# INVERSE INTERNAL MODEL CONTROL OF AN ETHYLENE POLYMERISATION REACTOR USING ARTIFICIAL NEURAL NETWORKS

RYAN DUNWOODIE

Submitted in partial fulfilment of the academic requirements for the degree of Master of Science in Engineering in the School of Chemical Engineering, University of Natal, Durban.

7th September 2001

# ABSTRACT

An artificial neural network is a mathematical black-box modelling tool. This tool can be used to model complex non-linear multivariable processes. In attempting to create an inverse process model of an industrial linear low density polyethylene reactor, several interesting results were encountered. Both time-invariant algebraic and time-invariant dynamic models could adequately represent the process, provided an identified 50-minute time lag was taken into account.

A novel variation of the traditional IMC controller was implemented which used two inverse neural network process models. This was named Inverse Internal Model Control (IIMC). This controller was initially tested on a real multivariable pump-tank system and showed promising results.

The IIMC controller was adapted to an on-line version for the polymer plant control system. The controller was run in open loop mode to compare the predictions of the controller with the actual PID ratio controllers. It was hoped that by incorporating neural network models into the controller, they would take the non-linearity and coupling of the variables into account, which the present PID controllers are unable to do. The existing PID controllers operate on separate loops involving the two main feeds (co-monomer and hydrogen) to the reactor, which constitute aspects of the control system in which the scope for advanced control exists. Although the control loop was not closed, the groundwork has been laid to implement a novel controller that could the operation of the plant.

This is dedicated to my parents, Ron and Julie Dunwoodie, for their tremendous support, and the commitment they have shown me during my studies. I am very grateful for their financial support, especially during my undergraduate studies. I would also like to thank them for the many hours that they spent proof-reading this dissertation.

# PREFACE

This dissertation is submitted as part of the requirement for the Degree of M.Sc. Eng. [DNC5DL1 – Chemical M.Sc.(C/W) Dissertation]

64 Coursework credits were required, and 72 have been completed. The coursework component was made up of the following subjects.

- Linear Multi-variable Control [DNL5NC1] , 16 Credits , 73%
- Real Time Process Data Analysis [DNC5RT1], 16 Credits , 69%
- Applied Control Theory [DNC5CT1] , 16 Credits , 72%
- Computer Science 2A [DSX2CS1], C/C++ Object Orientated Programming 24 Credits, 60%

Total Credits = 72

The remaining 80 credits required for the M.Sc. Eng. degree are to be obtained for this dissertation.

I certify that all the work in this dissertation is my own work except where otherwise indicated and referenced.

Ryan Dunwoodie
7th September 2001

Michael Mulholland
Supervisor
7th September 2001

# ACKNOWLEDGEMENTS

This dissertation is the culmination of nearly two years of work. There are many people that have contributed to the completion of this Masters Course, and to whom I am very grateful. However there are a few who deserve special mention.

- SASOL POLYMERS (formerly POLIFIN LTD) for funding this research and allowing us to work on the Unipol Polyethylene Reactor and Control System. In order to protect their interests all data presented in this dissertation have been deliberately scaled and manipulated.

- Professor Mike Mulholland from the University of Natal, for his contribution to almost every aspect of this M.Sc. Eng., from the Course Work component to the final proof reading of this dissertation.

- Nirmal Narotam, from SasTech (formerly of SASOL POLYMERS) whose expertise on the reactor and MOORE DCS were invaluable in obtaining meaningful data from the plant and setting up the foundations for the code necessary to test our controller on-line.

- Cilius Van Der Merve, from SASOL POLYMERS, for supplying us with logged plant data regularly. Cilius was also instrumental in the final debugging and commissioning of the on-line controller. Both Nirmal and Cilius are acknowledged for their many hours of hard work.

- Dr. Assia Chouai who kindly let us make use of her neural network training software.

# TABLE OF CONTENTS

# LIST OF FIGURES

Page

## CHAPTER 1



## CHAPTER 2



## CHAPTER 3



## CHAPTER 4

## CHAPTER 5



## CHAPTER 6

## CHAPTER 7



## CHAPTER 8

## APPENDIX

# Nomenclature

## UPPERCASE LETTERS ( AND ACRONYMS)

| | |
|---|---|
| A | Reference point – output from IIMC controller model (inverse model No. 1 - |
| ActValue | the actual value from the training/testing set |
| ANN | Artificial Neural Network |
| B | Reference point–output from IIMC predictor model (inverse model 2 |
| BTR | Bed Turnover Rate (Reactor fluidised bed) |
| C | Reference point – 'Old' value of plant inputs ($u_{t-1}$) |
| $C_2$ | monomer Ethylene ($C_2H_4$) – 1 double bond |
| $C_4$ | co-monomer Butene ($C_4H_8$) – 1 double bond |
| $C_6$ | co-monomer Hexene ($C_6H_{12}$) – 1 double bond |
| CSTR | Continuous Stirred Tank Reactor |
| D | Reference point – Current value of IIMC suggested plant inputs ($u_{sugg\,t}$) |
| DCS | Distributed Control System |
| DMC | Dynamic Matrix Control |
| EKF | Extended Kalman Filter |
| F | Flow rate of species (i.e. FC2 – Ethylene flow rate) |
| Fo | Flow rate in the Tanks equation (Section 6.1) |
| FF | Feed Forward |
| G | Reference point – Current value of actual plant inputs ($u_t$) |
| $H_2$ | Hydrogen |
| HDPE | High Density Polyethylene |
| IMC | Internal Model Controller |
| IIMC | Inverse Internal Model Controller |
| $L_I$ | Tank Level (i) |
| LLDPE | Linear Low Density Polyethylene |
| LDPE | Low Density Polyethylene |
| MATLAB | Matrix Laboratory © . Computer program by Mathworks Inc. |
| Max_lim | Maximum scaling limit (0.9 in this work) |
| MIMO | Multiple Input Multiple Output |
| MI | Polymer Melt Index (Melt Flow Index) |
| Min_lim | Minimum scaling limit (0.1 in this work) |
| MPC | Model Predictive Controller |
| MSDEV | Microsoft Developer Studio©[C/C++ compilation/application software] |
| MV | Manipulated Variable |
| N | number of records tested |
| $N_2$ | Nitrogen |
| P | A representation (forward model) of the plant |
| P^ | A forward model of the plant |
| PE | Processing Element |
| PFD | Process Flow Diagram |
| PID | Proportional Integral Derivative (a controller) |
| PredValue | Predicted value of ANN model |
| PT | total number of examples presented to the ANN |
| PV | Process Variable – Value from the plant |
| Q | A controller (can be an inverse plant model) |
| R | Squared Correlation Coefficient |
| RLS | Recursive Least Squares |
| RPC | Resin Property Control |
| SCADA | Supervisory Control and Data Acquisition |
| SD | Standard deviation of the error |
| SISO | Single Input Single Output |
| SP | Set Point |
| SSE | Sum Squared Error |

| | |
|---|---|
| TEAL | TriEthylAluminium (Reactor Co-catalyst) |
| T | present time, use as a reference time to future and previous time steps |
| UNIX | Operating platform of the Polymer plant DCS |
| $V_i$ | Valve (i) Position |
| $Y_i$ | neural network prediction of $i^{th}$ output node |

LOWERCASE LETTERS

| | |
|---|---|
| d | unknown plant (load) disturbance |
| $dL_i$ | change in Tank Level |
| e | error correction |
| f | species flow rate |
| fi | unique weighting factor |
| ktpa | kilo-tons per annum |
| n | number of network outputs |
| $p_t$ | index of the example |
| r | setpoint for plant outputs |
| t | time steps, an interval |
| u | plant input |
| u^ | predicted plant inputs |
| u | plant input (can be a vector) |
| $u_{sugg}$ | suggested control action (plant input) from the controller |
| $w_{bias}$ | weight associated with the bias connection in a neural network model |
| $x_i$ | output signal from another node |
| $x_{min}$ | low scaling factor from the set of neural network training data |
| $x_{max}$ | high scaling factor from the set of neural network training data |
| y (or $y_p$) | measured plant output (can be a vector) |
| $y_i$ | actual value of $i^{th}$ output node |
| $y_{pred}$ (y^) | predicted plant output (can be a vector) |
| $y_{obs}$ | observed or actual plant output (can be a vector) |
| $y_{avg}$ | average (of observed data set) plant output (can be a vector) |
| $w_i$ | weighting factor for connection i |

SYMBOLS

| | |
|---|---|
| $\rho$ | density |
| $\Delta$ | change |
| $\alpha 1$ | filter constant for output 1 |
| % | composition of species (%) i.e. $\%C_2$ |

SUBSCRIPTS

| | |
|---|---|
| cont | controller model prediction |
| p | plant values |
| pred | predicted value |
| raw | raw variables to be scaled |
| sugg | suggested value |
| t | time at present |
| t+1 | time present plus 1 step |

# CHAPTER 1

## 1. SUMMARY OF THE WORK PRESENTED

### *1.1 LLDPE Polymerisation Reactor*

#### 1.1.1 Introduction

The primary system under consideration for the work in this dissertation was an ethylene polymerisation reactor. This gas phase reactor is situated on the SASOL POLYMERS Midlands site. The plant produces linear low-density polyethylene (LLDPE) which is an important commodity world-wide. The polymer is sorted into grades, characterised by its density and melt flow index. If these properties do not meet the quality specifications then the polymer is termed to be 'off specification' and must be sold for a lower price. This reactor and the plant are hereafter referred to as the polymer reactor and the polymer plant.

#### 1.1.2 Objective/ Control Problem

The desire for precise polymer control, minimum polymer wastage through grade transitions and early instrument fault detection has led to a significant effort in the modelling and control of ethylene polymerisation world-wide. The control of the feed flow ratios into this gas phase reactor is an area where further investigation is necessary since the present PID controllers are inefficient.

These flow ratios determine the gas composition ratios in the reactor and hence the product specifications in terms of density, melt index and production rate. The time lag between the changes to the feed flows and the resulting gas composition changes is approximately 50 minutes. The relationship between these flow ratios and the composition ratios is highly non-linear. At present individual PID controllers set the gas flow ratios for co-monomer:ethylene and hydrogen:ethylene. It was thought that some form of non-linear model predictive control algorithm would perform better than the current PID controllers if the model could take into account the process peculiarities.

This polymerisation process and the potential advanced control techniques that could improve the operation of the reactor were investigated, both onsite and in the literature. It was decided that two different control/modelling approaches would be necessary. These two proposals were as follows.

Proposal 1

1.1) An open loop on-line model predictive controller (MPC) was to be developed for the reactor. The models used were to be obtained by artificial neural network modelling. These neural network models are easy to train, very versatile and known to handle non-linear systems well. The models are to take into account the large time lags in the system and the complex non-linear coupled nature of the feed flow ratio/composition ratio relationships.

1.2) An innovative method of combining the traditional IMC (Internal Model Control) controller with inverse neural network models was developed. This was termed IIMC (Inverse Internal Model Control).

Proposal 2

2.) An on-line model/Kalman filter giving predictions of the states of the system would be developed. The parameters in the Kalman filter could be updated by an on-line recursive identification scheme (RLS). This work, which is not covered here, was developed in parallel to the work in this dissertation by Thomason [2000], a fellow M.Sc. Eng. research student at the University of Natal.

The relevant areas of Proposal 1 are covered in detail in this dissertation. This first chapter (Introduction) serves to introduce the reader to the aims of the project and the important theory that will be covered and developed later on.

## *1.2 Advanced Control Techniques*

Advanced control can be described as a systematic, studied approach to choosing pertinent techniques and their integration into a co-operative management and control system that will significantly enhance plant operation and profitability [de Vaal, 1999].

The description above of advanced control embodies the reason for initiating this project. The gains that would be realised by improving control of the gas ratios would be very quickly translated into financial savings. These savings would result mainly from the minimisation of off-specification polymer product.

Model Predictive Control (MPC) is the collective name given to the family of controllers that directly use an explicit and separately identifiable model [Garcia, Prett, Morari, 1989]. The form of controller that was proposed falls within the category of IMC. This form of advanced

controller was chosen because of its proven track record and it's suitability for almost any kind of problem [Garcia et al, 1989].

The major difference of the proposed IIMC controller to the traditional IMC structure as shown in figure 1.1 is that the internal forward model (P^) is replaced with an inverse internal model. There would be several advantages to using two inverse models in this type of controller. This is discussed in detail in sections 3.4 and 5.2.3.

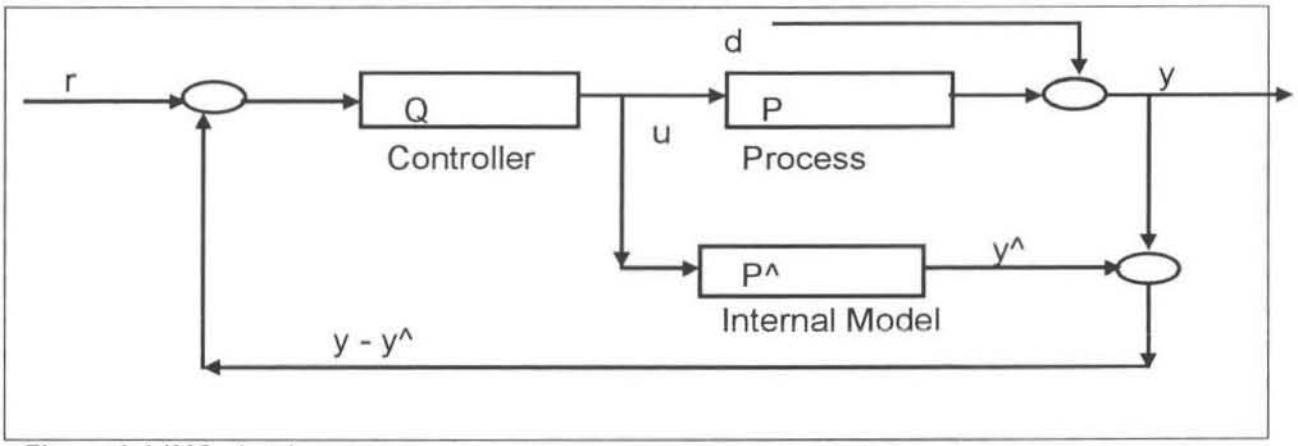


*Figure 1.1 IMC structure*

The IIMC control structure was tested (section 6.2) on an interacting tanks system in the School of Chemical Engineering at the University of Natal and was found to compare favourably to a DMC (Dynamic Matrix Control) controller.

