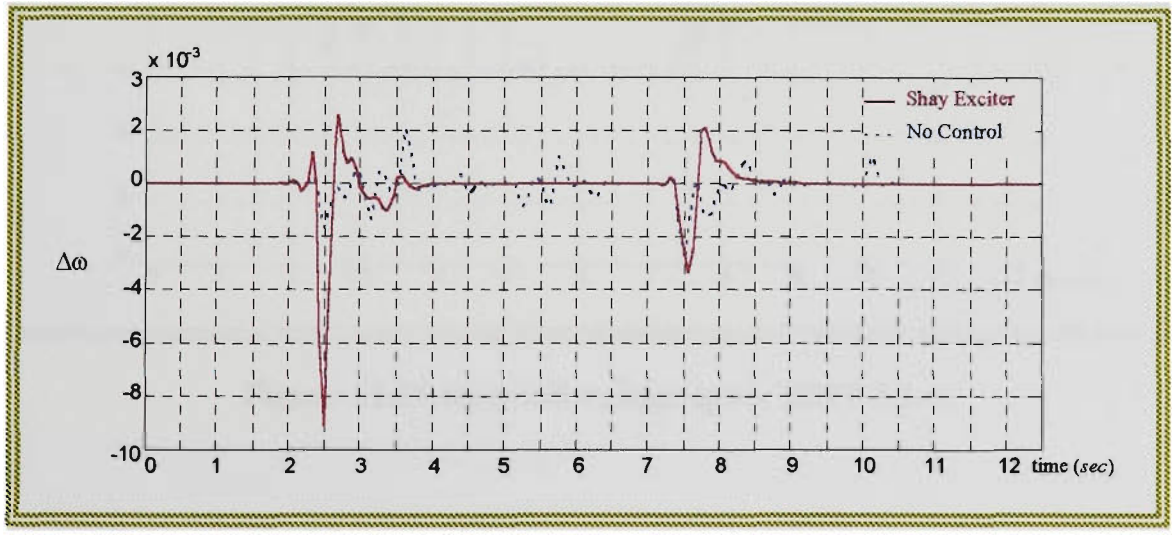


Figure 12.23 speed deviation (*pu*), TEST-LL3

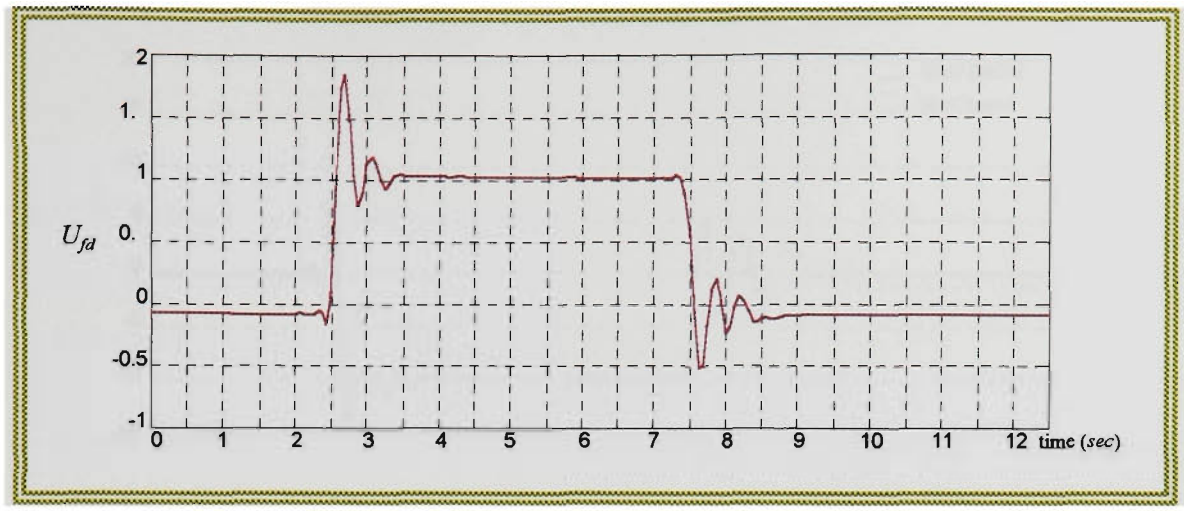


Figure 12.24 excitation signal (*volts*), TEST-LL3

12.1.2.4 TEST-LL4

The result of this test demonstrate the efficiency of Shay-Exciter under the large inductive load change. Figures 12.25 and 12.26 show that the terminal voltage and the rotor angle reached their steady-state in a very short time after the disturbance. The speed deviation and U_{fd} signals are shown in figures 12.27 and 12.28.

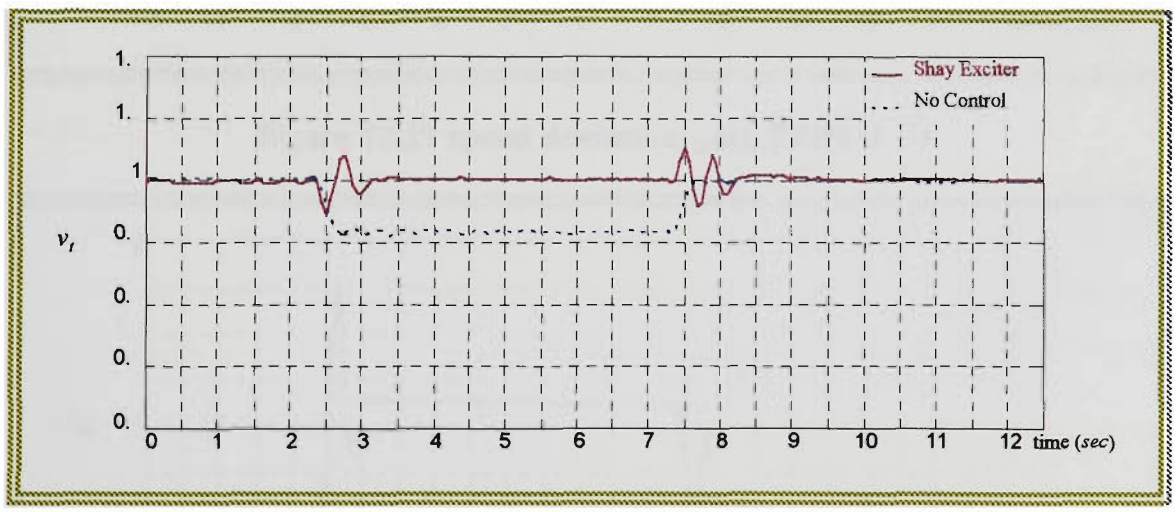


Figure 12.25 terminal voltage (*pu*), TEST-LL4

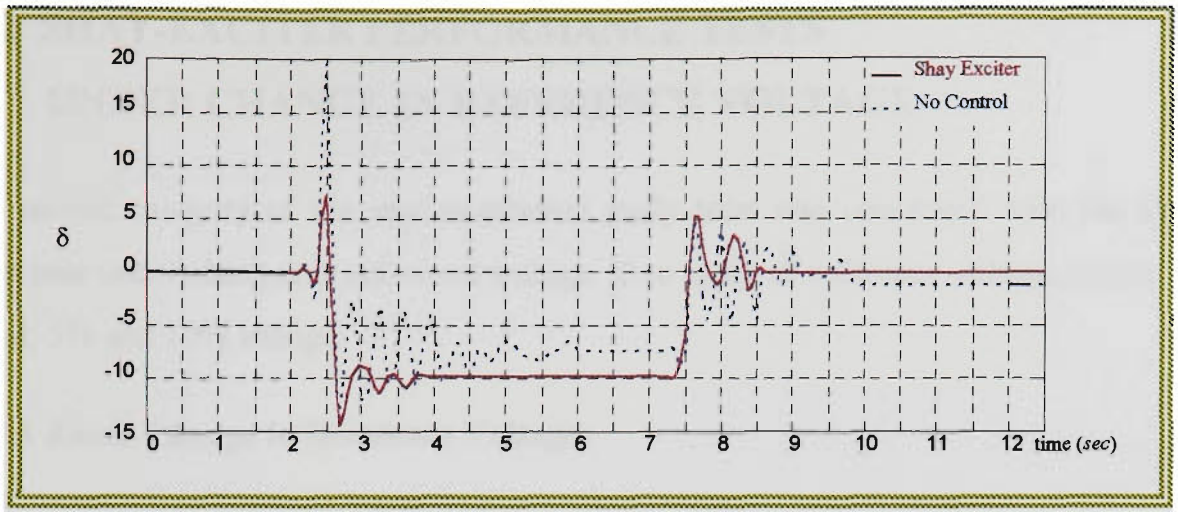


Figure 12.26 rotor angle (electrical degrees), TEST- LL4

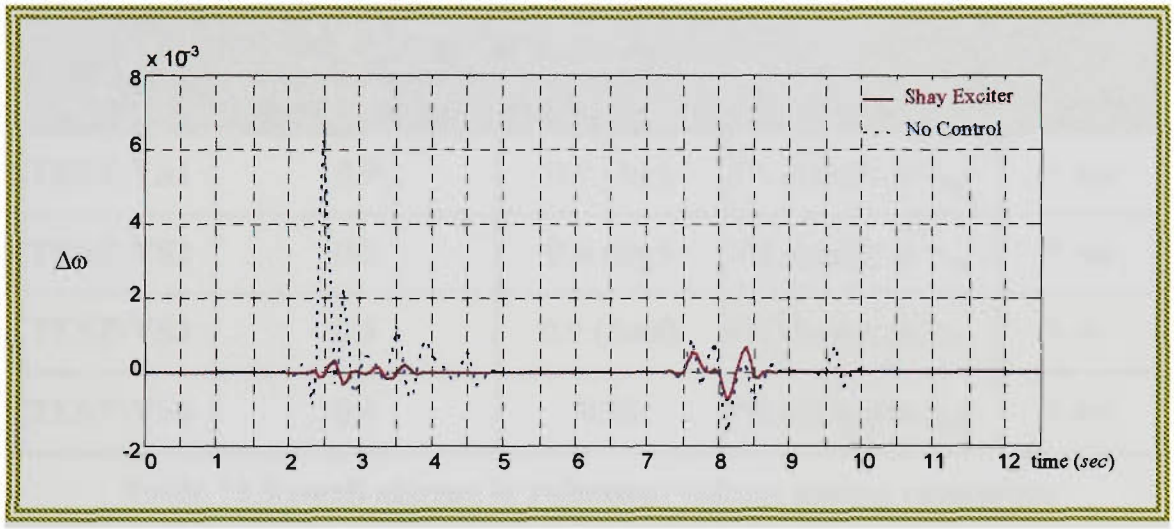


Figure 12.27 speed deviation (*pu*), TEST-LL4

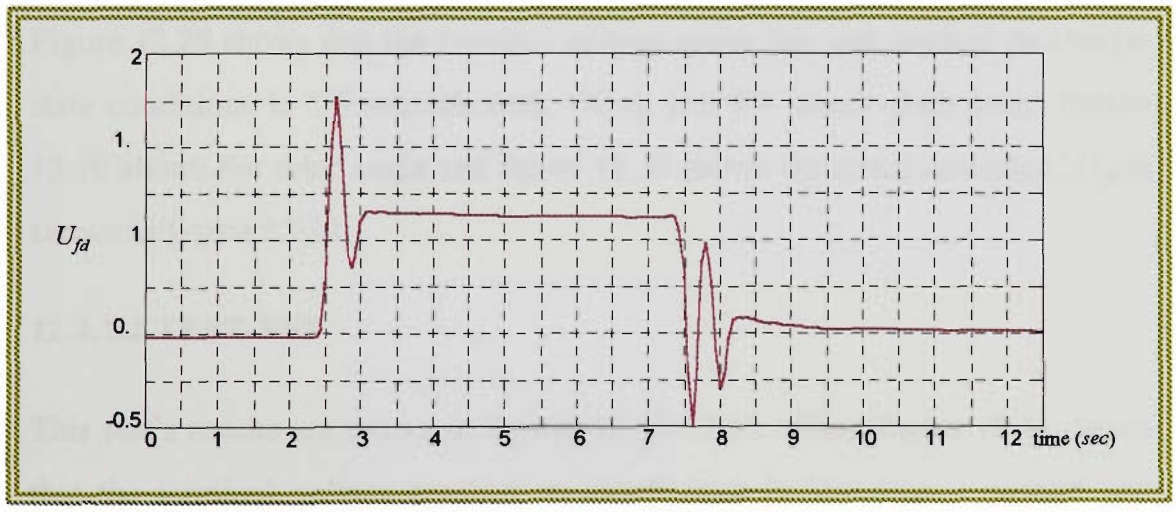


Figure 12.28 excitation signal (volts), TEST-LL4

12.2 SHAY-EXCITER PERFORMANCE TESTS UNDER CHANGE IN REFERENCE VOLTAGE

The second category of the implementation study tests was concerned with the system behaviour under changes in reference voltage. Two types of reference voltage ranges were tested, 5% and 10% change.

12.2.1 Small Change in Reference Voltage

Table 12.3 describes the tests and the operation conditions used for theses tests.

Test	Operating conditions		Disturbance	Duration
	Active power (*pu*)	Power Factor		
TEST-VS1	0.9	0.9 (*lag*)	5% change in v_{ref}	5 *sec*
TEST-VS2	0.5	0.6 (*lag*)	5% change in v_{ref}	5 *sec*
TEST-VS3	0.5	0.9 (*lead*)	5% change in v_{ref}	5 *sec*
TEST-VS4	0.4	0.98	5% change in v_{ref}	5 *sec*

Table 12.3 small change in reference voltage testing conditions

12.2.1.1 TEST-VS1

Figure 12.29 shows that the terminal voltage under this test reached its steady-state conditions in 1.5 seconds with 1% μ_p and 1% steady-state error. Figure 12.30 shows the rotor angle and figure 12.31 shows the speed deviation. U_{fd} is shown in figure 12.32.

12.2.1.2 TEST-VS2

This test's results are shown in figures 12.33-12.36, where figure 12.33 shows that the terminal voltage reached its steady-state in less than 1 second and figure 12.34 shows that the rotor angle reached its steady-state in 2 seconds. $\Delta\omega$ and U_{fd} are shown in figures 12.35 and 12.36 respectively.

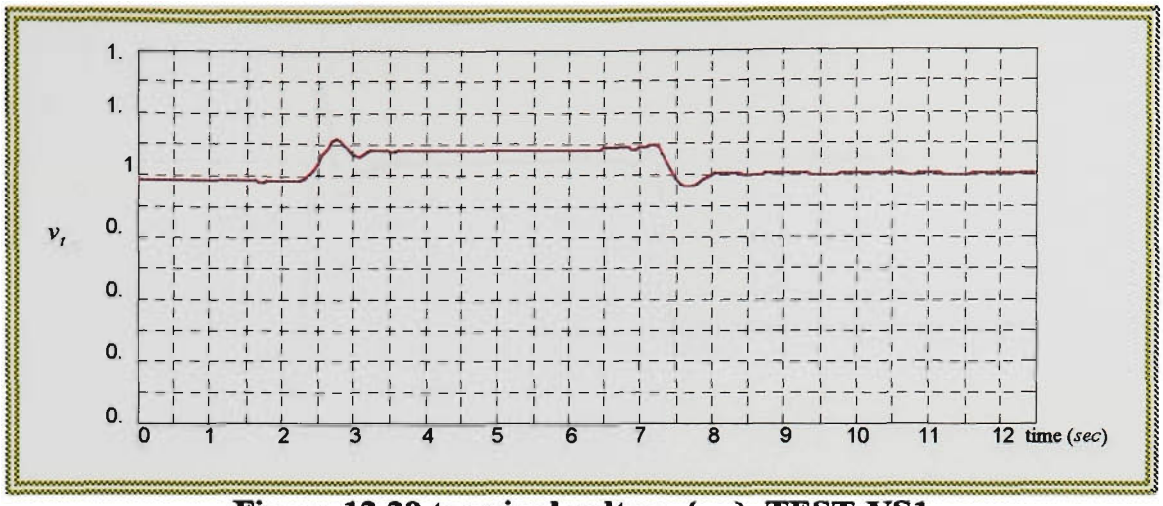


Figure 12.29 terminal voltage (*pu*), TEST-VS1

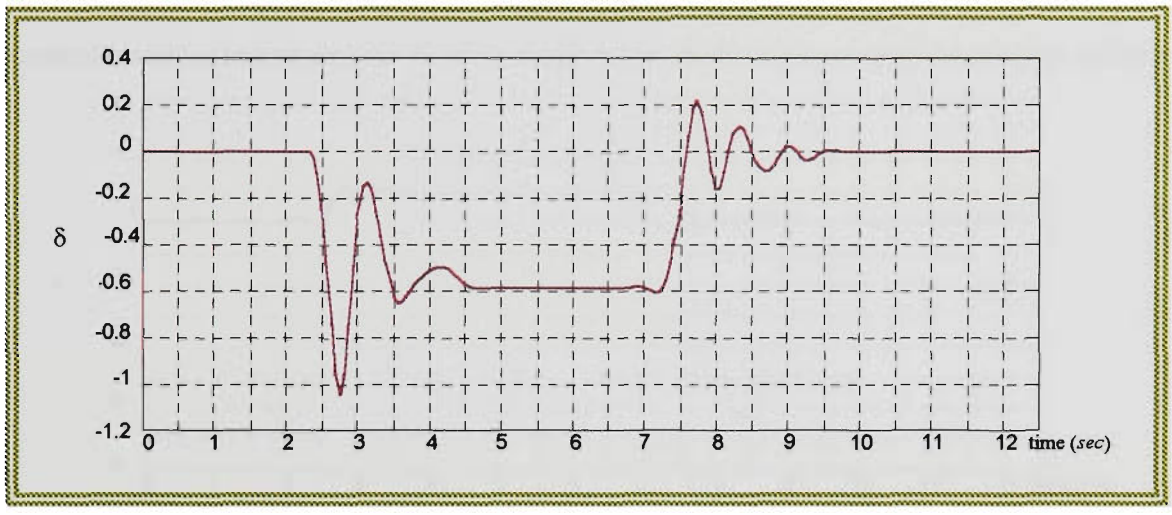


Figure 12.30 rotor angle (electrical degrees), TEST- VS1

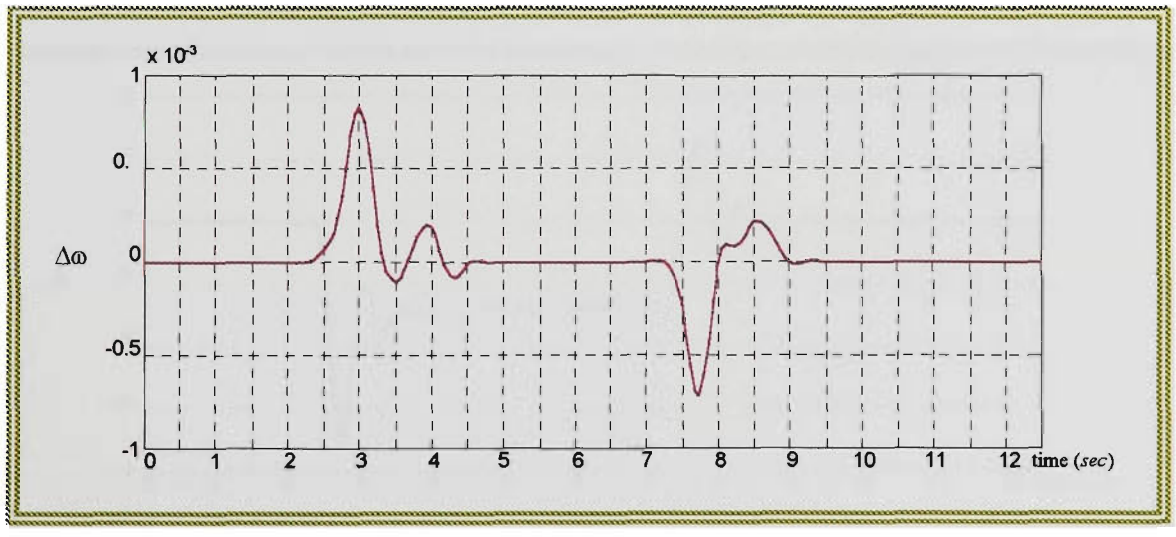


Figure 12.31 speed deviation (*pu*), TEST-VS1

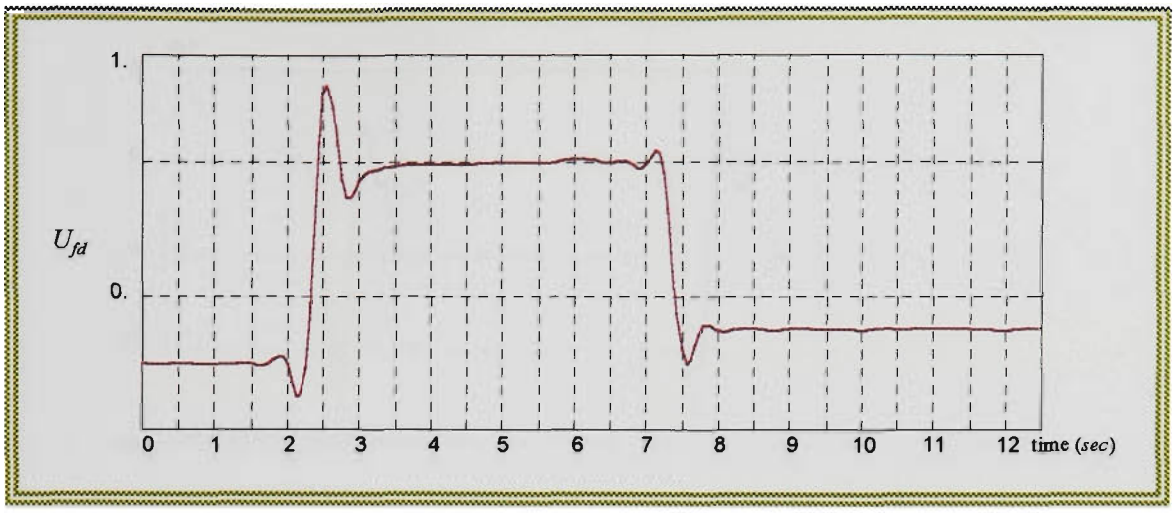


Figure 12.32 excitation signal (*volts*), TEST-VS1

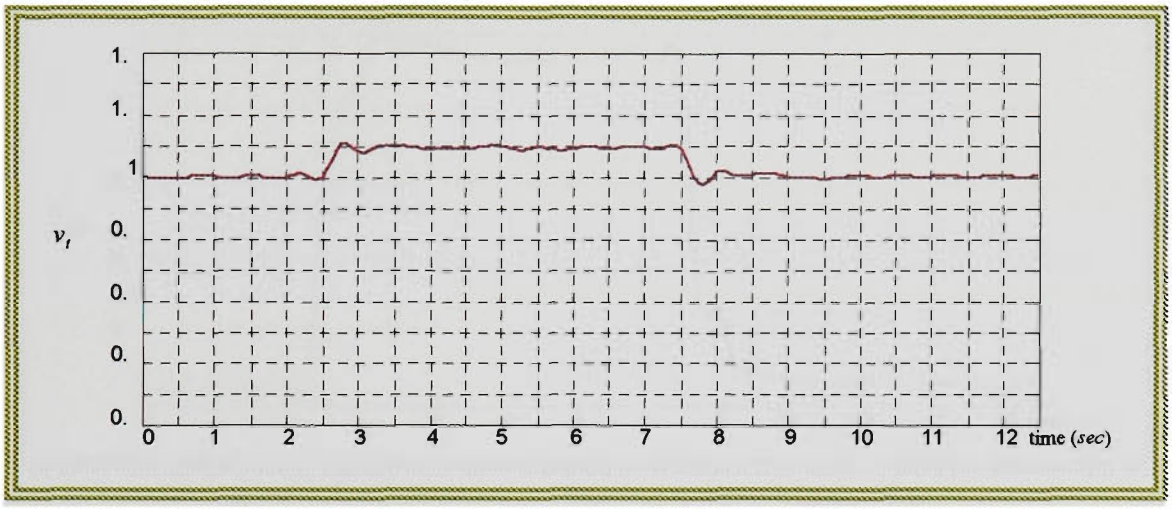


Figure 12.33 terminal voltage (*pu*), TEST-VS2

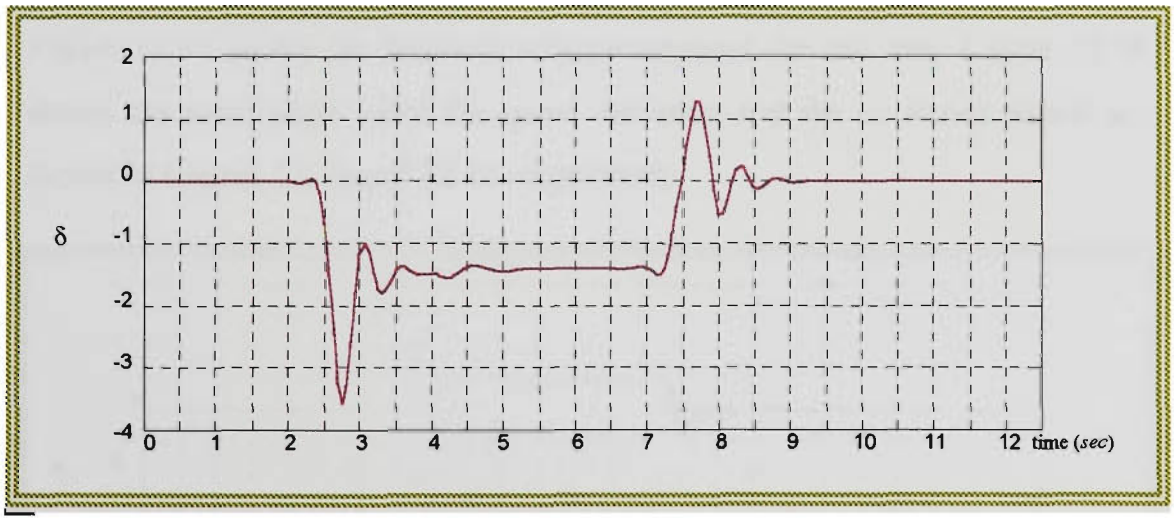


Figure 12.34 rotor angle (electrical degrees), TEST- VS2

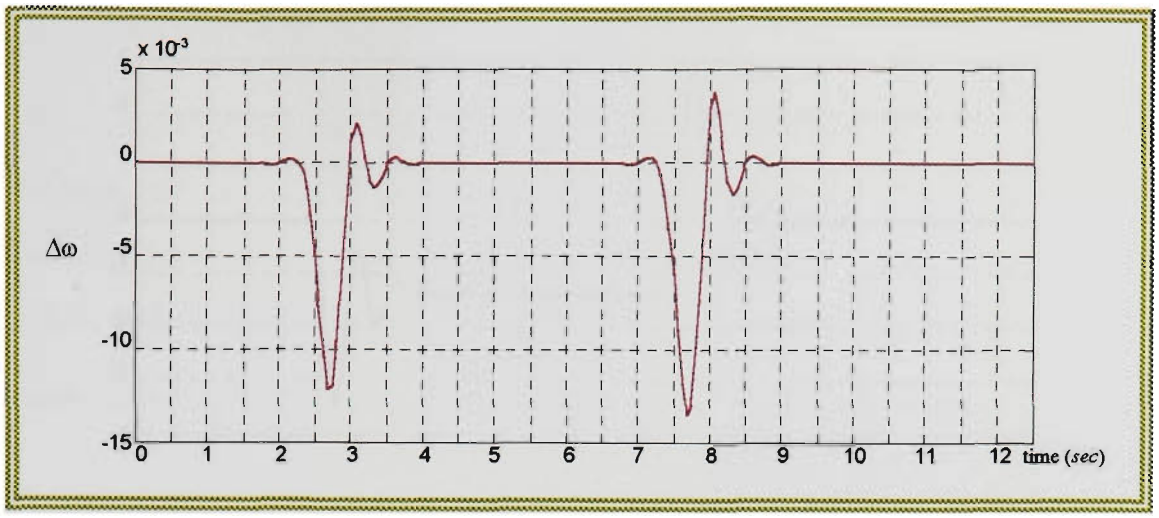


Figure 12.35 speed deviation (*pu*), TEST-VS2

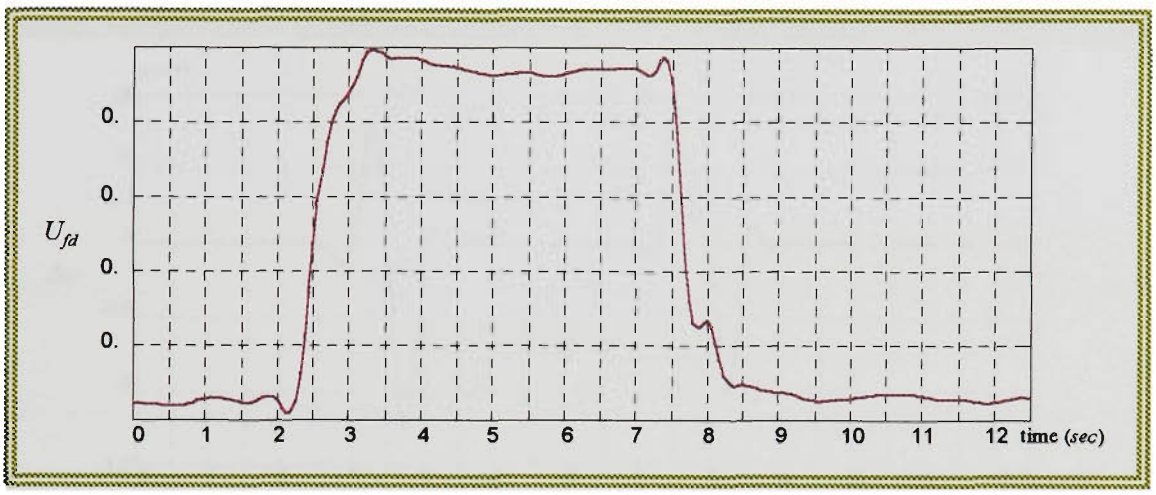


Figure 12.36 excitation signal (*volts*), TEST-VS2

12.2.1.3 TEST-VS3

Figure 12.37 shows the terminal voltage response for this test. Figure 12.38 shows the rotor angle while the speed deviation and the excitation signal are shown in figures 12.39 and 12.40 respectively.

