AUTOMATED DESIGN OF MULTI-MODE FUZZY CONTROLLERS

ETIENNE M. HUGO

PROMOTER: PROF. J.J. DU PLESSIS

AUTOMATED DESIGN OF MULTI-MODE FUZZY CONTROLLERS

ETIENNE M. HUGO

Dissertation presented for the Degree of Doctor of Philosophy in Engineering at the University of Stellenbosch. Promoter: Prof. J.J. du Plessis.

December 2000

DECLARATION

I, the undersigned, hereby declare that the work contained in this dissertation is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

ABSTRACT

A standard fuzzy logic controller is not robust enough to guarantee consistent closed-loop performance for highly non-linear plants. A finely tuned closed-loop response loses relevance as the system dynamics change with operating conditions. The self-adaptive fuzzy logic controller can track changes in the system parameters and modify the controller parameters accordingly. In most cases, self-adaptive fuzzy logic controllers are complex and rely on some form of mathematical plant model.

The multi-mode fuzzy logic controller extends the working range of a standard fuzzy logic controller by incorporating knowledge of the non-linear system dynamics into the control rule-base. The complexity of the controller and difficulty in finding control rules have limited the application of multi-mode fuzzy logic controllers.

An automated design algorithm is proposed for the design of a multi-mode control rule-base using qualitative plant knowledge. The design algorithm is cost function-based. The closed-loop response, local to a domain of the non-linear state space, can be tuned by manipulation of the cost function weights. Global closed-loop response tuning can be done by manipulation of the controller input gains. Alternatively, a self-learning or self-adaptive algorithm can be used in a model reference adaptive control architecture to optimise the control rule-base. Control rules responsible for unacceptable closed-loop performance are identified and their consequences modified.

The validity of the proposed design method is evaluated in five case studies. The case studies illustrate the advantages of the multi-mode fuzzy logic controller. The results indicate that the proposed self-adaptive algorithm can be used to optimise a rule-base given a required closed-loop specification. If the system does not conform to the model reference adaptive architecture then the intuitive nature of the cost function based design algorithm proves to be an effective method for rule-base tuning.

OPSOMMING

Standaard wasige logika beheerders is nie noodwendig robuust genoeg om goeie geslote lus werkverrigting vir hoogs nie-liniere aanlegte te waarborg nie. 'n Perfek ge-optimeerde beheerder se geslote lus werkverrigting mag verswak indien die aanleg-parameters weens bedryfstoestande verander. Self-aanpassende beheerders kan die verandering in die aanleg-parameters volg en die beheerder dienooreenkomstig optimeer. As 'n reël is 'n self-aanpassende beheerder kompleks en afhanklik van 'n wiskundige model van die aanleg.

Die multi-modus wasige logika beheerder vergroot die werksbereik van die standaard wasige logika beheerder deur kennis aangaande die stelsel se bedryfstoestand en stelselparameters in die reël-basis in te bou. Die aanwending van die multi-modus beheerder word tans beperk deur die struktuur kompleksiteit en moeilike optimering van die reël-basis.

'n Ge-outomatiseerde multi-modus reël-basis ontwerps-algoritme wat gebruik maak van kwalitatiewe kennis van die aanleg en 'n kostefunksie word in hierdie proefskrif voorgestel. Die geslote lus gedrag beperk tot 'n gebied in die toestands-ruimte kan ge-optimeer word deur die kostefunksie gewigte te manipuleer. Die globale werkverrigting kan ge-optimeer word met die beheerder intree aanwinste. 'n Self-aanpassende algoritme in 'n model-verwysings aanpassende argitektuur word as alternatief tot reël-basis optimering voorgestel. Reëls verantwoordelik vir swak werkverrigting word ge-identifiseer en verbeter deur modifikasie van die reëls se gevolgtrekkings.

Die voorgestelde ontwerps-metode word deur middel van vyf gevallestudies ondersoek. Die studies dui die voordele van die multi-modus struktuur aan. Die self-aanpassende argitektuur is 'n kragtige hulpbron om 'n reël-basis te optimeer vir 'n gegewe geslote lus spesifikasie. Hierdie proefskrif toon aan dat indien die stelsel nie aan die vereistes van 'n model verwysingstelsel voldoen nie, is die kostefunksie benadering tot reël-basis ontwerp 'n aantreklike en intuïtief verstaanbare opsie om die reël-basis te optimeer.

ACKNOWLEDGEMENTS

I would like to thank the following people for their contributions and help in this study.

My sponsors, the FRD and the Dept. of Electric and Electronic Engineering at the University of Stellenbosch.

Prof. Jan du Plessis, my supervisor. Thank you for your excellent guidance and supervision. Without your help this study would never have reached completion. Thank you also for being sympathetic in letting me take sunny weekends off to go rock climbing and rainy weekends for diving.

Prof. Herman Steyn and Dr Derick Moolman. Thank you for your help in the initial stages of this study.

Francois Gouws, Gregor Schmidt, JP Barnard and Dirko van Schalkwyk. My friends and fellow students, thank you for your help, advice and good company.

The technical and administrative staff at the Dept. of Electric and Electronic Engineering at the University of Stellenbosch.

Mark Stoneman, Lizette Smith, Prof. Justus Roux and Dr Edwin Hees for proofreading this document and ensuring that at least some of it makes sense.

My parents; thank you for your support and motivation through all my studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVEATIONS

ξ - Damping Ratio.

ω_d – Damped Natural Frequency (rad/s).

ω_n – Undamped Natural Frequency (rad/s).

Cond – Condition.

CSTR – Continuously Stirred Tank Reactor.

E_{ss} – Steady-State Error.

FAM - Fuzzy Associated Memory.

FLC – Fuzzy Logic Controller.

FLS – Fuzzy Logic System.

FMRLC - Fuzzy Model Reference Learning Control.

FSMC - Fuzzy Sliding Mode Controller.

GA - Genetic Algorithm.

G_I = Gain information matrix used in automated rule-base design algorithm.

Ipl – Implication.

K_d – Derivative gain of PID or fuzzy logic controller.

K_i – Integral gain of PID or fuzzy logic controller.

K_p – Proportional gain of PID or fuzzy logic controller.

KPI - Key Performance Indicator.

K_{REd} - Adaptive Fuzzy Logic Controller Response Error Derivative Gain.

K_{REp} – Adaptive Fuzzy Logic Controller Response Error Gain.

MIMO – Multi-Input, Multi-Output.

MMAFLC – Multi-Mode Adaptive Fuzzy Logic Controller.

MMFLC – Multi-Mode Fuzzy Logic Controller.

M_p – Maximum Overshoot.

MRAC – Model Reference Adaptive Control.

MRAS - Model Reference Adaptive System.

PD – Proportional and Derivative Control.

PI – Proportional and Integral Control.

PID – Proportional, Integral and Derivative Control.

RMS – Root Mean Square.

SAFLC – Self-Adaptive Fuzzy Logic Controller.

S_I = Speed information matrix used in automated rule-base design algorithm.

SISO – Single-Input, Single-Output.

SLFLC – Self-Learning Fuzzy Logic Controller.

SMAFLC – Single-Mode Adaptive Fuzzy Logic Controller.

SMFLC – Single-Mode Fuzzy Logic Controller.

SOC – Self-Organizing Controller.

T_r – Rise Time.

$T_{r\,(10\%\,-\,90\%)}$ - 10% to 90% Rise Time.

T_{ss} – Settling Time.

$\mu(x)$ – Membership function.

1 INTRODUCTION

At present control system engineers and designers are confronted with highly complex plants requiring stringent closed-loop control without the benefit of good mathematical models. Plants in the chemical process industry are of a high-dimensional order and non-linear, and the fundamental physics poorly understood. In certain fields, such as the aerospace industry, control systems engineers have a good understanding of plant structure and dynamics which allows mathematical models to be derived. These models are often non-linear, with system dynamics changing substantially as a function of the operating domain. The lack of analytical plant models and the complexity of available models limits the application of model-based controller design techniques.

Fuzzy logic systems were developed to model and mimic human linguistic reasoning. Fuzzy logic systems are rule-based and thus easy to interpret and understand. In many plants expert control knowledge is available in the form of heuristic control rules or operator handbooks that can be implemented in a fuzzy system and used for closed-loop control. This non-model-based approach to controller design provides a solution to complex control problems if expert knowledge is available and it has led to the application of fuzzy logic systems in a wide variety of industrial plants and consumer products.

1.1 MOTIVATION FOR RESEARCH AND PROBLEM STATEMENT

Plant dynamics are generally a function of operating conditions. A controller optimised for a specific operating condition may not perform acceptably for all operating conditions encountered. Gain scheduling and self-adaptive controllers have traditionally been used to modify controller parameters and ensure acceptable performance for the variations in operating conditions [4]. The application of these techniques requires accurate mathematical models of the plant dynamics in all the operating domains. A fuzzy logic controller (FLC) does not require a mathematical plant model for its design and is thus ideally suited to application in systems with complex, unmodelled non-linear dynamics.

Although a standard FLC has proven to be extremely robust, a standard PID-type FLC may not perform acceptably for all operating conditions. Various self-adaptive fuzzy logic control schemes have been proposed [54, 87, 89]. The adaptive mechanism is required to run online to track changes in operating domain. These schemes are often complex and rely on plant knowledge in the form of inverse models or parameter sensitivity functions. Fuzzy inverse models can suffer from identical infinite bandwidth and non-causality as linear inverse models. The derivation of accurate parameter sensitivity functions is complex and limited to special types of fuzzy system such as systems using radial basis function.

Control system designers can often gain valuable qualitative information regarding system parameters, states or domains influencing plant dynamics from plant experts or knowledge of the plant structure. A typical example is asking a pilot how rapidly or sluggishly a high-performance aircraft reacts at various mach/altitude numbers. Knowledge about the system states or domains affecting system dynamics can be incorporated into the control rule-base. This will allow the controller to change its control strategy according to the system state, similar to a gain-scheduling controller. This type of controller is termed a multi-mode fuzzy logic controller (MMFLC). Multi-mode control rule selection is difficult and tedious for high-dimensional rule bases. The lack of non-model-based design methods and tools for multi-mode rule-base design has limited the application of the multi-mode structure.

The Takagi-Sugeno-Kang control structure is a multi-mode structure with a clearly defined design technique [54, 87]. A Takagi-Sugeno-Kang model is a fuzzy state space model combining linear state space dynamics from different operating domains. A Takagi-Sugeno-Kang state space controller can be designed from the derived models. The design procedure relies on the derivation of accurate mathematical models. However, once accurate mathematical models have been derived, model-based non-linear and robust controller design techniques can be used. The resultant controllers should take preference to fuzzy logic-based controllers if they deliver acceptable performance or have a lower complexity than fuzzy logic-based systems.

Genetic algorithms have been used to optimise complex control problems. The application of genetic algorithms is limited by the availability of fast simulation models, fast computational platforms and adequate optimisation time.

1.2 SUGGESTED SOLUTION

In this dissertation it is postulated that non-model-based design methods can be found for designing MMFLCs for plants with complex, unmodelled non-linearities. The design methods should incorporate qualitative knowledge of the plant, as described in the previous paragraph, without resorting to accurate mathematical plant models. Due to the high dimensionality of the MMFLC rule base, the design process must be automated to ensure a fast design turn-around time. The following advantages of using a multi-mode fuzzy logic control structure are clearly shown.

1. The operating domain of the MMFLC is larger than that of standard non-adaptive controllers and similar to that of self-adaptive controllers.
2. Generally, MMFLCs are mathematically less complex than existing self-adaptive control techniques.
3. MMFLCs do not lose the intuitive simplicity of the fuzzy system associated with some existing techniques.
4. It is possible to automate design procedures for MMFLC rule-base design.
5. The optimised MMFLC can easily be implemented as a look-up table in low-cost control platforms, reducing the cost of mass-produced products.

Two automated methods of multi-mode rule-base design are proposed. These methods form powerful tools in assisting the control system designer in finding and implementing appropriate control rules. In this dissertation the term automated design method does not imply an autonomous or automatic design method. The designer still forms an integral part of the design loop and requires expert knowledge to determine the structure of the proposed controller and in tuning the design algorithm input parameters. The automated design tools assist the designer in finding rules to implement a chosen control strategy.

The first method proposed is a cost-function-based approach, using qualitative plant knowledge. The designer uses expert knowledge regarding the comparative small-signal gain and bandwidth between various operating domains in selecting cost function weights. The cost function weights determine the equivalent controller gain and damping. The closed-loop performance of the controller can be tuned globally using the input gains to the controller, or locally using the cost function weights.

In the second method the rule-base performance is evaluated and improved in a model reference adaptive architecture. The dominant rules responsible for poor closed-loop performance are identified by means of the rule firing strengths. The conclusions of these rules are modified by the selection of an alternative output membership function and then tested. The designer must select appropriate reference models, tuner gains and terminating conditions.

In this dissertation the application of the controllers is for set point tracking and disturbance rejection. The controller set points are thus assumed to form part of the problem definition. In multiple control loop systems these set points become control or manipulated variables in a higher-dimensional control loop with new specifications to be satisfied. Although the application of the design approach is valid for multiple control loops, this investigation falls beyond the scope of this study.

The main contribution made in this dissertation is disproving the traditional criticism of structure complexity and design difficulty against the multi-mode fuzzy logic control structure. It is shown that the multi-mode structures have superior closed-loop response performance over the single-mode control structure. It is further shown that the design process can be facilitated by using the proposed automated design algorithms and plant structure knowledge. The proposed methods do not rely on the determination of accurate mathematical plant models.

1.3 CHOICE OF CASE STUDIES

Testing of the proposed design approach was done on five simulation studies, characteristic of various types of systems. The choice of case study systems was influenced by the availability of fast and accurate simulation models. The plants chosen in this dissertation all show a non-linear dynamic behaviour dependent on system states or external parameters. As far as possible literature examples were chosen to allow for direct comparison of results.

Small-signal, linear transfer functions can be determined from the non-linear model in most of the case studies. These linear models are used to illustrate the change in dynamic behaviour between operating conditions and gain valuable qualitative process information normally obtained from experts. In the design process it was assumed that accurate dynamic models are not available and the small-signal model parameters were not used in the design of the presented controllers.

The first three case studies were used to illustrate the validity of the proposed design approach and to compare the performance of the cost-function based and the self-adaptive designs. The cargo ship steering system is a published benchmark case study for self-adaptive control. The advantage of the system is its low-dimensional complexity. The primary influence on the non-linear behaviour is the ship forward velocity. This allows for the precise design of controllers for specific forward velocities. In addition, this type one system does not require PID control. The results from this problem are ideal for presentation in a research paper currently being written. The two hypothetical second-order plants were developed to illustrate the ability of the MMFLC to control plants with dynamic responses dependent on system states. In the second case study, the effective damping of the second-order plant is modified as an exponential function of position illustrating large variations in plant damping. In the third case study, the velocity feedback is a non-linear function of the velocity. The non-linear velocity feedback is similar to the non-linear drag force encountered by high-performance aircraft approaching the sound barrier.

The fourth and fifth case studies were chosen to illustrate the application of the two proposed design methods on systems from the chemical process industry. The fourth case study is on the temperature control of a homogenous, azeotropic distillation column subjected to disturbances in input feed flow. This process was included primarily as an example of applying the proposed design approach to complex, high-order system. In this example the SMFLC delivers an acceptable transient response but excites unacceptable steady-state oscillations in certain domains, prevented with the use of the MMFLC. The dynamic modelling of this process formed part of a masters project at the University of Stellenbosch currently nearing completion. A detailed model of the distillation column and process knowledge in the form of distillation experts was available to help in the definition of the control objectives and controller structure. The disturbance rejection problem does not conform to the model reference adaptive control architecture required by the self-adaptive design approach and is thus used to illustrate the cost-function-based design method. The fifth case study is on the control of the reactant concentration of an exothermic CSTR system. This system is a well-known literature example and the plant dynamics have been investigated in detail. The concentration control problem is used to illustrate the application of the self-adaptive design technique. As in the distillation column, this reaction was included to illustrate the ability of the MMFLC in preventing steady-state oscillation or limit cycles excited by the SMFLC.

1.4 DISSERTATION LAYOUT

A literature review, as motivation for the proposed design approach, is presented in Chapter 2. Paragraph 2.1 presents a summary of the basic fuzzy system properties with an explanatory example. The standard FLC structures and the fuzzy sliding mode controller (FSMC) are presented in paragraph 2.2. An FLC design procedure is presented in paragraph 2.3. The traditional rule-base design methods are reviewed in paragraph 2.4. Self-adaptive fuzzy logic control (SAFLC) is presented in paragraph 2.5. The problem definition and suggested solution are detailed in paragraph 2.6.

The proposed design and adaptive algorithms are presented in Chapter 3. The general assumptions for the application of the algorithm are defined in paragraph 3.1. A general approach and guidelines to MMFLC design using a cost function method of rule-base generation are detailed in paragraph 3.2. The model reference adaptive architecture and rule-base adaption algorithm is presented in paragraph 3.3.

The performance of the proposed MMFLC design method is evaluated and compared to the single-mode fuzzy logic controller (SMFLC) in Chapter 4. Controllers are designed for a cargo ship (paragraph 4.1), a second-order plant with exponential non-linearity (paragraph 4.2) and a second-order plant with non-linear velocity feedback (paragraph 4.3). The controller rule-bases are optimised by gain and cost function weight manipulation and using the proposed adaptive architecture. Paragraph 3.4 ends the chapter with the conclusions drawn from the three case studies.

Chapter 5 explains the application of the proposed design method for temperature regulation of an azeotropic distillation column. A short introduction to distillation and the simulation model is given in paragraph 5.1 and paragraph 5.2. A state space observer for estimation of the error rate of change is presented in paragraph 5.3.1. The system architecture does not allow for the use of the model reference adaptive algorithms and the SMFLC (paragraph 5.3.2) and the MMFLC (paragraph 5.3.3) is optimised by adapting the controller input gains and cost function weights. The performance of the SMFLC is compared to the MMFLC in paragraph (5.4).

Chapter 6 explains the application of the rule-base adaptive algorithms in designing a concentration controller for a CSTR system. The plant and simulation model is presented in paragraph 6.1. The design of the SMFLC is presented in paragraph 6.2.1 and the MMFLC in 6.2.2. The performance of the controllers is compared in paragraph 6.3 .

The conclusions drawn from the work presented in this dissertation and suggested future research are presented in Chapter 7.

2 FUZZY SYSTEMS

Fuzzy logic systems are based on fuzzy set theory, as suggested by Lotfi Zadeh in 1965, and are a generalization of the first theory on sets by Georg Cantor (1845-1918) [21, 80, 89, 95]. In an attempt to model and simulate human linguistic reasoning, Zadeh suggested that humans think in terms of fuzzy sets [61, 95]. Zadeh defines a fuzzy algorithm as an ordered set of fuzzy instructions to yield an approximate solution to a specific problem [95]. A fuzzy logic algorithm makes decisions based on imprecise, non-numerical data and can thus be thought of as a continuously valued state machine [10, 60, 61]. Numerous discussions on fuzzy systems exist and only a short summary of the basic principles and properties as applicable to this work will be given in paragraph 2.1 [3, 21, 34, 36, 37, 42, 54, 80, 87, 89, 95].

2.1 FUZZY LOGIC SYSTEMS

The inputs and outputs of a fuzzy logic system are called linguistic variables [10, 54, 95]. A linguistic variable can be defined as a variable that takes a word or a sentence, called a fuzzy variable (as defined by a fuzzy set and taken from a term set), as its value [21, 36, 42, 54, 87, 89].

Elements of a fuzzy set are taken from a universe of discourse of all the considered elements [21]. A fuzzy set gives a qualitative measure of the membership of its elements. The gradual transition in membership between zero and one is defined by a membership function [36, 45, 80, 89, 95]. Elements of a fuzzy set that have non-zero membership values are called the support of a fuzzy set [21, 36, 89, 95]. A fuzzy set with a single point as support is called a fuzzy singleton [36, 89, 95]. A fuzzy set can be thought of as a collection of ordered pairs or a union of constituent singletons [21, 95]. A fuzzy set A, with elements x from the universe of discourse X in which A is defined and characterized by a membership function $\mu(x)$, can be represented by Equation 1.

$$A = \{(x, \mu(x))\}$$

Equation 1

Consider two fuzzy sets A and B, defined by membership functions $\mu_A(x)$ and $\mu_B(x)$, on a single universe of discourse. The standard fuzzy set operations of complement, intersection and union are defined as given by Equation 2 a, Equation 2 b and Equation 2 c [21, 42, 45, 54, 87].

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$
$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \otimes \mu_B(x) = \min[\mu_A(x), \mu_B(x)]$$
$$\mu_{A \cup B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \oplus \mu_B(x) = \max[\mu_A(x), \mu_B(x)]$$

Equation 2

The intersection operator corresponds to a fuzzy logic AND operation and can be implemented by taking the minimum (Equation 2 b) or the algebraic product, while the union operator corresponds to the fuzzy logic OR operation and can be implemented by taking the maximum (Equation 2 c) or the bounded algebraic sum of the two membership functions [21, 42, 45, 54, 87, 94]. The defined fuzzy set operations of compliment, intersection and union have standard distributive and associative properties, allowing for the operation on more than two fuzzy sets [45]. In the literature the fuzzy intersection operator is also referred to as the T-Norm while the union operator is referred to as the T-Conorm or S-Norm operator, with various suggested implementations [20, 21, 37, 42, 45, 54, 87, 89].

The fuzzy Cartesian product, or fuzzy relation, is used to implement fuzzy operators on fuzzy sets from different universes of discourse [21, 54, 87]. A fuzzy relation R is a fuzzy set in product space $U \times V$ with membership function $\mu_R(u,v)$ with $u \in U$ and $v \in V$ [21, 54, 87, 89, 95]. The fuzzy composition of fuzzy relation R in $U \times V$ and S in $V \times W$ that share a common set (V) is a fuzzy relation defined by Equation 3.

$$\mu_{R \circ S}(u,v) = \sup_{v \in V}(\mu_R(u,v) \oplus \mu_S(v,w))$$

Equation 3

The Sup-Min and Sup-Product composition methods are the popular forms of implementing the relational operator [21, 37, 87, 89].

Fuzzy logic system are based on fuzzy rules or fuzzy implications and various methods of implementation have been proposed [21, 53]. Fuzzy implication functions are used to quantify fuzzy rules [54]. The three types of fuzzy implication functions of importance to this work are: fuzzy conjunction (logical AND), disjunction (logical OR) and implications (logical THEN) and are normally implemented using T-Norms and T-Conorms [37, 42, 87]. Based on the specific method of T-Norm and T-Conorm implementation, different implication methods have been suggested by various authors [21, 87]. Consider the fuzzy rule:

IF (*Fuzzy condition*) **THEN** (*Fuzzy implication*).

The condition is defined as a fuzzy relation in *U* and the fuzzy implication defined as a fuzzy relation in *V*, with *x* and *y* linguistic variables in *U* and *V*. According to the implication method proposed by Zadeh, the fuzzy rule is interpreted as a fuzzy relation with membership function as given by Equation 4 a. The subscripts *Cond* and *Ipl* refers to the condition and implication membership functions respectively. Mamdani's implication is a fuzzy relation with membership function as given by Equation 4 b. The simplicity of the Mamdani-type implication accounts for its popularity amongst control system designers. Mamdani inference engines were thus chosen for all the systems studied in this dissertation.

$$\mu_{Q_Z}(x,y) = \max\left[\min\left[\mu_{Cond}(x),\mu_{Ipl}(y)\right],1-\mu_{Cond}(x)\right]$$
$$\mu_{Q_{MM}}(x,y) = \min\left[\mu_{Cond}(x),\mu_{Ipl}(y)\right]$$

Equation 4

In order to draw a single conclusion from a set of fuzzy rules, an inference operator is required [21, 87]. In individual rule-base inference, each rule in the rule-base qualifies an output fuzzy set. The consequences of all the rules are combined with the union or intersection operator to form a single output fuzzy set [54, 87]. Of the various proposed inference methods, the minimum inference engine is used in this work [42, 87]. In the minimum inference method individual rule-base inference with union combination is used in conjunction with Mamdani's implication method as given by Equation 5 for the case of *M* rules in the rule-base [87].

$$\mu_{B'}(y) = \max_{l=1}^{M}\left[\min\left[\mu_{Cond}(x),\mu_{Ipl}(y)\right]\right]$$

Equation 5

The structure of a typical fuzzy logic system and the application of the above equations is best explained using a simple example. For the sake of simplicity consider a two-input, one-output fuzzy logic system that maps the inputs *x* and *y*, in their universes of discourse *X* and *Y*, to output *z* in universe of discourse *Z*. The structure and elements of the fuzzy logic system are shown in Figure 1.

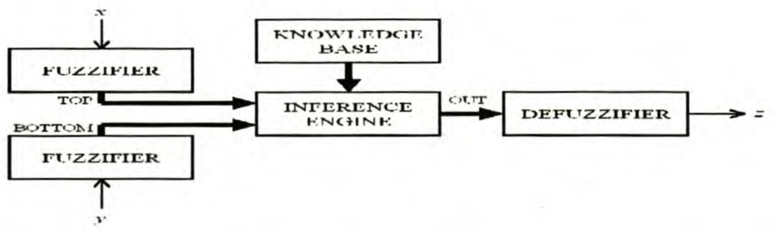

Figure 1 Structure of a two-input, one-output (MISO) fuzzy logic system.

Assume that the linguistic variable *TOP* is associated with *x*, *BOTTOM* with *y* and *OUT* with *z*. In this example, as in all the fuzzy systems in this work, the minimum operator is used to implement the T-Norm, the maximum operator implements the T-Conorm, while Mamdani's implication and inference methods are used. Assume that each linguistic variable has three fuzzy sets in each term set, called *Low*, *Medium* and *High*, described by triangular membership functions as indicated in Figure 2.

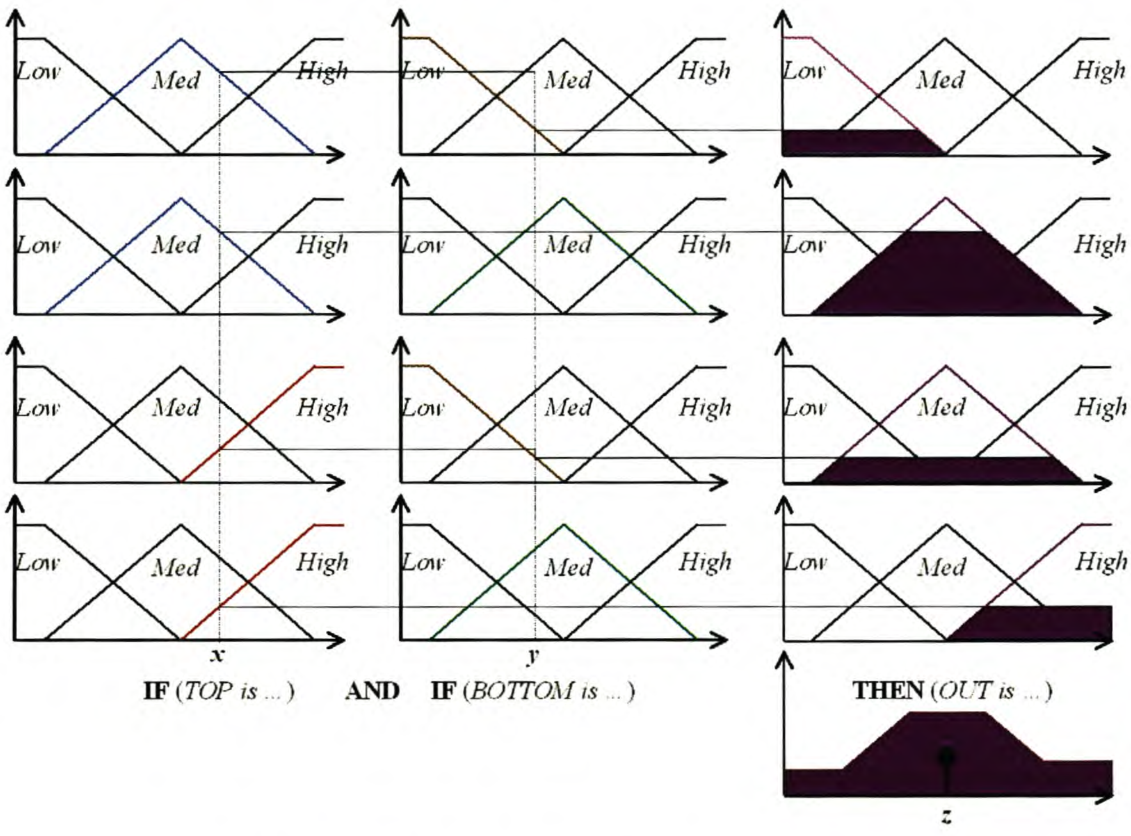

Figure 2 Mamdani inference for a two-input, one-output fuzzy logic system.

The four rules indicated in Figure 2 are:

Rule 1: IF (*TOP is Med*) AND (*BOTTOM is Low*) THEN (*OUT is Low*)

Rule 2: IF (*TOP is Med*) AND (*BOTTOM is Med*) THEN (*OUT is Med*)

Rule 3: IF (*TOP is High*) AND (*BOTTOM is Low*) THEN (*OUT is Med*)

Rule 4: IF (*TOP is High*) AND (*BOTTOM is Med*) THEN (*OUT is High*)

The crisp inputs (*x* and *y*) to the fuzzy logic system are processed by the fuzzifier (Figure 1) [10]. The fuzzifier calculates the grade of membership for each fuzzy set using the associated membership function (Figure 2) [20].

The knowledge base (Figure 1) consists of the fuzzy **IF** – **THEN** rules and is referred to as the fuzzy associated memory (FAM) [9, 10]. Each fuzzy **IF** – **THEN** consists of an antecedent and consequence. The antecedent is the input condition in the application domain and forms an input space, defined by the combined fuzzy relation, in the input universe of discourse [36]. The consequence is an output action in the output universe of discourse, as determined by the fuzzy inference engine [36].

The inference engine uses fuzzy logic principles and inference operators to draw a conclusion from the rule-base [20, 21, 36]. The inference engine thus combines the fuzzy implications from the rule-base into a mapping from the input fuzzy sets to an output fuzzy set [36, 89]. The first step is to use the T-Norm and T-Conorm to resolve the support or firing strength of each rule according to the rule antecedent. In the example of Figure 2, this is done by taking the minimum of the membership values as indicated [21]. The next step is to use the individual rule support to shape the fuzzy output membership function [21]. In Figure 2 the output fuzzy sets are limited to the rule support value as proposed by Mamdani. The last step is to aggregate all the output fuzzy sets to determine a single fuzzy set as output, specifying possible output actions [21, 36]. According to Mamdani's method, this is done by taking the logical OR of the individual output fuzzy sets as shown in Figure 2.

A defuzzifier (Figure 1) is used to determine a crisp output from the output fuzzy set [10]. One of the most popular methods is to calculate the centroid of the resulting output fuzzy set [20]. The centroid method is sensitive to the consequence of all the rules activated, while other methods are biased towards the most dominant rules (rules with higher truth values or firing strengths) [10]. Various other defuzzification methods have been proposed in the literature [10, 21, 42, 54, 93]. In this work the centroid method is used in all the examples.

2.2 FUZZY LOGIC CONTROLLER STRUCTURES

The first application of fuzzy logic theory in the control of dynamic systems was reported by Mamdani in 1974 with the control of a model steam engine [46]. In 1976 Kickert and van Nauta Lemke reported the application of fuzzy logic control to a process control problem, with the control of a laboratory-scale warm water plant [26]. The first significant industrial application of fuzzy logic was the control of a cement kiln in Denmark [60, 87]. Since then, fuzzy logic controllers have found application in industrial processes and consumer products as wide-ranging as nuclear reactors, cement kilns, railway train control, automotive engine and transmission control, machine centres, auto-focus cameras, high-performance aircraft and household washing machines [7, 17, 21, 36, 60, 61, 65, 83, 87]. Fuzzy logic controllers have successfully been used when [10]:

1. Mathematical models of the plant are complex or lacking;
2. Inexpensive or noisy sensors are used in state measurements;
3. Low-precision micro-controllers are used as control platforms; and
4. Expert knowledge is available to specify the control rule-base.

Various structures and implementations of the non-adaptive FLC have been proposed in the literature. A short survey of the more popular control structures will now be given, while the methods of designing these controllers will be explained in paragraphs 2.3 and 2.4. The structure of the adaptive FLC is reported in paragraph 2.5.

The structure of a simple form of the FLC is shown in Figure 3.

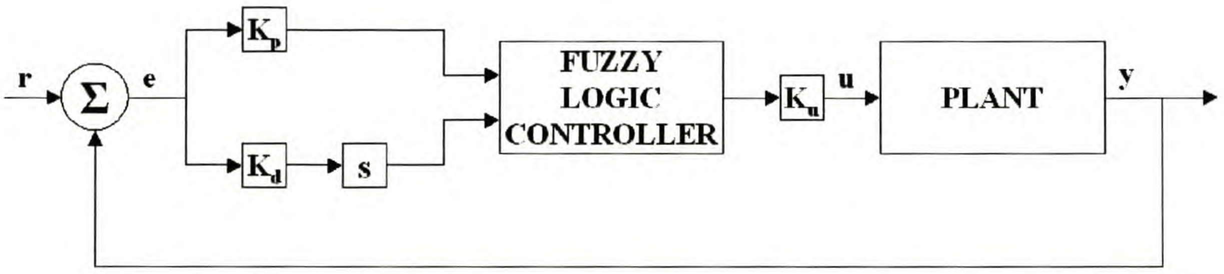

Figure 3 Structure of a PD-type FLC.

The plant output (y) is compared to the set point (r) to generate a tracking error (e) [17]. The tracking error, henceforth just called the error (e), and error rate of change (de/dt) are amplified through the controller input gains and fed to a two-input, one-output fuzzy systems as indicated in Figure 3. The fuzzy system has rules of the form:

IF (*ERROR is E_k*) AND (*ERROR RATE OF CHANGE is R_l*) THEN (*OUTPUT is U_{kl}*).

The output from the fuzzy system is amplified with the controller output gain and fed to the plant input. Since the fuzzy system maps the error and error rate of change to the output, it forms a non-linear proportional and derivative (PD) controller. Since the output of the fuzzy system directly specifies the control value to be applied to the plant, without any dynamic filtering, it is termed an absolute-type controller [21]. The addition of the derivative information helps to limit overshoot at the expense of noise, due to the numerical derivative process [21]. Due to its simplicity this form of FLC is quick to implement and intuitively understandable. This form of control does not allow for perfect set point tracking if the plant is not a type 0 system.

The incremental PD-type FLC overcomes the perfect set point tracking limitations of the PD-type FLC by adding a pure integrator to the plant as shown in Figure 4 [26, 30, 94].

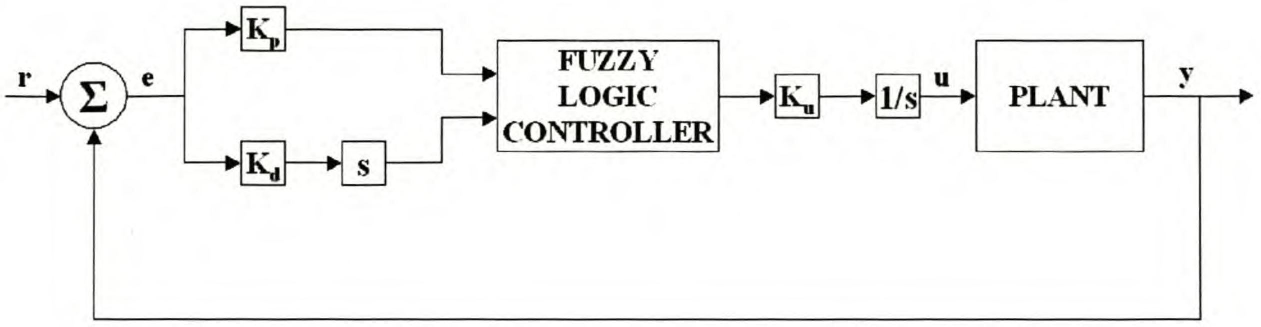

Figure 4 Structure of an incremental PD-type FLC.

For this type of controller the form of the rule-base is identical to the standard PD-type FLC. A different approach is to incorporate the integral action into the rule-base by changing the rule form to

IF (*ERROR is* E_k) **AND** (*ERROR RATE OF CHANGE is* R_l) **THEN** (*CHANGE IN OUTPUT is* U_{kl}).

The fuzzy system suggests a change or increment to the current control action and is termed an incremental fuzzy system [21]. This control structure has proven to be a popular choice with designers following the successful application of this structure by Mamdani, King, Østergaard, Assilian and others [70, 78, 80]. A combination of the absolute and incremental PD-type controller has been proposed by Tong in which absolute control is used for large errors and incremental control for small errors [80]. The addition of the integral action effectively changes the controller from a PD type to a Proportion and Integral (PI) type of controller [77, 88]. This type of controller is simple to implement and easy to understand intuitively. Experiences with this type of controller have shown this author that the bandwidth reduction caused by the additional integrator severely limits the speed of closed-loop response. Higher loop gains used to increase closed-loop speed of response often caused large overshoot and unacceptable oscillations close to the set point.

The linear augmented PD controller is shown in Figure 5.

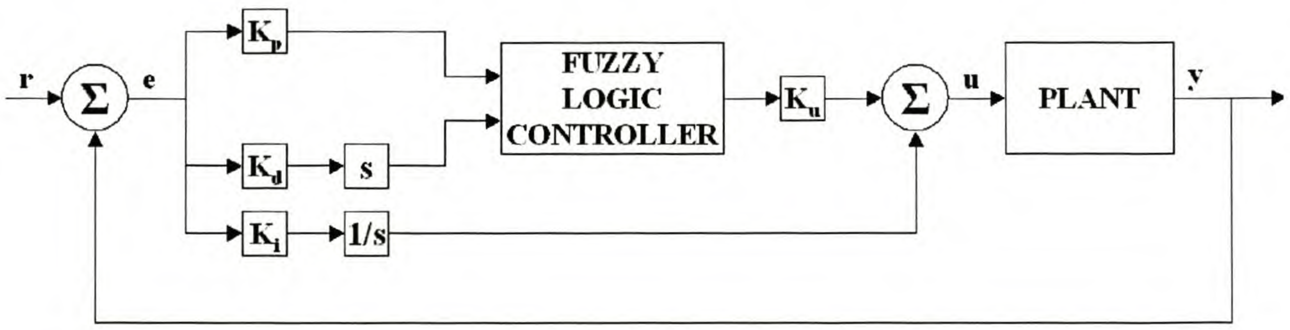

Figure 5 Structure of a linear augmented PD-type FLC.

The output from a fuzzy system, identical to that of the PD-type FLC, is added to a scaled integral of the error signal. This effectively forms a non-linear proportional, integral and derivative (PID) type controller. This form of controller does not suffer from the bandwidth reduction problems of the incremental type PD controller and is still able to attain perfect set point tracking for type one plants.

It is possible to incorporate the integral contribution of the linear augmented type PD controller directly into a PID-type controller as shown in Figure 6.

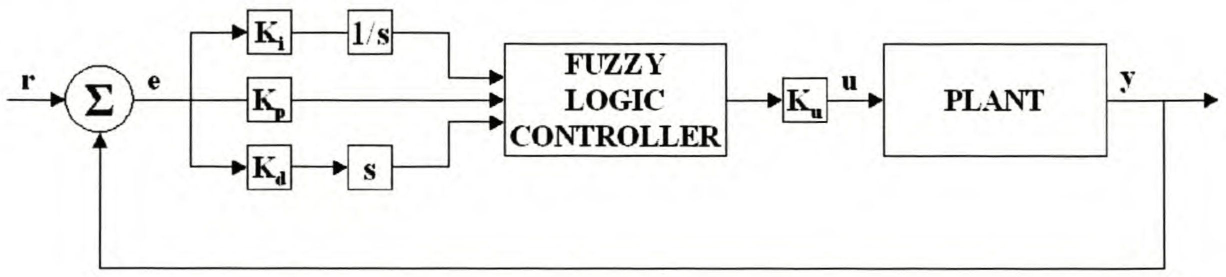

Figure 6 Structure of a PID-type FLC.

This form of controller maps the error integral ($\int e.dt$), error and error rate of change to the controller output using rules of the form

IF (*ERROR INTEGRAL is* I_j**) AND (***ERROR is* E_k**) AND (***ERROR RATE OF CHANGE is* R_l**)**
THEN (*OUTPUT is* U_{jkl}**).**

An alternative form of the PID-type controller uses the error, error rate of change and error acceleration as inputs and an incremental output structure [1, 75]. The format of the control rules would then be

IF (*ERROR is* I_j**) AND (***ERROR RATE OF CHANGE is* E_k**) AND (***ERROR ACCELERATION is* R_l**)**
THEN (*CHANGE IN OUTPUT is* U_{jkl}**).**

The disadvantages of the PID-type of controller are the added complexity of the additional input expanding the rule-base dimensions and the difficulty of effectively adding the error integral information to the rule-base [21]. Control system engineers normally have a better intuitive understanding of PD type rule composition than PID-type composition.

Although the FLC has been proven to be extremely robust in the control of non-linear plants, the control structures explained above use the same rule-base throughout the non-linear state space.

Large variations in the dynamic model of the plant due to changing operating conditions will cause a finely tuned closed-loop behaviour to lose relevance as the plant dynamics change [7, 48]. A rule-base optimised in one domain may lead to unacceptable or even unstable performance in other domains. One method of solving this problem would be to incorporate non-linear domain knowledge into the rule-base and design an optimised controller for each domain. The structure of such a multi-mode controller is illustrated in Figure 7.

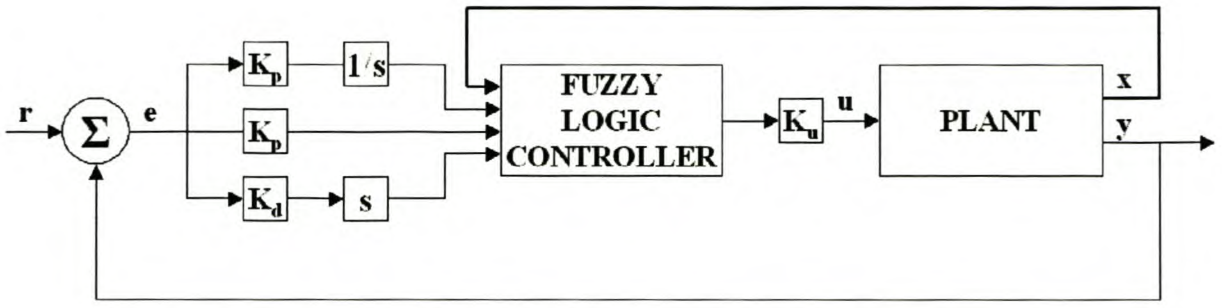

Figure 7 Structure of a multi-mode PID-type FLC.

The control rules would then be of the form

IF (*STATE is X_i***) AND (***ERROR INTEGRAL is I_j***) AND (***ERROR is E_k***) AND (***ERROR RATE OF CHANGE is R_l***) THEN (***CONTROL is U_{ijkl}***).**

The linguistic variable *STATE* is indicative of the non-linear domains or operating conditions of the plant. The big disadvantage of such a controller is the added complexity caused by the higher dimensionality of the rule-base and the difficulty in finding suitable control rules [48].

A variable structure or sliding mode FLC is an FLC designed on sliding mode principles [87]. It has been shown that sliding mode control (SMC) gives consistent performance for non-linear and uncertain processes [79, 87]. The properties and design methods of SMC have been thoroughly documented and what follows is just a short explanation of SMC without a boundary condition [52, 68, 82]. Consider an nth order, non-linear, dynamic SISO system as described by Equation 6.

$$\dot{x} = f(x) + u$$
$$x = \left[x, \dot{x}, \ddot{x}, \dddot{x}, \ldots, x^{n-1} \right]^T$$
$$f(x) \leq \hat{f}(x) + F(x)$$

Equation 6

Assume that the non-linear function *f(x)* is unknown, but can be approximated with a known maximum error bound *F(x)*. The object is to determine a control signal *u*, so that the system states x track a desired system state x_d. Define the state tracking vector *e* as given by Equation 7.

$$e = x - x_d$$
$$= \left[e, \dot{e}, \ddot{e}, \dddot{e}, \ldots, e^{n-1} \right]^T$$

Equation 7

Further, define a scalar function S as a linear combination of the elements of the state tracking vector, as defined by Equation 8, where λ is a positive constant.

$$\begin{aligned} S(x,t) &= \left(\frac{d}{dt}+\lambda\right)e \\ &= e^{n-1} + C_{n-1}^{1}\lambda e^{n-2} + C_{n-2}^{2}\lambda^{2}e^{n-2} + C_{n-3}^{3}\lambda^{3}e^{n-3} + \ldots + \lambda^{n-1}e \end{aligned}$$

Equation 8

It is clear that our tracking problem is equivalent to keeping the system states on the time-varying surface in the non-linear state space, as defined by *S(x,t) = 0*. The surface defined by *S(x,t) = 0* is referred to as the sliding surface. When the system states are on the sliding surface the stability of the closed-loop system can be investigated by considering the candidate Lyapunov stability function of Equation 9 a.

$$V = \tfrac{1}{2} S^{T} S \le \eta |S|$$

$$\dot{V} = S^{T} \dot{S} \le 0$$

Equation 9

Closed-loop stability is assured if the first derivative of the Lyapunov stability function, Equation 9 b, is smaller than 0. This implies that, if the states move away from the sliding surface, the control action will force the state trajectory back towards the sliding surface. For example, a second-order system having a linear sliding surface is defined by Equation 10.

$$S(x,t) = \dot{e} + \lambda e = \dot{x} + \lambda x - \dot{x}_d - \lambda x_d$$

Equation 10

The sliding conditions of Equation 9 a are then given by Equation 11.

$$S\left[f(x) + u + \lambda \dot{e} - \ddot{x}_d \right] \le \eta |S|$$

Equation 11

Now choose the control signal as defined in Equation 12, where *sgn()* refers to the saturation function.

$$u = -\hat{f}(x) + \ddot{x}_d - \lambda \dot{e} - K\left(x, \dot{x} \right) \operatorname{sgn}(s)$$

Equation 12

The sliding condition is now defined by Equation 13.

$$\operatorname{sgn}(S)\left[f(x)-\hat{f}(x)+K\left(x,\dot{x}\right)\operatorname{sgn}(s)\right]\leq -\eta$$

Equation 13

The sliding condition is guaranteed if we choose the saturation function gain K, according to Equation 14.

$$K\left(x,\dot{x}\right)=\eta+F(x)$$

Equation 14

Due to the discontinuous or switching characteristics of the saturation function in the control signal (Equation 12), the closed-loop system exhibits high-frequency chatter, caused by the large changes in control signal value across the switching surface. This chatter can be eliminated by the inclusion of a boundary layer as explained in the references [68, 82, 87].

The fuzzy sliding mode controller allows for various forms of implementation. The base of the FSMC is realising that the FLC is an extension of a SMC with boundary layer [43, 87]. The FSMC uses a composite state defined by the sliding surface for control action, thus requiring fewer rules to implement [14].

Choosing the FSMC for a second-order system as given by Equation 15 guarantees the sliding condition and thus closed-loop performance [87].

$$u_{fuzzy}(s,x)\leq -\eta-\left[f(x)+\lambda\dot{e}-\ddot{x}_{d}\right],\ \mathit{if}\ \operatorname{sgn}(s)>0$$
$$u_{fuzzy}(s,x)\geq \eta-\left[f(x)+\lambda\dot{e}-\ddot{x}_{d}\right],\ \mathit{if}\ \operatorname{sgn}(s)<0$$

Equation 15

As indicated in the references, an FLS can be used as a smooth approximation of a sliding control rule with a boundary layer eliminating the undesirable chatter of a standard SMC [25, 43, 63, 68, 82, 87]. The advantage of the FSMC is that clear analytical design methods exist [87]. All the proposed design methods rely on the availability of an approximation of the system and therefore the FSMC and the adaptive FSMC will not be investigated further in this work [63, 79].

2.3 FUZZY LOGIC CONTROLLER DESIGN METHODS

An FLC is difficult to design since no theoretical base exists to aid in the design of controllers for high-order systems which ensure their stability [14, 44]. Some analysis and design techniques assume plant knowledge and contradict the fuzzy logic control system objectives [88]. The author shares the opinion that linear, or non-linear, model-based design techniques should be used if plant models are available [88]. From paragraphs 2 and 2.2 it is clear that the design of an FLC entails the determination of: the input and output variables, membership functions, fuzzification strategy, the rule-base, inference engine logic and the defuzzification strategy [1, 36, 74]. In this study, it is assumed that no mathematical model of the plant is available and a model-based design approach cannot be used. In this paragraph an FLC design method, based on the general method advocated by Jantzen, is proposed and used in the subsequent work [21]. Some of the existing methods of rule-base design are evaluated in paragraph 2.4.

The first step is to define the structure of the required FLC. Since it is assumed that no mathematical plant model is available, the SFMC and variants cannot be used. This author suggests the use of a multi-mode PID-type controller and the automated rule-base design algorithms proposed in Chapter 3. The FLC inputs and output are as defined in paragraph 2.2.

The second step is to define the membership functions for the input and output fuzzy sets. The shape of the membership function is subjective and depends largely on user preference [40]. Experience has shown that the closed-loop performance is not very sensitive to the shape of the membership functions [21]. Due to its simplicity of implementation and speed of calculation, symmetrical triangular membership functions are recommended [7]. Some authors preferred asymmetrical triangular membership functions, crowded closer to origin, to give a rapid response far from the set point and smooth, precision control close to steady state set point [48, 53, 70, 75]. The membership functions should be sufficiently wide to ensure that the whole universe of discourse is covered and ensure a margin of noise immunity [36]. A smooth control action is established by allowing for enough overlap in membership functions so that at least two rules fire for any input [16]. Jantzen recommends starting with three fuzzy sets per linguistic variable on a normalised universe of discourse and then to increase the amount of sets as required.

The next step is to choose a fuzzy inference engine and design the rule-base. The simplicity and intuitive understandability of the Mamdani-type fuzzy system account for its popularity amongst fuzzy system designers. The rule-base characterises the control rules and is fundamental in determining the closed-loop performance [36]. A complete rule-base has at least one active rule for each possible input combination and will thus produce a valid output for any input in the input universe of discourse [36]. A consistent rule-base implies that the rules do not contradict each other by mapping to multiple peaks in the output universe of discourse [1]. The standard methods of obtaining a good rule-base is expert knowledge, modelling of operator control actions, a fuzzy inverse model, genetic algorithms and self-learning [1]. These methods are detailed in paragraphs 2.4 and 2.5.

The last step in the design process is choosing a defuzzifier. The centre of area method is a popular choice for use with Mamdani-type systems and is used in all case studies reported in this dissertation [40].

The final controller is the result of an iterative design or tuning process to obtain a desired closed-loop response. The controller input and output gains, membership functions, rules, inference method or defuzzification method can be modified [1, 9, 30, 64, 80]. Various methods have been proposed to tune the closed-loop response by adjusting the high degree of free parameters [21, 54]. Membership function tuning is ineffective due to the low sensitivity of the control policy to changes in membership function shapes and parameters [64]. Closed-loop response tuning by controller gain adjustment, analogous to classical PID tuning, are effective in obtaining a better closed-loop response but not robust enough to deal with incorrect rules [30, 64, 98]. Since the character of a fuzzy system is determined primarily by its rule-base, a modification of the rules is the most effective way of controller tuning [64, 79, 80, 86, 92].

A lookup table implementation of the fuzzy logic control system by off-line generation of the control rules for discrete inputs provides a controller with minimum overheads and calculation time [30, 36, 64]. Abdelnour *et al.* have shown how a PID-type controller can be implemented as a two-dimensional lookup table by using an offset into a PD-type lookup table [1]

2.4 RULE-BASE DESIGN METHODS

The choice of control rules is crucial in the determination of the closed-loop system response and therefore the most difficult aspect of FLC design [3, 53]. A short review of FLC rule-base design by using expert knowledge, modelling of operator control action and genetic algorithms is given. FLC rule-base design by means of self-learning can be classified as a self-adaptive fuzzy logic controller (SAFLC) or self-learning fuzzy logic controller (SLFLC) and is reviewed in paragraph 2.5.

Designing an FLC rule-base by means of expert knowledge is based on the intuition and experience of experts for the determination of heuristic rules of thumb for control [17, 26, 40, 70, 74, 85]. The source of expert knowledge can be a control engineer's intuition, the expert knowledge of a person familiar with the plant dynamics or the experience of a skilled plant operator [42]. Expert knowledge normally indicates that large errors require the largest possible control for fast response, while small errors require almost linear control. This results in a time-optimal control approach for large errors combined with a robust control approach for small errors [16]. The MacVicar-Whelan meta rules of incremental PD-type rule-base design recommend that the present control setting should be maintained for zero errors or errors diminishing at an acceptable rate [77]. If the error is not diminishing at an acceptable rate, the existing control setting should be modified in proportion to the error and error rate of change [77]. The system response far from the set point can be improved by imposing damping close to the set point only, which will prevent an overdamped response [7]. In the case of the control of industrial processes, such as cement kilns, operator handbooks provide a vital source of expert knowledge. Heuristic design of an FLC rule-base is a difficult and error-prone optimisation technique that may result in a stable but not always optimal controller [7, 31]. Since the manual coding of expert knowledge into a rule-base is a tedious process, the application of this method has been limited to small rule-base system [21, 65, 69]. An added disadvantage is that operator actions are often erratic, inconsistent and not based on a single measurement [30, 44].

Fuzzy logic modelling is a qualitative modelling technique that describes the process to be modelled in linguistic terms [74]. It has been shown that an FLS can be used as a universal approximator [13, 96]. A fuzzy model provides a smoothly connected approximation, in contrast to a piece-wise linear approximation [76]. An FLS can thus be tuned to model a skilled plant operator by observing appropriate input/output data associated with certain control actions [42, 73, 76]. Various methods have been developed for dynamic system modelling by means of an FLS [2, 32, 33, 42, 72, 76, 87, 92, 96]. These techniques are beyond the scope of the work included in this dissertation.