## *1.3 Artificial Neural Networks*

Artificial neural networks were chosen to model the system because of their suitability to this problem. They are known to handle non-linear models well and this, combined with their ease of use, made them a favourable choice.

They are classified as a black box mathematical non-linear regression tool. This means that neural networks can be used to learn and identify correlative patterns between sets of input data and corresponding outputs. In terms of process control, we can equally train them to predict the control actions which lead to a given desired response. Once trained these networks can be used to predict outputs from new input sets. The training of these networks is a form of non-linear regression and has developed into a fairly tried and trusted technique.

It has been shown [Hornik & Stinchcombe, 1989; Cybenko, 1989] that any continuous function can be approximated to an arbitrary degree of exactness by an appropriate neural network model.

One of the main foci in this dissertation has been to find accurate plant models. This has proved to be very challenging since the system is complex, and the reactor gas-space acts as a pure integrator. There are a number of important variables to be included in the modelling and the time lag between the flow ratio change and the composition response is of the order of about 50 minutes. Several inverse plant models have been developed. These range from simplified steady state models considering only feed flow ratios and gas compositions to large time-invariant dynamic models, spanning several points in time, taking many variables and states of the system into account.

In MPC the quality of the controller depends to a large degree on the accuracy of the plant model. If a model was able to exactly replicate the plant's behaviour then the controller would be able to perfectly control the plant [de Vaal, 1999] - however this is not feasible when considering a time-invariant dynamic, complex MIMO plant. This is where a suitable controller structure must be chosen, preferably one that is robust is the face of plant-model mismatch. It is hoped that the proposed IIMC controller will achieve this robustness.

# CHAPTER 2

## 2. THE PRODUCTION OF POLYETHYLENE

### *2.1 Polyethylene*

#### 2.1.1 An important world-wide commodity

"Linear Low-Density Polyethylene (LLDPE) has established itself as the third major member of the world polyethylene business." This is according to an article by Schumacher [1996]. Another article by Davidovici [2000] states that in 1999 polyethylene accounted for 46.5 million tons (45%) of the total of 106 million tons of polyolefins used world-wide. It also projects that the expected growth for the next 5 years is expected to be 5-6%, which will result in the total demand reaching 60 million tons by 2005.

SASOL POLYMERS (formerly POLIFIN) is the largest producer of LLDPE in Southern Africa ahead of SAFRIPOL, the principal competitor, and is in an ideal position to keep its status as the main supplier to the Southern African market [Royle, 1998]. SAFRIPOL has recently been taken over by DOW CHEMICALS. It is clear that the polyethylene market and in particular the LLDPE market is set to grow well into the future and that SASOL POLYMERS should do everything possible to keep its competitive edge, especially with the influx of foreign competitors like DOW CHEMICALS.

The figure 2.1 shows the uses of polyethylene. The properties of the polymer are characterised by its MI (Melt Index or Melt Flow Index - MFI) and density. These properties are discussed later.

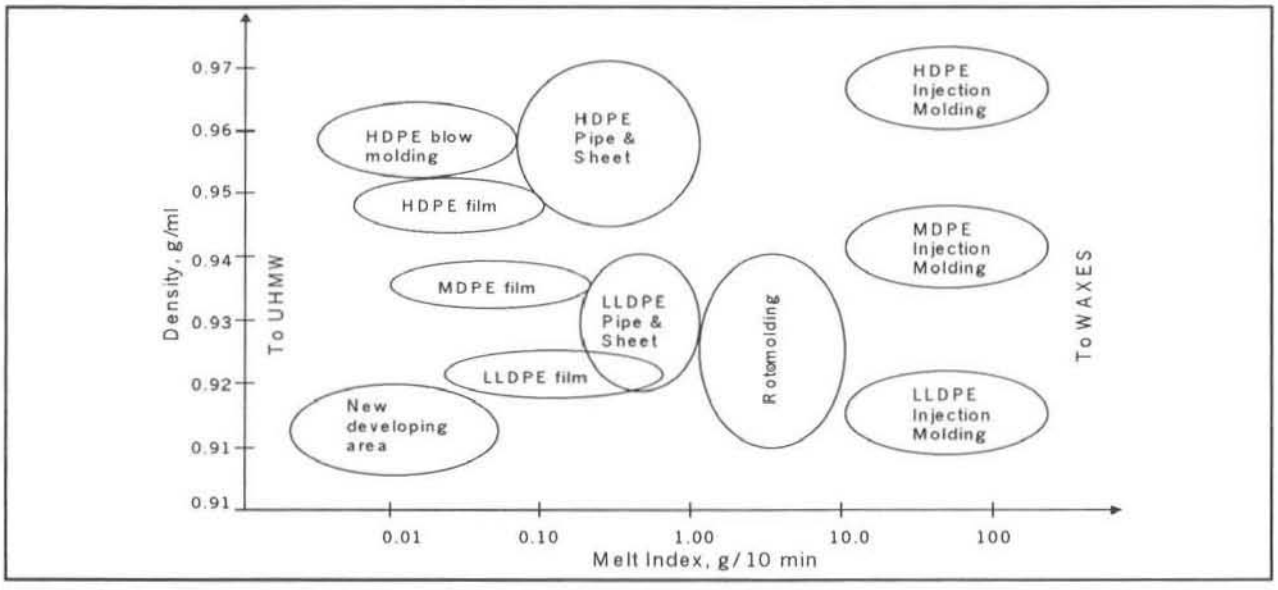


*Figure 2.1: The Uses of Polyethylene (James, 1986 and Foster, 1991)*

### 2.1.2 The Production of Polyethylene

According to Davidovici [2000] there are about 56 distinct process variations for the production of polyethylene world-wide. Of these 25-30 are available commercially for licensed production. The main processes are:

High Pressure

- Autoclave
- Tubular

Low Pressure

- Gas Phase
- Solution
- Slurry

The linear low density polyethylene (LLDPE) grades are produced mainly by the Gas Phase and Solution Phase processes. Both of these processes have the ability to operate as swing plants, producing LLDPE and HDPE (high density polyethylene).

The Gas Phase process makes use of a fluidised bed reactor. A bed of solid polymer powder is maintained in fluidised state by the flow of reactant gases. The heat of polymerisation is removed by circulating the hot gas through a large cooler. The gas is compressed and returned to the base of the reactor. The productivity of the reactor is determined by the catalyst activity and the heat removal capabilities.

The main feedstock to the reactor is high purity ethylene and an alpha-olefin co-monomer (typically 1-butene or 1-hexene). The polymerisation occurs upon the introduction of a transition metal catalyst (Ziegler-Natta or Chromium) at operating conditions of between 80-100°C and 20–30 bar.

The inventors of the licensed UNIPOL Gas Phase process, UCC (Union Carbide Corporation) discovered in the 1980's that by using a heavier co-monomer (1-hexene) plant production could be increased by up to 60%. This is because it condenses in the cooler and then needs to re-evaporate in the reactor, thereby removing heat in the form of latent heat of evaporation. This is known as running the plant in 'condensed mode' [Davidiovici 2000].

### 2.1.3 The Structure of Polyethylene

Polyethylene is ethylene that has polymerised to form long chains, see figure 2.3. The amount of branching in these chains is determined by the amount of co-monomer and hydrogen added

to the reaction. This branching affects the polymer characteristics, namely MI (melt index) and density.

Ethylene ($C_2H_4$) is a simple molecule with a double bond between the 2-carbons (see figure 2.2). It is a gas under normal conditions and can be produced from crude oil or by SASOL'S unique coal to gas process. 1-Hexene ($C_6H_{12}$) is simply a longer version of ethylene, having 6-carbon atoms per chain and 1 double bond. Similarly 1-butene ($C_4H_8$) has a 4-carbon atom chain and 1 double bond. 1-Butene is a gas under normal conditions while 1-hexene is a liquid. The '1' signifies that the double bond is in the first position of these straight simple carbon chains. Hereafter in this work the 1 prefix is dropped and the co-monomers are known as butene or hexene. The addition of hexene or butene to the reactor results in short chains forming on the side of the otherwise straight polyethylene molecules, see figure 2.4. This effectively reduces the density of the polymer since the chain cannot pack together as tightly as before. The use of hexene results in longer side chains and the formation of a superior product to using butene [Royle, 1998].

The addition of hydrogen to the reactor acts as a terminating agent for the polyethylene chains. This lowers the average molecular weight of the polymer and increases the melt index.

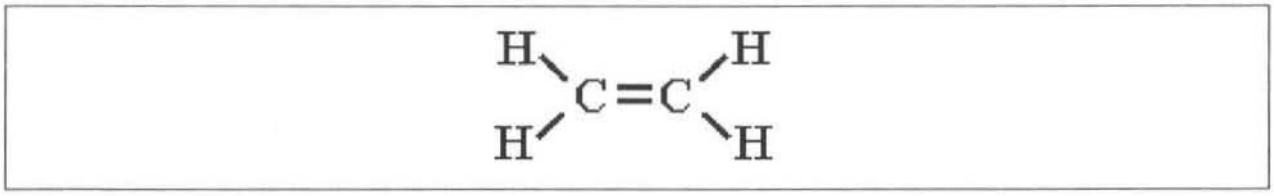

*Figure 2.2: A single ethylene molecule*

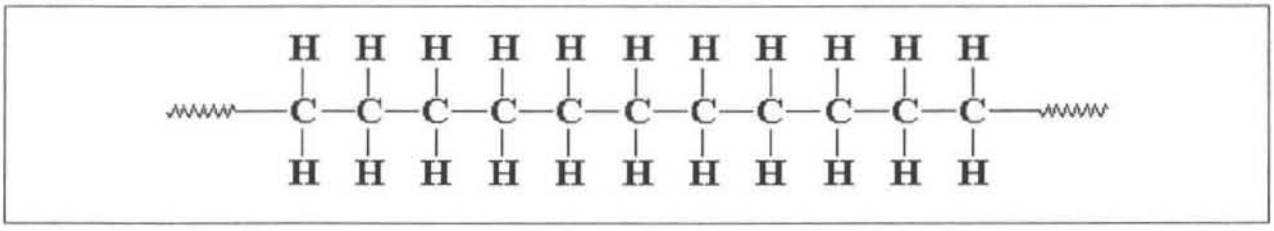

*Figure 2.3: An un-branched Polyethylene molecule*

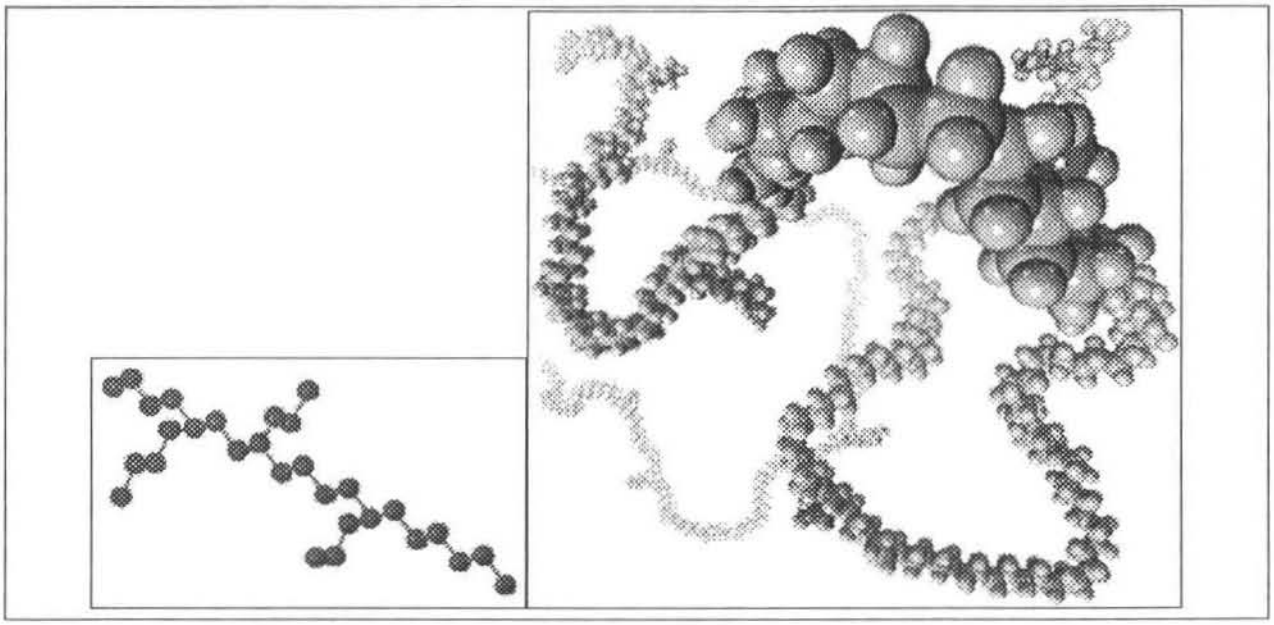

*Figure 2.4: Branching of a Polyethylene molecule [The Scientific American (cover page), May 1997]*

### 2.1.4 Chemical reactions

There is a wide range of reactions that occur during the polymerisation process. Many models have been developed that are based on site balances and these are very complex [McAuley, MacGregor, Hamielec, 1990]. The reactions can be summarised into four main stages that result in the formation of polymer:

- Formation of an active site on the catalyst
- Initiation of the polymerisation (polymer molecule at active site)
- Propagation of the polymer chain
- Transfer (or termination) of the polymer chain

To go into any more detail is beyond the scope of the work in this dissertation, and the reader is referred to the work of Thomason [2000] where the reactions had to be modelled as part of a deterministic description.