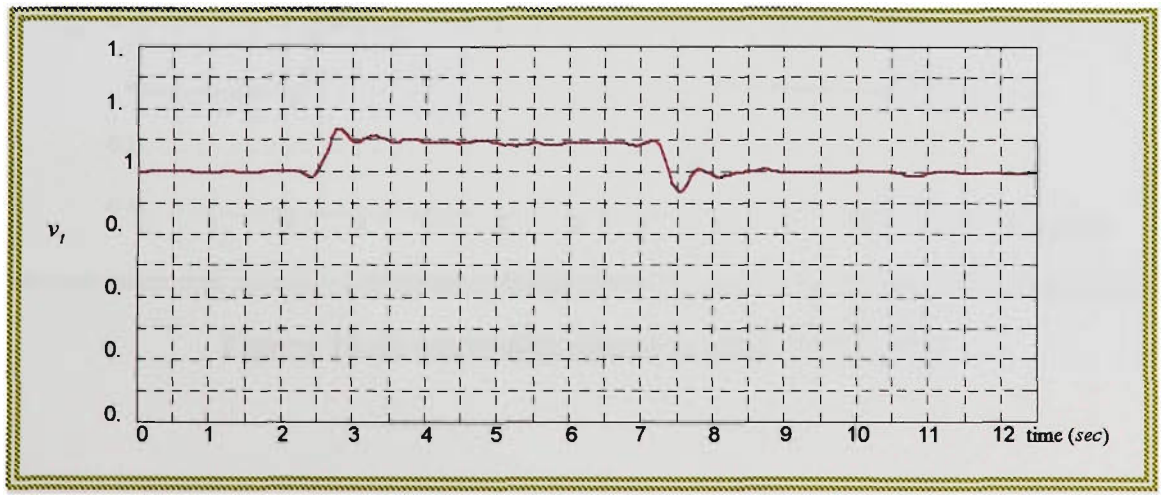


Figure 12.37 terminal voltage (*pu*), TEST-VS3

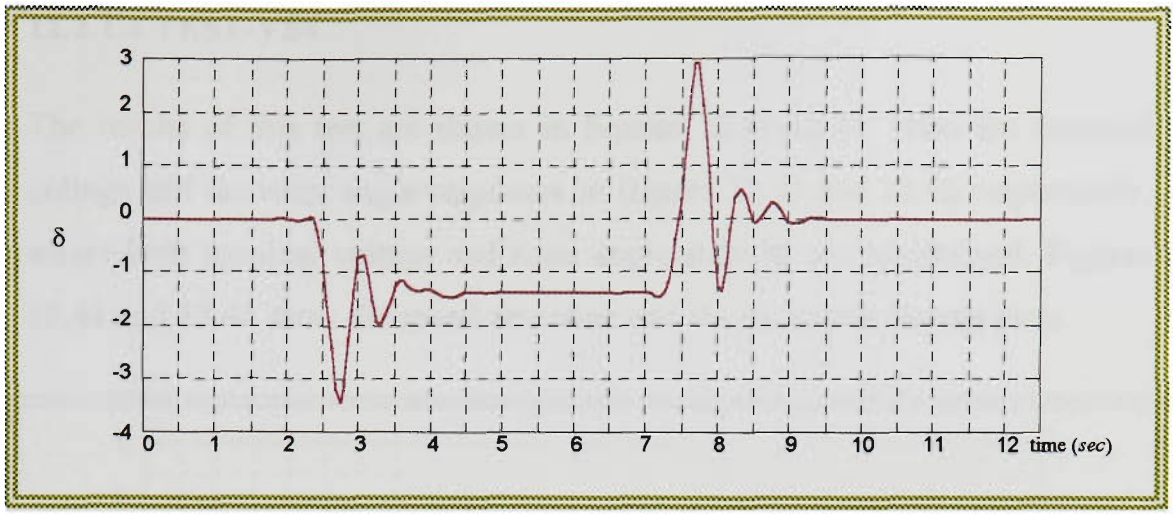


Figure 12.38 rotor angle (electrical degrees), TEST- VS3

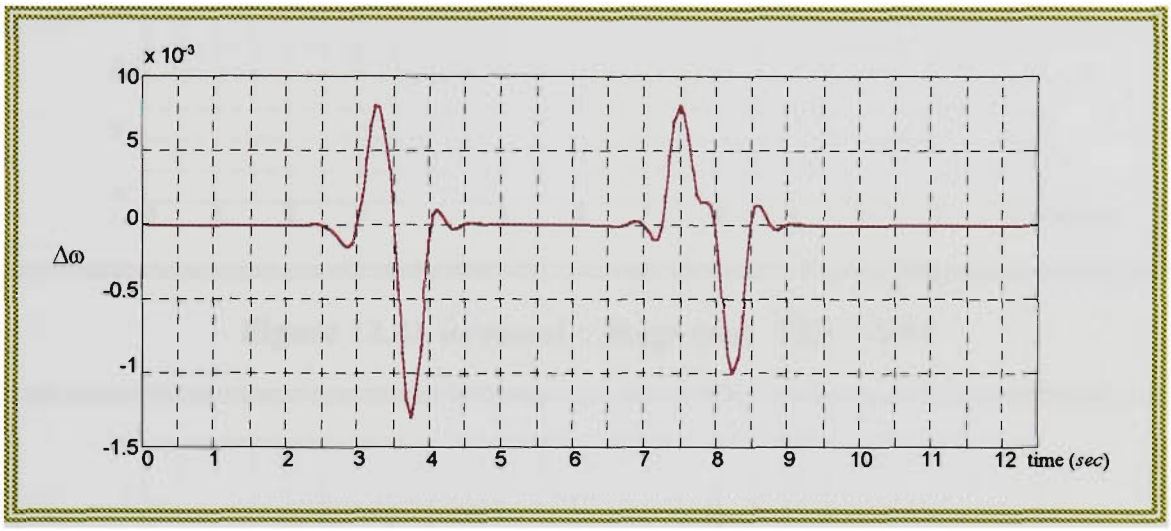


Figure 12.39 speed deviation (*pu*), TEST-VS3

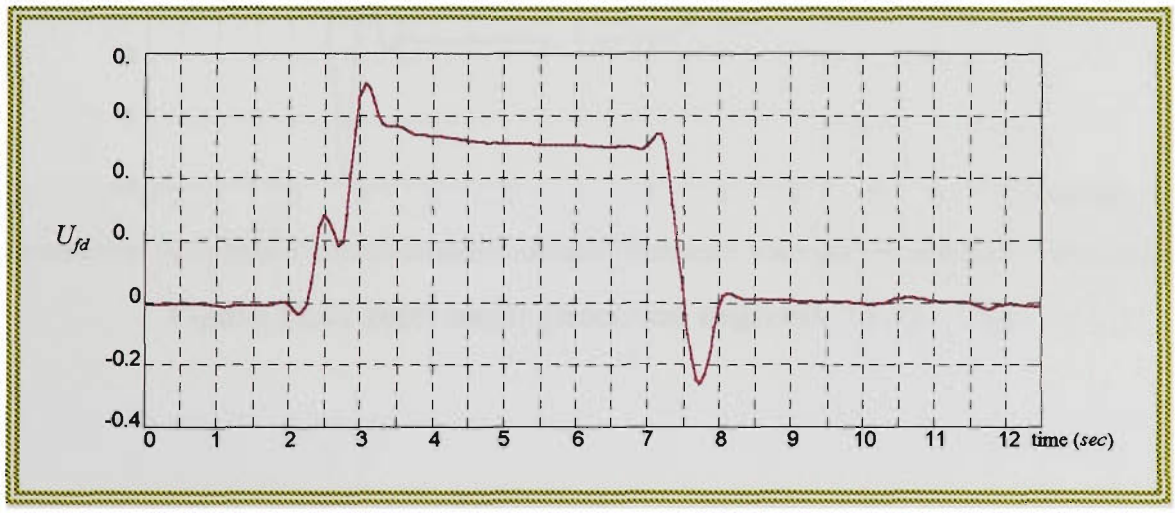


Figure 12.40 excitation signal (volts), TEST-VS3

12.2.1.4 TEST-VS4

The results of this test are shown in figures 12.41-12.44. Note the terminal voltage and the rotor angle responses in figures 12.41 and 12.42 respectively, where both terminal voltage and rotor angle stability are maintained. Figures 12.43 and 12.44 show the speed deviation and the excitation signals plots.

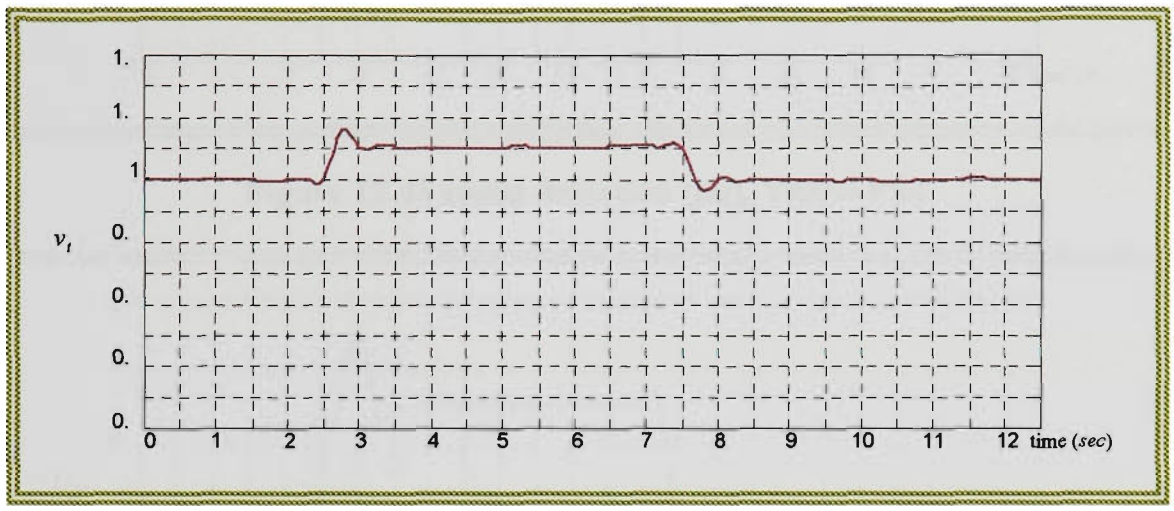


Figure 12.41 terminal voltage (*pu*), TEST-VS4

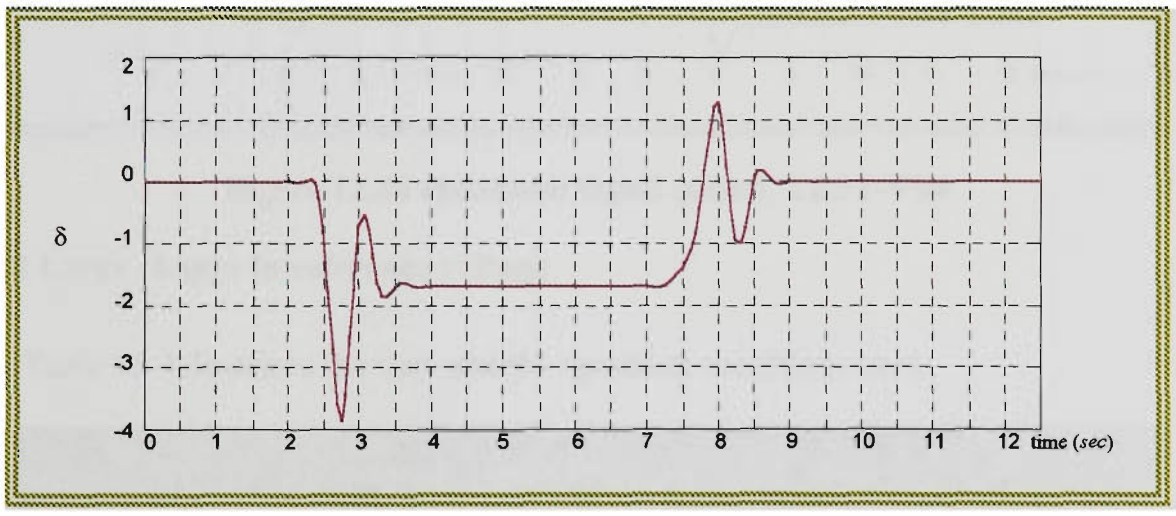


Figure 12.42 rotor angle (electrical degrees), TEST- VS4

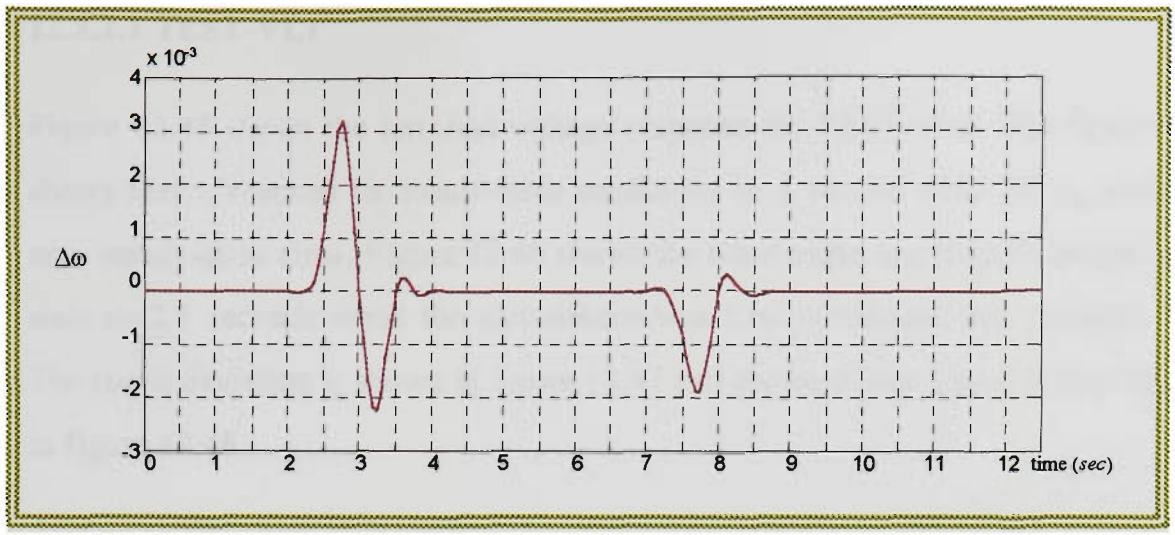


Figure 12.43 speed deviation (*pu*), TEST-VS4

Figure 12.44 excitation signal (volts), TEST-VS4

12.2.2 Large change in reference voltage

Table 12.4 describes the tests and the operation conditions used.

Test	Operating conditions		Disturbance	Duration
	Active power (*pu*)	Power factor		
TEST-VL1	0.9	0.9 (*lag*)	10% change in v_{ref}	5 *sec*
TEST-VL2	0.5	0.6 (*lag*)	10% change in v_{ref}	5 *sec*
TEST-VL3	0.5	0.9 (*lead*)	10% change in v_{ref}	5 *sec*
TEST-VL4	0.4	0.98	10% change in v_{ref}	5 *sec*

Table 12.4 large change in reference voltage testing conditions

12.2.2.1 TEST-VL1

Figure 12.45 shows the terminal voltage response for TEST-VL1. The figure shows that v_t reached its steady-state conditions in 1 second with 5% μ_p and zero steady-state error. Figure 12.46 shows the rotor angle reaching its steady-state in 2.5 seconds when the disturbance was first introduced and released. The speed deviation is shown in figure 12.47 and the excitation signal is shown in figure 12.48.

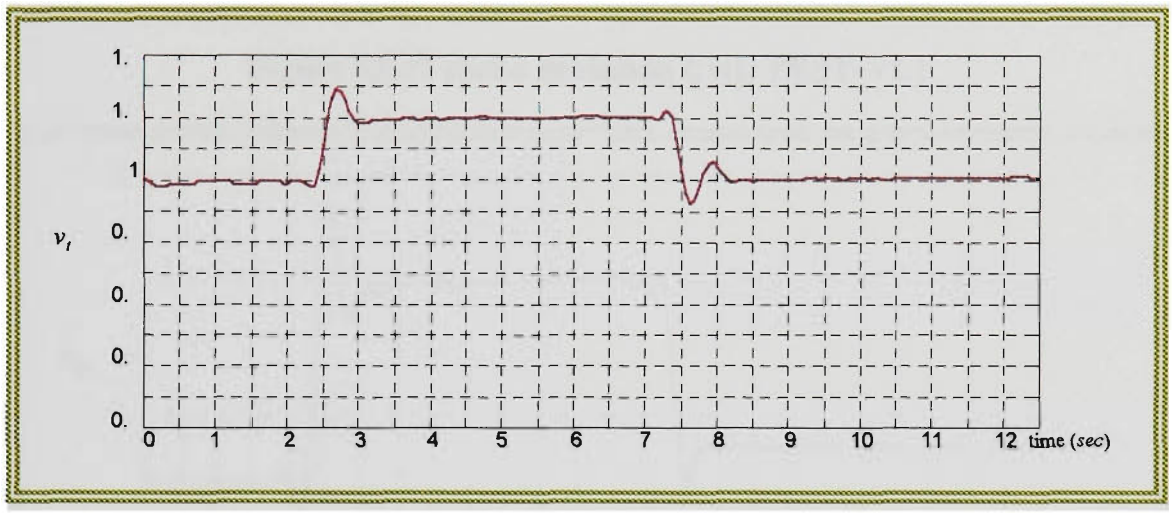


Figure 12.45 terminal voltage (*pu*), TEST-VL1

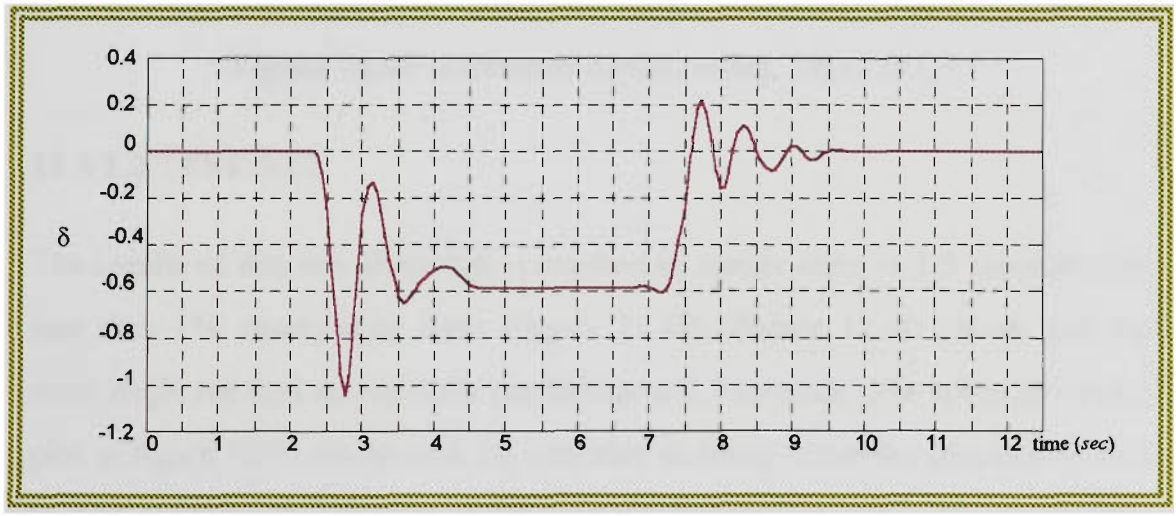


Figure 12.46 rotor angle (electrical degrees), TEST- VL1

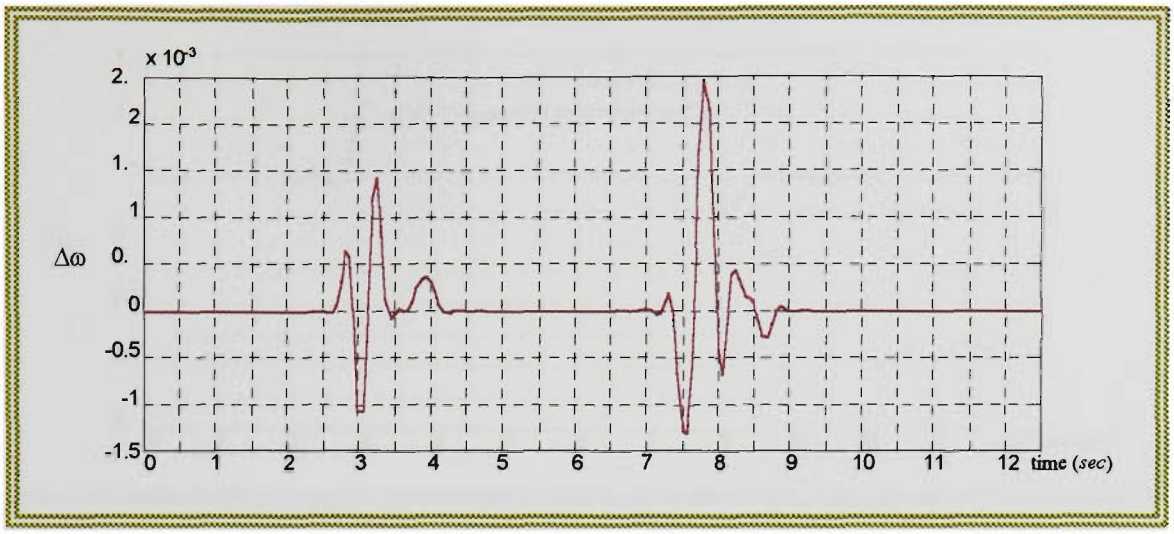


Figure 12.47 speed deviation (*pu*), TEST-VL1

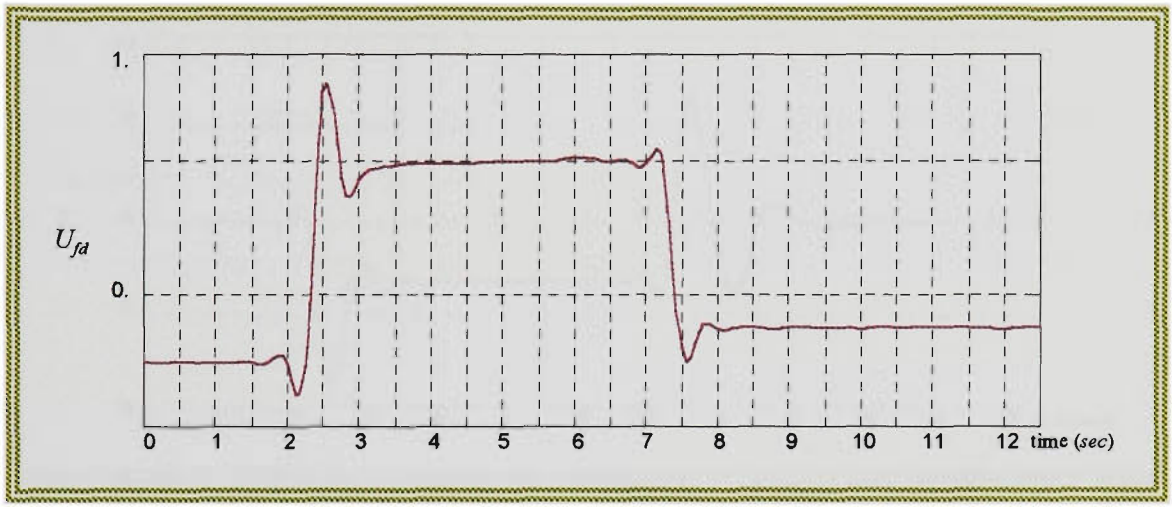


Figure 12.48 excitation signal (*volts*), TEST-VL1

12.2.2.2 TEST-VL2

The results of this test show that v_t reached its steady-state in 1.5 seconds with less than 1% steady-state error (figure 12.49). Figure 12.50 shows that the rotor angle reached steady-state conditions in 2.5 seconds. The speed deviation plot in figure 12.51 shows that U_{fd} was able to dump down the oscillations in δ in 2.25 seconds after the introduction of the disturbance and in 2 seconds after removing it. U_{fd} is shown in figure 12.52.

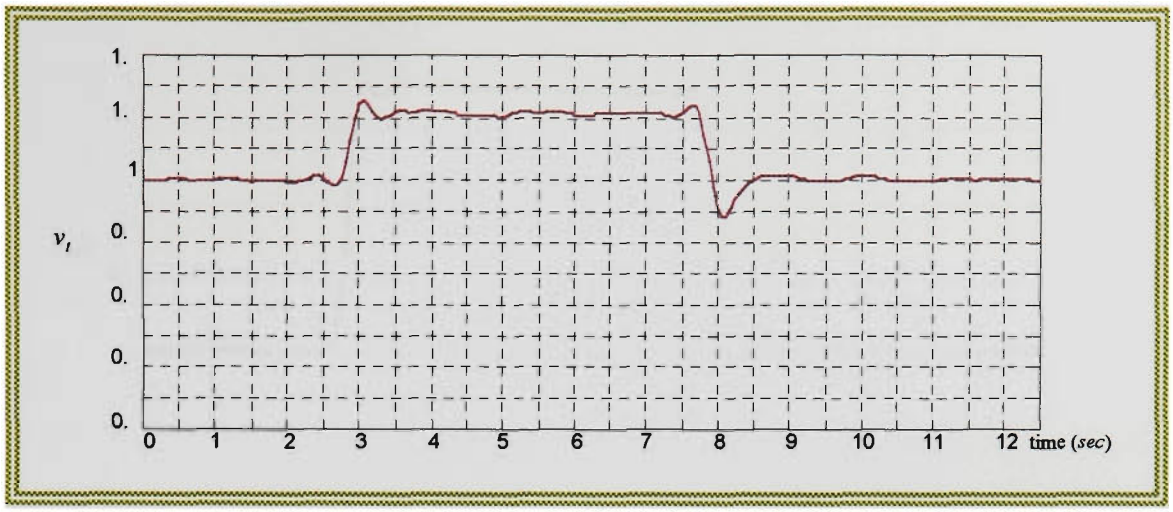


Figure 12.49 terminal voltage (*pu*), TEST-VL2

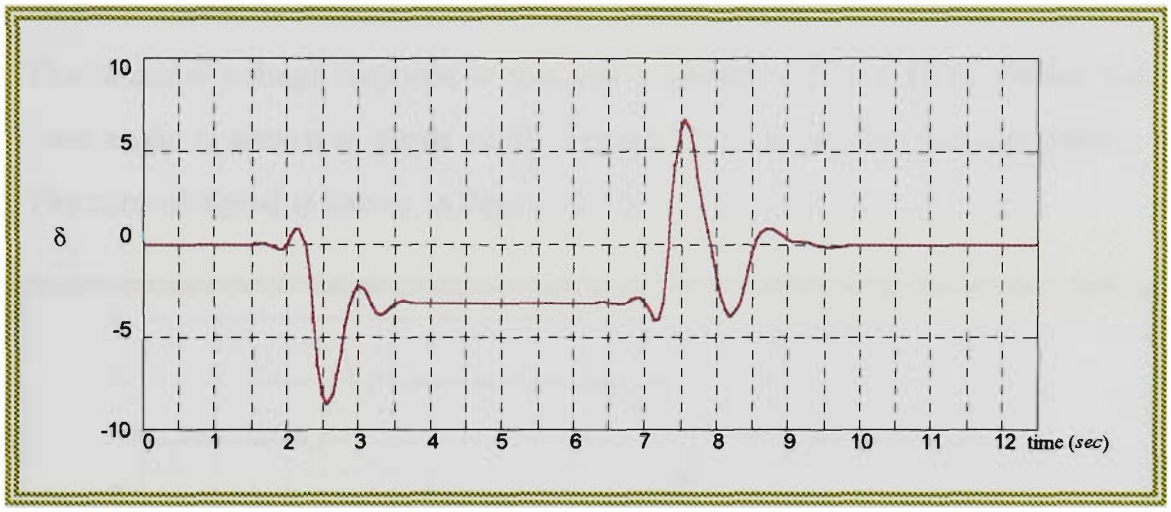


Figure 12.50 rotor angle (electrical degrees), TEST- VL2

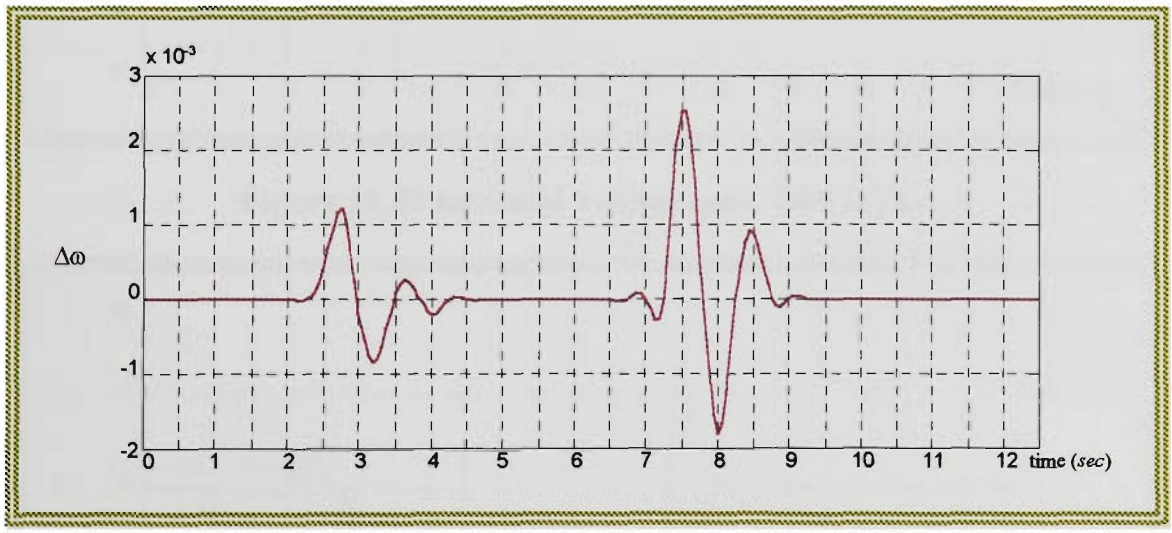


Figure 12.51 speed deviation (*pu*), TEST-VL2

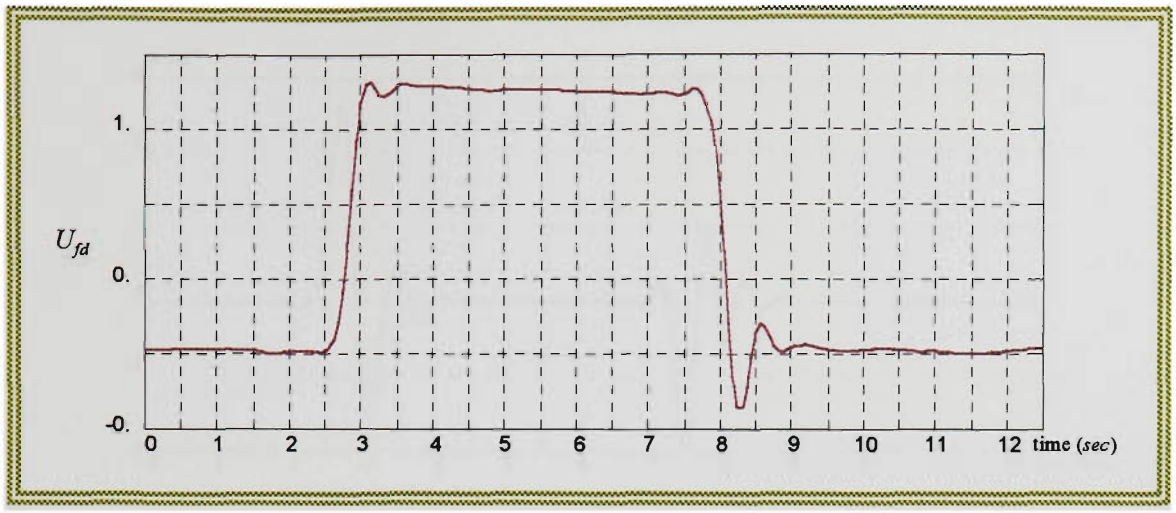


Figure 12.52 excitation signal (volts), TEST-VL2

12.2.2.3 TEST-VL3

The terminal voltage response in this test is shown in figure 12.53. While the rotor angle is shown in given 12.54. Figure 12.55 shows the speed deviation. The control signal is shown in figure 12.56.