An FLC may be regarded as an inverse model of the plant to be controlled [21]. A fuzzy model of the plant to be controlled may be inverted to supply control rules [21]. A number of heuristic design methods based on linear control, optimal control methods and feedback linearization techniques have been proposed [19, 23, 42, 53]. The Takagi-Kang-Sugeno method of modelling provides a fuzzy model based on localised, linear state space models, suitable for the design of a fuzzy state space controller [44, 87]. The method requires an accurate dynamic model and will not be considered further in this work.

A genetic algorithm (GA) can be used for the offline optimisation of an FLC by artificial evolution of the fittest controller. Gradient-based search methods cannot be relied upon to find a global optimum in a multi-parameter search space with discontinuity, multi-modality and non-linear constraints [27]. The genetic algorithm is an optimisation or search technique based on the mechanics of biological selection, genetics and evolution [42, 54]. The algorithm attempts to simulate Darwin's theory of natural selection and Mendel's theory on genetic inheritance to seek an optimum for a fitness function $J(\theta)$, where θ is the parameter vector to be optimised [54]. The GA performs a stochastic but directed search to evolve the fittest members in a population of chromosomes by searching many local optimum peaks in parallel [42, 54]. The exchange in information between peaks prevents trapping at a local optimum by forcing the search to jump to a new region of the search space that could contain a global optimum [42, 54].

A population is formed by taking a collection of chromosomes formed by the stringing together of genes [54]. A gene is thus a location in a chromosome that can take on various values from a number system, determined by an encoding mechanism that codes the parameters to be tuned into chromosomes [42, 54]. It has been shown that binary coding is the optimum coding scheme [42].

The genetic operators are used to produce a new generation from a previous generation [54]. Normally the number of chromosomes is kept constant from generation to generation [42]. According to Darwin's theory on natural selection, the first operator selects parent members for the next generation based on a fitness value determined for each individual chromosome, according to how close to an optimum it is. Members with a high fitness value have a higher probability of survival to "mate" and form the next generation compared to unfit members [54]. The selection process is done with a roulette wheel parent selection algorithm [42]. Selection directs the search towards the best existing individuals but does not create new individuals [42]. The crossover operation is a swap in genetic material, identical to the process in which a child inherits genes from both parents [42, 54]. Parents are paired off at random and a random crossover site is selected at which the genes are separated and crossed to produce children [42]. After the crossover operation, the mutation process inverts or change randomly selected genes in the population [42, 54]. The new generation is now tested against the fitness value. Optimisation stops when some predetermined fitness value or a set number of generations are reached.

In FLC design a population of chromosomes is built up where each chromosome represents a controller with the parameters to be optimised coded into genes [24, 27]. Using a pre-selected fitness function and computer simulations, a fitness value is determined for each controller. The fitness value quantifies the success of the closed-loop response in terms of the key performance indicators (KPIs) such as rise time, overshoot, settling time, etc. [24, 54]. The genetic operators are then applied to form a new generation that can be tested. The genetic algorithm provides a powerful tool for FLC development and has been successfully used to optimise controllers for systems as wide-ranging as pH neutralisation, space-based oxygen production systems and inverted pendulum stabilisation [24, 27, 28]. The algorithm can be used to optimise the membership function parameters or the rule-base, or both simultaneously [18, 24, 27, 28]. The disadvantage of the GA is that it requires a fast and accurate simulation model and is computationally intensive. The author does not consider the GA suitable for complex controller design, due to its numerical intensity.

Various design methods based on cell mapping and gradient parameter modification algorithms have been proposed and successfully applied to various systems [69, 83]. These techniques are computationally intensive and require accurate mathematical models. These methods are not considered further in this study.

2.5 SELF-ADAPTIVE FUZZY CONTROL

An adaptive control system can be defined as a non-linear controller that tracks slow changes in the system parameters and updates the controller parameters in an attempt to extend the working range of the control system [4]. Adaptive fuzzy systems can modify the system input scaling factors, the characteristics of the rules, the topology of the fuzzy sets and the method of fuzzification and defuzzification [9, 48, 62, 98]

The first adaptive FLC was proposed by Procyk and Mamdani in 1979 and was termed a self-organising controller (SOC) [21, 55]. The SOC uses a combination of system identification and control experience to automatically develop and improve rules by monitoring the process performance [53, 62]. The SOC uses a control decision table in conjunction with an output correction signal to modify the previously active rules responsible for the present poor performance [55, 75, 78]. The output correction is determined by a performance index table, similar to the control decision table, and a small signal, linear dynamic model of the plant [9, 55, 62]. The performance index table contains the minimum tolerable response [78]. The SOC can be adapted to modify the control surface directly, preventing the recalculation of the lookup table following a rule adaption [62, 64].

Since the advent of the theory of adaptive control and the SOC, the proposed SLFLC architectures have been divided into two classes according to the structure of the parameter adaption mechanism [4, 54, 87].

2.5.1 INDIRECT ADAPTIVE CONTROL

The structure of an indirect adaptive controller is shown in Figure 8.

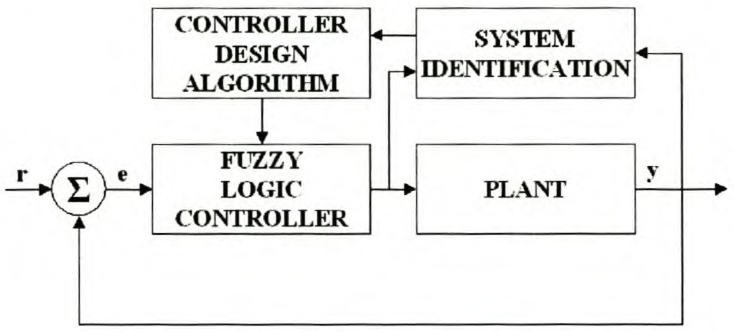

Figure 8 Indirect adaptive fuzzy logic control architecture.

In this method, the plant parameters are updated using an online system identification algorithm and the controller parameters are determined by solving a design algorithm [4, 54, 87]. This structure is also referred to as the self-tuning regulator [4].

One approach is based on the online tuning of fuzzy models approximating the non-linear plant dynamics and cancelling the plant non-linearity in a state feedback scheme [87]. Consider an n^{th} order non-linear system defined by Equation 16.

$$x^{(n)} = f\left(x,\dot{x},\ddot{x},\dddot{x},\ldots,x^{(n-1)}\right) + g\left(x,\dot{x},\ddot{x},\dddot{x},\ldots,x^{(n-1)}\right)u$$

Equation 16

The state feedback linearisation control law is given by Equation 17.

$$u = \frac{1}{\hat{g}(x,\theta_g)}\left[-\hat{f}(x,\theta_f) + y_m + k^T e\right]$$

Equation 17

The online identification algorithm supplies the approximation function parameter vectors θ_f and θ_g. The reference or required state performance is given by y_m and the tracking error vector by e. The gain vector k is chosen so that the polynomial formed by $s^n + k_1 s^{n-1} + \ldots + k_n = 0$ has stable roots in the left half s-plane.

The underlying design problem is that of finding a controller design algorithm that guarantees closed-loop stability and allows for automated implementation. Since the direct adaptive method is the preferred method for use in this work, no further detail will be given on the indirect adaptive techniques.

2.5.2 DIRECT ADAPTIVE CONTROL

The structure of a direct adaptive controller is shown in Figure 9.

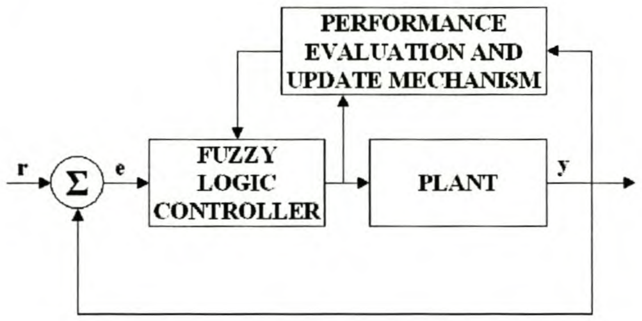

Figure 9 Direct adaptive fuzzy logic control architecture.

The performance evaluation and update mechanism implements a direct change in the controller parameters without estimating the model parameters [4, 51, 54, 87].

In the model reference adaptive system (MRAS) the closed-loop specifications are given in terms of a reference model, with the adaptive mechanism adjusting the controller parameters to reduce the response error as indicated in Figure 10 [4].

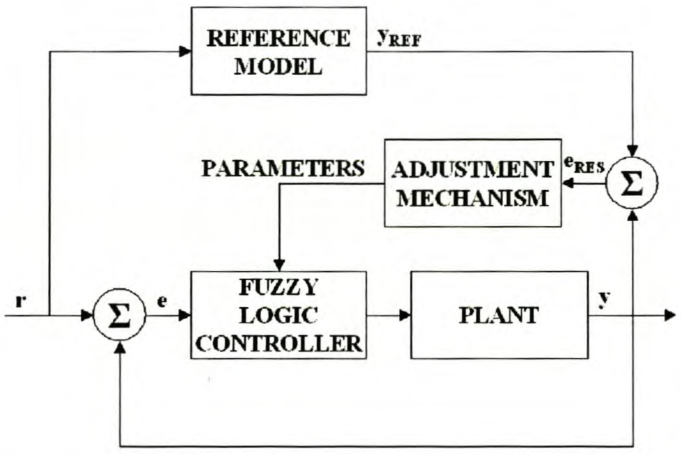

Figure 10 Model reference adaptive control system architecture.

The adjustment or learning mechanism adjusts the controller parameters (θ) and has been implemented in various ways. The gradient-based parameter modification algorithm is a popular method and is generally based on the MIT rule given by Equation 18 [4].

$$\frac{d\theta}{dt} = -\gamma e_{RES} \frac{\partial e_{RES}}{\partial \theta}$$

Equation 18

The sensitivity-based fuzzy self-learning scheme can only be applied if the fuzzy input-output mapping is analytical and differentiable [31]. A popular method based on fuzzy basis functions and Lyapunov functions uses a modification rule similar to the MIT rule as given by Equation 19.

$$\frac{d\theta}{dt} = -\gamma e_{RES} p_n \xi(x)$$

Equation 19

The p_n vector is the last column of the chosen Lyapunov weight matrix, e_{RES} the closed-loop system error dynamics and $\xi(x)$ the fuzzy basis function. The fuzzy basis functions are defined for fuzzy systems using singleton fuzzifiers, product inference and center average defuzzifier [87, 88]. This approach has been used successfully to tune the centers of the output membership function [88]. The method is limited to the specified form of fuzzy system for which the fuzzy basis function can be defined and will not be considered further in this work.

Learning methods based on a neuro-fuzzy architecture, with reinforced learning employing stochastic controller output variations and the approximated Jacobian, have been proposed [6, 84]. Learning methods based on reinforcement are complex, lack an intuitive feel and will not be considered further in this study [22].

The method proposed by Passino *et al*. uses a fuzzy inverse model that suggests the required change in the plant input. A knowledge-base modifier changes the active output membership function centers to ensure the controller output will be modified by the suggested amount [54]. This approach has been successfully used to design controllers for cargo ships, flexible link robots and high performance-aircraft [54]. A sensitivity function-based learning scheme can replace the fuzzy inverse model as shown in [31]. It is the opinion of this author that the proposed methods suffer from the following disadvantages:

1. The fuzzy inverse model generally has the same dimension as the controller, thus doubling the controller complexity.
2. The inverse model requires fundamental plant knowledge to suggest the appropriate amount of modification.
3. The sensitivity-based learning method requires an analytical and differentiable model of the fuzzy system and plant.
4. Since the knowledge-base modifier changes the centres of the output membership functions, it is possible to degrade the performance of a rule that uses the same output membership functions presently under modification. Online retuning is required to prevent

the possible degradation of non-active rules sharing consequences with active rules. An alternative would be to define a separate membership function for every rule raising the calculation burden of the defuzzifier.

The method proposed in Chapter 3 is based on the model reference adaptive architecture. A scheme to modify the consequence of rules responsible for poor closed-loop performance based on the response error and response error rate of change is proposed. The learning mechanism is modelled on the process a human would use to change the consequence of rules responsible for extreme overshoot or a response that is too slow.

2.6 PROBLEM DEFINITION AND SUGGESTED SOLUTION

From the preceding literature survey it is apparent that the design of an appropriate rule-base is critical in the determination of the closed-loop behaviour of complex non-linear plants with no or ill-defined mathematical models. The complexity of designing an FLC is influenced by the dimensions of the rule-base and the difficulty of obtaining the appropriate control rules.

The design problem addressed in this work is best explained considering a simple example. Assume that an FLC has to be designed for a non-linear, type zero, second-order plant. Assume that the small signal, linear dynamics differ substantially over the non-linear state space. Assume further that the FLC has to be implemented in a cheap hardware platform, typically a micro-controller or programmable logic controller-based system. Due to the hardware limitations imposed, the controller structure should be as simple as possible.

The two problems encountered in the design of an FLC are the selection of an appropriate control structure and finding the optimum control rules that will satisfy the required closed-loop specifications in the absence of expert control knowledge. The simplest form of FLC is a PID-type controller with error integral, error and error rate of change as inputs.

At present no simple method of rule-base design exists that uses qualitative plant knowledge. Of the existing methods of control rule-base design, the genetic algorithm seems to be the ideal solution. The application of a genetic algorithm requires a fast and accurate simulation model and many simulation runs to cover the search space. The practical application of the GA is limited and will thus not be considered further in this work. A direct adaptive controller is a good alternative for optimising of a rule-base. The self-adaptive algorithm can be used to tune a controller online (at the risk of system instability) or offline using a simulation model. If the hardware allows for the implementation of a direct adaptive algorithm in addition to the proposed controller, assume a rule-base can be found that performs well for a section of the non-linear state space. As the system states move away from the optimum zone, the closed-loop performance deteriorates. For example, if the system moves to a zone with a smaller open-loop bandwidth and lower damping than the optimum zone, the closed-loop response may become oscillatory with unacceptable overshoot due to high controller gain. A self-adaptive controller would be able to compensate for the unsatisfactory performance by adapting the rule-base. After an initial period of poor performance, the controller would converge to the new optimum. If the system states move back to the old domain, the controller would clearly have to re-adapt. This approach suffers from the following limitations:

1. The necessity of running a directly adaptive algorithm, in addition to the control algorithm to optimise the controller from domain to domain requires a more complex and costly hardware platform.
2. Due to limits placed on the hardware, it may not be possible to implement a self-adaptive algorithm, resulting in a controller with poor performance in certain domains of the state space.
3. In the time it takes to re-adjust the control rules, the closed-loop system may enter an unstable zone, or the plant can be damaged due to unacceptable closed-loop performance.

This would suggest the incorporation of non-linear domain knowledge into a non-adaptive multi-mode control structure. This knowledge can take the form of system states or variable system parameters. The addition of domain knowledge increases the rule-base dimensions and complexity of design. A method of rule-base design using qualitative plant information is thus required to automate the design of these complex rule-bases. In this dissertation it is shown how measurable plant parameters (such as the system states in the previous example) can be used to construct multi-mode controllers. A cost function approach is suggested to automate the process of the multi-mode rule-base design. The designed rule-base is then tested and modified using a model reference adaptive architecture to improve the closed-loop performance. The optimised controller incorporates sufficient control knowledge to allow for a simple lookup table implementation on most hardware platforms.

3 AUTOMATED RULE-BASE DESIGN ALGORITHM

In this chapter the automated rule-base design and rule-base modification algorithms are explained. A fuzzy logic controller design method is then proposed.

3.1 INTRODUCTION

Consider a time invariant, non-linear SISO plant where some measurable system state or external variable gives an indication of the system's non-linear modes. Assume that the plant is asymptotically stable in the large, according to the definitions of Lyapunov and that for any steady-state input there exists a unique steady state output [50].

3.2 FUZZY LOGIC CONTROLLER DESIGN ALGORITHM

The design problem can be defined as designing a stable MMFLC for the plant that will give a stable response over the non-linear domain.

For rapid response the FLC should react to the error and error rate of change signals (PD rule-base). For zero steady-state tracking error, error integration action should be included (PID rule-base). Most control engineers have a good feel for the design of a good PD rule-base, but inclusion of the integral information is complex. In the proposed method the integral action is incorporated into the rule-base. The control philosophy is to use the error and error rate of change to dominate control action far from the set point and the integral information closer to the set point. If the plant is highly non-linear, a PID rule-base that works well in one domain of operation may not work well in another. Therefore the concept followed in this work is to use different rules in different domains. This is similar to the concept of gain scheduling.

The structure of a PID type MMFLC, as proposed in this study, is shown in Figure 11.

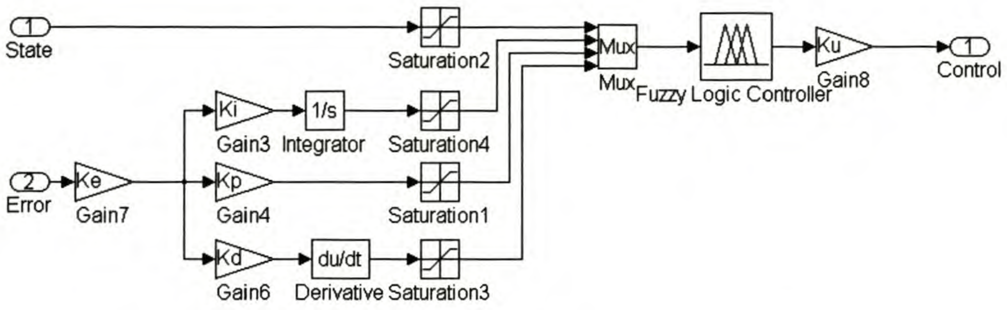

Figure 11 PID-type MMFLC block diagram.

The inputs to the PID-type MMFLC are the system state (x_s), error integral ($\int e.dt$), error (e) and error rate of change (de/dt). Assume that the number of fuzzy terms and corresponding membership functions have been defined for each input. In the work that follows the membership functions are chosen according to the following principles [21]:

1. The triangular membership functions are evenly spaced and symmetrical on the error integral, error and error rate of change universe of discourse. Triangular membership functions are used to define the fuzzy sets of the state input.
2. The membership functions overlap 100%. This implies that a membership function reaches zero at its neighbour's apex. Each value of the universe of discourse will thus always be a member of two sets.
3. There should be at least five sets for the error input, error integral and error rate of change with two sets for the state input. More sets can be added as required.

The form of the PID rules is:

IF (*STATE is* X_i) **AND** (*ERROR INTEGRAL is* I_j) **AND** (*ERROR is* E_k) **AND** (*ERROR RATE OF CHANGE is* R_l) **THEN** (*CONTROL is* U_{ijkl}).

The error integral term can be ignored for PD control of type 0 systems.

3.2.1 AUTOMATED RULE-BASE DESIGN ALGORITHM

The automated rule-base design algorithm must design a complete rule-base using information regarding the speed of response and the small signal gain in the different non-linear state domains. The non-linear domains are defined by the fuzzy sets chosen for x_s. In domains where the plant responds rapidly or with a large gain, the control action should be less severe (lower control gain due to higher bandwidth or plant gain) compared to a section where the response is slow or small. The rule-base design algorithm generates a PD-type rule-base, based on the gain and speed information in the various system domains. The rule-base is then augmented with the integral term.

The augmented rule-base consequences are clustered and normalised. Finally, the output fuzzy sets are defined to correspond to the generated rule-base.

The rule-base generation is based on the sum of weights assigned to each fuzzy set in the $\int e.dt$, e and *de/dt* universe of discourse. Assume there are *J* fuzzy sets defined in the $\int e.dt$ universe of discourse with corresponding weights in the I^W vector (dimension $1 \times J$). Similarly define for the *K* fuzzy sets in the *e* universe of discourse the E^W weight vector (dimension $1 \times K$) and for the *L* fuzzy sets in the *de/dt* universe of discourse the R^W weight vector (dimension $1 \times L$). The elements of the weight vectors correspond to the fuzzy sets defined by the membership functions, in order from left to right in the universe of discourse. The choice of weights can be done in an arbitrary fashion as long as it conforms to the following guidelines:

6. The zero fuzzy set has a weight of zero.
7. The sign of the weight corresponds to the negative or positive side of the universe of discourse where the defining membership function is located.
8. The size of the weight indicates the distance of the defining membership function from zero.
9. The distribution of weights can be linear or non-linear.

In this study the membership functions are distributed evenly over the universes of discourse and a linear distribution of weights is used.

The choice of the gain (G_I) and speed (S_I) information vectors should reflect the relative small signal gain and bandwidth of the plant in the various non-linear domains as defined by the state input fuzzy sets. Assume that *I* fuzzy sets are defined in the x_s universe of discourse. The Gain and Speed information vectors can be chosen based on information obtained from small signal linear models, physical plant properties or operator knowledge of the plant. The algorithm normalises the G_I and S_I vector relative to its minimum and maximum elements as shown by Equation 20. This allows for a relative comparison in gain and bandwidth of the various non-linear domains.

$$G^W = \frac{\min(G_I)}{G_I}$$

$$S^W = \frac{S_I}{\max(S_I)}$$

Equation 20

The PD rule-base is generated so that the error will give a fast initial drive towards the desired state while the derivative information will anticipate possible overshoot and start reducing the drive signal close to the set point. The algorithm generates a rule-base with higher proportional gain in domains where the system small signal gain is low. The addition of anticipation with the derivative term is not as simple and various approaches can be developed. The assumption is made that system domains with a lower bandwidth and damping could be more prone to overshoot due to rapid transient responses and are therefore closer to instability than domains with a higher bandwidth and damping. A larger penalty should thus be imposed on derivative information in the lower bandwidth and damping domains in an attempt to limit overshoot caused by a high proportional gain. In cases where this initial choice of S_I does not deliver acceptable results, the rule-base damping could be modified experimentally.

The normalised gain weight (G^W) and speed weight (S^W) vectors are used with the E^W and R^W vectors to form a control weight indicative of the sign and amount of control energy to be used. Let the control weight associated with a given rule be U_{ijkl}, where the subscript i refers to a specific fuzzy set in x_s, j in $\int e.dt$, k in e and l in de/dt. The control weight is determined according to Equation 21.

$$U_{ijkl} = E_k^W \times G_i^W + \frac{R_l^W}{S_i^W}$$

Equation 21

In a given system domain a larger error would thus increase the control weight as determined by E^W. System domains with a large gain would thus reduce the control weight. In a similar fashion the control weight is increased or decreased according to R^W and S^W.

The next step is to augment the control rule weight with an integral term to eliminate any steady-state errors. The proposed method is to add a term to the control rule weight proportional to the weight associated with fuzzy set j in I^W. The added term is proportional to the difference between the control rule weight and the maximum control rule weight as shown in Equation 22.

$$U_{ijkl}^{Aug} = U_{ijkl} + \frac{\left(1 - \left|U_{ijkl}\right|\right) \times I_j^W}{\max\left(I^W\right)}$$

Equation 22

This ensures a saturation of the controller for small errors and large error integrals and vice versa.

The augmented control rule weight is indicative of the required control energy. This weight should thus be associated with a fuzzy set in the output universe of discourse. It is clear that the proposed algorithm can generate an arbitrarily large number of weights and thus proposed consequences.

Some of the control weights could be sufficiently close together to be grouped. The next step is to cluster the control rule weights and associate each cluster with a separate control output fuzzy set. The consequence of a rule is then the fuzzy set associated with the cluster closest to the control rule weight. To allow for precise control in different dynamic regions, care should be taken not to reduce the amount of clusters beyond the limits required for proper control. The fuzzy c-means clustering method was chosen for reduction [21].

3.2.2 FUZZY LOGIC CONTROLLER DESIGN RULES

The proposed design algorithm reduces the design complexity to an appropriate choice of G_I and S_I. The design process can now be summarised as follows.

1. *Define controller inputs and output*: Determine what type of controller is required (PD or PID). Find a measurable system state or external variable that reflects the non-linearity of the plant. Define the universe of discourse for each of the inputs and the output.

2. *Define fuzzy sets*: Choose fuzzy sets and define the corresponding membership functions for each input as suggested in paragraph 3.2. For each fuzzy set, choose a weight according to the rules defined in paragraph 3.2.

3. *Choose gain and speed information*: Choose the G_I and S_I vectors. The elements should describe the relative gain, bandwidth and damping for each domain defined by a fuzzy set in the x_s universe of discourse.

4. *Run automated design algorithm*: Choose the required number of fuzzy sets in the output and run the automated design algorithm as detailed in paragraph 3.2.1

5. *Test controller performance*: Choose the controller input gains (K_i, K_p and K_d). The controller can be tested online or with computer simulation.

6. *Design modification*. Evaluate the closed-loop response. Modify the controller input gains to shape the closed-loop response in all the non-linear domains as required (global tuning) [21]. Response modification limited to individual domains can be done by modification of G_I and S_I (local tuning) [21].

The rule-base generation algorithm as described in paragraph 3.2.1 can be fully automated. The automation of the process allows for the fast design of complex controllers. The process is intuitively understandable and user modification of the controller structure and rule-base can be eliminated. The MATLAB® file implementation of the proposed algorithm is given in Appendix I.

3.3 RULE-BASE TUNING ALGORITHM

The structure of a PID-type SAFLC, as proposed in this study, is shown in Figure 12.

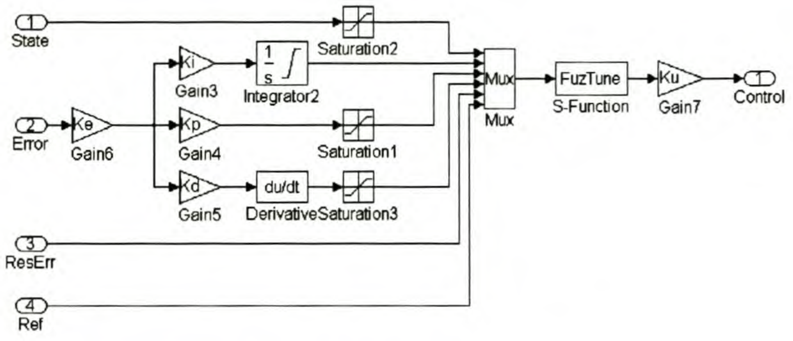

Figure 12 SAFLC block diagram.

The rule-base tuning algorithm is based on the concept of evaluating the effect of the firing rules on the closed-loop response. The consequences of rules with unsatisfactory behaviour are modified. The closed-loop response is evaluated against a reference response defined by a reference model. The rule-base update is based on the difference between the reference response and the actual closed-loop response (response error).

Model Reference Adaptive Control (MRAC) has traditionally been used to tune the control parameters of linear controllers when used with non-linear, time varying plants [4]. In the following work the rule-base update signal (e_u) is composed of the sum of the response error (e_r) and response error rate of change (de_r/dt) as shown in Equation 23.

$$e_u = K_{REp}\left(e_r + K_{REd} \times \frac{de_r}{dt} \right)$$

Equation 23

The addition of the derivative term anticipates possible improvements in the response and limits unnecessary rule modifications. This allows for faster rule-base update with less overshoot. The block diagram implementation of this procedure is shown in Figure 13.

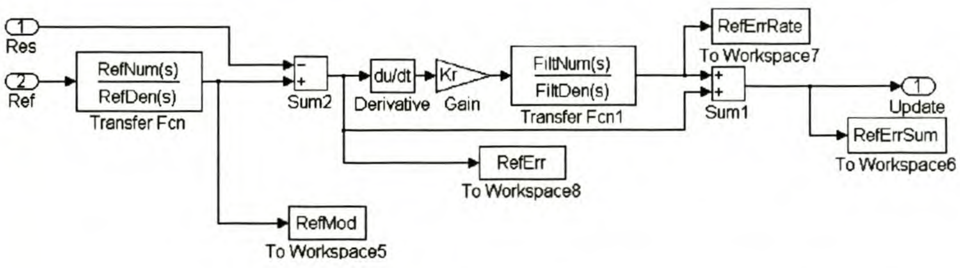

Figure 13 Reference model and response update block diagram.

Assume that a change in the plant output at time t_x is the response to a control action at time $t_x - \Delta T$. Due to the overlap in the membership functions defining the fuzzy sets, a number of rules are activated at any given time. The controller output is determined primarily by rules with a high firing strength. Blame for poor response should be assigned to active rules proportional to their firing strengths. A negative response error at time t_x is caused by the activation of rules at time $t_x - \Delta T$ with too much control energy. The faulty rules should be modified by changing their consequences to the left of their present ones. A positive response error requires a shift in consequence to the right. The consequence of a rule is confined to being an element of the term set defined to the output universe of discourse. A decision mechanism is required to control the switching in rule consequence based on rule performance.

Consider a SAFLC at time t_x with e_u as defined by Equation 23. Assume that e_r is caused by a control action at $t_x - \Delta T$, and that rules $R_{x-\Delta}$ were the active rules with firing strengths $\alpha_{x-\Delta} > 0$. Define an integrator for each rule in the rule-base with input the product of e_u at t_x and the rule firing strength (α) at $t_x - \Delta T$. Only rules participating in an unsatisfactory control action will show a change in integrator output. The dominant rules will show a larger change in integrator output than non-dominant rules. As soon as the output of an integrator exceeds a set limit (Γ or $-\Gamma$) the rule consequence is changed one membership function to the right for a positive integrator output, or to the left for a negative output. The integrator output is then reset to zero, and the performance of the modified rules is evaluated.

From Equation 23 it is clear that the tuning speed of the SAFLC is determined by Γ, K_{REp} and K_{REd}. In this work Γ was arbitrarily set to the average distance between output fuzzy set membership function centers. The tuning speed was controlled with the constants defined in Equation 23. It was found that choosing K_{REd} so that the maximum value of de_r/dt is almost the same as the maximum value of e_r delivers good tuning performance. K_{Rep} can then be tuned experimentally to deliver fast tuning, but prevent fast switching in the consequence of a specific rule (rule consequence limit cycling).

From the finite choices available for a rule consequence, it may not be possible to find a perfect solution to the problem. A certain rule consequence may react with too little control energy, while the next consequence may respond too strongly. The user must thus define a philosophy to determine when a rule-base can be considered as giving satisfactory performance and the tuner switched off. One possible solution would be to study the RMS value of the response error and terminate the tuning procedure as soon as the RMS value has been reduced by a set percentage. If rule consequence limit cycling is detected and the termination criteria not reached, more output fuzzy sets may be required in the output term set.

3.4 CONCLUSIONS

An automated rule-base design algorithm is proposed in paragraph 3.2.1. Design rules for MMFLCs are given in paragraph 3.2.2. A model reference adaptive scheme for evaluating and improving the performance of the controller by modifying the rule-base is proposed in paragraph 3.3. This work suggests the following approach to fuzzy logic controller design.

Based on the non-linearity of the plant, decide if a linear or fuzzy logic controller is required. Due to its simplicity (smaller rule-base) a SMFLC should initially be tested. The SMFLC can be designed using the proposed rule-base design algorithm. If the control objectives do not allow for the use of the model reference adaptive architecture, iterate on the rule-base design by tuning the controller input gains and the cost function weights until a satisfactory response is obtained. If model reference adaptive control is possible, choose a reference model based on a realistic performance specification and apply the self-adaptive scheme as proposed in paragraph 3.3. If the SMFLC provides satisfactory performance, this controller can be implemented or a MMFLC can then be tried and tested for performance improvement.

4 AUTOMATED RULE-BASE DESIGN ALGORITHM EVALUATION

The performance and working of the automated rule-base design algorithm and rule-base adaptive algorithm are tested by application to three highly non-linear plants: a cargo ship auto-pilot, a hypothetical second-order plant with exponential non-linear position feedback and a hypothetical second-order plant with non-linear velocity feedback similar to the non-linear drag force encountered by high-performance aircraft. For each plant a single mode fuzzy logic controller (SMFLC) and multi-mode fuzzy logic controller (MMFLC) are designed and tested. Both controllers are designed using the method proposed in paragraph 3.2.2. The SMFLC input gains are determined experimentally to improve the closed-loop performance. The MMFLC uses identical input gains with performance tuning accomplished by experimental modification of the gain and speed information vectors. The non-adaptive controller rule-bases form the initial conditions for the single mode adaptive fuzzy logic controller (SMAFLC) and the multi-mode adaptive fuzzy logic controller (MMAFLC).

The performance of the non-adaptive controllers is based on the closed-loop step responses. The 10% to 90% rise time ($T_{r\,(10\%\text{-}90\%)}$), maximum overshoot ($M_p$) and steady-state error (E_{ss}) are used as Key Performance Indicators (KPIs). The performance of the self-adaptive controllers is based on the RMS values of the response errors. The RMS values of the tuning controller and final controller response error are compared to those of the initial controllers. The RMS values of the response error for individual step changes are used to determine the speed of convergence. The point midway between the maximum and minimum RMS values is taken as the critical point. Since the RMS value is indicative of the energy in a signal, this form of measurement corresponds to a –3dB point in response error energy.

4.1 PLANT I: CARGO SHIP STEERING

The cargo ship steering problem is a literature case study [35,54]. The authors report on the performance of a self-learning fuzzy logic controller (SLFLC) for the 161m-long cargo ship. A comparison is made between the SLFLC and standard self-adaptive techniques. Passino *et al.* tested the ship heading response to ±45° heading reference step changes at a forward velocity of 5.0m/s. A second-order, critically damped system with poles at $s_{1,2} = -0.05$ formed the reference model. This closed-loop performance specification, relative to the open-loop system response, forms the benchmark for this study.

4.1.1 PLANT AND SIMULATION MODEL

The simulations conducted here assume the ship is big enough to ignore motion in the vertical plane. The motion of the ship is defined in a coordinate system fixed to the ship as shown in Figure 14.

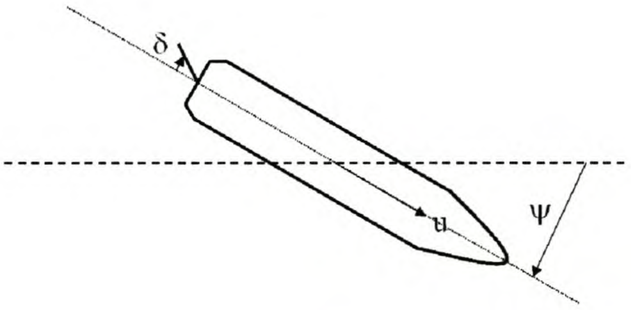

Figure 14 Plant I – Cargo ship.

Define the ship forward velocity as u, the rudder deflection angle as δ and the ship heading relative to a fixed coordinate system as ψ. Note the directions of the rudder deflection angle and the heading response angle. The approximate heading dynamics as a function of u and δ are given by Equation 24 [35,54].

$$\dddot{\psi} = -k_1 \ddot{\psi} - k_2 H\left(\dot{\psi}\right) + k_3 \delta + k_4 \dot{\delta}$$

$$H\left(\dot{\psi}\right) = a\left(\dot{\psi}\right)^3 + b\dot{\psi}$$

Equation 24

The ship length (L) and u determines the system gain and time constants as given by Equation 25.

$$k_1 = \frac{\tau_1 + \tau_2}{\tau_1 \tau_2}$$

$$k_2 = \frac{1}{\tau_1 \tau_2}$$

$$k_3 = \frac{K}{\tau_1 \tau_2}$$

$$k_3 = \frac{K\tau_3}{\tau_1 \tau_2}$$

$$\tau_i = \tau_{i0}\left(\frac{l}{u}\right) : i = 1,2,3$$

$$K = K_0\left(\frac{u}{l}\right)$$

Equation 25

The ship parameters as determined by the spiral tests are shown in the parameter list below [35]. The maximum rudder angle deflection is given as ±80° [35].

- $L = 161m$
- $\tau_{10} = 5.66s$
- $\tau_{20} = 0.38s$
- $\tau_{30} = 0.89s$
- $K_0 = -3.86$
- $A = 1.0$
- $B = 1.0$

For the simulation studies, the third-order non-linear differential equation (Equation 25) needs to be transformed into a non-linear state space system. Equation 26 gives the non-linear state space equation with the assumed state definitions [54].

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\dot{x}_3 = -k_1 x_3 - k_2 H(x_2) + (k_3 - k_1 k_4)\delta$$

$$x_1 = \psi,\ x_2 = \dot{\psi},\ x_3 = \ddot{\psi} - k_4 \delta$$

Equation 26

The constants are identical to those defined in Equation 25. In the simulation study the system is regarded as a two-input (*u* and δ), two-output system (ψ and dψ/dt). For simulation purposes, the maximum rudder deflection angle was constrained to ±60°. Extracting small signal linear models from Equation 26 is achieved by assuming x_2 to be small and approximating $H(x_2)$ by only the linear part. Table 1 summarises the small signal models obtained at three steady state operating points of *u*. The faster the forward velocity, the larger the small signal gains and quicker the response times.

δ_0	x_{10}	x_{20}	u_0	Poles			Zero	*G*
0	**0**	**0**	**2.5m/s**	**0**	**-0.0027**	**-0.0409**	**-0.0174**	**-0.0599**
0	**0**	**0**	**5.0m/s**	**0**	**-0.0055**	**-0.0817**	**-0.0349**	**-0.1199**
0	**0**	**0**	**7.5m/s**	**0**	**-0.0087**	-0.1226	**-0.0523**	**-0.1798**

Table 1 Plant I – Small signal linear model properties.

The open-loop heading response of the cargo ship to a 60° step change in δ is shown in Figure 15. The response is shown at three forward velocities (2.5m/s, 5.0m/s and 7.5m/s). As *u* increases, the system response time decreases and turning speed increases. The cargo ship requires 80s, 40s and 28s to make 45° of turn at the respective velocities. The reference model responses used in the SMAFLC and MMAFLC, at the three forward velocities, are also shown. At 5.0m/s the required closed-loop settling time is given as 100s [35,54]. The closed-loop settling time specification at 2.5m/s and at 7.5m/s is accordingly taken as 100s and 70s. This gives the ratio of closed-loop settling speed relative to the maximum possible response speed as 2.5.

The SIMuLINK simulation S-Function file is listed in Appendix III.I.

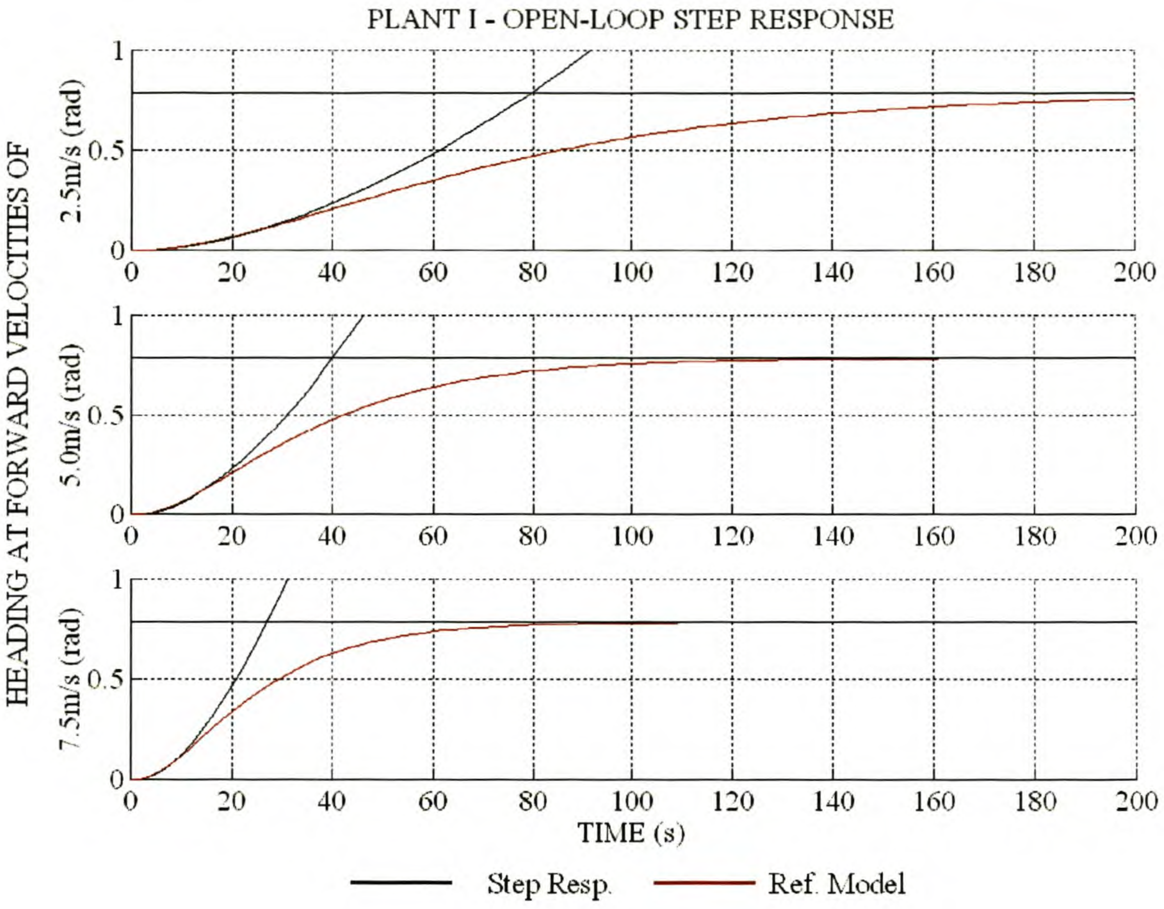

Figure 15 Plant I – Open-loop step response.

4.1.2 FUZZY LOGIC CONTROLLER DESIGN AND PERFORMANCE

The block diagram of the SMFLC and MMFLC is shown in Figure 16.

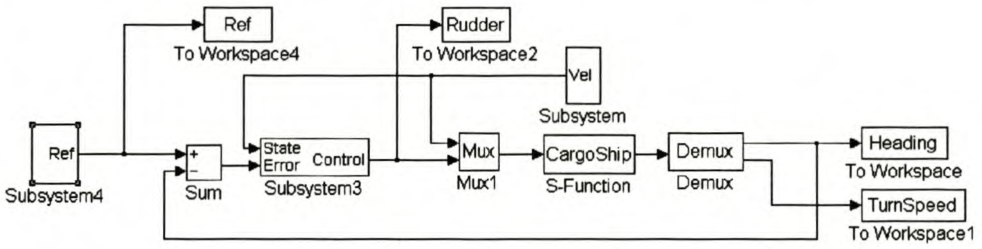

Figure 16 Plant I – Non-adaptive fuzzy logic control.

Since the ship is a type one system, a PD-type FLC is used. The inputs to the FLC is u, the heading error (e) and heading error rate of change (de/dt). The controller output is rudder deflection angle (δ). The ship's forward velocity (u) is an independent parameter allowing the separate investigation of the different system domains. The heading response was tested at 2.5m/s, 5.0m/s and 7.5m/s. The simulation is divided into three time zones. In zone 1 (0s – 1800s) the forward velocity is

2.5m/s, in zone 2 (1800s – 3600s) 5.0m/s and in zone 3 (3600s – 5400s) 7.5m/s. The $T_{r\,(10\%\text{-}90\%)}$, M_p and E_{ss} are used as KPIs for controller evaluation.

In accordance with the references [35, 54], the SMFLC and MMFLC implements 11 fuzzy sets for *e*, *de/dt* and δ. The SMFLC implements one fuzzy set for *u* and the MMFLC three fuzzy sets. A normalised universe of discourse was chosen for *e* and *de/dt*. The SMFLC and MMFLC structures are summarised in Table 2.

Input	SMFLC			MMFLC		
	Minimum	Maximum	Sets	Minimum	Maximum	Sets
State	**2.5**	**7.5**	**1**	**2.5**	**7.5**	**3**
Error	**-1**	**1**	**11**	**-1**	**1**	**11**
Rate	**1**	**1**	**11**	**1**	**1**	**11**
Control	**-1.047**	**1.047**	**11**	**-1.047**	**1.047**	**11**

Table 2 Plant I – SMFLC and MMFLC structure summary.

Since the integrator dominates the system dynamics, the initial S_I vector was chosen to be the unity vector and the initial G_I vector the linear small signal model transfer function gains. The final S_I and G_I vectors were determined experimentally and are given by Equation 27.

$$G_I = \begin{bmatrix}0.0599 & 0.1198 & 0.1798\times 0.75\end{bmatrix}$$
$$S_I = \begin{bmatrix}0.4 & 1 & 1\end{bmatrix}$$

Equation 27

The maximum δ of ±60° constrained the output universe of discourse. The controller input gains (K_p and K_d) were determined experimentally to map the corresponding universes of discourse to $e \in$ [-10° 10°] and *de/dt* to $\in$ [-0.01rad/s 0.01rad/s]. The gains were chosen to give a strongly damped response at 5.0m/s. The MATLAB® design files are listed in Appendix III.II for the SMFLC and Appendix III.III for the MMFLC.

The closed-loop response of the SMFLC and MMFLC is shown in Figure 17 and Figure 18.

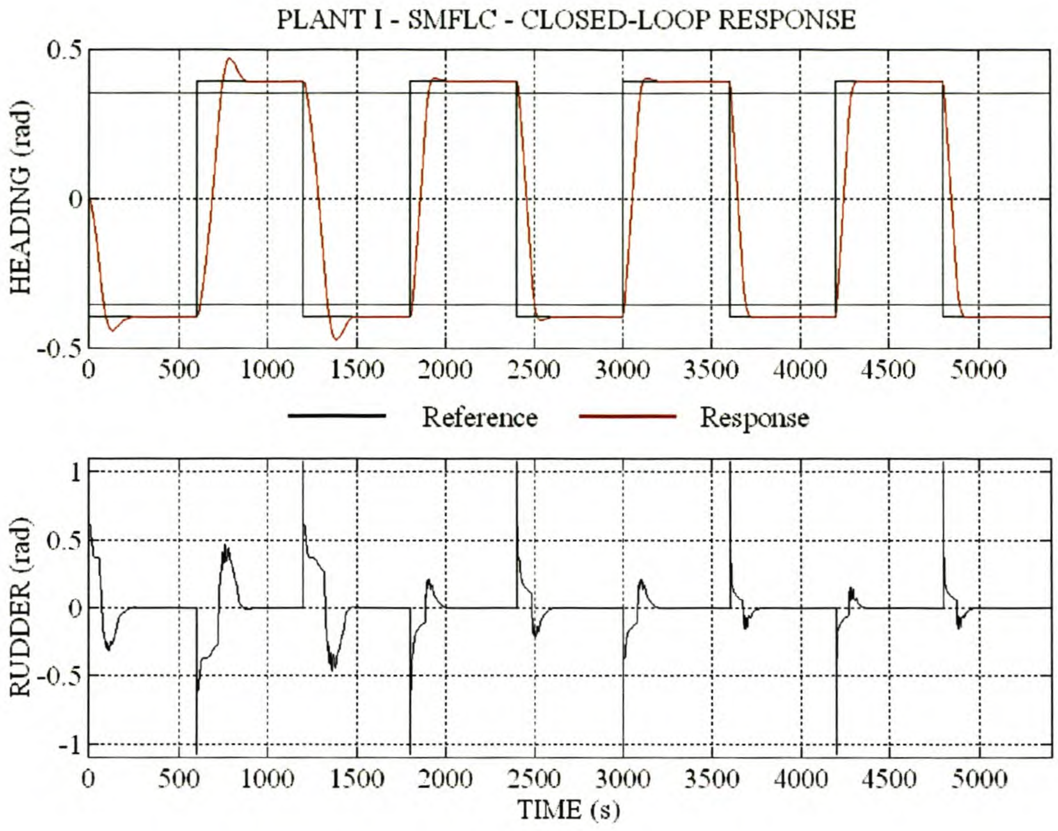

Figure 17 Plant I – SMFLC – Closed-loop response.

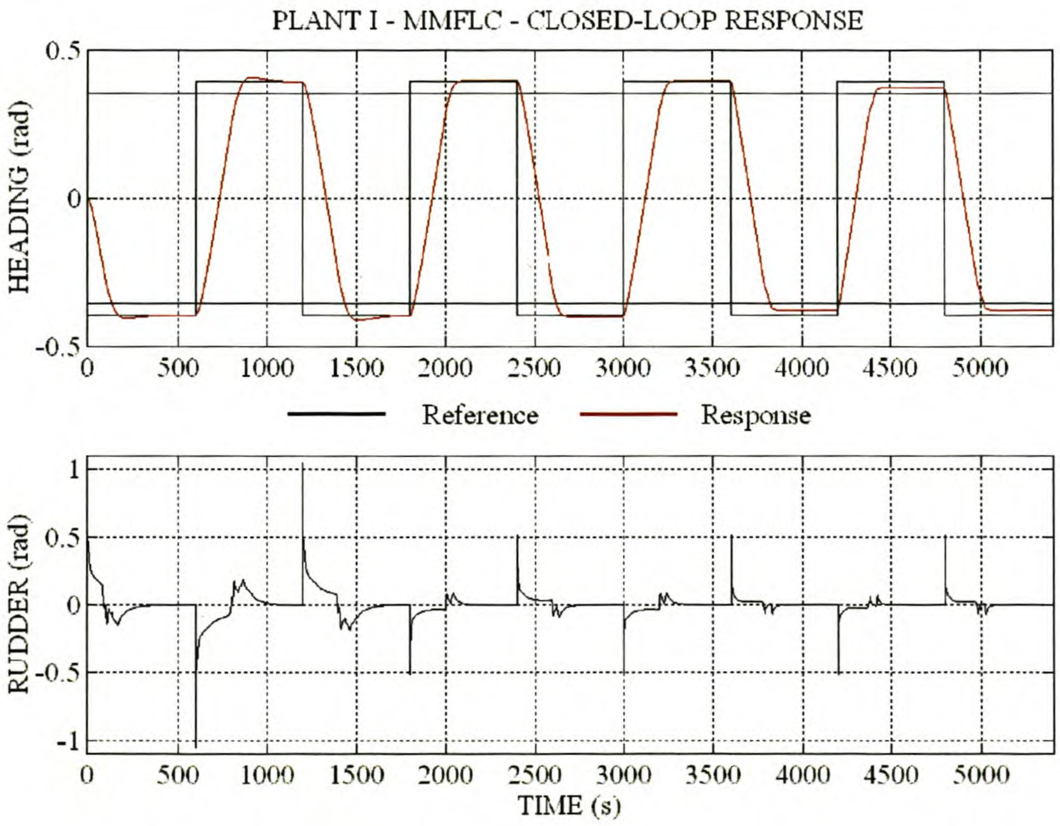

Figure 18 Plant I – MMFLC – Closed-loop response.

The KPIs for the SMFLC and MMFLC are compared in Table 3.

Velocity (u)	SMFLC			SMFLC		
	$T_{r\,(10\%-90\%)}$	M_p	E_{ss}	$T_{r\,(10\%-90\%)}$	M_p	E_{ss}
2.5m/s	**118s**	**10.07%**	**0%**	**177s**	**2.1%**	**0%**
5.0m/s	**90s**	**1.69%**	**0%**	**173s**	**0.0%**	**0.8%**
7.5m/s	**85s**	**0.28%**	**0%**	**174s**	**0.0%**	**2.24%**

Table 3 Plant I – Performance comparison of SMFLC vs MMFLC.

The SMFLC responded faster but with more overshoot than the MMFLC. As the ship's forward velocity increased, the response time of the SMFLC decreased significantly (Figure 17). The response time of the MMFLC is not affected by the change in forward velocity (Figure 18). The longer response time of the MMFLC is due to the decreased gain of the rule-base due to the automated design algorithm. Since the controller gain is reduced in certain domains of the multi-mode rule-base, retuning of the controller input gains are required to reach a speed of response equal to that of the SMFLC.

4.1.3 SELF-ADAPTIVE CONTROLLER DESIGN AND PERFORMANCE

The block diagram of the SMAFLC and MMAFLC is shown in Figure 19.

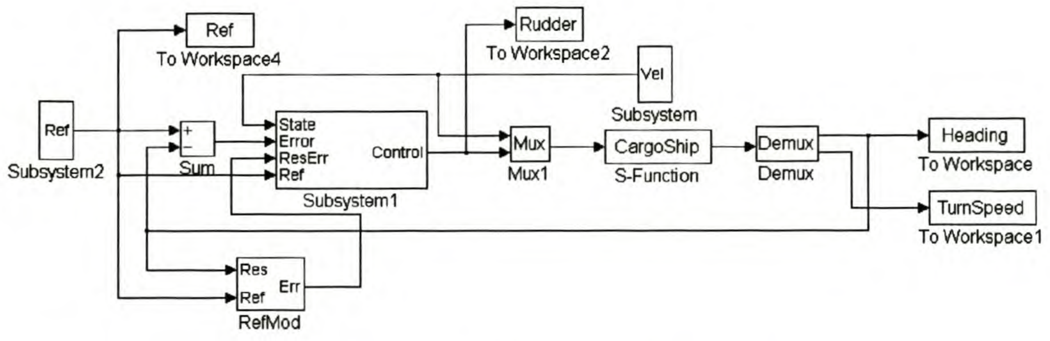

Figure 19 Plant I – Adaptive fuzzy logic control.

The structures of the SMAFLC and MMAFLC are identical to those given in paragraph 4.1.2. The reference models are three critically damped, second-order systems with time constants τ_{r1} = 200, τ_{r2} = 100 and τ_{r3} = 70 at 2.5m/s, 5.0m/s and 7.5m/s respectively, as discussed in paragraph 4.1.1.

The controller input gains (K_p and K_d) were chosen to map the maximum encountered values to the normalised universes of discourse. The tuner gain parameters were determined experimentally to give good tuning speed without starting limit cycle oscillations in the rule-base. The values of the SMAFLC and MMAFLC parameters are $K_p = 1.273$, $K_d = 0.01$, $K_{REp} = 0.1$ and $K_{REd} = 25$. The closed-loop position response to 45° step changes was used as test conditions.