## *2.2 The Polymer Reactor*

### 2.2.1 Introduction

The polymer reactor is the heart of a 105ktpa linear low-density polyethylene (LLDPE) plant operated by SASOL POLYMERS (formerly POLIFIN LIMITED) in Sasolburg. The technology used is Union Carbide Corporation's (UCC) gas-phase UNIPOL process. In 1998 this process was

upgraded to allow for hexene to be used as a co-monomer. This upgrade resulted in the plant capacity increasing from 85ktpa to it's present capacity and resulted in improved reliability [Doyle & Rackham, 1998].

The UNIPOL process has the capability to produce resins with different grades that can be used in the manufacture of plastic sheeting, packets, roto-moulded objects and piping. The polymer plant is only licensed to produce LLDPE resins.

The ethylene feedstock used in the process is supplied via a gas pipeline from the nearby SASOL plant also in Sasolburg. This supply of ethylene can vary as a result of fluctuations in SASOL'S production. On the SASOL POLYMERS Midlands site an older polyethylene plant (Poly 1), which produces LDPE (low-density polyethylene), has first preference for the available ethylene feedstock. The remaining ethylene is then used by the polymer plant under consideration. The reason for this is that the local market demands for LDPE are more favourable than for LLDPE.

The LLDPE produced by the polymer plant is mainly used for the production of plastic bags, a commodity that will no doubt be in demand for the next few years at least. The various grades of polymer produced can be processed into a variety of different plastics, ranging from the thin grocery type to the thicker more durable type that fertiliser usually comes in.

Considering these factors it seems reasonable to assume that the polymer plant should have sufficient market for it's LLDPE products in the near future to warrant attempts to improve the control strategies on the plant to increase it's profitability. The advantages of better control of the reactor are fairly obvious without going into an in-depth cost-benefit analysis.

At full capacity the plant produces about 19 tons per hour of polymer and uses a similarly large amount of ethylene feedstock. The polymer is the main source of revenue for the plant and the ethylene is the main raw material cost. The figures involved here are in the order of millions of Rands per annum and the savings from efficiency improvements could be very significant.

If a disturbance causes the operating conditions to change slightly, upsetting the plant for a few hours, and the polymer produced falls out of product specification, then the losses incurred can quickly escalate to significant proportions. Similarly, during a grade transition the amount of wasted polymer is very important. By controlling the gas ratios optimally through such disturbances and transitions it would be possible to reduce these losses.

### 2.2.2 The Polymer Plant Process

As mentioned the polymer plant uses Unipol gas phase technology to produce LLDPE. The Unipol reactor forms an integral part of the polymer plant process, which begins with catalyst production and finishes with the product polymer pellets. A very simplified process flow diagram of the major components of the process is shown in figure 2.5.

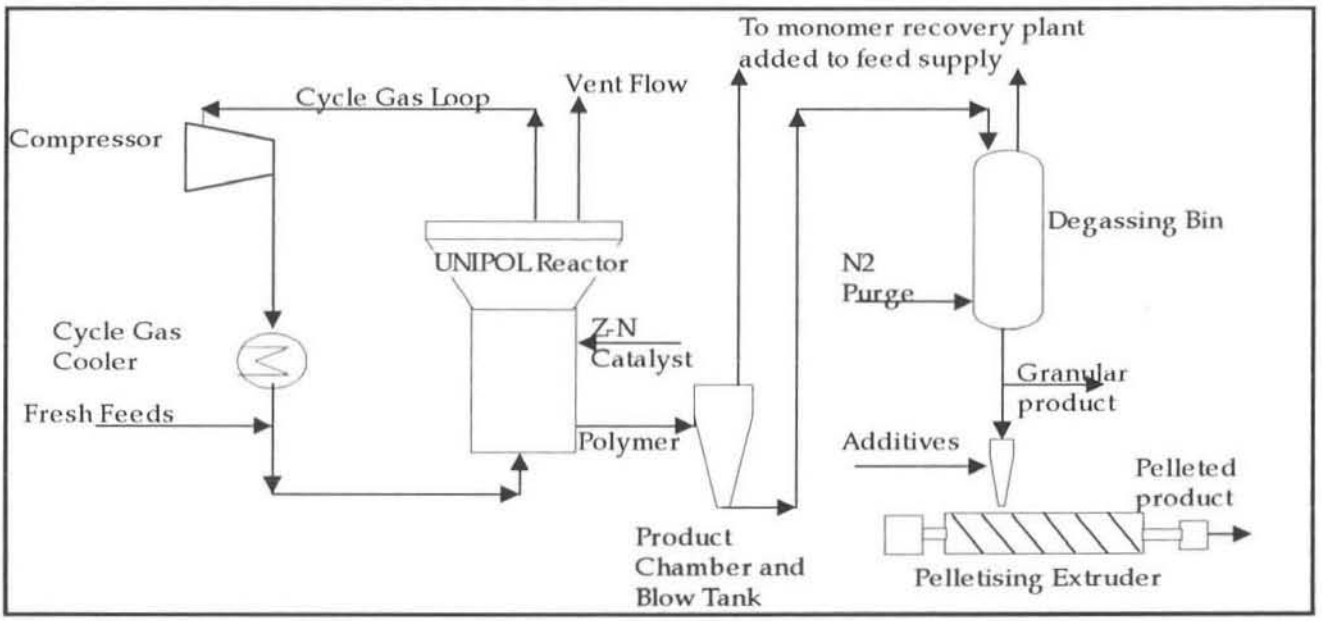


*Figure 2.5: Simplified PFD of the Polymerisation Process*

The Ziegler-Natta catalyst, which is continuously consumed as it passes out in the product resin, is produced on-site in a separate catalyst plant. The ethylene and co-monomer feedstocks are supplied from nearby plants. From the reactor the solid polymer particles are transported to several downstream processing units before the pelletised product is obtained.

The solid polymer product is discharged from the bottom of the reactor periodically, usually 10 – 14 times per hour. This depends on the volume of polymer in the bed reaching a certain setpoint. From the discharge chamber the product moves to another vessel, the product blow tank, in which the polymer resin and the gas separate. The gas is fed back into the reactor.

All processes hereafter are referred to as downstream processes. The solid product is moved with the use of de-oxygenated nitrogen to a large degassing bin. In here the remainder of the gas, monomer and co-monomer, is removed from the resin and the gas goes to a co-monomer recovery plant. The resin at this stage of the process is very similar in appearance and texture to washing powder. The granular polymer is stored in large silos until it is processed and blended to the final product at the extruder plant. Special additives and chemicals are added here to ensure the final polymer meets industry standards of impact strength, tensile strength,

tack etc. The final form of the polymer is that of a small, clear spherical pellet. These are packaged and sold depending on grade and quality.

### 2.2.3 The UNIPOL Reactor

The heart of the polymer plant process is the UNIPOL gas phase polyethylene reactor. It is in this fluidised bed reactor that the gas phase ethylene feedstock is converted into the solid polyethylene particles.

This large reactor vessel, which is designed to produce 19tons per hour of polymer, stands over 40 metres high and consists of a straight section and an expanded upper section. It is considerably narrower below its expanded top section. The latter allows the fluidised polymer particles to disengage from the reactant gas.

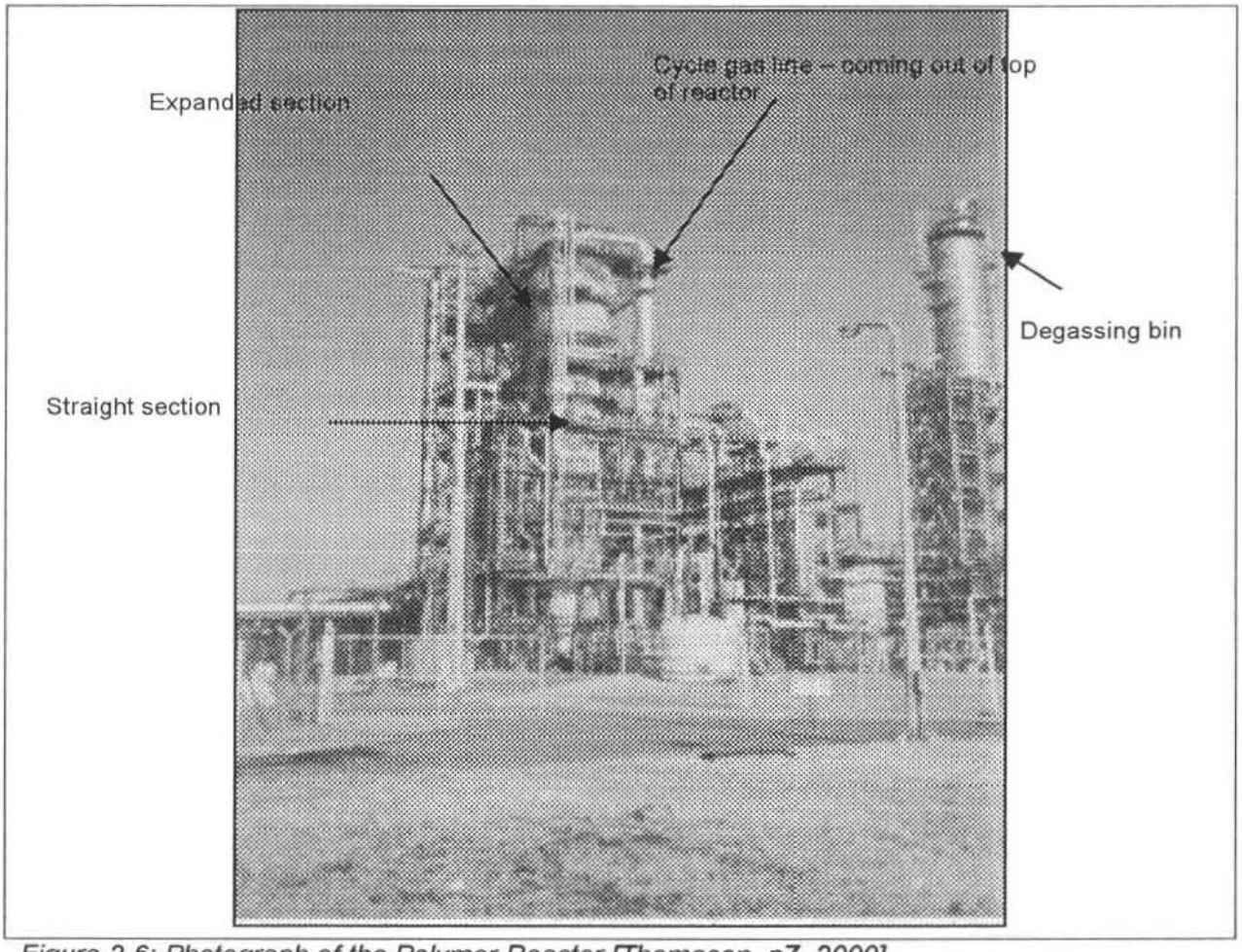


*Figure 2.6: Photograph of the Polymer Reactor [Thomason, p7, 2000]*

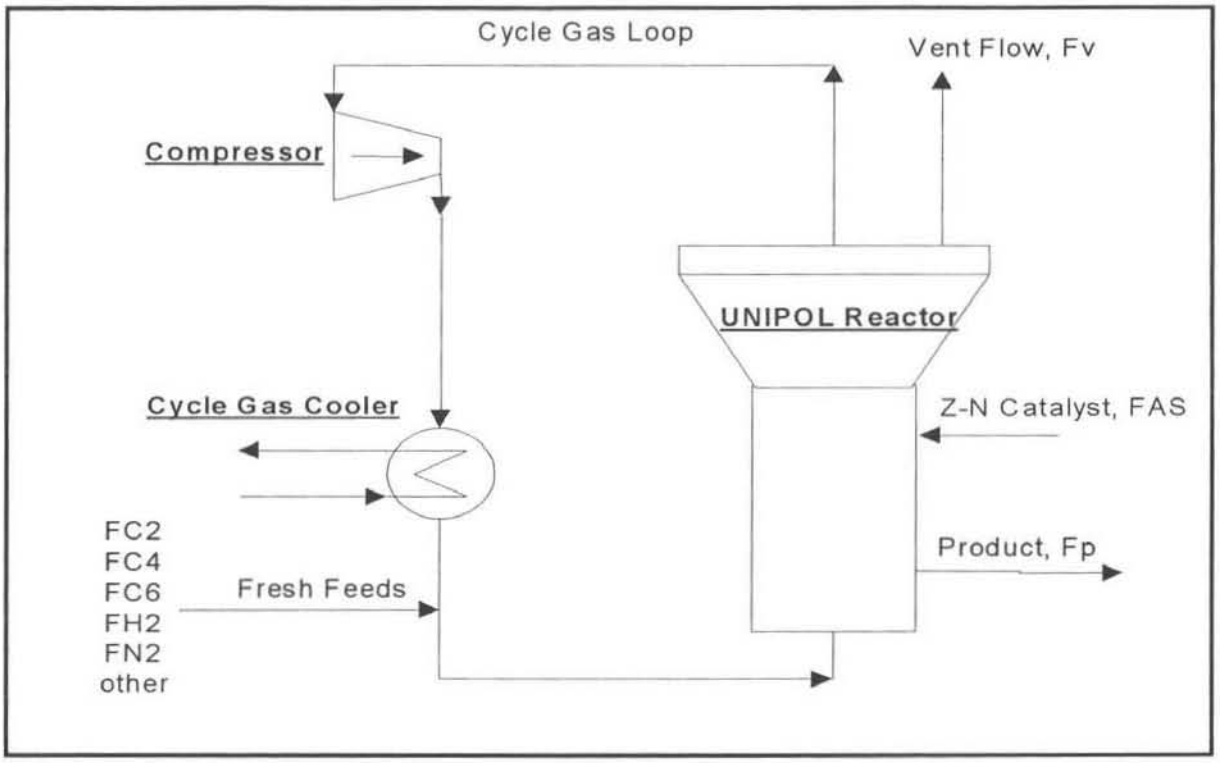


*Figure 2.7: The UNIPOL Reactor*

The feed to the reactor consists mainly of the monomer ethylene, the co-monomer (butene or hexene) and hydrogen and nitrogen. These gaseous feed streams enter the cycle gas line after it is cooled in the cycle gas cooler. Depending on the mode of operation (condensed or non-condensed) a certain amount of 1-pentane may also be added. The pentane effectively increases the boiling point of the gas mixture, thus increasing the amount of condensate formed in the gas cooler. This helps to increase the productivity of the reactor by increasing the amount of heat transferred (see section 2.1.2).