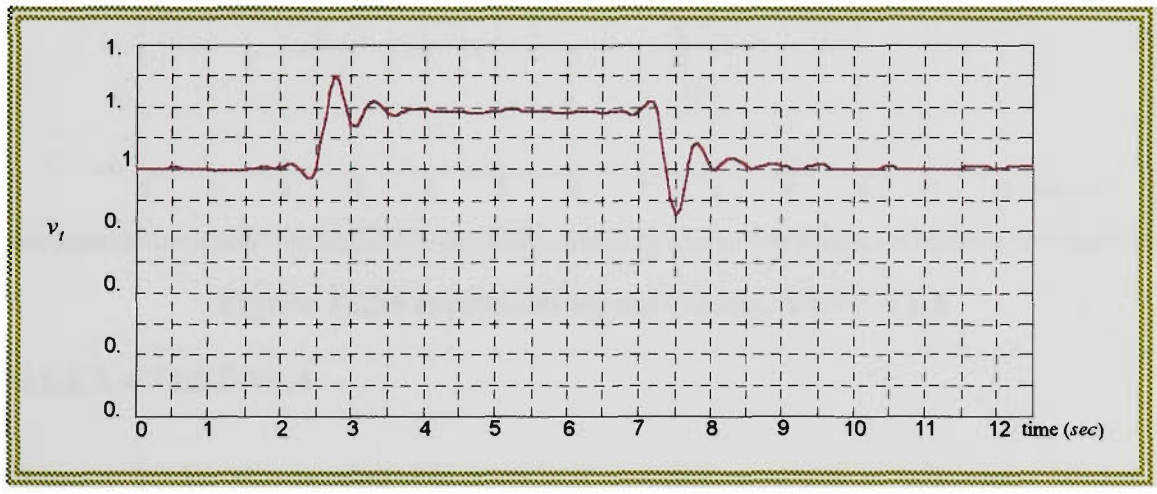


Figure 12.53 terminal voltage (*pu*), TEST-VL3

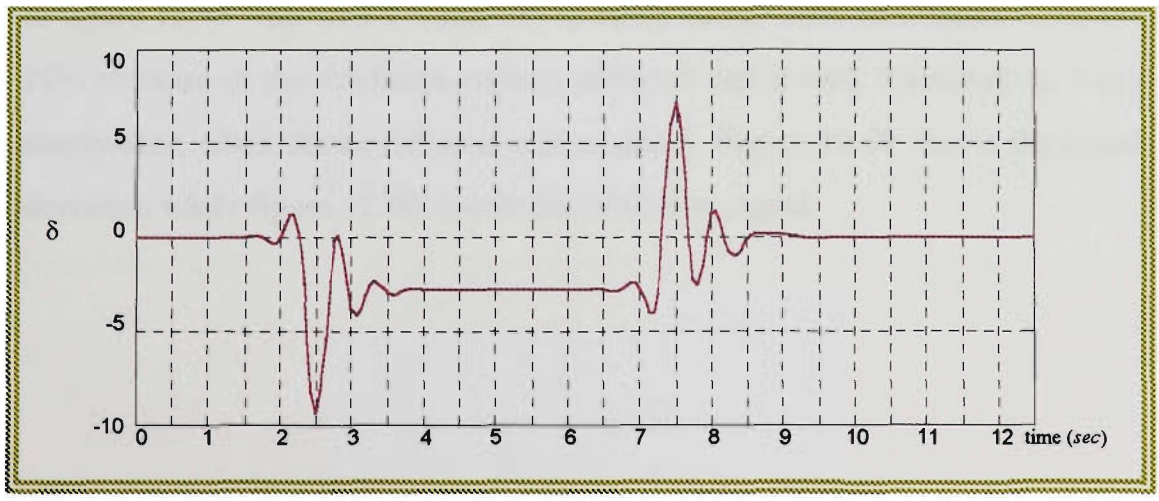


Figure 12.54 rotor angle (electrical degrees), TEST- VL3

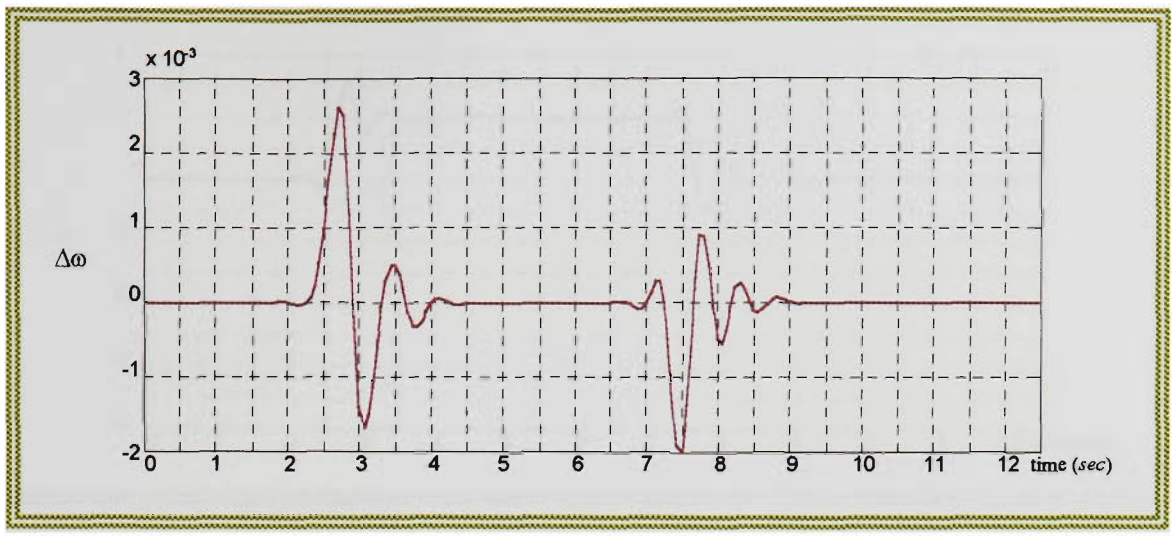


Figure 12.55 speed deviation (*pu*), TEST-VL3

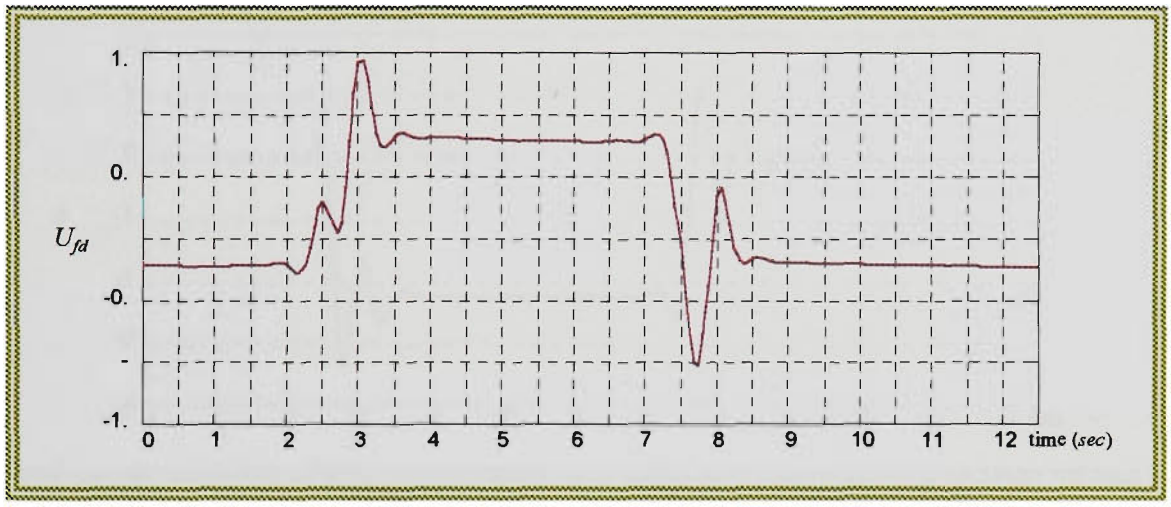


Figure 12.56 excitation signal (volts), TEST-VL3

12.2.2.4 TEST-VL4

Figure 12.57 shows that the terminal voltage reached steady-state in 1.5 seconds with 55 μ_p when the disturbance was first introduced and removed. δ in figure 12.58 required 1.5 seconds to reach steady-state conditions when the 10% increase in the reference voltage occurred and it took 2 seconds to reach steady-state when the disturbance was released. Figure 12.59 shows the speed deviation while figure 12.60 shows the excitation signal.

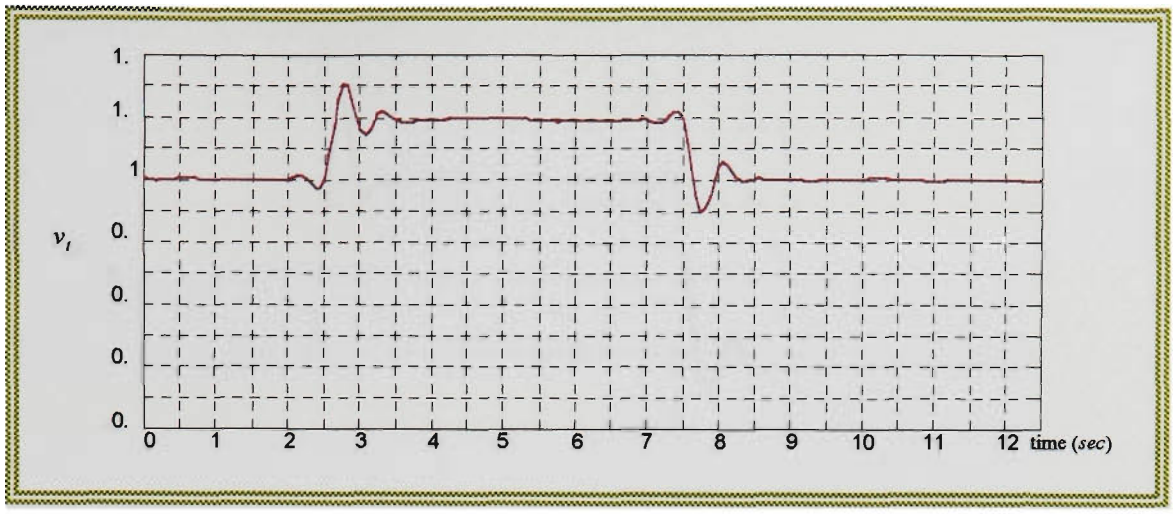


Figure 12.57 terminal voltage (*pu*), TEST-VL4

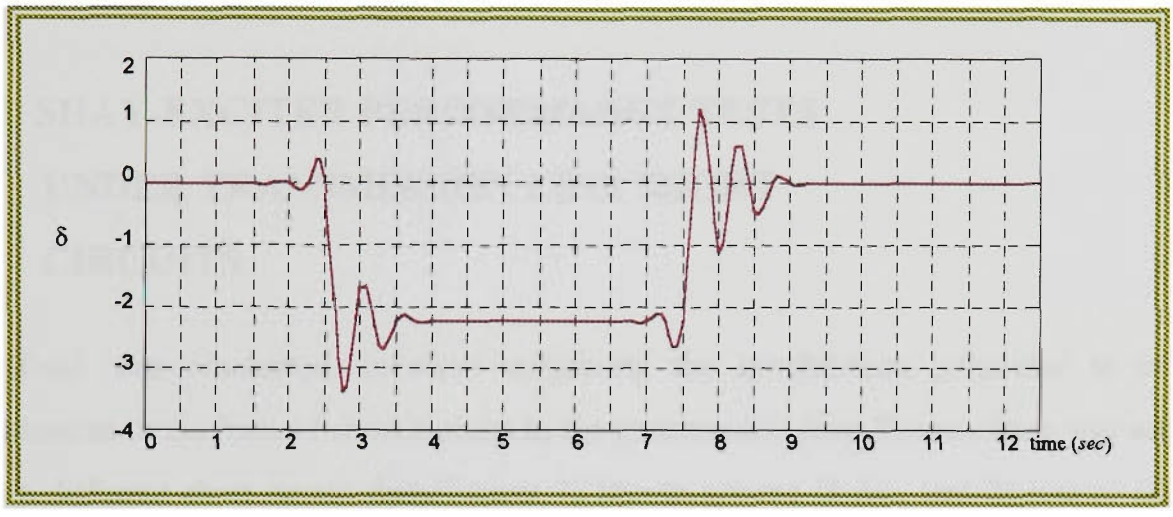


Figure 12.58 rotor angle (electrical degrees), TEST- VL4

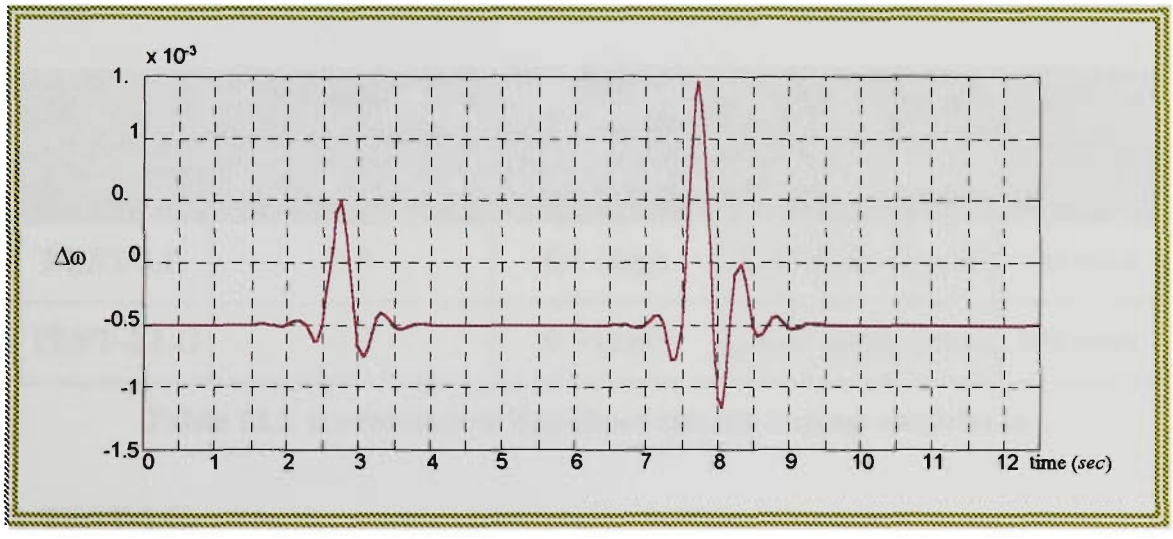


Figure 12.58 speed deviation (*pu*), TEST-VL4

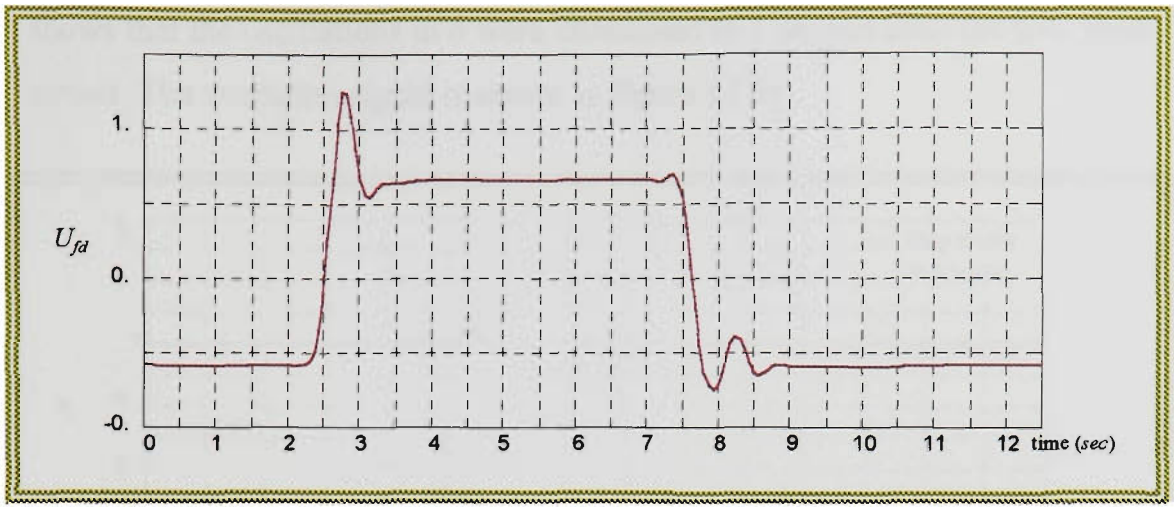


Figure 12.59 excitation signal (volts), TEST-VL4

12.3 SHAY-EXCITER PERFORMANCE TESTS UNDER TRANSMISSION LINE SHORT CIRCUITS

The final tests conducted, involved subjecting the synchronous generator to severe disturbances in the form of short circuits in the transmission line. The machine was subject to two different short circuit disturbances 1) line to ground (L-G), and 2) line to line to ground (L-L-G) short circuits. Table 12.5 describes the short circuit tests and operation conditions.

Test	Operating conditions		Disturbance	Duration
	Active power (*pu*)	Power factor		
TEST-LG	0.9	0.9 (*lag*)	L-G short circuit	100 *msec*
TEST-LLG	0.9	0.9 (*lag*)	L-L-G short circuit	100 *msec*

Table 12.5 transmission line short circuit testing conditions

12.3.1 TEST-LG

The L-G short circuit test results show that Shay-Exciter was able to maintain the terminal voltage level in 1.5 seconds after the disturbance with 2.5% μp as shown in figure 12.61. Figure 12.62 shows the rotor angle and figure 12.63

shows that the oscillations in δ were eliminated in 1 second after the L-G short circuit. The excitation signal is shown in figure 12.64.

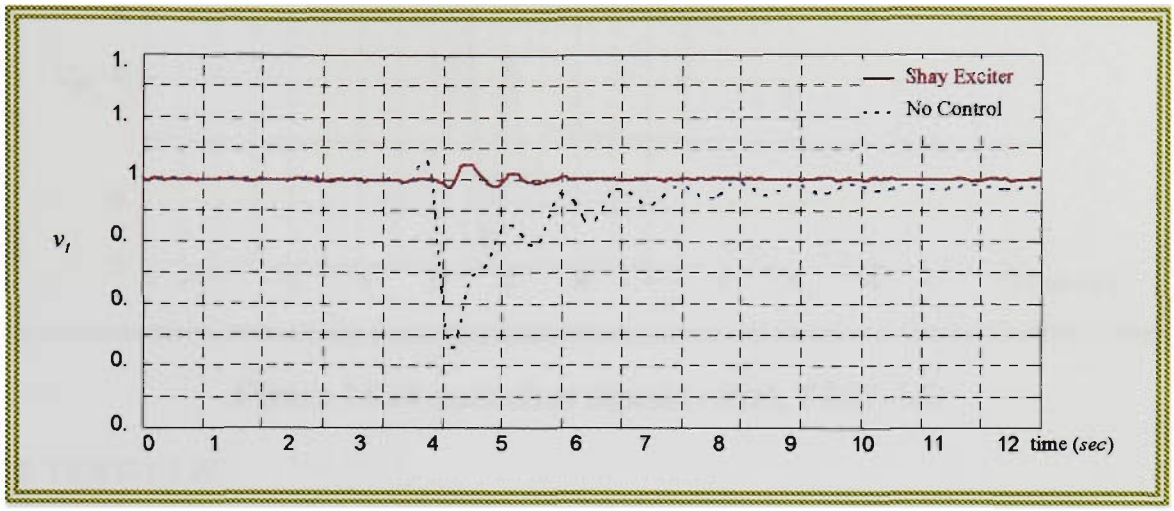


Figure 12.61 terminal voltage (*pu*), TEST-LG

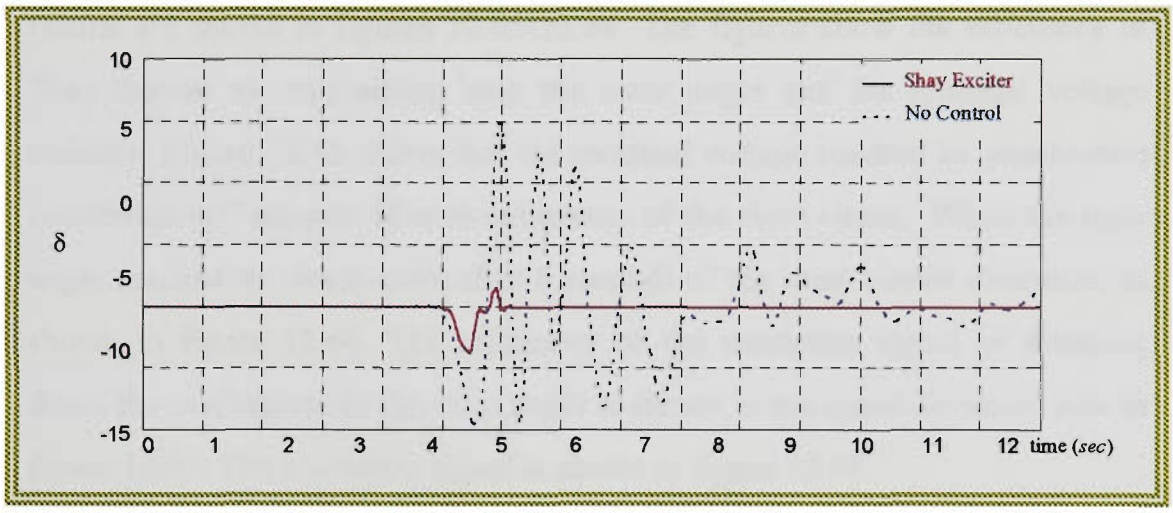


Figure 12.62 rotor angle (electrical degrees), TEST- LG

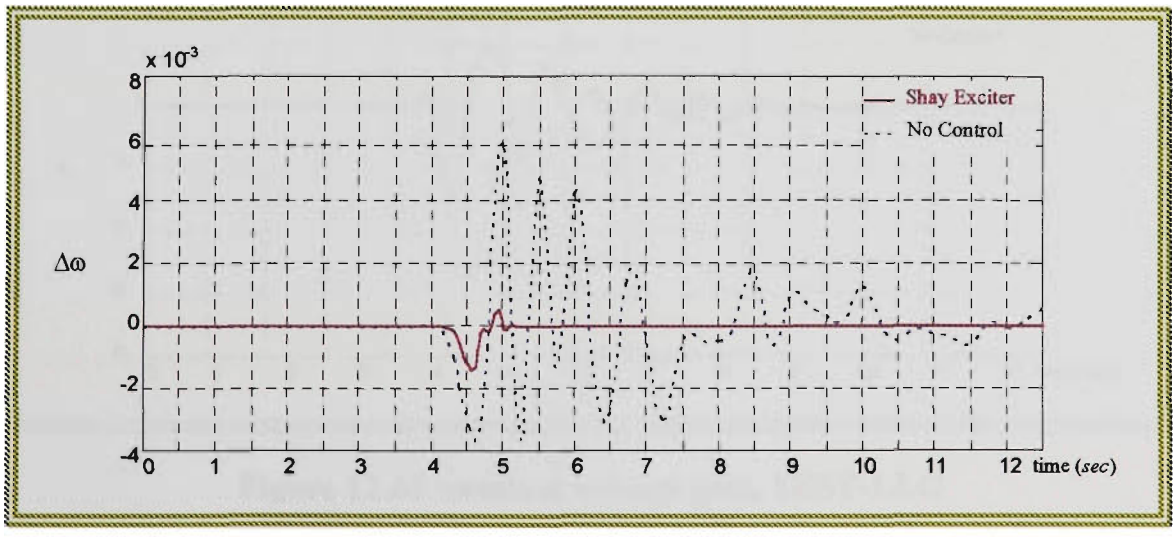


Figure 12.63 speed deviation (*pu*), TEST-LG

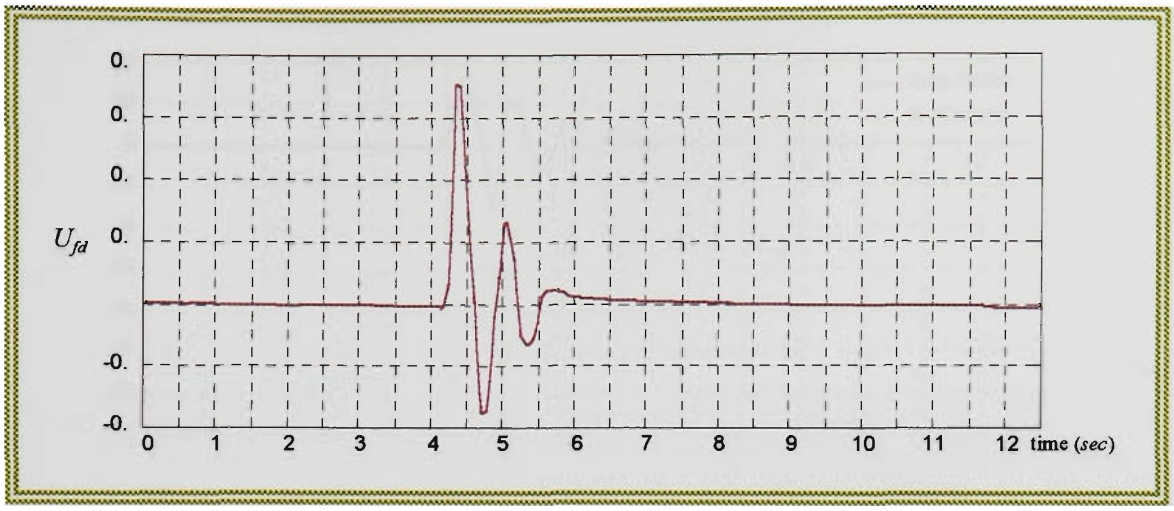


Figure 12.64 excitation signal (volts), TEST-LG

12.3.2 TEST-LLG

The very difficult L-L-G short circuit test sustained for 100 milli seconds results are shown in figures 12.65-12.68. The figures show the efficiency of Shay-Exciter to maintaining both the rotor angle and the terminal voltage stability. Figure 12.65 shows that the terminal voltage reached its steady-state conditions in 7 seconds after the clearance of the short circuit. While the rotor angle reached its steady-state after 6 seconds of the short circuit clearance, as shown in figure 12.66. The efficiency of the excitation signal in dumping down the oscillations in the rotor angle is shown in the speed deviation plot in figure 12.67. The excitation signal is shown in figure 12.68.

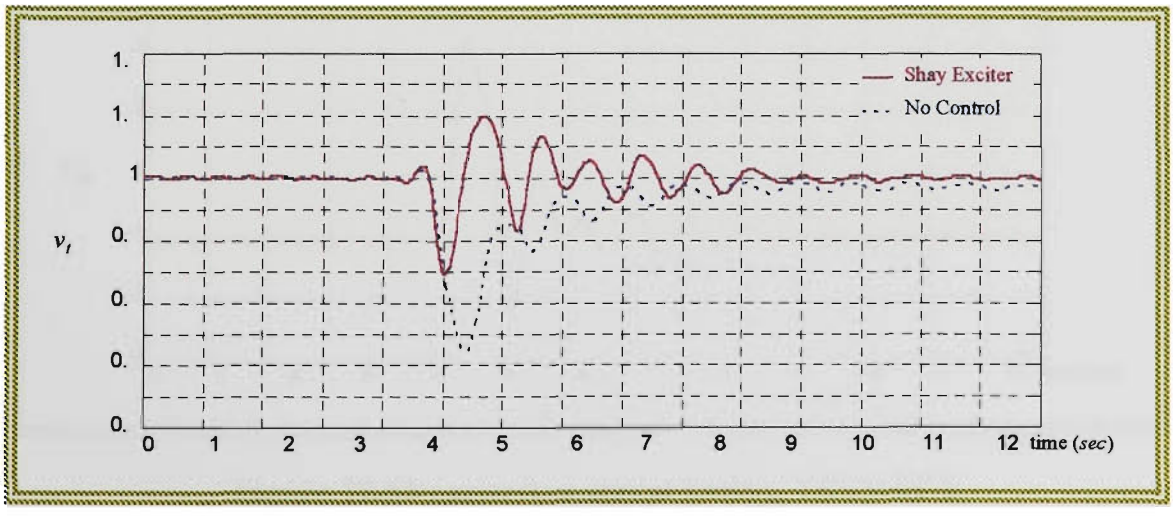


Figure 12.65 terminal voltage (*pu*), TEST-LLG

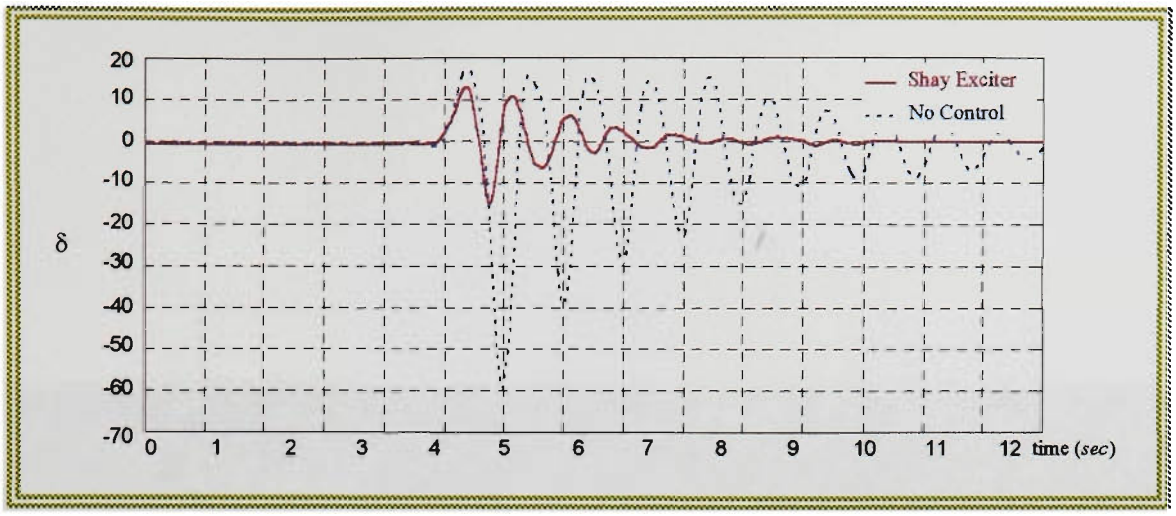


Figure 12.66 rotor angle (electrical degrees), TEST-LLG

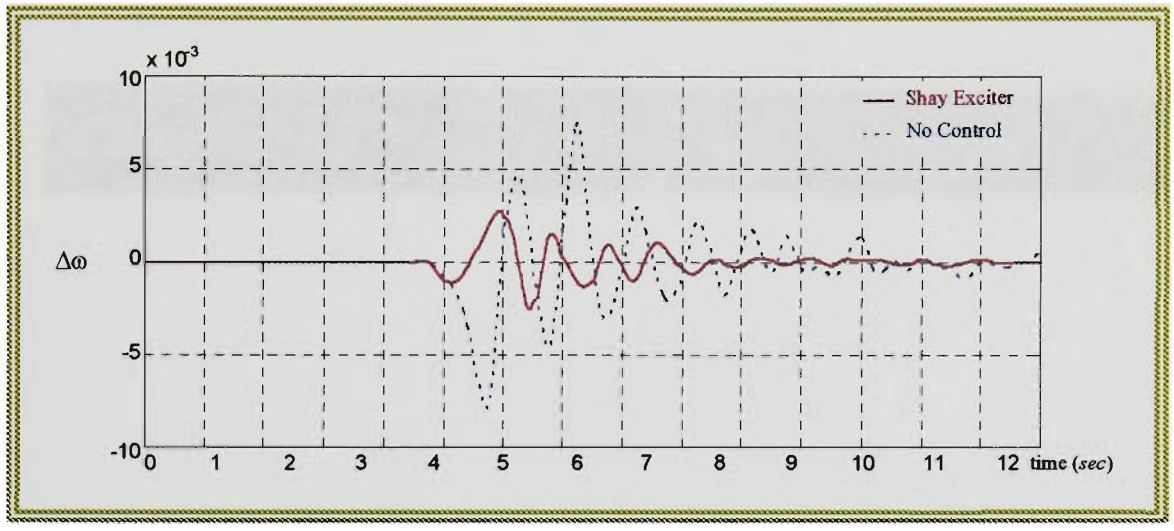


Figure 12.67 speed deviation (*pu*), TEST-LLG

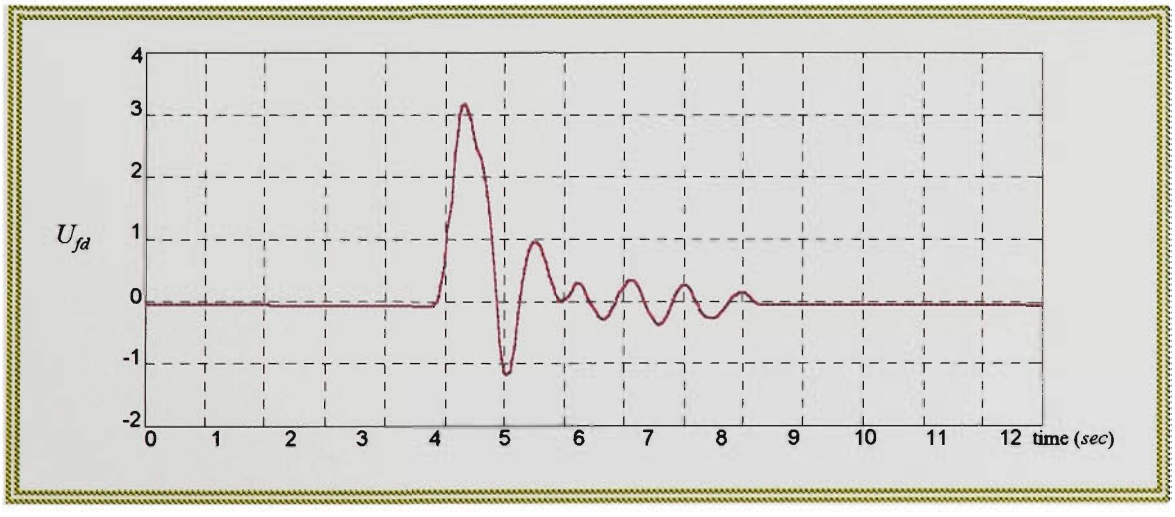


Figure 12.68 excitation signal (*volts*), TEST-SC2

SECTION 4 MAIN RESULTS AND CONCLUSION

Chapter 13 *Main Results and Conclusion*

Chapter 13 *Main Results and Conclusion*

13.0 INTRODUCTION

This chapter presents the main results and achievements of this thesis. Practical and theoretical achievements are presented in section 13.1 followed by the publication and presentation list in section 13.2. The chapter ends with discussion of future work directions in the domain of fuzzy logic control.