The closed-loop performance of the SMAFLC and MMAFLC is shown in Figure 20 and Figure 25. The simulation period can be divided into three time zones. In zone 1 (0s – 8000s) the forward velocity is 2.5m/s, in zone 2 (8000s – 12000s) 5.0m/s and in zone 3 (12000s – 14800s) 7.5m/s. The detailed response of the SMAFLC and MMAFLC is shown in Figure 21 and Figure 26. The response of the initial controller is shown in Figure 22 and Figure 27. The final controller response is shown in Figure 23 and Figure 28. The RMS value of the response error for each step during the tuning operation is shown in Figure 24 for the SMAFLC and in Figure 29 for the MMAFLC. The RMS values of the response error for the SMAFLC and the MMAFLC are compared in Table 4.

Zone	SMAFLC			MMAFLC		
	Initial	Tuning	Final	Initial	Tuning	Final
1 (0s - 8000s)	**0.0830**	**0.0076**	**0.0741**	**0.2126**	**0.0094**	**0.0181**
2 (8000s - 12000s)	**0.1730**	**0.0398**	**0.0680**	**0.3023**	**0.0660**	**0.0119**
3 (12000s - 14800s)	**0.2104**	**0.0606**	**0.1447**	**0.3020**	**0.1348**	**0.0252**

Table 4 Plant I – RMS error comparison of SMAFLC vs MMAFLC.

In zone 1 the SMAFLC response (Figure 21 a) improved the initial controller response (Figure 22 a) to almost perfect tracking, with a reduction of 90% in the RMS value of the response error. The tuning process converged in three step changes (Figure 24 a). The final controller response (Figure 23 a) showed excessive overshoot with a reduction in the RMS value of the response error of only 10%. In zone 2 the SMAFLC response (Figure 21 b) showed good tracking compared to the initial controller response (Figure 22 b), with a reduction in the RMS value of the response error of 76%. The tuning operation shows convergence to an oscillating rule-base in six step changes (Figure 24 b). The final controller response (Figure 23 b) shows an improvement in the RMS value of the response error of 60%. In zone 3 the SMAFLC response (Figure 21 c) reduced the RMS value of the response error by 71% compared to the initial controller response (Figure 22 c). The tuning operation converged after one step change to an oscillating rule-base (Figure 24 c). The final controller response (Figure 23 c) showed a reduction of 31% in the RMS value of the response error. A detail investigation of the modified rules indicated large oscillations in the conclusions of rules active in the initial period of the step response of zone 2 and zone 3. This is caused by a too high tuning speed gain (selected for fast convergence in zone 1) in the two domains and results in the unexpected poor performance of the final controller in zone 3.

The MMAFLC allows for the selection of different tuning speed gains for the various domains of the rule base. To prevent the large limit cycles in rule conclusions in zone 2 and zone 3 the local tuning speed gains were reduced by factor 7.5 and 17.5 in the respective zones. In zone 1 the MMAFLC response (Figure 26 a) improved the initial controller response (Figure 27 a) to almost perfect tracking, with a reduction of 95% in the RMS value of the response error. The tuning process converged in four step changes (Figure 29 a). The final controller response (Figure 28 a) showed very good tracking, with a reduction in the RMS value of the response error of 91%. In zone 2 the MMAFLC response (Figure 26 b) showed good tracking compared to the initial controller response (Figure 27 b), with a reduction in the RMS value of the response error of 78%. The tuning operation shows convergence after five step changes (Figure 29 b). The final controller response (Figure 28 b) shows an improvement in the RMS value of the response error of 96%. In zone 3 the MMAFLC response (Figure 26 c) reduced the RMS value of the response error by 55% compared to the initial controller response (Figure 27 c). The tuning operation converged after ten step changes (Figure 29 c). The final controller response (Figure 28 c) showed a reduction of 91% in the RMS value of the response error.

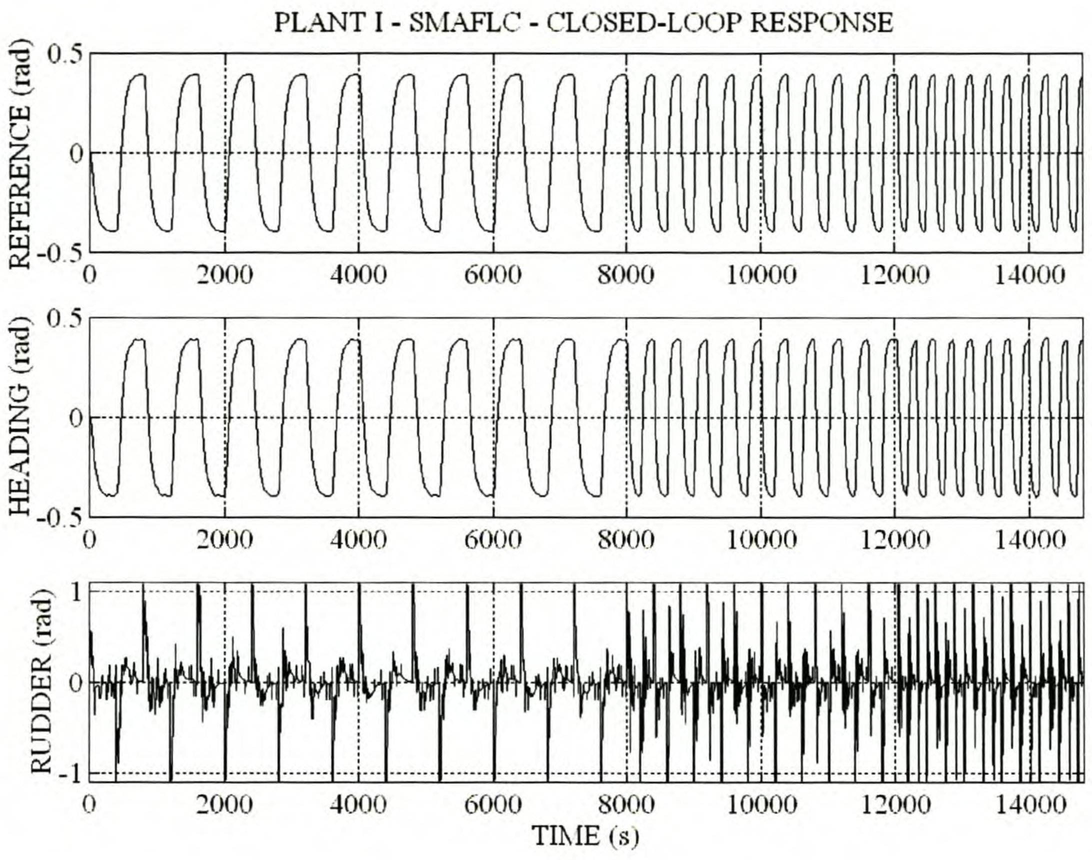

Figure 20 Plant I – SMAFLC – Closed-loop response.

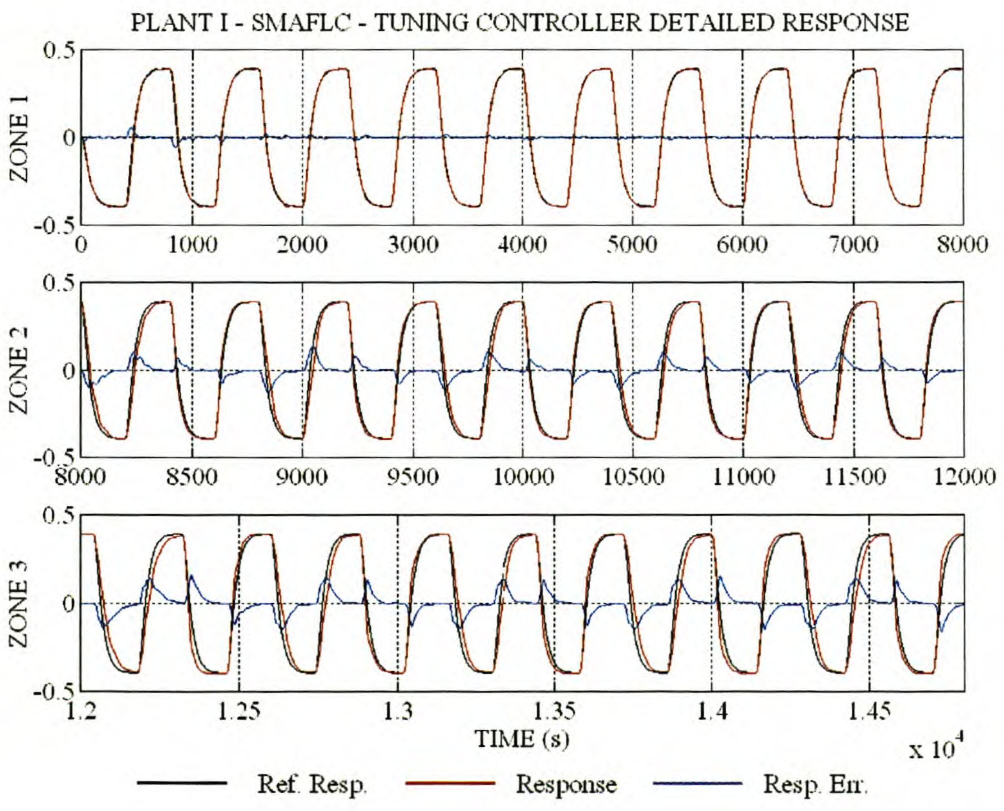

Figure 21 Plant I – SMAFLC – Tuning controller detailed response.

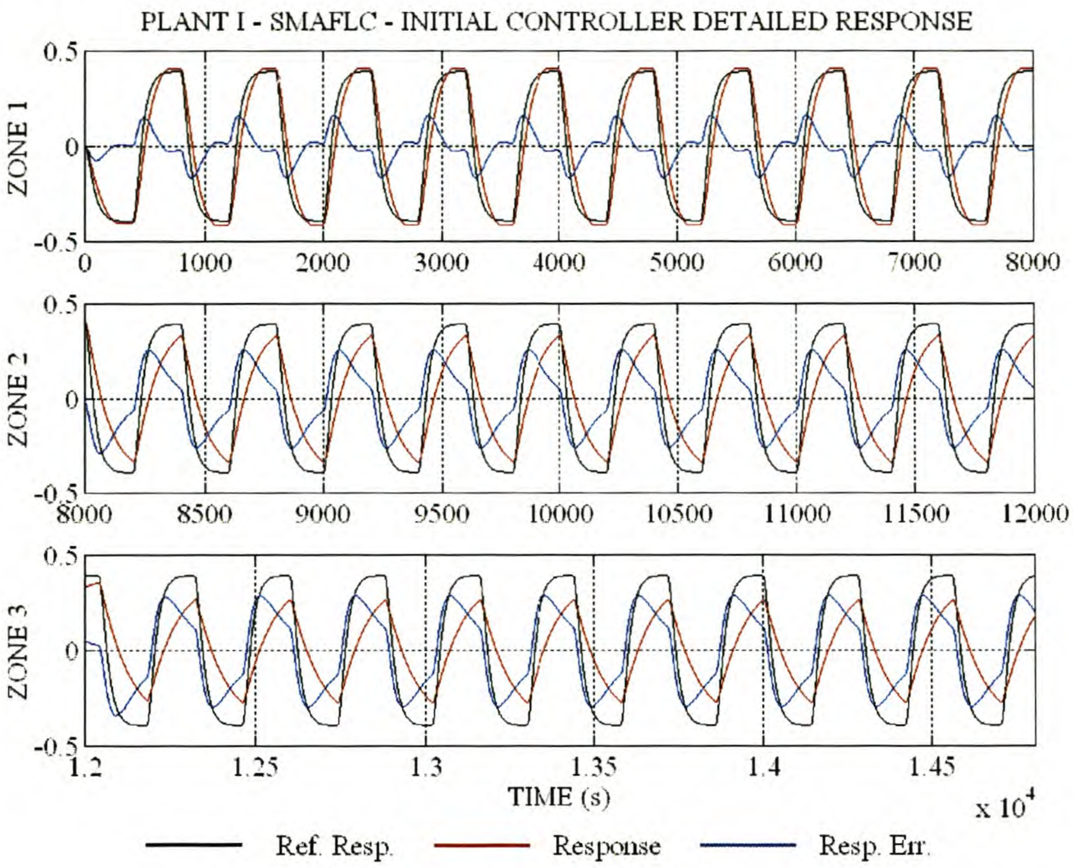

Figure 22 Plant I – SMAFLC – Initial controller detailed response.

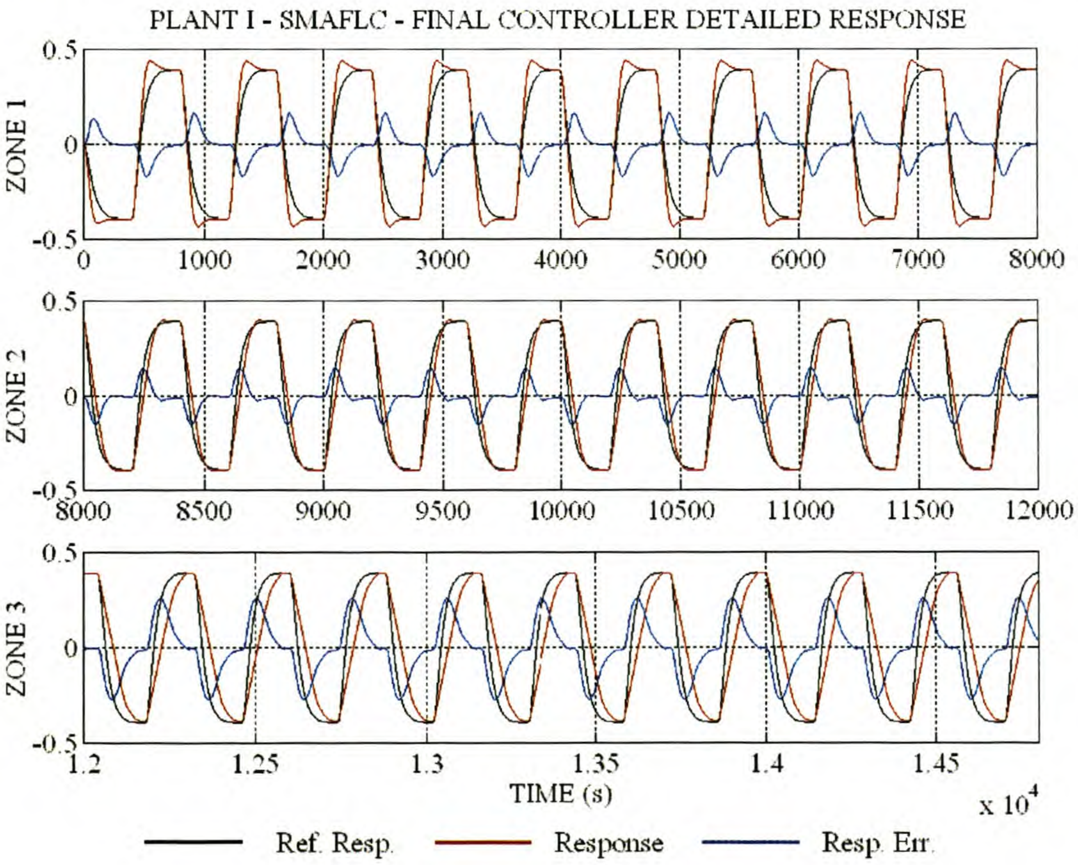

Figure 23 Plant I – SMAFLC – Final controller detailed response.

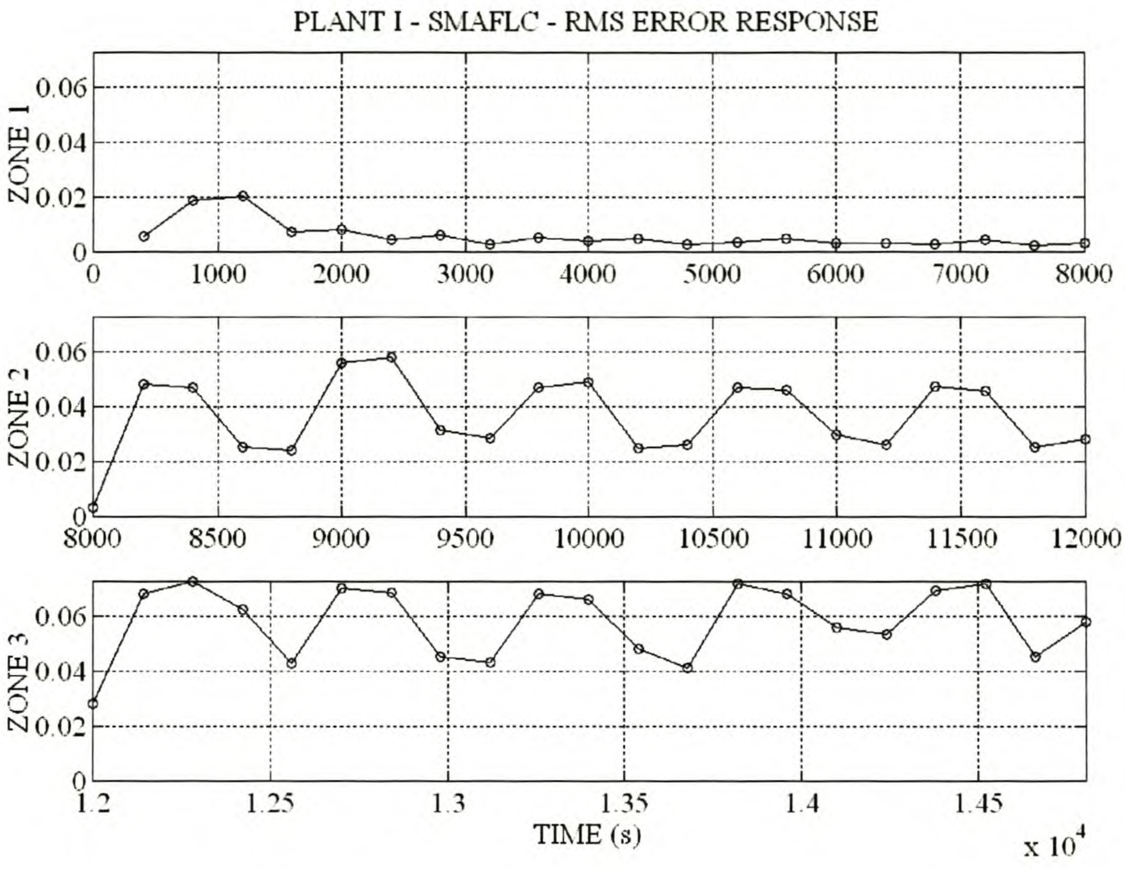

Figure 24 Plant I – SMAFLC – Response error RMS value for each step change.

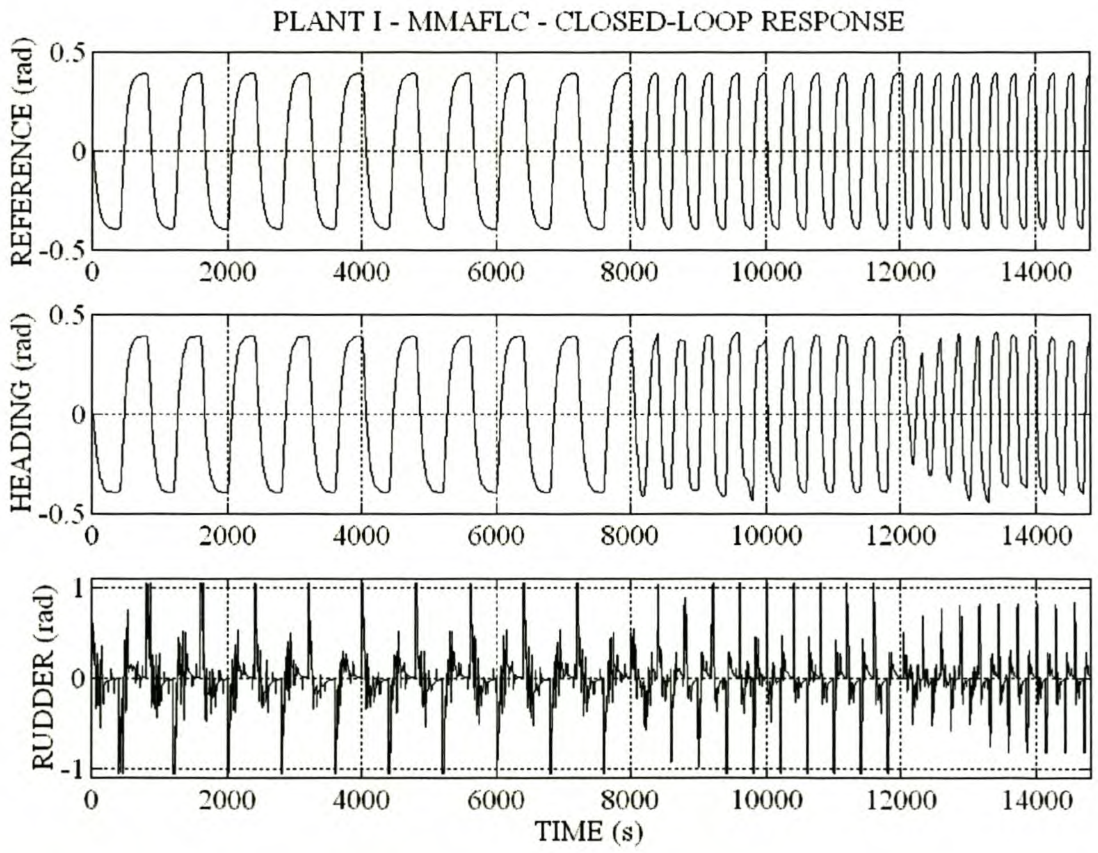

Figure 25 Plant I – MMAFLC – Closed-loop response.

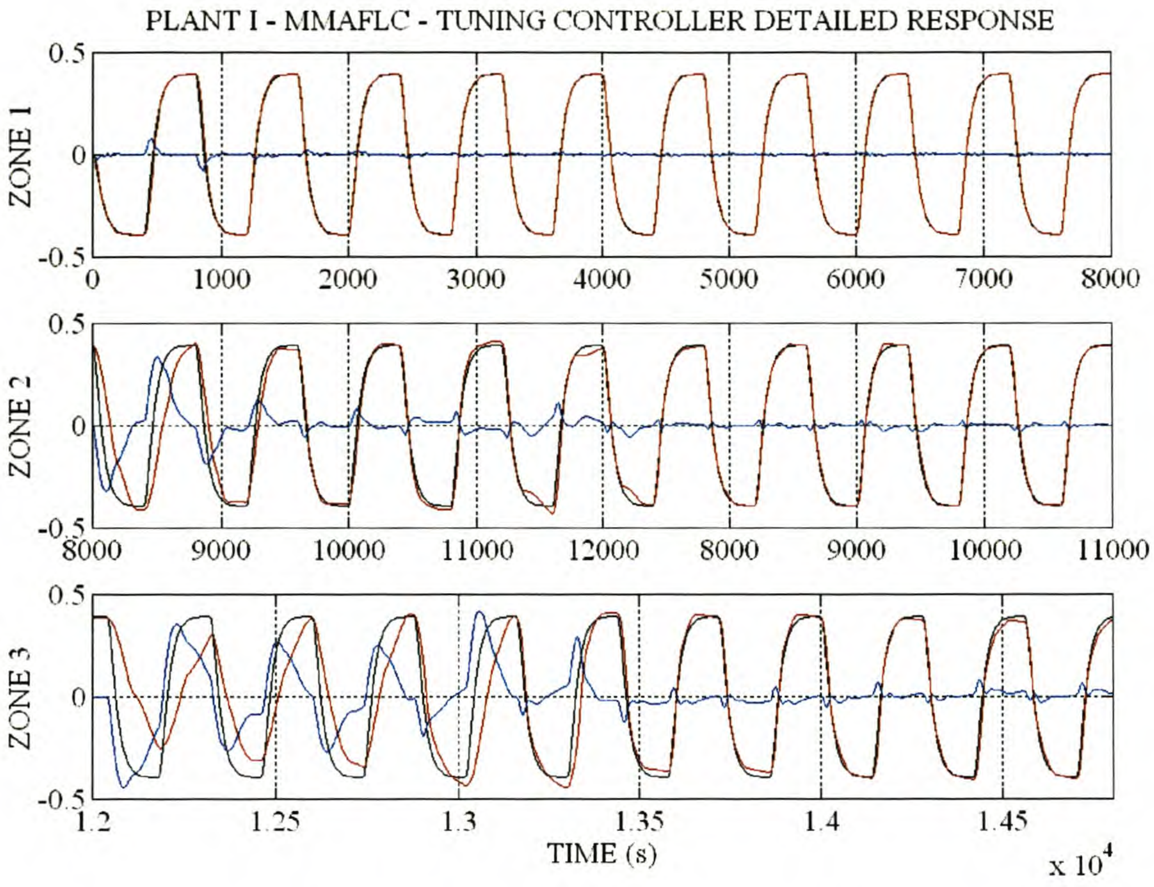

Figure 26 Plant I – MMAFLC – Tuning controller detailed response.

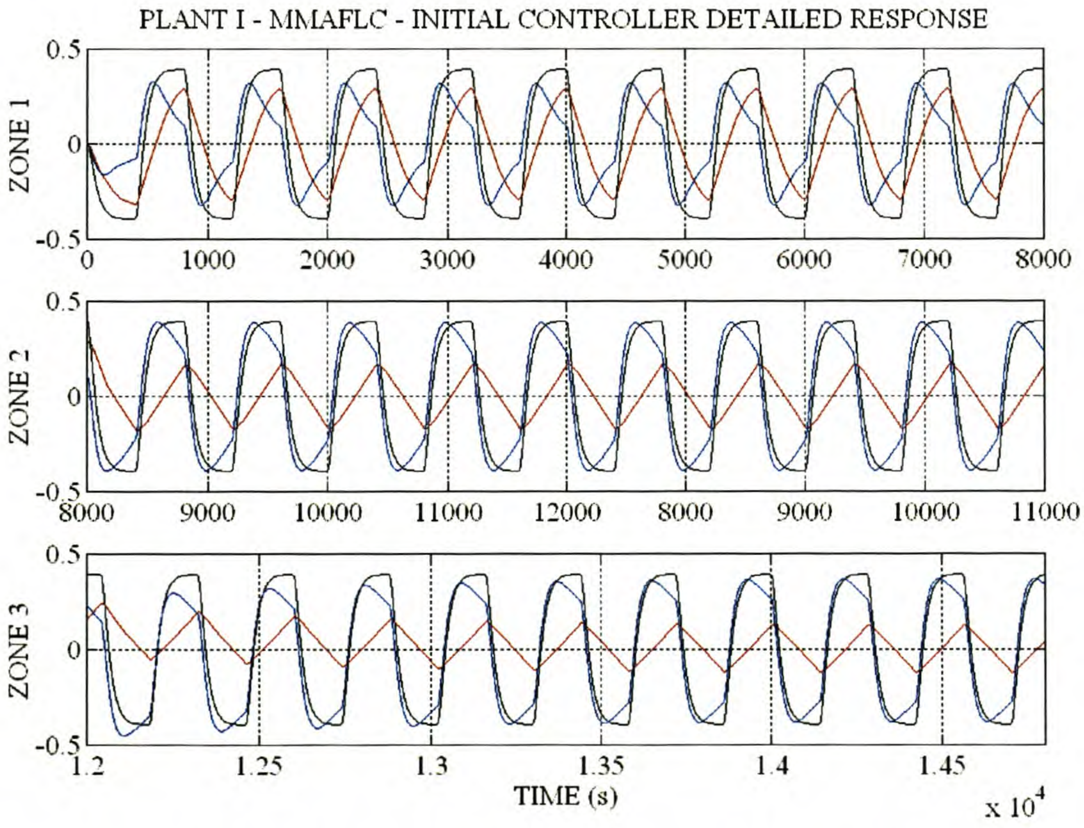

Figure 27 Plant I – MMAFLC – Initial controller detailed response.

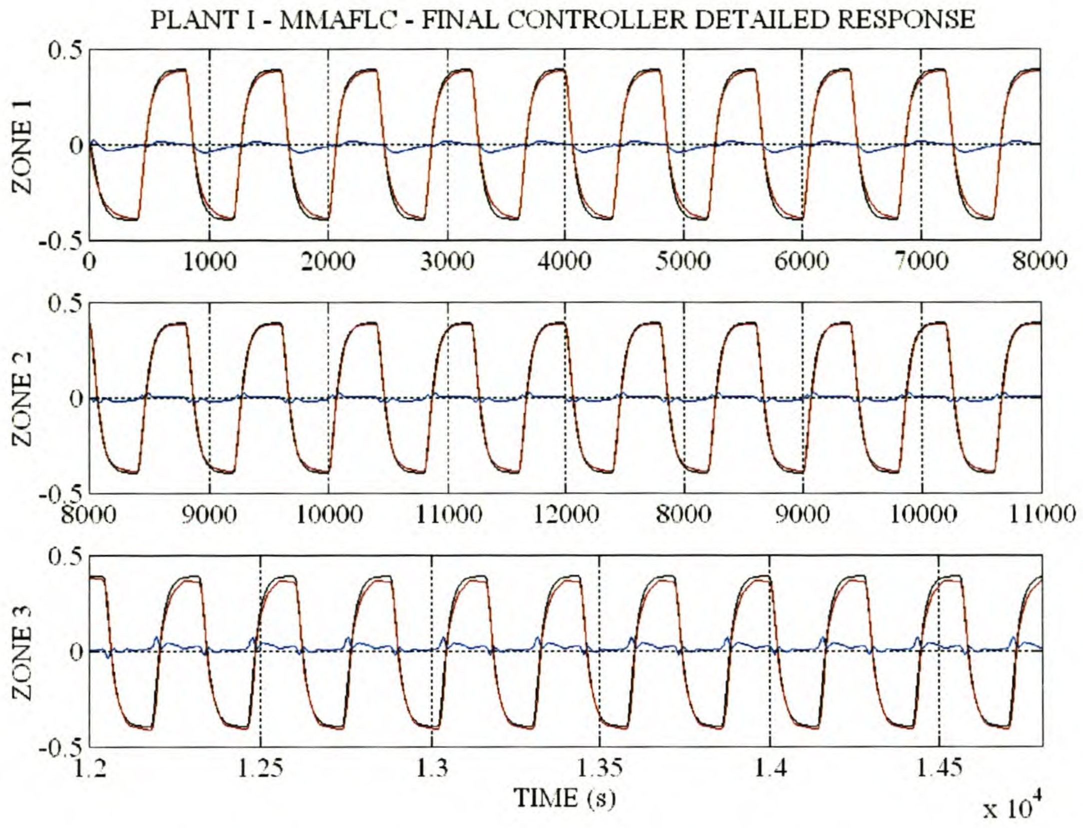

Figure 28 Plant I – MMAFLC – Final controller detailed response.

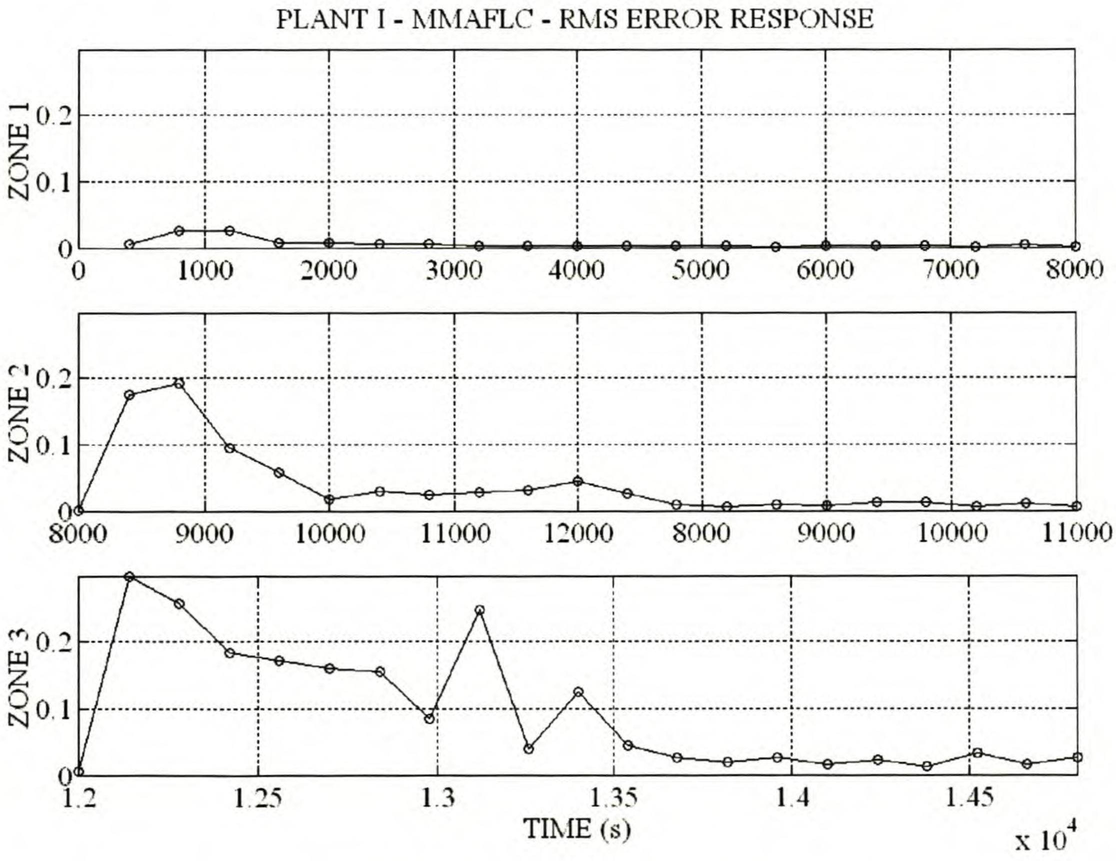

Figure 29 Plant I – MMAFLC – Response error RMS value for each step change.

4.1.4 PERFORMANCE EVALUATION AND COMPARISON

The non-adaptive SMFLC results in unacceptable performance (excessive overshoot) in domains with a lower bandwidth than where it has been tuned. The MMFLC achieves a uniform response in all the domains at the expense of increased reaction time for the same error input scaling.

The adaptive algorithm is successful in reducing the RMS values of the response error for both the SMAFLC and MMAFLC. The performance of the final MMAFLC controller obtained after tuning in all domains is far superior to that of the SMAFLC. When the system transfers between domains the SMAFLC tunes for the new domain and de-tunes for old domains. The SMAFLC would thus have to be re-tuned should the system go back into the old domain. The MMAFLC will require less, if any, re-tuning.

4.2 PLANT II: SECOND-ORDER PLANT WITH EXPONENTIAL NON-LINEARITY

The non-linear second-order plant presented in 4.2.1 does not correspond to a specific physical system, but is used to illustrate the SMFLC and MMFLC of a system with exponential non-linearity.

4.2.1 PLANT AND SIMULATION MODEL

The non-linear second-order plant is shown in Figure 30.

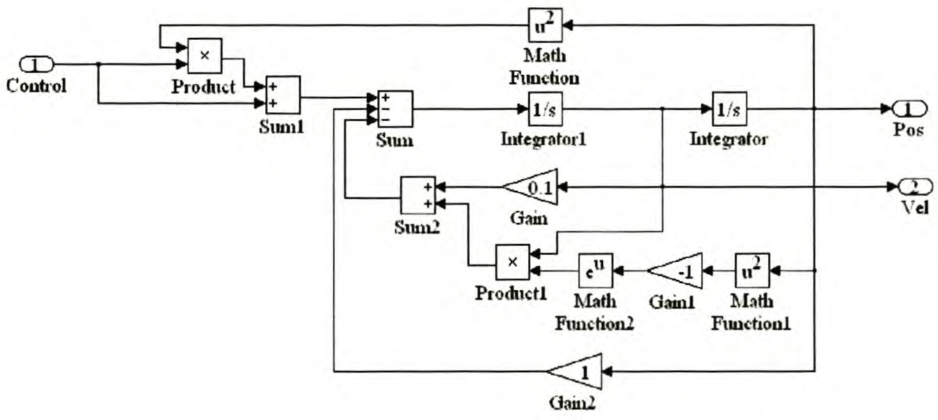

Figure 30 Plant II – Second-order plant with exponential non-linearity.

The velocity feedback gain and input gain are non-linear functions of the position. The system is described by the non-linear state space equation given in Equation 28.

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_1 - \left(e^{-x_1^2} + 0.1\right)x_2 + \left(1 + x_1^2\right)u$$

Equation 28

Using standard techniques [12], small signal linear models can be extracted at fixed operating points in the system domain. Equation 29 gives the small signal linear state space model.

$$\begin{bmatrix} \delta\dot{x}_1 \\ \delta\dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 + 2x_{10}x_{20}e^{-(x_{10})^2} + 2x_{10}u_0 & -0.1 - e^{-(x_{10})^2} \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 + (x_{10})^2 \end{bmatrix} \delta u$$
$$\delta y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix}$$

Equation 29

The fixed operating points are given by [x_{10} x_{20} u_0]. The steady-state relation between u_0 and x_{10} is determined by solving Equation 28 equals zero. This result is shown in Equation 30.

$$x_{10} = \frac{1/u_0 \pm \sqrt{\left(1/u_0\right)^2 - 4}}{2} \qquad \left(x_{20} = 0\right)$$

Equation 30

Real solution of Equation 30 requires that $u_0 \in (-0.5;0.5)$ with $x_{10} \in (-1;1)$. The small signal dynamic model properties extracted at four fixed working points (using Equation 29 and Equation 30) are listed in Table 5.

u_0	x_{10}	x_{20}	Poles	G	ω_n	δ
0	**0**	**0**	**$S_{1,2}$ = -0.50±0.8660j**	**1.0000**	**1.0000**	**0.5000**
±0.2353	**±0.25**	**0**	**$S_{1,2}$ = -0.44±0.8293j**	**1.2042**	**0.9393**	**0.4697**
±0.4	**±0.5**	**0**	**$S_{1,2}$ = -0.30±0.7141j**	**2.0833**	**0.7746**	**0.3873**
±0.48	**±0.75**	**0**	**$S_{1,2}$ = -0.14±0.5103j**	**5.5804**	**0.5292**	**0.2646**

Table 5 Plant II – Small signal linear model properties.

Due to the quadratic terms, the non-linearity is symmetrical for positive and negative x_1 and u. As the system is driven away from the equilibrium position, the small signal dynamics become slower, more oscillatory and the small signal gain increases.

The system open-loop step response is shown in Figure 31. The response to large ($\Delta u = \pm 0.95$) and small step changes ($\Delta u = \pm 0.5$) is shown. The response of the reference model used in paragraph 4.2.3 to the same driving signal is included to illustrate the system's non-linearity. The increased overshoot for the larger step changes (M_p = 18.746% vs. M_p = 15.584%) confirms the reduced system damping as x_1 increases. The sudden increase in steady state transfer gain (Equation 30) is also apparent. The steady-state output to $u_0 = 0.45$ is $x_{10} = 0.63$ and for $u_0 = 0.25$, $x_{10} = 0.27$.

As in the work on the cargo ship, the SMFLC and MMFLC are designed using the proposed automated design algorithm. These controllers are used as the initial conditions for the self-adaptive algorithm. Due to its strong influence on the small signal dynamics, x_1 is chosen as the state parameter for the MMFLC. Due to the symmetrical nature of the system non-linearities, the state information input needs only to evaluate the absolute value x_1. Due to the non-linear dependency of the plant dynamics on the system's states, it is not possible to investigate the performance of the MMFLC and the MMAFLC separately in the various sections of the non-linear state space, as was the case for the cargo ship.

The SIMuLINK S-Function implementation of the plant is given in Appendix IV.I.

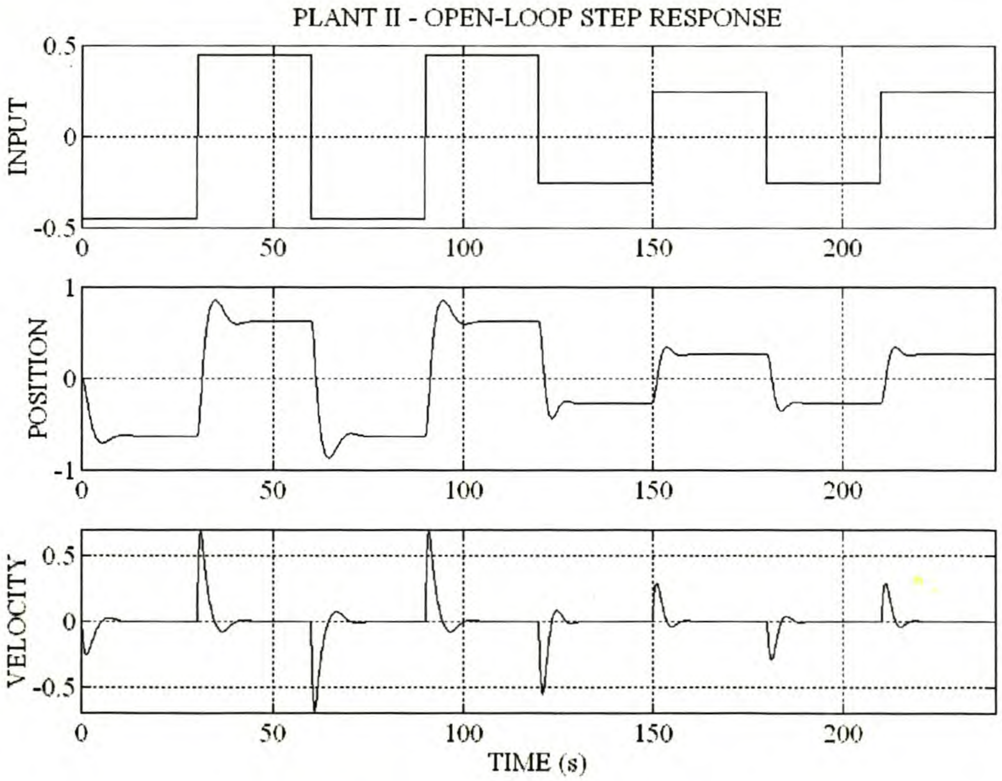

Figure 31 Plant II – Open-loop step response.

4.2.2 FUZZY LOGIC CONTROLLER DESIGN AND PERFORMANCE

The block diagram of the SMFLC and MMFLC is shown in Figure 32.

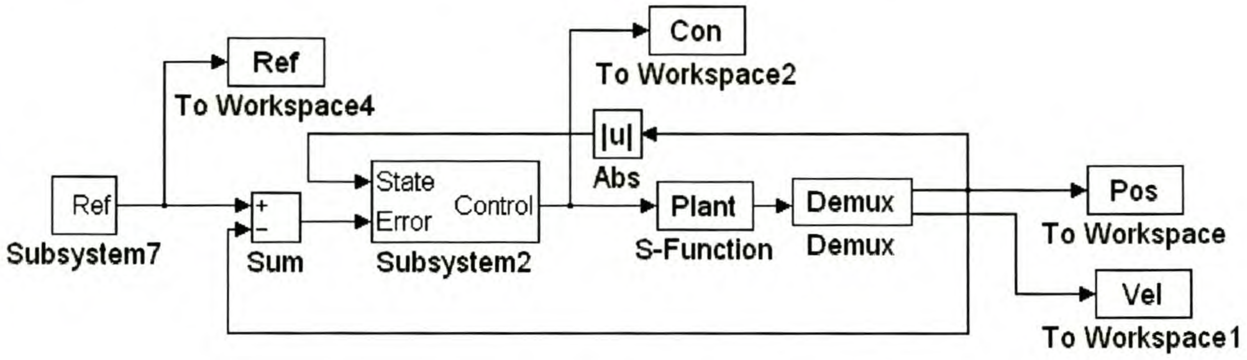

Figure 32 Plant II – Non-adaptive fuzzy logic control.

Since the plant is a type 0 system, a PID-type FLC is implemented. The inputs to the controllers are the state ($|x_1|$), error integral ($\int e.dt$), error (e) and error rate of change (de/dt). The closed-loop position response to a square wave reference signal at 0.0166Hz, with amplitude switching between 0.75 and 0.25, is used as test condition for both the SMFLC and MMFLC. The $T_{r\ (10\%-90\%)}$, M_p and E_{ss} are used as KPIs for controller evaluation.

The SMFLC and MMFLC have seven fuzzy sets defined by triangular membership functions placed on normalised universes of discourse for $\int e.dt$, e and de/dt. A single fuzzy set, defined by a trapezium membership function, was placed on the SMFLC state input spanning the universe of discourse defined by the interval [0,0.75]. The MMFLC state input have four fuzzy sets defined by triangular membership functions placed on the state universe of discourse. The centres of these membership functions coincide with the operating points defined in Table 5. The output universe of discourse was constrained to the interval [-0.5,0.5] with 21 fuzzy sets. This implies a complete rule-base with 343 rules for the SMFLC and 1372 rules for the MMFLC. The SMFLC and MMFLC structure is summarised in Table 6. The automated rule-base design algorithm was used to design the rule-base for the SMFLC and MMFLC.

Input	SMFLC			MMFLC		
	Minimum	Maximum	Sets	Minimum	Maximum	Sets
State	**0**	**0.75**	**1**	**0**	**0.75**	**4**
Integral	**-1**	**1**	**7**	**-1**	**1**	**7**
Error	**-1**	**1**	**7**	**-1**	**1**	**7**
Rate	**-1**	**1**	**7**	**-1**	**1**	**7**
Control	**0.5**	**0.5**	**21**	**0.5**	**0.5**	**21**

Table 6 Plant II – SMFLC and MMFLC structure summary.

The S_I and G_I vectors for the MMFLC was chosen based on the data obtained from the small signal linear models at the centres of the four state membership functions (0, 0.25, 0.5, 0.75), as summarised in Table 5. The G_I vector reflects the small signal gain and the S_I vector the undamped natural frequency. The vectors are given in Equation 31.

$$\begin{aligned} G_I &= [1 \quad 1.2 \quad 2.0 \quad 5.0] \\ S_I &= [1 \quad 0.95 \quad 0.8 \quad 0.6] \end{aligned}$$

Equation 31

The control input gains were determined experimentally to give a strongly damped, non-oscillatory response with the MMFLC. The gains used are $K_p = 40$, $K_d = 2.5$ and $K_i = 0.75$. The MATLAB® design files are given in Appendix IV.II for the SMFLC and in Appendix IV.III for the MMFLC.

The closed-loop response of the SMFLC and MMFLC is shown in Figure 33 and Figure 34. The KPIs for the SMFLC and MMFLC are compared in Table 7.

ΔRef	SMFLC			MMFLC		
	$\mathbf{T_{r\,(10\%-90\%)}}$	$\mathbf{M_p}$	$\mathbf{E_{ss}}$	$\mathbf{T_{r\,(10\%-90\%)}}$	$\mathbf{M_p}$	$\mathbf{E_{ss}}$
±1.5	**4.7s**	**1.76%**	**0.0%**	**9.2s**	**0.0%**	**0.90%**
±0.5	**2.7s**	**2.59%**	**1.16%**	**3.3s**	**0.25%**	**2.24%**

Table 7 Plant II – Performance comparison of SMFLC vs MMFLC.

The $T_{r\ (10\%\ -\ 90\%)}$ of the MMFLC to large step responses is almost double that of the SMFLC. The faster response of the SMFLC comes at the expense of excessive overshoot compared to the MMFLC. The slower response of the MMFLC is due to the increased damping and reduced gain of the rule-base in the domains of larger position. The MMFLC control signal does not show the steady state oscillations of the SMFLC (Figure 33 b vs Figure 34 b). The low pass filtering characteristics of the plant suppress the limit cycle amplitude of the SMFLC significantly, as shown in Figure 33 a. The significant decrease in the overshoot of the MMFLC compared to that of the SMFLC comes at the expense of the slightly longer response times and larger steady-state errors. The longer response time is caused by the general decrease in controller gain in some of the non-linear domains. The response time of the MMFLC could be improved by increasing the controller input gains and manipulating the speed and gain information vectors to limit overshoot and oscillations.

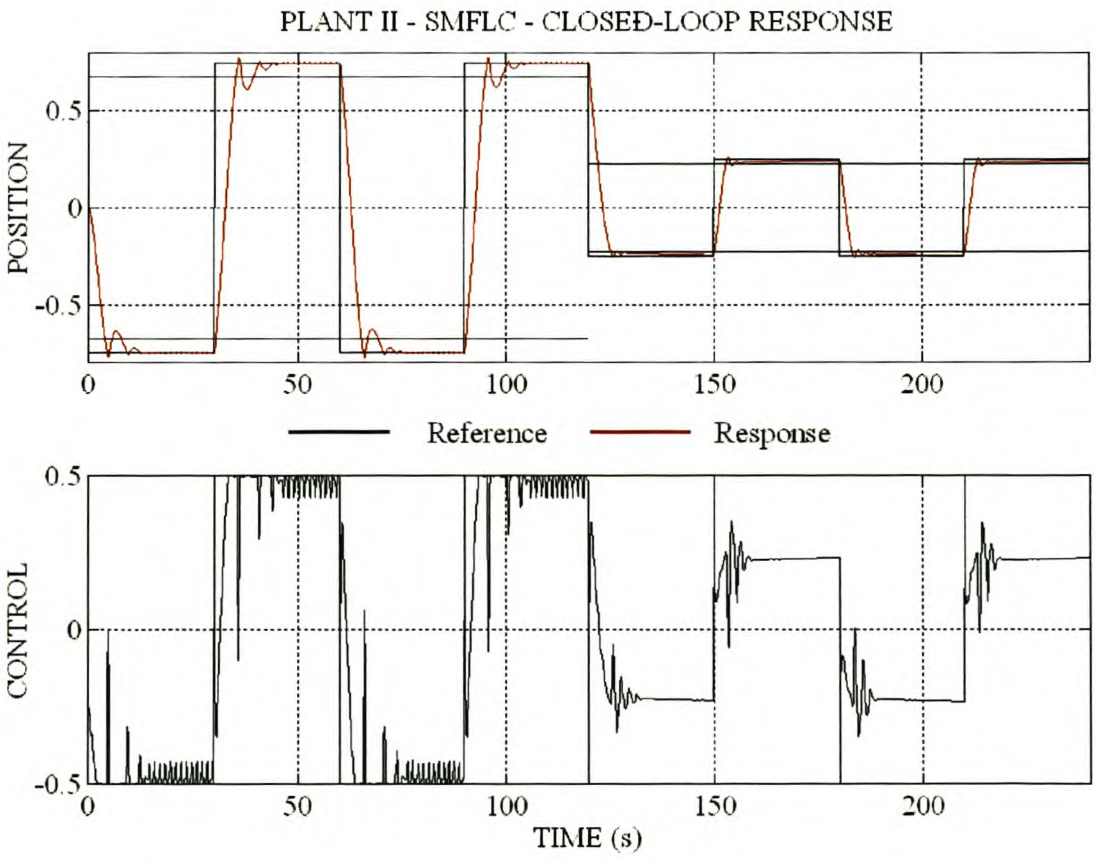

Figure 33 Plant II – SMFLC – Closed-loop response.

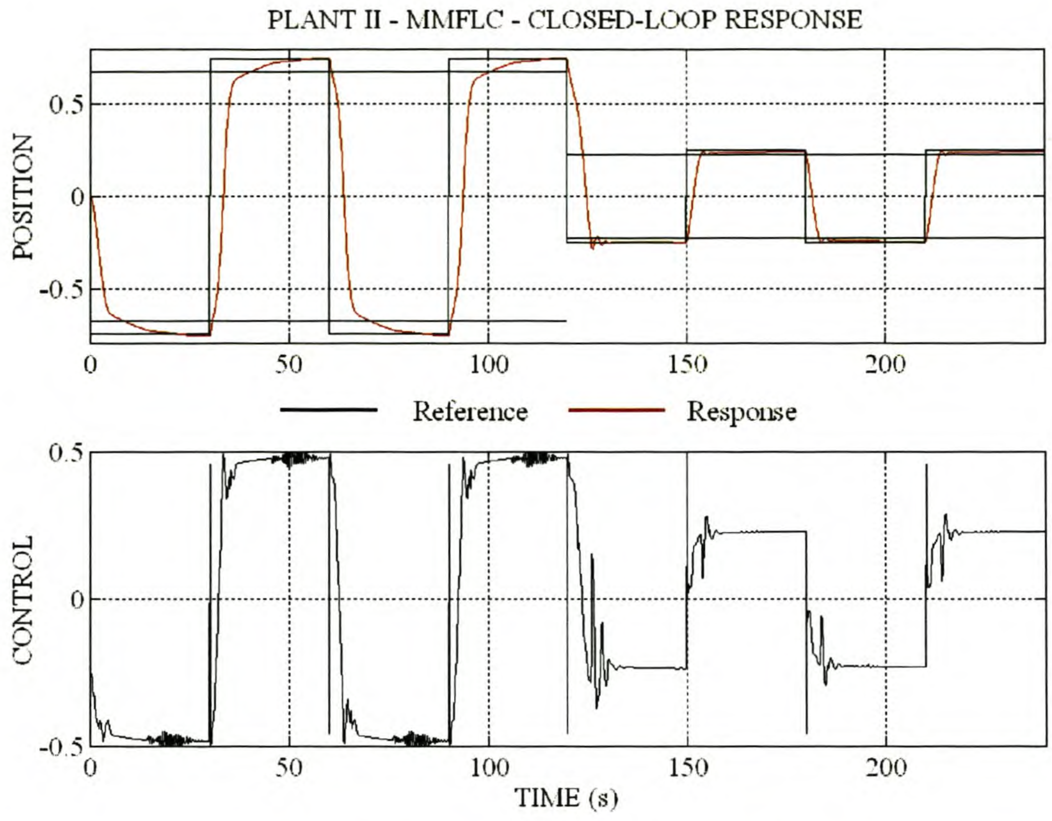

Figure 34 Plant II – MMFLC – Closed-loop response.