The un-reacted gases are recycled, by means of a large gas compressor, and combined with the fresh feeds before re-entering the base of the reactor. Conversion in the reactor per pass is very low (2-3 %) therefore the recycle gas stream is much larger than fresh feed streams. This recycle stream usually contains about 7 tons of gas.

Due to the highly exothermic nature of the reaction, heat must be removed from the system via the cycle gas stream. This operation is done in a large heat exchanger, the cycle gas cooler, which uses tempered water as a cooling medium on the shell side. This gas cooler is very important in the reactor's operation since it's part of the main temperature control mechanism. The temperature of the reactor feed is carefully controlled since it affects the bed temperature and consequently the polymer grade. For example if the required bed temperature is 88°C at a certain production rate, then the feed needs to be at 33°C. The tempered water supplied to the

gas cooler is in a closed loop and can be heated or cooled quickly depending on control requirements.

Catalyst and co-catalyst are also fed continuously into the reactor. The heterogeneous Ziegler-Natta catalyst has a high reactivity and only small flows are required. This catalyst is added as a solid powder and is delivered by two screw feeders that operate simultaneously. These identical feeders are at approximately 1/3 of the way up the side of the reactor. The cocatalyst is triethyl aluminium (Teal, $AlEt_3$). This is only used with certain polymer grades.

The ethylene flow is manipulated to maintain the ethylene partial pressure in the reactor and the co-monomer and hydrogen flows are fed in a set ratio to this ethylene flow. The gases in the cycle gas line supply reactants to the growing polymer chains and provide the fluidisation and heat transfer media. These polymer chains initiate and grow in the fluidised bed.

The UNIPOL gas-phase technology is highly versatile in that it allows a wide range of polymer grades to be made. These grades are classified in terms of the properties melt index (MI) and density ($\rho$) and are characterised by grade codes. For example the product code HFM2010, indicates a hexene grade polymer that is intended for use in plastic film sheeting and has a density of 910 and MI of 2.0.

These properties are measured every hour in an on-site laboratory. The measurements are currently used to update the model in the proprietary on-line resin property control software. These properties are a function of the gas compositions in the reactor and several operating conditions. The relationships are highly coupled and non-linear.

## *2.3 Current Control Scheme on the Polymer Plant*

### 2.3.1 The Distributed Control System

The polymer plant is equipped with a sophisticated modern DCS (Distributed Control System). The immense functionality of such a system means that all control functions from low level PID control, to operator inputs and decisions, and high level advanced control algorithms are handled from a central location with ease.

A basic figure, taken from Narotam [1999], depicting the architecture of this system is shown in figure 2.8. The code that was written for the Internal Model Controller was programmed into the system on a User Application workstation. From this workstation the code could be compiled

and executed and run on-line with real time plant data. This code and the implementation are discussed in more detail in chapter 8 and Appendices C to H.

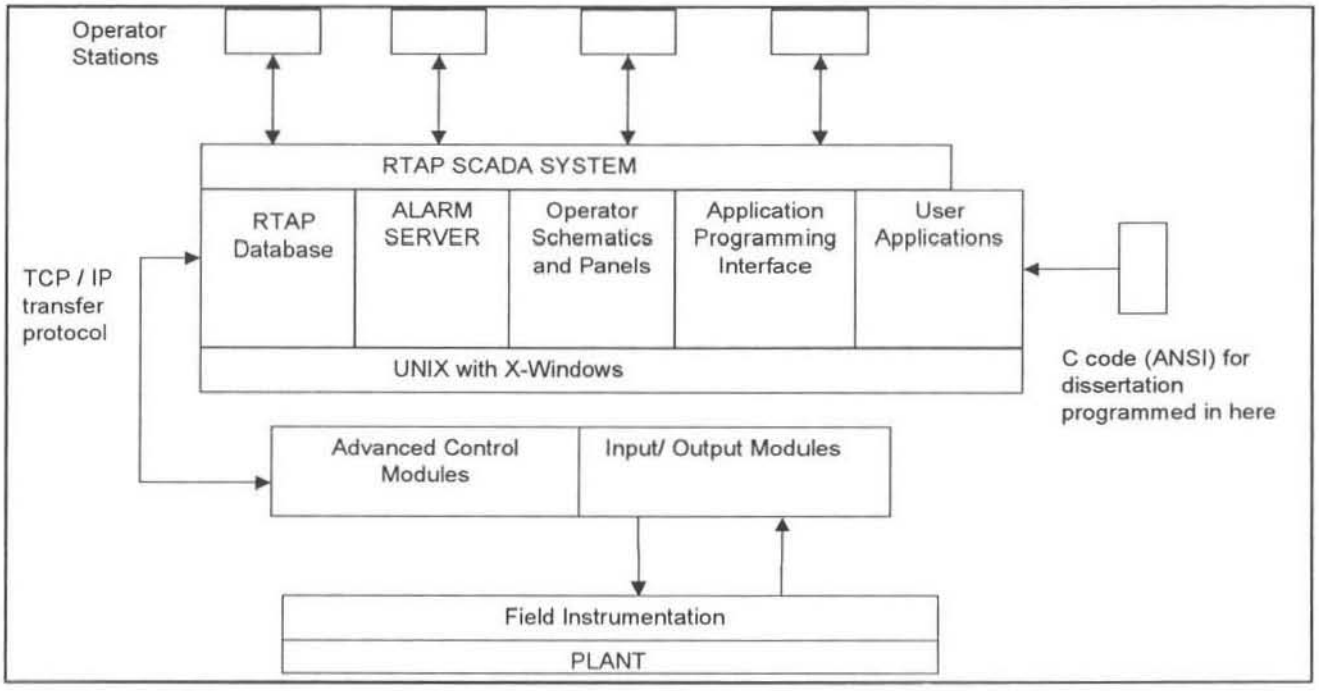


*Figure 2.8: The DCS (Distributed Control System) [Narotam, p 69, 1999]*

### 2.3.2 The Overall Control Objective

The DCS system has been configured to meet certain operating objectives:

The primary objective of the control scheme is to minimise the quantity of off-specification polymer product. This is achieved in several ways.

- Maintaining gas compositions and reactor conditions as specified by the specific grade recipe of polymer being produced.
- Early detection of faulty plant measurements, especially on-line gas analysers.

A secondary objective is to minimise the time taken to make polymer grade transitions. This is an area where MPC could be especially useful in predicting optimum trajectories of the gas compositions and flow rates.

### 2.3.3 The Plant Control Scheme

This DCS controls the operation of the entire process, excluding the extruder and blending plant. The gas-space acts as a pure integrator – ie. feeds which are not in proportion to the reaction ratios simply accumulate, requiring expensive venting. The process is open-loop

unstable, and it has a large dead-time associated with the gas space compositions. A general overview of the control system topology is shown in figure 2.9.

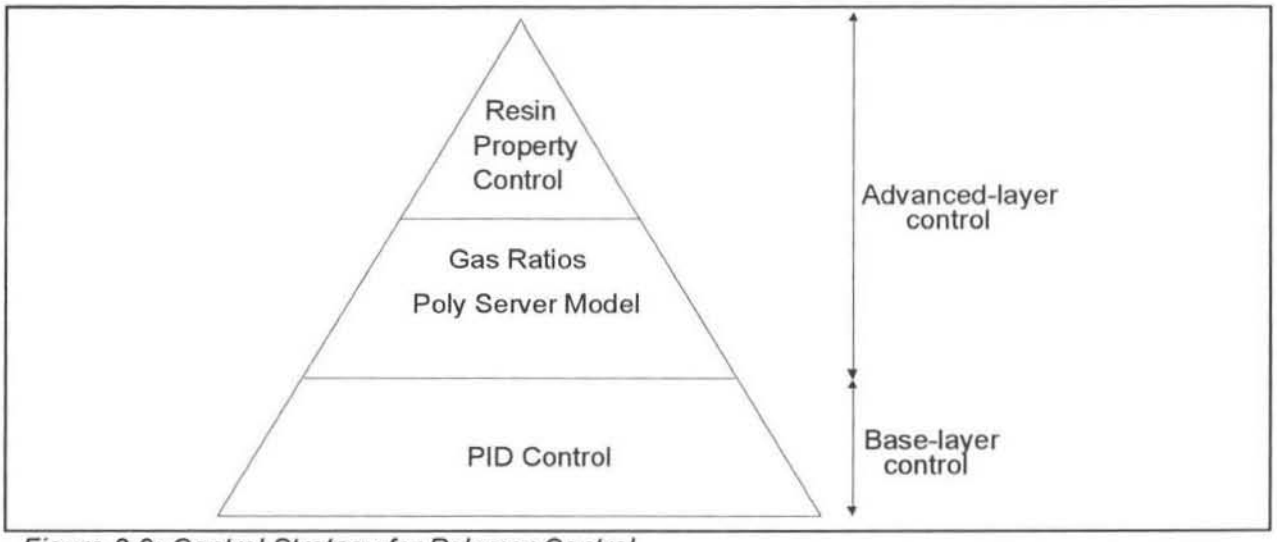


*Figure 2.9: Control Strategy for Polymer Control*

The base-layer is used for control over the reactor environment, that is temperatures, pressures and flow PID loops. These basic manipulated variables are handled by the ACM (Advanced Control Modules), see figure 2.8. The advanced-layer of control predicts the gas compositions required in the reactor loop to produce a certain polymer grade. This consists of two further sub-layers.

The top layer is known as the resin property control (RPC). The polymer properties (MI and density) cannot be measured on-line. These measurements are inferred by an on-line model and are corrected periodically by actual lab measurements. Polymer property setpoints can be set based on the current properties and the desired grade of the future polymer. The purpose of this layer is to provide the next control layer with these polymer property setpoints.

The setpoints and current MI and density measurements are the inputs to the next control layer. Here the Poly Server Model software, supplied by the technology vendor (UNIPOL), predicts the gas compositions ($H_2/C_2$, $C_4/C_2$ or $C_6/C_2$) required to achieve the desired polymer grade.

These gas composition setpoints are sent to the base-layer PID controllers. A PID ratio controller considers the setpoints and the current gas compositions (from the gas analysers) and predicts the flow ratios ($H_2/C_2$, $C_4/C_2$ or $C_6/C_2$) required. The output of these ratio controllers is multiplied with the current measured ethylene flow rate to achieve a setpoint for the butene, hexene and hydrogen flow control loops. This ethylene flow rate is manipulated in an independent loop to maintain the ethylene partial pressure.

A diagram showing the flow of information through the control hierarchy is shown in figure 2.10. Only the example for the co-monomer hexene is illustrated here. The procedure is the same for hydrogen flow rate. This control scheme, shown in figure 2.10, is the most important from the point of view of the work in this dissertation.

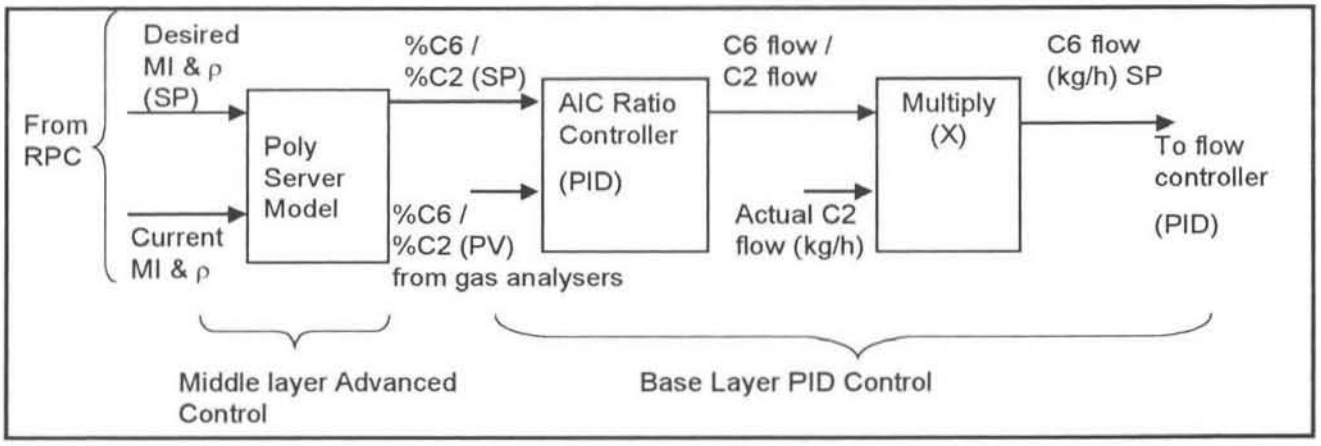


*Figure 2.10: Information flow for determining 1-Hexene flow rate SP*

The other manipulated and controlled variables that are also relevant to the operation and modelling of the reactor are :

- The temperature in the reactor
- The pressure in the reactor
- Production rate of the reactor
- The catalyst feed rate into the reactor
- The flow of catalyst activator (TEAL) to the reactor.