13.1 MAIN RESULTS AND ACHIEVEMENTS

The main result of this work is the development of the new methodology of FLC design and implementation for different applications. The methodology includes a number of innovative methods and concepts, developed by the author, ie. the operational angle, operational sector, FLC operational status, a universal FLC structure (Shay) and a method of simultaneous tuning of both input and output FLC scaling factors. Practical recommendations on the application of this methodology and its implementation tools are given as well. The proposed methodology has been comprehensively tested by computer simulation and prototype. The universal FLC structure was implemented for the electric power system excitation control.

MAIN RESULTS IN THEORETICAL DEVELOPMENT

1) **Improvement of the Knowledge Representation**

   The thesis provided new techniques and methods of refining the data obtained by on-line measurements. These techniques give better representation of the plant performance and status. This goal was achieved by the introduction of the operational angle concept in chapter 7. The chapter shows that the operational angle concept is much broader than the differential one and that the differential principle is a subdivision of the operational angle concept.

2) Improvement of Fuzzy Reasoning Methods

The thesis developed the L/R COG method (see appendix A). This reasoning method makes use of the already available positional information with respect to the membership function prototype. The L/R COG method provides the mechanism of implementing and processing the positional information. The method treats every single convex membership function (class) in the universe of discourse (input/output) as two non-intersected classes separated by a singleton at the prototype of the class.

3) Development of A Structured On-Line FLC Parameter Tuning

The thesis includes the introduction of Shay-Tune in chapter 8, Section 2. The method developed enables simultaneous tuning of both the input and output ranges of the FLC. Shay-Tune applies a standardised algorithm that requires minimal changes for different applications. The tuning mechanism used requires no off-line parameter adjustment. The superiority of Shay-Tune was clear from the practical implementation results shown in chapter 12. Shay-Tune enabled the excitation control system to operate and prove robustness under different operational points. Shay-Tune enables the designer to largely influence the tuning process by altering two clearly defined parameters, L and T.

4) Development of the Universal FLC Design Structure

The Shay structure developed in the thesis can be grouped into one controller block with only one input as shown in chapter 10. The structure proposed is universal as it is not based on a particular model or application. Shay applies mechanisms and means of processing the plant information according to pre-defined performance criteria. The processing is performed regardless of the characteristics of the plant under control. Reusability is a major feature of Shay where few and simple alterations are required when using Shay in different applications. Shay structure employs two basic concepts, the operational status and the operational

sector. The operational status concept used in Shay provides a means of measuring the FLC ranges suitability upon which the input and output ranges are updated. The operational sector concept used in Shay gives a clear representation of the plants dynamics.

PRACTICAL ACHIEVEMENTS

1) **Provision to the Designer of Practical Recommendations on the Application of the Proposed Methodology and Shay Structure Parameter Choice**

Based on the research made in the thesis (chapter 9, section 2) recommendations are given to choose the proposed structure parameters to satisfy different criteria traditionally applied in control engineering. The chapter highlighted the general and specific guidelines of how the designer can change the *SALT* parameters so the FLC meets some predefined performance criteria. The influence of every individual factor is detailed in chapter 9. The chapter shows a heuristic approach in developing general patterns of how each factor effects the system behaviour.

2) **Provision of Practical Guidelines for FLC Design**

The thesis includes an experience based description of how to utilise the available theoretical knowledge about the system. This was shown in Chapter 10 while setting the general control strategy. In chapter 10 the analysis of the equal area criteria was directed towards the evolution of the control strategy. Secondly it gives a practical guide on how to manipulate the *SALT* parameters to meet the control criteria based on simulation results.

3) **Provision of a Model for A Practical Implementation**

The thesis gives a provision of a model for practical implementation of the Shay FLC in section 3. In this section the design procedure is explained in detail. The knowledge engineering part of manipulation of the

information captured and utilised from field and theoretical resources is shown in chapter 10. Chapter 11 shows the practical implementation model where Shay-Exciter is implemented for the excitation control of a synchronous generator connected to an infinite bus through a transmission line. The chapter shows the hardware and software components used in the implementation. It is important to indicate here that the implementation procedure involves the use of generalised protocols and interfacing tools. It would have been much easier to implement the Shay-Exciter in a more application specific manner.

4) Provision of General Purpose Software and Control Protocols

The handshaking protocol developed by the author and used to synchronise the operation of the host PC and the DSP microprocessors is a generalised protocol. This protocol can be used in any other multiprocessing application. The concept of having shared memory proved to be very useful in such applications where more than one processor is operating in parallel. The thesis provided software tools that perform bit-by-bit manipulation and conversion of the input data registers in the DSP. This is very important in most DSP applications as the registers size and organisation can vary from one board to another and is in most cases different than the 32-bit organisation used in most IBM compatible PCs. Thus, having a high level language code (ANSI-C) that can perform this conversion is a very useful tool.

The thesis contributes general purpose fuzzy processing functions written in ANSI-C (see Appendices B.1-9). These functions are robust and require no modifications when used to implement different FLCs. The DSP and host PC drivers as well as the machine/computer interfacing hardware and software developed in this work are currently used in the Electrical and Electronic Engineering Department laboratories for many applications with the author's limited permission.

IMPLEMENTATION

1) **Implementation of Fuzzy Excitation Control System**

The thesis contributed a fully fuzzy excitation control system. Shay-Exciter introduced in chapter 10 replaced both the automatic voltage regulator and the power system stabiliser. The author achieved this by the introduction of the Pre-Control stage where a single parameter representing the rotor angle and the terminal voltage was made possible.

2) **Implementation Circuity and Hardware**

The thesis gives a full description of the practical implementation circuity and hardware used in the implementation study (see chapter 11). These procedures can be followed for any other similar application.

3) **Implementation Results**

A large number of tests have been conducted. The tests presented in chapter 12 were conducted under a wide range of operational points and disturbances that vary in severity. Shay-Exciter was able to maintain the rotor angle and the terminal voltage stability for all the tests and without the need of any intervention or parameter adjustment from one test to another. This confirms the effectiveness and robustness of the controller.

13.2 PUBLICATIONS AND PRESENTATIONS

The results of this thesis have been published and presented in a number of national and international journals, conferences and seminars. A list of the publications is given below.

Chapters in Books

- **Ghanayem O. and L. Reznik**, *Excitation Control Of A Synchronous Generator Using Universal Adaptive Fuzzy Logic Controller Structure*, to be published in the book titled: "Fuzzy System Design: Social and Engineering Applications"/Ed.: L. Reznik, V. Dimitrov, J. Kacprzyk. PHYSICA-VERLAG (a Springer-Verlag Company), Heidelberg, to be published in 1997.

International Journals

- **Ghanayem O. and A. Stoica**, *Fuzzy Reasoning Using Positional Information*, accepted for publication in the International Journal of General Systems, the Special Issue on Approximate Reasoning (IJGS), to be published in 1997.

International Refereed Conferences

- **Ghanayem O. and L. Reznik**, *A Universal Adaptation Procedure For Fuzzy Controller Design With The Application To Power System Stability Control*, Proceedings of the 35th Conference on Decision and Control, pp. 1141-1146, December 11-13, 1996, Kobe, Japan

- **Ghanayem O. and L. Reznik**, *A Fuzzy Logic Structure For On-Line Parameter Tuning With The Application To Power System Excitation Control*, accepted for presentation at the 7th International Fuzzy Systems Association (IFSA) World Congress, to be held in, June 25 - 29, 1997, Prague, Czech republic, to be published in 1997.

- **Ghanayem O and L. Reznik**, *Excitation Control of a Synchronous generator using An On-Line Adaptive Fuzzy Logic Controller Structure*, accepted for presentation at the 6th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97), to be held in, July 1-5, 1997, Barcelona, Spain, to be published in 1997.

- **Ghanayem O. and L. Reznik**, *Hierarchical Versus Adaptive Fuzzy Logic Controllers: Design and Performance*, Proceedings of the 2nd Australian and New Zealand Conference on Intelligent Information Systems, pp. 224-228, November 29-December 2, 1994, Brisbane, Australia.

- **Ghanayem O.**, *An Alternative Reasoning Method*, Proceedings of the Industrial and Engineering Applications of Artificial Intelligence and Expert Systems Conference (IEA/AIE) (invited and additional papers), pp 41-45, June 6-8, 1995, Melbourne, Australia.

- **Ghanayem O. and L. Reznik**, *A New Reasoning Approach and its Application in Power System Stability*, Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing (EUFIT'95), vol. 3, pp. 1527-1532, August 29-September 1, 1995, Aachen, Germany.

- **A. Stoica, L. Herron, M. Wingate, and O. Ghanayem,** *Fuzzy Neural Networks as Distributed Fuzzy reasoning Systems,* Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing (EUFIT'95), vol. 1, pp. 86-90, August 29-September 1, 1995, Aachen, Germany.

- **Ghanayem O. and L. Reznik**, *A Hybrid AVR-PSS Controller Based on Fuzzy Logic Technique,* Proceedings of the Control-95 Conference, vol. 2, pp. 347-351, October 20 -24, 1995, Melbourne, Australia.

- **Ghanayem O. and R. Berangi**, *On Line Identification Of Input/Output Ranges Of A FLC Scheme*, Proceedings of the First International Discourse on Fuzzy Logic and the Management of Complexity (FLAMOC'96), pp 150-154, January 15-18, 1996, Sydney, Australia.

- **Ghanayem O. and L. Reznik**, *A Novel Excitation Control Scheme: Design And Implementation*, Proceedings of the 4th Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS 96), November 18-20, 1996, Adelaide, Australia.

13.3 DIRECTIONS OF FUTURE DEVELOPMENT

- **Fuzzy Logic Stability and Evaluation Criteria**

  Although this thesis contributes performance based evaluation criteria, we still feel that more work is necessary in this field. Efforts should be directed towards new definitions of FLC stability and evaluation criteria. Research should avoid the attempts of mapping FLCs to linear controllers evaluation methods. The criteria should reflect the nature of fuzzy logic as an integrated mixture that combines knowledge engineering, control and artificial intelligence and not to focus on the control part only. Some non-deterministic measures are required. We believe that extra work should be done to generalise the definitions introduced in this work as they may form a suitable foundation for such research.

- **Fuzzy Logic Hardware**

  To bring fuzzy logic more in line with real time applications, simple fuzzy hardware should be available. Most of the available hardware on the market provides a one set kit where the processor and the peripherals come in one single compact board limiting the upgrading and improvement of the kit. We need small chips that implement different parts of the fuzzy mathematics. This

will make the upgrading and reusability of the hardware possible as you can always update your system without the need of buying a new one.

- **Fuzzy Logic Education**

  This is a very urgent need for fuzzy logic. This super concept should be transferred from higher level research laboratories to the standard engineering or other courses in the first years of the undergraduate studies. This can be achieved by introducing the subject as a general studies subject in universities.

REFERENCES

REFERENCES

[1] L. A. Zadeh, *Fuzzy Sets*, Information and Control, vol. 8, pp. 338-353, 1965.

[2] L. A. Zadeh, *Outline of a New Approach to the Analysis of Complex Systems and Decision Process*, IEEE Transactions on Systems, Man and Cybernetics, vol. 3, pp. 28-44, Jan. 1973.

[3] C. Elkan, *The Paradoxical Success of Fuzzy Logic*, IEEE Expert, vol. 9, no. 4, pp. 3-8, Aug. 1994.

[4] L. A. Zadeh, *Fuzzy Logic = Computing With Words*, IEEE Transactions on Fuzzy Systems, vol. 4, no. 2, pp. 103-111, May 1996.

[5] B. Kosko, *Fuzzy Thinking*, New York, Hyperion, 1993.

[6] E. H. Mamdani, *Twenty Years of Fuzzy Control: Experiences Gained and Lessons Learnt*, in Proc. of the Second IEEE International Conference on Fuzzy Systems, vol. 1, San Francisco, CA, USA, Mar./Apr. 1993,.

[7] E. H. Mamdani, *An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller*, International Journal Man-Machine Studies, vol. 7, pp. 1-13, 1975.

[8] E. Cox, *The Seven Noble Truths of Fuzzy Logic*, Internet Article, http://freeware.aus.sps.mot.com/freeweb/pub/fuzzy/news5.txt.

[9] N. Soylemezoglu, *The Logic of Fuzziness*, Internet Article, http://www.elsevier.com/cas/estoc/contents/sab/01655876/sz967613.htm.

[10] M. Sugeno, Fuzzy Algorithmic Control of a Model Car by Oral Instructions, Fuzzy Sets and Systems, vol. 32, no. 2, pp. 135-156, 1989.

[11] M. Sugeno (editor), *Industrial Applications of Fuzzy Logic Control*, North-Holland, 1985.

[12] T. Brabazon, *The Advantages of Being Fuzzy*, Accountancy Ireland, vol. 28, no. 5, pp. 40-41, Oct. 1996.

[13] M. Attaran, *The Coming Age of Fuzzy*, Journal of Systems Management, vol. 47, no. 2, pp. 4-12, Mar./Apr. 1996.

[14] D. S. Roger, *A Case for Fuzzy Thinking*, Transportation & Distribution, vol. 37, no. 3, pp. 108-110, Mar. 1996.

[15] K. J. Kim, H. Moskowitz and M. Koksalan, *Fuzzy Versus Statistical Linear Regression*, European Journal of Operational Research, vol. 92, no. 2, pp. 417-434, Jul. 1996.

[16] M. Dohnal, D. M. Fraser and M. Kerkovsky, *A Fuzzy Pooling of Investment Cost Knowledge*, International Journal of Production Economics, vol. 43, no. 2,3, pp. 91-106, Jun. 1996.

[17] M. F. Shipley, A. De Korvin and K. Omer, *A Fuzzy Logic Approach for Determining Expected Values: A Project Management Application*, Journal of the Operational Research, vol. 47, no. 4, pp. 562-569, Apr. 1996.

[18] M. Viswanathan, M. Bergen, S. Dutta and T. Childers, *Does a Single Response Category in a Scale Completely Capture a Response?*, Psychology & Marketing [PSY], vol. 13, no. 5, pp. 457-479, Aug. 1996.

[19] P. M. Stanfield, R. E. King and J. A. Joines, *Scheduling Arrivals to a Production System in a Fuzzy Environment*, European Journal of Operational Research, vol. 93, no. 1, pp. 75-87, Aug. 1996.

[20] M. Kuroda and Z. Wang, *Fuzzy Job Shop Scheduling*, vol. 44, no. 1,2, pp. 45-51, Jun. 1996.

[21] M. A. Coffin and B. W. III Taylor, *Multiple Criteria R&D Project Selection and Scheduling Using Fuzzy Logic*, Computers & Operations Research, vol. 23, no. 3, pp. 207-220, Mar. 1996.

[22] H. J. Ock, *Activity Duration Quantification Under Uncertainty: Fuzzy Set Theory Application*, Cost Engineering, vol. 38, no. 1, pp. 26-30, Jan. 1996.

[23] L-H Chen, C. Kao, S. Kuo, T-Y Wang and Y-C Jang, *Productivity Diagnosis Via Fuzzy Clustering and Classification: An Application to Machinery Industry*, Omega, vol. 24, no. 3, pp. 309-319, Jun. 1996.

[24] M. Grabisch, *The Application of Fuzzy Integrals in Multicriteria Decision Making*, European Journal of Operational Research, vol. 89, no. 3, pp. 445-456, Mar. 1996.

[25] W. G. de Ru and J. H. P. Eloff, *Risk Analysis Modeling with the Use of Fuzzy Logic*, Computers & Security, vol. 15, no. 3, pp. 239-248, 1996.

[26] V. R. Young, *Insurance Rate Changing: A Fuzzy Logic Approach*, Journal of Risk & Insurance, vol. 63, no. 3, pp. 461-484, Sept. 1996.

[27] S. Price, *Identifying Potentially Profitable Credit Card Customers With Artificial Intelligence*, Bank Systems & Technology, vol. 33, no. 5, pp. 18, May 1996.

[28] H-S. Shih, J-Y. Lai and E. S. Lee, *Fuzzy Approach for Multi-Level Programming Problems*, Computers & Operations Research, vol. 23, no. 1, pp. 73-91, Jan. 1996.

[29] T. W. Liao, *A Fuzzy Multicriteria Decision-Making Method for Material Selection*, Journal of Manufacturing Systems, vol. 15, no. 1, pp. 1-12, 1996.

[30] F. Karray, *Applied Fuzziness*, IEEE Spectrum, vol. 33, no. 10, pp. 10-14, Oct. 1996.

[31] P. Narayanaswamy, C. R. Bector and D. Rajamani, *Fuzzy Logic Concepts Applied to Machine-Component Matrix Formation in Cellular Manufacturing*, European Journal of Operational Research, vol. 93, no. 1, pp. 88-97, Aug. 1996.

[32] S. Goonatilake, *Risk Assessment Using Intelligent Systems*, Insurance Systems Bulletin, vol. 11, no. 10, pp. 2-3, Apr. 1996.

[33] B. Grabot, J-C. Blanc and C. Binda, *A decision Support System for Production Activity Control*, Decision Support Systems, vol. 16, no. 2, pp. 87-101, Feb. 1996.

[34] D. Cabello, S. Barro, R. Ruiz, E. L. Zapata and J. Mira, *Fuzzy Clustering Application to the Diagnosis of Ventricular Arrhythmia's*, in Proc. of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, vol. 1, pp. 5-6, New Orleans, LA, USA, 1988.

[35] D. G. Bounds, P. J. Lloyd, B. Mathew and G. Waddell, *A Multilayer Perception Network for the Diagnosis of Low Back Pain*, in Proc. of the IEEE International Conference on Neural Networks, vol. 2, pp. 481-489, San Diego, CA, USA, Jul. 1988.

[36] S. Isaka and A. V. Sebald, *An Adaptive Fuzzy Controller for Blood Pressure Regulation*, in Proc. of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, vol. 6, pp. 1763-1764, Seattle, WA, USA, Nov. 1989.

[37] A. L. Berardinis, *Appliances On-Line*, Machine Design, vol. 68, no. 14, pp. 72-79, Aug. 1996.

[38] ____, *First Fuzzy Logic for Cooktop Controller*, Appliance Manufacturer, vol. 44, no. 2, pp. 86-87, Feb. 1996.

[39] H. Oshima, S. Yasunobu and S-I. Sekino, *Automotive Train Operation Based on Predictive Fuzzy Control*, in Proc. of the International Workshop on Artificial Intelligence for Industrial Applications: IEEE AI'88, pp. 485-489, Hitachi City, Japan, May 1988.

[40] J. A. Bernard, *Use of a Rule-Based System for Process Control*, IEEE Control Systems Magazine, vol. 8, no. 5, pp. 3-13, Oct. 1988.

[41] A. da Rocha, C. K. Morooka and L. Alegre, *Smart Oil Recovery*, IEEE Spectrum, vol. 33, no. 7, pp. 48-51, Jul. 1996.

[42] G. Bortolan, R. Degani and W. Pedrycz, *A Fuzzy Pattern Matching Technique for Diagnostic ECG Classification*, in Proc. of Computers in Cardiology Conference, pp. 551-554, Washington, DC, USA, Sept. 1988.

[43] C. E. Kim, Y. J. Ju and M. Gens, *Multilevel Fault Tree Analysis Using Fuzzy Numbers*, Computers & Operations Research, vol. 23, no. 7, pp. 695-703, Jul. 1996.

[44] H-H. Huang and H-P. B. Wang, *Machine Fault Diagnostics Using a Transputer Network*, Computers & Industrial Engineering, vol. 30, no. 2, pp. 269-281, Apr. 1996.

[45] X. He, *Weighted Fuzzy Logic and its Applications*, in Proc. of COMPSAC 88: the Twelfth International Computer Software and Applications Conference, pp. 485-489, Chicago, IL, USA, Oct. 1988.

[46] Y. Hayashi and M. Nakai, *Efficient Method for Multidimensional Fuzzy Reasoning and its Application to Fault Diagnosis*, in Proc. of the International Workshop on Artificial Intelligence for Industrial Applications: IEEE AI'88, pp. 27-32, Hitachi City, Japan, 1988.

[47] K. Aoyagi, K. Tanemura, H. Matsumoto, Y. Eki and S. Nigawara, *An Expert System for Startup Scheduling and Operation Support in Fossil power Plants*, in Proc. of the International Workshop on Artificial Intelligence for Industrial Applications: IEEE AI'88, pp. 167-172, Hitachi City, Japan, 1988.

[48] G. I. Adamopoulos and C. P. Pappis, *A Fuzzy-Linguistic Approach to a Multi-Criteria Sequencing Problem*, European Journal for Operational Research, vol. 92, no. 3, pp. 628-636, Aug. 1996.

[49] L. O. Hall, M. Friedman and A. Kandel, *On the Validation and Testing of Fuzzy Expert Systems*, IEEE Transactions on Systems, Man and Cybernetics, vol. 18, no. 6, pp. 1023-1027, Dec. 1988.

[50] K. S. Leung and W. Lam, *Fuzzy Concepts in Expert Systems*, Computer, vol. 12, no. 9, pp. 43-56, Sept. 1988.

[51] M. Kurano, M. Yasuda, J-I. Nakagami and Y. Yoshida, *Markov-Type Fuzzy Decision Processes with a Discounted Reward on a Closed Interval*, European Journal of Operational Research, vol. 92, no. 3, pp. 649-662, Aug. 1996.

[52] P. Vieira and F. Gomide, *Computer-Aided Train Dispatch*, IEEE Spectrum, vol. 33, no. 7, pp. 51-53, Jul. 1996.

[53] J-Y. Han and J-L. Han, *Fuzzy Logic Systolic Array for Real Time Approximate Reasoning*, in Proc. of the IEEE International Symposium on Intelligent Control 1988, pp. 628-632, Arlington, VA, USA, 1988.

[54] Y. Dote and B. K. Bose, *Fuzzy CAD for Variable Structure PI(D) Controller*, in Proc. IECON'89: 15th Annual Conference of IEEE Industrial Electronic Society, vol. 4, pp. 705-708, Philadelphia, PA, USA, Nov. 1989.

[55] S. G. Romaniuk and L. O. Hall, *Fuzzy Connectionist Expert systems*, in Proc. IJCNN: International Joint Conference on Neural Networks, vol. 2, pp. 629, Washington, DC, USA, Jun. 1989.

[56] X.-Z. Zeng, H.-D. Sun, J.-J. Ruan, Z.-J. Yuan and C.-Y. Tu, *STRATEGY I: Distributed Military Strategy Expert System on a Microcomputer Ethernet*, in Proc.

of the Twenty-Second Annual Hawaii International Conference on System Sciences, vol. III, pp. 660-668, Kailua-Kona, HI, USA, Jan. 1989.

[57] S. Tsuji, M. Amano and S. Hikita, *Application of the Expert System to Elevator Group-Supervisory Control*, in Proc. of the Fifth Conference on Artificial Intelligence Applications, pp. 287-294, Miami, FL, USA, Mar. 1989.

[58] L. Godo and C. Sierra, *A New Approach to Connective Generation in the Framework of Expert Systems Using Fuzzy Logic*, in Proc. of the Eighteenth International Symposium on Multi-Valued Logic, pp. 157-162, palma de Mallorca, Spain, 1988.

[59] D. Driankov, H. Hellendoorm and M. Reinfrank, *An Introduction to Fuzzy Control*, Springer-Verlag Berlin Heidelberg, USA, 1993.

[60] C. J. Harris and C. G. Moore, *Intelligent Identification and Control for Autonomous Guided Vehicles using Adaptive Fuzzy-Based Algorithms*, Engineering Applications of Artificial Intelligence, vol. 2, pp. 267-285, Dec. 1989.

[61] C. C. Lee, *Fuzzy Logic in Control Systems*, Fuzzy Logic Controller-Part II, IEEE Transactions on Fuzzy Systems, Man and Cybernetics, vol. 20, no. 2, pp. 419-435, 1990.

[62] G.J. Klir and T. A. Folger, *Uncertainty and Information*, Prentice-Hall, USA, 1988.

[63] J. Buckley and W. Siller, *Fuzzy Operations for Possibility Interval Sets*, Fuzzy Sets and Systems, vol. 22, pp. 215-227, 1987.

[64] H. Hellendoorn, *Design and Development of A Fuzzy System at Siemens R&D*, in Proc. of the Second IEEE International Conference on Fuzzy Systems: Fuzzy-IEEE 1993, pp. 1365-1370, San Francisco, CA, USA, Mar. 1993.

[65] T. Takagi and M. Sugeno, *Derivation of Fuzzy Control Rules from Human Operator's Control Actions*, in Proc. of IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, pp. 55-60, Marseilles, France, Jul. 1983.

[66] M. Sugeno and M. Nishida, *Fuzzy Control of Model Car*, Fuzzy Sets and Systems, vol. 16, pp. 103-113, 1985.

[67] M. Sugeno and K. Murakami, *An Experimental Study on Fuzzy Parking Control Using A Model Car*, Industrial Applications of Fuzzy Control, Elsevier Science Publications B.V. (North-Holland), pp. 125-138, 1985.

[68] C. J. Harris and C. G. Moore, *Phase Plane Analysis Tools for a Class of Fuzzy Control Systems*, IEEE Transactions on Fuzzy Systems, vol. 2, no. 1, pp. 511-518, 1992.

[69] R. Jager, H. B. Verbruggen and P. M. Bruijin, *The Rule of Defuzzification Methods in Application of Fuzzy Control*, in Proc. of the IFAC Symposium on Intelligent

Components and Instruments for Control Applications, A. Ollero and E. F. Camacho, the Netherlands, 1992.

[70] L. I. Larkin, *A Fuzzy Logic Controller for Aircraft Control*, Industrial Applications of Fuzzy Control, pp. 87-103, 1985.

[71] W. Pedrycz, *Fuzzy Control and Fuzzy Systems*, Research Study Press/J Wiley and Sons, 1989.

[72] R. R. Yager and D. P. Filev, *A Simple Adaptive Defuzzification Method*, IEEE Transactions on Fuzzy Systems, vol. 1, no. 1, pp. 69-78, 1993.

[73] O. Ghanayem, *An Alternative Reasoning Method*, in Proc. of the Industrial and Engineering Applications of Artificial Intelligence and Expert Systems Conference (IEA/AIE) (invited and additional papers), pp. 41-45, Melbourne, VIC, Australia, Jun. 1995

[74] C. Butur and V. Kasparianm, *Predictive Fuzzy Expert Controllers*, Computer Industrial Engineering, vol. 20, no. 2, 1991.

[75] P. Isomursu and T. Rauma, *A self-Tuning Fuzzy Logic Controller for Temperature Control of Superheated Steam*, in Proc. of the Third IEEE Conference on Fuzzy Systems: IEEE World Congress on Computational Intelligence, vol. 3, pp. 1560-1563, Orlando, FL, USA, Jun. 1994.

[76] D. Wang and T. Chai, *A Self-Tuning Fuzzy Controller Via fuzzy Inference Network*, in Proc. of the 1995 IEEE International Conference on Fuzzy Systems. The International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and the Second International Fuzzy Engineering Symposium, vol. 4, pp. 2201-2206, Yokohama, Japan, Mar. 1995.

[77] F. Ashrafzadeh, E. P. Nowicki and J. C. Salmon, *A Self-Organising and Self-Tuning Fuzzy Logic Controller for Field Oriented Control of Induction Motor Drives*, in Proc. IAS'95: IEEE Industry Applications Conference, Thirteenth IAS Annual Meeting, vol. 2, pp. 1656-1552, Orlando, FL, USA, Oct. 1995.

[78] J.-T. Gau and W. F. Lu, *Application of Self-Organising Fuzzy Logic Controller in 2-D Motion Tracking Control*, in Proc. of the Third IEEE Conference on Fuzzy Systems, IEEE World Congress on Computational Intelligence, vol. 3, pp. 1660-1665, Orlando, FL, USA, Jun. 1994.

[79] Y.-M. Park, U.-C. Moon and K. Y. Lee, *A Self-Organising Fuzzy Logic Controller for Dynamic Systems using a Fuzzy Auto-Regressive Moving Average (FARMA) Model*, IEEE Transactions on Fuzzy Systems, vol. 3, no. 1, pp. 75-82, Feb. 1995.

[80] H. T. Nguyen and D. Sands, *Real-Time Self-Organising Fuzzy Logic Controller for DC Servo*, Fifth European Conference on Power Electronics and Applications, vol. 4, pp. 174-179, Sept. 1993.

[81] B. S. Zhang and J. M. Edmunds, *Self-Organising Fuzzy Logic Controller*, IEE Proceedings D [Control Theory and Applications], vol. 139, no. 5, pp. 460-464, Sept. 1992.

[82] J.-Y. Han and V. McMurray, *Adaptive Fuzzy Logic Controller Based on Parameter Estimation*, in Proc. of the 1992 IEEE International Conference on Systems, Man and Cybernetics, vol. 2, pp. 1645-1650, Chicago, IL, USA, Oct. 1992.

[83] H. Le-Huy, *An adaptive Fuzzy Controller for Permanent-Magnet AC servo Drives*, in Proc. of IAS'95. the 1995 IEEE Industry Applications Conference, Thirtieth Annual Meeting, vol. 1, pp. 104-110, Orlando, FL, USA, Oct. 1995.

[84] L. Pavel and M. Chelaru, *Neural Fuzzy Architecture for Adaptive Control*, in Proc. of the IEEE International Conference on Fuzzy systems, pp. 1115-1122, San Diego, CA, USA, Mar. 1992.

[85] C. L. Karr and E. J. Gentry, *Fuzzy Control of pH using Genetic Algorithms*, IEEE Transactions on Fuzzy Systems, vol. 1, no. 1, pp. 46-53, Feb. 1993.

[86] E. Cox, *Adaptive Fuzzy Systems*, IEEE Spectrum, pp. 27-31, Feb. 1993.

[87] N. Matsunaga and S. Kawaji, *Fuzzy Hybrid Control of DC Servomotor*, Electrical Engineering in Japan, vol. 111, no. 6, pp. 105-111, 1991.

[88] A. Suyitno, J. Fujikawa, H. Kobayashi and Y. Dote, *Variable Structured Robust Controller by Fuzzy Logic for Servomotors*, IEEE Transactions on Industrial Electronics, vol. 40, no. 1, pp. 80-87, 1993.

[89] O. Ghanayem and L. Reznik, *Hierarchical Versus Adaptive Fuzzy Logic Controllers: Design and Performance*, in Proc. Second Australian and New Zealand Conference on Intelligent Information Systems, pp. 224-228, Brisbane, Qld, Australia, Dec. 1994.

[90] C. N. Kim and L. Yun, *Design of Sophisticated Fuzzy Logic Controllers using Genetic Algorithms*, in Proc. of the Third IEEE Conference on Fuzzy Systems. IEEE World Congress on Computational Intelligence, vol. 3, pp. 1708-1712, Orlando, FL, USA, Jun. 1994.

[91] J. Kim and B. P. Zeigler, *Hierarchical Distributed Genetic Algorithms: a Fuzzy Logic Controller Design Application*, IEEE Expert, vol. 11, no. 3, pp. 76-84, Jun. 1996.

[92] G. M. Abdelnour, C.-H. Chang, F.-H. Huang and J. Y. Cheng, *Design of a Fuzzy Controller Using Input and Output Mapping Factors*, IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 5, Sept./Oct. 1991.

[93] P. Albertos, M. Martinez, J. L. Navarro and F. Morant, *Fuzzy Controllers Design: A Methodology*, in Proc. Second IEEE Conference on Control Applications, vol. 1, pp. 69-75, Vancouver, BC, Canada, Sept. 1993.

[94] L. Reznik and J. Shi, *Simulation Study of the Choice of a Membership Function Model for Control Applications*, in Proc. of the Second International Conference on Modeling and Simulation-MS'93. vol. 1, pp. 209-217, Melbourne, VIC, Australia, Jul. 1993.

[95] S. Singh, *Stability Analysis of Discrete Fuzzy Control Systems*, in Proc. IEEE International Conference on Fuzzy Systems, pp. 527-534, San Diego, CA, USA, Mar. 1992.

[96] G. Abdelnour, J. Y. Cheung and C.-H. Chang, *Steady-State Analysis of a Three-Term Fuzzy Controller*, IEEE Transactions on Systems, Man, and Cybernetics, vol. 23, no. 2, Mar./Apr. 1993.