4.2.3 SELF-ADAPTIVE CONTROLLER DESIGN AND PERFORMANCE

The block diagram of the SMAFLC and MMAFLC is shown in Figure 35.

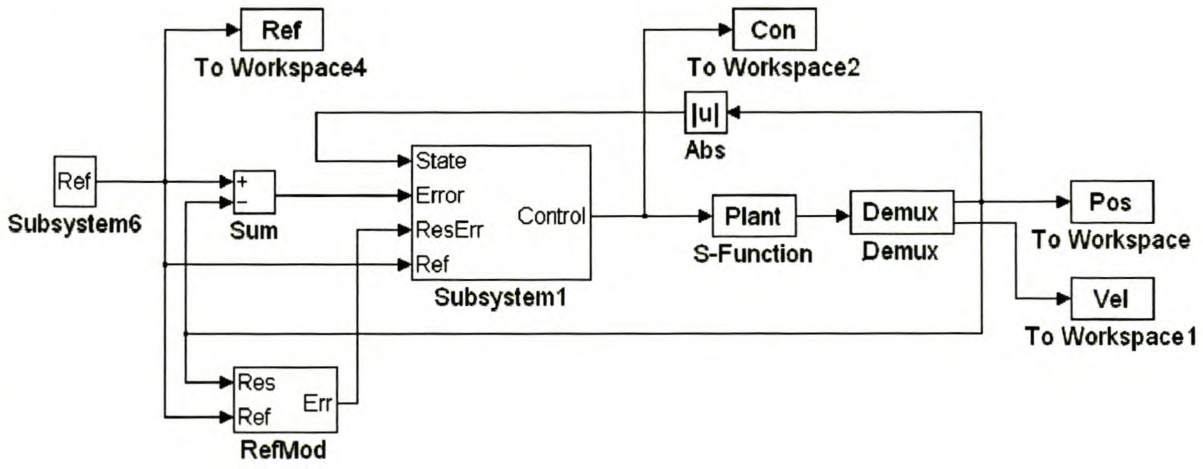

Figure 35 Plant II – Adaptive fuzzy logic control.

A single reference model was chosen for all the domains. The structures of these controllers are identical to those given in paragraph 4.2.2. The reference model used (Equation 32) is a critically damped system with a bandwidth (0.5rad/s) equal to that of the slowest part of the system. The

SAFLC would thus give a uniform response over the non-linear domain by decreasing the overshoot and bandwidth.

$$G_{\mathrm{Re}f}(s) = \frac{0.25}{s^2 + s + 0.25}$$

Equation 32

The controller input gains $K_i = 0.25$, $K_p = 0.5$, and $K_d = 5$ were selected to map the maximum encountered values to the normalised universe of discourse. The tuner gain parameters were determined experimentally as $K_{REp} = 0.05$ and $K_{REd} = 5$. The closed-loop position response to a square wave reference signal at 0.0166Hz, with amplitude switching between 0.75 and 0.25, was used as test conditions for both the SMAFLC and MMAFLC.

The closed-loop performance of the SMAFLC and MMAFLC is shown in Figure 36 and Figure 41. The simulation period can be divided into three time zones of 900s each. In zone 1 (0s – 900s) and zone 3 (1800s-2700s) the reference input amplitude is 0.75, while the reference input amplitude is 0.25 in zone 2 (900s – 1800s). The detailed responses of the SMAFLC and MMAFLC are shown in Figure 37 and Figure 42. The response of the initial controller is shown in Figure 38 and Figure 43, with the final controller response shown in Figure 39 and Figure 44. The RMS value of the response error for each step during the tuning operation is shown in Figure 40 for the SMAFLC and in Figure 45 for the MMAFLC. The RMS values of the response error for the SMAFLC and the MMAFLC are compared in Table 8.

Zone	SMAFLC			MMAFLC		
	Initial	Tuning	Final	Initial	Tuning	Final
1 (0s – 900s)	**0.3107**	**0.0798**	**0.0169**	**0.3759**	**0.1257**	**0.0444**
2 (900s – 1800s)	**0.1025**	**0.0338**	**0.0766**	**0.0968**	**0.0234**	**0.0349**
3 (1800s – 2700s)	**0.3114**	**0.0619**	**0.0154**	**0.3728**	**0.0547**	**0.0447**

Table 8 Plant II – RMS error comparison of SMAFLC vs MMAFLC.

In zone 1 the SMAFLC response (Figure 37 a) improved the initial controller response (Figure 38 a) to almost perfect tracking with a reduction of 74% in the RMS value of the response error. The tuning process converged after 12 step changes (Figure 40 a). The final controller response (Figure 39 a) showed a reduction in the RMS value of the response error of 94%. In zone 2 the SMAFLC response (Figure 37 b) showed good tracking compared to the initial controller response (Figure 38 b) with a reduction in the RMS value of the response error of 66%. The tuning operation shows convergence after one step change (Figure 40b). The final controller response (Figure 39 b) shows an improvement in the RMS value of the response error of only 25%, due to the limit cycle oscillations. In zone 3 the SMAFLC response (Figure 37 c) reduced the RMS value of the response error by 80% compared to the initial controller response (Figure 38 c). The tuning operation converged to an oscillating rule-base after 14 step changes (Figure 40 c). The final controller response (Figure 39 c) showed a reduction of 95% in the RMS value of the response error and delivers almost perfect tracking.

In zone 1 the MMAFLC response (Figure 42 a) improved the initial controller response (Figure 43 a) with a reduction of 66% in the RMS value of the response error. The tuning process converged after 11 step changes (Figure 45 a). The final controller response (Figure 44 a) showed very good tracking with a reduction in the RMS value of the response error of 88%. In zone 2 the MMAFLC response (Figure 42 b) showed good tracking compared to the initial controller response (Figure 43 b) with a reduction in the RMS value of the response error of 75%. The tuning operation shows convergence after one step change (Figure 45 b). The final controller response (Figure 44 b) shows an improvement in the RMS value of the response error of 63% with a reduced and decaying oscillation compared to the SMAFLC. In zone 3 the MMAFLC response (Figure 42 c) reduced the RMS value of the response error by 85% compared to the initial controller response (Figure 43 c). The tuning operation converged after 14 step changes (Figure 45 c). The final controller response (Figure 44 c) showed a reduction of 88% in the RMS value of the response error.

The final SMAFLC performed better in zones 1 and 3 than the final MMAFLC. The overall performance of the final MMAFLC is better in terms of the final 11% reduction of the RMS value of the response error in all three zones (0.08 for the SMAFLC vs 0.07 for the MMAFLC). The larger RMS values of the final MMAFLC response error are due to the initial peak in the response error.

4.2.4 PERFORMANCE EVALUATION AND COMPARISON

The non-adaptive MMFLC responds more slowly, but with less overshoot than the SMFLC. This is to be expected due to the proportional gain and increased damping realised by the automated design algorithm. The self-adaptive MMFLC forces the system to exhibit a uniform response over the non-linear domain investigated. The amount of de-tuning caused by overlap into the different domains is markedly less with the MMAFLC than with the SMAFLC.

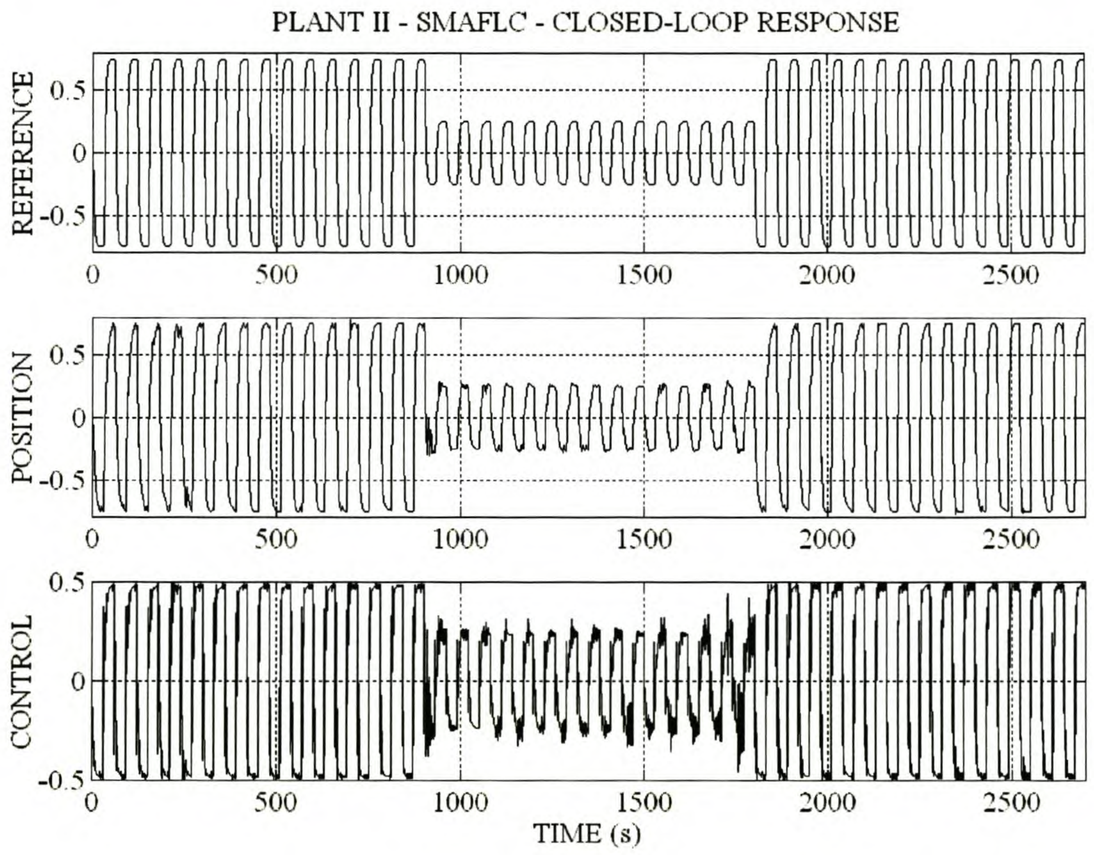

Figure 36 Plant II – SMAFLC – Closed-loop response.

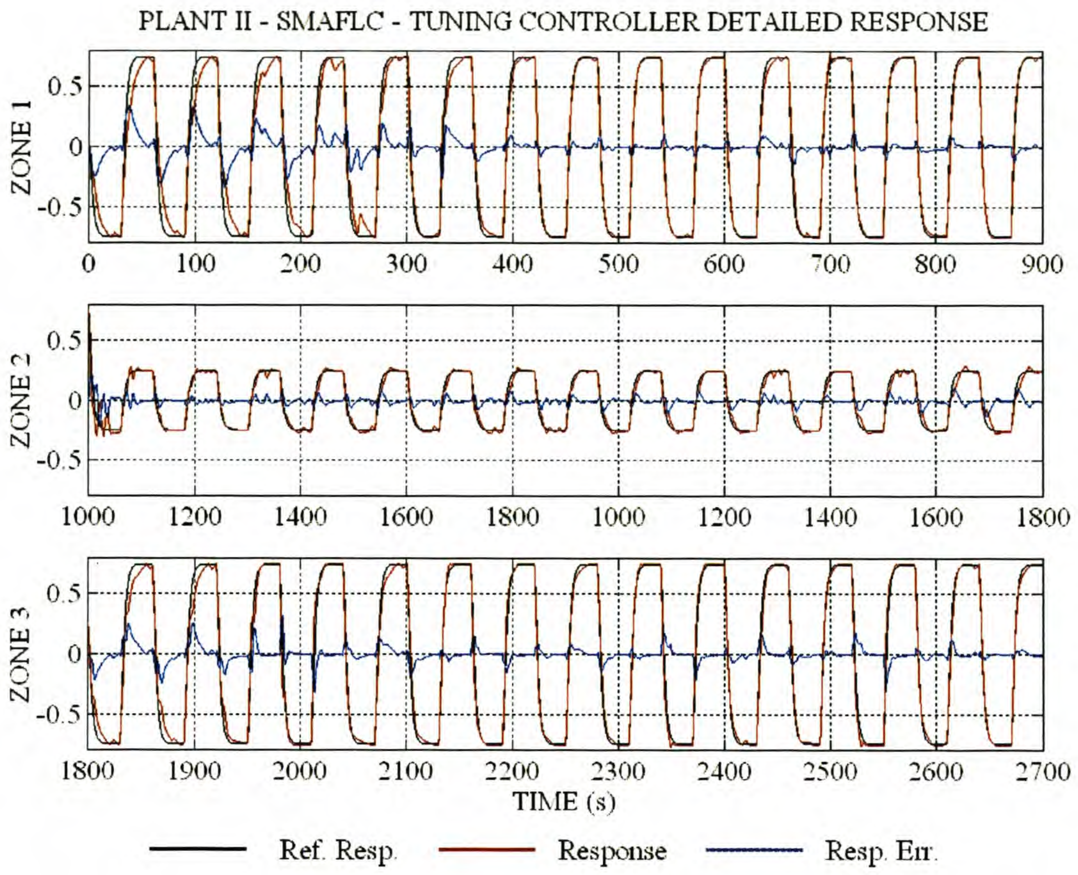

Figure 37 Plant II – SMAFLC – Tuning controller detailed response.

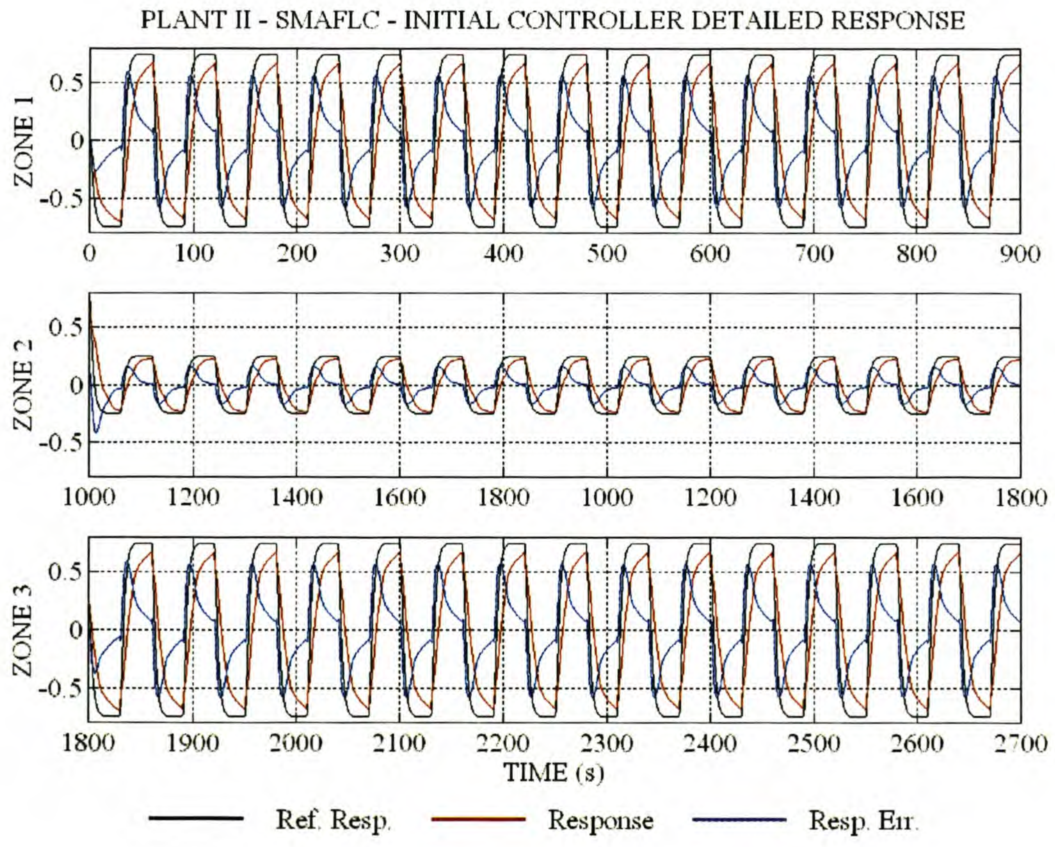

Figure 38 Plant II – SMAFLC –Initial controller detailed response.

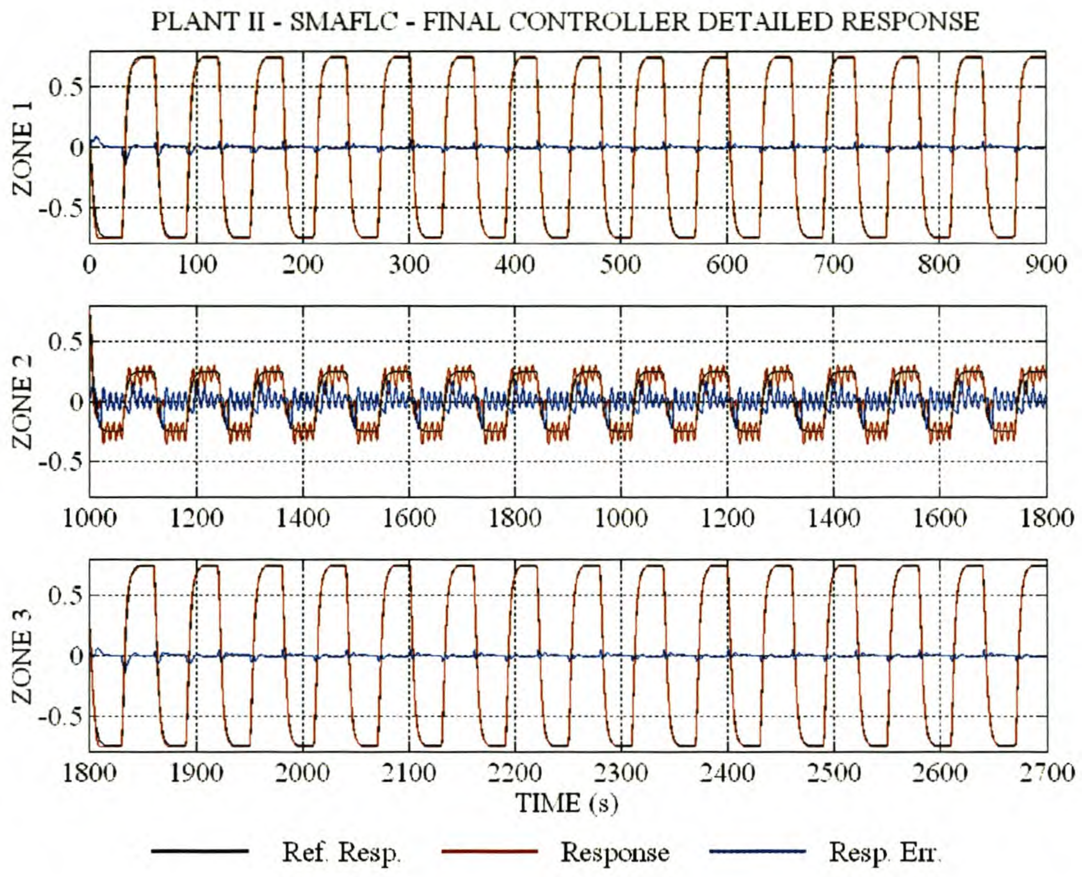

Figure 39 Plant II – SMAFLC – Final controller detailed response.

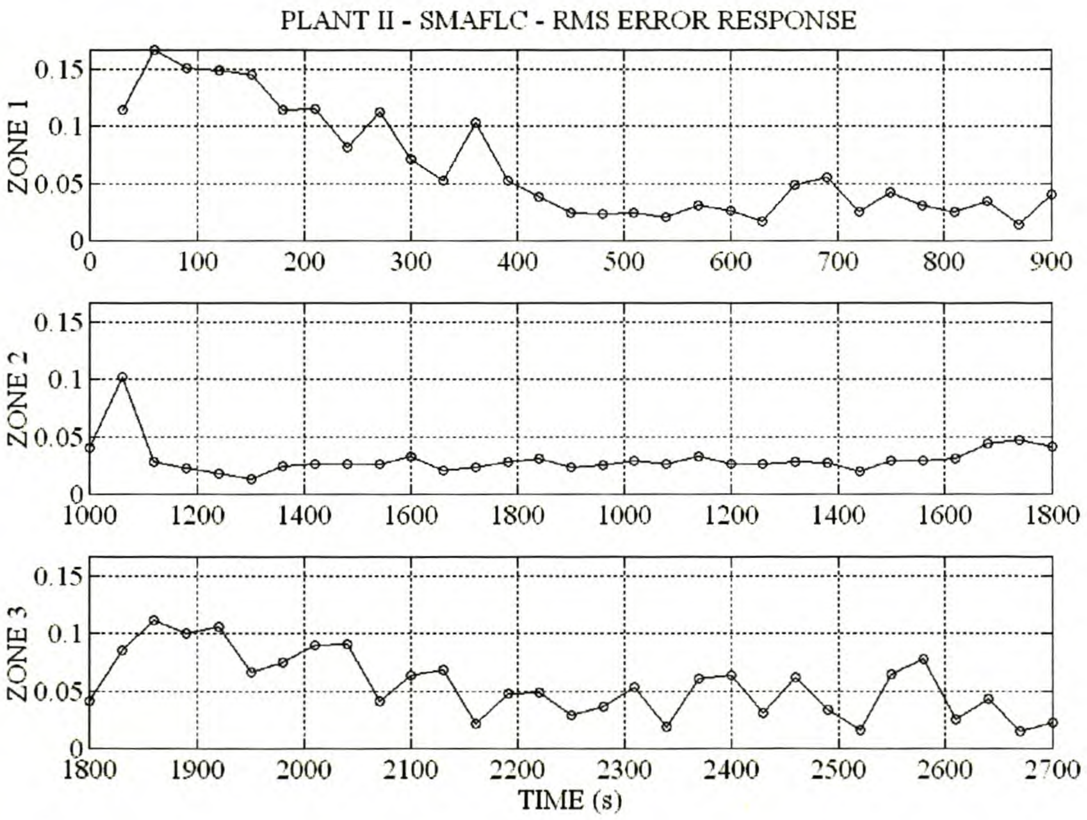

Figure 40 Plant II – SMAFLC – Response error RMS value for each step change.

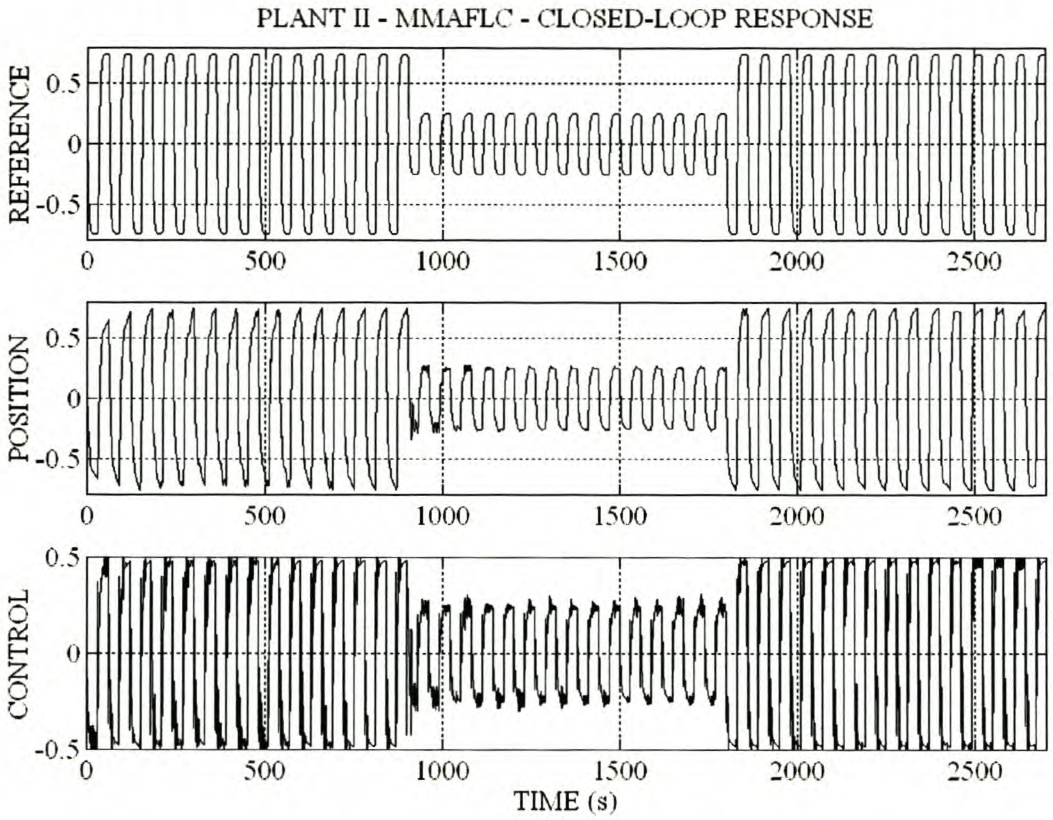

Figure 41 Plant II – MMAFLC – Closed-loop response.

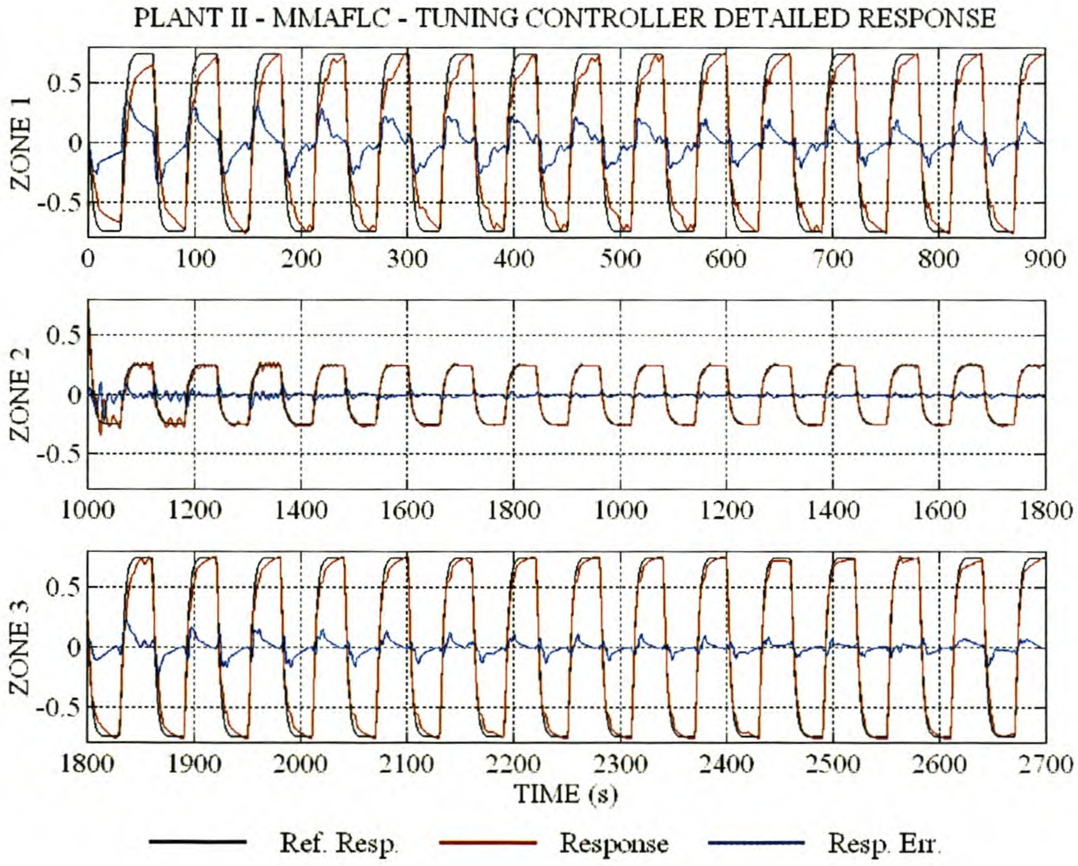

Figure 42 Plant II – MMAFLC – Tuning controller detailed response.

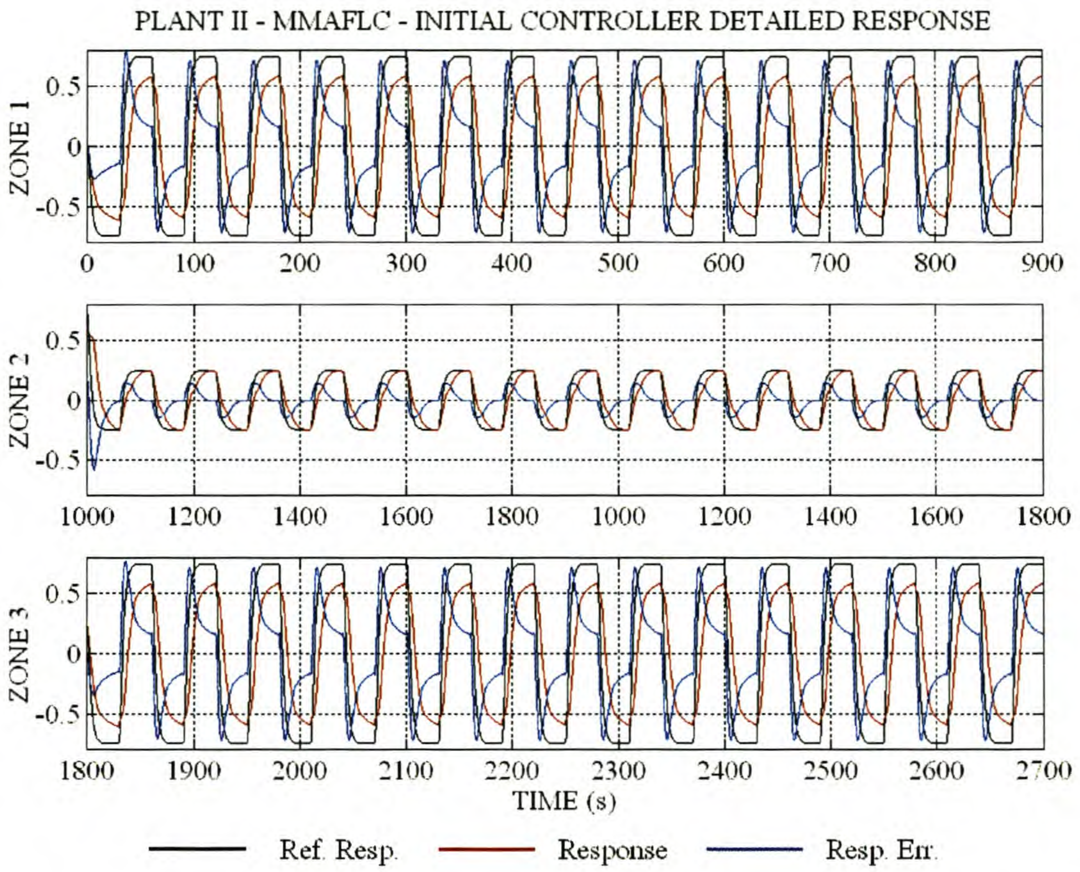

Figure 43 Plant II – MMAFLC – Initial controller detailed response.

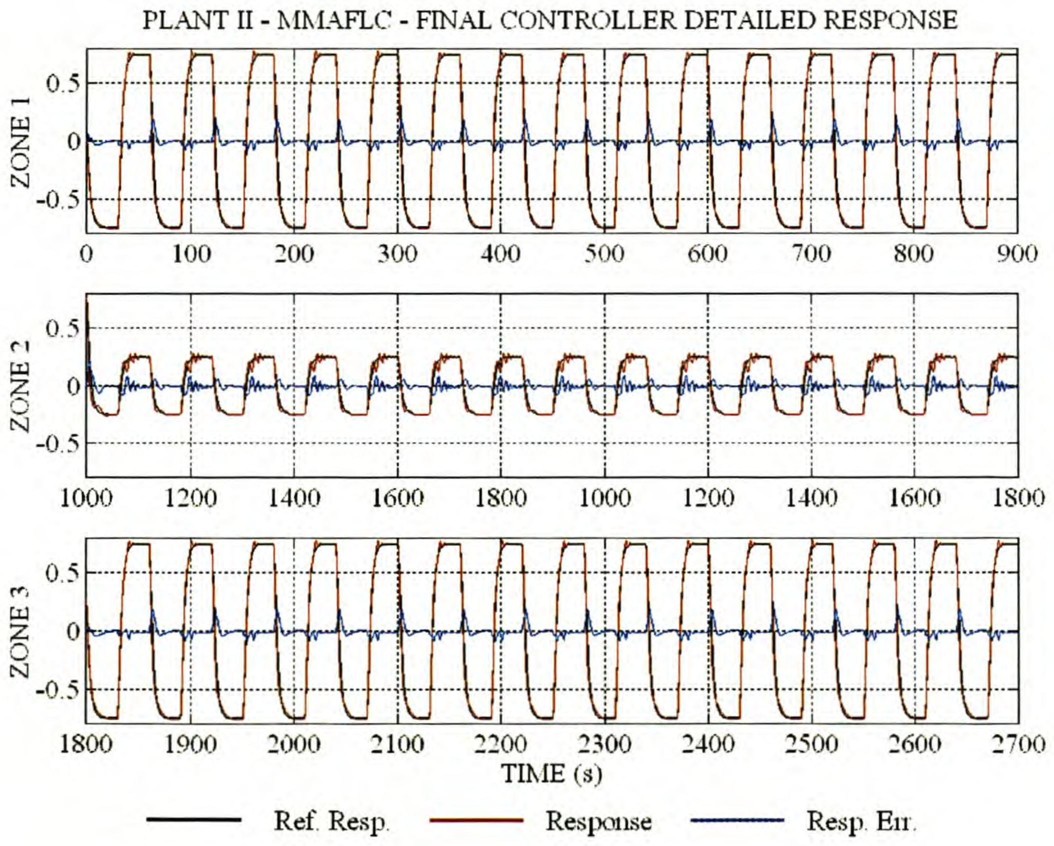

Figure 44 Plant II – MMAFLC – Final controller detailed response.

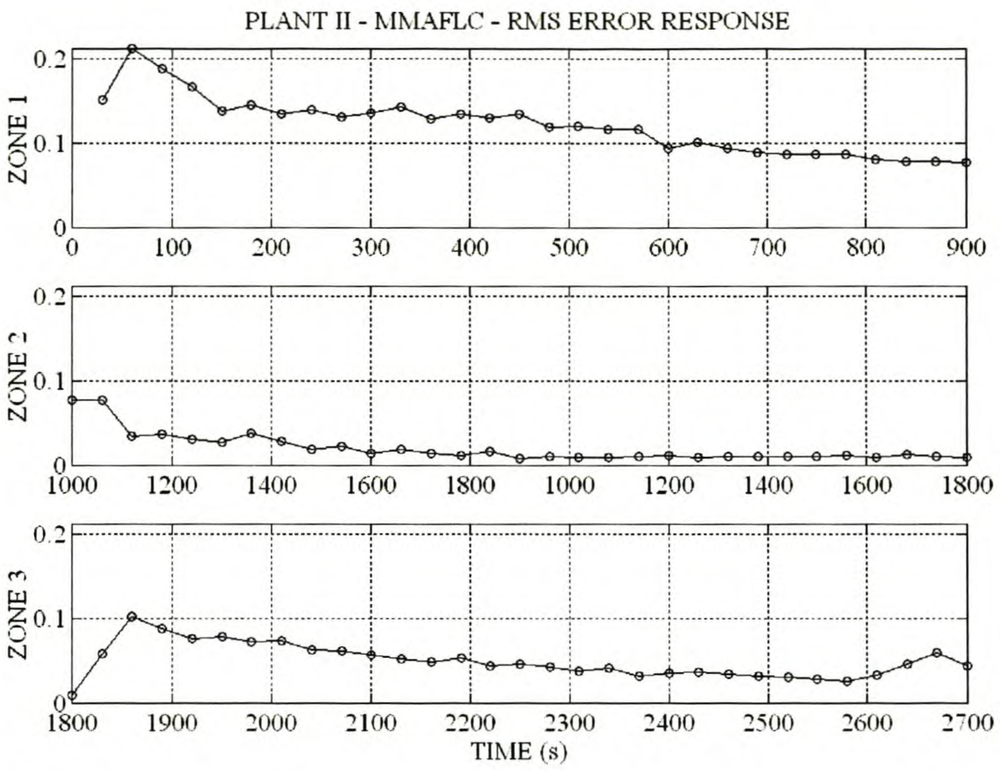

Figure 45 Plant II – MMAFLC – Response error RMS value for each step change.

4.3 Plant III: Second-order plant with non-linear velocity feedback

The second-order plant used in this section is non-linear in the velocity feedback coefficient. The non-linearity is similar to the non-linear drag force coefficient encountered in aircraft flight dynamics.

4.3.1 Plant and simulation model

The second-order plant with non-linear velocity feedback is shown in Figure 46.

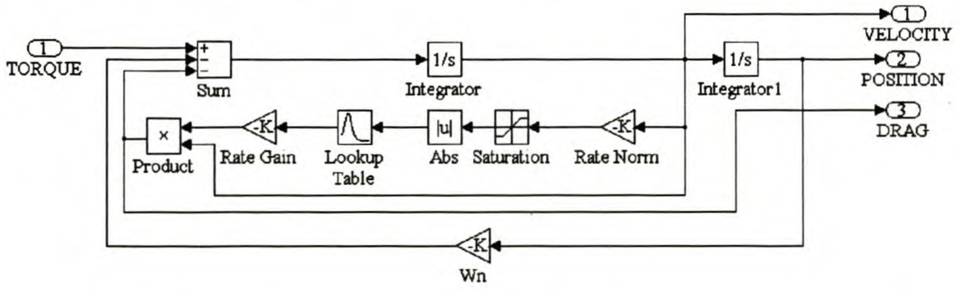

Figure 46 Plant III – Second-order plant with non-linear velocity feedback.

The non-linear velocity feedback gain, or drag force coefficient, is implemented as a lookup table and is shown in Figure 47.

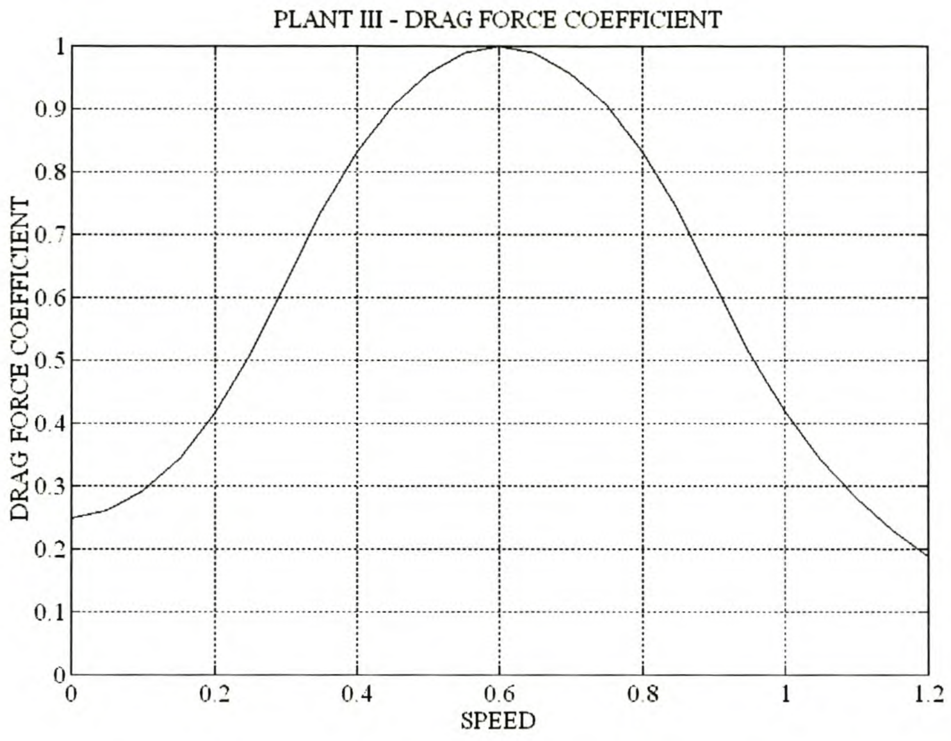

Figure 47 Plant III- Drag force coefficient as a function of velocity.

The drag force increases non-linearly from 0.25 at 0 to reach a maximum at 0.6. For velocities higher than 0.6, the drag force coefficient decrease non-linearly. The saturation element (Figure 46) ensures that the drag force coefficient remains at a constant value for velocities higher than 1.8.

The non-linear model of the plant is shown in Equation 33.

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -x_1 - 5 \times f(x_2) \times x_2 + T$$

$$f(x_2) = \begin{cases} 1.5(1.667 \times x_2)^2 + 0.25 & 0 \le |x_2| < 0.3 \\ -1.5(1.667 \times x_2)^2 + 3|x_2| - 0.5 & 0.3 \le |x_2| < 0.6 \\ 22.87e^{-2.4|1.667 \times x_2|} & 0.6 \le |x_2| \end{cases}$$

Equation 33

The determination of linear small signal models at steady-state operating points is straightforward but tedious, and will not be presented here [12]. In this work small signal models are derived at four values of velocity. The fixed work point is given by [x_{10} x_{20} u_0]. The pole positions, damping ratio (ξ) and natural frequency (ω_n) of the small signal linear models are shown in Table 9.

u_0	x_{10}	x_{20}	Poles		G	ω_n	ξ
0	**0**	**0**	**-0.6250±0.7806j**		**1**	**1**	**0.6250**
0	**0**	**0.4**	**-0.1358**	**-7.3623**	**1**	**1**	**3.6811**
0	**0**	**0.8**	**2.0155**	**0.4981**	**1**	**1**	**-1.2250**
0	**0**	**1.2**	**3.2702**	**0.3058**	**1**	**1**	**-1.7825**

Table 9 Plant III- Small signal linear model properties.

The open-loop step response is shown in Figure 48. The destabilising effect of the non-linear drag force coefficient at high velocities is clearly seen as large overshoot and very lightly damped oscillations for large step changes. For smaller step changes, the velocity does not go beyond the critical point (0.6) and the step response is overdamped.

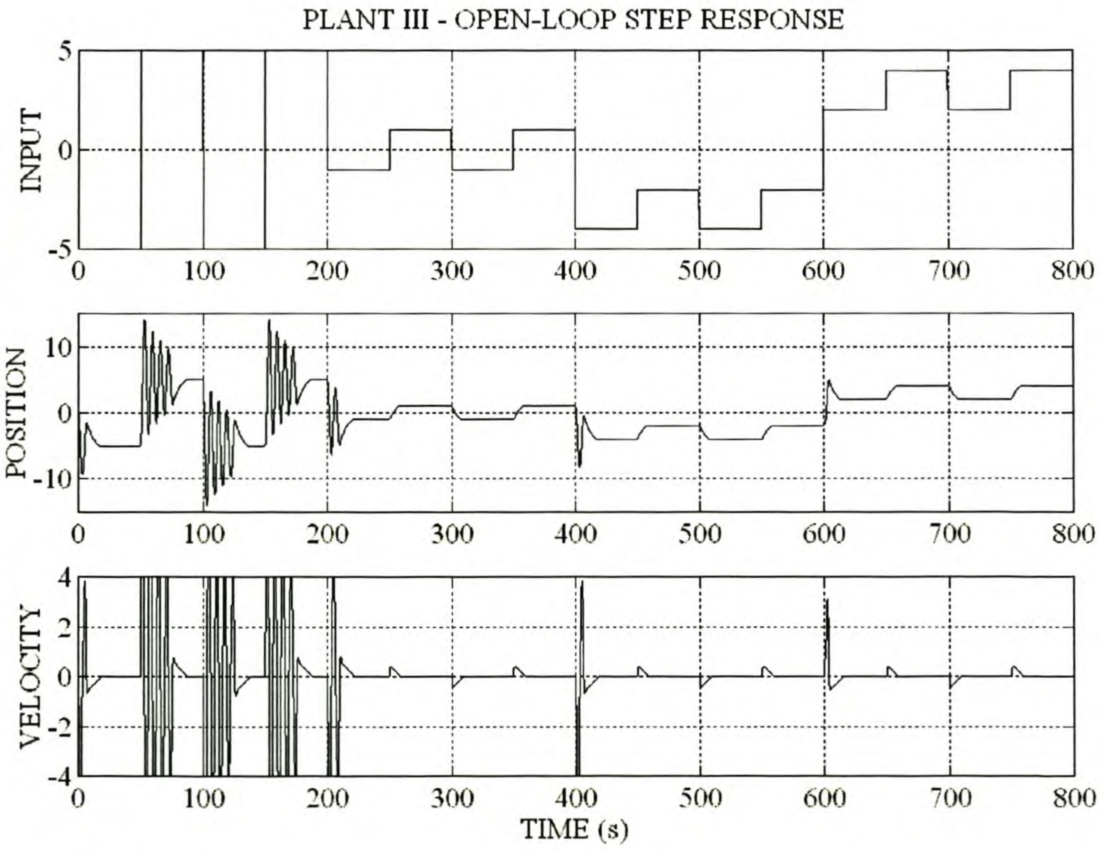

Figure 48 Plant III – Open-loop step response.

4.3.2 FUZZY LOGIC CONTROLLER DESIGN AND PERFORMANCE

The block diagram of the SMFLC and MMFLC is shown in Figure 49.

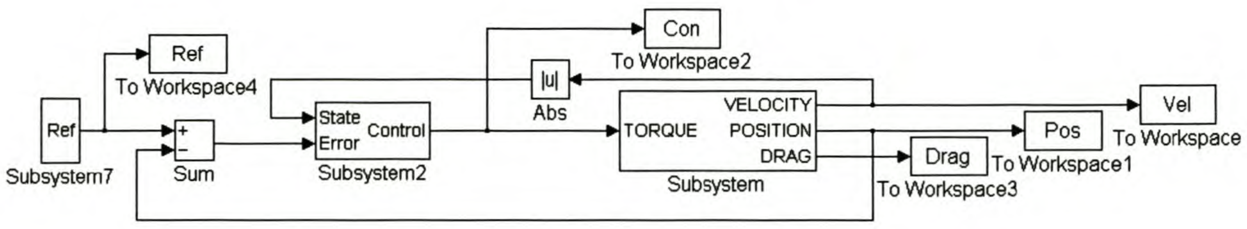

Figure 49 Plant III – Non-adaptive fuzzy logic control.

Since the plant is a type 0 system, a PID-type FLC is implemented. The inputs to the controllers are the state ($|x_2|$), error integral ($\int e.dt$), error (e) and error rate of change (de/dt). The closed-loop position response to a square wave reference signal at 0.02Hz, with amplitude switching between 5.0 and 1.0, was used as test condition for both the SMFLC and MMFLC. The $T_{r\ (10\%-90\%)}$, M_p and E_{ss} are used as KPIs for controller evaluation.

Input	SMFLC			MMFLC		
	Minimum	Maximum	Sets	Minimum	Maximum	Sets
State	**0**	**1.2**	**1**	**0**	**1.2**	**4**
Integral	**-1**	**1**	**9**	**-1**	**1**	**9**
Error	**-1**	**1**	**9**	**-1**	**1**	**9**
Rate	**-1**	**1**	**9**	**-1**	**1**	**9**
Control	**-5**	**5**	**21**	**-5**	**5**	**21**

Table 10 Plant III – SMFLC and MMFLC structure summary.

The SMFLC and MMFLC implement nine fuzzy sets for $\int e.dt$, e and de/dt. The MMFLC implements four fuzzy sets for the state input and the one for the SMFLC. The universes of discourse for $\int e.dt$, e and de/dt are the normalised interval [–1, 1]. The state input universe of discourse is the interval [0 1.2]. The four sets are defined by triangular membership functions with centers at 0, 0.4, 0.8 and 1.2. The SMFLC and MMFLC structure is summarised in Table 10.

The speed and gain information vectors for the MMFLC are taken from the information obtained from the small signal models defined in Table 9 (4.3.1). The gain in the various domains remains constant at one. The gain information is thus the same for all four domains. The problem is choosing the speed information for the unstable section of the state space. The initial choice was that of the slowest system pole. In the domains of high velocity the system is lightly damped, requiring increased derivative gain. The final values of G_I and S_I are given in Equation 34.

$$\begin{aligned} G_I &= [1 \quad 1 \quad 1 \quad 1] \\ S_I &= [0.62 \quad 0.13 \quad 0.1\times 0.49 \quad 0.1\times 0.3] \end{aligned}$$

Equation 34

The control output universe of discourse is constrained to the interval [-5, 5]. The SMFLC and MMFLC have 21 sets in the output universe of discourse. The control gains were determined experimentally as $K_i = 0.1$, $K_p = 2$ and $K_d = 0.2$. The MATLAB® design files are given in Appendix V.I for the SMFLC and in Appendix V.II for the MMFLC.

The closed-loop response of the SMFLC and MMFLC is shown in Figure 50 and Figure 51. The KPIs for the SMFLC and MMFLC are compared in Table 11.

ΔRef	SMFLC			MMFLC		
	$T_{r\,(10\%-90\%)}$	M_p	E_{ss}	$T_{r\,(10\%-90\%)}$	M_p	E_{ss}
±5.0	**2.4077s**	**20.1311%**	**4.7%**	**20.8332s**	**0.0%**	**0.0%**
±1.0	**7.0069s**	**26.8000%**	**0.0%**	**5.1805ss**	**43.45%**	**0.0%**

Table 11 Plant III – Performance comparison of SMFLC vs MMFLC

The SMFLC responded very quickly to a large step change. The light system damping at high velocities resulted in 20% overshoot. Although the MMFLC responded much more slowly (21s), the system showed no overshoot and a smaller steady-state error. Due to the oscillations of the SMFLC the settling time of the SMFLC is comparable to that of the MMFLC (Figure 50 a vs Figure 51 a). For small step changes, the rise time of the SMFLC and MMFLC differed by about 2s, while the MMFLC exhibited twice the overshoot of the SMFLC. Attempts to limit the overshoot of the MMFLC in this domain by manipulation of S_I failed due to the normalisation of S_I in the design algorithm.

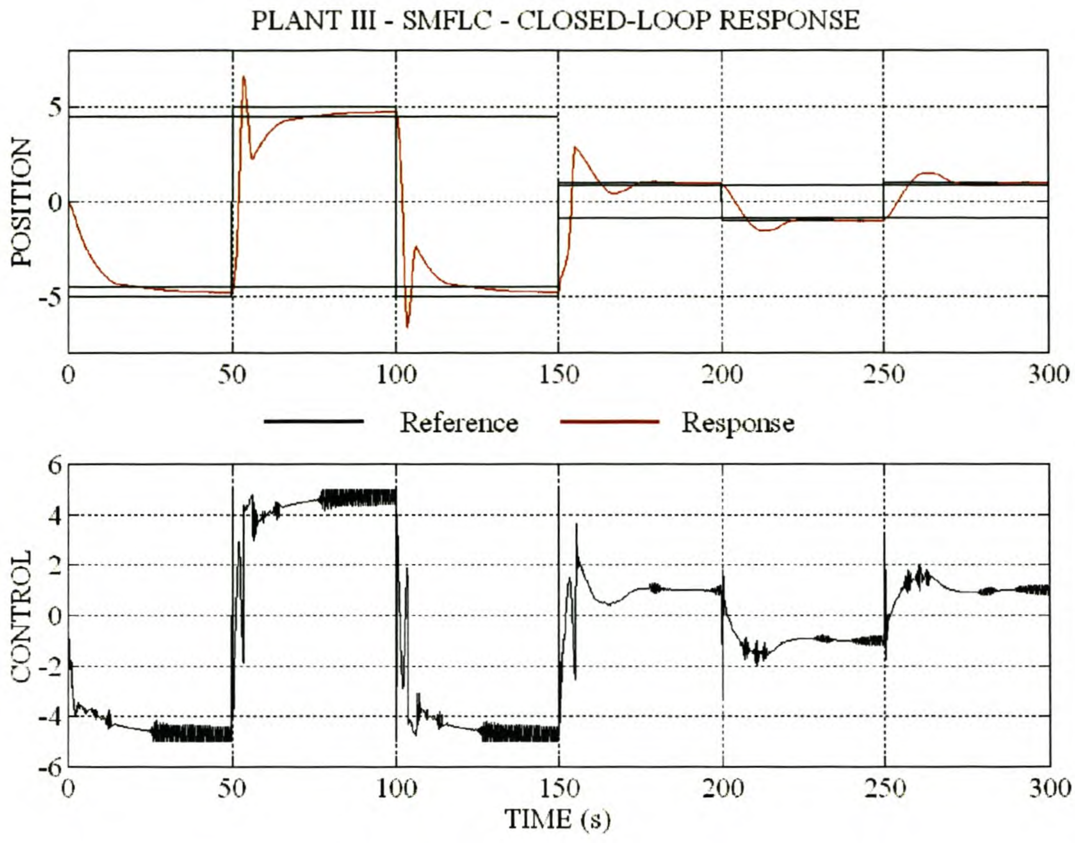

Figure 50 Plant III – SMFLC – Closed-loop response.