The control of these variables is not discussed in detail here.

### 2.3.4 Control Problems and Proposals for Research

A problem area that was identified by the polymer plant representatives is shown in figure 2.10 above. The PID controller that converts that composition ratio setpoints to flow ratio setpoints is not ideally suited to the task. The reactor has certain characteristics that make the work of this PID controller difficult:

- The gas space in the reactor is an integrating system, which means that if a certain gas flow is increased step wise, its composition will continue to ramp until something is done to prevent the rise. This basically means that the system is open-loop unstable.
- A long response time exists between flow ratio changes and visible composition changes. This is in the order of 20 to 60 minutes.

- The relationships between the flow ratios and composition ratios are highly non-linear. These relationships are also coupled, requiring a multivariable approach. The current PID configuration does not take this coupling into account. Hence if the hexene ratio needs to be raised, more hexene will be added to the system. However, this will cause the other ratios to change. Their respective controllers will then try and accommodate for this and so the controllers are continuously interacting sub-optimally.
- The process variables (gas composition ratios) are available from the gas analysers. These analysers have a dead time (cycle time) of approximately 3 minutes.

In the light of these problems it was decided to investigate an alternative controller to convert the composition setpoints to flow setpoints. An MPC (Model Predictive Controller) format was thought to be the best option, using a multivariable model able to handle process non-linearities. The controller was required to be stable and robust and should also be able to account for plant disturbances. A modelling technique was needed that would incorporate all of the dead time and process non-linearities into the model structure, leading to the choice of an artificial neural network (ANN). The reason for this choice is discussed in more detail in chapter 4. The decision was made to test an IMC structure using these neural network models. This proposal is depicted in figure 2.11.

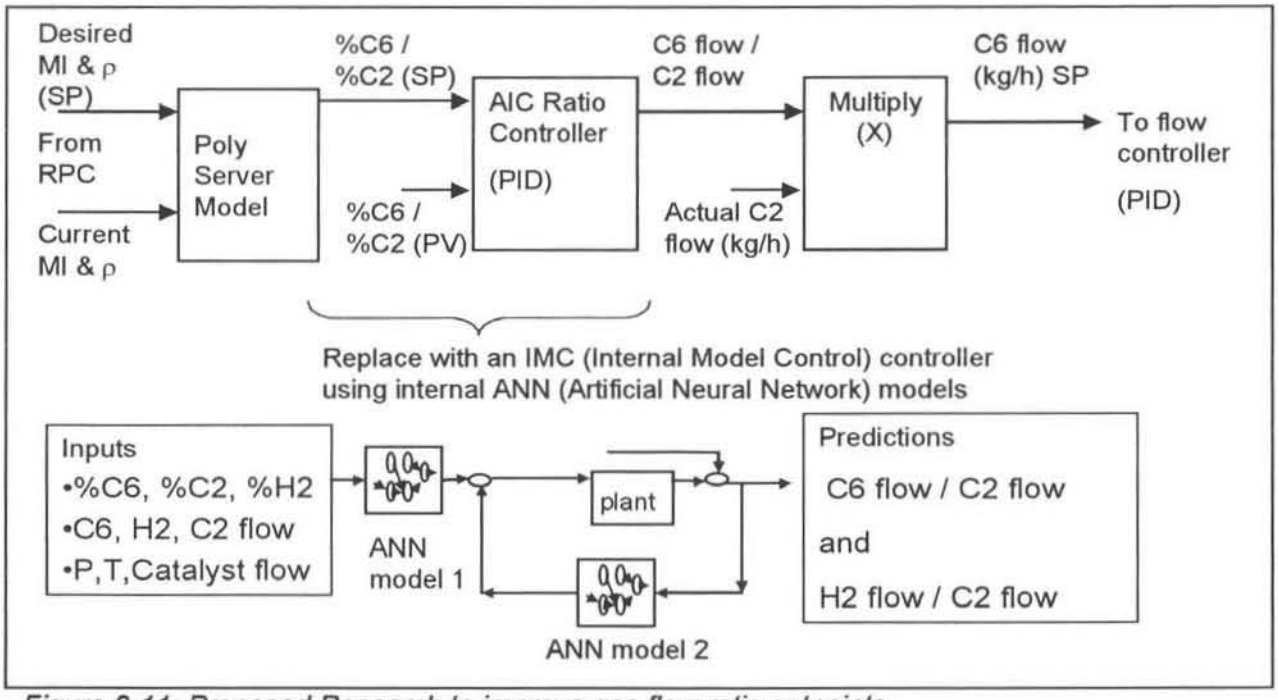


*Figure 2.11: Proposed Research to improve gas flow ratio setpoints*

# CHAPTER 3

# 3. MODEL PREDICTIVE CONTROL

## *3.1 Introduction to Model Based Control Techniques*

In order to retain a competitive edge in today's marketplace, process plants are going to need to make use of more advanced control techniques.

Modern control techniques have progressed from the traditional base-layer PID feedback loops to model-based optimisation algorithms. These algorithms are designed to result in overall plant optimisation and efficiency. The process models that form the foundation of these controllers can be anything that describes the system's behaviour. Examples of common models used are first principle kinetic models, statistical correlations, lumped parameter models, black box neural networks, time series functions or qualitative fuzzy logic process descriptions. These models can be linear, non-linear, simple or complex.

All models however are bound by the same limitation of being imperfect. They will always contain some degree of error, usually termed "plant-model mismatch". This is inevitable, since no model will be able to exactly replicate the complex behaviour of a multi-variable, multi-state industrial process.

An important factor to consider when choosing a modelling technique and developing a model is whether the model is invertible or not. There may not be a unique relationship in the forward or reverse direction of the model – for example there may be several combinations of the inputs that will achieve the same output. In such cases other criteria are brought to bear to determine "optimal" control actions, eg. the expense of the actions or penalties for deviations from set-point.

### 3.1.1 Types of Process Models

A forward model is designed to mimic the system, giving predictions of the process outputs, perhaps taking a setpoint or the current and/or future control action into consideration, see figure 3.1b below. Take the example of a simplified plant where the system is a reactor, the inputs to the system are reactant flow rates, and the outputs are the compositions of the products. In this case the inputs to the forward model could be the current and/or past flow rates and perhaps the current and/or past compositions. The model could predict the next product compositions given this set of inputs.

This type of model, when accurate, is useful since it can be used to test the plant's response to selected different types of input disturbances whereas the model itself may have been trained with multiple variations in effect. Controlled sensitivity tests can be performed on the model, making it possible to identify the interaction between variables, the pairing of input and output variables and the extent and conditions of interactions in the system. In other words we can separate out simple relationships.

Another use of forward plant models is seen in the IMC structure in figures 3.2a and 3.2b. The process model (P^) can be used as the 'Internal Model' and is used to calculate a model offset error from the plant outputs. The feedback error term is filtered and is used to correct the controller (Q), on the assumption that the controller itself is an exact inverse of the model.

An inverse model of a system does the opposite to a forward model. The inputs to the model are the desired plant outputs (setpoints). It will predict the future input (control action) to a system that is required to produce those outputs. Continuing the example from above, an inverse model would take as one of its inputs a desired product composition (usually 1 time step ahead) and the model would predict the control action that would move the plant towards achieving that composition, as in figure 3.1c. For the purpose of process control, inverse plant models are far more useful than forward models. In figures 3.2a and 3.2Sb, the controllers (Q) can be inverse process models.

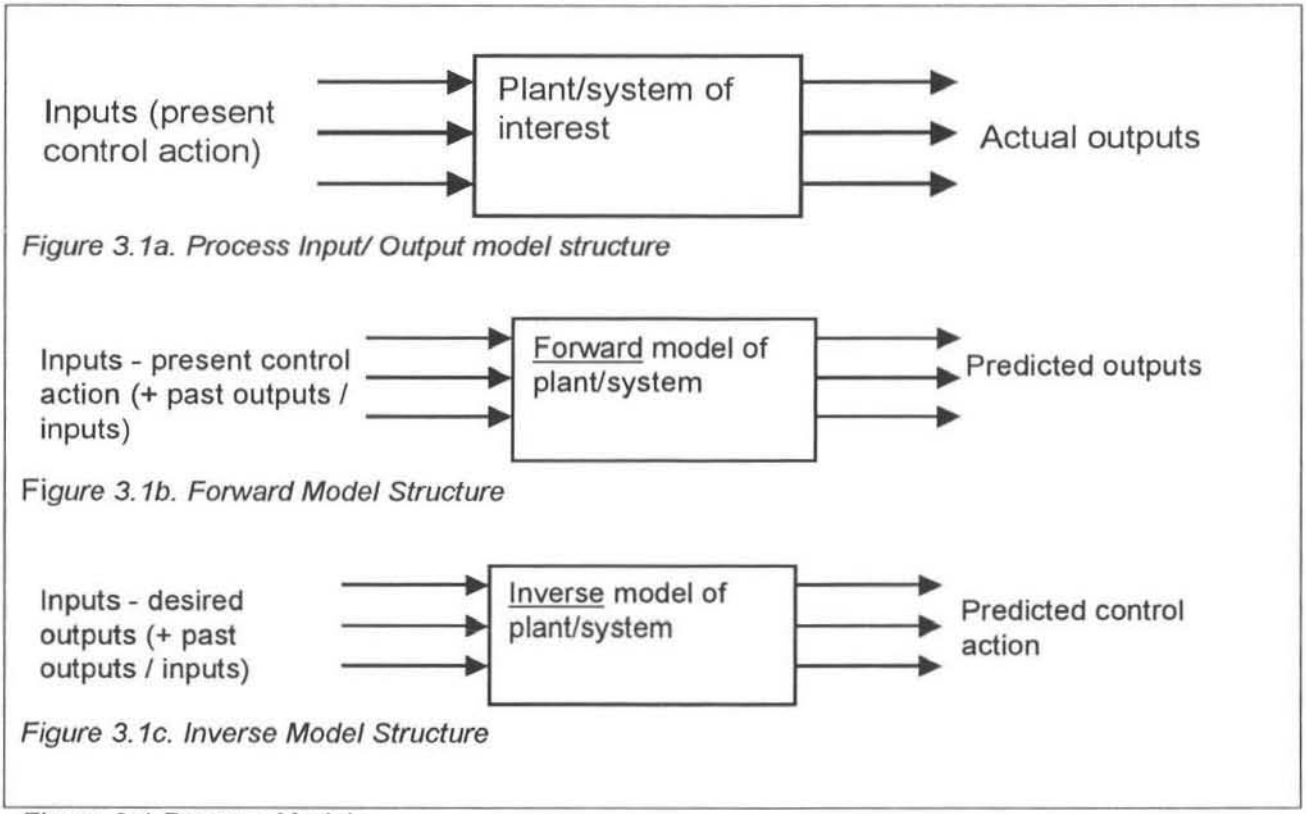


*Figure 3.1a. Process Input/ Output model structure*

*Figure 3.1b. Forward Model Structure*

*Figure 3.1c. Inverse Model Structure*

*Figure 3.1 Process Models*

## 3.2 Model Predictive Control – Introduction

As mentioned in the introduction MPC is the collective name given to the family of controllers that directly use an explicit and separately identifiable model [Garcia et al, 1989]. Basically what this means is that the controller makes use of a model of the system. This controller gives predictions of the control action that should be taken to result in a certain plant setpoint being reached. After the control action is taken a plant-model mismatch is calculated and this error is used to correct the future control action.

MPC controllers have increased in popularity in recent years as is shown by the number of applications in industry [Pike et al, 1996]. As the name suggests, MPC controllers rely on a process model for robustness and stability. The quality of the controller is affected to a large degree by the accuracy of the models. The theory of these MPC controllers and their stability issues are dealt with in great detail in many references [Garcia et al, 1989; de Vaal, 1999].

The main areas where MPC controllers have be shown to be effective are in systems with the following characteristics [de Vaal, 1999].

- a large number of manipulated and controlled variables
- constraints imposed on both the manipulated and controlled variables
- changing control objectives and/or equipment (sensor/actuator ) failure
- time delays

As seen in the previous chapter the polymer plant meets all of the above criteria. It is in systems like those above that other methods of controller design become non-trivial. Although MPC methods are by no means simple, many of the MPC structures such as the IMC technique result in robust and stable controllers without much effort. This allows the designer to concentrate more on obtaining accurate plant representations by any of the many modelling techniques available than on ensuring the stability of the multivariable controller.

According to Pike et al [1996] the main advantages and disadvantages then of using MPC methods are:

Advantages

- relatively easy to tune
- able to handle constraints
- can be applied to non-linear processes
- open methodology in designing predictive controllers within the common MPC framework

- applicable to a wide range of processes including those with dead time, and/or open-loop instability.

Disadvantages

- an accurate process model is required
- MPC methods assume that the future setpoints are known
- Theoretical guaranteed properties for MPC methods is still a developing subject

As mentioned earlier the models used in MPC controllers may be non-linear or linear. An example of a linear MPC algorithm is DMC (Dynamic Matrix Control). This is a very well known and widely used control technique. However since it extrapolates step responses in a linear fashion to obtain control actions, it is not entirely suitable for controlling non-linear processes. For this reason a DMC controller was not considered as an option for advanced control for the polymer reactor.