[97] H. Kang, *Stability and Control of Fuzzy Dynamic Systems via Cell-State Transitions in Fuzzy Hypercubes*, IEEE Transactions on Fuzzy Systems, vol. 1, no. 4, pp. 267-279, Nov. 1993.

[98] D. P. Kwok, P. Tam, C. K. Li and P. Wang, *Analysis and Design of Fuzzy PID Control System*, in Proc of the International Conference on Control '91, vol. 2, pp. 955-960, Edinburgh, UK, Mar. 1991.

[99] S. Tzafestas and N. P. Papanikolopoulos, *Incremental Fuzzy Expert PID Control*, IEEE Transactions on Industrial Electronics, vol. 37, no. 5, Oct. 1990.

[100] D. I. Brbaker, *Fuzzy PD+I Control*, Huntinton Technical Brief, Apr. 1994.

[101] K. D. Pham, *Co Generation Application: Interconnection of Induction Generators with Public Electric Utilities*, in Proc. of the 1991 Rural Electric Power Conference, pp. D4/1-7, Dearborn, MI, USA, Apr. 1991.

[102] V. I. Utkin, *Sliding Mode Control Design Principles and Applications to Electric Drives*, IEEE Transactions on Industrial Electronics, vol. 40, no. 1, 1993.

[103] D. E. Schneider and P. P. Wang, *Design of a Fuzzy Controller for a Target Tracking System*, IEEE Transactions on Industrial Electronics, vol. 20, no. 2, 1992.

[104] P. Ramasway, R. M. Edwards and K. Y. Lee, *An Automatic Tuning Method of a Fuzzy Logic Controller for Nuclear Reactors*, IEEE Transactions on Industrial Electronics, vol. 40, no. 4, 1993.

[105] J. Shi, L. H. Herron and A. Kalam, *Comparison of Fuzzy Logic Based and Rule Based Power System Stabiliser*, in Proc. First IEEE Conference on Control Applications, vol. 2, pp. 692-697, Dayton, OH, USA, Sept. 1992.

[106] J. G. Hollinger, R. A. Bergstrom and J. S. Bay, *A Fuzzy Logic Force Controller for a Stepper Motor Robot*, in Proc. of the 1992 IEEE International Symposium on Intelligent Control, pp. 204-209, Glasgow, UK, Aug. 1992.

[107] T. Yamagushi, K. Goto, T. Takagi, K. Doya and T. Mita, *Intelligent Control of a Flying Vehicle using Fuzzy Associated Memory System*, in Proc. of the IEEE

International Conference on Fuzzy Systems, pp. 1139-1149, San Diego, CA, USA, Mar. 1992.

[108] R. Meir, J. Nieuwland, S. Hacisalihzade, D. Steck and A. Zbinden, *Fuzzy Control of Blood Pressure During Anesthesia with Isoflurane*, in Proc. of the IEEE International Conference on Fuzzy Systems, pp. 981-987, San Diego, CA, USA, Mar. 1992.

[109] Y. Egusa, H. Akahori, A. Morimura and N. Wakami, *An Electronic Video Camera Image Stabiliser Operated on Fuzzy Theory*, in Proc. IEEE International Conference on Fuzzy Systems, pp. 851-858, San Diego, CA, USA, Mar. 1992.

[110] H. Eichfeld, M. Lohner and M. Muller, *Architecture of a CMOS Fuzzy Logic Controller with Optimised Memory Organisation and Operator Design*, in Proc. IEEE International Conference on Fuzzy Systems, pp. 1317-1323, San Diego, CA, USA, Mar. 1992.

[111] T. Yamakawa, *A Fuzzy Programmable Logic Array (Fuzzy PLA)*, in Proc. IEEE International Conference on Fuzzy Systems, pp. 459-465, San Diego, CA, USA, Mar. 1992.

[112] M. A. Manzoul and D. Jayabharathi, *Fuzzy Controller on FPGA Chip*, in Proc. IEEE International Conference on Fuzzy Systems, pp. 1309-1316, San Diego, CA, USA, Mar. 1992.

[113] K. Ozawa, K. Hirota, L. T. Koczy, W. Pedrycz and N. Ikoma, *Summary of Fuzzy Flip-Flop*, in Proc. The International Joint Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium, vol. 3, pp. 1641-1648, Yokohama, Japan, Mar. 1995.

[114] J. Chen and M. J. Patyra, *VHDL Modeling of a Multivariable Fuzzy Logic Controller Hardware System*, in Proc. Third IEEE Conference on Fuzzy Systems. IEEE World Congress on Computational Intelligence, vol. 1, pp. 129-132, Orlando, FL, USA, Jun. 1994.

[115] M. Sasaki and F. Ueno, *A VLSI Implementation of Fuzzy Logic Controller using Current Mode CMOS Circuits*, in Proc. IFIS'93: the Third international Conference on Industrial Fuzzy Control and Intelligent Systems, pp. 215-220, Houston, TX, USA, Dec. 1993.

[116] P. Kundor, *Power System Stability and Control*, McGraw-Hill, NY, USA, 1994.

[117] ANSI/IEEE Std. 421.1-1986 (Revision of IEE Std. 421-1972).

[118] N. B. Bhatt, A. G. DeGroff, M. Heyeck, S. L. Ridenbaugh and R. P. Schultz, *Benefits of Excitation Control System Testing at AEP's Rockport Plant*, IEEE Transactions on Energy Conversion, vol. 6, no. 1, pp. 21-28, Mar. 1991.

[119] J. R. Ribeiro, *Minimum Excitation Limiter Effect on Generator Response to System Disturbances*, IEEE Transactions on Energy Conversion, vol. 6, no. 1, pp. 29-38, Mar. 1991.

[120] P. Demello and C. Concordia, *Concepts of Synchronous Machine Stability as Affected by Excitation Control*, IEEE Transactions on Power Apparatus and Systems, vol. PAS-88, no. 4, pp. 316-324, Apr. 1969.

[121] Z. O. Yonah, A. M. El-Serafi and A. E. Krause, *Performance of a Computer-Based Two-Phase Excitation System for a Variable-Speed Dual-Excited Synchronous Generator*, in Proc. of the IEEE WESCANEX 93. Communications, Computers and Power in the Modern Environment Conference, pp. 263-268, Saskatoon, Sask, Canada, May 1993.

[122] J. W. Chapman and M. D. Ilic, *Some Robustness Results for Feedback Linearizing Control of Generator Excitation*, in Proc. of the 31st Conference on Decision and Control, vol. 1, pp. 1123-1128, Tucson, AR, USA, Dec. 1992.

[123] M. M. Begovic and A. G. Phadke, *Control of Voltage Stability using Sensitivity Analysis*, IEEE Transactions on Power Systems, vol. 7, no. 1, pp. 114-123, Feb. 1992.

[124] M. K Pal, *Voltage Stability Conditions Considering Load Characteristics*, IEEE Transactions on Power Systems, vol. 7, no. 1, pp. 243-249, Feb. 1992.

[125] N. Yorino, H. Sasaki, Y. Masuda, Y. Tamura, M. Kitagawa and A. Oshimo, *An Investigation of Voltage Stability Problems*, IEEE Transactions on Power Systems, vol. 7, no. 2, pp. 600-611, May 1992.

[126] T. L. Huang, S. C. Chen, T. Y. Hwang and W. T. Yang, *Power System Output Feedback Stabiliser Design via Optimal Subeigen structure Assignment*, Electric Power Systems Research, vol. 21, no. 2, pp. 107-1114, Jun. 1991.

[127] C. J. Wu and C. H. Wang, *Damping of Power System Oscillation by a Superconducting Magnetic Energy Storage Unit*, Electric Power Systems Research, vol. 22, no. 1, pp. 19-26, Sept. 1991.

[128] T. C. Yang and N. Munro, *Power System Stabiliser Design Based on the Pole Assignment for SIMO Systems*, International Journal of Electrical Power & Energy Systems, vol. 13, no. 6, pp. 298-302, Dec. 1991.

[129] D. J. Trudnowski, J. R. Smith, T. A. Short and D. A. Pierre, *An application of Pony Methods in PSS Design for Multimachine Systems*, IEEE Transactions on Power Systems, vol. 6, no. 1, pp. 118-126, Feb. 1991.

[130] G. Radman and Y. Smaili, *Performance Evaluation of PID Power System Stabiliser for Synchronous Generator*, IEEE, pp. 597-601, 1988

[131] Z. Ao, T. S. Sidhu and R. J. Fleming, *Stability Investigation of Longitudinal Power System and its Stabilisation by Coordinated Application of Power System*

Stabilisers, IEEE Transactions on Energy Conversion, vol. 9, no. 3, pp. 466-474, Sept. 1994.

[132] G. K. Morison, B. Gao and P. Kundur, *Voltage Stability Analysis using Static and Dynamic Approaches*, IEEE Transactions on Power Systems, vol. 8, no. 3, pp. 1159-1171, Aug. 1993.

[133] M. L. Kothari, J. Nanda and K. Bhattacharya, *Discrete Mode Power System Stabilisers*, IEE Proceedings-C, vol. 140, no. 6, Nov. 1993.

[134] K. Hirayama, Y. Tone, K. Takagi, H. Murakami, M. Shibata, H. Nagamura and Y. Takagi, *Digital AVR Application to Power Plants*, IEEE Transactions on Energy Conversion, vol. 8, no. 4, pp. 602-609, Dec. 1993.

[135] A. A. Ghandakly and J. J. Dai, *An Adaptive Synchronous Generator Stabiliser Design by Generalised Multivariable Pole Shifting Technique*, IEEE Transactions on Power Systems, vol. 7, no. 3, pp. 1239-1244, Aug. 1992.

[136] C. Mao and O. P. Malik, *An Adaptive Optimal Controller and its Application to an Electric Generating Unit*, International Journal of Electrical Power and Energy Systems, vol. 15, no. 3, pp. 169-178, Feb. 1993.

[137] A. A. Ghandakly and A. M. Farhoud, *A Parametrically Optimised Self-Tuning Regulator for Power System Stabilisers*, IEEE Transactions on Power Systems, vol. 7, no. 3, pp. 1245-1250, Aug. 1992.

[138] J. Y. Fan, T. H. Ortmeyer and R. Mukundan, *Power System Stability Improvement with Multivariable Self-Tuning Control*, IEEE Transactions on Power Systems, vol. 5, no. 1, pp. 227-234, Feb. 1990.

[139] A. Rubaai, *Transient Stability Control: a Multi-Level Hierarchical Approach*, IEEE Transactions on Power Systems, vol. 6, no. 1, pp. 262-268, Feb. 1991.

[140] W. E. Norum and K. E. Bollinger, *Lab. and Field Tests of a Self-Tuning Power System Stabiliser*, IEEE Transactions on Energy Conversion, vol. 8, no. 3, pp. 476-483, Sept. 1993.

[141] H. Mori, Y. Tamaru and S. Tsuzuki, *An Artificial Neural-Net Based Technique for Power System Dynamic Stability with Kohonen Model*, IEEE Transactions on Power Systems, vol. 7, no. 2, pp. 856-864, May 1992.

[142] P. K. Kalra, A. Srivastava and D. K. Chaturvedi, *Possible Applications of Neural Nets to Power System Operation and Control*, Electric Power System Research, vol. 25, no. 2, pp. 101-110, Nov. 1992.

[143] Y. Y. Hsu and C. R. Chen, *Tuning of Power System Stabilisers using Artificial Neural Networks*, IEEE Transactions on Energy Conversion, vol. 6, no. 4, pp. 612-619, Dec. 1991.

[144] Y. Zhang, G. P. Chen, O. P. Malik and G. S. Hope, *An Artificial Neural Network Based Adaptive Power System Stabiliser*, IEEE Transactions on Energy Conversion, vol. 8, no. 1, pp. 71-77, Mar. 1993.

[145] H. Bissig, K. Reichert and T. S. Kulig, *Modeling and Identification of Synchronous Machine, a New Approach with an Extended Frequency Range*, IEEE Transactions on Energy Conversion, vol. 8, no. 2, Jun. 1993.

[146] E. C. Shaffer and C. A. Gross, *Methods for Determinig Linear Synchronous Machine Parameters*, in Proc. of the 26th Southeastern Symposium on Systems Theory, pp. 411-415, Athens, OH, USA, Mar. 1994.

[147] A. Walton and J. S. Croft, *The Modeling of Synchronous Machines*, IEEE, pp. 187-191, 1992.

[148] K. Liu, S. Subbarayan, R. R. Shoults, M. T. Manry, C. Kwan, F. I. Lewis and J. Naccarino, *Comparison of Very Short-Term Load Forecasting Techniques*, IEEE Transactions on Power Systems, vol. 11, no. 2, pp. 877-882, May 1996.

[149] A. G. Parlos, E. Oufi, J. Muthusami, A. D. Patton and A. F. Atiya, *Development of An Intelligent Long-Term Electric Load Forecasting System*, in Proc. ISAP'96, International Conference on Intelligent Systems Applications to Power System Proceedings, pp. 288-292, Orlando, FL, USA, Feb. 1996.

[150] D. Srinivasan, C. S. Chang and A. C. Liew, *Demand Forecasting Using Neural Computation, with Special Emphasis on Weekend and Public Holiday Forecasting*, IEEE Transactions on Power Systems, vol. 10, no. 4, pp. 1897-1903, Nov. 1995.

[151] P. K. Dash, G. Ramakrishna, A. C. Liew and S. Rahman, *Fuzzy Neural Networks for Time-Series Forecasting of Electric Load*, IEE Proceedings-Generation, Transmission and Distribution, vol. 142, no. 5, pp. 535-544, Sept. 1995.

[152] D. Srinivasan, A. C. Liew and C. S. Chang, *Forecasting Daily Load Curves Using A Hybrid Fuzzy-Neural Approach*, IEE Proceedings- Generation, Transmission and Distribution, vol. 141., no. 6, pp. 561-577, Nov. 1994.

[153] J. Haddad, G. Lambert-Torres, C. I. A. Costa and G. M. Jannuzzi, *A Fuzzy Logic Decision-Making Approach for Industrial Power System Planning*, in Proc. of the 1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century, vol. 4, pp. 3244-3248, Vancouver, BC, Canada, Oct. 1995.

[154] M. Djukanovic, B. Babic, B. Milosevic, D. J. Sobajic and P. Yoh-Han, *Fuzzy Linear Programming Based Optimal Fuel Scheduling Incorporating Blending/Transloading Facilities*, IEEE Transactions on Power Systems, vol. 11, no. 2, pp. 1017-1023, May 1996.

[155] H.-T. Yang, C.-M. huang, H.-M. Lee and C.-L. Huang, *Multiobjective Power Dispatch using Fuzzy Linear Programming*, in Proc. of the 1995 International

Conference on Energy Management and Power Delivery, vol. 2, pp. 738-743, Singapore, Nov. 1995.

[156] K. Bhattacharyya and M. L. Crow, *A Fuzzy Logic Based Approach to Direct Load Control*, IEEE Transactions on Power Systems, vol. 11, no. 2, pp. 708-714, May 1996.

[157] J. Heydeman, R. Reijntjes, R. Babuska, U. Kaymak and H. R. Lemke, *Fuzzy Logic Based Security Assessment for Power Networks*, in Proc. of the International Conference on Intelligent Systems Applications to Power Systems, pp. 405-409, Orlando, FL, USA, Feb. 1996.

[158] R. Khare and R. D. Christie, *On Designing a Better Operator Assistant for Emergency Control in Power Systems*, in Proc. of The International Conference on Intelligent Systems Applications to Power Systems, pp. 362-366, Orlando, FL, USA, Feb. 1996.

[159] Y.-H. Pao, *Process Monitoring and Optimising for Power Systems Applications*, in Proc. of 1994 IEEE International Conference on Neural Networks. IEEE World Congress on Computational Intelligence, vol. 6, pp. 3697-3702, Orlando, FL, USA, Jul. 1994.

[160] V. N. A. L. da Silva and R. S. Zubulum, *An Integration of Neural Networks and Fuzzy Logic for Power Systems Diagnosis*, in Proc. of the International Conference on Intelligent Systems Applications To Power Systems, pp. 237-241, Orlando, FL, USA, Feb. 1996.

[161] M.-Y. Chow and P. V. Goode, *The Advantages and Challenges of Machine Fault Detection Using Artificial Neural Network and Fuzzy Logic*, in Proc. of the 36th Midwest Symposium on Circuits and Systems, vol. 1, pp. 708-711, Detroit, MI, USA, Aug. 1993.

[162] A. T. P. So, W. L. Chan, C. T. Tse and K. K. Lee, *Fuzzy Logic Based Automatic Diagnosis of Power Apparatus by Infrared Imaging*, in Proc. of the Second International Forum on Applications of Neural Networks to Power Systems, pp. 187-192, Yokohama, Japan, Apr. 1993.

[163] K. A. El-Metwally and O. P. Malik, *Fuzzy Logic Power System Stabiliser*, IEE Proceedings-Generation, Transmission and Distribution, vol. 142, no. 3, pp. 277-281, May 1995.

[164] P. Hoang and K. Tomsovic, *Design and Analysis of an Adaptive Fuzzy Power System Stabiliser*, IEEE Transactions on Energy Conversion, vol. 11, no. 2, pp. 455-461, Jun. 1996.

[165] T. Hiyama, *Application of Rule-Based Stabilising Controller to Electrical Power System*, IEE Proceedings, vol. 136, Pt. C, no. 3, pp. 175-181, May 1989.

[166] T. Hiyama, *Robustness of Fuzzy Logic Power System Stabilisers Applied to Multimachine Power System*, IEEE Transactions on Energy Conversion, vol. 9, no. 3, pp. 451-459, Sept. 1994.

[167] T. Hiyama, S. Oniki and H. Nagashima, *Experimental Studies on Micro-Computer Based Fuzzy Logic Power System Stabiliser*, in Proc. of the Second International Forum on Applications of Neural Networks to Power Systems, pp. 212-217, Yokohama, Japan, Apr. 1993.

[168] M. A. M. Hassan and O. P. Malik, *Laboratory Evaluation and Test Results for Rule-Based and Fuzzy Logic-Based Power System Stabilisers: a Comparative Study*, Intelligent Systems Engineering, vol. 2, No. 1, pp. 52-60, Spring 1993

[169] T. Hiyama, *Real time Control of Micro-Machine System using Micro-Computer Based Fuzzy Logic Power System Stabiliser*, IEEE Transactions on Energy Conversion, vol. 9, no. 4, Dec. 1994.

[170] T. Hiyama, S. Oniki and H. Nagashima, *Evaluation of Advanced Fuzzy Logic PSS on Analog Network Simulator and Actual Installation on Hydro Generators*, IEEE Transactions on Energy Conversion, vol. 11, no. 1, Mar. 1996.

[171] T. H. Ortmeyer and T. Hiyama, *Frequency Response Characteristics of the Fuzzy Polar Power System Stabiliser*, IEEE Transactions on Energy Conversion, vol. 10, no. 2, Jun. 1995.

[172] P. K. Dash and A. C. Liew, *Anticipatory Fuzzy Control of Power Systems*, IEE Proceedings-Generation, Transmission and Distribution, vol. 142, no. 2, pp. 211-218, Mar. 1995.

[173] T. Hiyama, *Application of Neural Network to Real Time Tuning of Fuzzy Logic PSS*, in Proc. of the Second International Forum on Applications of Neural Networks to Power Systems Control, Operation and Management, pp. 421-426, Hong Kong, Dec. 1993.

[174] R. M. Draper and R. L. King, *Fuzzy Controller for the Exciter of a Salient-Pole Synchronous Generator*, in Proc. of the 26th Southeastern Symposium on Systems Theory, pp. 177-180, Athens, OH, USA, Mar. 1994.

[175] E. Handschin, W. Hoffmann, F. Reyer, Th. Stephanblome, U. Schlucking, D. Westermann and S. S. Ahmed, *A New Method for Excitation Control Based on Fuzzy Set Theory*, IEEE Transactions on Power Systems, vol. 9, no. 1, pp. 533-539, Feb. 1994.

[176] T. Hiyama, K. Miyazaki and H. Satoh, *A Fuzzy Logic Excitation System for Stability Enhancement of Power Systems with Multi-Mode Oscillations*, IEEE Transactions on Energy Conversion, vol. 11, no. 2, pp. 449-454, Jun. 1996.

[177] R.P. Copeland and K.S. Rattan, *A Fuzzy Logic Supervisor for PID Control of Unknown Systems*, 1994 IEEE International Symposium on Intelligent Control, pp. 22-26, Columbus, Ohio, USA, Aug. 1994.

[178] P. M. Anderson and A.A. Fouad, *Power System Control and Stability*, Iowa State University Press, Ames, Iowa, USA, 1977.

[179] IEEE Guide: Test Procedures for Synchronous machines, IEEE Std 115-1983, (revision of IEEE Std 115-1965)

APPENDICES

APPENDIX A

L/R COG

Left/Right Center Of Gravity (L/R COG)

The Left-Right Centre Of Gravity (L/R COG) seeks a compromise between additional knowledge (precision) and processing calculations (time). This is achieved by the use of positional information already available from the input. The method treats every single convex membership function (class) in the universe of discourse (input/output) as two non-intersected classes separated by a singleton at the prototype of the class, as in figure *A*.1 (L: left of the prototype, C: the prototype and R: right of the prototype).

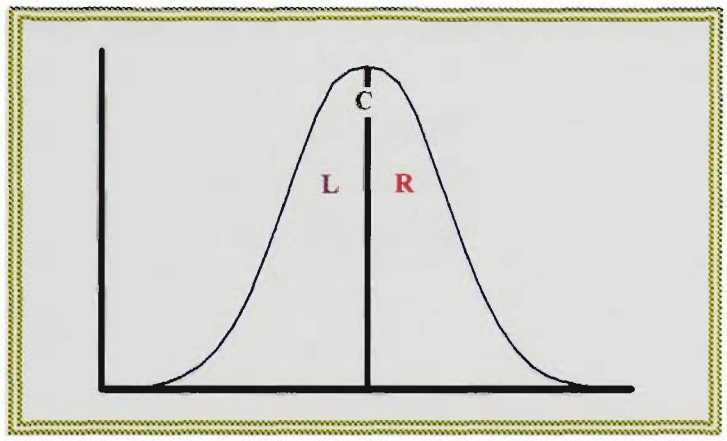


Figure *A*.1 convex membership function

This division of the class allows L/R COG to ensure a unique output for every input in the class. The way this method operates is explained for a SISO for which figure *A*.2 shows the input and output classes. The rule table is shown in table *A*.1. The following explains L/R COG in detail.

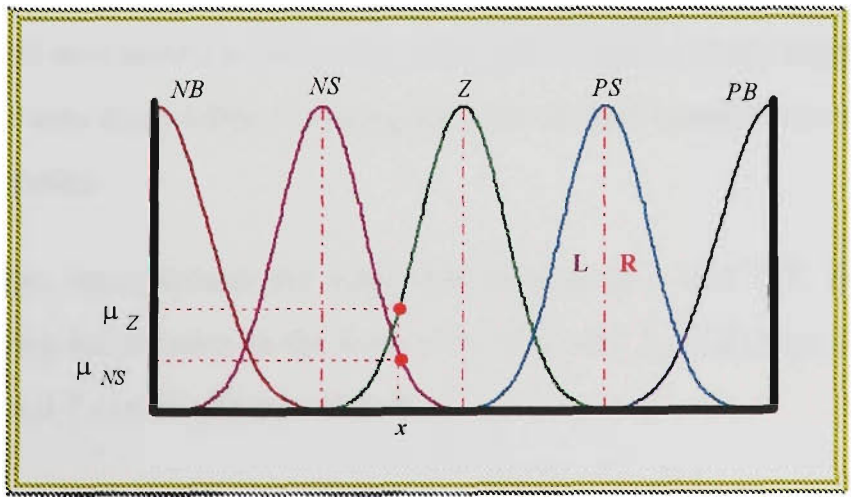


Figure *A*.2 input/output classes

INPUT	*NB*	*NS*	*Z*	*PS*	*PB*
OUTPUT	*NB*	*NS*	*Z*	*PS*	*PB*

Table *A*.1 SISO system rule table

Fuzzification:

Fuzzification in the L/R COG is related to the fact that the input at x is to the **R**ight of the input class NS and the **L**eft of the Z class. To consider this positional information L/R COG makes the association

$$x \Rightarrow \{Set_i, \mu_i\{x\}, S_i\}$$

where S_i is the input class side and P is the prototype of the class, and

$$S_i = \begin{cases} L & x < P \\ C & x = P \\ R & x > P \end{cases}$$

Fuzzy Processing:

To make use of the positional information provided by fuzzification, the inference uses an extended rule base expressing the mapping of S_i to S_o (input class side to output class side). For a SISO fuzzy system, the sides rules can be expressed as in table *A*.2.

S_I	*L*	*C*	*R*
S_O	*L*	*C*	*R*

Table *A*.2 sides rule table

The sides rule table can be treated in the same manner as the classes rule table, and one may have different setting for both of them based on the system under consideration.

For a two input system the sides rule table is of a size 3x3, ie. Table *A*.3, expressing information in the form if S_{i1} is X_1 and S_{i2} is X_2 then S_o is Y, where X_1, X_2 and Y can be either L, C or R.

		S_{I1}		
		L	*C*	*R*
S_{I2}	*L*	*L*	*C*	*C*
	C	*C*	*C*	*C*
	R	*C*	*C*	*R*

Table *A*.3 sides rule table for a two input one output FLC

As mentioned above, the knowledge in the sides rule table varies according to the type of system under consideration. The size of the table depends on the number of inputs based on the fixed division of the class shown in figure *A*.1, which for n inputs is 3^n.

The rule base used in this section comprises of two rule tables, one describing the class mapping and the other describing the sides mapping. Both operating in parallel. For each particular input an even number of rules will fire at the same time. The fuzzy output resulting from the inference process is expressed by $\{Set_o, \mu_o(x), S_o\}$.

Defuzzification:

The defuzzification phase in the L/R COG differs from the classical defuzzification in considering the parameter S_o. The output class is subjected to the same treatment as the input class, (divided into three non-intersected classes as in figure *A*.1). L/R COG uses the centre of gravity for defuzzification (at this point other defuzzification methods can be used). If an inference process results in a fuzzy output of the form $\{S, 0.5, L\}$ the L/R COG would result in a crisp output which is the COG of the shaded area in figure *A*.3.

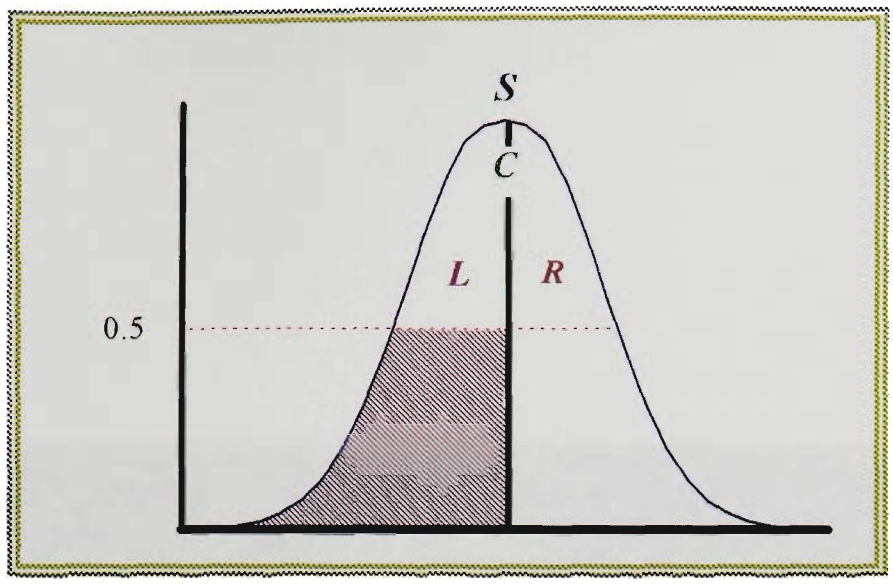


Figure *A*.3 defuzzification area selected by the L/R COG

Figure *A*.4 explains the overall fuzzy processing using the L/R COG method for the input at x.