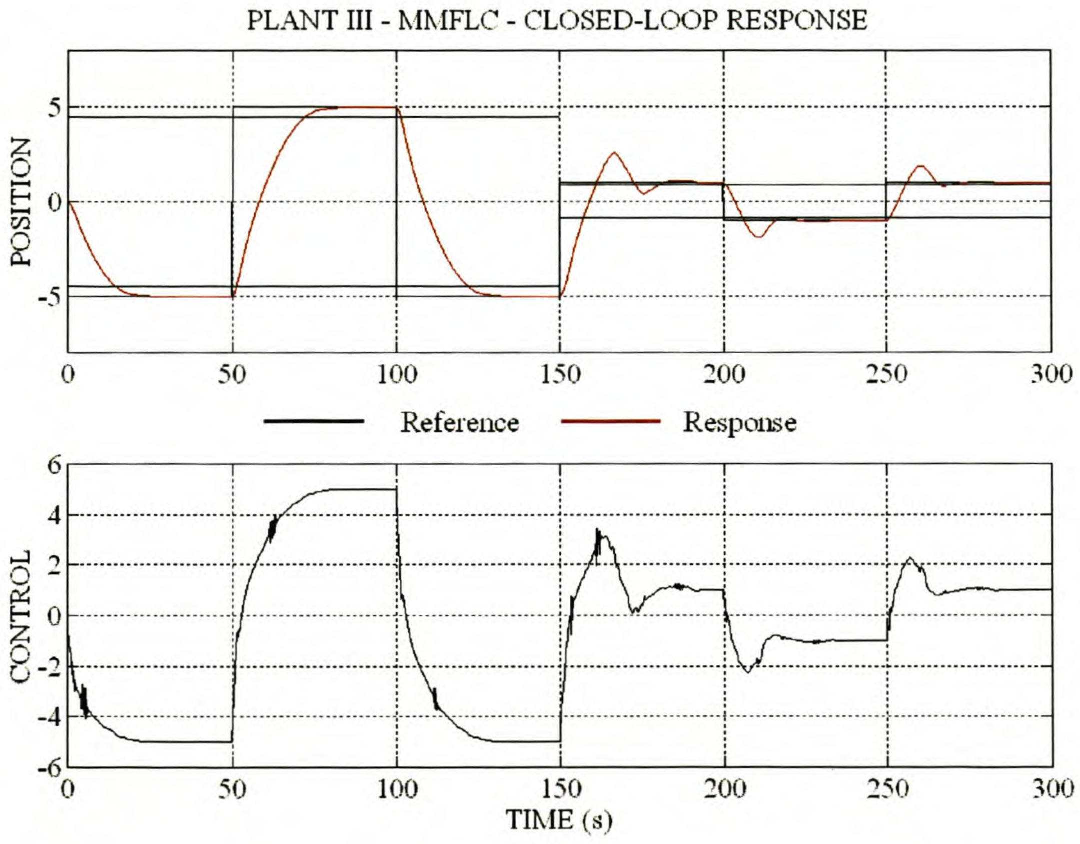

Figure 51 Plant III – MMFLC – Closed-loop response.

4.3.3 SELF-ADAPTIVE CONTROLLER DESIGN AND PERFORMANCE

The block diagram of the SMAFLC and MMAFLC is shown in Figure 52.

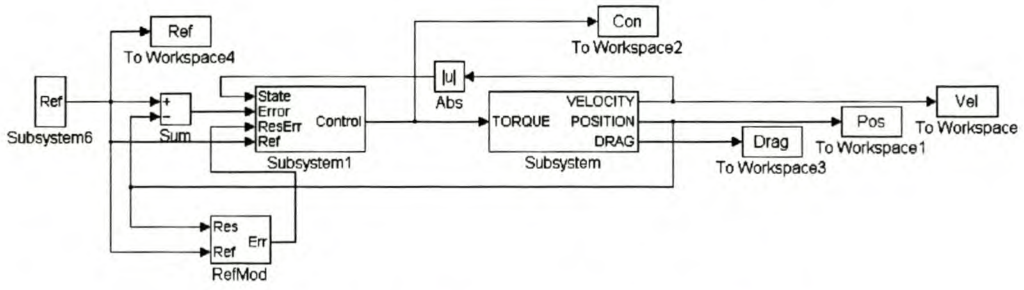

Figure 52 Plant III – Adaptive fuzzy logic control.

The structures of these controllers are identical to those given in paragraph 4.3.2. The reference model was taken as a critically damped second order system with ω_n = 1rad/s. The closed-loop performance specifications thus have the same bandwidth as the open-loop plant, but with improved damping. The controller input gains were chosen to map the maximum encountered values to the normalised universes of discourse. The tuner gain parameters were determined experimentally to give good tuning speed without starting limit cycle oscillations in the rule-base. The values of the SMAFLC and MMAFLC parameters are $K_i = 0.05$, $K_p = 0.1$, $K_d = 5$, $K_{REp} = 0.025$ and $K_{REd} = 4$. The closed-loop position response to a square wave reference signal at 0.02Hz, with amplitude switching between 5.0 and 1.0, was used as test condition for both the SMAFLC and MMAFLC.

The closed-loop performance of the SMAFLC and MMAFLC is shown in Figure 53 and Figure 58. The simulation period can be divided into three time zones of 600s each. In zone 1 (0s – 600s) and zone 3 (1200s-1800s) the reference input amplitude is 5.0, while the reference input amplitude is 1.0 in zone 2 (600s – 1200s). The detailed responses of the SMAFLC and MMAFLC are shown in Figure 54 and Figure 59. The response of the initial controller is shown in Figure 55 and Figure 60, with the final controller response shown in Figure 56 and Figure 61. The RMS value of the response error for each step during the tuning operation is shown in Figure 57 for the SMAFLC and in Figure 62 for the MMAFLC. The RMS values of the response error for the SMAFLC and the MMAFLC are compared in Table 12.

Zone	SMAFLC			MMAFLC		
	Initial	Tuning	Final	Initial	Tuning	Final
1 (0s-600s)	**2.1898**	**1.1858**	**1.0118**	**5.3939**	**1.6166**	**0.9946**
2 (600s-1200s)	**0.6010**	**0.3877**	**0.4642**	**0.9395**	**0.3057**	**0.4484**
3 (1200s-1800s)	**2.2200**	**1.0478**	**1.3207**	**5.4088**	**1.0448**	**1.0477**

Table 12 Plant III – RMS error comparison of SMAFLC vs MMAFLC.

The SMAFLC improved the initial response in zone 1 (Figure 54 a) significantly compared to the initial controller (Figure 55 a), with a reduction in the RMS value of the response error of 45%. After about five step changes the tuning operation converged (Figure 57 a). The final controller (Figure 56 a) responded smoothly with a reduction of 53% in the RMS value of the response error compared to the initial controller. The best reference model tracking occurred in zone 2 (Figure 54 b vs. Figure 55 b) with a reduction of 35% in the RMS value of the response error. Convergence required just one step change (Figure 57 b). The final controller performed poorly with a reduction of 22% in the RMS value of the response error (Figure 56 b). At the start of zone 3 the SMAFLC response (Figure 54 c) was degraded due to tuning in zone 2. The tuning operation required six step changes for convergence. The RMS value of the response error is reduced by 52% compared to that of the initial controller response. The final controller response (Figure 56 c) showed a reduction of 40% in the RMS value of the response error.

The MMAFLC reduced the response error at the end of zone 1 to an almost perfect response (Figure 59 a). The RMS value of the response error is 70% relative to the slow response of the initial controller in zone 1 (Figure 60 a). The tuning operation required six step changes to converge (Figure 62 a). The final controller response (Figure 61a) showed a reduction of 81% in the RMS value of the response error. The MMAFLC reduced the large overshoot of the initial controller in zone 2 (Figure 59 b vs Figure 60 b) with a reduction in the RMS value of the response error of 67%. The MMAFLC converged after one step change (Figure 62 b). The final controller response (Figure 56 b) was slightly degraded compared to the MMAFLC with a reduction in the

RMS value of the response error of only 52%. In zone 3 the MMAFLC (Figure 59 c) showed a reduction of 80% in the RMS value of the response error compared to the initial controller (Figure 60c). The final controller response (Figure 61 c) showed a reduction of 80% in the RMS value of the response error. Convergence required three step changes (Figure 62 c).

Overall, the final controller obtained from the MMAFLC performed better than the SMFLC with the RMS value of the response error of the final MMAFLC 12% lower than the final SMAFLC.

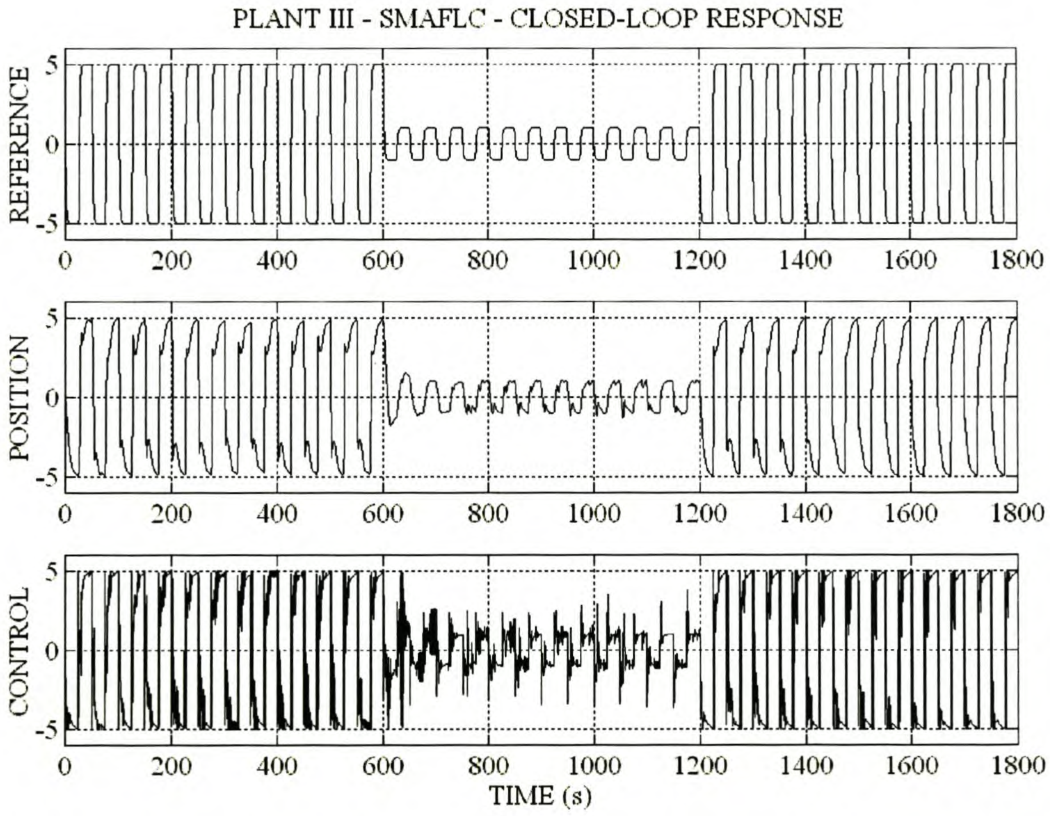

Figure 53 Plant III – SMAFLC – Closed-loop response.

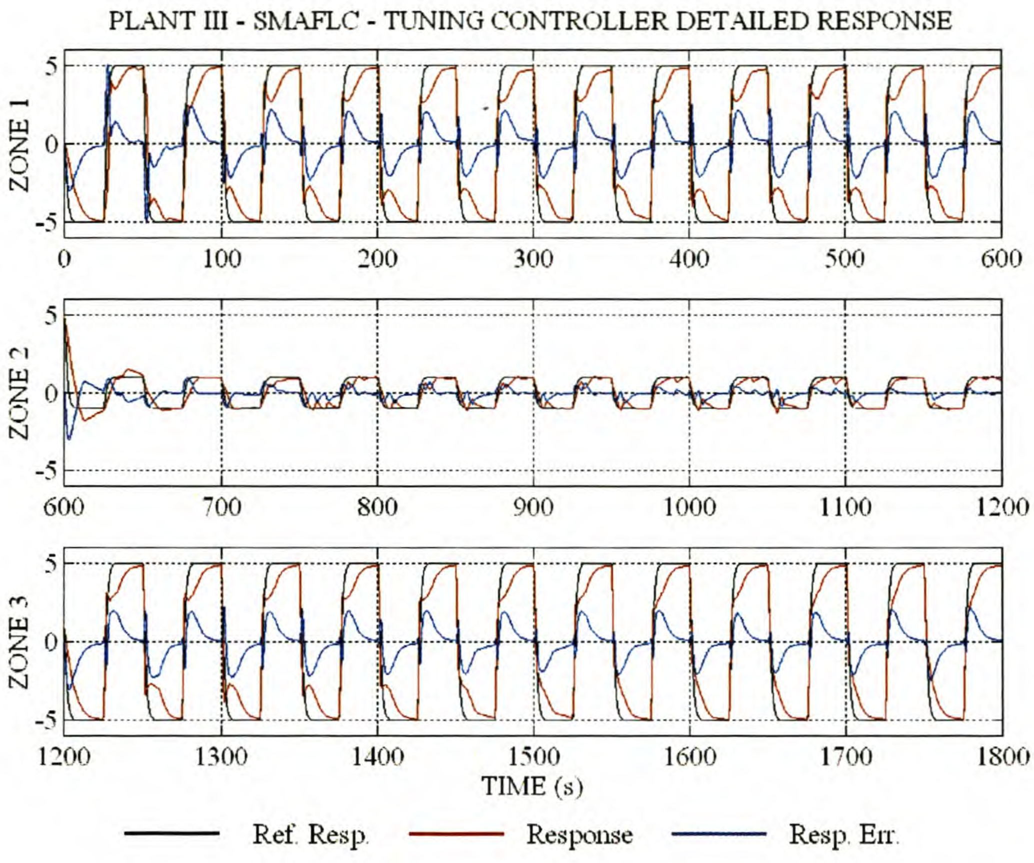

Figure 54 Plant III – SMAFLC – Tuning controller detailed response.

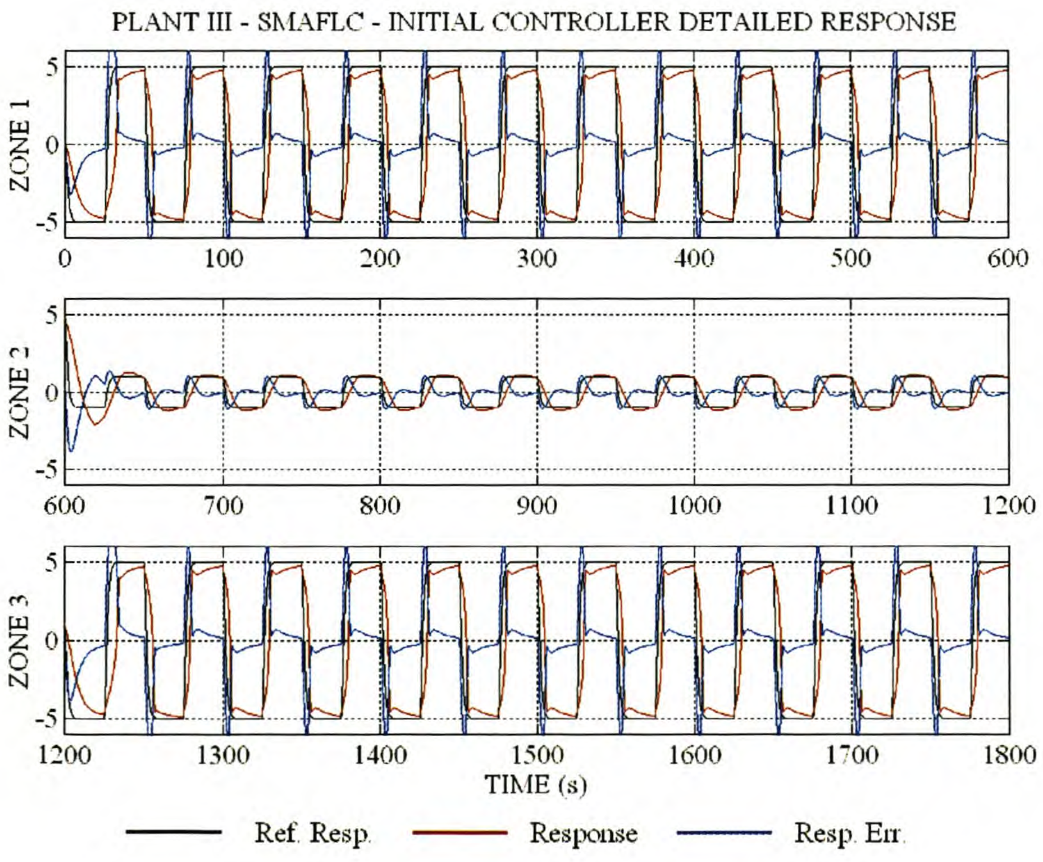

Figure 55 Plant III – SMAFLC – Initial controller detailed response.

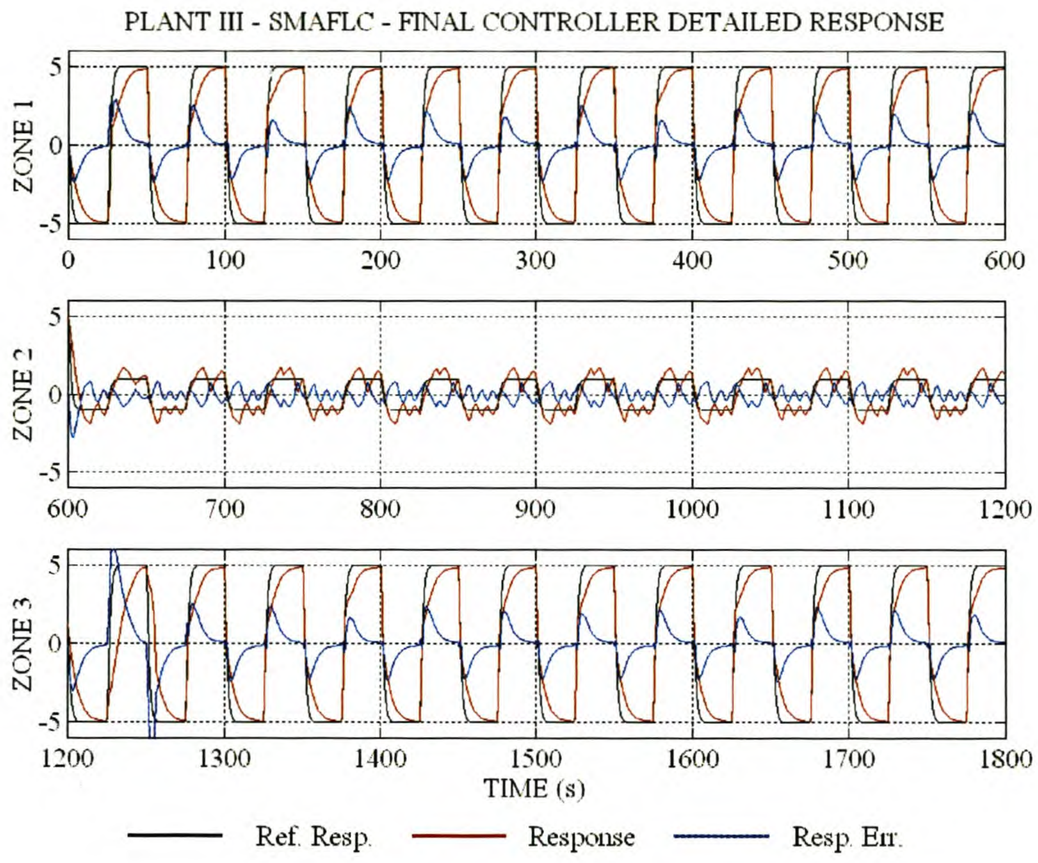

Figure 56 Plant III – SMAFLC – Final controller detailed response.

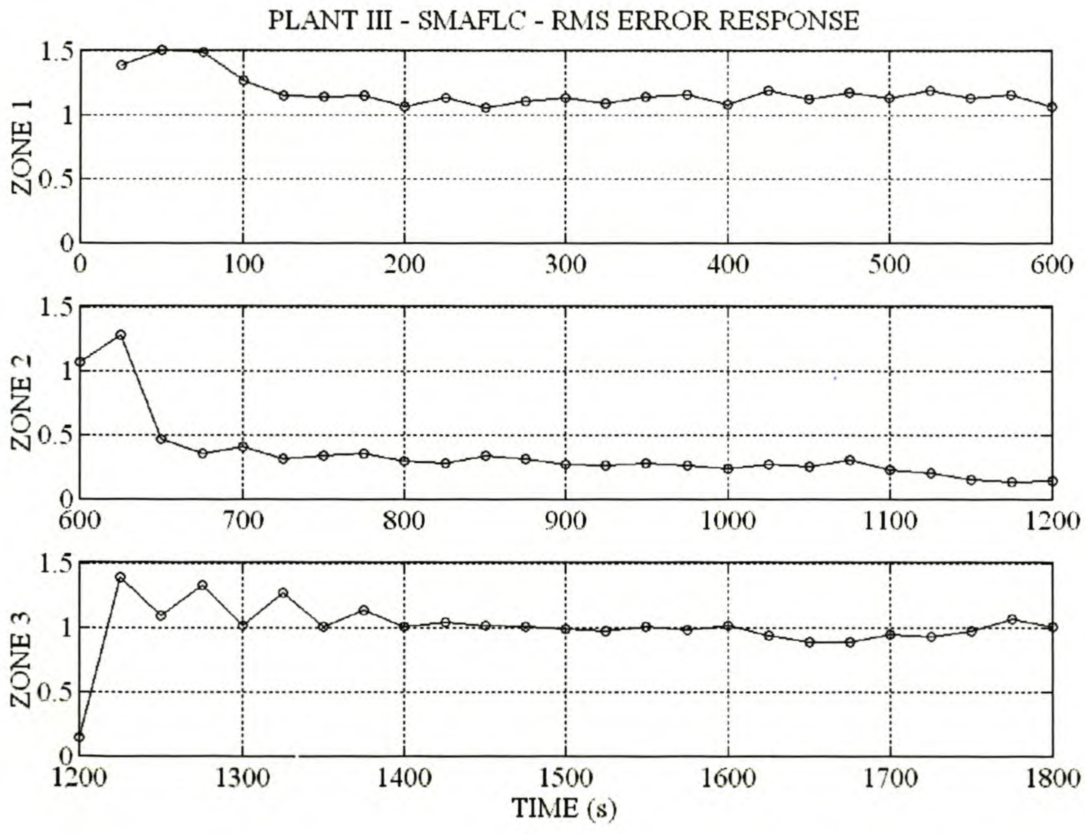

Figure 57 Plant III – SMAFLC – Response error RMS value for each step change.

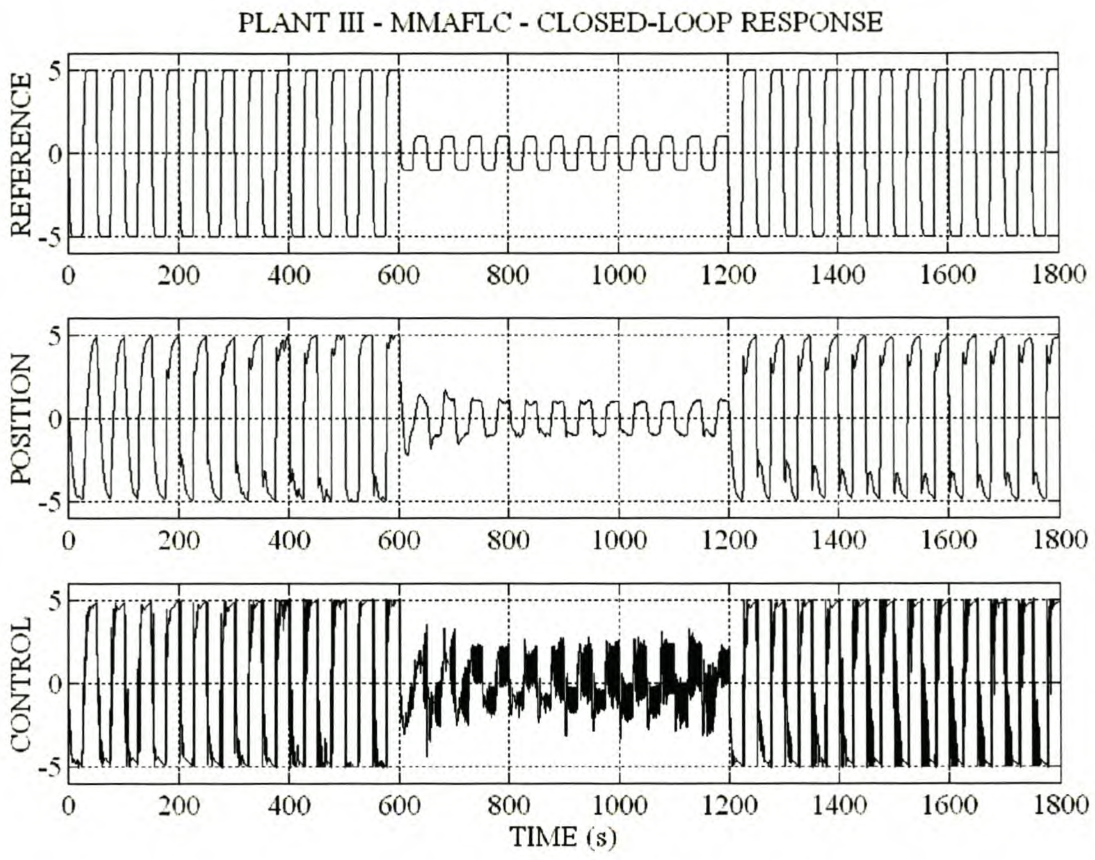

Figure 58 Plant III – MMAFLC – Closed-loop response.

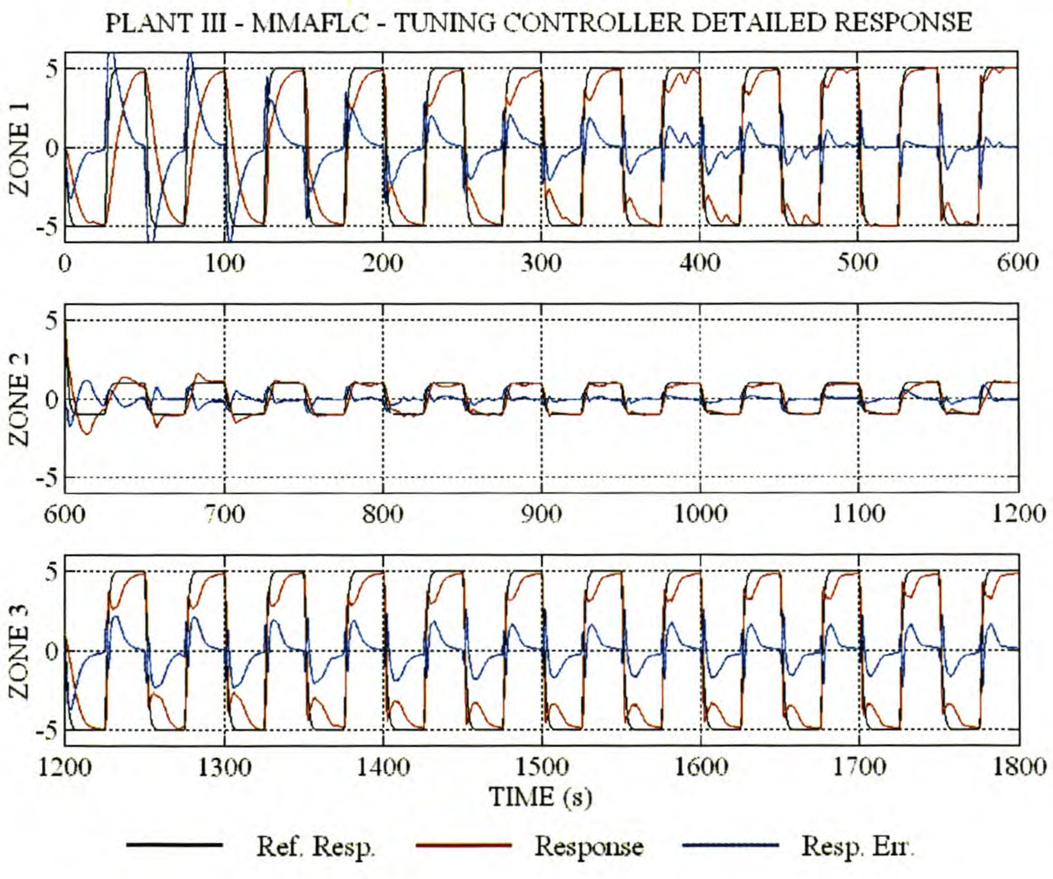

Figure 59 Plant III – MMAFLC – Tuning controller detailed response.

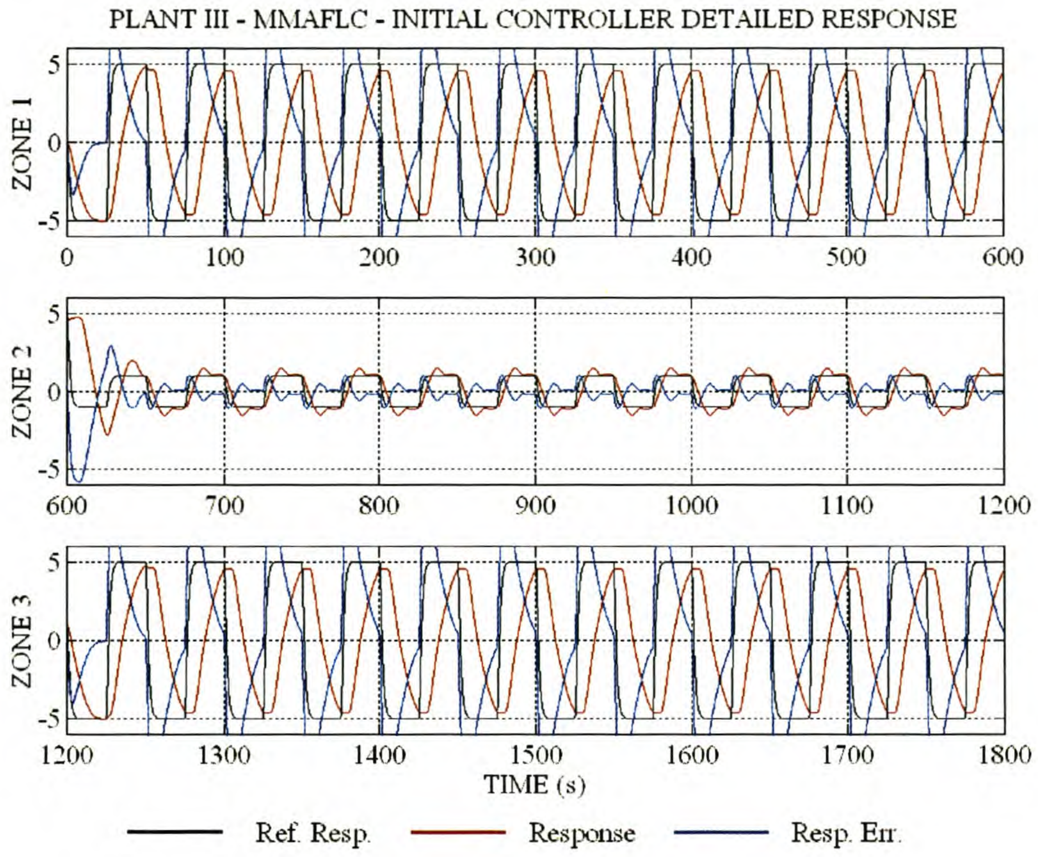

Figure 60 Plant III – MMAFLC – Initial controller detailed response.

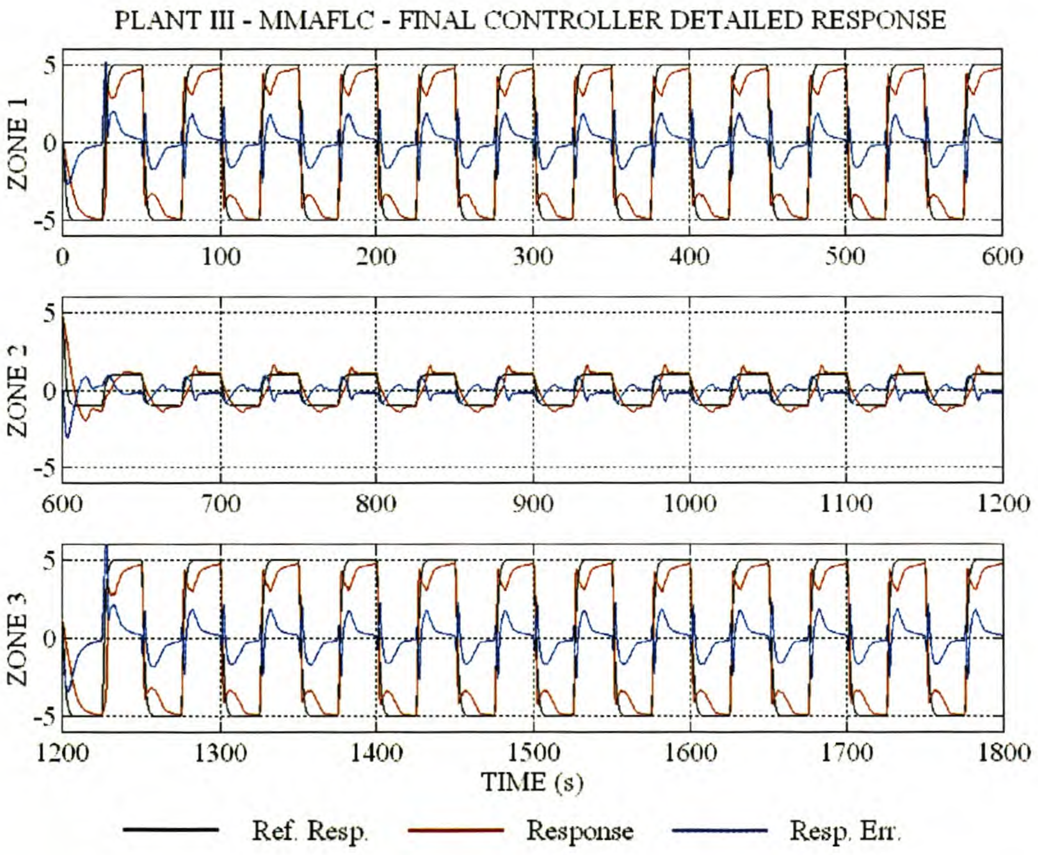

Figure 61 Plant III – MMAFLC – Final controller detailed response.

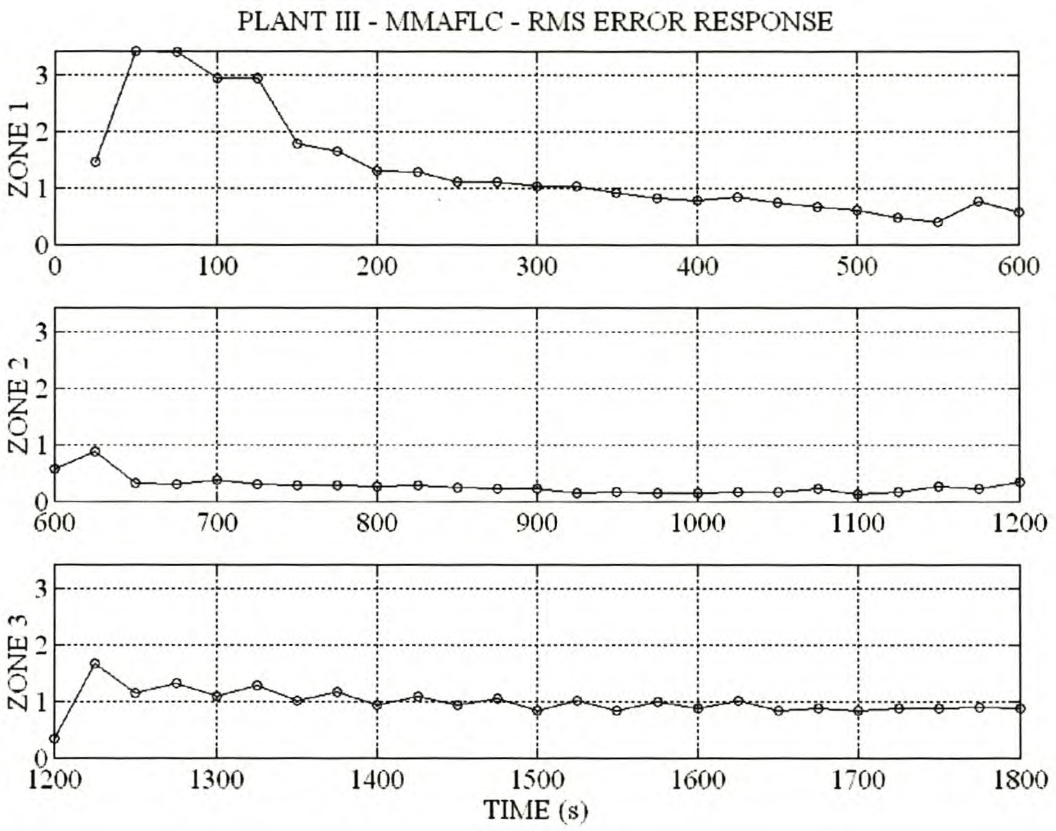

Figure 62 Plant III – MMAFLC – Response error RMS value for each step change.

4.3.4 PERFORMANCE EVALUATION AND COMPARISON

As shown in paragraph 4.3.2, the MMFLC responded more slowly and with less overshoot than the SMFLC to large step changes. For small step changes the SMFLC exhibited lower overshoot with a slight increase in rise time compared to the MMFLC. The poor performance of the MMFLC to small step changes is due to the similarity in the rule-base in the slowest velocity domain to that of the SMFLC. This similarity is caused by the normalisation process of the S_I and G_I vectors in the automated design algorithm.

As shown in paragraph 4.3.3 the MMAFLC showed overall better performance than the SMAFLC based on the RMS values of the response errors. The final multi-mode controller showed smaller response errors than the final single-mode controller. The oscillations of the final single-mode controller to small step changes have been eliminated by the final multi-mode controller. Both the SMAFLC and MMAFLC showed high-frequency noise in the control signal (Figure 53 c and Figure 58 c). The frequency of the oscillations was higher than the plant bandwidth, resulting in an essentially clean response (Figure 53 b and Figure 58 b). The cause of the noise can be attributed to differentiation noise and plant excitation due to rule switches.

4.4 CONCLUSIONS

As shown in paragraphs 4.1.2, 4.2.2 and 4.3.2 the automated rule-base design algorithm sacrifices speed of response in an attempt to limit overshoot and establish a more homogenous closed-loop response in the non linear domains. The choice of gain and speed information used in the design algorithm is critical and may require experimental modification. The design method thus greatly reduces the amount of manipulated parameters from the rule-base entries to the intuitively understandable S_I and G_I vectors.

The rule-base adaptive algorithm (paragraphs 4.1.3, 4.2.3 and 4.3.3) can greatly improve the performance of a fuzzy logic controller by adapting the rule-base to give a required tracking. Since a rule has a discrete number of consequences, based on the fuzzy sets defined for the output universe of discourse, perfect tracking may not always be possible. Care should be taken to avoid oscillation in the rule-base.

In general the addition of the extra state input to the fuzzy logic controller allowed more knowledge to be incorporated into the controller and improved performance across the whole non-linear system domain. The performance improvement and ease of design with the proposed method offsets the complexity disadvantage.

5 TEMPERATURE CONTROL OF HOMOGENEOUS AZEOTROPIC DISTILLATION COLUMN

5.1 INTRODUCTION

The control of the reaction front temperature of a heterogeneous azeotropic distillation column with SMFLC and MMFLC is investigated.

The distillation process is used to split a mixture of fluids based on their relative volatility [8]. A binary fluid mixture at a steady temperature will reach an equilibrium condition between the vapour and the fluid. The vapour will be richer in the more volatile component and the fluid richer in the less volatile component, leading to a separation in components [8, 66]. The principles of distillation have been thoroughly investigated and documented [8, 66]. Various control structures and strategies have been reported, with robust and non-linear control techniques slowly finding applications in different fields of distillation control [5, 11, 15, 29, 39, 49, 66, 67, 71, 91, 97]. For the purpose of this work, an investigation of the distillation process and the various control strategies is not required.

The phase diagram of a binary azeotropic or constant boiling point mixture is shown in Figure 63.

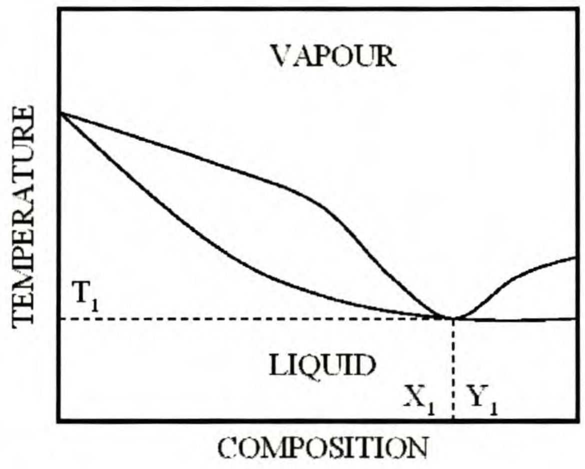

Figure 63 Phase diagram of a binary, azeotropic mixture.

In an azeotropic mixture like water and ethanol, the vapor and fluid have the same composition during equilibrium conditions, as indicated by the touching of the two-phase boundaries shown in

Figure 63. This mixture can thus not be separated with normal distillation techniques [8, 66]. The separation of water and ethanol requires the addition of an entrainer. Benzene is added to form an unstable ternary minimum azeotrope that splits into two liquid phases in the decanter: a light benzene-rich phase (the L_{do} fraction) and a heavy water-rich phase (the L_{da} fraction) [8, 58, 66, 90]. The addition of the entrainer thus adds a degree of freedom to the system [56].

The homogenous azeotropic distillation column studied in this work is shown in Figure 64.

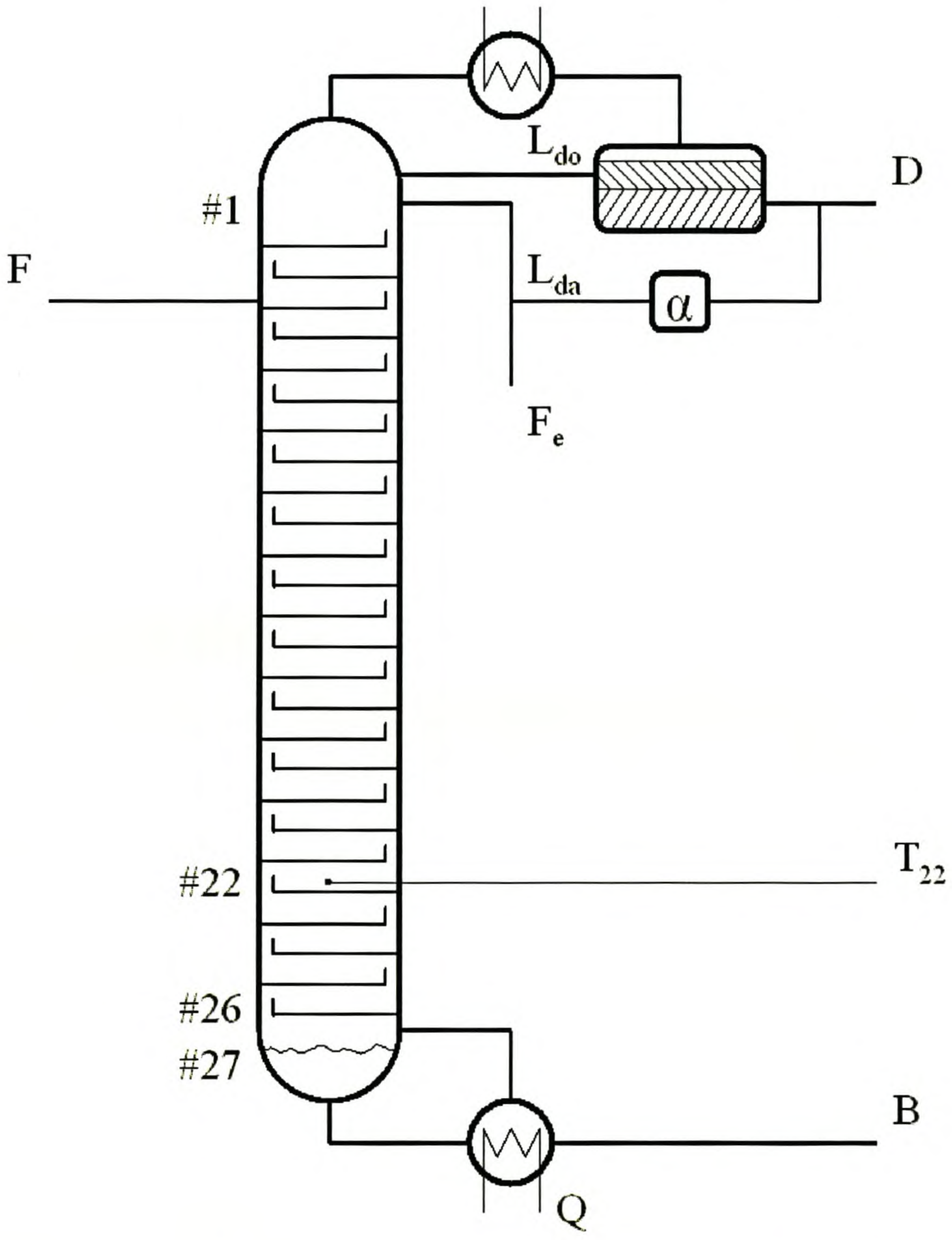

Figure 64 Homogeneous azeotropic distillation column with 27 stages.

The column has 26 trays with the reboiler forming the 27th stage. The column is fed with an ethanol and water mixture at a feed rate of F mol/min. The benzene entrainer is added at the top of the column, at a flow of F_e mol/min. Energy input to the column is manipulated with the reboiler input Q J/min. The distillate is extracted from the top of the column, condensed and fed to a decanter, where it splits into a top organic phase (L_{do}) and bottom aqueous phase (L_{da}). The complete organic phase is refluxed, while a fraction of the aqueous phase is refluxed with reflux ratio (α). The distillate product is drawn from the aqueous phase. The purified ethanol is drawn from the bottom of the column, at a flow rate of B mol/min. Ethanol quality is traditionally controlled by manipulation of the reaction front temperature profile indicated by the sharp temperature break in the lower sections of the column [56,58]. The reaction front temperature profile before and after an input flow disturbance is shown in Figure 65.

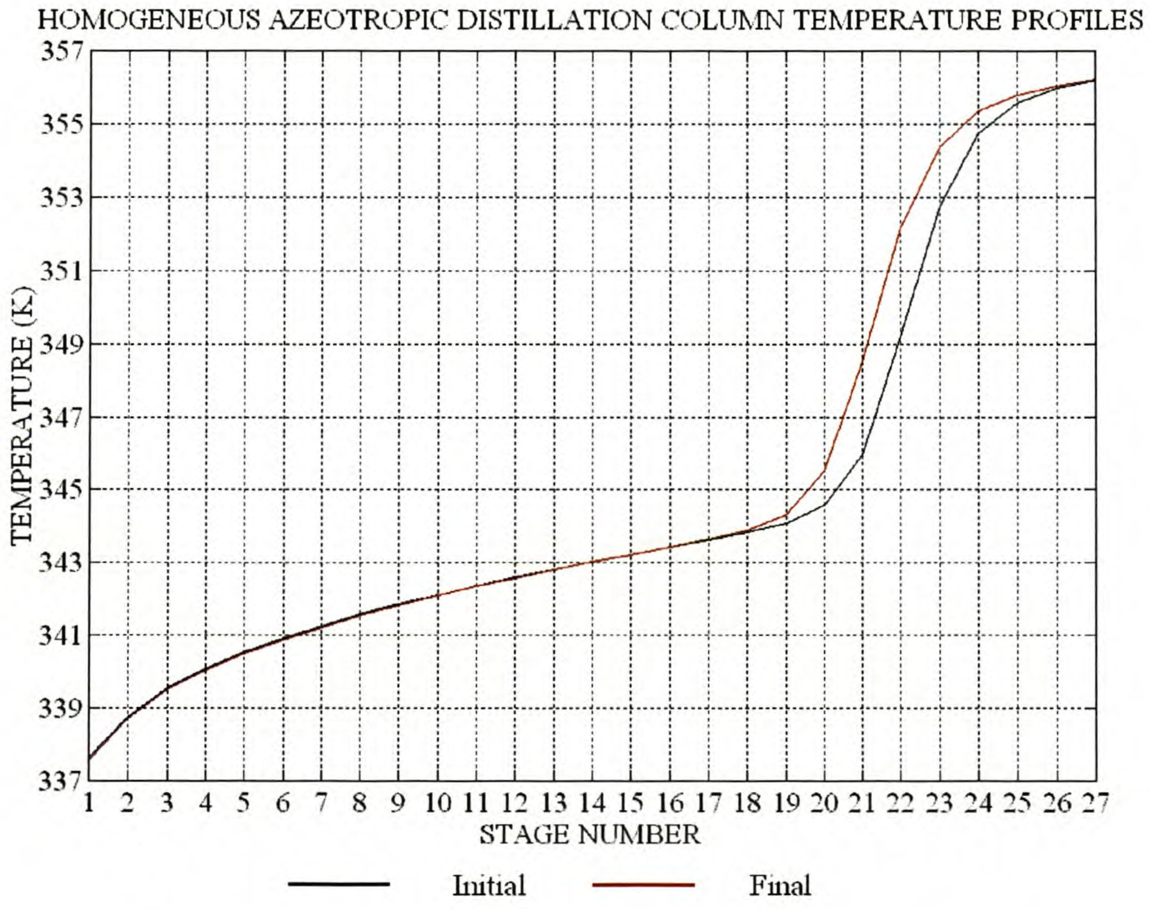

Figure 65 Distillation column: Reaction front temperature profile before and after a 5% input flow disturbance.

The position of the sharp temperature break can be regulated by controlling the temperature of a single tray, as shown in Figure 64. Alternatively, the average of a number of tray temperatures can be regulated [56, 57, 58, 59].

5.2 PLANT AND SIMULATION MODEL

The column used in this work is a literature example [90] and the SIMuLINK® model was adapted from a master's project currently nearing completion. The SIMuLINK® S-Function file is given in Appendix VI.I. For the purpose of this work, the simulation model of the azeotropic distillation column is seen as a black box model. The SIMuLINK® block diagram of the azeotropic distillation column, as studied in this work, is shown in Figure 66.

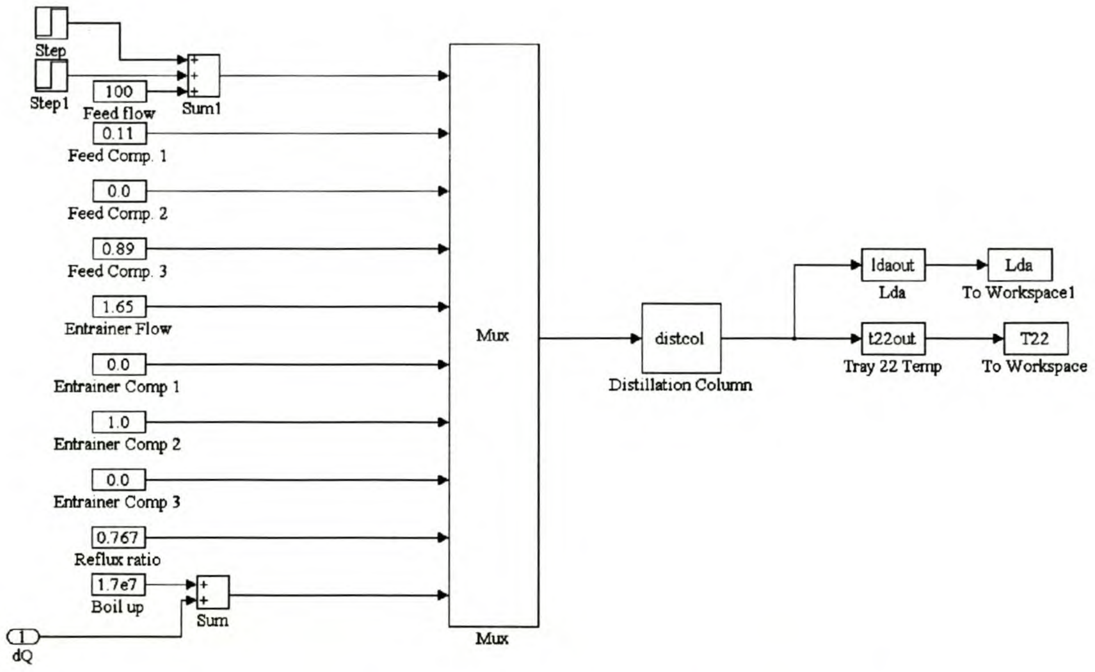

Figure 66 Distillation column – Input and output structure.

The ten inputs with their nominal values are shown. For the purpose of this work, the reflux ratio and the compositions of all the input flows are kept constant at their nominal values. The input feed flow rate (F_e) is regarded as the disturbance input. The boil-up rate (Q) is manipulated around its nominal values as the controlled system input. The temperature on tray 22 is extracted from the 164 element output vector and is used to regulate the position of the reaction temperature front. The output vector consists of the following elements with only the L_{da} fraction and the tray 22 temperature of importance in this study.

- $Z = y(1{:}108)$
- $T = y(109{:}135)$
- $V = y(136{:}162)$
- $L_{do} = y(163)$
- $L_{da} = y(164)$

The Z output vector gives the composition ratios of the water, benzene and ethanol on the 26 column trays. The temperature on each stage of the column is given by T. The V output vector is the vapour flow on the 26 trays. The last two elements in the output vector are the L_{do} and L_{da} fractions.

For test purposes the column is subjected to a disturbance in input feed flow rate between 100 mol/min and 105 mol/min. The response to a 5 mol/min increase in feed flow at 0 min and a 5 mol/min decrease at 180 min is shown in Figure 67.