## *3.3 Internal Model Control – Introduction*

The most common application of MPC is known as Internal Model Control (IMC). The internal model is an explicit part of the controller. Some of the main advantages of the IMC structure are:

- It explicitly takes into account model uncertainty and corrects for this using feedback error(e).
- It allows the designer to trade-off control system performance against control system robustness to process changes and modelling errors.

The basic block diagram depicting an IMC structure is shown in figure 3.2a. Here P is the actual plant, P^ is the plant model (forward model) and Q is the controller (inverse plant model). In figure 3.2b, $Q_1$ and $Q_2$ are identical controllers.

The other symbols shown in the figures are:

y is the plant's measured output

y^ is the predicted output

r is the setpoint signal to the controller (Q)

d is the effect of unknown disturbances on the plant

u is the inputs to the plant/ plant model

u^ is the prediction of the inputs by the Q1 inverse model

e is the feedback error term (calculated as: $y - y^{\wedge}$ )

In figures 3.2a it is important to note that the controller (Q) cannot simply be the inverse of the process (P) or the process model (P^). If the inverse of the process model (P^) had been used then the controller would contain a prediction term or unstable pole [de Vaal, 1999]. It is also important to note is that in most applications using these structures the feedback error term (e) is filtered. This filtering helps to ensure that the controller is physically realisable and stable and helps reduce the impact of fast plant disturbances (d).

For the purposes of this dissertation the following important conclusion are stated regarding IMC controllers [de Vaal, 1999] without delving into too much detail or intricate stability analyses:

- The IMC structure allows independent design of the control block and the filter, which affects the quality of the process response and robustness, respectively. These objectives are hopelessly intertwined in most other design procedures.
- The stability of the closed loop system is not an issue, only the robustness.

The use of an impulse response model is advantageous because a structural model identification is not required and the non-minimal representation adds robustness to the scheme. Furthermore, this model representation is very suitable for a theoretical robustness analysis and thus for tuning of the filter.

The structure shown in figure 3.2a has been developed for use with neural network models and many applications of this type of controller have been published [Hussain, 1999]. This controller and some of the publications dealing with its use with neural network models are discussed in the section 5.2.2.

The structure shown in figure 3.2b, referred to as the two-degrees-of-freedom IMC structure by Garcia et al [1989], is an interesting variation of the general IMC structure, although it is not discussed in detail as part of this dissertation. No examples of applications using this controller structure could be found, although on paper it looks very promising. The most unique feature of this structure is the fact that two identical controllers ($Q_1$ and $Q_2$) are used. The internal controller model ($Q_1$ ) is used to correct the prediction of the controller model ($Q_2$ ), by calculating a feedback error term (e) based on the values (y^) from the internal forward model (P^) and the actual plant outputs (y). This feedback error should enable the controller to be robust even if the controller (inverse) models available are not entirely accurate.

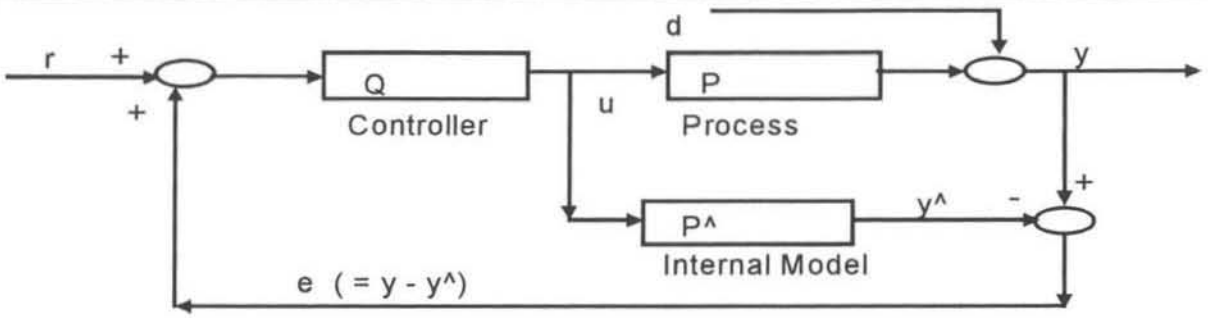


*Figure 3.2a. General Internal Model Control (IMC) structure*

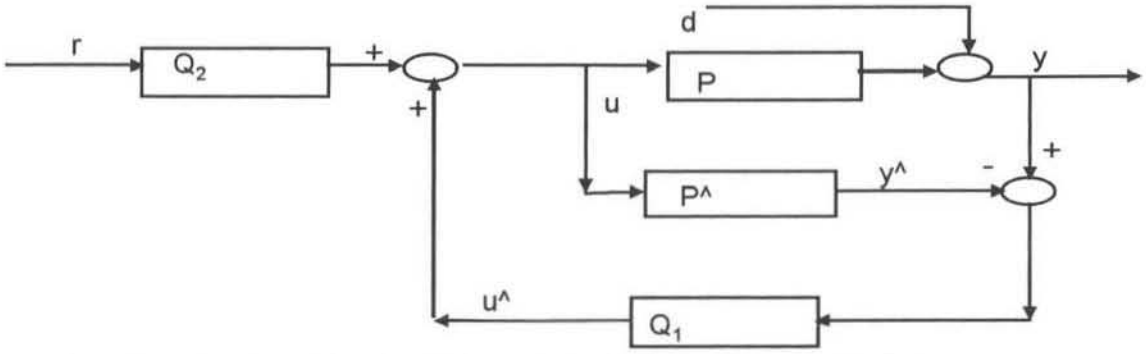


*Figure 3.2b. Two-degrees-of-freedom Internal Model Control (IMC) structure*

*Figure 3.2 General Internal Model Control (IMC) structure*

## 3.4 Modified Internal Model Control

The use of the traditional IMC structure is widely reported and is well known. A unique combination of an IMC structure and neural network models was desired for an advanced control study (shown in figure 3.3). One of the aims in this dissertation was to test an artificial neural network model based controller that had a slightly modified structure to that shown in figure 3.2b. No previous reference to the use of this structure has been found so it has been termed "Inverse Internal Model Control " (IIMC).

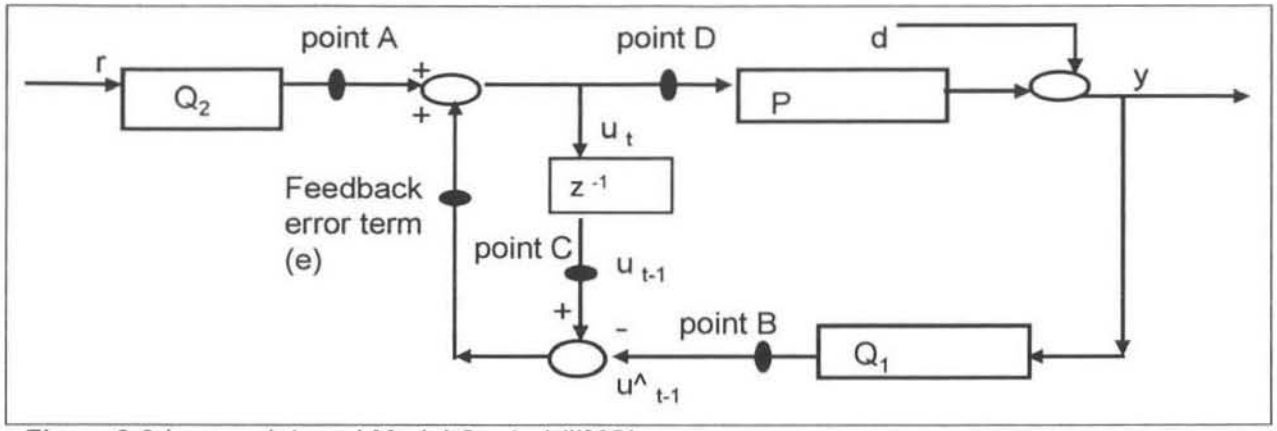


*Figure 3.3 Inverse Internal Model Control (IIMC)*

The main differences of the IIMC controller (figure 3.3) from the traditional IMC structures (figures 3.2a & 3.2b) are:

- The internal model which is normally a forward (P^) plant model has been replaced with an inverse plant model ($Q_1$).
- The feedback error term (e) is used to correct the controllers ($Q_2$ -point A) prediction. The error term (e) is usually filtered.

The error term (e) is the difference between two values of the plant inputs, a predicted value ($u^{\wedge}_{t-1}$ -point B) and an actual value ($u_{t-1}$ -point C). The actual value is an 'old' value ($u_{t-1}$), from the previous sampling time (depending on the model and controller structure used). The inverse model ($Q_1$) predicts the corresponding control action (point B), taking the present plant conditions (y) into account.

The main advantages of using this structure are:

- According to Hussain [1999] nearly 75% of the cost of an advanced control project is spent on model development. Developing one inverse model (Q) instead of two models (one forward (P^) and one inverse model (Q) in the case of normal IMC could result is savings of time, resources and money.
- The possibility of the 'controller model'– 'internal model' mismatch is eliminated since the two models ($Q_1$ and $Q_2$) are identical.
- The feedback error (e) is fed back in to the control algorithm after the controller prediction ($Q_2$), hence this error only offsets the control action, instead of passing through the entire computation.
- Filtering the error term (e) reduces the effect of unusual plant disturbances (d) on the controller.

As is the case with any research there will be disadvantages to experimenting with this new structure:

- No previous work using this structure has been found as a reference. The reference where the idea was taken from (Garcia et al [1989]) does not cover any practical aspects of implementing this controller.
- The stability of the controller can not be guaranteed. Garcia et al [1989] mentions aspects of the controller stability but it is based mainly on the structure shown in figure 3.2b. As is mentioned, this is slightly differently to the IIMC controller. However, as mentioned earlier (de Vaal, 1999), the closed loop stability is not a major issue with IMC controllers.

The neural network model based form of this controller is developed and discussed in section 5.2.3.

# CHAPTER 4

# 4. ARTIFICIAL NEURAL NETWORKS

## *4.1 Feedforward Artificial Neural Networks*

### 4.1.1 Introduction to Artificial Neural Networks

Artificial neural networks are classified as black box mathematical non-linear regression tools. One of their main advantages is their ability to represent non-linear processes. They can be used to learn and identify correlative patterns between sets of input data and corresponding outputs. Once trained these neural networks can be used to predict outputs from new input sets. The training of these neural networks is a form of non-linear regression and has developed into a tried and trusted technique.

As mentioned previously it has been shown [Hornik & Stinchcombe; 1989; Cybenko, 1989] that any continuous function can be approximated to an arbitrary degree of exactness by an appropriate neural network model.

### 4.1.2 Development of Artificial Neural Networks

The brain has long been heralded as the one of the most complex organs known to man. It's ability to learn, comprehend and store information and then make logical decisions based on this sometimes incomplete and/or incorrect information is amazing. These decisions, some conscious and some unconscious, are the result of the brain processing the relevant information in a situation and making suggestions based on previous experiences and perhaps some interpolation or extrapolation. Artificial neural networks work in a similar manner.

Researchers studying the working of the human brain have established that the brain consists of millions of neurons, the basic building block or fundamental biological unit of the brain. Each neuron has of the order of ten thousand connections coming in to it from other neurons. Electrical signals are transmitted from neurons along their axon, which branches off and meets (without making physical contact) other neurons at points called synapses, or axon-dendrite connection points, on the dendrites (input branches/roots) of those other neurons.

The electrical impulses received at the ends of the axon triggers chemical transmissions from the axon to the dendrites, across synaptic gaps. The impulse, or signal, is processed by a chemical reaction in the cell body within the neuron, known as the synaptic receiver. This

reaction alters the signal strength. The processing that occurs within the cell body does not change as the learning process takes place. This synaptic strength is what is altered when the brain learns, i.e. right now in your brain if you never knew this fact before. These synaptic strengths are constantly changing as the brain learns more. The physical effect of these chemical contents is that they determine the weight of the output signal that is transmitted from the neuron along the axon to other dendrites and hence other neurons.

The development of artificial neural networks began many years ago in the 1940's as people recognised the possibilities of mimicking the way the way that the brain worked. However it wasn't until recently in the 1980's that the study of neural networks and especially its many applications experienced such a sudden growth. The reason for this was the drastic increase in computing power that meant that many desktop PC's could perform neural network calculations easily.

## *4.2 Basics Of Artificial Neural Network Theory*

### 4.2.1 The Node

A neural network has a similar structure to the brain. The analogous unit to the brain's neuron is a node or PE (processing element). Each of the nodes has distinct input and output connections, as well as a transfer function for the output of the node, except in the input layer, which receives external signals. A node is shown in figure 4.1. All the individual signals entering into a node are summed (the term 'net') as in equation 4.1. Every signal into a node is the result of the output signal from another node ($x_i$) multiplied by a unique weighting factor ($w_i$). These weighting factors are the equivalent to the synaptic weightings in the brain and are what give the neural network the many unique computational properties that it has.