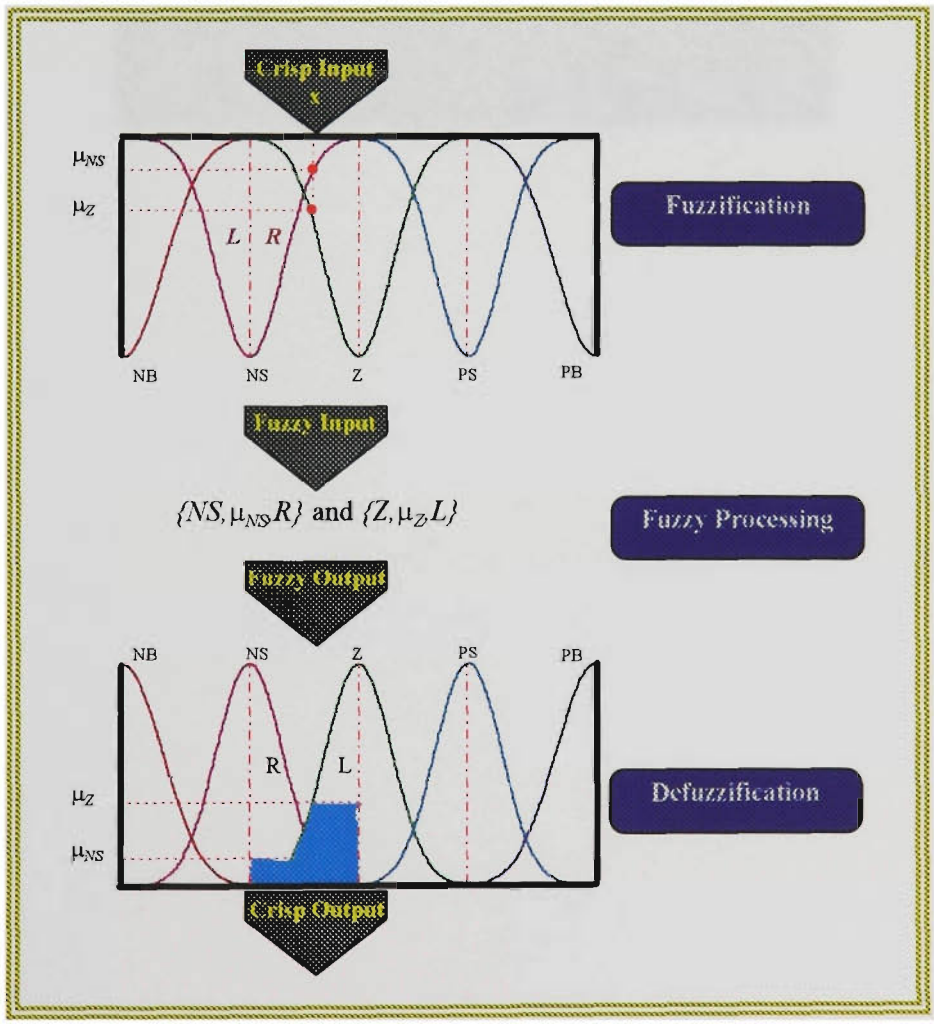


Figure *A*.4 L/R COG fuzzy processing

APPENDIX B.1

PC-μ_p Driver Source Code

PC-μ_p Driver Source Code

```
/*******************
  Header Files
********************/
        #include <stdlib.h>
        #include <stdio.h>
        #include <time.h>
        #include <conio.h>
        #include "c4xapp.h"
        #include "dpram.h"
        #include "declare3.h"
        #include "timer.h"

/**********************
  Define Constants
**********************/
        #define  SITE_A_C40_FILE  "hussein.out"

/**********************
define DPRAM locations
**********************/
        #define  DPRAM_BUFFER1_START  0xA0000200L
        #define  BUFFER1_LENGTH      0x100
        #define     Start_DSP_Prog              DPRAM_HANDSHAKE0
        #define     PC_Ready_to_Recieve         DPRAM_HANDSHAKE1
        #define     PC_In_Data_Ready   DPRAM_HANDSHAKE2
        #define     Sample_Start                DPRAM_HANDSHAKE3
        #define Sample_End             DPRAM_HANDSHAKE4
        #define Relay_On               DPRAM_HANDSHAKE5
        #define bb 65535

/**********************
  Function ProtoTypes
***********************/
        void checkReturnCode(UINT returnCode);

/******************
  Main Program
*******************/
void main()
        {
        UINT      ret;
        ULONG     dataReady,DSPReady;
        ULONG     zero;
        ULONG     one;

        PROC_ID    *handle;
        char       proc_name[MAX_PROC_NAME_SIZE]="CPU_A",FileName[40];
        float      i,data_buf[3],SampleRate;
        float      RunTime,TTm=0,R1On,R1Off,R2On,R2Off,SaveOn;
        float      Nw_st,Nw_End,Nw_Ref,scOff=0,sv_pnt;
        int        Ref_Flg=0,Load_Flg=0,Sc_Flg=0,Dst_Type;
        float      count=0 ,CNTRL;
        int        keyboardHit;
        unsigned int Smpl_Diff;
        unsigned long int start_p,end_p,ellaps_p;
        unsigned long int start_rl,elps_rl;
        unsigned long int Smpl_St,Smpl_End,Smpl_Elps;
        unsigned long int elps_sc=0;
        FILE          *ofp;
        system("cls");
        zero=0;
        one=1;
        printf("\n\n\n ENTER RUN TIME: ");
        scanf("%f",&RunTime);
        data_buf[0]=RunTime;
        printf("\n ENTER SAMPLING TIME: ");
        scanf("%f",&SampleRate);
        data_buf[1]=SampleRate;
        printf("\n ENTER initial tacho reading: ");
```

```
scanf("%f",&tch_1st);
data_buf[2]=tch_1st;
Tachok=tch_1st;
data_buf[3]=Tachok;
printf("\n enter disturbance type : ");
printf("\n   1) Change Reference Voltage");
printf("\n   2) Load change");
printf("\n   3) Short circuit (Tr. L.)");
printf("\n            choose.....");
scanf("%d",&Dst_Type);
data_buf[4]=Dst_Type;
switch (Dst_Type)
            {
            case 1:
                        {
                         printf ("\n ********** Ref Volt ********");
                         printf("\n Input New reference voltage : ");
                         scanf("%f",&Nw_Ref);
                         data_buf[5]=Nw_Ref;
                         printf("\n Input Disturbance start time :");
                         scanf("%f",&Nw_st);
                         data_buf[6]=Nw_st;
                         printf("\n Input Disturbance stop time :");
                         scanf ("%f",&Nw_End);
                         data_buf[7]=Nw_End;
                         Ref_Flg=1;
                         data_buf[8]=Ref_Flg;
                         break;
                         }
            case 2:
                        {
                        printf ("\n ********** load change *********");
                        printf("\n ENTER RELAY_1 On TIME : ");
                        scanf("%f",&R1On);
                        data_buf[5]=R1On;
                        printf("\n ENTER RELAY_1 Off TIME : ");
                        scanf("%f",&R1Off);
                        data_buf[6]=R1Off;
                        printf("\n ENTER RELAY_2 On TIME : ");
                        scanf("%f",&R2On);
                        data_buf[7]=R2On;
                        printf("\n ENTER RELAY_2 Off TIME : ");
                        scanf("%f",&R2Off);
                        data_buf[8]=R2Off;
                        Load_Flg=1;
                        data_buf[9]=Load_Flg;
                        break;
                        }
            case 3:
                        {
                        printf("\n ********** short circuit ********");
                        printf("\n ENTER RELAY (short circuit, sec) On TIME : ");
                        scanf("%f",&R1On);
                        data_buf[5]=R1On;
                        printf("\n ENTER RELAY (short circuit, m_sec) duration : ");
                        scanf("%f",&scOff);
                        data_buf[6]=scOff;
                        printf("\n scOff=%f",scOff);
                        Sc_Flg=1;
                        data_buf[7]=Sc_Flg;
                        break;
               }
               }
printf("\n Enter INITIAL Global Input Range ");
scanf("%f",&G_In_Rng);
data_buf[10]=G_In_Rng;
printf("\n Enter INITIAL Global Output Range ");
scanf("%f",&G_Out_Rng);
data_buf[11]=G_Out_Rng;
printf("\n Apply Learning yes ==>1 ");
scanf("%f",&Learn);
data_buf[12]=Learn;
```

```
        if (Learn==1)
                {
                printf("\n ENTER Learning Rate: ");
                scanf("%f",&Learn_T);
                data_buf[13]=Learn_T;
                }
        printf("\n control (y==1;no==0): ");
        scanf("%f",&CNTRL);
        data_buf[14]=CNTRL;
        printf("\n enter data file name ");
        scanf("%40s",FileName);
        ofp=fopen(FileName,"w");
        printf("\nRebooting the C40 network ....");
        ret = Global_Network_Reboot();
        checkReturnCode(ret);
        printf("OK");
        printf("\nOpening processor .....");
        ret=Open_Processor_ID(&handle,proc_name,NULL);
        checkReturnCode(ret);
        printf("OK");
        ret=Change_Board_Register_Fields(handle,CONFIG_REG,0,DPC_DB_TIM_0_SEL);
        checkReturnCode(ret);
        ret=Change_Board_Register_Fields(handle,CONFIG_REG,DPC_DB_TIM_1_SEL,0);
        checkReturnCode(ret);
        ret=Change_Board_Register_Fields(handle,CONFIG_REG,DPC_DB_MCLK_0_SEL,0);
        checkReturnCode(ret);
        ret=Change_Board_Register_Fields(handle,CONFIG_REG,DPC_DB_MCLK_1_SEL,0);
        checkReturnCode(ret);

        ret = Write_DPRAM_Words_32(handle,Start_DSP_Prog,1,&zero);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,PC_Ready_to_Recieve,1,&zero);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,PC_In_Data_Ready,1,&zero);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,Start_Sample,1,&zero);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,Sample_End,1,&zero);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,Relay_On,1,&zero);
        checkReturnCode(ret);
        printf("\nLoading program to C40 in Site A.....");
        ret=Load_And_Run_File_DPRAM(handle,SITE_A_C40_FILE);
        checkReturnCode(ret);
        printf("OK");
        ret = Write_DPRAM_Floats_32(handle,DPRAM_BUFFER1_START,15,data_buf);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,Start_DSP_Prog,1,&one);
        checkReturnCode(ret);

/*************************
start the processing
*************************/
        init8h(1000);
        ticks_8h=0;
        TTm=0;
        start_p=ticks_8h;
        ellaps_p=0;
        Smpl_St=ticks_8h;
        ret = Write_DPRAM_Words_32(handle,Sample_Start,1,&one);
        checkReturnCode(ret);
        ret = Write_DPRAM_Words_32(handle,PC_Ready_to_Recieve,1,&one);
        checkReturnCode(ret);
        while(ellaps_p<RunTime)
                {
                ret = Write_DPRAM_Words_32(handle,PC_Ready_to_Recieve,1,&one);
                checkReturnCode(ret);

/*************************
 sampling time calculations
*************************/
                 Smpl_End=ticks_8h;
```

```
                Smpl_Elps=time8h(Smpl_St,Smpl_End);
                if (Smpl_Elps<SampleRate)
                        {
                        Smpl_Diff=abs(Smpl_Elps-SampleRate);
                        delay8h(Smpl_Diff);
                        }
                ret = Write_DPRAM_Words_32(handle,Sample_End,1,&one);
                checkReturnCode(ret);

/*******************
Check the relays
*******************/
                start_rl=ticks_8h;
                elps_rl=time8h(start_p,start_rl);
                elps_sc=elps_rl;
                elps_rl=elps_rl/1000;
                if (Ref_Flg==1)
                        {
                        if ((elps_rl>=Nw_st) && (elps_rl<Nw_End))
                                V_Ref_sg=Nw_Ref;
                        if (elps_rl>=Nw_End)
                                V_Ref_sg=1;
                        }
                else if (Load_Flg==1)
                        {
                        if (elps_rl<R1On)
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&zero);
                                checkReturnCode(ret);
                                }
                        else if ((elps_rl>=R1On)&&(elps_rl<R1Off))
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&one);
                                checkReturnCode(ret);
                                }
                        else if ((elps_rl>=R1Off)&&(elps_rl<R2On))
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&zero);
                                checkReturnCode(ret);
                                }
                        else if ((elps_rl>=R2On)&&(elps_rl<R2Off))
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&one);
                                checkReturnCode(ret);
                                }
                        else
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&zero);
                                checkReturnCode(ret);
                                }
                        }
                else if (Sc_Flg==1)
                        {
                        if ((elps_rl>=R1On)&&(elps_sc<=((R1On*1000)+scOff)))
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&one);
                                checkReturnCode(ret);
                                }
                        if (elps_sc>=((R1On*1000)+scOff))
                                {
                                ret = Write_DPRAM_Words_32(handle,Relay_On,1,&zero);
                                checkReturnCode(ret);
                                }
                        }

                Smpl_St=ticks_8h;
                ret = Write_DPRAM_Words_32(handle,Sample_Start,1,&one);
                checkReturnCode(ret);
                do
                        {
                        ret = Read_DPRAM_Words_32(handle,PC_In_Data_Ready,1, &dataReady);
                        checkReturnCode(ret);
```

```
                    keyboardHit = kbhit();
                    }
                while( (dataReady != 1) && (!keyboardHit) );
                ret = Write_DPRAM_Words_32(handle,PC_In_Data_Ready,1,&zero);
                checkReturnCode(ret);

/***************************************
reading the inputs and processing it
***************************************/
                ret = Read_DPRAM_Floats_32(handle,DPRAM_BUFFER1_START,5,data_buf);
                checkReturnCode(ret);
                for (i=0;i<5;i++)
                        fprintf(ofp,"%7.4f  ",data_buf[i]);
                fprintf(ofp,"\n");
                end_p=ticks_8h;
                ellaps_p=time8h(start_p,end_p);
                TTm=(float)(ellaps_p);
                TTm=TTm/1000;
                ellaps_p=ellaps_p/1000;
        }
        clrscr();
        ret=Close_Processor_ID(handle);
        checkReturnCode(ret);
        Clear_All_Lib_Memory();
        }
```

APPENDIX B.2

Timer Source Code

Timer Source Code

```
#include <dos.h>
#include <bios.h>
#include <conio.h>
#include "tcint8.h"
#ifdef __cpluspls
# define __CPPARGS ...
#else
# define __CPPARGS void
#endif

char        *get_cmostime(void);
void interrupt  new8h(__CPPARGS);
void        init8h(unsigned int);
void        quit8h(void);
unsigned long   time8h(unsigned long, unsigned long);
void        delay8h(unsigned int);
void        sound8h(int, int);

extern volatile unsigned long int ticks_8h;

#define IRQ0 0x8
#define PIT0 0x40
#define PIT1 0x41
#define PIT2 0x42
#define PITMODE 0x43
#define PITCONST 1193180L
#define PIT0DEF 18.2067597
#define KBCTRL 0x61
#define NEW8H 1

static float    tick_per_ms = 0.0182068;
static float    ms_per_tick = 54.9246551;
static float    freq8h = 18.2067597;
static unsigned char flag8h = 0;
static void     interrupt(*old8h) (__CPPARGS);

volatile int    counter_8h;
volatile int    counter_reset;
volatile unsigned long int ticks_8h;

void        init8h(unsigned int Hz)
{
  unsigned int    pit0_set,
              pit0_value;

  if (flag8h != NEW8H) {
    disable();
    old8h = getvect(IRQ0);
    setvect(IRQ0, new8h);
    outportb(PITMODE, 0x36);
    pit0_value = PITCONST / Hz;
    pit0_set = (pit0_value & 0x00ff);
    outportb(PIT0, pit0_set);
    pit0_set = (pit0_value >> 8);
    outportb(PIT0, pit0_set);
    enable();

    flag8h = NEW8H;
    freq8h = Hz;
    counter_8h = 0;
    counter_reset = freq8h / PIT0DEF;
    tick_per_ms = freq8h / 1000;
    ms_per_tick = 1000 / freq8h;
  }
}

void        quit8h(void)
{
```

```
  unsigned int   pit0_set,
                 pit0_value;
  long           tick;
  char           *cmostime;

  if (flag8h == NEW8H) {
    disable();
    outportb(PITMODE, 0x36);
    outportb(PIT0, 0x00);
    outportb(PIT0, 0x00);
    setvect(IRQ0, old8h);
    enable();

    cmostime = get_cmostime();
    tick = PIT0DEF *
      (
       (((float) *cmostime) * 3600) +
       (((float) *(cmostime + 1)) * 60) +
       (((float) *(cmostime + 2)))
      );
    biostime(1, tick);

    flag8h = 0;
    freq8h = PIT0DEF;
    counter_reset = freq8h / PIT0DEF;
    tick_per_ms = freq8h / 1000;
    ms_per_tick = 1000 / freq8h;
  }
}

void          interrupt
              new8h(__CPPARGS)
{
  disable();
  ticks_8h++;
  counter_8h++;

  if (counter_8h == counter_reset) {
    counter_8h = 0;
    enable();
    old8h();
  } else {
    outportb(0x20, 0x20);
  }
}

unsigned long   time8h(unsigned long start, unsigned long stop)
{
  unsigned long   duration,
                  millisec;

  if (stop < start)
    return 0;
  else {
    duration = stop - start;
    millisec = duration * ms_per_tick;
    return millisec;
  }
}

void          delay8h(unsigned int delayms)
{
  unsigned long int delaybegin = 0;
  unsigned long int delayend = 0;
  unsigned int    delaytick;

  delaytick = delayms * tick_per_ms;

  if (flag8h == NEW8H)
    delaybegin = ticks_8h;
  else
    biostime(0, (long) delaybegin);
```

```
    do {
        if (flag8h == NEW8H)
            delayend = ticks_8h;
        else
            biostime(0, (long) delayend);
    } while ((delayend - delaybegin) < delaytick);
}

void        sound8h(int freq, int duration)
{
    int         byte;
    unsigned int    freq1;

    freq1 = PITCONST / freq;
    outportb(PITMODE, 0xb6);
    byte = (freq1 & 0xff);
    outportb(PIT2, byte);
    byte = (freq1 >> 8);
    outportb(PIT2, byte);
    byte = inportb(KBCTRL);
    outportb(KBCTRL, (byte | 3));

    delay8h(duration);
    outportb(KBCTRL, (byte & 0xfc));
}

char        *get_cmostime(void)
{
    union REGS      inreg;
    union REGS      outreg;
    char        *buff;
    static char     buffer[6];
    char        ch;

    buff = buffer;
    inreg.h.ah = 0x02;
    int86(0x1a, &inreg, &outreg);

    ch = outreg.h.ch;
    buffer[0] = (char) ((int) (ch & 0x0f) + (int) ((ch >> 4) & 0x0f) * 10);
    ch = outreg.h.cl;
    buffer[1] = (char) ((int) (ch & 0x0f) + (int) ((ch >> 4) & 0x0f) * 10);
    ch = outreg.h.dh;
    buffer[2] = (char) ((int) (ch & 0x0f) + (int) ((ch >> 4) & 0x0f) * 10);
    buffer[3] = outreg.h.dl;
    buffer[4] = (char) (outreg.x.cflag & 0x0001);
    buffer[5] = 0x00;

    return (buff);
}
```

APPENDIX B.3

DSP-μ_p Driver Source Code

DSP-μ_p Driver Source Code

```
/*global variables*/
float fuz_mat[30][3]={{0}},rules_mat[3][10]={{0}},bdr_mat[3][5]={{0}};
float Glob_AVR=1,Glob_PSS=1,U_AVR[10],U_PSS[10],ekm1=0,ekm2=0,ekm3=0,sectrs[6]={0};
/**** filters parameters ****/
float f_tach,f_tch_k1=0,f_tch_k2=0,f_tch_k3=0,f_tch_k4=0,f_tch_k5=0,f_tch_k6=0;
float tch_k1=0,tch_k2=0,tch_k3=0,tch_k4=0,tch_k5=0,tch_k6=0;
/* vt filter*/
float f_VT_k1=0,f_VT_k2=0,f_VT_k3=0,f_VT_k4=0,f_VT_k5=0,f_VT_k6=0;
float VT_k1=0,VT_k2=0,VT_k3=0,VT_k4=0,VT_k5=0,VT_k6=0;
float speed_k1=0,speed_k2=0,speed_k3=0,speed_k4=0;
float f_speed_k1=0,f_speed_k2=0,f_speed_k3=0,f_speed_k4=0;
float er_ang_km=0,speed,m_ang,er_ang,ed_ang,f_speed,Dw_k1=0;
/* updating variables*/
int updt_count=-1;
float sect1_FLG=0,sect2_FLG=0,sect3_FLG=0,sect4_FLG=0;
float sect5_FLG=0,sect6_FLG=0,Learn_T,sum_err=0,er_av_km=0;
float flc_sts[3]={0},Main_FLC_k=0,Main_FLC_Sum=0,Main_FLC_S_km=0;
float C_In_Rng=0,C_Out_Rng=0,G_In_Rng=1,G_Out_Rng=1,mode=0,Learn;
float tch_1st=1;

/*functions*/

float bez_bck(float ,float,float,float,float);
float shay_fuz(float ,float [][4] ,float ,float [][4], int);
float shay_rl(float [][3], int [][20] );
float shay_bdr(float [][10], float [][3]);
float shay_dff(float [][5], float [][3]);
void sectors(float,float,float);
float Main_FLC(float );
float *shay_pa_n(float ,float ,char );
float angg(float ,float );
float sw_pa_n(float ,float, char );
void PSS_PARM(float );
float sw_sct1(float ,float );
float sw_sct2(float ,float );
float sw_sct3(float ,float );
float sw_sct4(float ,float );
float sw_sct5(float ,float );
float sw_sct6(float ,float );
void Shay(float ,float ,float );
float vt_fltr(float );
float PRE_CONTROL(float ,float );
void updt_in_out(float , float,float);
void opra_sts_in(float ,float );
void opra_sts_out(float ,float );
float max_arry(float [],int  );
void Qmar(float *,float *,float ,int );
float Monkey(float );

/*******************
  Header Files
*******************/
        #include "intpt40.h"
        #include "amelia.h"
        #include "dpram.h"
        #include "out_sg.h"
        #include "dsp_head.h"
        #include "shay_fnc.h"
/*********************************
flags and data buffer in DPRAM
*********************************/
        #define   Start_DSP_Prog              ((long *) DPRAM_HANDSHAKE0)
        #define   PC_Ready_to_Recieve         ((long *) DPRAM_HANDSHAKE1)
        #define   PC_In_Data_Ready   ((long *) DPRAM_HANDSHAKE2)
        #define   Sample_Start                ((long *) DPRAM_HANDSHAKE3)
        #define Sample_End            ((long *) DPRAM_HANDSHAKE4)
        #define Relay_On              ((long *) DPRAM_HANDSHAKE5)
        #define BUFFER                        ((float *) 0xA0000200)
```

```
        #define BUFFER1_LENGTH    0x100

/*************************
  Function ProtoTypes
*************************/
        void c_int04(void);

/******************
  Main Program
******************/
void main(void)
        {
/***********************************
start IACK
***********************************/

        asm(" PUSH AR0");
        asm(" PUSH DP");
        asm(" LDI 030H,AR0");
        asm(" LSH 16,AR0");
        asm(" IACK *AR0");
        asm(" POP DP");
        asm(" POP AR0");

        INT_DISABLE();
        set_ivtp( (void *)0x002ffe00);
        install_int_vector((void *)c_int04, 0x04);
        load_iif(0x00B0);
        *USER_CONTROL_REG = 0xA400;
        *AMELIA_CONTROL_REG = 0x00B2;
        *TIMER1_REG = 0xFFC4;
        *INTERRUPT_MASK_REG = 0x0002;
        *CONFIGURATION_REG = 0x8DFF;

/***************************
Get Data from user
****************************/
        while(*Start_DSP_Prog==0);
        Start_DSP_Prog=0;
        Run_Time=*(BUFFER);
        SampleRate=*(BUFFER+1);
        tch_1st=*(BUFFER+2);
        Tachok=*(BUFFER+3);
        Dst_Type=*(BUFFER+4);
        switch(Dst_Type)
                {
                case 1:
                        {
                        Nw_Ref=*(BUFFER+5);
                        Nw_st=*(BUFFER+6);
                        Nw_End=*(BUFFER+7);
                        Ref_Flg=*(BUFFER+8);
                        break;
                        }
                case 2:
                        {
                        R1On=*(BUFFER+5);
                        R1Off=*(BUFFER+6);
                        R2On=*(BUFFER+7);
                        R2Off=*(BUFFER+8);
                        Load_Flg=*(BUFFER+9);
                        break;
                        }
                case 3:
                        {
                        R1On=*(BUFFER+5);
                        scOff=*(BUFFER+6);
                        Sc_Flg=*(BUFFER+7);
                        break;
                        }
                }
```

```
/*************************
 start enable interrupts
*************************/
        INT_ENABLE();
        while(1);
        }

/************************************
start interrupt service routine
************************************/
void c_int04(void)
        {
        volatile unsigned long clear;
        unsigned long tempA, tempB,bit_16,bit_15,bit_ng,out1,out2;
        int bb1,bb2,bb3,bb4;
        float tempA_f,tempB_f,op1,op2,In_Mod;
        bit_16=65535;
        bit_15=32767;
        bit_ng=32768;
        clear = *INTERRUPT_STATUS_REG;
        while(*Sample_Start==0);
        *Sample_Start=1;
        tempA = *CH0_IN_DATA_REG;
        tempB = *CH1_IN_DATA_REG;
        tempA=(~tempA)+1;
        tempB=(~tempB)+1;
        bb1=tempA;
        bb2=tempA;
        bb3=tempB;
        bb4=tempB;
        bb1=bb1&bit_16;
        bb3=bb3&bit_16;
        bb2=bb2&bit_15;
        bb4=bb4&bit_15;
        if (bb1<=32768)
                {
                tempA_f=(float)(bb1*3)/32767;
                }
        if (bb1>32768)
                {
                tempA_f=((float)((bb2-32768)*3)/32767);
                }
        if (bb3<=32768)
                {
                tempB_f=(float)(bb3*3)/32767;
                }
        if (bb3>32768)
                {
                tempB_f=((float)((bb4-32768)*3)/32767);
                }

/************************************************
PROCESSING THE INPUTS
************************************************/
                Vt=2.12*tempA_f;
                Vt=vt_fltr(Vt);
                Tacho=tempB_f;
                Tacho_diff=Tachok-Tacho;
                if (fabs(Tacho_diff)<0.1)
                        Tacho=Tachok;
                else
                        Tachok=Tacho;

                PSS_PARM (Tacho);
                Dw=f_speed;
                s_Dw=dw_scl2*f_speed;
                s_Dw_k1=dw_km_scl*Dw_k1;
                DW_acc=s_Dw-s_Dw_k1;
                ADW=PRE_CONTROL(s_Dw,DW_acc);
                ek=V_Ref_sg-Vt+ADW;
                AVR_acc=ek-ekm1;
                AVR_EK=pow((pow(ek,2)+pow(ekm1,2)),0.5);
```

```
            AVR_EKM=pow((pow(ekm2,2)+pow(ekm3,2)),0.5);
            Shay(ek,ekm1,AVR_EK);
            excitation=Glob_AVR;
            if (excitation>2.99)
                  excitation=2.99;
            if(excitation<-2.99)
                  excitation=-2.99;
            ekm3=ekm2;
            ekm2=ekm1;
            ekm1=ek;
            Dw_k1=Dw;
while(*PC_Ready_to_Recieve==0);
*PC_Ready_to_Recieve=0;
*(BUFFER)=Vt;
*(BUFFER+1)=Dw;
*(BUFFER+2)=excitation;
*(BUFFER+1)=ed_ang;
*PC_In_Data_Ready=1;
*DSP_Ready_to_Recieve=1;
while(*DSP_In_Data_Ready==0);
*DSP_Ready_to_Recieve=0;
*DSP_In_Data_Ready=0;
op1=excitation;
if (*Relay_On==1)
            op2=.6;
else
            op2=0;
while(*Sample_End==0);
Sample_End=0;
output_sg(op1,op2);
}
```

APPENDIX B.4

Shay Module Source Code

Shay Module Source Code

```
void Shay(float ek_a,float ekm_a,float AVR_EK_a)
        {
        float N_EE,U_Main_a,acc_a,opr_ang,sw_ang,PA;
        float min_r,max_r,x1,max_p,x2,min_n,P,N;

        N_EE=AVR_EK_a;
        AVR_EK_a=AVR_EK_a*G_In_Rng;
        ek_a=ek_a*G_In_Rng;
        ekm_a=ekm_a*G_In_Rng;
        /***************
        Main_FLC {AVR}
        ****************/
        U_Main_a=Main_FLC(AVR_EK_a);
        /*********************
        updt_in_out
        *********************/
        if(Learn==1)
                updt_in_out(ek_a,ekm_a,N_EE);
         acc_a=ek_a-ekm_a;
         sectors(ek_a,ekm_a,acc_a);
         opr_ang=angg(ekm_a,ek_a);

         if (sectrs[0]==1)
                {
                /*************************
                operation in sector 1
                *************************/

                opr_ang=opr_ang+11.25;
                sw_ang=sw_sct1(opr_ang,fabs(ek_a));

                /****************************************
                calculating final increment/decrement
                ****************************************/
                min_r=0;
                max_r=67.5;
                x1=sw_ang-min_r;
                max_p=x1+sw_ang;

                x2=max_r-sw_ang;
                min_n=sw_ang-x2;
                P=bez_bck(opr_ang,-0.1,0,0,max_p);
                N=bez_bck(opr_ang,min_n,67.5,67.5,68);
                PA=P-N;
                }

        else if (sectrs[1]==1)
                {
                /*************************
                operation in sector 2
                *************************/
                sw_ang=sw_sct2(opr_ang,fabs(ek_a));

                /****************************************
                calculating final increment/decrement
                ****************************************/
                min_r=33.75;
                max_r=101.25;
                x1=sw_ang-min_r;
                max_p=x1+sw_ang;

                x2=max_r-sw_ang;
                min_n=sw_ang-x2;
                P=bez_bck(opr_ang,30,33.75,33.75,max_p);
                N=bez_bck(opr_ang,min_n,101.25,101.25,102);
                PA=P-N;
                }
```

```
else if (sectrs[2]==1)
        {
        /*************************
        operation in sector 3
        *************************/
        sw_ang=sw_sct3(opr_ang,fabs(ek_a));

        /***************************************
        calculating final increment/decrement
        ***************************************/
        min_r=67.50;
        max_r=202.50;
        x1=sw_ang-min_r;
        max_p=x1+sw_ang;

        x2=max_r-sw_ang;
        min_n=sw_ang-x2;
        P=bez_bck(opr_ang,67,67.50,67.55,max_p);
        N=bez_bck(opr_ang,min_n,202.50,202.50,205);
        PA=N-P;
        }
else if (sectrs[3]==1)
        {
        /*************************
        operation in sector 4
        *************************/
        sw_ang=sw_sct4(opr_ang,fabs(ek_a));

        /***************************************
        calculating final increment/decrement
        ***************************************/
        min_r=168.75;
        max_r=236.25;
        x1=sw_ang-min_r;
        max_p=x1+sw_ang;

        x2=max_r-sw_ang;
        min_n=sw_ang-x2;
        P=bez_bck(opr_ang,168,168.75,168.7500,max_p);
        N=bez_bck(opr_ang,min_n,236.25,236.25,239);
        PA=N-P;
        }
else if (sectrs[4]==1)
        {
        /*************************
        operation in sector 5
        *************************/
        sw_ang=sw_sct5(opr_ang,fabs(ek_a));

        /***************************************
        calculating final increment/decrement
        ***************************************/
        min_r=213.75;
        max_r=281.25;
        x1=sw_ang-min_r;
        max_p=x1+sw_ang;
        x2=max_r-sw_ang;
        min_n=sw_ang-x2;
        P=bez_bck(opr_ang,213,213.75,213.75,max_p);
        N=bez_bck(opr_ang,min_n,281.25,281.25,282);
        PA=N-P;
        }
else if (sectrs[5]==1)
        {
        /*************************
        operation in sector 6
        *************************/
        sw_ang=sw_sct6(opr_ang,fabs(ek_a));

        /***************************************
        calculating final increment/decrement
        ***************************************/
```

```
                    min_r=247.5;
                    max_r=382.50;
                    x1=sw_ang-min_r;
                    max_p=x1+sw_ang;

                    x2=max_r-sw_ang;
                    min_n=sw_ang-x2;
                    P=bez_bck(opr_ang,247,247.5,247.5,max_p);
                    N=bez_bck(opr_ang,min_n,382.5,382.55,383);
                    PA=P-N;
                    }

/*****************************
start data arrangments
*****************************/
    Glob_AVR=(U_Main_a*PA)+Glob_AVR;
            if (Glob_AVR>2.99)
                    Glob_AVR=2.99;
            if (Glob_AVR<-2.9)
                    Glob_AVR=-2.9;
            U_AVR[0]=Glob_AVR;
            U_AVR[1]=PA*U_Main_a;
            U_AVR[2]=U_Main_a;
            U_AVR[3]=opr_ang;
            U_AVR[4]=sw_ang;
            U_AVR[5]=PA;
            }
/****************************** END Shay ******************/
```