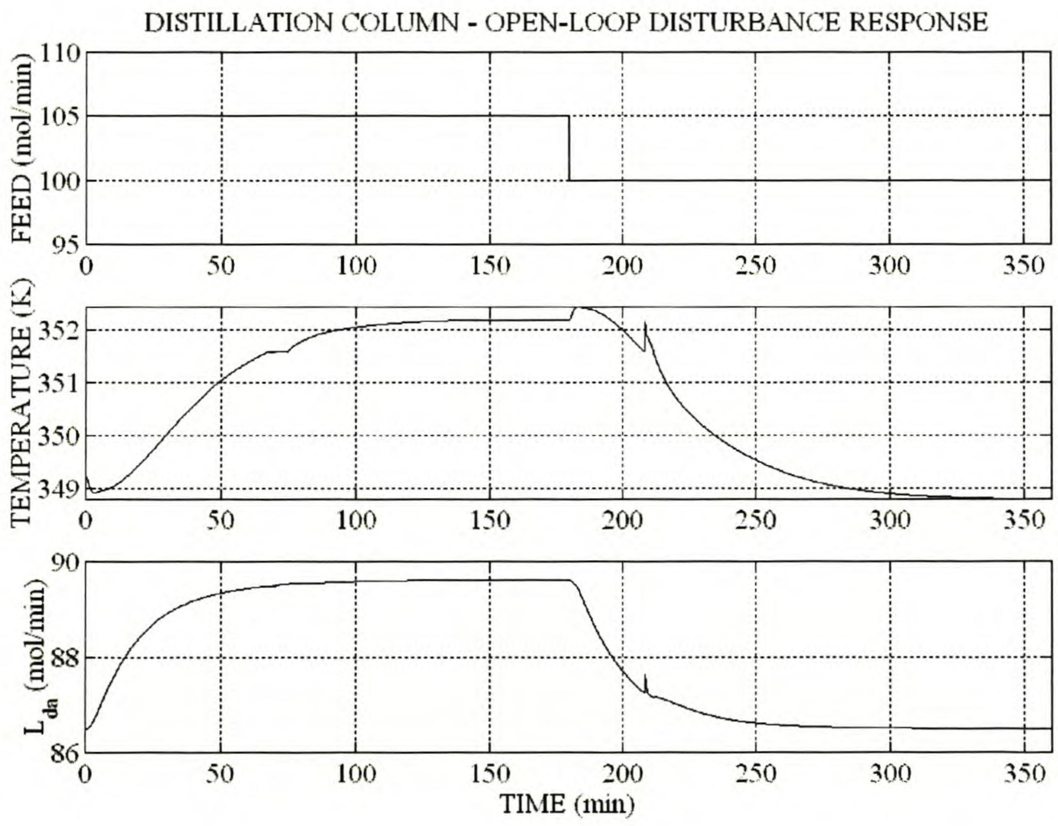

Figure 67 Distillation column – Open-loop temperature and L_{da} fraction response to input flow step change.

As the feed flow increases, the L_{da} fraction and the temperature increase. The temperature exhibits a non-minimal phase response. The RMS value of the temperature error relative to its nominal value of 349.22K is 1.997K. The non-linear characteristics of the plant are evident from the non-symmetric temperature response. The reaction front temperature profile at the start and end of the 5% increase in feed flow is shown in Figure 65.

5.3 TEMPERATURE CONTROL

The control objective is to eliminate disturbances in the reaction front temperature, as measured on tray 22 (T_{22}), caused by fluctuations in the feed flow rate (F_e) by manipulation of the boil-up rate (Q). All the distillation column inputs are modified around their nominal values as given in Figure 66. The block diagram of the SMFLC and MMFLC is shown in Figure 68

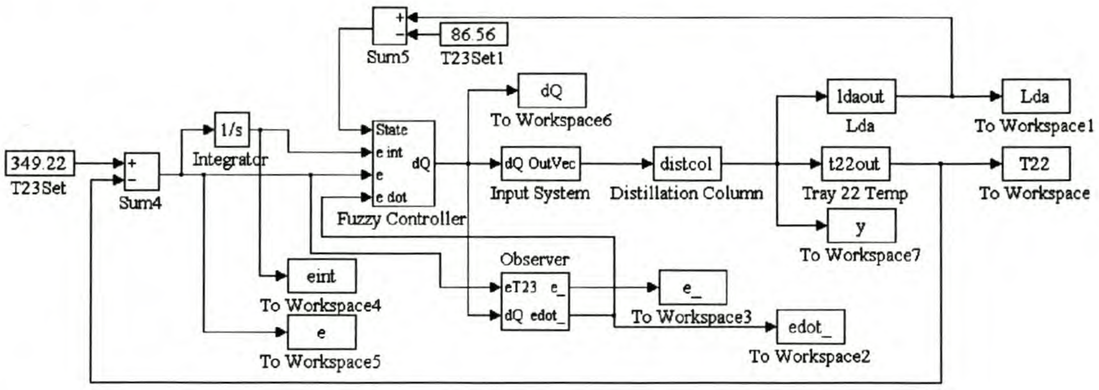

Figure 68 Distillation column – Non-adaptive fuzzy logic control.

Since the plant is of type zero, a PID-type controller is required. The error integral ($\int e.dt$), error (e) and error rate of change (de/dt), via gains K_i, K_p and K_d, are used as controller inputs. Initial controller designs indicated high system sensitivity to high-frequency noise generated by direct differentiation of the error signal. A linear state observer estimating the error signal ($\hat{e}$) supplies the estimated error rate of change ($d\hat{e}/dt$) for the controller. The deviation of the L_{da} fraction from its nominal value was chosen for the state input. The L_{da} fraction is indicative of the amount of water in the column. The water content influences the thermodynamic and chemical process and thus the plant dynamics. The rule-base design algorithm and gain optimisation technique are used to design an initial SMFLC and finally a MMFLC. The rule-base adaptive technique cannot be applied to this problem since the structure does not conform to the model reference adaptive control structure.

5.3.1 OBSERVER DESIGN

A linear state observer estimating the error ($\hat{e}$) and error rate of change ($d\hat{e}/dt$) eliminates injected high-frequency noise created by a discrete differentiating element. The observer is based on a small signal, linear, state space model, determined from the response of the temperature error to a step change in boil-up. The structure of a second-order observer with bias estimation is shown in Figure 69 [12, 50].

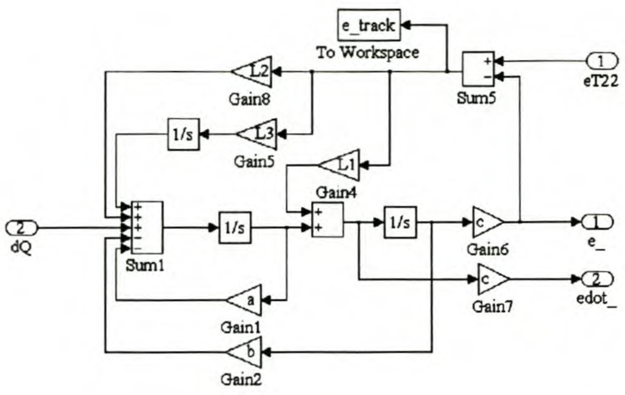

Figure 69 Distillation column – Linear observer with bias estimation.

The temperature response of the distillation column to a 2.5% step change in boil-up is shown in Figure 70.

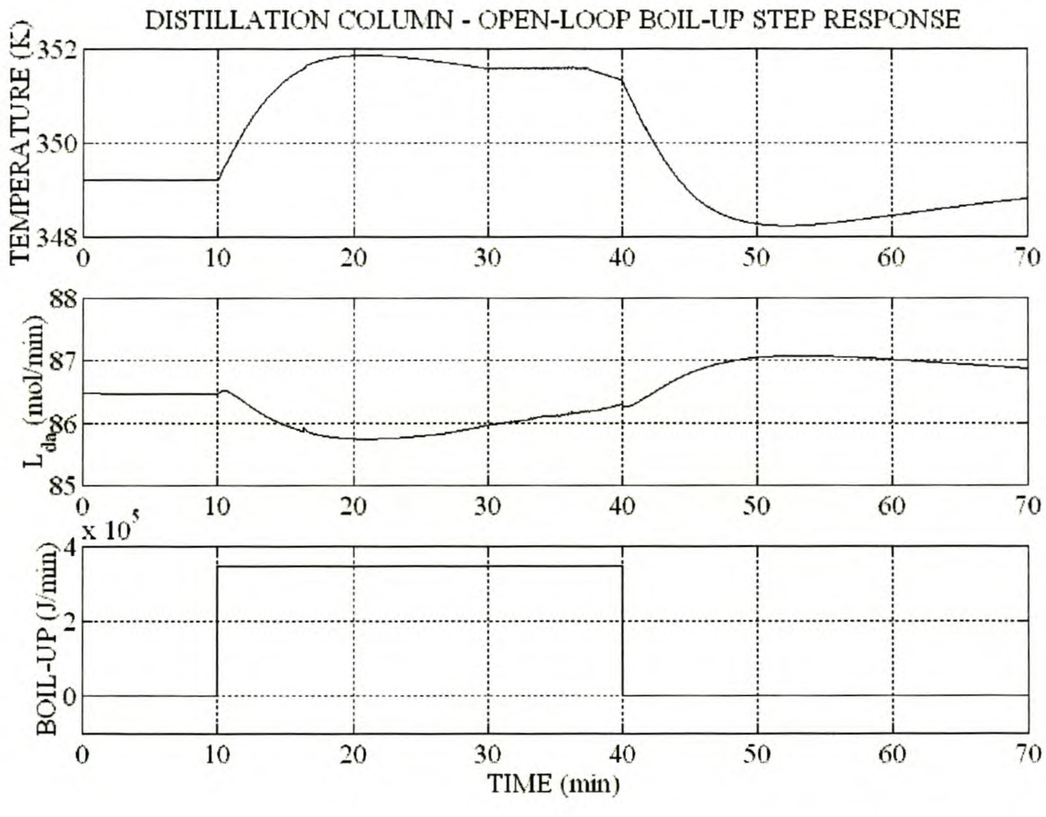

Figure 70 Distillation column – Open-loop temperature and L_{da} fraction response to boil-up step change.

The non-linearity of the system is clear from the unsymmetrical response in T_{22} and L_{da} to boil-up changes. The linear observer thus clearly requires the addition of a state for bias estimation. A second-order model with $\xi = 0.8$ results in the minimal overshoot and fast initial response. Based on the rise time, $\omega_n = 0.8$. The linear model poles are located at $s_{1,2} = -0.64 \pm 0.48j$. The error dynamics were chosen as five times the plant dynamics with poles selected at $s_1 = -3$, $s_2 = -3.5$ and $s_3 = -4$. The state space model, augmented with the bias state, is shown in Equation 35. The error feedback gain matrix is also given.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -0.64 & -1.28 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} dQ$$

$$y = \begin{bmatrix} -3.61\times10^{-6} & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ w \end{bmatrix}$$

$$L = \begin{bmatrix} -4.0083\times10^{6} & -1.7441\times10^{7} & -1.1634\times10^{7} \end{bmatrix}$$

Equation 35

The performance of the observer is shown in Figure 71.

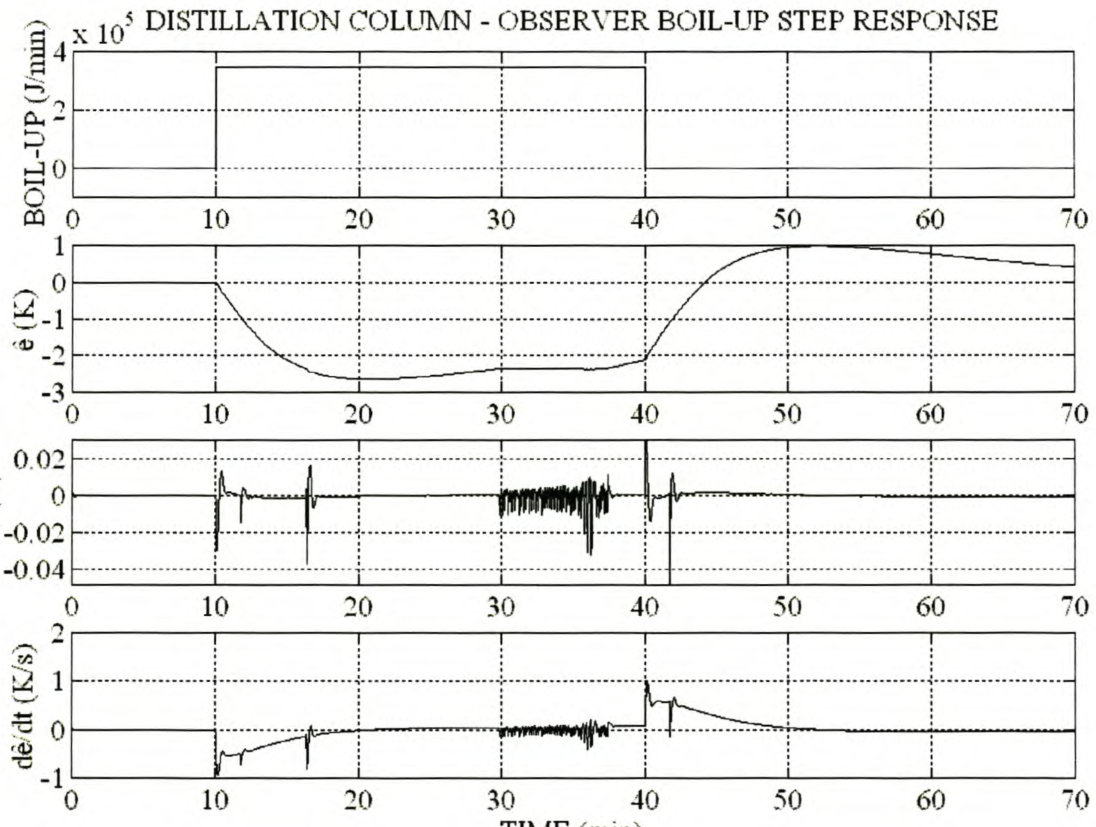

Figure 71 Distillation column – Observer response to boil-up step change.

The feed flow rate was increased at 10 minutes and decreased at 180 minutes. The estimated error ($\hat{e}$) tracks the true error very accurately. The observer tracking error (e-$\hat{e}$) shows a small error directly following the step change. The estimated error rate of change ($d\hat{e}/dt$) shows a little high-frequency noise in the steady state. Both controllers were subjected to 5% step change in feed flow rate.

5.3.2 SINGLE-MODE FUZZY LOGIC CONTROL

The universe of discourse and number of fuzzy sets for each input and output are listed in Table 13.

Input	Minimum	Maximum	Sets
State	**0**	**2**	**1**
Integral	**-60**	**60**	**9**
Error	**6**	**6**	**9**
Rate	**0.025**	**0.025**	**9**
Control	**-173400**	**173400**	**21**

Table 13 Distillation column – SMFLC structure summary.

As suggested by the design algorithm, triangular membership functions define all the fuzzy sets. The gain information vector is chosen as $G_I = [1]$ and the speed information matrix, $S_I = [1]$. The controller input gains ($K_i = 200$, $K_p = 200$ and $K_d = 0.01$) were determined experimentally. The MATLAB® design file for the SMFLC is listed in Appendix VI.II.

The closed-loop response of the SMFLC is shown in Figure 72.

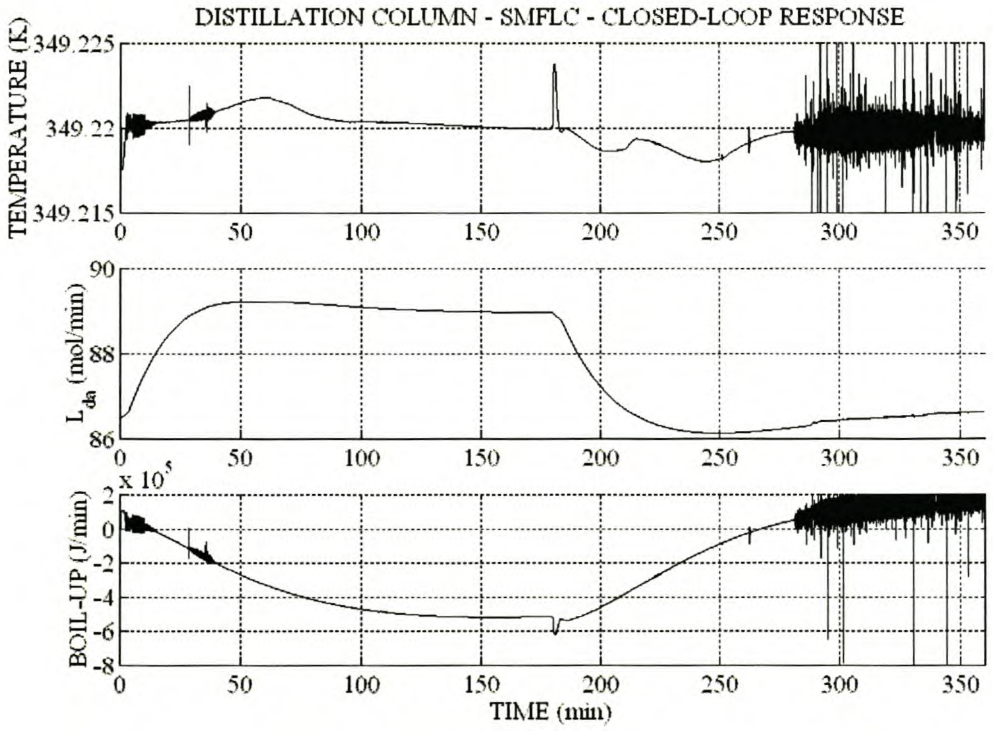

Figure 72 Distillation column – SMFLC – Closed-loop response.

The amplitude of the temperature disturbance is reduced to less than 0.005K. The RMS value of the temperature error is 0.0016. The temperature response shows a small, high-frequency oscillation for values of L_{da} close to the nominal value. Simulation work has shown that the system is sensitive to oscillations at the nominal operating point at high controller gain values, suggesting the possibility of employing a MMFLC.

5.3.3 MULTI-MODE FUZZY LOGIC CONTROL

The universe of discourse and number of fuzzy sets for each input and output are listed in Table 14.

Input	Minimum	Maximum	Sets
State	**0**	**2**	**2**
Integral	**-60**	**60**	**9**
Error	**6**	**6**	**9**
Rate	**0.025**	**0.025**	**9**
Control	**-173400**	**173400**	**21**

Table 14 Distillation column – MMFLC structure summary.

The structure and type of membership functions were chosen to be identical to those of the SMFLC given in 5.3.2. To limit the complexity of the MMFLC only two fuzzy sets were chosen for the L_{da} fraction, supplying information regarding the water content of the column. Based on qualitative thermodynamic reasoning, distillation experts thought the plant to have a lower small signal gain at a high water content. The gain information vector, G_I = [1 0.75] and the speed information matrix, S_I = [1 1] were determined experimentally to limit the oscillations at low values of L_{da} exhibited by the SMFLC without degrading closed-loop performance. The controller input gains ($K_i = 200$, $K_p = 200$ and $K_d = 0.01$) were chosen as identical to the SMFLC. The MATLAB® design file for the MMFLC is listed in Appendix VI.III.

The closed-loop response of the MMFLC is shown in Figure 73. The maximum amplitude of the temperature disturbance is less than 0.006K, with the RMS value of the temperature error 0.000913K. The temperature shows a smaller sensitivity to high-frequency oscillations at small values of L_{da} than the SMFLC.

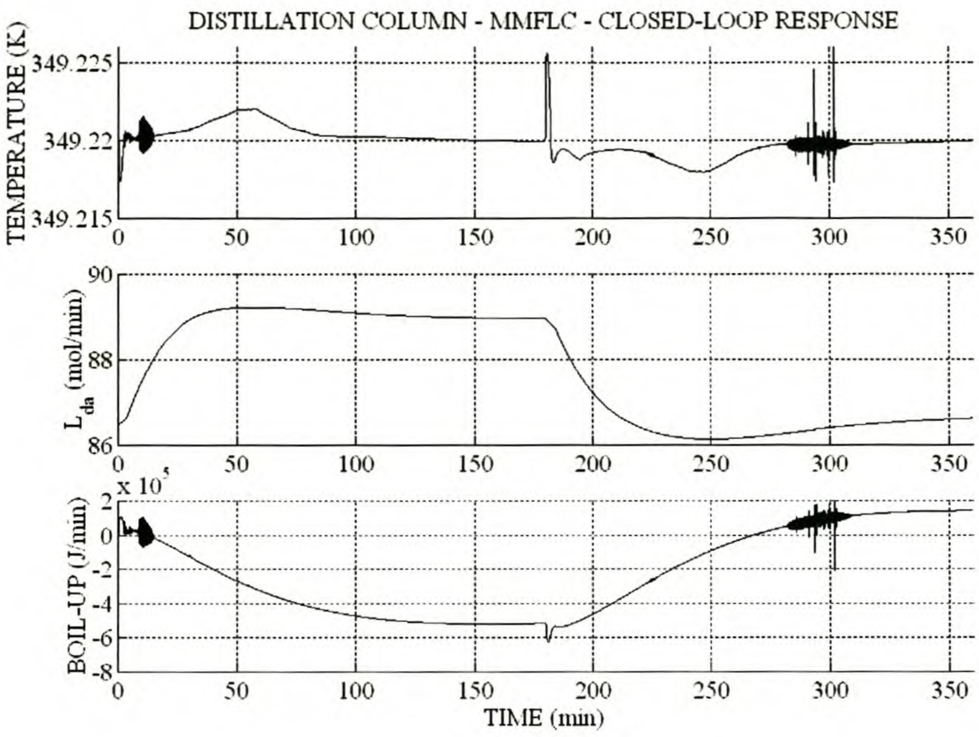

Figure 73 Distillation column – MMFLC – Closed-loop response.

5.4 CONTROLLER COMPARISON AND CONCLUSIONS

The closed-loop temperature response of the SMFLC and MMFLC is compared in Figure 74. The reduction in sensitivity (amplitude and duration) to high-frequency oscillations of the MMFLC compared to the SMFLC is clear. This reduction in sensitivity is caused by the lower controller gain at low L_{da} fractions and is responsible for the slightly larger peak amplitudes of the temperature error between 30s and 100s and 250s and 300s. As soon as the L_{da} fraction increases to start firing the second domain rules, the higher system gain decreases the error. The larger overshoot of the MMFLC compared to the SMFLC at 180s indicates that the system has less damping at higher values of L_{da}. An increase in the damping of the MMFLC in the high L_{da} fraction zone should reduce the overshoot peak.

Considering the small increase in error due to the overshoot, compared to the already significant reduction in the RMS value of the temperature error of the MMFLC relative to the SMFLC, this was not regarded as important and not investigated.

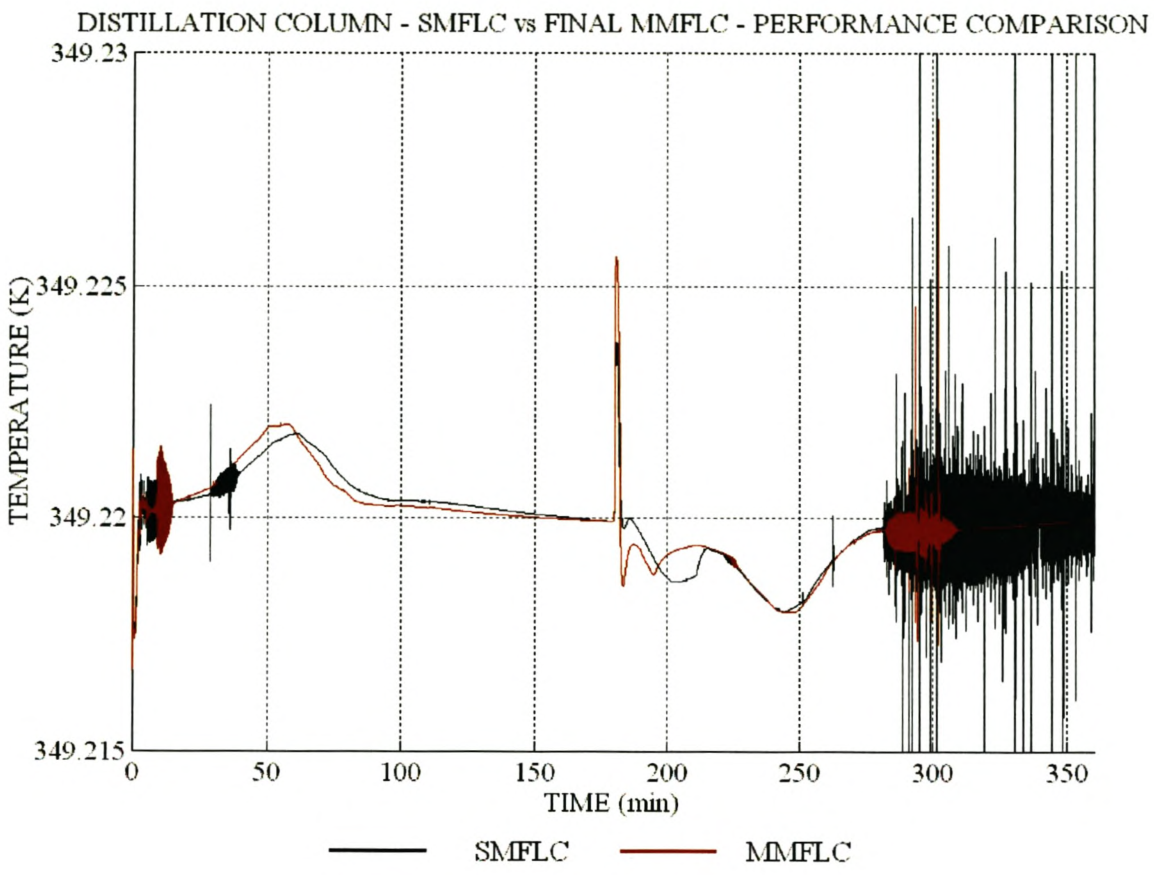

Figure 74 Distillation column – SMFLC vs MMFLC performance comparison.

6 CONTROL OF EXOTHERMIC CSTR

In this chapter the proposed design method is used to implement a SMAFLC and a MMAFLC for direct concentration control of an exothermic chemical reaction in a continuously stirred tank reactor (CSTR). Various authors have investigated the non-linear behaviour of the CSTR system and indicated that in certain domains of the non-linear state space the system exhibits multiple steady states, limit cycles or non-linear dynamics [38, 41, 81]. Non-linear feedback and predictive control schemes have been implemented successfully for temperature control [38, 41].

6.1 PLANT AND SIMULATION MODEL

An excellent discussion on the non-linear dynamic behaviour of the exothermic CSTR can be found in [81]. The two non-linear state equations for the CSTR are given in Equation 36 with the parameters as defined in the parameter list below [41].

$$\frac{dC}{dt} = -k_0 C e^{-E/RT} + \frac{Q}{V}\left(C_f - C\right)$$
$$\frac{dT}{dt} = \frac{-\Delta H}{\rho C_p} k_0 C e^{-E/RT} + \frac{Q}{V}\left(T_f - T\right) + \frac{UA}{\rho C_p V}\left(T_c - T\right)$$

Equation 36

- *A – Heat exchange surface area (m^2)*
- *C – Reactant concentration (mol/m^3)*
- *C_f – Feed concentration (mol/m^3)*
- *C_p – Specific heat capacity (J/kg.K)*
- *E – Activation energy (J/mol)*
- *k_0 – Arrhenius pre-exponential constant*
- *Q – Volumetric flow rate (m^3/s)*
- *R – Gas constant (J/mol.K)*
- *T – Reactor temperature (K)*
- *T_f – Feed temperature (K)*
- *T_c – Coolant temperature (K)*
- *U - Heat transfer coefficient (J/(s.K.m^2))*

- *V – Reactor volume (m^3)*
- *ΔH – Heat of reaction (J/mol)*
- *ρ- Reactant density (kg/m^3)*

Assume that the reactor volume (V) is kept constant and that the reactant temperature and coolant temperature are directly measurable. Assume further that the reactant concentration is directly measurable or can be determined from direct measurements.

Define the dimensionless parameters of Equation 37, with Q_0 and T_{f0} defined as the nominal concentration and volumetric flow-rate variables. Equation 36 can now be transformed into the dimensionless state space system of Equation 38. [38, 41, 81]. The parameters and variables of Equation 37 and Equation 38 are defined in the parameters lists given.

$$\beta = \frac{(-\Delta H)C_f}{\rho C_p T_{f0}}\gamma$$

$$\delta = \frac{UA}{\rho C_p Q_0}$$

$$\gamma = \frac{E}{RT_{f0}}$$

$$\varphi = \frac{V}{Q_0}k_0 e^{-\gamma}$$

$$q = \frac{Q}{Q_0}$$

Equation 37

- *β – Dimensionless heat of reaction.*
- *δ – Dimensionless activation energy*
- *γ – Dimensionless heat transfer coefficient*
- *φ – Damkohler number*

$$\frac{dx_1}{dt} = -\varphi x_1 K(x_2) + q(1 - x_1)$$

$$\frac{dx_2}{dt} = \beta\varphi x_1 K(x_2) - (q + \delta)x_2 + u + v$$

$$K(x_2) = e^{\frac{\gamma x_2}{\gamma + x_2}}$$

$$\tau = \frac{Q_0}{V}t$$

$$u = \frac{\gamma\delta}{T_{f0}}(T_c - T_{f0})$$

$$v = \frac{\gamma q}{T_{f0}}(T_f - T_{f0})$$

$$x_1 = \frac{C}{C_f}$$

$$x_2 = \frac{T - T_{f0}}{T_{f0}}\gamma$$

Equation 38

- *τ – Dimensionless time.*
- *u – Dimensionless process input*
- *x_1 – Dimensionless reactant concentration*
- *x_2 – Dimensionless reactant temperature*

From Equation 38 the non-linear dynamic behaviour of the plant is clearly a function of temperature. The different types of dynamic behaviour and corresponding domains of the state space are given in [81]. The steady state characteristics of the plant can be calculated for various values of *u* by solving for Equation 38 equal to 0. The nominal values of the parameters used in this study are given in the parameter list below [38].

- *$\beta = 8.0$*
- *$\delta = 3.0$*
- *$\gamma = 20$*
- *$\varphi = 0.072$*

The steady state temperature curve of the CSTR is shown in Figure 75 for three values of δ [38]. For small values of δ the plant shows multiple steady states, of which the middle state is unstable [81]. For compliance with the initial assumptions of the design method, δ = 3.0 was chosen. The steady-state concentration curves are shown in Figure 76. At δ = 3.0 the steady-state temperature curve is almost linear, with the steady-state concentration curve showing stronger non-linear deviation. The non-linear dynamics of the concentration and temperature are evident in the open-loop step response, as shown in Figure 77. According to [38] the system dynamics are first-order. It is the opinion of the author that the system is second-order, as is evident from the overshoot in concentration and temperature in certain domains. The MATLAB® S-Function implementation of the plant is given in Appendix VII.I.

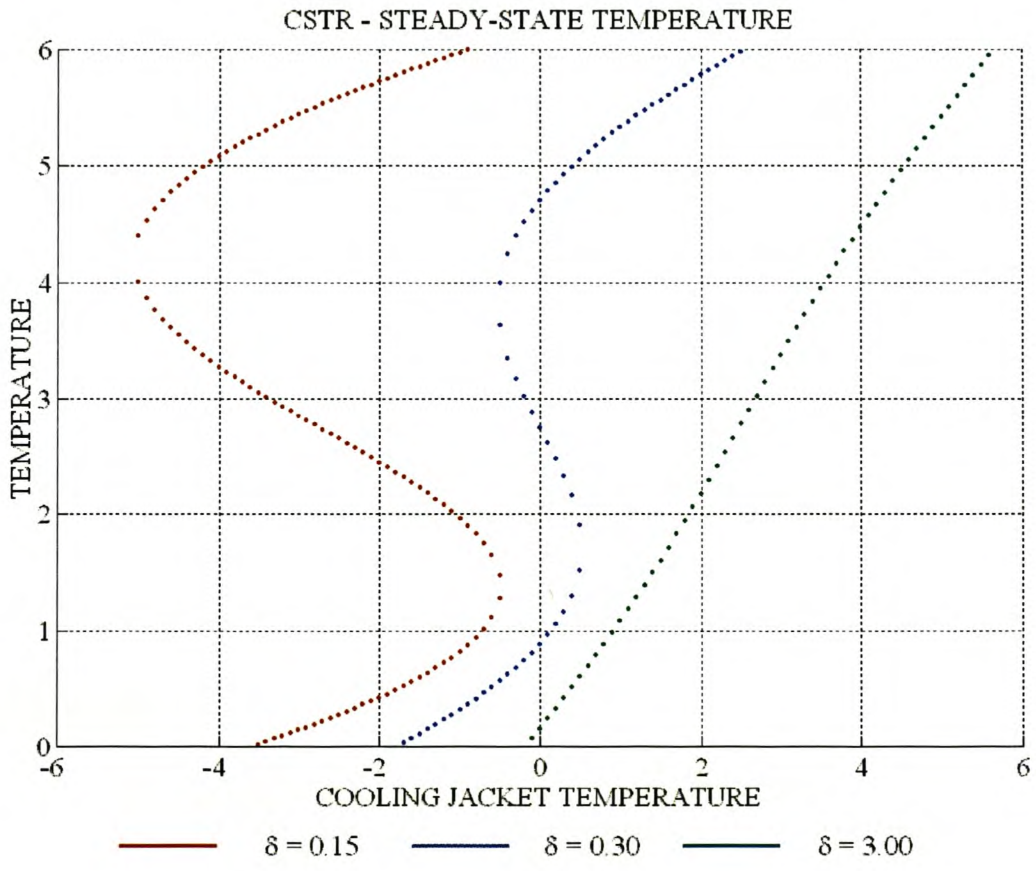

Figure 75 CSTR – Steady-state temperature.

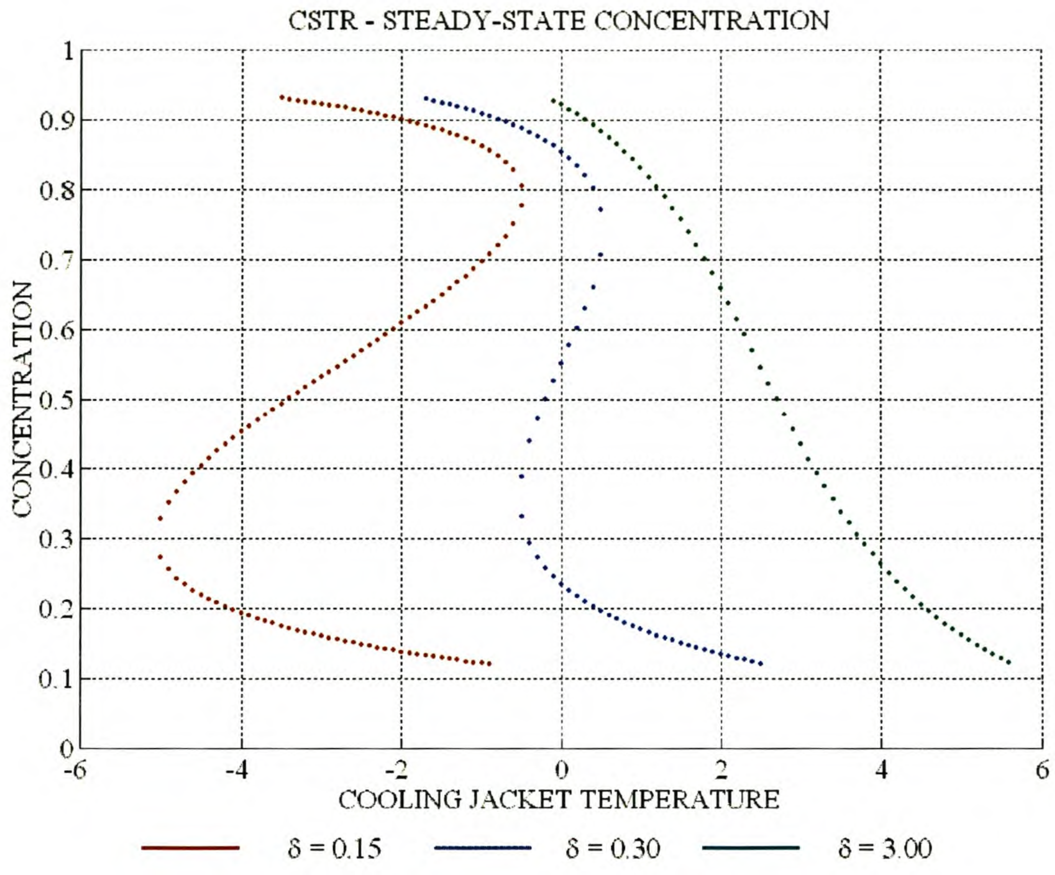

Figure 76 CSTR – Steady-state concentration.

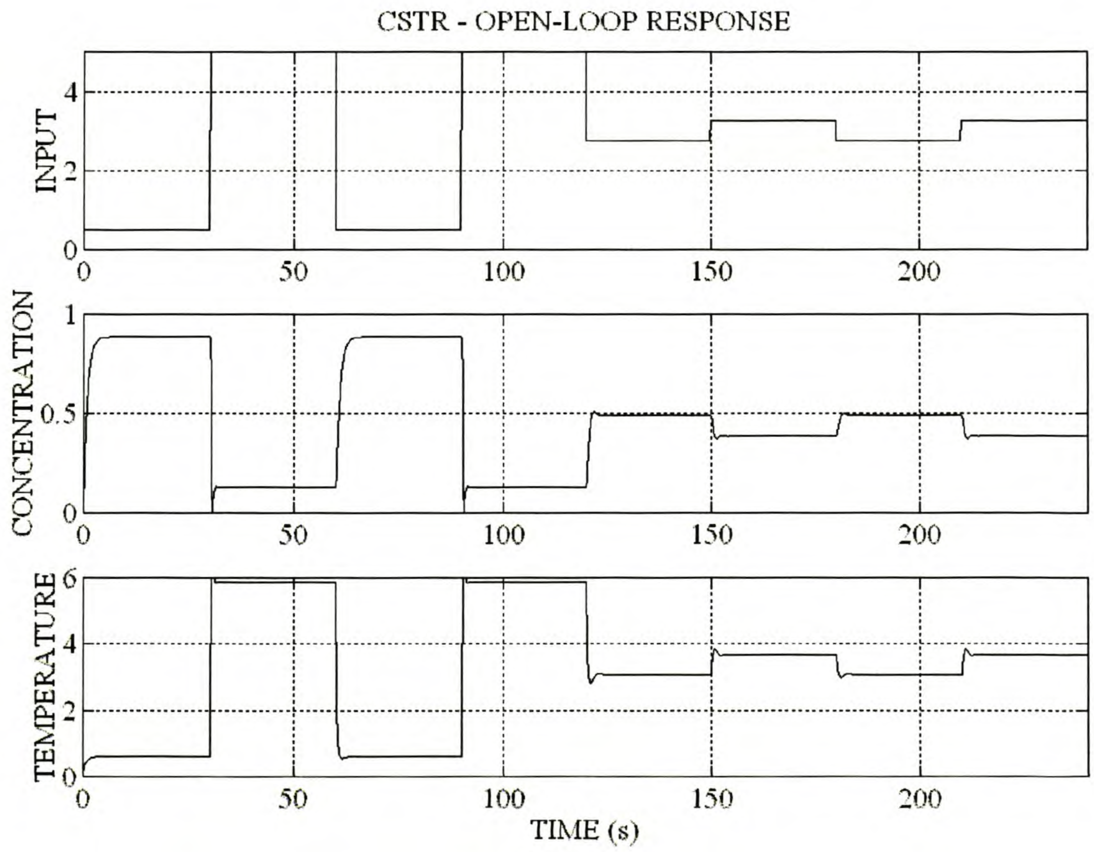

Figure 77 CSTR – Open-loop step response.

6.2 CONCENTRATION CONTROL

The control objective is to implement a stable concentration control system to track step changes in the concentration reference with a realistic closed-loop performance specification. This system is suitable for application of the self-adaptive algorithm. According to the proposed design algorithm a SMAFLC was initially tested. Although the final SMFLC showed a substantial improvement over the initial SMFLC, a MMAFLC was implemented with superior results. The design and performance results for the SMAFLC are reported in paragraph 6.2.1 and the MMAFLC in paragraph 6.2.2.

The block diagram of the SMAFLC and MMAFLC is shown in Figure 78.

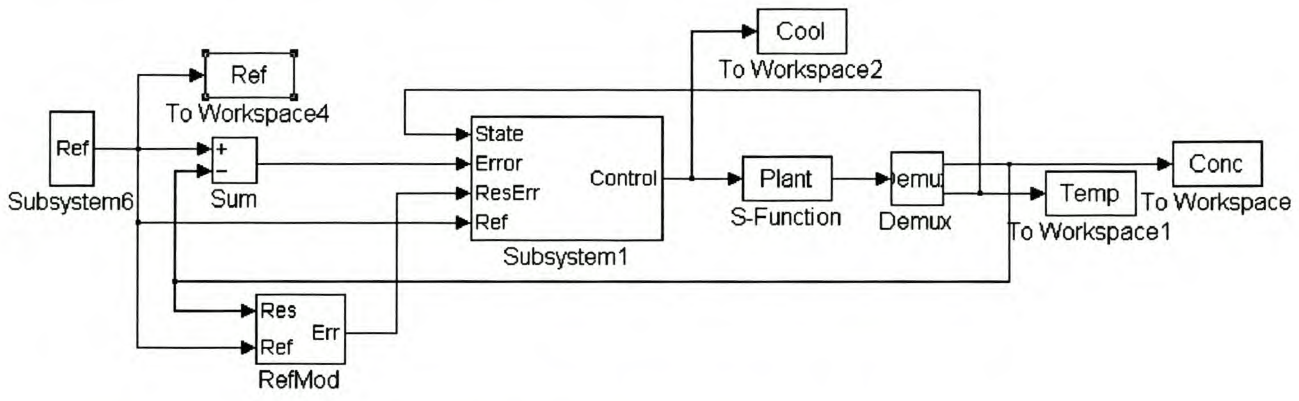

Figure 78 CSTR – Adaptive fuzzy logic control.

The MMAFLC uses the temperature for non-linear state information. Small signal step responses around a nominal control input of 1.5 indicated a plant bandwidth of $\omega_n \approx 1.7$. As a realistic closed-loop specification, the closed-loop bandwidth was chosen as 1.414, approximately 80% of the open-loop bandwidth. The reference model was chosen as the overdamped second-order system given by Equation 39.

$$G_{\mathrm{Ref}}(s) = \frac{2}{s^2 + 4s + 2}$$

Equation 39

Both controllers were subjected to the same amplitude-modulated input signal. According to the amplitude of the 0.05Hz square wave input, the simulation period can be divided into three time zones of 400s each. In zone 1 (0s – 400s) and zone 3 (800s-1200s) the reference input amplitude is 0.2, while the reference input amplitude is 0.1 in zone 2 (400s – 800s). The RMS value of the response error is used as the KPI for controller comparison.

6.2.1 SINGLE-MODE ADAPTIVE FUZZY LOGIC CONTROLLER

The automated rule-base design algorithm was used to design the initial rule-base. The universe of discourse and number of fuzzy sets for each input and output are listed in Table 15.

Input	Minimum	Maximum	Sets
State	**0**	**6**	**1**
Integral	**-1**	**1**	**9**
Error	**-1**	**1**	**9**
Rate	**-1**	**1**	**9**
Control	**-1**	**1**	**21**

Table 15 CSTR – SMFLC structure summary.

As suggested by the design algorithm, triangular membership functions define all the fuzzy sets. The gain information vector is chosen as $G_I = [1]$ and the speed information matrix, $G_S = [1]$. The controller input gains (K_i = 1.25, K_p = 1.667 and K_d = 10) were chosen to map the maximum encountered values to the normalised universes of discourse. The output gain (G_u = 3) maps the output universe of discourse to $u \in [-3, 3]$. The tuner gains were determined experimentally as K_{REp} = 0.1 and K_{REd} = 1.0. The MATLAB® design file for the SMAFLC is given in Appendix VII.II.

The closed-loop response of the SMAFLC is shown in Figure 79. The system exhibits high-frequency switching in the control signal (Figure 79 c) for lower reference values of the concentration. The bandwidth reduction of the plant limits the noise in the plant output significantly (Figure 79 a). The SMAFLC significantly improved the initial response in zone 1 and 3 (Figure 80 a and Figure 80 c) compared to the initial controller (Figure 81 a and Figure 81 c). The first step change shows a dramatic decrease in the RMS value of the response error (Figure 83 a), with a slower improvement for subsequent step changes (Figure 83 a and Figure 83 c). The final controller (Figure 82 a and (Figure 82 c) shows a reduction of 81% from 0.0964 to 0.0175 in the RMS value of the response error compared to the initial controller (Figure 81 a and Figure 81 c). In zone 2 the final controller (Figure 82 b) shows an improvement of 70% from 0.0408 to 0.0121 in the RMS value of the response error compared to the initial controller (Figure 81 b). The tuning operation converged after one step change (Figure 83 b).

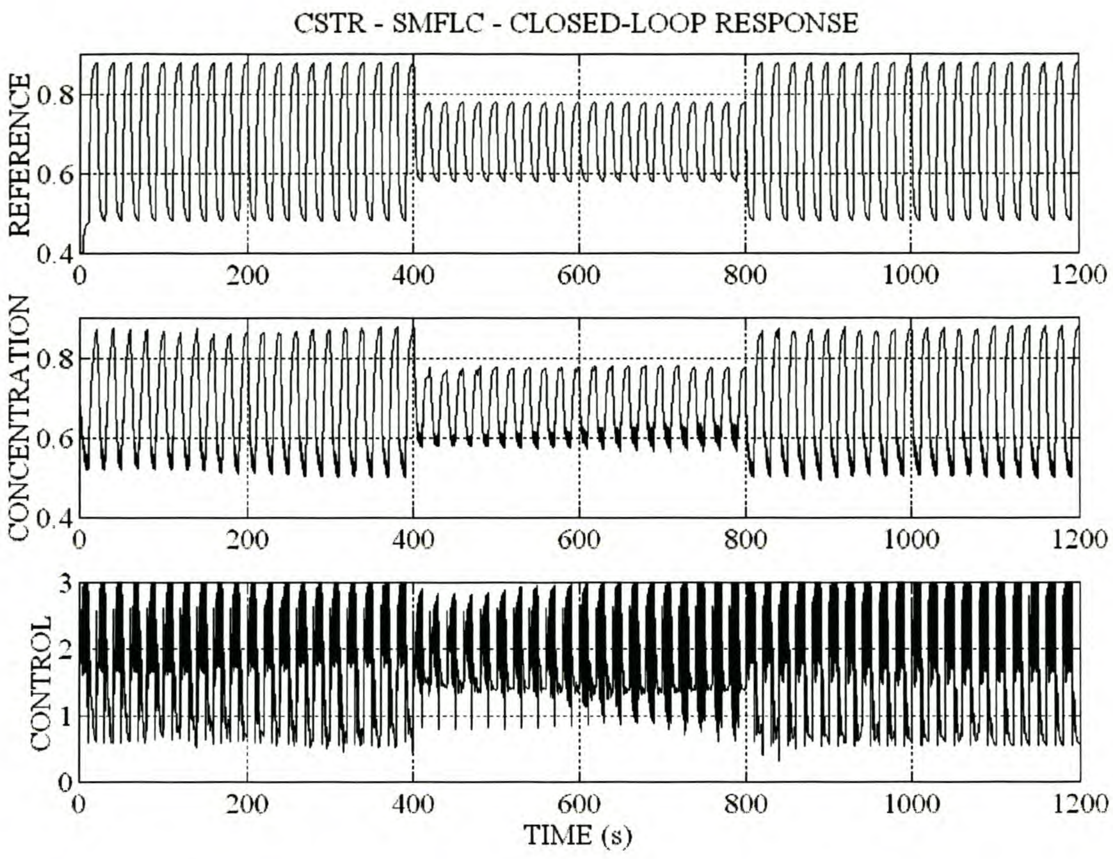

Figure 79 CSTR – SMAFLC – Closed-loop response.

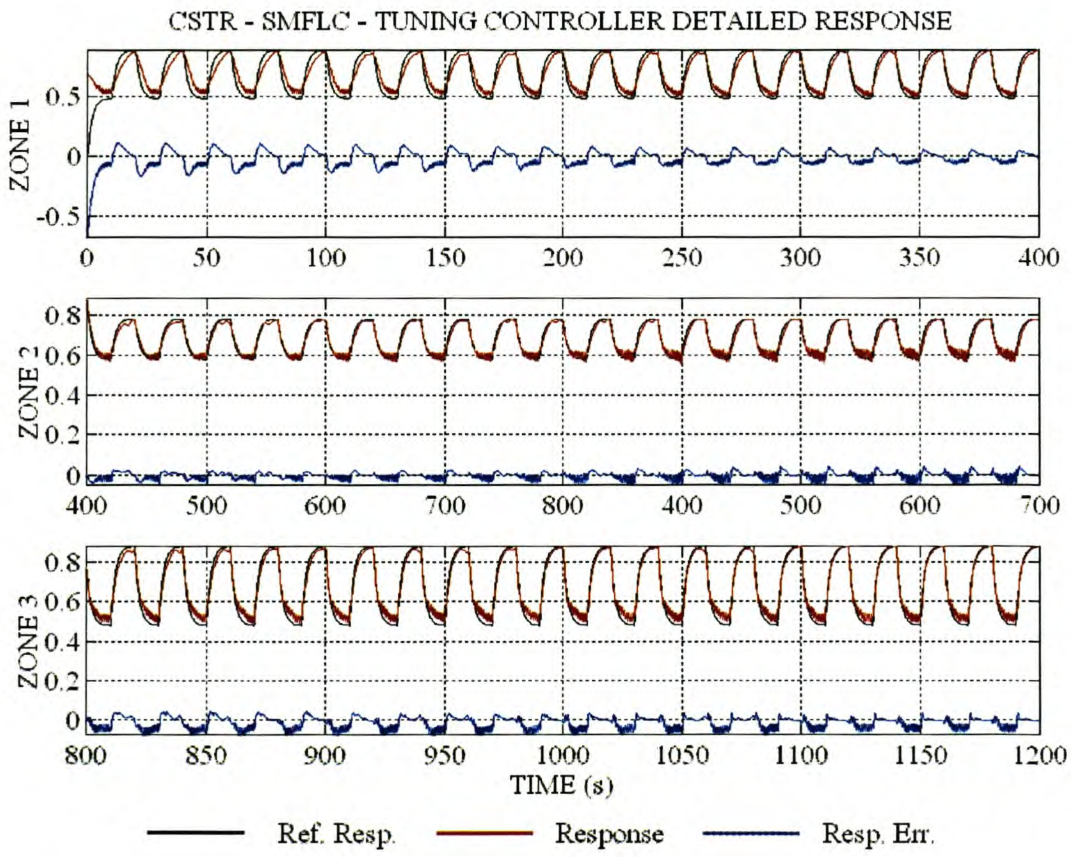

Figure 80 CSTR – SMAFLC – Tuning controller detailed response.

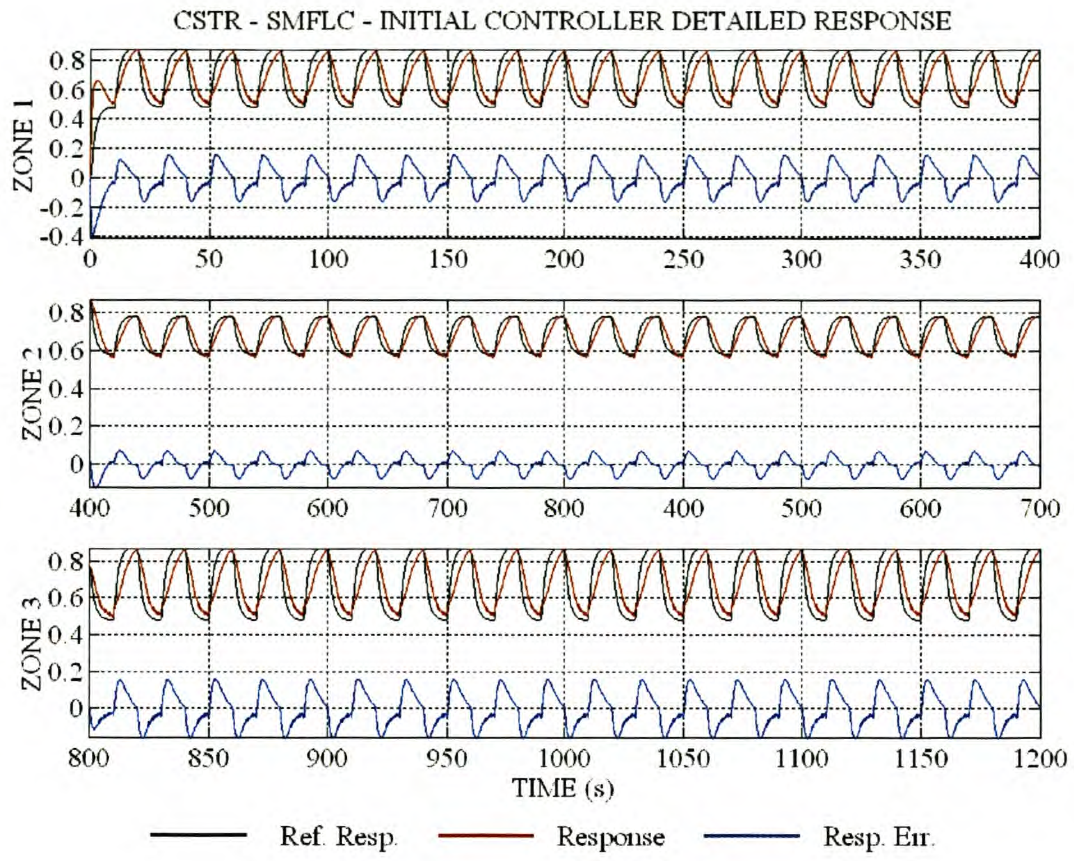

Figure 81 CSTR – SMAFLC – Initial controller detailed response.