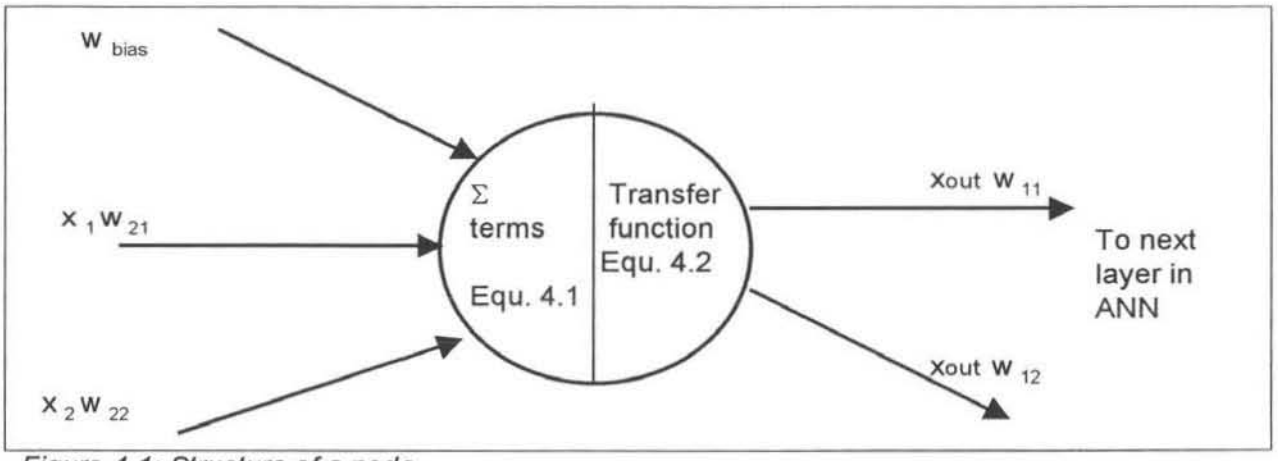


*Figure 4.1: Structure of a node*

$$net = \sum_{j=1}^{n} x_j w_j$$ Equation 4.1

$$x_{out} = F(net_i)$$ Equation 4.2

### 4.2.2 The transfer function

The summed value of all the inputs to a node (net) is then modified by the transfer function (F) to obtain the output value, which is passed along to the inputs to other nodes ($x_{out}$) , as in equation 4.2. The transfer function can be any continuous function with a bounded derivative. These functions are generally non-linear. Common functions used are shown in equations 4.3 and 4.4. The sigmoidal function (equation 4.3) is very commonly used. The outputs of this function are always between 0 and 1, which can be seen in figure 4.2.

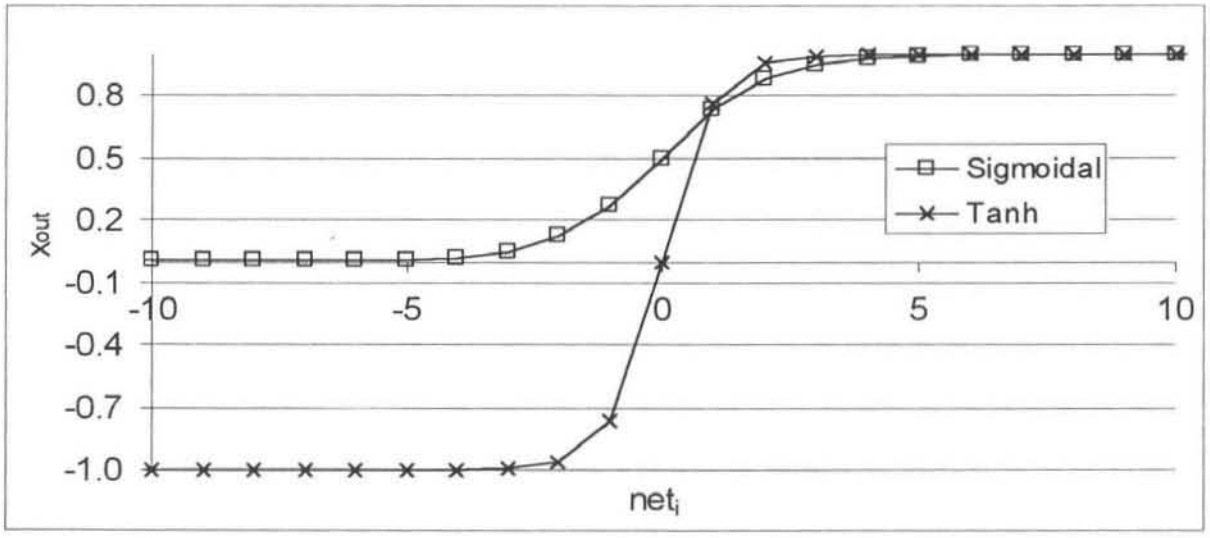


*Figure 4.2: Commonly used transfer functions*

$$x_{out} = \frac{1}{1 + e^{-net_i}}$$ Equation 4.3

The hyperbolic tangent function (equation 4.4) bias is also used, although not as widely as the sigmoidal function. The effect of passing a signal through these functions is shown in figure 4.2.

$$x_{out} = \tanh(net_i)$$

*Equation 4.4*

### 4.2.3 The Artificial Neural Network Structure

When nodes are combined in a structure it becomes known as a neural network. The usual topology of a neural network is that it has several distinct layers. Nodes are interconnected in an organised pattern. The flow of information is nearly always from the input layer of the network to the output layer.

A common type of network is the mutil-layer perceptron or feedforward neural network as shown in figure 4.3. This is also one of the simplest examples of a neural network to understand. As is clearly seen there are 3 layers in this neural network, the input, hidden and output layers. The flow of information is in the direction of the arrow, from input to output. In this neural network there are 3 input nodes, 3 hidden nodes in the hidden layer and 2 output nodes. In addition the input and hidden layers have an additional node known as a bias. The most common number of hidden layers a neural network can have is 1 or 2.

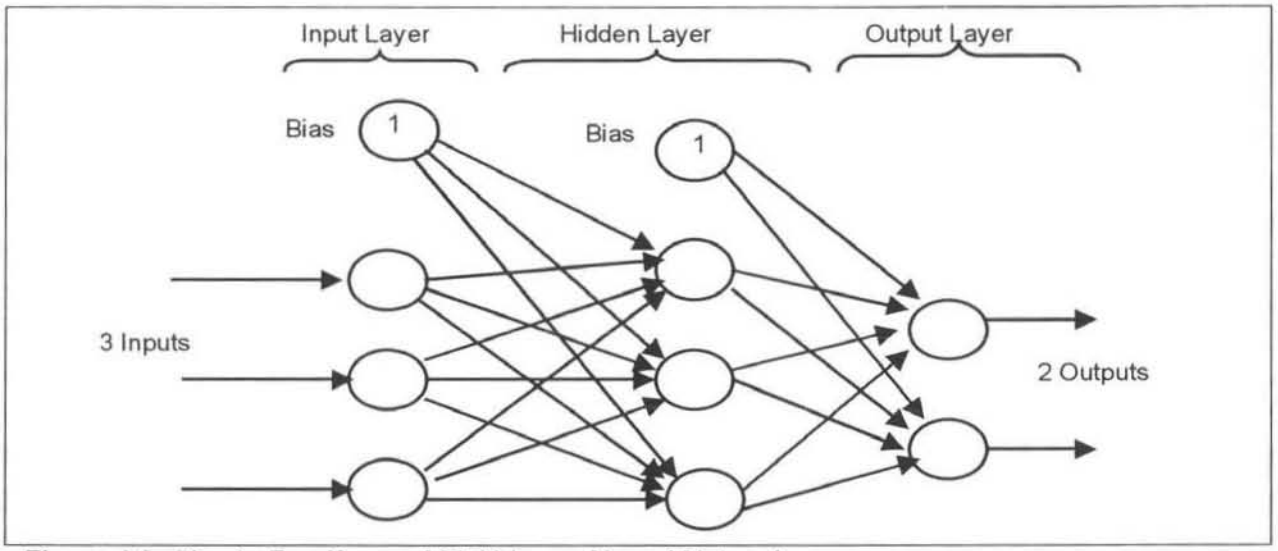


*Figure 4.3: Simple Feedforward Multi-Layer Neural Network*

$$y = m * x + c$$

*Equation 4.5*

This bias has the same function as the 'c' term in the equation 4.5. It adds an extra degree of freedom to the neural network. The value from a node bias is equal to the weight associated with that connection ($w_{bias}$).

The value on the input nodes is equal to the actual input ($x_i$) multiplied by the respective weight ($w_{ij}$). The summed value is changed by the transformation function. A node in the hidden layer has connections from each node in the previous layer as well as the bias. Each of these inputs is multiplied by its respective weighting factor and summed. Again the transformation function is applied to this summed value. The output layer nodes are connected to every hidden layer node including the bias. The same procedure happens again until the individual outputs from each output node are obtained.

As in the case of the human brain, a neural network needs to be trained before it can be used. As the saying goes "good judgement comes from experience, experience comes from bad judgement", so the neural network needs 'experience' to learn to make good 'judgements'. This experience comes from training it with rich, characteristic data of the system under consideration. This will be discussed in more detail in section 4.3.

### 4.2.4 Scaling of input/output data

An important issue in the neural network calculation is the scaling of input/output data. This scaling of data is important in both the training and application phase of the neural network. The range of outputs that the common transformation functions can produce is usually limited to between –1 and 1 or 0 and 1. Referring to the sigmoidal curve shown on figure 4.2 it can be seen that if the weighted sum of the inputs was less than –3 or greater than 3 would effectively mean that function was saturated. This is a normal part of training and a useful feature of sigmoidal and hyperbolic functions is that they can be driven to saturation [Cybenko, 1989].

Another important reason for the scaling of the input data is so as to place equal importance on all of the inputs during the training phase. This is so that during training the weights are kept within a similar size-range so that they do not dominate or become ignored. Likewise, the output of the neural network can only be a number within the range of the transfer function, usually between –1 and 1. Hence this output has to be re-scaled back to the original scale of the output. The most common type of scaling is done according the linear scaling function in equation 4.6. This equation can be rearranged to re-scale the output values.

$$x_{scaled} = Min\_\lim + (Max\_\lim - Min\_\lim) * \frac{x_{raw} - x_{\min}}{x_{\max} - x_{\min}} \qquad \textit{Equation 4.6}$$

In some cases where the range of data may be of orders of magnitudes apart then a logarithmic scaling function may be applied, however this was not needed on the data considered in this research.

### 4.2.5 Weighting Factors

The weights obtained during training are what make the neural network technique so unique. The number of weights is determined by the number of hidden nodes, hidden layers, inputs nodes and outputs nodes used. The more weights that are used, the more the model will simply learn the exact behaviour of the training set and may have difficulty resolving new sets of input data. What is usually desired is a model trained with fewer weights that learns to generalise the systems behaviour and interpolates between new input data. This is discussed more in section 4.3.3.

## *4.3 Training and Testing Neural Network Models (software and algorithms)*

### 4.3.1 Training Artificial Neural Network Models

Training of a neural network refers to the process of adapting the weights that form a neural network model. This training can be done by several different means and is performed until some criterion is met, usually until the error between the predicted and actual values in the training set is minimised. This error may be described by the term 'sum of the squared error' (SSE) [Narotam 1999].

$$SSE = \sum_{pt=1}^{P_T} \sum_{i=1}^{n} \frac{1}{2}(Y_i(p_t) - y_i(p_t))^2 \qquad \textit{Equation 4.7}$$

where

$P_T$ = total number of examples presented to the neural network

$p_t$ = index of the example

n = number of network outputs

$Y_i$ = neural network prediction of $i^{th}$ output node

$y_i$ = actual value of $i^{th}$ output node

## *4.3 Backpropagation*

The most popular method of training the weights for a neural network is known as 'backpropagation'. This method of training is used for all of the neural network models developed in the work that is presented in this dissertation. An excellent description of the actual algorithm, equations and the derivation of the backpropagation algorithm is given by Narotam [1999]. Some of the main advantages of the backpropagation training algorithm are:

- It is a relatively simple algorithm to understand and implement.
- A large amount of research has been done on the topic and there are many excellent references on the subject.
- Countless applications of this technique have provided reliable results.

Disadvantages

- As with any gradient-based search algorithm the possibility of becoming trapped in a local minimum exists. Basically this means that it is possible that the algorithm may yield sub-optimal results in some cases.

Following the discovery of the backpropagation technique research into the training field has led to other techniques being developed. Some of these methods offered slightly improved algorithms. However, today the backpropagation technique is still more than adequate. Some of the other techniques are:

- Enhanced Backpropagation
- Backpropagation with Weight Decay
- Quick-Prop

A very basic outline of a generic (algorithm independent) batch training procedure is shown in figure 4.4. An initial set of weights is given to the training software, and this can be a randomly generated set or a previous weights set. During the iterative process of training the weights, 1 complete cycle of presenting the new estimate of the set of weights to the training and test data sets, and updating this weight set is known as an 'epoch'.

### 4.3.3 Overtraining

There is a danger that the neural network model will not learn the 'general' behaviour of the system. It may learn to mimic only the actual data set that is used for training. Instead of creating a model of the process it learns to associate actual outputs with specific set of input

data. When the model is presented with a new data set it incorrectly tries to apply the same associations from the training set and the predictions are inaccurate.