APPENDIX B.5

Subfunctions Source Code

Subfunctions Source Code

```
/*******************
function: bez_bck
*********************/
float bez_bck(float t,float a1,float a2,float a3,float a4)
{
        float u;
        float b1=((a2-a1)/2)+a1;
        float b2=((a4-a3)/2)+a3;
        if ((a1==a2)&& (a3!=a4))
        {
                if (t<a1)
                        u=0;
                else if (t<=a3)
                        u=1;
                else if (t<=b2)
                        u=1-2*pow(((t-a4)/(a4-a3)),2);
                else if (t<=a4)
                        u=2*pow(((t-a4)/(a4-a3)),2);
                else
                        u=0;

        }
        else if ((a3==a4) && (a1 != a2))
        {
                if (t<a1)
                        u=0;
                else if (t<=b1)
                        u=2*pow(((t-a1)/(a2-a1)),2);
                else if (t<=a2)
                        u=1-2*pow(((t-a2)/(a2-a1)),2);
                else if (t<=a3)
                        u=1;
                else
                        u=0;
           }
        else
        {
                if (t<a1)
                        u=0;
                else if (t<=b1)
                        u=2*pow(((t-a1)/(a2-a1)),2);
                else if (t<=a2)
                        u=1-2*pow(((t-a2)/(a2-a1)),2);
                else if (t<=a3)
                        u=1;
                else if (t<=b2)
                        u=1-2*pow(((t-a3)/(a4-a3)),2);
                else if (t<=a4)
                        u=2*pow(((t-a4)/(a4-a3)),2);
                else
                        u=0;
        }
return u;
}


/*************
sectors
**************/
void sectors(float ek_s,float ekm_s, float acc_s)
        {
        int wkp,wkn,wkmp,wkmn,accp,accn;
        if (ek_s>=0)
                {
                wkp=1;
                wkn=0;
                }
        else
```

```
                {
                wkp=0;
                wkn=1;
                }
        if (ekm_s>=0)
                {
                wkmp=1;
                wkmn=0;
                }
        else
                {
                wkmp=0;
                wkmn=1;
                }
        if (acc_s>=0)
                {
                accp=1;
                accn=0;
                }
        else
                {
                accp=0;
                accn=1;
                }
    sectrs[0]=wkp*wkmp*accp;
    sectrs[1]=wkp*wkmp*accn;
    sectrs[2]=wkn*wkmp*accn;
    sectrs[3]=wkn*wkmn*accn;
    sectrs[4]=wkn*wkmn*accp;
    sectrs[5]=wkp*wkmn*accp;
        }


/*************
ANGG
**************/
float angg(float a,float s)
        {
        float tht;
        if ((s==0)&&(a==0))
                tht=0;
        else if ((s==0)&&(a>0))
                tht=90;
        else if ((s<0)&&(a==0))
                tht=180;
        else if ((s==0)&&(a<0))
                tht=270;
        else
                tht=180*(atan(a/s))/3.141593;
        if ((s<0)&&(a>0))
                tht=tht+180;
        if ((s<0)&&(a<0))
                tht=tht+180;
        if ((s>0)&&(a<0))
                tht=360+tht;
        return(tht);
        }


/*************
PSS_PARM
**************/
void PSS_PARM(float Dw1)
        {
        float f_tch_k,diffr;
        f_tch_k=1.627*f_tch_k1-1.89031*f_tch_k2+1.0776*f_tch_k3-0.46637*f_tch_k4+0.08469*f_tch_k5-
                0.0123*f_tch_k6+0.0550073*(Dw1+1.0491*tch_k1+2.19433*tch_k2+2.05156*tch_k3+2.19433*tch_k4+1.0491*
                tch_k5+1*tch_k6);
        f_tach=f_tch_k;
        diffr=f_tch_k-(tch_1st);
        m_ang=diffr/(0.14);
        ed_ang=m_ang*2;
```

```
        er_ang=ed_ang*3.14/180;
        speed=er_ang-er_ang_km;
        er_ang_km=er_ang;
        speed=speed/(.025*314);
        f_speed=1.0586*f_speed_k1-0.98622*f_speed_k2+0.389485*f_speed_k3-
                0.0789937*f_speed_k4+0.0385299*(speed+4*speed_k1+6*speed_k2+4*speed_k3+speed_k4);
        f_tch_k6=f_tch_k5;
        f_tch_k5=f_tch_k4;
        f_tch_k4=f_tch_k3;
        f_tch_k3=f_tch_k2;
        f_tch_k2=f_tch_k1;
        f_tch_k1=f_tch_k;
        tch_k6=tch_k5;
        tch_k5=tch_k4;
        tch_k4=tch_k3;
        tch_k3=tch_k2;
        tch_k2=tch_k1;
        tch_k1=Dw1;
        f_speed_k4=f_speed_k3;
        f_speed_k3=f_speed_k2;
        f_speed_k2=f_speed_k1;
        f_speed_k1=f_speed;
        speed_k4=speed_k3;
        speed_k3=speed_k2;
        speed_k2=speed_k1;
        speed_k1=speed;
        f_speed=f_speed;
        }


/*********
max_arry
**********/
float max_arry(float f_arry[],int szz )
        {
        float tempr=0;
        int i;
        for (i=0;i<szz;i++)
                if (tempr<f_arry[i])
                        tempr=f_arry[i];
        return(tempr);
        }


/*****
Qmar
******/

void Qmar(float in_ar[3],float out_ar[3],float GE,int md)
        {
        float Rs[9],Es[3],Fs[27],F_in[27],F_out[27],D_in[27],D_out[27];
        float I_in[27],I_out[27];
        float Sum_updt,Mnky,Fix_In,Fix_Out,Inc_In,Inc_Out,Dec_In,Dec_Out;
        int sszz,i,j,F=0,k=0;
        /******************
        calculating the R's
        ********************/
        for (i=0;i<3;i++)  /*out array*/
                for(j=0;j<3;j++)    /*in array*/
                        {
                        Rs[k++]=in_ar[j]*out_ar[i];

                        }
        /******************
        calculating ei's
        ******************/
        if (GE>0.1)
                GE=0.1;
        Es[0]=bez_bck(GE,-0.01,0.00,0.00,0.02);
        Es[1]=bez_bck(GE,0.01,0.04,0.04,0.07);
        Es[2]=bez_bck(GE,0.06,0.10,0.10,0.11);
```

```
/*********************
calculatinfg the F's
**********************/
for (i=0;i<k;i++)
        for(j=0;j<3;j++)
                Fs[F++]=Es[j]*Rs[i];
/************************
calculate update signal
************************/
Mnky=Monkey(GE);
/*****************************
calculating Dec,Inc and Fix
*****************************/
switch (md)
        {
        case 1 : /*over*/
                {
                /***************
                INPUT
                ****************/
                F_in[0]=Fs[0];
                F_in[1]=Fs[9];
                F_in[2]=Fs[12];
                F_in[3]=Fs[21];
                F_in[4]=Fs[4];
                F_in[5]=Fs[7];
                F_in[6]=Fs[16];
                F_in[7]=Fs[19];
                F_in[8]=Fs[25];
                F_in[9]=Fs[5];
                F_in[10]=Fs[8];
                F_in[11]=Fs[14];
                F_in[12]=Fs[17];
                F_in[13]=Fs[20];
                F_in[14]=Fs[26];
                sszz=15;
                Fix_In=max_arry(F_in,sszz);
                I_in[0]=Fs[18];
                I_in[1]=Fs[1];
                I_in[2]=Fs[10];
                I_in[3]=Fs[13];
                I_in[4]=Fs[22];
                I_in[5]=Fs[2];
                I_in[6]=Fs[11];
                I_in[7]=Fs[23];
                sszz=8;
                Inc_In=max_arry(I_in,sszz);
                D_in[0]=Fs[3];
                D_in[1]=Fs[6];
                D_in[2]=Fs[15];
                D_in[3]=Fs[24];
                sszz=4;
                Dec_In=max_arry(D_in,sszz);
                /***************
                OUTPUT
                ****************/
                F_out[0]=Fs[0];
                F_out[1]=Fs[3];
                F_out[2]=Fs[6];
                F_out[3]=Fs[15];
                F_out[4]=Fs[24];
                F_out[5]=Fs[1];
                F_out[6]=Fs[10];
                F_out[7]=Fs[13];
                F_out[8]=Fs[22];
                F_out[9]=Fs[2];
                F_out[10]=Fs[11];
                F_out[11]=Fs[23];
                sszz=12;
                Fix_Out=max_arry(F_out,sszz);
                I_out[0]=Fs[4];
                I_out[1]=Fs[7];
```

```
                I_out[2]=Fs[16];
                I_out[3]=Fs[19];
                I_out[4]=Fs[25];
                I_out[5]=Fs[5];
                I_out[6]=Fs[8];
                I_out[7]=Fs[14];
                I_out[8]=Fs[17];
                I_out[9]=Fs[20];
                I_out[10]=Fs[26];
                sszz=11;
                Inc_Out=max_arry(I_out,sszz);
                D_out[0]=Fs[9];
                D_out[1]=Fs[12];
                D_out[2]=Fs[18];
                D_out[3]=Fs[21];
                sszz=4;
                Dec_Out=max_arry(D_out,sszz);
                Sum_updt=Dec_In+Inc_In+Fix_In;
                Dec_In=Dec_In/Sum_updt;
                Inc_In=Inc_In/Sum_updt;
                Fix_In=Fix_In/Sum_updt;
                Sum_updt=Dec_Out+Inc_Out+Fix_Out;
                Dec_Out=Dec_Out/Sum_updt;
                Inc_Out=Inc_Out/Sum_updt;
                Fix_Out=Fix_Out/Sum_updt;
                mode=1;
                break;
                }
case 2 : /*under*/
                {
                /***************
                INPUT
                ****************/
                F_in[0]=Fs[0];
                F_in[1]=Fs[9];
                F_in[2]=Fs[12];
                F_in[3]=Fs[21];
                F_in[4]=Fs[4];
                F_in[5]=Fs[7];
                F_in[6]=Fs[16];
                F_in[7]=Fs[19];
                F_in[8]=Fs[25];
                F_in[9]=Fs[5];
                F_in[10]=Fs[8];
                F_in[11]=Fs[14];
                F_in[12]=Fs[17];
                F_in[13]=Fs[20];
                F_in[14]=Fs[26];
                sszz=15;
                Fix_In=max_arry(F_in,sszz);
                I_in[0]=Fs[18];
                I_in[1]=Fs[1];
                I_in[2]=Fs[10];
                I_in[3]=Fs[13];
                I_in[4]=Fs[22];
                I_in[5]=Fs[2];
                I_in[6]=Fs[11];
                I_in[7]=Fs[23];
                sszz=8;
                Inc_In=max_arry(I_in,sszz);
                D_in[0]=Fs[3];
                D_in[1]=Fs[6];
                D_in[2]=Fs[15];
                D_in[3]=Fs[24];
                sszz=4;
                Dec_In=max_arry(D_in,sszz);
                /***************
                OUTPUT
                ****************/
                F_out[0]=Fs[0];
                F_out[1]=Fs[3];
                F_out[2]=Fs[6];
```

```
                F_out[3]=Fs[15];
                F_out[4]=Fs[24];
                F_out[5]=Fs[1];
                F_out[6]=Fs[10];
                F_out[7]=Fs[13];
                F_out[8]=Fs[22];
                F_out[9]=Fs[2];
                F_out[10]=Fs[11];
                F_out[11]=Fs[23];
                sszz=12;
                Fix_Out=max_arry(F_out,sszz);
                I_out[0]=Fs[4];
                I_out[1]=Fs[7];
                I_out[2]=Fs[16];
                I_out[3]=Fs[19];
                I_out[4]=Fs[25];
                I_out[5]=Fs[5];
                I_out[6]=Fs[8];
                I_out[7]=Fs[14];
                I_out[8]=Fs[17];
                I_out[9]=Fs[20];
                I_out[10]=Fs[26];
                sszz=11;
                Inc_Out=max_arry(I_out,sszz);
                D_out[0]=Fs[9];
                D_out[1]=Fs[12];
                D_out[2]=Fs[18];
                D_out[3]=Fs[21];
                sszz=4;
                Dec_Out=max_arry(D_out,sszz);
                Sum_updt=Dec_In+Inc_In+Fix_In;
                Dec_In=Dec_In/Sum_updt;
                Inc_In=Inc_In/Sum_updt;
                Fix_In=Fix_In/Sum_updt;
                Sum_updt=Dec_Out+Inc_Out+Fix_Out;
                Dec_Out=Dec_Out/Sum_updt;
                Inc_Out=Inc_Out/Sum_updt;
                Fix_Out=Fix_Out/Sum_updt;
                mode=2;
                break;
                }
case 3 : /*osc*/
                {
                /***************
                INPUT
                ****************/
                F_in[0]=Fs[0];
                F_in[1]=Fs[9];
                F_in[2]=Fs[12];
                F_in[3]=Fs[18];
                F_in[4]=Fs[21];
                F_in[5]=Fs[24];
                F_in[6]=Fs[1];
                F_in[7]=Fs[4];
                F_in[8]=Fs[10];
                F_in[9]=Fs[13];
                F_in[10]=Fs[19];
                F_in[11]=Fs[22];
                F_in[12]=Fs[2];
                F_in[13]=Fs[5];
                F_in[14]=Fs[11];
                F_in[15]=Fs[20];
                sszz=16;
                Fix_In=max_arry(F_in,sszz);
                Inc_In=0;
                D_in[0]=Fs[3];
                D_in[1]=Fs[6];
                D_in[2]=Fs[15];
                D_in[3]=Fs[7];
                D_in[4]=Fs[16];
                D_in[5]=Fs[25];
                D_in[6]=Fs[8];
```

```
                    D_in[7]=Fs[14];
                    D_in[8]=Fs[17];
                    D_in[9]=Fs[23];
                    D_in[10]=Fs[26];
                    sszz=11;
                    Dec_In=max_arry(D_in,sszz);
                    /***************
                    OUTPUT
                    ****************/
                    F_out[0]=Fs[0];
                    F_out[1]=Fs[3];
                    F_out[2]=Fs[6];
                    sszz=3;
                    Fix_Out=max_arry(F_out,sszz);
                    Inc_Out=0;
                    D_out[0]=Fs[9];
                    D_out[1]=Fs[12];
                    D_out[2]=Fs[15];
                    D_out[3]=Fs[18];
                    D_out[4]=Fs[21];
                    D_out[5]=Fs[24];
                    D_out[6]=Fs[1];
                    D_out[7]=Fs[4];
                    D_out[8]=Fs[7];
                    D_out[9]=Fs[10];
                    D_out[10]=Fs[13];
                    D_out[11]=Fs[16];
                    D_out[12]=Fs[19];
                    D_out[13]=Fs[22];
                    D_out[14]=Fs[25];
                    D_out[15]=Fs[2];
                    D_out[16]=Fs[8];
                    D_out[17]=Fs[11];
                    D_out[18]=Fs[14];
                    D_out[19]=Fs[17];
                    D_out[20]=Fs[20];
                    D_out[21]=Fs[23];
                    D_out[22]=Fs[26];
                    D_out[23]=Fs[5];
                    sszz=24;
                    Dec_Out=max_arry(D_out,sszz);
                    Sum_updt=Dec_In+Inc_In+Fix_In;
                    Dec_In=Dec_In/Sum_updt;
                    Inc_In=Inc_In/Sum_updt;
                    Fix_In=Fix_In/Sum_updt;
                    Sum_updt=Dec_Out+Inc_Out+Fix_Out;
                    Dec_Out=Dec_Out/Sum_updt;
                    Inc_Out=Inc_Out/Sum_updt;
                    Fix_Out=Fix_Out/Sum_updt;
                    mode=3;
                    break;
                    }
            default :
                    {
                    Fix_In=1;
                    Fix_Out=1;
                    Dec_In=0;
                    Dec_Out=0;
                    Inc_In=0;
                    Inc_Out=0;
                    mode=0;
                    }
                    }/*end switch*/
    /*********************************
    calculate final updating signals
    *********************************/
            C_In_Rng=(1-Fix_In)*(Inc_In-Dec_In)*Mnky;
            G_In_Rng=G_In_Rng+C_In_Rng;
            C_Out_Rng=(1-Fix_Out)*(Inc_Out-Dec_Out)*Mnky;
            G_Out_Rng=G_Out_Rng+C_Out_Rng;
}
```

```
/*************
updt_in_out
**************/
void updt_in_out(float eRk,float eRkm,float N_E)
        {
        float In_Sts[3],Out_Sts[3],under,over,mode,er_av,In_R1=0,In_R2=0,In_R3=0;
        float AcC=0,Out_R1=0,Out_R2=0,Out_R3=0;
        /***************
        update sectors
        **************/
        updt_count++;
        AcC=eRk-eRkm;
        sectors(eRk,eRkm,AcC);
        sect1_FLG=sectrs[0]+sect1_FLG;
        sect2_FLG=sectrs[1]+sect2_FLG;
        sect3_FLG=sectrs[2]+sect3_FLG;
        sect4_FLG=sectrs[3]+sect4_FLG;
        sect5_FLG=sectrs[4]+sect5_FLG;
        sect6_FLG=sectrs[5]+sect6_FLG;
        Main_FLC_Sum=Main_FLC_Sum+Main_FLC_k;
        /******************
        update error array
        ******************/
        sum_err=pow((pow(eRk,2)+pow(eRkm,2)),0.5)+sum_err;
        /*start */
        if (updt_count==Learn_T-1)
                {
                er_av=sum_err/Learn_T;
                Main_FLC_Sum=Main_FLC_Sum/Learn_T;
                if (er_av>0.01)
                        {
                        /* check input and output operational status */
                        /**********************
                        input oprational status
                        **********************/
                        opra_sts_in(er_av,er_av_km);
                        In_R1=flc_sts[0];
                        In_R2=flc_sts[1];
                        In_R3=flc_sts[2];
                        In_Sts[0]=In_R1;
                        In_Sts[1]=In_R2;
                        In_Sts[2]=In_R3;
                        /********************
                        output oprational status
                        ********************/
                        opra_sts_out(Main_FLC_Sum,Main_FLC_S_km);
                        Out_R1=flc_sts[0];
                        Out_R2=flc_sts[1];
                        Out_R3=flc_sts[2];
                        Out_Sts[0]=Out_R1;
                        Out_Sts[1]=Out_R2;
                        Out_Sts[2]=Out_R3;
                        /******************
                        determine mode
                        ******************/
                        under=(sect1_FLG+sect2_FLG)/Learn_T;
                        over=(sect4_FLG+sect5_FLG)/Learn_T;
                        mode=max(under,over);
                        if (mode>0.75)
                                {
                                if (under>over)
                                        {
                                        Qmar(In_Sts,Out_Sts,N_E,1);
                                        }
                                else if (over>under)
                                        {
                                        Qmar(In_Sts,Out_Sts,N_E,2);
                                        }
                                 else
                                        {
```

```
                                        Qmar(In_Sts,Out_Sts,N_E,3);
                                        }
                                }
                        else
                                {
                                Qmar(In_Sts,Out_Sts,N_E,3);
                                }
                        /*****************
                        reset all
                        *****************/
                        er_av_km=er_av;
                        sum_err=0;
                        updt_count=-1;
                        sect1_FLG=0;
                        sect2_FLG=0;
                        sect3_FLG=0;
                        sect4_FLG=0;
                        sect5_FLG=0;
                        sect6_FLG=0;
                        Main_FLC_S_km=Main_FLC_Sum;
                        Main_FLC_Sum=0;
                        }
                else
                        {

                        er_av_km=er_av;
                        sum_err=0;
                        updt_count=-1;
                        sect1_FLG=0;
                        sect2_FLG=0;
                        sect3_FLG=0;
                        sect4_FLG=0;
                        sect5_FLG=0;
                        sect6_FLG=0;
                        Main_FLC_S_km=Main_FLC_Sum;
                        Main_FLC_Sum=0;
                        }
                }
        else
                {
                C_In_Rng=0;
                C_Out_Rng=0;
                }
}
```

APPENDIX B.6

FLC Drivers Source Code

FLC Drivers Source Code

```
/**************
Main_FLC
***************/
float Main_FLC(float Input_1)
        {
        float i,U_MAIN;
        int No_Input=1;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=0.2;
        float Min_In_1=0;
        float       In_1[6][4]=         {
                                        {5          ,4          ,0          },
                                        {-0.0200,0.0000         ,0.0100     },
                                        {0.0000     ,0.0200     ,0.0500     },
                                        {0.0200     ,0.0500     ,0.1000     },
                                        {0.0500     ,0.1000     ,0.1500     },
                                        {0.1000     ,0.2000     ,0.2100     }
                                        };
        /***************************
        OUTPUT CLASSES DEFINITIN
        ***************************/
        int No_Out=5;
        float Out_mat[6][3]=  {
                                        {5          ,3          ,0          },
                                        {-0.0500,0.0000         ,0.0500     },
                                        {0.0000     ,0.0500     ,0.2000     },
                                        {0.0500     ,0.2000     ,0.5000     },
                                        {0.2000     ,0.5000     ,0.7000     },
                                        {0.5000     ,1.0000     ,1.1000     }
                                        };
        /*****************************
        Rules
        *****************************/
        int Rules_m[2][20]=  {
                                        {1 ,5 ,5 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                                        {1 ,2 ,3 ,4 ,5 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                                        };
/***************************
  END DEFFINITIONS
***************************/
                Input_1=Input_1;
                /****************************
                SCALLING Input 1
                ****************************/
                if (Input_1>Max_In_1)
                        Input_1=Max_In_1;
                if (Input_1<Min_In_1)
                        Input_I=Min_In_1;
                /*******************************
                generation of final in_1 matrix
                *******************************/
                for (i=1;i<No_In_1+1;i++)
                        {
                        In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                        }
                /****************************
                        FUZZIFICATION
                ****************************/
                if (No_Input==1)
                        {
                        /*sinle input*/
                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                        }
                /****************************
                        Rules Evaluation
                ****************************/
```

```
                shay_rl(fuzzy_mat,Rules_m);
                /*****************************
                        BORDERS
                *****************************/
                shay_bdr(rules_mat,Out_mat);
                /*****************************
                        deffuzification
                *****************************/
                U_MAIN= shay_dff(bdr_mat,Out_mat);
                Main_FLC_k=fabs(U_MAIN);
        return  (U_MAIN);
        }


/********************
sw_sct1
********************/
float sw_sct1(float Input_1,float Input_2)
        {
        float i,swang;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=56.25;
        float Min_In_1=11.25;
        float   In_1[6][4]=     {
                                {5      ,4      ,0      },
                                {00.0000, 11.2500, 22.5000},
                                {11.2500, 22.5000, 33.7500},
                                {22.5000, 33.7500, 45.0000},
                                {33.7500, 45.0000, 56.2500},
                                {45.0000, 56.2500, 58.0000}
                                };
        /*********************
        INPUT 2 definitions
        *********************/
        int No_In_2=5;
        float Max_In_2=0.2;
        float Min_In_2=0;
        float   In_2[6][4]=     {
                                {5      ,4      ,0      },
                                {-0.0200,0.0000 ,0.0100 },
                                {0.0000 ,0.0200 ,0.0500 },
                                {0.0200 ,0.0500 ,0.1000 },
                                {0.0500 ,0.1000 ,0.1200 },
                                {0.1000 ,0.2000 ,0.2100}
                                };
        /*************************
        OUTPUT CLASSES DEFINITIN
        *************************/
        float Out_mat[8][3]= {
                                {7      ,3      ,7      },
                                {00.0000, 00.0000, 11.2500},
                                {00.0000, 11.2500, 22.5000},
                                {11.2500, 22.5000, 33.7500},
                                {22.5000, 33.7500, 45.0000},
                                {33.7500, 45.0000, 56.2500},
                                {45.0000, 56.2500, 68.5000},
                                {56.2500, 67.5000, 68.5000}
                                };
        /*****************************
        Rules
        *****************************/
        int Rules_m[6][20]=  {
                {5 ,5 ,7 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*1*/   {2 ,3 ,4 ,5 ,6 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*2*/   {3 ,4 ,5 ,6 ,6 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*3*/   {4 ,5 ,6 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*4*/   {5 ,6 ,6 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*5*/   {6 ,6 ,7 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

```
                        };
/*****************************
SCALLING Input 1
*****************************/
if (Input_1>Max_In_1)
        Input_1=Max_In_1;
if (Input_1<Min_In_1)
        Input_1=Min_In_1;
/*******************************
generation of final in_1 matrix
*******************************/
for (i=1;i<No_In_1+1;i++)
        {
        In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
        }
/*****************************
SCALLING Input 2
*****************************/
if (Input_2>Max_In_2)
        Input_2=Max_In_2;
if (Input_2<Min_In_2)
        Input_2=Min_In_2;
/*******************************
generation of final in_2 matrix
*******************************/
for (i=1;i<No_In_2+1;i++)
        {
        In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
        }
/*****************************
        FUZZIFICATION
*****************************/
if (No_Input==1)
        { /*sinle input*/
                shay_fuz(Input_1,In_1,Input_1,In_1,1);
        }
 if (No_Input==2)
        { /* two inputs */
                shay_fuz(Input_1,In_1,Input_2,In_2,2);
 /*             printf("\n input2=%f    input 1=%f ",Input_2,Input_1);*/
 }
/*****************************
        Rules Evaluation
*****************************/
        shay_rl(fuzzy_mat,Rules_m);

/*****************************
        BORDERS
*****************************/
        shay_bdr(rules_mat,Out_mat);
/*****************************
        deffuzification
*****************************/
         swang=shay_dff(bdr_mat,Out_mat);
return(swang);
}


/*******************
sw_sct2
*******************/
float sw_sct2(float Input_1,float Input_2)
        {
        float i,swang;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=90;
        float Min_In_1=45;
```

```
float	In_1[6][4]=		{
					{5	,4	,0	},
					{33.7500,45.0000,56.25000 },
					{45.0000, 56.2500, 67.5000},
					{56.2500, 67.5000, 78.7500},
					{67.5000, 78.7500, 90.0000},
					{78.7500, 90.0000, 91.0000}
					};
/*********************
INPUT 2 definitions
*********************/
int No_In_2=5;
float Max_In_2=0.2;
float Min_In_2=0;
float	In_2[6][4]=		{
					{5	,4	,0	},
					{-0.0200,0.0000	,0.0100	},
					{0.0000	,0.0200	,0.0500	},
					{0.0200	,0.0500	,0.1000	},
					{0.0500	,0.1000	,0.1200	},
					{0.1000	,0.2000	,0.2100}
					};
/****************************
OUTPUT CLASSES DEFINITIN
****************************/
int No_Out=14;
float Out_mat[8][3]= {
					{7	,3	,7	},
					{33.7500 ,33.7500, 45.0000},
					{33.7500,45.0000,56.25000 },
					{45.0000, 56.2500, 67.5000},
					{56.2500, 67.5000, 78.7500},
					{67.5000, 78.7500, 90.0000},
					{78.7500, 90.0000, 101.2500},
					{90.000, 101.2500, 102}
					};
/*******************************
Rules
*******************************/
int Rules_m[6][20]=  {
		{5 ,5 ,7 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*1*/	{2 ,3 ,3 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*2*/	{3 ,4 ,5 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*3*/	{3 ,4 ,5 ,5 ,6 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*4*/	{4 ,5 ,5 ,6 ,6 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*5*/	{4 ,5 ,6 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
					};
/*****************************************************************************
				END DEFFINITIONS
*****************************************************************************/
		/****************************
		SCALLING Input 1
		****************************/
		if (Input_1>Max_In_1)
			Input_1=Max_In_1;
		if (Input_1<Min_In_1)
			Input_1=Min_In_1;
		/*******************************
		generation of final in_1 matrix
		*******************************/
		for (i=1;i<No_In_1+1;i++)
			{
			In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
			}
		/****************************
		SCALLING Input 2
		****************************/
		if (Input_2>Max_In_2)
			Input_2=Max_In_2;
		if (Input_2<Min_In_2)
			Input_2=Min_In_2;
		/*******************************
```

```
                generation of final in_2 matrix
                ********************************/

                for (i=1;i<No_In_2+1;i++)
                        {
                        In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                        }
                /*****************************
                        FUZZIFICATION
                *****************************/
                if (No_Input==1)
                        { /*sinle input*/
                                shay_fuz(Input_1,In_1,Input_1,In_1,1);
                        }
                 if (No_Input==2)
                        { /* two inputs */
                                shay_fuz(Input_1,In_1,Input_2,In_2,2);
                 }
                /*****************************
                        Rules Evaluation
                *****************************/
                        shay_rl(fuzzy_mat,Rules_m);
                /*****************************
                        BORDERS
                *****************************/
                        shay_bdr(rules_mat,Out_mat);
                /*****************************
                        deffuzification
                *****************************/
                         swang=shay_dff(bdr_mat,Out_mat);
                return(swang);
                }


/*******************
sw_sct3
*******************/
float sw_sct3(float Input_1,float Input_2)
        {
        float i,swang;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=180.0;
        float Min_In_1=90;
        float   In_1[6][4]=     {
                                {5          ,4          ,0          },
                                {67.5000, 90.0000,112.50000},
                                {90.0000, 112.500, 135.000},
                                {112.500, 135.000, 157.500},
                                {135.000, 157.500, 180.000},
                                {157.500, 180.000, 181.000}
                                };
        /*********************
        INPUT 2 definitions
        *********************/
        int No_In_2=5;
        float Max_In_2=0.2;
        float Min_In_2=0;
        float   In_2[6][4]=     {
                                {5          ,4          ,0          },
                                {-0.0200,0.0000     ,0.0100     },
                                {0.0000     ,0.0200     ,0.0500     },
                                {0.0200     ,0.0500     ,0.1000     },
                                {0.0500     ,0.1000     ,0.1200     },
                                {0.1000     ,0.2000     ,0.2100}
                                };
        /*************************
        OUTPUT CLASSES DEFINITIN
        *************************/
```

```
int No_Out=14;
float Out_mat[8][3]= {
                        {7          ,3          ,7          },
                        {67.5000 ,90.0000, 112.5000},
                        {67.5000, 90.0000, 112.5000},
                        {90.0000, 112.500, 135.000},
                        {112.500, 135.000, 157.500},
                        {135.000, 157.500, 180.000},
                        {157.500, 180.000, 202.5000},
                        {157.500, 202.5000, 203.000}
                        };
/*****************************
Rules
*****************************/
int Rules_m[6][20]= {
        {5 ,5 ,7 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*1*/   {2 ,2 ,3 ,3 ,4 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*2*/   {3 ,3 ,4 ,4 ,5 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*3*/   {4 ,5 ,6 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*4*/   {5 ,6 ,7 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
/*5*/   {6 ,7 ,7 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                        };
/*****************************************************************************
                        END DEFFINITIONS
*****************************************************************************/
        /*****************************
        SCALLING Input 1
        *****************************/
        if (Input_1>Max_In_1)
                Input_1=Max_In_1;
        if (Input_1<Min_In_1)
                Input_1=Min_In_1;
        /*********************************
        generation of final in_1 matrix
        *********************************/
        for (i=1;i<No_In_1+1;i++)
                {
                In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                }
        /*****************************
        SCALLING Input 2
        *****************************/
        if (Input_2>Max_In_2)
                Input_2=Max_In_2;
        if (Input_2<Min_In_2)
                Input_2=Min_In_2;
        /*********************************
        generation of final in_2 matrix
        *********************************/
        for (i=1;i<No_In_2+1;i++)
                {
                In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                }
        /*****************************
                FUZZIFICATION
        *****************************/
        if (No_Input==1)
                { /*sinle input*/
                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                }
         if (No_Input==2)
                { /* two inputs */
                        shay_fuz(Input_1,In_1,Input_2,In_2,2);
                }
        /*****************************
                Rules Evaluation
        *****************************/
                shay_rl(fuzzy_mat,Rules_m);
        /*****************************
                BORDERS
        *****************************/
                shay_bdr(rules_mat,Out_mat);
```

```
                /*****************************
                        deffuzification
                *****************************/
                        swang=shay_dff(bdr_mat,Out_mat);
                return(swang);
                }
/*******************
sw_sct4
*******************/
float sw_sct4(float Input_1,float Input_2)
        {
        float i,swang;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=180;
        float Min_In_1=225;
        float   In_1[6][4]=     {
                                {5      ,4      ,0      },
                                {168.000, 180.000, 191.2500 },
                                {180.000, 191.250, 202.500},
                                {191.250, 202.500, 213.750},
                                {202.500, 213.750, 225.000},
                                {213.750, 225.000, 226.000}
                                };
        /*********************
        INPUT 2 definitions
        *********************/
        int No_In_2=5;
        float Max_In_2=0.2;
        float Min_In_2=0;
        float   In_2[6][4]=     {
                                {5      ,4      ,0      },
                                {-0.0200,0.0000 ,0.0100 },
                                {0.0000 ,0.0200 ,0.0500 },
                                {0.0200 ,0.0500 ,0.1000 },
                                {0.0500 ,0.1000 ,0.1200 },
                                {0.1000 ,0.2000 ,0.2100}
                                };
        /*****************************
        OUTPUT CLASSES DEFINITIN
        *****************************/
        int No_Out=14;
        float Out_mat[8][3]= {
                                {7      ,3      ,7      },
                                {168.750, 168.75 ,180.00 },
                                {168.000, 180.000, 191.2500 },
                                {180.000, 191.250, 202.500},
                                {191.250, 202.500, 213.750},
                                {202.500, 213.750, 225.000},
                                {213.750, 225.000, 226.000},
                                {213.750, 225.000, 236.250}
                                };
        /*******************************
        Rules
        *******************************/
        int Rules_m[6][20]=  {
                {5 ,5 ,7 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*1*/   {2 ,3 ,5 ,5 ,6 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*2*/   {3 ,4 ,5 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*3*/   {4 ,5 ,5 ,6 ,6 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*4*/   {5 ,6 ,6 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*5*/   {6 ,7 ,7 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                                };
/*************************************************************************
                        END DEFFINITIONS
*************************************************************************/
                /*************************
                SCALLING Input 1
                *************************/
```

```
        if (Input_1>Max_In_1)
                Input_1=Max_In_1;
        if (Input_1<Min_In_1)
                Input_1=Min_In_1;
        /*******************************
        generation of final in_1 matrix
        *******************************/
        for (i=1;i<No_In_1+1;i++)
                {
                In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                }
        /*****************************
        SCALLING Input 2
        *****************************/
        if (Input_2>Max_In_2)
                Input_2=Max_In_2;
        if (Input_2<Min_In_2)
                Input_2=Min_In_2;
        /*******************************
        generation of final in_2 matrix
        *******************************/
        for (i=1;i<No_In_2+1;i++)
                {
                In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                }
        /*****************************
                FUZZIFICATION
        *****************************/
        if (No_Input==1)
                { /*sinle input*/
                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                }
         if (No_Input==2)
                { /* two inputs */
                        shay_fuz(Input_1,In_1,Input_2,In_2,2);
                 }
        /*****************************
                Rules Evaluation
        *****************************/
                shay_rl(fuzzy_mat,Rules_m);
        /*****************************
                BORDERS
        *****************************/
                shay_bdr(rules_mat,Out_mat);
        /*****************************
                deffuzification
        *****************************/
                 swang=shay_dff(bdr_mat,Out_mat);
        return(swang);
        }


/*******************
sw_sct5
*******************/
float sw_sct5(float Input_1,float Input_2)
        {
        float i,swang;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=270;
        float Min_In_1=225;
        float   In_1[6][4]=     {
                                {5      ,4      ,0      },
                                {213.750, 225.000,236.2500 },
                                {225.000, 236.250, 247.500},
                                {236.250, 247.500, 258.750},
                                {247.500, 258.750, 270.000},
                                {258.750, 270.000, 271.000}
```