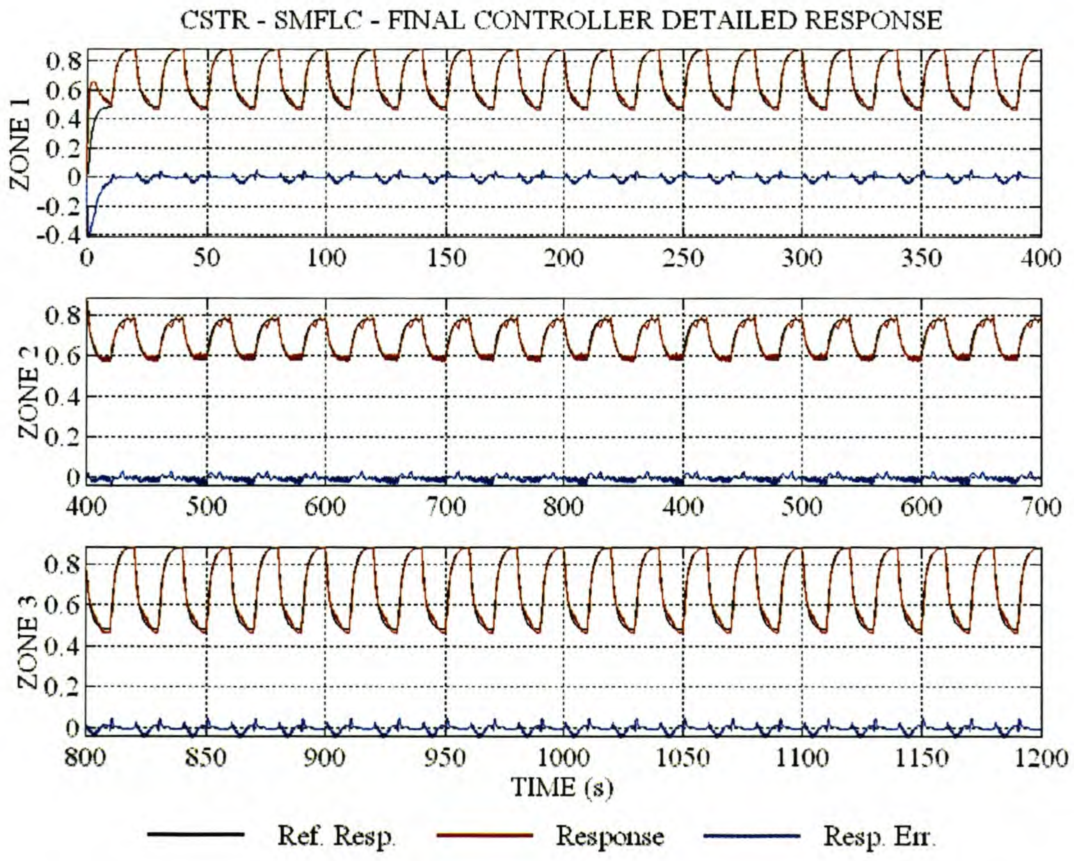

Figure 82 CSTR – SMAFLC – Final controller detailed response.

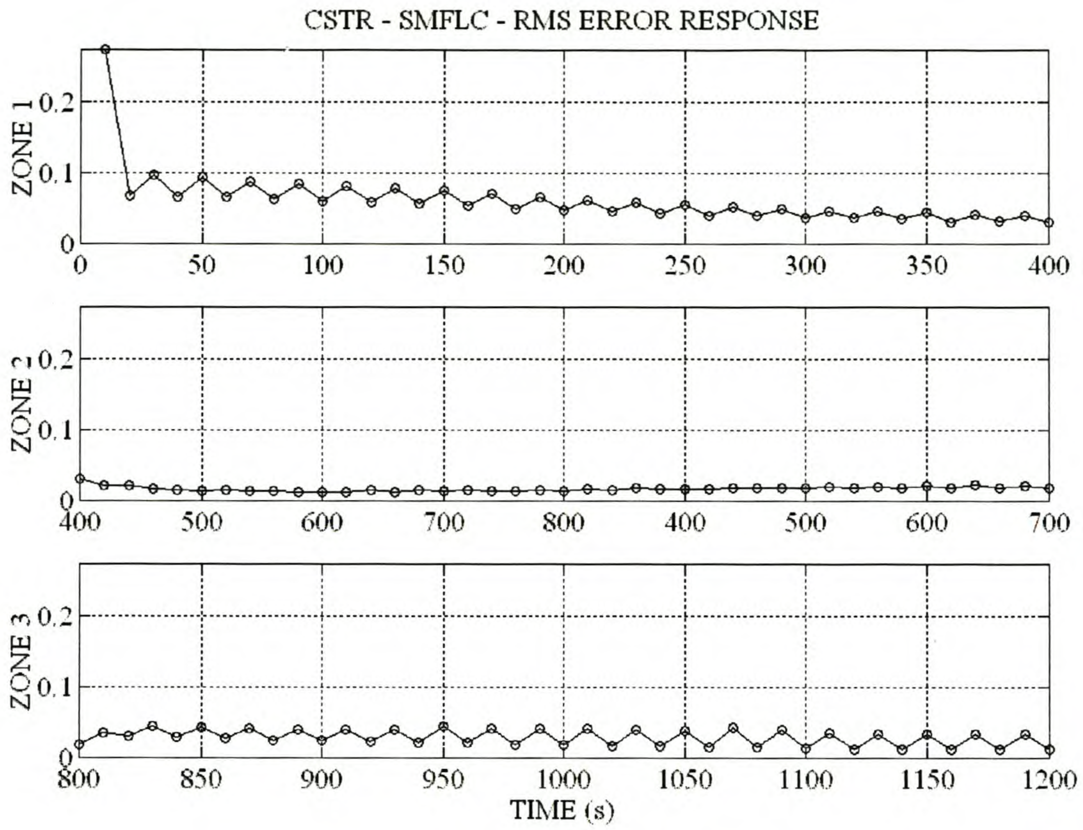

Figure 83 CSTR – SMAFLC – Response error RMS value for each step change.

6.2.2 MULTI-MODE ADAPTIVE FUZZY LOGIC CONTROLLER

The automated rule-base design algorithm is used to design the initial rule-base. Except for the three fuzzy sets defined on the state universe of discourse, the structure of the MMAFLC as given in Table 16 is identical to the SMAFLC of paragraph 6.2.1.

Input	Minimum	Maximum	Sets
State	**0**	**6**	**3**
Integral	**-1**	**1**	**9**
Error	**-1**	**1**	**9**
Rate	**-1**	**1**	**9**
Control	**-1**	**1**	**21**

Table 16 CSTR – MMFLC structure summary

The gain information vector is chosen as $G_I = [1\ 1\ 1]$, based on the almost linear steady-state temperature and concentration curves. The speed information matrix, $S_I = [0.5\ 1\ 2]$, was determined by investigating Equation 38. For high temperatures, the non-linear function K would give a larger value than with a small temperature. The amount of feedback thus increases with high temperatures. For linear systems the bandwidth increases with the feedback. The S_I vector reflects the increase in bandwidth relative to temperature as indicated. The entries in the vector were chosen arbitrarily. The controller input gains ($K_i = 1.25$, $K_p = 1.667$ and $K_d = 10$) were chosen to map the maximum encountered values to the normalized universes of discourse. The output gain ($G_u = 3$) maps the output universe of discourse to $u \in [-3, 3]$. The tuner gains were determined experimentally as $K_{REp} = 0.1$ and $K_{REd} = 1.0$. The MATLAB® design file for the MMAFLC is given in Appendix VII.III.

The closed-loop response of the SMAFLC is shown in Figure 84. The MMAFLC improved the initial response in zones 1 and 3 (Figure 85 a and Figure 85 c) significantly compared to the initial controller (Figure 86 a and Figure 86 c). The first step change shows a dramatic decrease in the RMS value of the step change (Figure 88 a) with a slower improvement for subsequent step changes (Figure 88 a and Figure 88 c). The final controller (Figure 87 a and Figure 87 c) shows a reduction of 90% (0.0986 vs. 0.0095) in the RMS value of the response error compared to the initial controller (Figure 86 a and Figure 86 c). In zone 2 the final controller (Figure 87 b) shows an improvement of 85% (0.0415 vs. 0.0058) in the RMS value of the response error, compared to the initial controller (Figure 86 b). The tuning operation converged almost instantaneously after one step change (Figure 88 b).

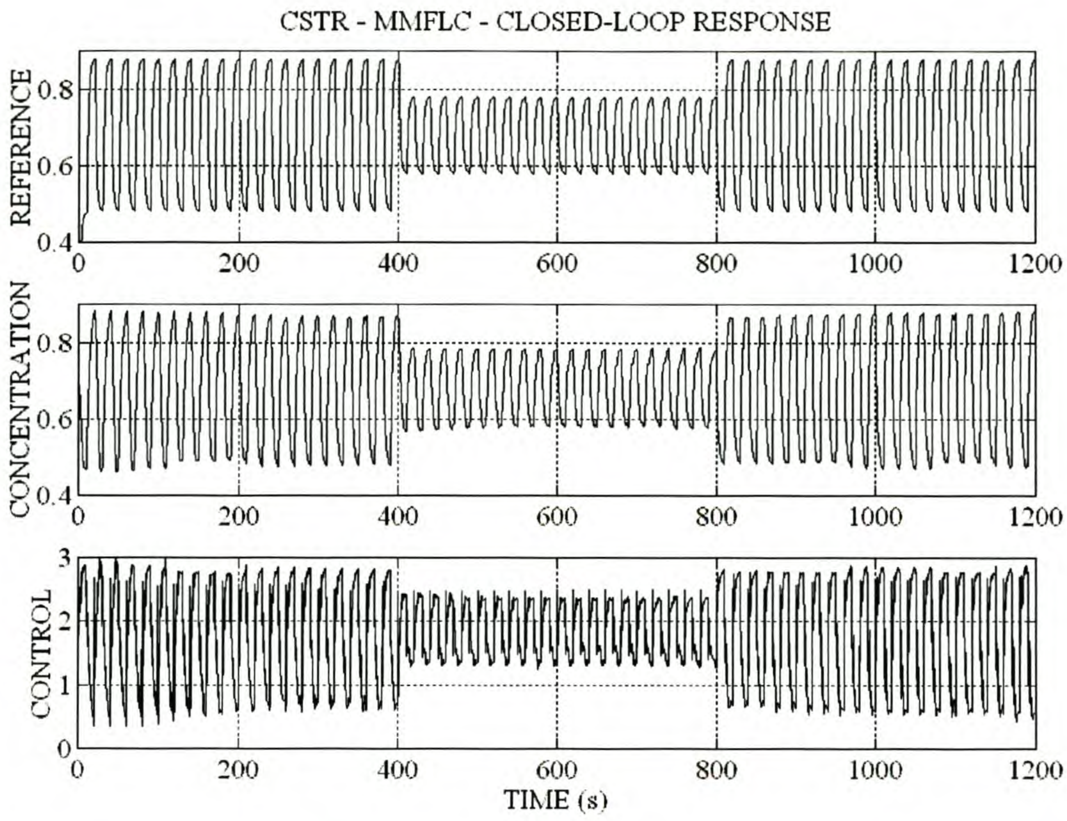

Figure 84 CSTR – MMAFLC – Closed-loop response.

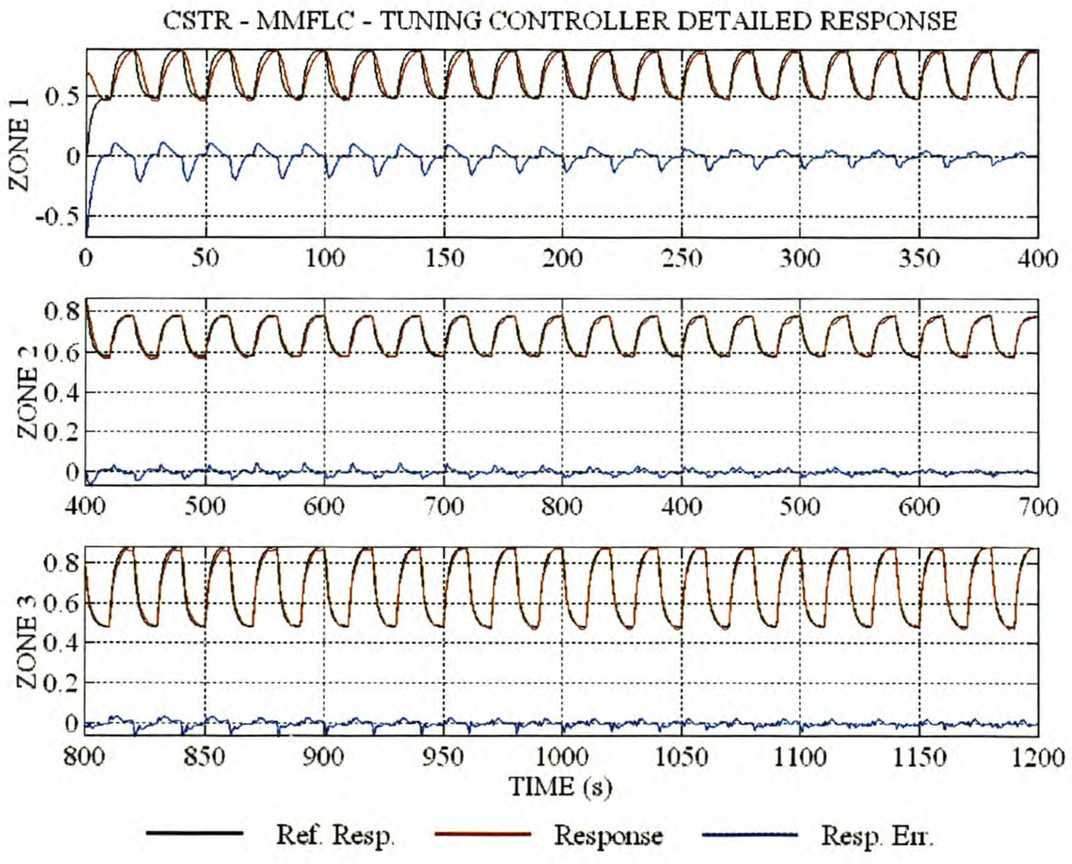

Figure 85 CSTR – MMAFLC – Tuning controller detailed response.

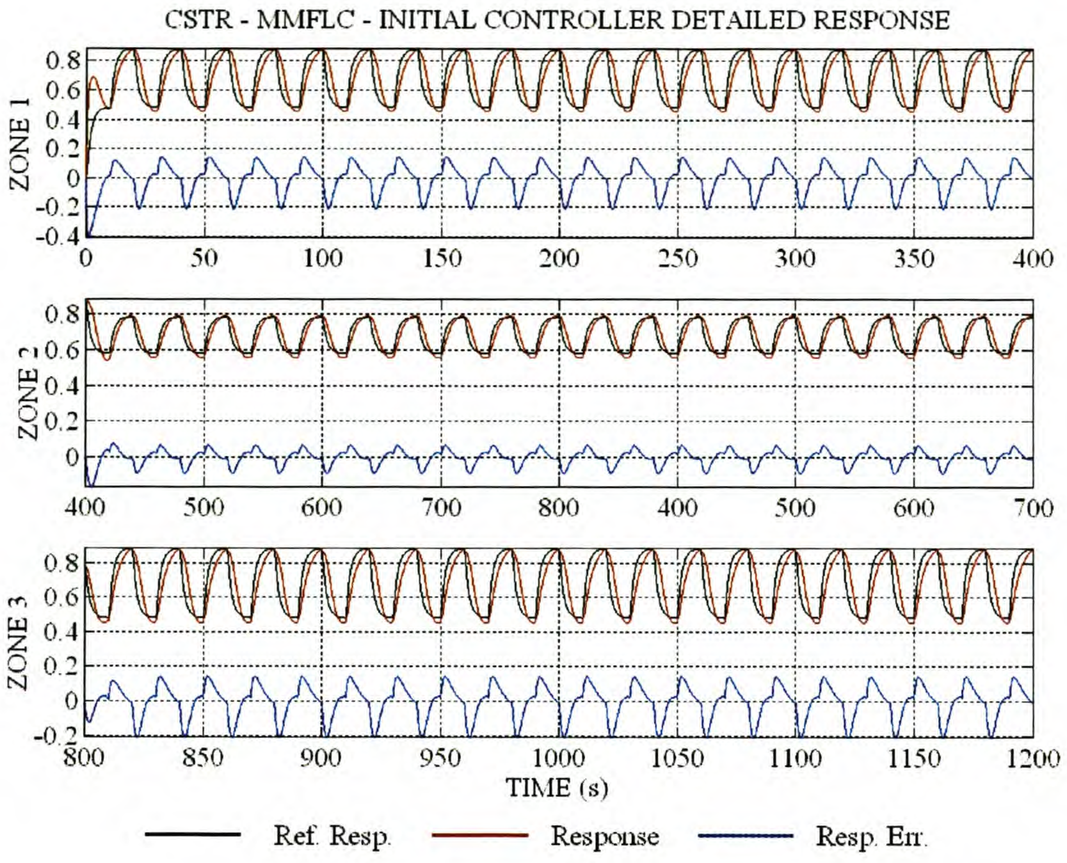

Figure 86 CSTR – MMAFLC – Initial controller detailed response.

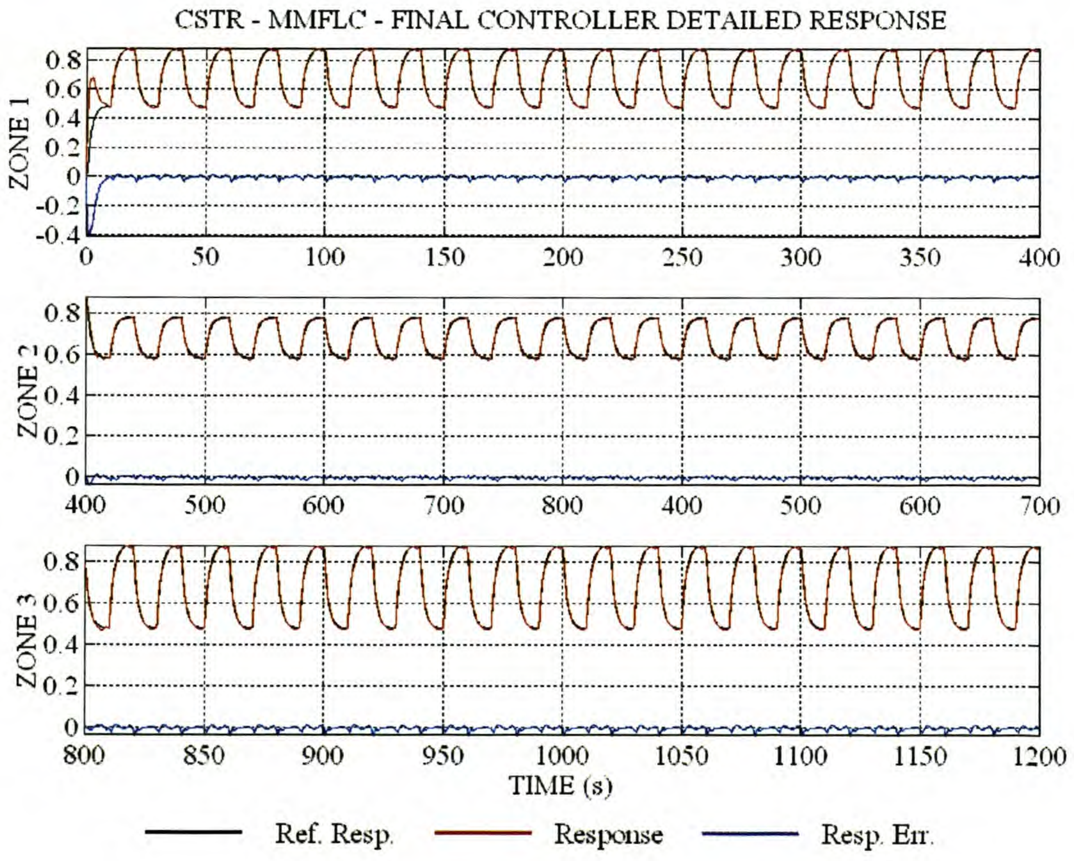

Figure 87 CSTR – MMAFLC – Final controller detailed response.

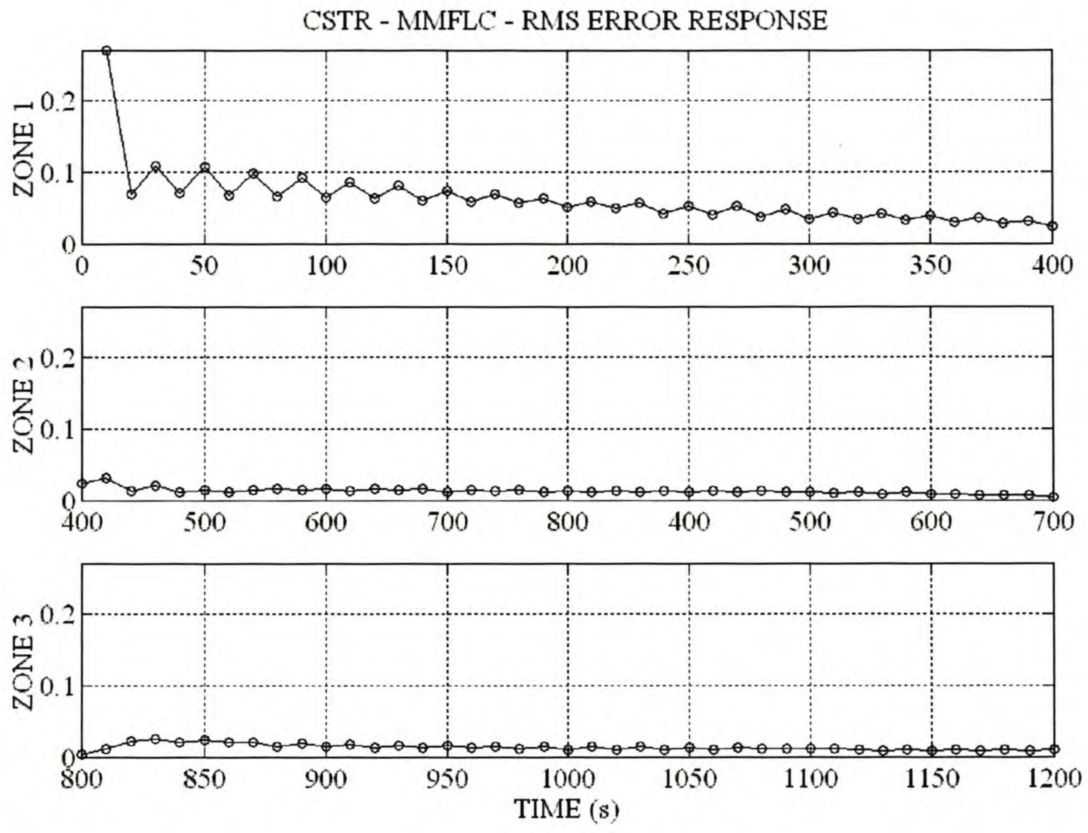

Figure 88 CSTR – MMAFLC – Response error RMS value for each step change.

6.3 Comparison and Conclusions

The RMS values of the response errors are compared in Table 17. Based on the RMS value of the response error, it was found that the final multi-mode controller performs 45% better than the single-mode controller to the large step changes of zones 1 and 3. During the small step changes of zone 2, the final multi-mode controller shows 51% better performance than the single-mode controller. The response of the final controllers to the large and small step changes is compared in Figure 89.

The final MMFLC shows better transient performance and a smaller tendency to steady-state oscillations than the final SMFLC. The added complexity of the final MMFLC is offset by the better transient response and superior reference model tracking.

Zones	SMAFLC		MMAFLC	
	Initial	Final	Initial	Final
1 (0s – 400s)	**0.09647**	**0.01754**	**0.09860**	**0.00958**
2 (400s – 800s)	**0.04089**	**0.01215**	**0.04158**	**0.00589**
3 (800s – 1200s)	**0.09647**	**0.01754**	**0.09560**	**0.00958**

Table 17 CSTR – RMS error comparison of SMAFLC and MMAFLC.

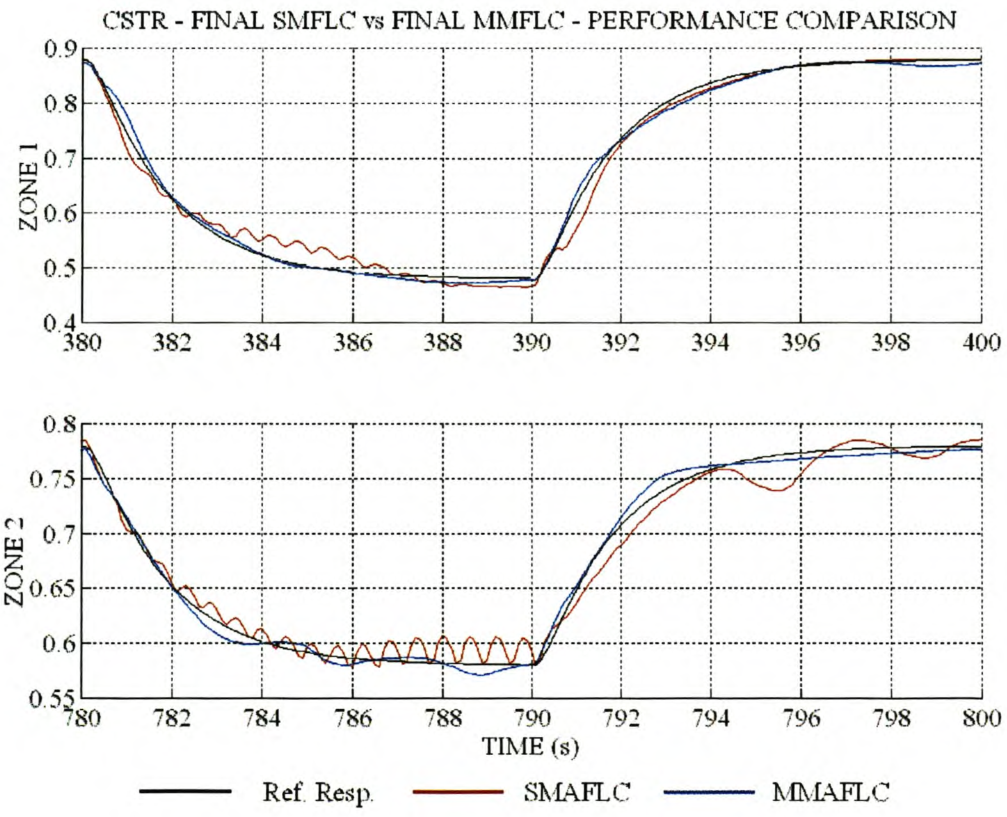

Figure 89 CSTR – Comparison of final MMAFLC vs final SMAFLC.

7 CONCLUSIONS

The closed-loop performance of a single-mode fuzzy logic controller (SMFLC) may become unacceptable if the plant dynamics change as a function of the operating conditions. The multi-mode fuzzy logic controller (MMFLC) proposed in paragraph 1.1 incorporates knowledge of the system operating conditions into the control rule-base. This allows the controller to modify its control strategy according to the operating conditions and ensures acceptable performance over the whole operating domain. The complexity of the MMFLC and the difficulties encountered in finding appropriate control rules have traditionally favoured the application of self-adaptive control techniques and led to the MMFLC being considered unsuitable for practical application. The work presented in this dissertation has shown that the MMFLC is capable of controlling complex, high-order plants with unmodelled non-linearities. The MMFLC is very effective in controlling plants with large variations in dynamics between operating domains without the relearning required by standard self-adaptive techniques. The MMFLC can also be used in cases where a SMFLC excites unacceptable limit cycles in certain operating domains.

The objective of this dissertation, stated in paragraph 1.2, was to develop automated tools for assisting the designer in the design and implementation of multi-mode rule-bases. Two methods for multi-mode rule-base design are proposed.

1. The proposed cost-function based method is suitable for application in set-point tracking and disturbance rejection problems. This method requires an iterative optimisation of the rule-base by modification of the design weights and controller input gains. Qualitative insight into the system structure is required in selecting and modifying the design weights.

2. The self-learning controller is based on the model reference adaptive architecture. This technique is suitable for applications where a stringent time domain specification exists for the closed-loop set-point response.

The five case studies investigated in this dissertation prove that the proposed automated design methods are effective in designing and optimising a MMFLC that can be implemented as look-up tables in low-cost hardware platforms.

In the dissertation it was assumed that reliable measurements are available. If the system is subjected to measurement noise, the design algorithms can be used in conjunction with a state observer or Kalman filter as shown in paragraph 5.3.1. Due to the integral action in the self-adaptive algorithm, the rule modification mechanism is insensitive to high-frequency measurement noise. The closed-loop system measurement noise robustness is similar to that of the standard Mamdani type fuzzy controller.

Process noise causing slow parameter changes modifying the system dynamics in certain domains will cause errors in the closed-loop response. In the design of controllers using the proposed methods, it is assumed that the plant is stationary. If the system uses on-line tuning, the appropriate rules will be modified to reduce the response error. If the cost-function-based design method is used, or the controller is implemented as a look-up table, redesign and implementation may be required. The robustness measure of any closed-loop system will have to be determined as required for each individual case.

As stated in paragraph 1.2, the investigation of the proposed methods for multi-loop systems is beyond the scope of this dissertation. The aim of this dissertation was to develop compensators for dynamics systems and not high level supervisory systems. In multi-loop systems, the set points of inner control loops are the manipulated variables in higher-dimensional, outer control loops. The proposed methods are applicable to the design of controllers for these higher dimensional systems. A higher level of structure knowledge may be required in selecting the parameters and states influencing the system dynamics. Process KPIs may supply valuable information in the selection of controlled variables and state information variables.

7.1 SUMMARY

Unless the plant is known to have operating domains or regions of the state space with large variations in system dynamics, a SMFLC should initially be tested due to its simplicity as a result of its smaller rule-base. The general rules of design presented below apply to both SMFLC and MMFLC.

1. *Define controller inputs and output*: Determine what type of controller is required (PD or PID). Find a measurable system state or external variable that reflects the non-linearity of the plant. Define the universe of discourse for the state input equal to its physical domain. The universes of discourse for the error integral, error and error rate of change can be chosen for simplicity as the normalised universes of discourse.

2. *Define fuzzy sets*: Based on the complexity of the plant non-linearity, choose at least seven fuzzy sets for the error integral, error and error rate of change and at least two fuzzy sets for the state input. Define the fuzzy sets membership functions and associated weights, as suggested in paragraph 3.2.

3. *Choose gain and speed information*: Choose the G_I and S_I vectors. The elements should describe the relative gain and bandwidth for each domain defined by a fuzzy set in the state input universe of discourse.

4. *Initial rule-base design*: Choose the required number of fuzzy sets in the output and run the automated design algorithm as detailed in paragraph 3.2.1.

5. *Run the self-adaptive rule-base modification algorithm*: Choose the controller input gains to map the maximum encountered values of the error integral, error and error rate of change to their normalised universes of discourse. Apply the model reference adaptive control architecture and proposed rule-base adaptive algorithm to reduce the RMS value of the closed-loop step response to an acceptable level. Add fuzzy sets to the controller inputs and outputs as required.

6. *Gain optimisation*: If it is not possible to apply the model reference adaptive control architecture, the closed-loop response must be optimised by manipulation of the controller input gains and the gain and speed information vectors (cost function weights). This will require redesigning and testing of the rule-base until an acceptable response is obtained.

The self-adaptive design algorithm, based on model reference adaptive control, has proven to be a very powerful tool in the automated design process. The case studies presented have proven that it is possible to automatically tune a rule-base, designed according to the proposed scheme, to satisfy a realistic closed-loop response time domain specification. The modification of rule consequence based on the integral of the response error has proven to be simple yet effective. Integration of the product of the response error and rule firing strength ensures that only rules participating in the control action are candidates for modification. Resetting the individual integrators after rule modification allows for a performance evaluation period for the modified rule-base. The addition of the derivative of the response error in the model reference adaptive architecture improves the general performance of the rule-base tuner by anticipating sudden changes in the response error. A rapid reduction in response error will thus delay the modification of the firing rules. Rule-base convergence can be gauged by the RMS value of the response error of successive reference step changes. In systems where it is not possible to use the model reference adaptive architecture, the closed-loop response in all the non-linear domains can be manipulated by modification of the controller input gains K_i, K_p and K_d. The performance localised to individual domains can be modified with the gain and speed information vectors. The complexity of the rule-base design problem has thus been reduced to selection of the controller input gains (K_i, K_p and K_d) and the gain and speed information vectors.

7.2 FUTURE WORK

At present the design method applies only to SISO problems conforming to the initial assumptions of paragraph 3.1. The application of the automated design method to MIMO problems is a challenging problem warranting future investigation. The addition of an algorithm for the automated determination of the gain and speed information vectors, from qualitative plant knowledge or response data, will greatly simplify the design process and increase the possible sphere of application of MMFLC. One method that seems promising to investigate is the variation in linear PID controller gains as linear plant damping, bandwidth and gain change. An expert system to modify the cost function weights based on such a study would greatly enhance the design power of the rule-base design algorithm. The robustness of the proposed design techniques for non-stationary systems in the presence of measurement noise should also be investigated.

8 REFERENCES

1. Abdelnour G.M., Chang C., Huang F., Cheung J.Y., "*Design of a Fuzzy Controller Using Input and Output Mapping Factors*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 21, no. 5, pp. 952 - 960, 1991.

2. Abe S., Lan M., "*A Method for Fuzzy Rules Extraction directly from Numerical Data and its Application to Pattern Classification*," IEEE Trans. On Fuzzy Systems, vol. 3, no. 1, pp. 18 – 28, 1995.

3. Andersen T.R., Nielsen S.B., "*An Efficient Single Output Fuzzy Control Algorithm for Adaptive Applications*," Automatica, vol. 21, no. 5, pp. 539 – 545, 1985.

4. Astrom K.J., Wittenmark B., "*Adaptive Control*," Addison Wesley Publishing Company, Reading, Massachusetts, 1995.

5. Bozenhart H.F., "*Modern control tricks solve distillation problems*," Hydrocarbon Processing, vol. 67, no. 6, pp. 47 – 50, June 1988.

6. Chen Y., "*A Self-Learning Fuzzy Controller*," Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, California, pp. 189 – 196, March 1992.

7. Chiu S., Chand S., Moore D., "*Fuzzy Logic Control of Roll and Moment for a Flexible Wing Aircraft*," IEEE Control Systems Magazine, vol. 11, no. 4, pp. 42 – 48, June 1991.

8. Coulson J.M., Richardson J.F., Backhurst J.R., Harker J.H., "*Chemical Engineer*," Vol. 2, 4th Edition, Pergamon Press, Oxford, 1991

9. Cox E., "*Adaptive Fuzzy Systems*," IEEE Spectrum, vol. 30, no. 2, pp. 27 – 31, February 1993.

10. Cox E., "*Fuzzy Fundamentals*," IEEE Spectrum, vol. 29, no. 10, pp.58 – 61. October 1992.

11. Fonyo Z., "*Design Modifications and Proper Plantwide Control*," Computers in Chemical Engineering, vol. 18, Suppl., pp. S483 – S492, 1994.

12. Franklin G.F., Powell J.D., Workman M.L., "*Digital Control of Dynamic Systems*," Addison Wesley Publishing Company, Reading, Massachusetts, 2nd Ed., 1990.

13. Galichet S., Foulloy L., "*Fuzzy Controllers: Syndissertation and Equivalence*," IEEE Transactions on Fuzzy Systems, Vol. 3, No. 2, pp. 140 –1148, 1995.

14. Glower J.S., Munighan J., "*Designing Fuzzy Controllers from a Variable Structure Standpoint*," IEEE Trans. on Fuzzy Systems, vol. 5, no. 1, pp. 138 – 144, 1997.

15. Gokhale V., Hurowitz S., Riggs J.B., "*A Comparison of Advanced Distillation Control Techniques for a Propylene/Propane Splitter*," Industrial Engineering and Chemical Research, vol. 34, no. 12, pp. 4413 – 4419, 1995.

16. Heckenthaler T., Sebastian E., "*Approximately Time-Optimal Fuzzy Control of a Two-Tank System*," IEEE Control Systems Magazine, vol. 14, no. 3, pp. 24 – 31, June 1994.

17. Huang L., Tomizuka M., "*A Self-Paced Fuzzy Tracking Controller for Two-Dimensional Motion Control*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 20, no. 5, pp. 1115 - 1124, 1990.

18. Ischibuchi H., Nozaki K., Yamamoto N., Tanaka H., "*Selecting Fuzzy If-Then Rules for Classification Problems Using Genetic Algorithms*," IEEE Trans. On Fuzzy Systems, vol. 3, no. 3, pp. 260 – 270, 1995.

19. Jagannathan S., "*Adaptive Fuzzy Logic Control of Feedback Linearizable Discrete-Time Dynamical Systems under Persistence of Excitation*," Automatica, Vol. 34, No. 11, pp. 1295 – 1310, 1998.

20. Jang. J.S.R., Gulley N., "*Fuzzy Logic Toolbox, For use with MATLAB®*," The MathWorks Inc., Natick, Mass., 1995.

21. Jantzen J., "*Fuzzy Control*," Lecture Notes in On-Line Process Control (5354), Publ. no. 9109, Technical University of Denmark, Lynby, 1991.

22. Jouffe L., "*Fuzzy Inference Systems Learning by Reinforcement Methods*," IEEE Trans on Man, Systems and Cybernetics, vol. 28 no. 3, pp. 338 – 355, 1998.

23. Kang H., Vachtsevanos G., "*Adaptive Fuzzy Logic Control: Explicit Adaptive Control with Lyapunov Stability and Learning Capability*," Proceedings of the IFAC American Control Conference, vol. 3, Chicago, USA, pp. 2279 - 2283, June 1992.

24. Karr C.L., Gentry E.J., "*Fuzzy Control of pH Using genetic Algorithms*," IEEE Trans. On Fuzzy Systems, vol. 1, no. 1, pp. 46 – 53, 1993.

25. Kawaji S., Matanaga N., "*Fuzzy Control of VSS Type and its Robustness*," Proceedings of the IFSA World Congress, Brussels, pp. 81 – 84, 1991.

26. Kickert W.J.M., Van Nauta Lemke H.R., "*Application of a Fuzzy Controller in a Warm Water Plant*," Automatica, vol. 12, no. 4, pp. 301 – 308, 1976.

27. Kim J., Moon Y., Ziegler B.P., "D*esigning Fuzzy Net Controllers Using Genetic Algorithms*," IEEE Control Systems Magazine, vol. 15, no. 3, pp. 66 – 72, June 1995.

28. Kim J., Ziegler B.P., "*Hierarchical Distributed Genetic Algorithms: A Fuzzy Logic Controller Design Application*," IEEE Expert Intelligent Systems & their Applications, vol. 11, no. 3, pp. 76 – 84, June 1996.

29. Kim Y., Kim M., "*Experimental Application of Combined Fuzzy and Predictive Control to a Binary Distillation Column*," Journal of Chemical Engineering of Japan, vol. 28, no. 5, pp. 617 – 620, 1995.

30. King P.J., Mamdani E.H., "*The Application of Fuzzy Control Systems to Industrial Processes*," Automatica, vol. 13, no. 2, pp. 235 – 242, 1977.

31. Kovacic Z., Balenovic M., Bogdan S., "*Sensitivity-Based Self-Learning Fuzzy Logic Control for a Servo System*," IEEE Control Systems Magazine, vol. 18, no. 3, pp. 41 – 51., June 1998.

32. Krishnapuram R., Frigui H., Nasaoui O., "*Fuzzy and Possibilistic Shell Clustering Algorithms and their Application to Boundary Detection and Surface Approximation – Part I*," IEEE Trans. On Fuzzy Systems, vol. 3, no. 1, pp. 29 – 43, 1995.

33. Krishnapuram R., Frigui H., Nasaoui O., "*Fuzzy and Possibilistic Shell Clustering Algorithms and their Application to Boundary Detection and Surface Approximation – Part II*," IEEE Trans. On Fuzzy Systems, vol. 3, no. 1, pp. 44 – 60, 1995.

34. Langari R., '*A Nonlinear Formulation of a Class of Fuzzy Linguistic Control Algorithms,*" Proceedings of the IFAC American Control Conference, vol. 3, Chicago, USA, pp. 2273 - 2278, June 1992.

35. Layne J.R., Passino K.M., "*Fuzzy Model Reference Learning Control for Cargo Ship Steering*," IEEE Control Systems Magazine, vol. 13, no. 5, pp. 23 – 34., Dec. 1993.

36. Lee C.C., "*Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part 1*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 20, no. 2, pp. 404 - 418, 1990.

37. Lee C.C., "*Fuzzy Logic in Control Systems: Fuzzy logic Controller – Part 2*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 20, no. 2, pp. 419 - 438, 1990.

38. Lee J., Sunwon P., "*Robust nonlinear predictive control using a disturbance estimator*," Chem. Eng. Comm., vol. 128, pp. 43 – 64, 1994.

39. Lee J.H., Kesavan P., Morari M., "*Control Structure Selection and Robust Control System Design for a High-Purity Distillation Column*," IEEE Trans. on Control System Technology, vol. 5, no. 4, pp. 402 – 416, 1997.

40. Li Y.F., Lau C.C., "*Development of Fuzzy Algorithms for Servo Systems*," IEEE Control Systems Magazine, vol. 9, no. 3, pp. 65 – 71, April 1989.

41. Limqueco L.C., Kantor J.C., "*Nonlinear output feedback control of an exothermic reactor*," Computers Chem. Engng., vol. 14, no. 4/5, pp. 427 – 437, 1990.

42. Lin C., Lee C.S.G., "*Neural Fuzzy System*," Prentice-Hall P T R, Upper Saddle River , NJ, 1996.

43. Lo J., Kuo Y., "*Decoupled Fuzzy Sliding Mode Control*," IEEE Trans. on Fuzzy Systems, vol. 6, no. 3, pp. 426 – 435, 1998.

44. Ma X., Sun Z., He Y., "*Analysis and Design of Fuzzy Controller and Fuzzy Observer*," IEEE Trans. on Fuzzy Systems, vol. 6, no. 1, pp. 41 – 51, 1998.

45. Maiers J., Sherif Y.S., "*Applications of Fuzzy Set Theory*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 15, no. 1, pp. 175 - 189, 1985.

46. Mamdani E.H., "*Application of Fuzzy Algorithms for Control of Simple Dynamic Plant*," Proc. IEE, Vol. 121, No. 12, pp. 1585 – 1588, 1974.

47. Mamdani E.H., "*Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Syndissertation*," IEEE Trans. On Computers, vol. C 26, no. 12, pp. 1182 – 1191, 1977.

48. Moudgal V.G., Kwong W.A., Passino K.M., Yorkovich S., "*Fuzzy Learning Control for a Flexible-Link Robot*," IEEE Trans. On Fuzzy Systems, vol. 3, no. 2, pp. 199 – 210, 1995.

49. Musch H.E., Steiner M., "*Robust PID Control for an Industrial Distillation Column*," IEEE Control Systems Magazine, vol. 15, no. 4, pp. 46 – 55, August 1995.

50. Ogata K. "*Modern Control Engineering*," Prentice-Hall Inc., Englewood Cliffs, N.J., 2nd. Ed., 1990.

51. Ordonez R., Zumberge J., Spooner J.T., Passino K.M., "*Adaptive Fuzzy Control: Experiments and Comparative Analyses*," IEEE Trans. on Fuzzy Systems, vol. 5, no. 2, pp. 167 – 188, 1997.

52. Palm R., "*Sliding Mode Fuzzy Control*," Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, California, pp. 519 – 526, March 1992.

53. Park Y., Moon U., Kwang Y., Lee Y., "*A Self-Organizing Fuzzy Logic Controller for Dynamic Systems Using a Fuzzy Auto-Regressive Moving Average (FARMA) Model*," IEEE Trans. On Fuzzy Systems, vol. 3, no. 1, pp. 75 – 82, 1995.

54. Passino K.M., Yurkovich S., "*Fuzzy Control*," Addison Wesley, Menlo Park, California, 1997.

55. Procyk T.J., Mamdani E.H., "*A Linguistic Self-Organizing Process Controller*," Automatica, vol. 15, no. 1, pp. 15 – 30, 1979.

56. Rovaglio M., Faravelli T., Biardi G., Gafurri P, Soccol S., "*The key role of entrainer inventory control of heterogeneous azeotropic distillation towers*," Computers in Chemical Engineering, vol. 17, no. 5/6, pp. 535-547, 1993.

57. Rovaglio M., Faravelli T., Biardi G., Gafurri P., Soccol S., Natta G., "*Precise Composition Control of Heterogeneous Azeotropic Distillation Towers*," Supplement to Computers and Chemical Engineering, vol. 16, no. S, pp. S181 – S188, 1992.

58. Rovaglio M., Faravelli T., Gaffuri P., Di Palo C., Doriga A., "*Controllability and Operability of Azeotropic Heterogeneous Distillation Systems*," Computers in Chemical Engineering, vol. 19, Suppl., pp. S525 – S530, 1995.

59. Rovaglio M., Ranzi E., Biardi M., Fontana M., Domenichini R., "*Rigorous Dynamics and Feedforward Control Design for Distillation Processes*," AIChE Journal, vol. 36, no. 4., pp. 576-586, 1990.

60. Schwartz D.G., Klir G.J., "*Fuzzy Logic Flowers in Japan*," IEEE Spectrum, vol. 29, no. 7, pp.32 – 35. July 1992.

61. Self K., "*Designing with Fuzzy Logic*," IEEE Spectrum, vol. 27, no. 11, pp. 42 – 44 & 105, November 1990.

62. Shao S., "*Fuzzy Self-Organizing Controller and its Application for Dynamic Processes*," Fuzzy Sets and Systems, vol. 26, no. 2, Elsevier Science Publishers B.V., North Holland, pp. 151 – 164, 1988.

63. Shaocheng T., Chai T., Cheng S., "*Adaptive Fuzzy Sliding Mode Control for Nonlinear Systems*" Proceedings of the fifth IEEE International Conference on Fuzzy Systems, Vol. 1, New Orleans, Louisiana, pp. 49 – 54, 1996.

64. Shenoi S., Ashenayi K., Timmerman.M., "*Implementation of a Learning Fuzzy Controller*," IEEE Control Systems Magazine, vol. 15, no. 3, pp.73 – 80, June 1995.

65. Sheridan S.E., Skjoth P., "*Automatic Kiln Control at Oregon Portland Cement Company's Durkee Plant Utilizing Fuzzy Logic*," IEEE Trans. On Industry Applications, vol. IA 20, no. 3, pp. 562 – 568, 1984.

66. Shinskey F.G., "*Distillation Control. For Productivity and Energy Conservation*," McGraw-Hill, New York, 1977.

67. Skogestad S., Lundström P., Jacobsen E.W., "*Selecting the Best Distillation Control Configuration*," AIChE Journal, vol. 36, no. 5, pp. 753-764, 1990.

68. Slotine J.E., "*Applied Nonlinear Control*," Prentice Hall Inc. Englewood Cliffs, NJ, 1991.

69. Smith S.M., Corner D.J., "*Automated Calibration of a Fuzzy Logic Controller Using a Cell State Space Algorithm*," IEEE Control Systems Magazine, vol. 11, no. 5, pp. 18 – 28, August 1991.

70. Sousa G.C.D., Bose B.K., "*A Fuzzy Set Theory Based Control of a Phase-Controlled Converter DC Machine Drive*," IEEE Trans. On Industrial Application, vol. IA 30, no. 1, pp. 34 – 44, 1994.

71. Stenz R., Kuhn U., "*Automation of a Batch Distillation Column Using Fuzzy and Conventional Control*," IEEE Trans. on Control Systems Technology, vol. 3, no. 2, pp. 171 – 176, 1995.

72. Su C., Stepanenko Y., "*Adaptive Control of a Class of Nonlinear Systems with Fuzzy Logic*," IEEE Trans. on Fuzzy Systems, vol. 2, no. 4, pp. 285 – 294, 1994.

73. Sugeno M., Nishida M., "*Fuzzy Control of Model Car*," Fuzzy Sets and Systems, vol. 16, no. 2, Elsevier Science Publishers B.V., North Holland, pp. 103 –113, 1985.

74. Sugeno M., Yasukawa T., "*A Fuzzy-Logic-Based Approach to Qualitative Modeling*," IEEE Trans. On Fuzzy Systems, vol. 1, no. 1, pp. 7 – 31, 1993.

75. Sugiyama K., "*Rule-Based Self-Organizing Controller*," Fuzzy Computing, Elsevier Science Publishers B.V., North Holland, pp. 341 - 353, 1988.

76. Takagi T., Sugeno M., "*Fuzzy Identification of Systems and its Applications to Modeling and Control*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 13, no. 1, pp. 116 - 132, 1985.

77. Tang K.L., Mulholland R.J., "*Comparing Fuzzy Logic with Classical Controller Designs*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 17, no. 6, pp. 1085 - 1087, 1987.

78. Tanscheit R., Scharf E.M., "*Experiments with the use of a Rule-Based Self-Organizing Controller for Robotics Applications*," Fuzzy Sets and Systems, vol. 26, no. 2, Elsevier Science Publishers B.V., North Holland, pp. 195 – 214, 1988.

79. Ting C., Li T., Kung F., "*An Approach to Systematic Design of the Fuzzy Control System*," Fuzzy Sets and Systems, vol. 77, no. 2, Elsevier Science Publishers B.V., North Holland, pp. 151 –166, 1996.

80. Tong R.M., "*A Control Engineering Review of Fuzzy Systems*," Automatica, vol. 13, no. 6, pp. 559 – 569, 1977.

81. Uppal A., Ray W.H., "*On the dynamic behavior of continuous stirred tank reactors*," Chemical Engineering Science, vol. 29, pp 967 – 985, 1974.

82. Utkin V.E., "*Sliding Modes and their Application in Variable Structure Systems*," MIR Publishers, Moskow, 1978.

83. Vachtsevanos G., Farinwata S.S., Pirovolou D.K., "*Fuzzy Logic Control of an Automotive Engine*," IEEE Control Systems Magazine, vol. 13, no. 3, pp. 62 – 67, June 1993.

84. Van Buijtenen W.A., Schram G., Babuska R., Verbruggen H.B., "*Adaptive Fuzzy Control of Satellite Attitude by Reinforcement Learning*," IEEE Trans. on Fuzzy Systems, Vol. 6, No. 2, pp. 185 – 194, 1998.

85. Vaneck T.W., "*Fuzzy Guidance Control for an Autonomous Boat*," IEEE Control Systems Magazine, vol. 17, no. 2, pp. 43 – 51., April 1997.

86. Von Altrock C., Krause B., Zimmerman H.J., "*Advanced Fuzzy Logic Control Technologies in Automotive Applications*," Proceedings of the IEEE International Conference on Fuzzy Systems, San Diego, California, pp. 835 – 842, March 1992.

87. Wang L., "*A Course in Fuzzy Systems and Control*," Prentice-Hall P T R, Upper Saddle River, NJ, 1997.

88. Wang L., "*Stable Adaptive Fuzzy Control of Non Linear Systems*," IEEE Transactions on Fuzzy Systems, Vol. 1, No. 2, pp, 146 –155, 1993.

89. Wang L., "*Adaptive Fuzzy Systems and Control*," Prentice-Hall P T R, Englewood Cliffs, NJ, 1994.

90. Wong D.S.H., Jang S.S., Chang C.F., "*Simulation of dynamics and phase pattern changes for an azeotropic distillation column*," Computers in Chemical Engineering, vol. 15, no. 5, pp. 325 – 335, 1991.

91. Wu W., Ko J., Lee H., "*Decoupling Control of a Multivariable System with a Desensitizer*," Industrial Engineering and Chemical Research, vol. 32, no. 11, pp. 2937 – 2941, 1993.

92. Xu C., Lu Y., "*Fuzzy Model Identification and Self-Learning for Dynamic Systems*," IEEE Trans. On Systems, Man and Cybernetics, vol. SMC 17, no. 4, pp. 683 - 689, 1987.

93. Yager R.R., Filev D.P., "*SLIDE: A Simple Adaptive Defuzzification Method*," IEEE Transactions on Fuzzy Systems, Vol. 1, No. 1, pp, 69 –78, 1993.

94. Ying H., Siler W., Buckley J.J., "*Fuzzy Control Theory: A Nonlinear Case,*" Automatica, vol. 26, no. 3, pp. 513 – 520, 1990.

95. Zadeh L.A., "*Outline of a New Approuch to the Analysis of Complex Systems and Decision Processes,*" IEEE Trans on Man, Systems and Cybernetics, vol. 3 no. 1, pp. 28 – 44, 1973.

96. Zeng X., Singh M., "*Approximation Theory of Fuzzy Systems – MIMO Case,*" IEEE Transactions on Fuzzy Systems, Vol. 3, No. 2, pp, 219 –235, 1995.

97. Zhao C., Whiteley J.R., Misawa E.A., Gasem K.A.M., "*Application of Enhanced LQG/LTR for Distillation Control,*" IEEE Control Systems Magazine, vol. 15, no. 4, pp. 56 – 63, August 1995.

98. Zhao Z., Tomizuka M., Sagara S., "*A Fuzzy Tuner for Fuzzy Logic Controllers,*" Proceedings of the IFAC American Control Conference, vol. 3, Chicago, USA, pp. 2268 - 2273, June 1992.

I RULE-BASE GENERATION ALGORITHM

The MATLAB® file listing of the automated rule-base design algorithm based on the linear cost function Approuch is detailed in this Appendix. The inputs and outputs of the function are detailed in the help section of the file.

```
function  [Fuzzy_System,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] =
pidrbgen(Fuzzy_System,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims)
%  function  [Fuzzy_System,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] =
pidrbgen(Fuzzy_System,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims)
%
%       PIDRBGEN.M
%       This function generates a general PID rule-base for a fuzzy control system using the
%       proposed method.
%
%       Fuzzy_System    -       Output:  Complete fuzzy system.
%       RuleBase        -       Output:  Rule-base.  Before clustering
%       RuleBaseClusters        -       Output:  Rule-base.  After clustering
%       RuleBaseNorm    -       Output:  Rule-base.  Normalized
%       RBM     -       Output:  Rule-base Matrix.  As used in standard MATLAB® format
%       Centres -       Output:  Centres of clusters
%       Fuzzy_System    -       Input:  Fuzzy system.
%       Gain    -       Input:  Gain information vector.
%       Speed   -       Input:  Speed information vector.
%       Error_Weight    -       Input:  Weight vector.
%       Rate_Weight     -       Input:  Weight vector.
%       Int_Weight      -       Input:  Weight vector
%       Dims    -       Input:  Vector defining rule-base dimensions.
%
%       Etienne M.Hugo - 90 1969 3
%       Process Control Group - University of Stellenbosch
%       09/04/98
%  Modifications
%  Ver 1.0:    22/02/1999:    Modify to work with arbitrary dims
%  Ver 1.2: 03/05/1999: Modify the speed and gain wieght calculations to be normalised;
%  Ver 1.3: 23/06/1999: Modify the initial speed and gain weight normalization
%  Declare the required variables
State_Num = Dims(1);
Err_Int_Num = Dims(2);
Err_Num = Dims(3);
Err_Der_Num = Dims(4);
U_Num = Dims(5);
Gain_Weight = zeros(1,State_Num);
Speed_Weight = zeros(1,State_Num);
U_MBF_Num = zeros(1,U_Num);
Data = zeros(State_Num*Err_Int_Num*Err_Num,Err_Der_Num,1);
RuleBase = zeros(State_Num*Err_Int_Num*Err_Num,Err_Der_Num);
RuleBaseClusters = zeros(State_Num*Err_Int_Num*Err_Num,Err_Der_Num);
RuleBaseNorm = zeros(State_Num*Err_Int_Num*Err_Num,Err_Der_Num);
RBM = zeros(State_Num*Err_Int_Num*Err_Num*Err_Der_Num,7);
%  Determine the fuzzy rules
Min_Gain = min(Gain);
Max_Speed = max(Speed);
for q = 1:State_Num
   Gain_Weight(q) = (Min_Gain/Gain(q));
   Speed_Weight(q) = (Speed(q)/Max_Speed);
end
for q = 1:State_Num
   qq = q-1;
   for t = 1:Err_Int_Num
      tt = t-1;
      for w = 1:Err_Num
         for e = 1:Err_Der_Num
            r = qq*Err_Int_Num*Err_Num+tt*Err_Num+w;
            RuleBase(r,e) = Gain_Weight(q)*Error_Weight(w)+Rate_Weight(e)/Speed_Weight(q);
         end
      end
   end
end
```

```
Max_Control_Weight = max(max(abs(RuleBase)));
Max_Int_Weight = max(abs(Int_Weight));
Int_Weight = Max_Control_Weight * Int_Weight / Max_Int_Weight;
for q = 1:State_Num
   qq = q-1;
   for t = 1:Err_Int_Num
      tt = t-1;
      for w = 1:Err_Num
         for e = 1:Err_Der_Num
            r = qq*Err_Int_Num*Err_Num+tt*Err_Num+w;
            Aug = (1 - abs(RuleBase(r,e))/Max_Control_Weight)*(Int_Weight(t));
            RuleBase(r,e) = RuleBase(r,e) + Aug;
         end
      end
   end
end
%  Reduce the rule-base by using fuzzy c means clustering
%  Firts cluster the negative section and then the positive section.
t = 1;
for q = 1:State_Num*Err_Int_Num*Err_Num
   for w = 1:Err_Der_Num
      Data(t,1) = RuleBase(q,w);
      t = t+1;
   end
end
Data = sort(Data);
Index = find(Data>0);
Start = Index(1);
Stop = max(size(Data));
Data_Neg = Data(1:Start-1);
Data_Pos = Data(Start:Stop);
%Centres_Neg = sort(fcm(Data_Neg',(U_Num-1)/2));
Centres_Pos = sort(fcm(Data_Pos',(U_Num-1)/2));
Centres_Neg = -Centres_Pos;
Centres = [Centres_Neg;0;Centres_Pos];
Centres = sort(Centres);
t = 1;
for t = 1:U_Num
   U_MBF_Num(t) = t;
end
for q = 1:State_Num*Err_Int_Num*Err_Num
   for w = 1:Err_Der_Num
      Rule = RuleBase(q,w);
      Difference = abs(Rule*ones(U_Num,1)-Centres);
      [Min_Diff,Index] = min(Difference);
      RuleBaseClusters(q,w) = Centres(Index);
      RuleBaseNorm(q,w) = U_MBF_Num(Index);
   end
end
%  Generate the fuzzy rule-base matrix
u=1;
for q = 1:State_Num
   qq = q-1;
   for t = 1:Err_Int_Num
      tt = t-1;
      for w = 1:Err_Num
         for e = 1:Err_Der_Num
            r = qq*Err_Int_Num*Err_Num+tt*Err_Num+w;
            rule = [q , t , w , e , RuleBaseNorm(r,e) , 1 , 1 ];
            RBM(u,:) = rule;
            u = u+1;
         end
      end
   end
end
%  Add the rule-base to the system
Fuzzy_System = addrule(Fuzzy_System,RBM);
```

II RULE-BASE MODIFICATION ALGORITHM

The proposed self-adaptive algorithm fuzzy logic controller algorithm is implemented as a SIMuLINK S-Function. The S-Function is implemented in the standard M-File format. The inputs to the block are the standard fuzzy logic controller inputs (error integral, error and error rate of change), the response error (determined by the model reference adaptive architecture) and the reference signal. The simulation time step, simulation end time, gain and delay information are entered as block parameters in SIMuLINK.

```
function [sys,x0,str,ts] =
FuzTune(t,x,u,flag,GP,DP,TransSpeed,FinSpeed,TransWin,FinWin,dT,FinTime,...
   IntCond,SaveCond)
%-----------------------------------------------------------------------------------------------
%  Workspace Globals
global FuzzyController Taboo RulesIntOut FrStr SugMod Rules GainHis
%-----------------------------------------------------------------------------------------------
%  S-Function Globals
global NumIn NumOut RuleBase AndMeth OrMeth NumRules StepTime Cnt RulesIntIn LastMove NumU RangeU
%-----------------------------------------------------------------------------------------------
switch flag,
  case 0,

[sys,x0,str,ts]=mdlInitializeSizes(t,x,u,GP,DP,TransSpeed,FinSpeed,TransWin,FinWin,dT,FinTime,...
        IntCond,SaveCond);
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
     sys=mdlOutputs(t,x,u,GP,DP,TransSpeed,FinSpeed,TransWin,FinWin,dT,FinTime,...
        IntCond,SaveCond);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
     error(['Unhandled flag = ',num2str(flag)]);
  end
%-----------------------------------------------------------------------------------------------
function [sys,x0,str,ts]=mdlInitializeSizes(t,x,u,GP,DP,TransSpeed,FinSpeed,TransWin,FinWin,dT,...
   FinTime,IntCond,SaveCond)
%-----------------------------------------------------------------------------------------------
%  Workspace Globals
global FuzzyController Taboo RulesIntOut FrStr SugMod Rules GainHis
%-----------------------------------------------------------------------------------------------
%  S-Function Globals
global NumIn NumOut RuleBase AndMeth OrMeth NumRules StepTime Cnt RulesIntIn LastMove NumU RangeU
%-----------------------------------------------------------------------------------------------

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 1;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 6;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [0];
str = [];
ts  = [0 0];
NumIn = getfis(FuzzyController,'NumInputs');
NumOut = getfis(FuzzyController,'NumOutputs');
RuleBase = getfis(FuzzyController,'rulelist');
AndMeth = getfis(FuzzyController,'andMethod');
OrMeth = getfis(FuzzyController,'orMethod');
NumRules = getfis(FuzzyController,'numRules');
NumU = getfis(FuzzyController,'numoutputmfs');
```

```
RangeU = getfis(FuzzyController,'outrange');
RulesIntIn = zeros(1,NumRules);
if IntCond == 0
   RulesIntOut = zeros(1,NumRules);
   Taboo = ones(1,NumRules);
else
   RulesIntOut = RulesIntOut;
   Taboo = Taboo;
end
NumRows = 1+(FinTime/dT);
GainHis = zeros(NumRows,1);
FrStr = zeros(NumRows,NumRules);
if SaveCond == 1
   SugMod = zeros(NumRows,NumRules);
   Rules = zeros(NumRows,NumRules);
else
   SugMod = [];
   Rules = [];
end
Cnt = 1;
StepTime = 0;
RulesIntIn = zeros(1,NumRules);
LastMove = zeros(1,NumRules);
%----------------------------------------------------------------------------------------------------
% end mdlInitializeSizes
%----------------------------------------------------------------------------------------------------
function sys=mdlOutputs(t,x,u,GP,DP,TransSpeed,FinSpeed,TransWin,FinWin,dT,FinTime,...
   IntCond,SaveCond)
%----------------------------------------------------------------------------------------------------
%  Workspace Globals
global FuzzyController Taboo RulesIntOut FrStr SugMod Rules GainHis
%----------------------------------------------------------------------------------------------------
%  S-Function Globals
global NumIn NumOut RuleBase AndMeth OrMeth NumRules StepTime Cnt RulesIntIn LastMove NumU RangeU
%----------------------------------------------------------------------------------------------------
RulesIntOut = RulesIntIn + RulesIntOut;
RefDiff = (u(6)-x(1))/dT;
if RefDiff ~= 0
   StepTime = t;
else
   StepTime = StepTime;
end
Gain = polyval(GP,u(1));
Delay = polyval(DP,u(1));
DelCnt = fix(Delay/dT);
[y,tv,otv,dc] = evalfismex(u(1:4),FuzzyController,101);
RuleStr = getrulestr(tv,AndMeth,OrMeth,RuleBase,NumIn,NumOut);
if ((t-Delay) > StepTime) & (t < (StepTime + TransWin))                       %  Tune in transient
window
   RulesIntIn = (FrStr(Cnt-DelCnt,:).*Taboo)*(TransSpeed*u(5)/GainHis(Cnt-DelCnt));
   UThresh = (RangeU(2)-RangeU(1))/NumU;
   Update = fix(RulesIntOut/UThresh);
   RuleBase(:,5) = RuleBase(:,5) + Update';
   TooSmall = find(RuleBase(:,5) < 1);
   RuleBase(TooSmall,5) = 1;
   TooBig = find(RuleBase(:,5) > NumU );
   RuleBase(TooBig,5) = NumU;
   RulesIntOut = RulesIntOut - UThresh*Update;
   if ~isempty(Update)
      FuzzyController = setfis(FuzzyController,'ruleList',RuleBase);
      Tuned = find(Update ~= 0);
      NumTuned = length(Tuned);
      for ChkCnt = 1:NumTuned
         ChkRule = Tuned(ChkCnt);
         ChkOsc = LastMove(1,ChkRule) + Update(ChkRule);
         LastMove(1,ChkRule) = Update(ChkRule);
         if ChkOsc == 0
            Taboo(ChkRule) = 1;
         end
                                    %       if ChkOsc == 0
      end
                                    %   for ChkCnt = 1:NumTuned
   end
elseif (t > StepTime + TransWin) & (t < (StepTime + TransWin + FinWin))                 %  Tune in
fin window
   RulesIntIn = (FrStr(Cnt-DelCnt,:).*Taboo)*(FinSpeed*u(5)/GainHis(Cnt-DelCnt));
   UThresh = (RangeU(2)-RangeU(1))/NumU;
   Update = fix(RulesIntOut/UThresh);
   RuleBase(:,5) = RuleBase(:,5) + Update';
   TooSmall = find(RuleBase(:,5) < 1);
   RuleBase(TooSmall,5) = 1;
   TooBig = find(RuleBase(:,5) > NumU );
   RuleBase(TooBig,5) = NumU;
   RulesIntOut = RulesIntOut - UThresh*Update;
   if ~isempty(Update)
      FuzzyController = setfis(FuzzyController,'ruleList',RuleBase);
      Tuned = find(Update ~= 0);
      NumTuned = length(Tuned);
      for ChkCnt = 1:NumTuned
         ChkRule = Tuned(ChkCnt);
         ChkOsc = LastMove(1,ChkRule) + Update(ChkRule);
         LastMove(1,ChkRule) = Update(ChkRule);
         if ChkOsc == 0
            Taboo(ChkRule) = 1;
         end
                              %      if ChkOsc == 0
      end
                              %   for ChkCnt = 1:NumTuned
   end
                                     %  if ~isempty(Update)
else
                                     %  No Tuning
   RulesIntIn = zeros(1,NumRules);
   Update = zeros(1,NumRules);
end
GainHis(Cnt,1) = Gain;
FrStr(Cnt,:) = RuleStr';
if SaveCond == 1
   SugMod(Cnt,:) = RulesIntOut;
   Rules(Cnt,:) = RuleBase(:,5)';
end
sys = [y];
Cnt = Cnt+1;
%---------------------------------------------------------------------------------------------------
% end mdlOutputs
%---------------------------------------------------------------------------------------------------
%---------------------------------------------------------------------------------------------------
function sys=mdlDerivatives(t,x,u)
sys = [];
function sys=mdlUpdate(t,x,u)
sys = [u(6)];
function sys=mdlTerminate(t,x,u)
sys = [];
```

III PLANT I: M-FILES

The MATLAB® M-Files pertaining to the cargo ship are given in this appendix. The cargo ship simulation model is implememted as a standard SIMuLINK S-Function with three continuous dynamic states as given in paragraph III.I. The design of the single-mode controller is given in paragraph III.II, while the multi-mode controller is detailed in paragraph III.III.

III.I SIMULATION MODEL

```
function [sys,x0,str,ts] = CargoShip(t,x,u,flag)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
     sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
     error(['Unhandled flag = ',num2str(flag)]);
end
%==========================================================================================
=======%
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 3;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
x0  = [ 0 ; 0 ; 0 ];
str = [];
ts  = [0 0];
%==========================================================================================
=======%
function sys=mdlDerivatives(t,x,u)
l = 161;
K0 = -3.86;
Tau10 = 5.66;
Tau20 = 0.38;
Tau30 = 0.89;
a = 1;
b = 1;
MaxRud = 60*2*pi/360;
MinVel = 2.5;
MaxVel = 7.5;
if u(1) > MaxVel
   u(1) = MaxVel;
elseif u(1) < MinVel
   u(1) = MinVel;
else
   u(1) = u(1);
end
if u(2) > MaxRud
   u(2) = MaxRud;
elseif u(2) < -MaxRud
   u(2) = -MaxRud;
else
   u(2) = u(2);
```

```
end
K = K0*u(1)/l;
Tau1 = Tau10*l/u(1);
Tau2 = Tau20*l/u(1);
Tau3 = Tau30*l/u(1);
K1 = (1/Tau1)+(1/Tau2);
K2 = 1/(Tau1*Tau2);
K3 = (K*Tau3)/(Tau1*Tau2);
K4 = K/(Tau1*Tau2);
H = a*(x(2)^3) + b*x(2);
xdot1 = x(2);
xdot2 = x(3) + K3*u(2);
xdot3 = -K1*(x(3)+K3*u(2)) - K2*H + K4*u(2);
sys = [xdot1,xdot2,xdot3];
%===================================================================================================
=======%
function sys=mdlUpdate(t,x,u)
sys = [];
%===================================================================================================
=======%
function sys=mdlOutputs(t,x,u)
sys = [x(1) , x(2)];
%===================================================================================================
=======%
function sys=mdlGetTimeOfNextVarHit(t,x,u)
% sampleTime = 1;    %  Example, set the next hit to be one second later.
% sys = t + sampleTime;
%===================================================================================================
=======%
function sys=mdlTerminate(t,x,u)
sys = [];
%===================================================================================================
=======%
```

III.II SINGLE-MODE FUZZY LOGIC CONTROLLER DESIGN

III.II.I CONTROLLER DESIGN FILE

```
%  FUZ_DESIGN.M
%  Designs a fuzzy controller for the cargo ship.
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  13/01/1999
%  Setup
clc;
close all;
clear all;
%  Define the size of the controller
Ni = 11;
Nc = 11;
%  Define the universe of discourse
U = [-60*2*pi/360 60*2*pi/360];
Err = [-1 1];
Rate = [-1 1];
State = [2.5 7.5];
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Err',Err);
FuzCon = addvar(FuzCon,'input','Rate',Rate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions;
```

```
Names = namegen(Ni,'Sym');
U_Names = namegen(Nc,'Sym');
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,1,State,Overlap,'Z');
FuzCon = symbfgen(FuzCon,'input',2,Ni,Err,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Rate,Overlap,Names);
% Auto design the rulebase according to the wikkel method
Error_Weight = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
Rate_Weight = [-5, -4, -3 -2, -1, 0, 1, 2, 3, 4, 5];
Gain = [ 1 ];
Speed = [ 1 ];
Dims = [ 1 , Ni , Ni , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pdrbgen(...
FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Dims);
% Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
writefis(FuzCon,'sys_con.fis');
global FuzzyController
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Err;
ErrRateLim = Rate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrRateLim Kp Kd
```

III.II.II GAIN DESIGN FILE

```
%       GAIN_DESIGN.M
%       Gain design for fuzzy controller
%       Etienne M. Hugo         -       9019693
%       University of Stellenbosch    -       Process Control Group
%       05/04/1999
% Setup
clc;
close all;
clear all;
% Load the system parameters
load con_param.mat;
% Set the limits of the universes of discourse
ErrMax = 10*2*pi/360;
RateMax = 0.01;
% Get the implementation gains
Kp = 1/ErrMax;
Kd = 1/RateMax;
Gu = 1.1163;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrRateLim Kp Kd Gu
```

III.III MULTI-MODE FUZZY LOGIC CONTROLLER DESIGN

III.III.I CONTROLLER DESIGN FILE

```
% FUZ_DESIGN.M
% Designs a fuzzy controller for the cargo ship.
% Etienne M. Hugo     90-1969-3
% University of Stellenbosch 90-1969-3
% 13/01/1999
% Setup
clc;
close all;
clear all;
% Define the size of the controller
Ni = 11;
```

```
Nc = 11;
        %  Define the universe of discourse
U = [-60*2*pi/360 60*2*pi/360];
Err = [-1 1];
Rate = [-1 1];
State = [2.5 7.5];
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Err',Err);
FuzCon = addvar(FuzCon,'input','Rate',Rate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions;
State_Names = namegen(3,'Sym');
Names = namegen(Ni,'Sym');
U_Names = namegen(Nc,'Sym');
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,3,State,Overlap,State_Names);
FuzCon = symbfgen(FuzCon,'input',2,Ni,Err,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Rate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
Rate_Weight = [-5, -4, -3 -2, -1, 0, 1, 2, 3, 4, 5];
% Gain = [ 1 1 1 ];
% Speed = [ 0.5 0.66 1];
Gain = [ 0.0599 0.1198 0.1798*0.75 ];
Speed = [ 0.4*1 1 1 ];
Dims = [ 3 , Ni , Ni , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pdrbgen(...
FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
writefis(FuzCon,'sys_con.fis');
global FuzzyController
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Err;
ErrRateLim = Rate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrRateLim Kp Kd
```

III.III.II GAIN DESIGN FILE

```
%       GAIN_DESIGN.M
%       Gain design for fuzzy controller
%       Etienne M. Hugo          -       9019693
%       University of Stellenbosch    -       Process Control Group
%       05/04/1999
%  Setup
clc;
close all;
clear all;
%  Load the system parameters
load con_param.mat;
%  Set the limits of the universes of discourse
ErrMax = 10*2*pi/360;
RateMax = 0.01;
%  Get the implementation gains
Kp = 1/ErrMax;
Kd = 1/RateMax;
Gu = 1.1163;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrRateLim Kp Kd Gu
```

IV PLANT II: M-FILES

The MATLAB® M-Files pertaining to the second-order plant with exponential damping is given in this Appendix. The plant simulation model is implemented as a standard SIMuLINK S-Function with two continuous dynamic states as given in paragraph IV.I. The design of the single-mode controller is given in paragraph IV.II, while the multi-mode controller is detailed in paragraph IV.III.

IV.I SIMULATION MODEL

```
function [sys,x0,str,ts] = Plant(t,x,u,flag,X0)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(t,x,u,X0);
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% end sfuntmpl
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
function [sys,x0,str,ts]=mdlInitializeSizes(t,x,u,X0)
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
% initialize the initial conditions
x0  = X0;
% str is always an empty matrix
str = [];
% initialize the array of sample times
ts  = [0 0];
% end mdlInitializeSizes
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
function sys=mdlDerivatives(t,x,u)
if u > 0.5
   u = 0.5;
elseif u < -0.5
   u = -0.5;
end
x1_dot = x(2);
x2_dot = -x(1) - ( 0.1 + exp(-1*(x(1)^2)))*x(2) + (1 + x(1)^2)*u;
sys = [x1_dot;x2_dot];
% end mdlDerivatives
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
function sys=mdlUpdate(t,x,u)
sys = [];
% end mdlUpdate
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
function sys=mdlOutputs(t,x,u)
sys = x;
% end mdlOutputs
%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
function sys=mdlTerminate(t,x,u)
sys = [];
% end mdlTerminate
```

IV.II SINGLE-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  FUZ_DESIGN.M
%  Designs a fuzzy controller for the 3'rd order plant
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  21/07/1999
%  Setup
clc;
close all;
clear all;
%  Design a fuzzy control system using the wikkel method
%  Define the size of the controller
Ni = 7;
Nc = 21;
%  Define the universe of discourse
U = [-0.5 0.5];
X = [-1 1];
Xdot = [-1 1];
State = [0 0.75];
Error = X;
ErrInt = Error;
ErrRate = Xdot;
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Int',ErrInt);
FuzCon = addvar(FuzCon,'input','Error',Error);
FuzCon = addvar(FuzCon,'input','Rate',ErrRate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions
Names = namegen(Ni,'Sym');
StateNames = ['S1'];
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,1,State,Overlap,StateNames);
FuzCon = symbfgen(FuzCon,'input',2,Ni,ErrInt,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Error,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',4,Ni,ErrRate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-3, -2, -1, 0 , 1 , 2, 3];
Int_Weight = [-3, -2, -1, 0 , 1 , 2, 3];
Rate_Weight = [-3, -2, -1, 0 , 1 , 2, 3];
Gain =  [ 1 ];
Speed = [ 1 ];
Dims = [ 1 , Ni*ones(1,3) , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(...
   FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Nc,'Sym');
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
global FuzzyController
Ki = 1;
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Error;
ErrIntLim = ErrInt;
ErrRateLim = ErrRate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrIntLim ErrRateLim Ki Kp Kd
```

IV.III MULTI-MODE FUZZY LOGIC CONTROLLER DESIGN

```
multi-mode fuzzy logic controller design
%  FUZ_DESIGN.M
%  Designs a fuzzy controller for the 3'rd order plant
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  13/01/1999
%  Setup
clc;
close all;
clear all;
%  Design a fuzzy control system using the wikkel method
%  Define the size of the controller
Ni = 7;
Nc = 21;
%  Define the universe of discourse
U = [-0.5 0.5];
X = [-1 1];
Xdot = [-1 1];
State = [0 0.75];
Error = X;
ErrInt = Error;
ErrRate = Xdot;
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Int',ErrInt);
FuzCon = addvar(FuzCon,'input','Error',Error);
FuzCon = addvar(FuzCon,'input','Rate',ErrRate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions;
Names = namegen(Ni,'Sym');
StateNames = ['S1';'S2';'S3';'S4'];
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,4,State,Overlap,StateNames);
FuzCon = symbfgen(FuzCon,'input',2,Ni,ErrInt,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Error,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',4,Ni,ErrRate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-3, -2, -1, 0 , 1 , 2, 3];
Int_Weight = [-3, -2, -1, 0 , 1 , 2, 3];
Rate_Weight = [-3, -2, -1, 0 , 1 , 2, 3];
Gain =  [ 1 ; 1.2 ; 2 ; 5];
Speed = [ 1 ; 0.95 ; 0.8  ; 0.6];
Dims = [ 4, Ni*ones(1,3) , Nc ];
```

```
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(...
FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Nc,'Sym');
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
global FuzzyController
Ki = 1;
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Error;
ErrIntLim = ErrInt;
ErrRateLim = ErrRate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrIntLim ErrRateLim Ki Kp Kd
```

V PLANT III: M-FILES

The MATLAB® M-Files pertaining to the second-order plant with non-linear velocity feedback are given in this Appendix. The plant simulation model is implemented directly using standard SIMuLINK blocks. The design of the single-mode controller is given in paragraph V.I, while the multi-mode controller is detailed in paragraph V.II.

V.I SINGLE-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  FUZ_DESIGN.M
%  Designs a fuzzy controller for the 3'rd order plant
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  21/07/1999
%  Setup
clc;
close all;
clear all;
%  Design a fuzzy control system using the wikkel method
%  Define the size of the controller
Ni = 9;
Nc = 21;
%  Define the universe of discourse
U = [-1 1];
X = [-1 1];
Xdot = [-1 1];
State = [0 1.2];
Error = X;
ErrInt = Error;
ErrRate = Xdot;
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Int',ErrInt);
FuzCon = addvar(FuzCon,'input','Error',Error);
FuzCon = addvar(FuzCon,'input','Rate',ErrRate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions
Names = namegen(Ni,'Sym');
StateNames = ['S1'];
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,1,State,Overlap,StateNames);
FuzCon = symbfgen(FuzCon,'input',2,Ni,ErrInt,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Error,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',4,Ni,ErrRate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Int_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Rate_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Gain =  [ 1 ];
Speed = [ 1 ];
Dims = [ 1 , Ni*ones(1,3) , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(...
   FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
```

```
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Nc,'Sym');
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
global FuzzyController
Ki = 1;
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Error;
ErrIntLim = ErrInt;
ErrRateLim = ErrRate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrIntLim ErrRateLim Ki Kp Kd
```

V.II MULTI-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  FUZ_DESIGN.M
%  Designs a fuzzy controller for the 3'rd order plant
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  21/07/1999
%  Setup
clc;
close all;
clear all;
%  Design a fuzzy control system using the wikkel method
%  Define the size of the controller
Ni = 9;
Nc = 21;
%  Define the universe of discourse
U = [-1 1];
X = [-1 1];
Xdot = [-1 1];
State = [0 1.2];
Error = X;
ErrInt = Error;
ErrRate = Xdot;
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Int',ErrInt);
FuzCon = addvar(FuzCon,'input','Error',Error);
FuzCon = addvar(FuzCon,'input','Rate',ErrRate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions
Names = namegen(Ni,'Sym');
StateNames = ['S1';'S2';'S3';'S4'];
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,4,State,Overlap,StateNames);
FuzCon = symbfgen(FuzCon,'input',2,Ni,ErrInt,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Error,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',4,Ni,ErrRate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Int_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Rate_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Gain =  [ 1 1 1 1 ];
Speed = [0.62 0.13 0.1*0.49 0.1*0.30 ];
Dims = [ 4 , Ni*ones(1,3) , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(...
   FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
```

```
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Nc,'Sym');
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
global FuzzyController
Ki = 1;
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Error;
ErrIntLim = ErrInt;
ErrRateLim = ErrRate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrIntLim ErrRateLim Ki Kp Kd
```

VI DISTILLATION COLUMN: M-FILES

The MATLAB® M-Files pertaining to homogenous, azeotropic distillation column are given in this appendix. The simulation model is implemented as a standard SIMuLINK S-Function as detailed in paragraph VI.I. The block have 10 inputs, 164 outputs and 423 discrete states. The block implements a multi-step Runge-Cutta integration method to predict the response of the state vector. The design of the single-mode fuzzy logic controller is detailed in paragraph VI.II, while the multi-mode fuzzy logic controller is detailed in paragraph VI.III.

VI.I SIMULATION MODEL

```
Simulation model S-Function M-File
function [sys,x0,str,ts] = distcol2(t,x,u,flag)
I = eye(107);                     % 107by107 Unity matrix
k1 = 0.4358662;                          %
k2 = 0.75;                        %
k3 = 0.63020209;                  % Runge-Kutta
k4 = 0.24233789;                  % constants
k5 = 1.037609496;                 %
k6 = 0.83493048;
% Dispatch the flag
switch flag,
  case 0,
     [sys,x0,str,ts]=mdlInitializeSizes(I,k1,k2,k3,k4,k5,k6);
  case 1,
     sys=mdlDerivatives(t,x,u,I,k1,k2,k3,k4,k5,k6);
  case 2,
     sys=mdlUpdate(t,x,u,I,k1,k2,k3,k4,k5,k6);
  case 3,
     sys=mdlOutputs(t,x,u,I,k1,k2,k3,k4,k5,k6);
  case 4,
     sys=mdlGetTimeOfNextVarHit(t,x,u,I,k1,k2,k3,k4,k5,k6);
  case 9,
     sys=mdlTerminate(t,x,u,I,k1,k2,k3,k4,k5,k6);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end                                     % end sfuntmpl
function [sys,x0,str,ts]=mdlInitializeSizes(I,k1,k2,k3,k4,k5,k6)
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 423;
sizes.NumOutputs     = 164;
sizes.NumInputs      = 10;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
global dyn_factor WA liqC WH
global Cpvint Cplcon Cplint Tfeed
global dt liqC Zra Tcrit Pcrit T_reb dvrref Lreb_SS
global bubtol stop
global U eta fugC W TS S H Zconst G E
global elipstol
global TAU RU QU QP RL
global dec_logger dec_logstatus T_aqu T_org dvaref dvoref Lda_SS Ldo_SS
load Lda0
load Ldo0
load V0
load Z0
load xad0
load xod0
load dva0
load dvo0
load dvr0
load Mad0
load Mod0
load aquctrl.txt
load orgctrl.txt
load rebctrl.txt
load inmat.txt
dt = inmat(8,1);
tsample = inmat(8,2);
dyn_factor = inmat(8,3);
stabtest = inmat(8,4);
dvaref = aquctrl(1);
T_aqu = aquctrl(2);
Lda_SS = aquctrl(3);
dvoref = orgctrl(1);
T_org = orgctrl(2);
Ldo_SS = orgctrl(3);
dvrref = rebctrl(1);
T_reb = rebctrl(2);
Lreb_SS = rebctrl(3);
U = [0, 115.13, 573.61; 2057.42, 0, 1131.13; -163.72, -149.34, 0]; % UNIQUAC binary interaction
parameter constants of Prausnitz
RL = [-2.32; 1.76; -4.1e-1];                          % Liq-Liq Flash parameters from Prausnitz
RU = [0.92; 3.19; 2.11];                              % Appendix C-1, ref.10
QU = [1.40; 2.40; 1.97];                             %
QP = [1.00; 2.40; 0.92];                             %
% Non-polar acentric parameters
W = [1.1316e-002            0            0
     9.0766e-002  1.7022e-001            0
     5.7712e-002  1.3716e-001  1.0411e-001];
% Energy parameters
TS = [221.1344         0          0
      326.9346  507.5722          0
      274.1875  416.1372  344.9629];
% Calculate temp-independant terms in virial coefficients for
% PC's and pairs
% Molecular size parameter
S =  [56.3522         0          0
      98.3811  171.7564          0
      86.3498  150.7519  132.3161];
H =  [3.1967          0          0
      1.9900    1.9900          0
      2.2751    1.9900    2.0554];
Zconst =  [-0.4228          0          0
      -0.3000   -0.3000          0
      -0.3597   -0.3000   -0.3286];
G =  [2.2064          0          0
           0          0          0
      0.9439          0    0.3219];
% Energy terms for non-associating pairs (for terms with positive ETA)
E = [-3.0227          0          0
           0          0          0
     -3.1380          0    -3.2622];
eta = [1.70 0 0; 0 0 0; 1.55 0 1.40];
Tcrit = [647.37; 562.16; 516.26];                    % Critical params for use with Hayden &
O'Connell correllation for
Pcrit = [221.20; 48.98; 63.80];                      % virial coefficients - ref.10
Zra = [0.2380 0.2696 0.2520];                        % Rackett compressability factor - ref.10
fugC =  [5.7042e1  -7.0048e3   3.5888e-3  -6.6689e0  -8.5054e-7;  % Constants for zero-pressure
referance fugacity eqn.
         9.7209e1  -6.9761e3   1.9082e-2  -1.4212e1  -6.7182e-6;  % ref.10
        -9.0910e1  -3.4659e3  -6.2301e-2   2.0486e1   2.0664e-5];
Cplcon = [18.2964 4.72118e-1 -1.33878e-3 1.31424e-6;
         -7.27329 7.70541e-1 -1.64818e-3 1.89794e-6;              % Liq. Heat capacity correlation
constants
         -3.25137e2 4.13787 -1.40307e-2 1.70354e-5];              % by Reklaitis - ref.14
Cplint =  Cplcon*diag([1 0.5 (1/3) 0.25]);                        % Constants for integrating liq.
heat capacity
                                                                  % to liquid enthalpy
Cpvcon = [34.0471 -9.65064e-3 3.29983e-5 -2.04467e-8 4.30228e-12;  % Same as above for vapor
         18.5868 -1.17439e-2 1.27514e-3 -2.07984e-6 -1.05329e-9;
         17.6907  1.49532e-1 8.94815e-5 -1.97384e-7 8.31747e-11];
Cpvint = Cpvcon*diag([1 0.5 (1/3) 0.25 0.2]);
splinepol = [2.553438952e9 -4.818800201e9 4.0189686031e9 -1.948239037e9 6.071029437e8 ...
            -1.269128763e8 1.800492176e7 -1.710395049e6 1.041105423e5 -3.672003074e3 ...
            57.850421156];
                                                     % Spline-fitted polynomial describing
heterogeneous liquid envelope
```

```
                                                            % Predicted by UNIQUAC eqn. with Prausnitz
parameters
%---------------------------------------------------------------
% Thermodynamical subroutine tolerances
bubtol = 1e-4;
fbtol = 1e-3;
elipstol = 1e-4;
%---------------------------------------------------------------
% Fixed column parameters
WA = 2300;                                                  % ACTIVE tray area [cm^2]
WH = 2.54*ones(1,26);                                       % Weir height [cm]
liqC = 6.53e-4;                                             % Scaling factor for tray hydraulics [van
Wynkle]
Tfeed = 351.3;                                              % Saturated feed temperature
%---------------------------------------------------------------
% Set up initial profiles and column variables
P0 = 1.013:((1.216-1.013)/26):1.216;
xfe = [0;1;0];
Fe = 1.65;
alpha = 0.767;
[rm,xom,xam,ym,T0,K_0] = bub(Z0,P0);                        % Initial temperature profile & equilibrium
constants
[M_0,p0,pp0,Lreb0,d1] = molhold(rm,xom,xam,T0,Z0,V0(27),dvr0);   % Initial molar hold-up for column
& reboiler
[rd0,d1,d2,d3,status,err] = elips(ym(:,1),298.15);   % Initial organic- and aqueous phase
compositions
clear rm xom xam ym xa xo Td Zdec d1 d2 d3
Z = Z0;
V = V0;
K_ = K_0;
M_ = M_0;
Mod = Mod0;
Mad = Mad0;
Lreb = Lreb0;
P = P0;
T = T0;
p = p0;
ppure = pp0;
Ldo = Ldo0;
Lda = Lda0;
xod = xod0;
xad = xad0;
rd = rd0;
dva = dva0;
dvo = dvo0;
dvr = dvr0;
L_1 = Ldo + alpha*Lda + Fe;
x_1_temp = (xod*Ldo + xad*Lda*alpha + xfe*Fe)/L_1;
x_1 = x_1_temp/([1 1 1]*x_1_temp);
State1 = [Z;V';K_(1,:)';K_(2,:)';K_(3,:)';M_';Mod;Mad;Lreb;P';T';p'];
State2 = [ppure(:,1);ppure(:,2);ppure(:,3);Ldo;Lda;xod;xad;rd;dva;dvo;dvr;L_1;x_1];
State = [State1;State2];
x0 = State;
str = [];
ts  = [0 0];
% end mdlInitializeSizes
function sys=mdlDerivatives(t,x,u,I,k1,k2,k3,k4,k5,k6)
sys = [];
% end mdlDerivatives
function sys=mdlUpdate(t,x,u,I,k1,k2,k3,k4,k5,k6)
global dyn_factor WA liqC WH
global Cpvint Cplcon Cplint Tfeed
global dt liqC Zra Tcrit Pcrit T_reb dvrref Lreb_SS
global bubtol stop
global U eta fugC W TS S H Zconst G E
global elipstol
global TAU RU QU QP RL
global dec_logger dec_logstatus T_aqu T_org dvaref dvoref Lda_SS Ldo_SS

Z = x(1:107);
V = x(108:134)';
K_ = [x(135:161)';x(162:188)';x(189:215)'];
M_ = x(216:242)';
Mod = x(243);
Mad = x(244);
Lreb = x(245);
P = x(246:272)';
T = x(273:299)';
p = x(300:326)';
ppure = [x(327:353),x(354:380),x(381:407)];
```

```
Ldo = x(408);
Lda = x(409);
xod = x(410:412);
xad = x(413:415);
rd = x(416);
dva = x(417);
dvo = x(418);
dvr = x(419);
L_1 = x(420);
x_1 = x(421:423);
F = [zeros(1,4) u(1) zeros(1,22)];
z = [u(2);u(3);u(4)];
zmat = [zeros(3,4) z zeros(3,22)];
Fe = u(5);
xfe = [u(6);u(7);u(8)];
alpha = u(9);
Q = [zeros(1,26) u(10)];
%------------------------------------------------------------------------------------------------
%                                                   -- STEP 1 --
% MAIN LOOP STARTS HERE
  [a1,a2,a3,a4,b1,b2,b3,b4,b5,d1,e1,e2,e3,e4,e5] = abde(Z,V,K_,M_,F,zmat,L_1,x_1,p,ppure);
  [Jac1,zp1] = azf(1,Z,V,L_1,x_1,a1,a2,a3,a4,b1,b2,b3,b4,b5,d1,e1,e2,e3,e4,e5);
  dZ1 = inv(I - k1*dt*Jac1)*dt*zp1;
%------------------------------------------------------------------------------------------------
%                                                   -- STEP 2 --
% 1st update of state vector & normalize component mole fractions
  Z_old = Z;
  Z = Z + k2*dZ1;
  sum_molefrac = Z(1:27) + Z(28:54) + Z(55:81);
  Z(1:81) = Z(1:81)./[sum_molefrac; sum_molefrac; sum_molefrac];
  dZ = [(Z(1:27) - Z_old(1:27)), (Z(28:54) - Z_old(28:54)), (Z(55:81) - Z_old(55:81))]';
  nz = nnz(~dZ);
  if nz,
    dZ = augzero(dZ);                 % Non-zero dZ needed to calculate vapour flow
  end
%------------------------------------------------------------------------------------------------
%                                                  -- STEP 3 & 4 --
% Liquid-liquid flash and bubble point temperature calculation for stages 1-27
  Told = T;
[rm,xom,xam,ym,T,K_] = bub(Z,P);
T = real(T);
  dT = T - Told;
  nz = nnz(~dT);
  if nz,
    dT = augzero(dT);                 % Non-zero dT needed to calculate vapour flow
  end
%------------------------------------------------------------------------------------------------
%                                                  -- STEP 5 & 6 --
% Calculates vapor flow (V) for stages 1-27 and liquid flow for stage 27 (Lreb)
  delta = (ones(3,1)*dT)./dZ;
  V = vlenth(zmat,F,Z,T,delta,K_,rm,xom,xam,ym,x_1,L_1,Q);
%------------------------------------------------------------------------------------------------
%                                                    -- STEP 7 --
% Calculates molar hold up for stages 1-26; hold up for reboiler stays constant
  [M_,p,ppure,Lreb,dvr] = molhold(rm,xom,xam,T,Z,V(27),dvr);
%------------------------------------------------------------------------------------------------
%                                                   -- STEP 8 --
% 2nd correction in Z
  [a1,a2,a3,a4,b1,b2,b3,b4,b5,d1,e1,e2,e3,e4,e5] = abde(Z,V,K_,M_,F,zmat,L_1,x_1,p,ppure);    %
Updated values for Z,V,K,M
  [dummy,zp2] = azf(0,Z,V,L_1,x_1,a1,a2,a3,a4,b1,b2,b3,b4,b5,d1,e1,e2,e3,e4,e5);       % !NB!
Jacobian stays unchanged over iteration
  dZ2 = inv(I - k1*dt*Jac1)*dt*zp2;
%------------------------------------------------------------------------------------------------
%                                                   -- STEP 9 --
% 3rd correction in Z
   dZ3 = inv(I - k1*dt*Jac1)*(-k3*dZ1 - k4*dZ2);
%------------------------------------------------------------------------------------------------
%                                                  -- STEP 10 --
% Total correction in Z & normalize component mole fractions
  Z_old = Z;
  Z = Z + k5*dZ1 + k6*dZ2 + dZ3;
  sum_molefrac = Z(1:27) + Z(28:54) + Z(55:81);
  Z(1:81) = Z(1:81)./[sum_molefrac; sum_molefrac; sum_molefrac];
  dZ = [(Z(1:27) - Z_old(1:27)), (Z(28:54) - Z_old(28:54)), (Z(55:81) - Z_old(55:81))]';
  nz = nnz(~dZ);
  if nz,
    dZ = augzero(dZ);                 % Non-zero dZ needed to calculate vapour flow
  end
%------------------------------------------------------------------------------------------------
```

```
%                                              -- STEP 11 --
% Update r(j), xo(ij), xa(ij), T(j), K_(ij), M_(j), V(j) and L_(27) for next iteration (k+1)
% Repeat steps 3-7 with new state vector Z
  Told = T;
  [rm,xom,xam,ym,T,K_] = bub(Z,P);
  T = real(T);
  dT = T - Told;
 nz = nnz(~dT);
  if nz,
    dT = augzero(dT);                 % Non-zero dT needed to calculate vapour flow
  end
  delta = (ones(3,1)*dT)./dZ;
  V = vlenth(zmat,F,Z,T,delta,K_,rm,xom,xam,ym,x_1,L_1,Q);
  [M_,p,ppure,Lreb,dvr] = molhold(rm,xom,xam,T,Z,V(27),dvr);
%----------------------------------------------------------------------------------------
%                                              -- STEP 12 --
% Calculate reflux flowrate and -composition
[rd,Mod,Mad,Ldo,Lda,xod,xad,L_1,x_1,dva,dvo] =
decanter(ym(:,1),V(1),Mod,Mad,Ldo,Lda,xod,xad,alpha,Fe,xfe,dva,dvo);
State1 = [Z;V';K_(1,:)';K_(2,:)';K_(3,:)';M_';Mod;Mad;Lreb;P';T';p'];
State2 = [ppure(:,1);ppure(:,2);ppure(:,3);Ldo;Lda;xod;xad;rd;dva;dvo;dvr;L_1;x_1];
State = [State1;State2];
State = real(zeros(423,1)+State);
sys = State;
% end mdlUpdate
function sys=mdlOutputs(t,x,u,I,k1,k2,k3,k4,k5,k6)
Z = x(1:107);
V = x(108:134)';
K_ = [x(135:161)';x(162:188)';x(189:215)'];
M_ = x(216:242)';
Mod = x(243);
Mad = x(244);
Lreb = x(245);
P = x(246:272)';
T = x(273:299)';
p = x(300:326)';
ppure = [x(327:353),x(354:380),x(381:407)];
Ldo = x(408);
Lda = x(409);
xod = x(410:412);
xad = x(413:415);
rd = x(416);
dva = x(417);
dvo = x(418);
dvr = x(419);
L_1 = x(420);
x_1 = x(421:423);
Output = [ Z ; Lreb ; T' ; V' ; Ldo ; Lda ];
sys = Output;
% end mdlOutputs
function sys=mdlGetTimeOfNextVarHit(t,x,u,I,k1,k2,k3,k4,k5,k6)
sampleTime = 1;    %  Example, set the next hit to be one second later.
sys = t + sampleTime;
function sys=mdlTerminate(t,x,u,I,k1,k2,k3,k4,k5,k6)
sys = [];
% end mdlTerminate
```

VI.II SINGLE-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  FUZDES.M
%  Fuzzy design program.
%  Etienne M. Hugo     90-1969-2
%  University of Stellenbosch        Process Control Group
%  07/04/1998
clc;
clear all;
close all;
%  Give design parameters
Dims = [1,9,9,9,21];
Lda = [0,3];
Err_Int = [-60,60];
Err = [-6,6];
```

```
Err_Rate = [-0.025,0.025];
dQ = [-30,30];
Overlap = 1;
%  Set the fuzzy system parameters
tfront_1 = newfis('fuz','mamdani');
tfront_1 = setfis(tfront_1,'andMethod','min');
tfront_1 = setfis(tfront_1,'orMethod','max');
tfront_1 = setfis(tfront_1,'impMethod','min');
tfront_1 = setfis(tfront_1,'aggMethod','max');
tfront_1 = setfis(tfront_1,'andMethod','min');
tfront_1 = setfis(tfront_1,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
tfront_1 = addvar(tfront_1,'input','State',Lda);
tfront_1 = addvar(tfront_1,'input','Integral',Err_Int);
tfront_1 = addvar(tfront_1,'input','Error',Err);
tfront_1 = addvar(tfront_1,'input','Rate',Err_Rate);
tfront_1 = addvar(tfront_1,'output','Control',dQ);
%  Auto design of membership functions;
Names = namegen(9,'Sym');
tfront_1 = symbfgen(tfront_1,'input',1,Dims(1),Lda,Overlap,['L']);
tfront_1 = symbfgen(tfront_1,'input',2,Dims(2),Err_Int,Overlap,Names);
tfront_1 = symbfgen(tfront_1,'input',3,Dims(3),Err,Overlap,Names);
tfront_1 = symbfgen(tfront_1,'input',4,Dims(4),Err_Rate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Int_Weight = [-4 -3 -2 -1 0 1 2 3 4];
Error_Weight = [-4 -3 -2 -1 0 1 2 3 4];
Rate_Weight = 1.0*[-4 -3 -2 -1 0 1 2 3 4];
Gain = [1];
Speed = [1];
[tfront_1,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(tfront_1,Gain,Speed,...
   Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(dQ)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Dims(5),'Sym','U');
tfront_1 = mbfgen(tfront_1,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
writefis(tfront_1,'tfront_1.fis');
```

VI.III MULTI-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  FUZDES.M
%  Fuzzy design program.  Uses the Wikkel method of design.
%  Etienne M. Hugo     90-1969-2
%  University of Stellenbosch        Process Control Group
%  07/04/1998
clc;
clear all;
close all;
%  Give design parameters
Dims = [2,9,9,9,21];
Lda = [0,2];
Err_Int = [-60,60];
Err = [-6,6];
Err_Rate = [-0.025,0.025];
dQ = [-30,30];
Overlap = 1;
%  Set the fuzzy system parameters
tfront_1 = newfis('fuz','mamdani');
tfront_1 = setfis(tfront_1,'andMethod','min');
tfront_1 = setfis(tfront_1,'orMethod','max');
tfront_1 = setfis(tfront_1,'impMethod','min');
tfront_1 = setfis(tfront_1,'aggMethod','max');
tfront_1 = setfis(tfront_1,'andMethod','min');
tfront_1 = setfis(tfront_1,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
tfront_1 = addvar(tfront_1,'input','State',Lda);
tfront_1 = addvar(tfront_1,'input','Integral',Err_Int);
tfront_1 = addvar(tfront_1,'input','Error',Err);
tfront_1 = addvar(tfront_1,'input','Rate',Err_Rate);
tfront_1 = addvar(tfront_1,'output','Control',dQ);
%  Auto design of membership functions;
```

```
Names = namegen(9,'Sym');
tfront_1 = symbfgen(tfront_1,'input',1,Dims(1),Lda,Overlap,['L';'H']);
tfront_1 = symbfgen(tfront_1,'input',2,Dims(2),Err_Int,Overlap,Names);
tfront_1 = symbfgen(tfront_1,'input',3,Dims(3),Err,Overlap,Names);
tfront_1 = symbfgen(tfront_1,'input',4,Dims(4),Err_Rate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Int_Weight = [-4 -3 -2 -1 0 1 2 3 4];
Error_Weight = [-4 -3 -2 -1 0 1 2 3 4];
Rate_Weight = 1.0*[-4 -3 -2 -1 0 1 2 3 4];
Gain = [1 0.75];
Speed = [1 1];
[tfront_1,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(tfront_1,Gain,Speed,...
   Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(dQ)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Dims(5),'Sym','U');
tfront_1 = mbfgen(tfront_1,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
writefis(tfront_1,'tfront_1.fis');
```

VIICSTR: M-FILES

The MATLAB® M-Files pertaining to the CSTR system are given in this appendix. The simulation model is implemented as a standard SIMuLINK S-Function as detailed in paragraph VII.I. The block have a single input, two outputs and two dynamic states. The design of the single-mode fuzzy logic controller is detailed in paragraph VII.II, while the multi-mode fuzzy logic controller is detailed in paragraph VII.III.

VII.I SIMULATION MODEL

```
function [sys,x0,str,ts] = Plant(t,x,u,flag,X0)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(t,x,u,X0);
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% end sfuntmpl
%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(t,x,u,X0)
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
%
% initialize the initial conditions
%
x0  = X0;
%
% str is always an empty matrix
%
str = [];
%
% initialize the array of sample times
%
ts  = [0 0];
% end mdlInitializeSizes
%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)
q = 1.0;
a = 0.072;
b = 8;
d = 3.0;
```

```
g = 20;
x1f = 1.0;
x2f = 0;
K1 = x(2)/(1+x(2)/g);
K2 = exp(K1);
x1_dot = -a*x(1)*K2 + q*(x1f-x(1));
x2_dot = b*a*x(1)*K2 - (q+d)*x(2) + q*x2f + d*u;
sys = [x1_dot;x2_dot];
% end mdlDerivatives
%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)
sys = [];
% end mdlUpdate
%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)
sys = x;
% end mdlOutputs
%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)
sys = [];
% end mdlTerminate
```

VII.II SINGLE-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  FUZ_DESIGN.M
%  Designs a fuzzy controller for the cstr
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  13/01/1999
%  Setup
clc;
close all;
clear all;
%  Design a fuzzy control system using the wikkel method
%  Define the size of the controller
Ni = 9;
Nc = 21;
%  Define the universe of discourse
U = [-1 1];
X = [-1 1];
Xdot = [-1 1];
State = [0.5 3.0];
Error = X;
ErrInt = Error;
ErrRate = Xdot;
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Int',ErrInt);
FuzCon = addvar(FuzCon,'input','Error',Error);
FuzCon = addvar(FuzCon,'input','Rate',ErrRate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions;
Names = namegen(Ni,'Sym');
StateNames = ['S1'];
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,1,State,Overlap,StateNames);
FuzCon = symbfgen(FuzCon,'input',2,Ni,ErrInt,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Error,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',4,Ni,ErrRate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Int_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Rate_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Gain =  [ 1 ];
Speed = [ 1 ];
Dims = [ 1, Ni*ones(1,3) , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(...
FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Nc,'Sym');
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
global FuzzyController
Ki = 1;
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Error;
ErrIntLim = ErrInt;
ErrRateLim = ErrRate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrIntLim ErrRateLim Ki Kp Kd
```

VII.III MULTI-MODE FUZZY LOGIC CONTROLLER DESIGN

```
%  Designs a fuzzy controller for the cstr
%  Etienne M. Hugo     90-1969-3
%  University of Stellenbosch 90-1969-3
%  13/01/1999
%  Setup
clc;
close all;
clear all;
%  Design a fuzzy control system using the wikkel method
%  Define the size of the controller
Ni = 9;
Nc = 21;
%  Define the universe of discourse
U = [-1 1];
X = [-1 1];
Xdot = [-1 1];
State = [0.5 3.0];
Error = X;
ErrInt = Error;
ErrRate = Xdot;
%  Define the fuzzy system
FuzCon = newfis('FuzzyController','mamdani');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'orMethod','max');
FuzCon = setfis(FuzCon,'impMethod','min');
FuzCon = setfis(FuzCon,'aggMethod','max');
FuzCon = setfis(FuzCon,'andMethod','min');
FuzCon = setfis(FuzCon,'defuzzmethod','centroid');
%  Set the fuzzy system inputs and outputs
FuzCon = addvar(FuzCon,'input','State',State);
FuzCon = addvar(FuzCon,'input','Int',ErrInt);
FuzCon = addvar(FuzCon,'input','Error',Error);
FuzCon = addvar(FuzCon,'input','Rate',ErrRate);
FuzCon = addvar(FuzCon,'output','Control',U);
%  Auto design of membership functions;
Names = namegen(Ni,'Sym');
StateNames = ['S1';'S2';'S3'];
Overlap = 1;
FuzCon = symbfgen(FuzCon,'input',1,3,State,Overlap,StateNames);
FuzCon = symbfgen(FuzCon,'input',2,Ni,ErrInt,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',3,Ni,Error,Overlap,Names);
FuzCon = symbfgen(FuzCon,'input',4,Ni,ErrRate,Overlap,Names);
%  Auto design the rulebase according to the wikkel method
Error_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Int_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Rate_Weight = [-4, -3, -2, -1, 0 , 1 , 2, 3, 4];
Gain =  [ 1 ; 1 ; 1 ];
Speed = [ 0.5 ; 1 ; 2 ];
Dims = [ 3, Ni*ones(1,3) , Nc ];
[FuzCon,RuleBase,RuleBaseClusters,RuleBaseNorm,RBM,Centres] = pidrbgen(...
FuzCon,Gain,Speed,Error_Weight,Rate_Weight,Int_Weight,Dims);
%  Generate the output membership functions
Centres_Scaled = max(U)*Centres/(max(Centres));
MidPoints = sort(Centres_Scaled);
U_Names = namegen(Nc,'Sym');
FuzCon = mbfgen(FuzCon,'output',1,MidPoints,Overlap,U_Names);
% Save the fuzzy system
global FuzzyController
Ki = 1;
Kp = 1;
Kd = 1;
StateLim = State;
ErrLim = Error;
ErrIntLim = ErrInt;
ErrRateLim = ErrRate;
FuzzyController = FuzCon;
OldFuzzyController = FuzCon;
save CON_PARAM.MAT FuzzyController OldFuzzyController StateLim ErrLim ErrIntLim ErrRateLim Ki Kp Kd
```