```
                                        };
        /*********************
        INPUT 2 definitions
        *********************/
        int No_In_2=5;
        float Max_In_2=0.2;
        float Min_In_2=0;
        float       In_2[6][4]=         {
                                        {5          ,4          ,0          },
                                        {-0.0200,0.0000         ,0.0100     },
                                        {0.0000     ,0.0200     ,0.0500     },
                                        {0.0200     ,0.0500     ,0.1000     },
                                        {0.0500     ,0.1000     ,0.1200     },
                                        {0.1000     ,0.2000     ,0.2100}
                                        };
        /***************************
        OUTPUT CLASSES DEFINITIN
        ***************************/
        int No_Out=14;
        float Out_mat[8][3]=  {
                                        {7          ,3          ,7          },
                                        {213.750, 213.750, 225.000 },
                                        {213.750, 225.000, 236.2500 },
                                        {225.000, 236.250, 247.500},
                                        {236.250, 247.500, 258.750},
                                        {247.500, 258.750, 270.000},
                                        {258.750, 270.000, 271.000},
                                        {270.000, 281.25,282}
                                        };
        /*******************************
        Rules
        *******************************/
        int Rules_m[6][20]=   {
                {5 ,5  ,7  ,0  ,0  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*1*/   {2  ,3  ,3  ,4  ,4  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*2*/   {3  ,3  ,4  ,4  ,6  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*3*/   {3  ,4  ,4  ,5  ,6  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*4*/   {3  ,4  ,5  ,6  ,6  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*5*/   {4  ,5  ,6  ,7  ,7  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                                        };
/*****************************************************************************
                        END DEFFINITIONS
*****************************************************************************/
                /***************************
                SCALLING Input 1
                ***************************/
                if (Input_1>Max_In_1)
                        Input_1=Max_In_1;
                if (Input_1<Min_In_1)
                        Input_1=Min_In_1;
                /*********************************
                generation of final in_1 matrix
                *********************************/
                for (i=1;i<No_In_1+1;i++)
                        {
                        In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                        }
                /***************************
                SCALLING Input 2
                ***************************/
                if (Input_2>Max_In_2)
                        Input_2=Max_In_2;
                if (Input_2<Min_In_2)
                        Input_2=Min_In_2;
                /*********************************
                generation of final in_2 matrix
                *********************************/
                for (i=1;i<No_In_2+1;i++)
                        {
                        In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                        }
                /***************************
```

```
                FUZZIFICATION
        ****************************/
        if (No_Input==1)
                { /*sinle input*/
                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                }
         if (No_Input==2)
                { /* two inputs */
                        shay_fuz(Input_1,In_1,Input_2,In_2,2);
         }
        /****************************
                Rules Evaluation
        ****************************/
                shay_rl(fuzzy_mat,Rules_m);
        /****************************
                BORDERS
        ****************************/
                shay_bdr(rules_mat,Out_mat);
        /****************************
                deffuzification
        ****************************/
                 swang=shay_dff(bdr_mat,Out_mat);
        return(swang);
        }


/*******************
sw_sct6
*******************/
float sw_sct6(float Input_1,float Input_2)
        {
        float i,swang;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=5;
        float Max_In_1=360;
        float Min_In_1=270;
        float   In_1[6][4]=     {
                                {5      ,4      ,0      },
                                {247.500, 270.000, 292.5000 },
                                {270.000, 292.500, 315.000},
                                {292.500, 315.00, 337.500},
                                {315.000, 337.500, 360.000},
                                {337.500, 360.000, 382.500}
                                };
        /*********************
        INPUT 2 definitions
        *********************/
        int No_In_2=5;
        float Max_In_2=0.2;
        float Min_In_2=0;
        float   In_2[6][4]=     {
                                {5      ,4      ,0      },
                                {-0.0200,0.0000 ,0.0100 },
                                {0.0000 ,0.0200 ,0.0500 },
                                {0.0200 ,0.0500 ,0.1000 },
                                {0.0500 ,0.1000 ,0.1200 },
                                {0.1000 ,0.2000 ,0.2100}
                                };
        /***************************
        OUTPUT CLASSES DEFINITIN
        ***************************/
        int No_Out=14;
        float Out_mat[8][3]=  {
                                {7      ,3      ,7      },
                                {247.500, 247.500,270.000 },
                                {247.500, 270.000, 292.5000 },
                                {270.000, 292.500, 315.000},
                                {292.500, 315.00, 337.500},
                                {315.000, 337.500, 360.000},
```

```
                                        {337.500, 360.000, 382.500},
                                        {360.000, 382.500, 383.00}
                                        };
        /*****************************
        Rules
        *****************************/
        int Rules_m[6][20]=  {
                {5 ,5 ,7 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*1*/   {2 ,2 ,3 ,4 ,5 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*2*/   {3 ,3 ,4 ,5 ,5 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*3*/   {4 ,4 ,5 ,5 ,5 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*4*/   {5 ,5 ,6 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*5*/   {6 ,6 ,7 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                                        };
/*****************************************************************************
                        END DEFFINITIONS
*****************************************************************************/
                /*****************************
                SCALLING Input 1
                *****************************/
                if (Input_1>Max_In_1)
                        Input_1=Max_In_1;
                if (Input_1<Min_In_1)
                        Input_1=Min_In_1;
                /*********************************
                generation of final in_1 matrix
                *********************************/
                for (i=1;i<No_In_1+1;i++)
                        {
                        In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                        }
                /*****************************
                SCALLING Input 2
                *****************************/
                if (Input_2>Max_In_2)
                        Input_2=Max_In_2;
                if (Input_2<Min_In_2)
                        Input_2=Min_In_2;
                /*********************************
                generation of final in_2 matrix
                *********************************/
                for (i=1;i<No_In_2+1;i++)
                        {
                        In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                        }
                /*****************************
                        FUZZIFICATION
                *****************************/
                if (No_Input==1)
                        { /*sinle input*/
                                shay_fuz(Input_1,In_1,Input_1,In_1,1);
                        }
                 if (No_Input==2)
                        { /* two inputs */
                                shay_fuz(Input_1,In_1,Input_2,In_2,2);
                 }
                /*****************************
                        Rules Evaluation
                *****************************/
                        shay_rl(fuzzy_mat,Rules_m);
                /*****************************
                        BORDERS
                *****************************/
                        shay_bdr(rules_mat,Out_mat);
                /*****************************
                        deffuzification
                *****************************/
                         swang=shay_dff(bdr_mat,Out_mat);
                return(swang);
                }

/*******************
```

```
PRE_CONTROL
*******************/
float PRE_CONTROL(float Input_1,float Input_2)
        {
        float SCLD_DW;
        int No_Input=2,i;
        /********************
        INPUT 1 definitions
        ********************/
        int No_In_1=9;
        float Max_In_1=1;
        float Min_In_1=-1;
        float       In_1[10][4]=        {
                                        {9          ,4          ,0          },
                                        {-1.1000, -1.0000, -0.7500 },
                                        {-1.0000, -0.7500, -0.5000 },
                                        {-0.7500, -0.5000, -0.2500 },
                                        {-0.5000, -0.2500,  0.0000 },
                                        {-0.2500,  0.0000,  0.2500 },
                                        { 0.0000,  0.2500,  0.5000 },
                                        { 0.2500,  0.5000,  0.7500 },
                                        { 0.5000,  0.7500,  1.0000 },
                                        { 0.7500,  1.0000,  1.1000 }
                                        };
        /********************
        INPUT 2 definitions
        ********************/
        int No_In_2=9;
        float Max_In_2=1;
        float Min_In_2=-1;
        float       In_2[10][4]=        {
                                        {9          ,4          ,0          },
                                        {-1.1000, -1.0000, -0.7500 },
                                        {-1.0000, -0.7500, -0.5000 },
                                        {-0.7500, -0.5000, -0.2500 },
                                        {-0.5000, -0.2500,  0.0000 },
                                        {-0.2500,  0.0000,  0.2500 },
                                        { 0.0000,  0.2500,  0.5000 },
                                        { 0.2500,  0.5000,  0.7500 },
                                        { 0.5000,  0.7500,  1.0000 },
                                        { 0.7500,  1.0000,  1.1000 }
                                        };
        /***************************
        OUTPUT CLASSES DEFINITIN
        ***************************/
        int No_Out=14;
        float Out_mat[10][3]=       {
                                        {9          ,3          ,9          },
                                        {-1.1000, -1.0000, -0.7500 },
                                        {-0.8000, -0.500, -0.3000 },
                                        {-0.3500, -0.1000, -0.0500 },
                                        {-0.1000, -0.0500,  0.0000 },
                                        {-0.100,  0.0000,  0.100 },
                                        { 0.0000,  0.0500,  0.1000 },
                                        { 0.0500,  0.1000,  0.3500 },
                                        { 0.3000,  0.500,  0.8000 },
                                        { 0.7500,  1.0000,  1.1000 }
                                        };
        /*****************************
        Rules
        *****************************/
        int Rules_m[10][20]= {
                {9 ,9 ,10 ,0 ,0 ,0 ,0 ,0 ,0 ,0,0,0,0,0,0,0,0,0,0,0},
        /*1*/   {1 ,1 ,1 ,2 ,2 ,3 ,4 ,5 ,5 ,0,0,0,0,0,0,0,0,0,0,0},
        /*2*/   {1 ,2 ,2 ,2 ,3 ,4 ,4 ,5 ,5 ,0,0,0,0,0,0,0,0,0,0,0},
        /*3*/   {1 ,2 ,2 ,3 ,4 ,4 ,5 ,5 ,6 ,0,0,0,0,0,0,0,0,0,0,0},
        /*4*/   {1 ,2 ,3 ,4 ,4 ,5 ,5 ,6 ,7 ,0,0,0,0,0,0,0,0,0,0,0},
        /*5*/   {2 ,2 ,3 ,4 ,5 ,6 ,6 ,7 ,7 ,0,0,0,0,0,0,0,0,0,0,0},
        /*6*/   {3 ,4 ,5 ,5 ,6 ,6 ,7 ,8 ,9 ,0,0,0,0,0,0,0,0,0,0,0},
        /*7*/   {4 ,5 ,5 ,6 ,6 ,7 ,8 ,8 ,9 ,0,0,0,0,0,0,0,0,0,0,0},
        /*8*/   {5 ,5 ,6 ,6 ,7 ,8 ,8 ,8 ,9 ,0,0,0,0,0,0,0,0,0,0,0},
        /*9*/   {5 ,5 ,6 ,7 ,8 ,8 ,9 ,9 ,9 ,0,0,0,0,0,0,0,0,0,0,0}
```

```
                                        };
/*****************************************************************************
                                END DEFFINITIONS
*****************************************************************************/
                        /*****************************
                        SCALLING Input 1
                        *****************************/
                        if (Input_1>Max_In_1)
                                Input_1=Max_In_1;
                        if (Input_1<Min_In_1)
                                Input_1=Min_In_1;
                        /*********************************
                        generation of final in_1 matrix
                        *********************************/
                        for (i=1;i<No_In_1+1;i++)
                                {
                                In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                                }
                        /*****************************
                        SCALLING Input 2
                        *****************************/
                        if (Input_2>Max_In_2)
                                Input_2=Max_In_2;
                        if (Input_2<Min_In_2)
                                Input_2=Min_In_2;
                        /*********************************
                        generation of final in_2 matrix
                        *********************************/
                        for (i=1;i<No_In_2+1;i++)
                                {
                                In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
                                }
                        /*****************************
                                FUZZIFICATION
                        *****************************/
                        if (No_Input==1)
                                { /*sinle input*/
                                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                                }
                         if (No_Input==2)
                                { /* two inputs */
                                        shay_fuz(Input_1,In_1,Input_2,In_2,2);
                         }
                        /*****************************
                                Rules Evaluation
                        *****************************/
                                shay_rl(fuzzy_mat,Rules_m);
                        /*****************************
                                BORDERS
                        *****************************/
                                shay_bdr(rules_mat,Out_mat);
                        /*****************************
                                deffuzification
                        *****************************/
                                 SCLD_DW=shay_dff(bdr_mat,Out_mat);
                        return(SCLD_DW);
                        }


/*****************************
opra_sts_in
*****************************/
void opra_sts_in(float Input_1,float Input_2)
        {
        int rl_rows,m;
        float i,R1,R2,R3;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/

        int No_In_1=3;
```

```
float Max_In_1=1;
float Min_In_1=0;
float	In_1[4][4]=		{
				{3	,4	,0	},
				{-0.0100,0.0000	,0.1000	},
				{0.0000	,0.4000	,1.0000	},
				{0.5000	,1.0000	,1.1000	}
				};
/*********************
INPUT 2 definitions
*********************/
int No_In_2=3;
float Max_In_2=1;
float Min_In_2=0;
float	In_2[4][4]=		{
				{3	,4	,0	},
				{-0.0100,0.0000	,0.1000	},
				{0.0000	,0.4000 ,1.0000	},
				{0.5000	,1.0000	,1.1000	}
				};
/*******************************
Rules
*******************************/
int Rules_m[4][20]=  {
		{3 ,3  ,3  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
		{1  ,2  ,2 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
		{2  ,2  ,2 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
		{2  ,2  ,3 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
				};
/*****************************************************************************
			END DEFFINITIONS
*****************************************************************************/
		/***************************
		SCALLING Input 1
		***************************/
		Input_1=Input_1*5;
		if (Input_1>Max_In_1)
			Input_1=Max_In_1;
		if (Input_1<Min_In_1)
			Input_1=Min_In_1;
		/*******************************
		generation of final in_1 matrix
		*******************************/
		for (i=1;i<No_In_1+1;i++)
			{
			In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
			}
		/***************************
		SCALLING Input 2
		***************************/
		Input_2=Input_2*5;
		if (Input_2>Max_In_2)
			Input_2=Max_In_2;
		if (Input_2<Min_In_2)
			Input_2=Min_In_2;
		/*******************************
		generation of final in_2 matrix
		*******************************/
		for (i=1;i<No_In_2+1;i++)
			{
			In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
			}
		/***************************
			FUZZIFICATION
		***************************/
		if (No_Input==1)
			{
			/*sinle input*/
			shay_fuz(Input_1,In_1,Input_1,In_1,1);
			}
		 if (No_Input==2)
			{
```

```
                    /*two inputs*/
                    shay_fuz(Input_1,In_1,Input_2,In_2,2);
                    }
                /*****************************
                    Rules Evaluation
                *****************************/
                 shay_rl(fuzzy_mat,Rules_m);
                rl_rows=rules_mat[0][0];
                R1=0;
                R2=0;
                R3=0;
                for (m=1;m<=rl_rows;m++)
                                {
                                if (rules_mat[m][0]==1)
                                        R1=rules_mat[m][1];
                                if (rules_mat[m][0]==2)
                                        R2=rules_mat[m][1];
                                if (rules_mat[m][0]==3)
                                        R3=rules_mat[m][1];
                                }
                        flc_sts[0]=R1;
                        flc_sts[1]=R2;
                        flc_sts[2]=R3;
                }


/*****************************
opra_sts_out
*****************************/
void opra_sts_out(float Input_1,float Input_2)
        {
        int rl_rows,m;
        float i,R1,R2,R3;
        int No_Input=2;
        /*********************
        INPUT 1 definitions
        *********************/
        int No_In_1=3;
        float Max_In_1=1;
        float Min_In_1=0;
        float   In_1[4][4]=     {
                                {3      ,4      ,0      },
                                {-0.0100,0.0000 ,0.0500 },
                                {0.0000 ,0.4000 ,1.0000 },
                                {0.5000 ,1.0000 ,1.1000 }
                                };
        /*********************
        INPUT 2 definitions
        *********************/
        int No_In_2=3;
        float Max_In_2=1;
        float Min_In_2=0;
        float   In_2[4][4]=     {
                                {3      ,4      ,0      },
                                {-0.0100,0.0000 ,0.0500 },
                                {0.0000 ,0.4000 ,1.0000 },
                                {0.5000 ,1.0000 ,1.1000 }
                                };
        /*******************************
        Rules
        *******************************/
        int Rules_m[4][20]=  {
                {3 ,3  ,3  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {1  ,2  ,2 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {2  ,2  ,2 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {2  ,2  ,3 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                                };
/*****************************************************************************
                        END DEFFINITIONS
*****************************************************************************/
                /*****************************
                SCALLING Input 1
```

```
*****************************/
if (Input_1>Max_In_1)
        Input_1=Max_In_1;
if (Input_1<Min_In_1)
        Input_1=Min_In_1;
/*******************************
generation of final in_1 matrix
*******************************/
for (i=1;i<No_In_1+1;i++)
        {
        In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
        }
/*****************************
SCALLING Input 2
*****************************/
if (Input_2>Max_In_2)
        Input_2=Max_In_2;
if (Input_2<Min_In_2)
        Input_2=Min_In_2;
/*******************************
generation of final in_2 matrix
*******************************/
for (i=1;i<No_In_2+1;i++)
        {
        In_2[i][3]=bez_bck(Input_2,In_2[i][0],In_2[i][1],In_2[i][1],In_2[i][2]);
        }
/*****************************
        FUZZIFICATION
*****************************/
if (No_Input==1)
        {
        /*sinle input*/
        shay_fuz(Input_1,In_1,Input_1,In_1,1);
        }
 if (No_Input==2)
        {
        /*two inputs*/
        shay_fuz(Input_1,In_1,Input_2,In_2,2);
        }
/*****************************
        Rules Evaluation
*****************************/
 shay_rl(fuzzy_mat,Rules_m);
rl_rows=rules_mat[0][0];
R1=0;
R2=0;
R3=0;
for (m=1;m<=rl_rows;m++)
                {
                if (rules_mat[m][0]==1)
                        R1=rules_mat[m][1];
                if (rules_mat[m][0]==2)
                        R2=rules_mat[m][1];
                if (rules_mat[m][0]==3)
                        R3=rules_mat[m][1];
                }
        flc_sts[0]=R1;
        flc_sts[1]=R2;
        flc_sts[2]=R3;
}


/**************
Monkey
***************/
float Monkey(float Input_1)
        {
        float i,U_MAIN;
        int No_Input=1;
        /********************
        INPUT 1 definitions
        ********************/
```

```
        int No_In_1=3;
        float Max_In_1=0.1;
        float Min_In_1=0;
        float       In_1[4][4]=         {
                                        {3          ,4          ,0          },
                                        {-0.0200,0.0000         ,0.0200     },
                                        {0.0000     ,0.0400     ,0.0700     },
                                        {0.0600     ,0.1000     ,0.1100     }
                                        };
        /***************************
        OUTPUT CLASSES DEFINITIN
        ***************************/
        int No_Out=5;
        float Out_mat[4][3]=  {
                                        {3          ,3          ,0          },
                                        {-0.0200,0.0000         ,0.0200     },
                                        {0.0100     ,0.0300     ,0.0500     },
                                        {0.0400     ,0.1000     ,0.1100     }
                                        };
        /*****************************
        Rules
        ******************************/
        int Rules_m[2][20]=   {
                                        {1  ,3  ,3  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                                        {1  ,2  ,3  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
                                        };
/***************************
  END DEFFINITIONS
***************************/
                Input_1=Input_1;
                /*****************************
                SCALLING Input 1
                *****************************/
                if (Input_1>Max_In_1)
                        Input_1=Max_In_1;
                if (Input_1<Min_In_1)
                        Input_1=Min_In_1;
                /*******************************
                generation of final in_1 matrix
                *******************************/
                for (i=1;i<No_In_1+1;i++)
                        {
                        In_1[i][3]=bez_bck(Input_1,In_1[i][0],In_1[i][1],In_1[i][1],In_1[i][2]);
                        }
                /*****************************
                        FUZZIFICATION
                *****************************/
                if (No_Input==1)
                        {
                        /*sinle input*/
                        shay_fuz(Input_1,In_1,Input_1,In_1,1);
                        }
                /*****************************
                        Rules Evaluation
                *****************************/
                 shay_rl(fuzzy_mat,Rules_m);
                /*****************************
                        BORDERS
                *****************************/
                 shay_bdr(rules_mat,Out_mat);
                /*****************************
                        deffuzification
                *****************************/
                U_MAIN= shay_dff(bdr_mat,Out_mat);
        return  (U_MAIN);
        }
```

APPENDIX B.7

Shay_fz Source Code

Shay_fz Source Code

```
/*****************************
function:   shay_fz
*****************************/
float shay_fz(float ip_1,float thyi1[][4],float ip_2, float thyi2[][4], int no_ip)
        {
        float tol=1e-12;
        int i,rows_x1,rows_x2;
        /*****************
         data required
        *****************/
                /*****************
                 input 1
                *****************/
                rows_x1=thyi1[0][0];

                /*****************
                 input 2
                *****************/
                rows_x2=thyi2[0][0];
        /************************
        start fuzzification
        ************************/
        switch (no_ip)
                {
                case 1:/*single input */
                        {
                        for (i=1;i<rows_x1+1;i++)
                                {
                                /************************
                                detrmine fired classes
                                ************************/
                                if (thyi1[i][3] >tol)
                                        fuz_mat[i][0]=1;
                                else
                                        fuz_mat[i][0]=0;
                                /**********************
                                 detrmine sides
                                **********************/
                                fuz_mat[i][2]=0;
                                if (((i!=1) && (i!=(rows_x1))) && (ip_1<thyi1[i][1]))
                                        fuz_mat[i][2]=1;        /*left*/
                                if (((i!=1) && (i!=(rows_x1))) && (ip_1>thyi1[i][1]))
                                        fuz_mat[i][2]=2;        /*right*/
                                /*****************
                                alpha cuts
                                *****************/
                                fuz_mat[i][1]=thyi1[i][3];
                                }
                        fuz_mat[0][0]=rows_x1;
                        fuz_mat[0][1]=rows_x2;
                        fuz_mat[0][2]=3;
                        break;
                        }
                case 2:     /*two inputs */
                        {
                        for (i=0;i<rows_x1+1;i++)   /* for 1st input*/
                                {
                                /*********************
                                detrmine fired classes
                                *********************/
                                if (thyi1[i][3] >tol)
                                        fuz_mat[i][0]=1;
                                else
                                        fuz_mat[i][0]=0;
                                /*********************
                                detrmine sides
                                *********************/

                                fuz_mat[i][2]=0;
```

```
                        if (((i!=1) && (i!=(rows_x1))) && (ip_1<thyi1[i][1]))
                                fuz_mat[i][2]=1;        /*left*/
                        if (((i!=1) && (i!=(rows_x1))) && (ip_1>thyi1[i][1]))
                                fuz_mat[i][2]=2;        /*right*/
                        /******************
                        alpha cuts
                        ******************/
                        fuz_mat[i][1]=thyi1[i][3];
                        }
                /* input 2 fuz_mat2*/
                for (i=1;i<rows_x2+1;i++)  /* for 2st input */
                        {
                        /********************
                        detrmine fired classes
                        ********************/
                        if (thyi2[(i)][3] >tol)
                                fuz_mat[i+rows_x1][0]=1;
                        else
                                fuz_mat[i+rows_x1][0]=0;
                        /****************
                        detrmine sides
                        ****************/
                        fuz_mat[i+rows_x1][2]=0;
                        if ((((i)!=1) && ((i)!=(rows_x2))) && (ip_2<thyi2[(i)][1]))
                                fuz_mat[i+rows_x1][2]=1;        /*left*/
                        if ((((i)!=1) && ((i)!=(rows_x2))) && (ip_2>thyi2[(i)][1]))
                                fuz_mat[i+rows_x1][2]=2;        /*right*/
                        /****************
                        alpha cuts
                        ****************/
                        fuz_mat[i+rows_x1][1]=thyi2[(i)][3];
                        } /* end of generating fuz_mat1 */
                fuz_mat[0][0]=rows_x1+rows_x2;
                fuz_mat[0][1]=3;
                fuz_mat[0][2]=3;
                } /*end of multiple input*/
        }
        return 0;
}
```

APPENDIX B.8

Shay_rl Source Code

Shay_rl Source Code

```
/************************
start RULE EVALUATION
************************/
float shay_rl(float fuzzy_in[][3],int rule_mat[][20])
        {
        int No_In_1,No_In_2,indx,j,i,indx2;
        float rules[20][3],f_rules[20][3],TemP1,TemP2,TemP3;
        No_In_1=rule_mat[0][0];
        No_In_2=rule_mat[0][1];
        indx=0;
        if (No_In_1==1)
                {
                for(j=1;j<=No_In_2;j++)
                        if( fuzzy_in[j][0]!=0)
                                {
                                indx++;
                                rules[indx][0]=rule_mat[1][j-1];
                                rules[indx][1]=fuzzy_in[j][1];
                                rules[indx][2]=0;
                                }
                        rules[0][0]=indx;
                        rules[0][1]=3;
                 }
        /*******************
        multiple inputs
        *******************/
        if (No_In_1>1)
                {
                for(i=1;i<=No_In_1;i++)
                        for(j=1;j<=No_In_2;j++)
                                if (((fuzzy_in[i][0])*(fuzzy_in[No_In_1+j][0]))!=0)
                                        {
                                        indx++;
                                        rules[indx][0]=rule_mat[i][j-1];
                                        rules[indx][1]=min(fuzzy_in[i][1],fuzzy_in[No_In_1+j][1]);
                                        rules[indx][2]=0;
                                        }
                rules[0][0]=indx;
                rules[0][1]=3;
                }
        indx2=0;
        for (i=1;i<indx+1;i++)
                if(rules[i][0]!=100)
                        {
                        indx2++;
                        f_rules[indx2][0]=rules[i][0];
                        f_rules[indx2][1]=rules[i][1];
                        f_rules[indx2][2]=rules[i][2];
                        rules[i][0]=100;
                        for (j=i+1;j<=indx;j++)
                                if (rules[j][0]==f_rules[indx2][0])
                                        {
                                        if (rules[j][1]>f_rules[indx2][1])
                                                {
                                                f_rules[indx2][0]=rules[j][0];
                                                f_rules[indx2][1]=rules[j][1];
                                                f_rules[indx2][2]=rules[j][2];
                                                }
                                        rules[j][0]=100;
                                        }
                        }
        f_rules[0][0]=indx2;
        f_rules[0][1]=3;
 /****************
  SOERTING
  ****************/
        for(i=1;i<indx2;i++)
                 for (j=2;j<=indx2;j++)
                        if (f_rules[i][1]<f_rules[j][1])
```

```
                        {
                        TemP1=f_rules[i][0];
                        TemP2=f_rules[i][1];
                        TemP3=f_rules[i][2];
                        f_rules[i][0]=f_rules[j][0];
                        f_rules[i][1]=f_rules[j][1];
                        f_rules[i][2]=f_rules[j][2];
                        f_rules[j][0]=TemP1;
                        f_rules[j][1]=TemP2;
                        f_rules[j][2]=TemP3;
                        }
indx=0;
for (i=1;i<=indx2;i++)
            if (i<3)
                        {
                        indx++;
                        rules_mat[indx][0]=f_rules[i][0];
                        rules_mat[indx][1]=f_rules[i][1];
                        rules_mat[indx][2]=f_rules[i][2];
                        }
rules_mat[0][0]=indx;
rules_mat[0][1]=3;
return 0;
}
```

APPENDIX B.9

Shay_bdr Source Code

Shay_bdr Source Code

```
/*********************
start:  SHAY BORDER
*********************/
float shay_bdr(float c_c[][10],float thy[][3])
        {
        int r_ot,r_thyo,rthyo,row4,kk,sz;
        float temp1,temp2,temp3,temp4,temp5;
        r_ot=c_c[0][0];
        r_thyo=thy[0][0];
/****************
organisation
****************/
kk=0;
for (rthyo=1;rthyo<r_thyo+1;rthyo++)
        for(row4=1;row4<r_ot+1;row4++)
                if (c_c[row4][0]==(rthyo))
                        {
                        if (c_c[row4][2]==1)
                                {
                                kk++;
                                bdr_mat[kk][0]=c_c[row4][0];
                                bdr_mat[kk][1]=c_c[row4][1];
                                bdr_mat[kk][2]=thy[rthyo][0];
                                bdr_mat[kk][3]=thy[rthyo][1];
                                bdr_mat[kk][4]=c_c[row4][2];
                                }
                        if (c_c[row4][2]==4)
                                {
                                kk++;
                                bdr_mat[kk][0]=c_c[row4][0];
                                bdr_mat[kk][1]=c_c[row4][1];
                                bdr_mat[kk][2]=thy[rthyo][1];
                                bdr_mat[kk][3]=thy[rthyo][2];
                                bdr_mat[kk][4]=c_c[row4][2];
                                }
                        if ((c_c[row4][2]!=1) && (c_c[row4][2]!=4))
                                {
                                kk++;
                                bdr_mat[kk][0]=c_c[row4][0];
                                bdr_mat[kk][1]=c_c[row4][1];
                                bdr_mat[kk][2]=thy[rthyo][0];
                                bdr_mat[kk][3]=thy[rthyo][2];
                                bdr_mat[kk][4]=c_c[row4][2];
                                }
                        }
                                bdr_mat[0][0]=kk;
                                bdr_mat[0][1]=5;
                                bdr_mat[0][2]=kk;
                                bdr_mat[0][3]=kk;
                                bdr_mat[0][4]=kk;
sz=bdr_mat[0][0];
/*************
filter
*************/
        if (sz==2)
                if (bdr_mat[1][1] < bdr_mat[2][1])
                        {
                        temp1=bdr_mat[2][0];
                        temp2=bdr_mat[2][1];
                        temp3=bdr_mat[2][2];
                        temp4=bdr_mat[2][3];
                        temp5=bdr_mat[2][4];
                        bdr_mat[2][0]=bdr_mat[1][0];
                        bdr_mat[2][1]=bdr_mat[1][1];
                        bdr_mat[2][2]=bdr_mat[1][2];
                        bdr_mat[2][3]=bdr_mat[1][3];
                        bdr_mat[2][4]=bdr_mat[1][4];
                        bdr_mat[1][0]=temp1;
                        bdr_mat[1][1]=temp2;
```

```
                    bdr_mat[1][2]=temp3;
                    bdr_mat[1][3]=temp4;
                    bdr_mat[1][4]=temp5;
                    }
return 0;
}
```

APPENDIX B.10

Shay_dff Source Code

Shay_dff Source Code

```
/*******************
start shay_dff
*****************/
float shay_dff(float rmat[][5],float bezs[][3])
        {
        float sum1,sum2,s,ss,a,stp1,y1,s11,s12,s21,s22,ss1,s1,a1,a2,stp2,y2,y,KK,crisp_out=0.0000;
        int r_mat,e1,e2;
        r_mat=rmat[0][0];
        if (r_mat==1)
                {
                sum1=0;
                sum2=0;
                s=min(rmat[1][2],rmat[1][3]);
                ss=max(rmat[1][2],rmat[1][3]);
                stp1=(ss-s)/25;
                e1=rmat[1][0];
                a=rmat[1][1];
                for (KK=s;KK<=ss;KK=KK+stp1)
                        {
                        y1=bez_bck(KK,bezs[e1][0],bezs[e1][1],bezs[e1][1],bezs[e1][2]);
                        if (y1>a)
                                y=a;
                        else
                                y=y1;
                        sum1=y*KK+sum1;
                        sum2=y+sum2;
                        }
                crisp_out=sum1/sum2;
                }
        if (r_mat==2)
                {
                sum1=0;
                sum2=0;
                s11=min(rmat[1][2],rmat[1][3]);
                s12=max(rmat[1][2],rmat[1][3]);
                s21=min(rmat[2][2],rmat[2][3]);
                s22=max(rmat[2][2],rmat[2][3]);
                ss1=max(s22,s12);
                s1=min(s11,s21);
                e1=rmat[1][0];
                e2=rmat[2][0];
                a1=rmat[1][1];
                a2=rmat[2][1];
                stp2=(ss1-s1)/35;
                for (KK=s1;KK<ss1;KK=KK+stp2)
                {
                        y1=bez_bck(KK,bezs[e1][0],bezs[e1][1],bezs[e1][1],bezs[e1][2]);
                        y2=bez_bck(KK,bezs[e2][0],bezs[e2][1],bezs[e2][1],bezs[e2][2]);
                        if (y2>a2)
                                y2=a2;
                        if (y1>a1)
                                y1=a1;
                        if (y1>y2)
                                y=y1;
                        else
                                y=y2;
                        sum1=y*KK+sum1;
                        sum2=y+sum2;
                        }
                crisp_out=sum1/(sum2);
        }
        return crisp_out;
        }
```