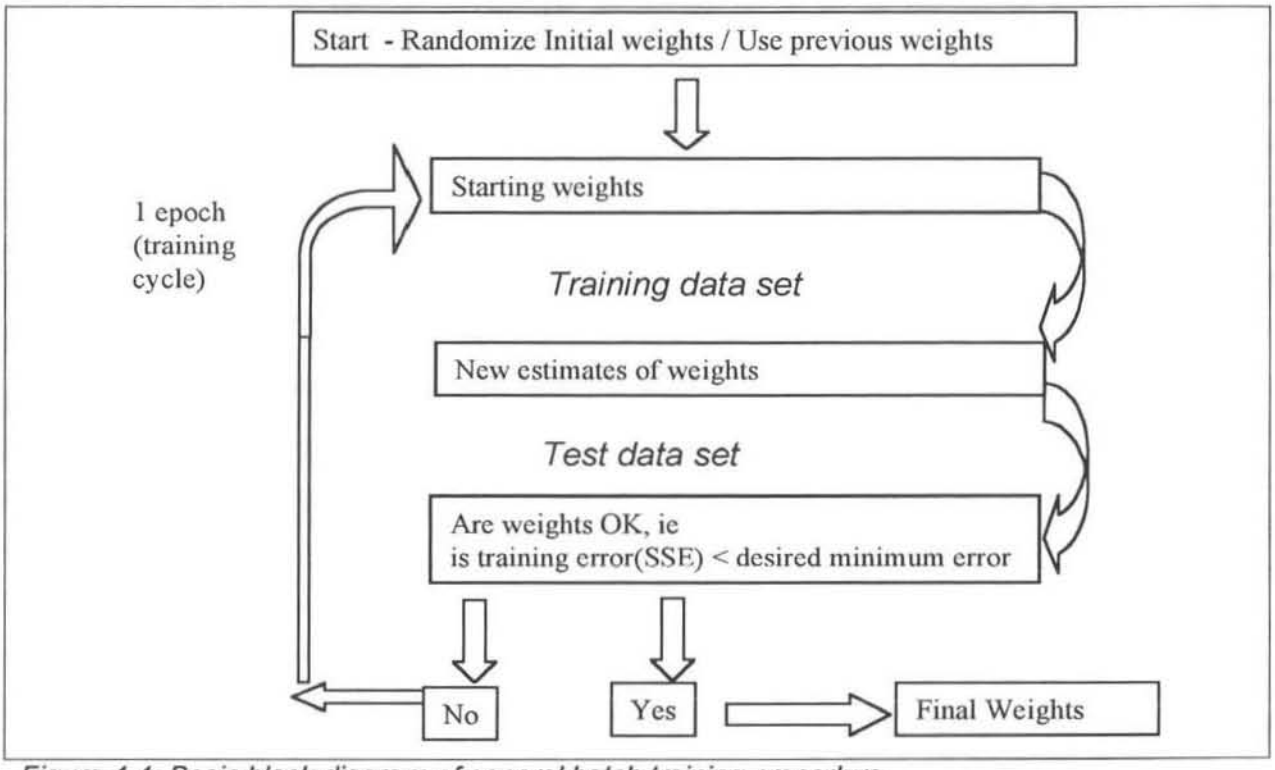


*Figure 4.4: Basic block diagram of general batch training procedure*

This may be a problem when the data set presented to the neural network model for training does not have many records or there is very little variation in the variables over the training period. Another case where this may occur is when too many hidden nodes are used in the neural network model. The model tries to learn from the training set but too many weights are available to approximate the process and the 'fit' of the model ends up being to tight.

Some commercial software packages have built in functions to avoid this overtraining from occurring. These include giving sensible predictions of the number of hidden nodes to use and the number of training iterations to perform. By monitoring the model error (SSE) during training it is possible to detect if overtraining occurs.

The software used in this dissertation did not have these built in checks so the models had to be chosen sensibly and the number of training iterations restricted to reasonable figures.

### 4.3.4 Training Software

The focus of the work in this dissertation was not on developing the neural network training software but rather on developing innovative and useful models. Several software packages were available on which to develop the neural network models.

Chouai [1999] was an academic visitor to the School of Chemical Engineering, University of Natal in June/July '99, and had developed neural network training software that was available for use. The training algorithm in this software worked very well and most of the models in this dissertation are a product of that software.

Another option that was used to a lesser extent was the software package BASIC MODELGEN 1.06 [1998], which was developed by Crusader Systems. This software allows for the training of neural network models along with many other useful pre and post data analysis options. This software was not used to the same extent since the provided weights did not work well in our model, external to the package. The scaling equations seemed to differ slightly from what was expected. Bearing in mind our requirement to export the model, the work was largely performed using the software of Chouai [1999].

Both these software packages presented the final weights in the form of a text file. Other software programs written to test the weights as described in section 4.5 could use this text file as an input.

## *4.4 Choosing the Structure for the Neural Network Model*

As mentioned already it has been shown [Hornik et al, 1989; Cybenko, 1989] that any continuous function can be approximated to an arbitrary degree of exactness by an appropriate neural network model. This is very reassuring however it doesn't give much advice on choosing the model, which is one of the key decisions to make when using neural networks.

For the work in this dissertation only feedforward artificial neural networks with 1 hidden layer as shown in figure 4.3 are considered for model selection. This narrows the important choices down to a few:

1. Which variables to use as outputs of the model?
2. Which variables to use as inputs to the model?
3. Of the variables chosen as inputs, how many past values to include to obtain dynamic process information?
4. The number of hidden nodes to use in the neural network hidden layer?

The answer to the first question is fairly straightforward. What does the model hope to achieve? If the model is going to predict a level in a tank then this is obviously one of the outputs. In some cases other outputs could be unobservable values, such as the MI and density on the polymer reactor.

For questions 2 and 3 above, one would initiatively know that there is an exact number of variables and past data history in the model that will give the best approximation of the desired function. Limiting the size of models is an issue that is becoming less important these days as the computational power of computers increases. This is discussed in sections 4.4.1 to 4.4.4.

For question 4, choosing the number of hidden nodes is a trial and error process. Some software packages will give an initial estimate for this number given the number of model inputs and outputs. This is an estimate only and the best number is found by optimising the model error (SSE) with respect to this number. However it is possible to choose too many hidden nodes and this can lead to overtraining, see section 4.3.2. This number has to be chosen sensibly taking the above information into account.

For neural network control to be successfully implemented in any form it is essential to have an accurate representation of the system under consideration. The model used for the system has to be trained on data that reveals the full range of expected operating conditions on the plant. This is so that the controller is able to deal with variations of the plant outputs.

### 4.4.1 Time-Invariant Dynamic Forward Models

As mentioned in section 3.1.1 a time-invariant dynamic forward model of a process will attempt to replicate the behaviour of that process, by predicting it's outputs a given number of time steps into the future. The model is dynamic since it takes into account the process behaviour over time.

Refer to figure 4.5 which shows the level in a tank and the input flow to the tank over a time horizon. The plant input is the flow rate and the plant output is the tank level. (The liquid drains from the tank from a side valve, in other words the system is not an integrating system. It will adjust to a new steady state. )

To predict the plant outputs 1 time interval ahead of the present time (T+1*t), the inputs to the model should probably include:

- The present (time T) and a few past (T- 1,2,3..*t) plant outputs
- The present (time T) and a few past (T- 1,2,3..*t) plant inputs (control actions)
- Some of the current (time T) conditions in the process (i.e. temperature, density of the fluid in the tank etc).

The predicted value would then be

- The future (T+ t) plant output

The current operating conditions in the tank can be included to give the model more information about its present condition. Information about the past trajectory of these variables can also be given to increase the model accuracy. However one has to remember that it is a trade-off between perhaps a slight increase in model performance and creating too bulky a model.

The number of past points of the plant inputs (u) and outputs (y) is a choice that the designer must make carefully. Again the model should not become too bulky, however if not enough information is included then the model will not be a true dynamic model. The strategy adopted in this research was to model the process around a step response in the plant outputs. Looking at figure 4.5 again it can be seen that the dead time from the flow step change to the level response is about 1 ½ time units. The level response is defined in about 2 time units.

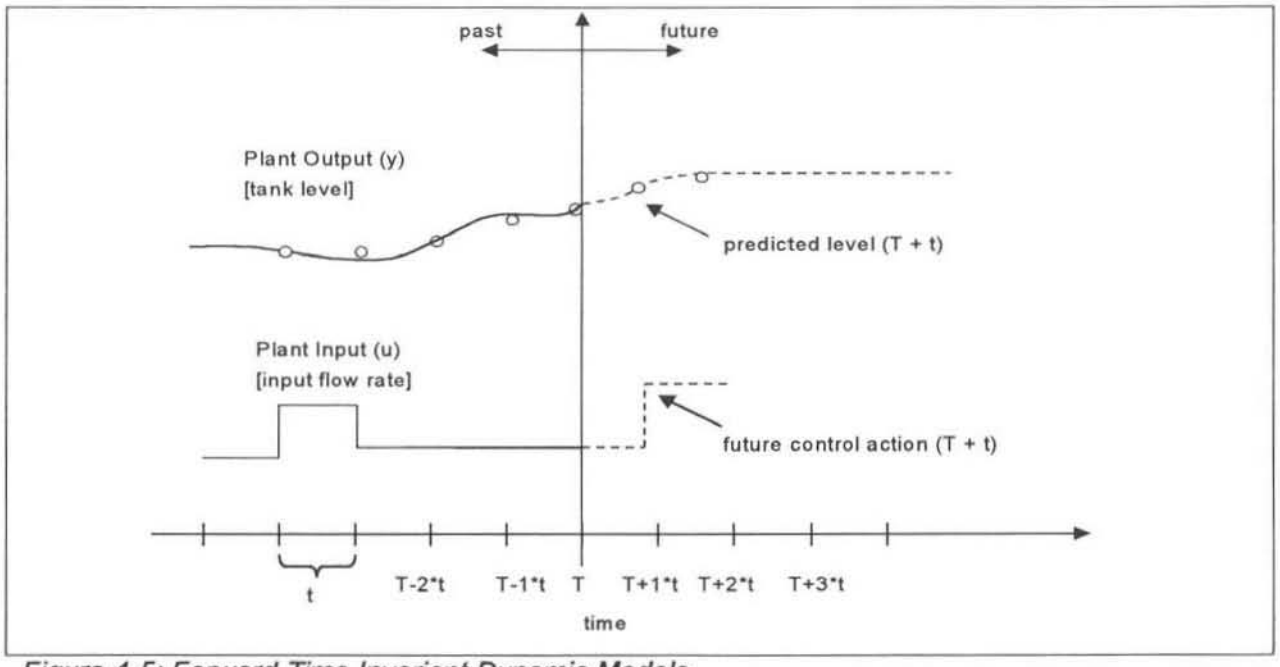


*Figure 4.5: Forward Time Invariant Dynamic Models*

Therefore the number of past inputs and outputs chosen to include in the model would be 3 (i.e. up until time T – 3*t). This value of 3 was obtained by rounding off the sum of the two individual time lags. This information is sufficient to allow the model to predict the effect that any changes in the plant input will have on the predicted output.

### 4.4.2 Time-Invariant Algebraic Forward Models

A time-invariant algebraic model does not take the process variation over time into account. Instead it creates what can be described as a 'look-up' table of average plant conditions. This is illustrated in figure 4.6. By using the current conditions in the tank (such as temperature etc) it gives the model more information about the position of the current plant output (level). This

effectively means that there could be different relationships between the plant inputs and outputs under varying operating conditions. This can be useful in aiding the model when these operating conditions have a large effect on the plant outputs, as in the case of the polymer plant reactor.

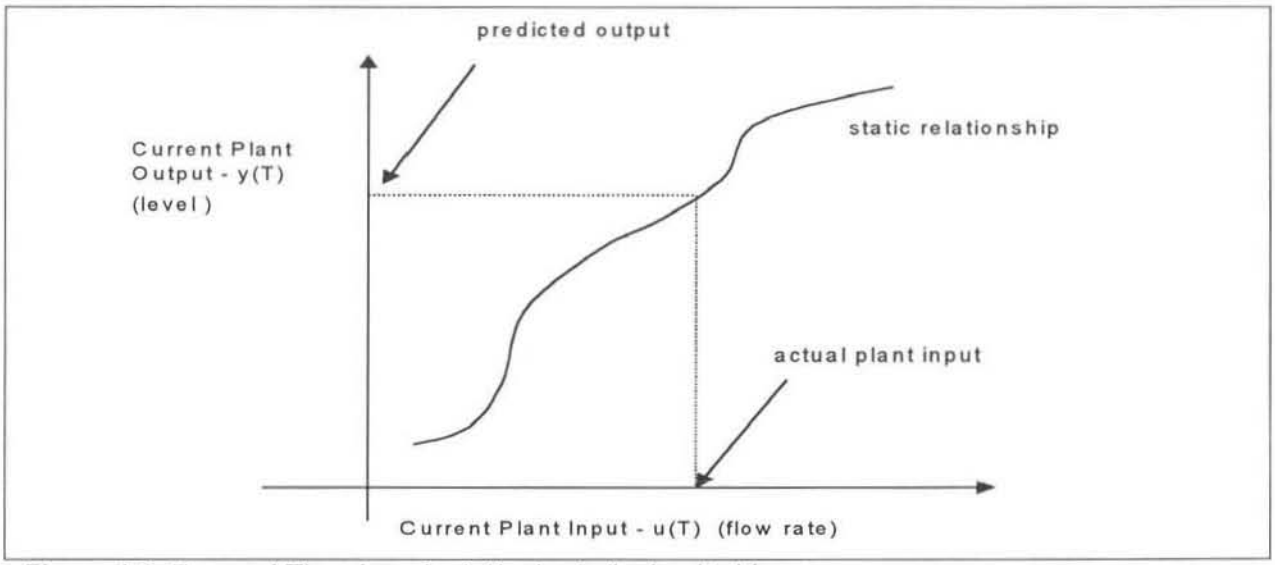


*Figure 4.6: Forward Time-Invariant Algebraic 'lookup' table*

The downside to these time-invariant algebraic models, representing as they do steady-state behaviour, is that they must be trained on steady data, or data which compensates for dynamic variations. The trick to effectively using time-invariant algebraic models is to train them intelligently. In this example it is known the current plant input (flow rate) could have an effect on the next three outputs (levels). Hence when training the model instead of using the current level and flow rate on each training set rather use an average of the four levels (during off-line training the future values are known).

$$level_{average} = average\ [\ level(T) + level(T + t) + level(T + 2t) + level(T+ 3t)\ ]$$

*Equation 4.8*

This average level is used with the current flow rate (time T) during training so that the combined dead time and process response can be incorporated into the neural network model. When new data is processed the neural network model will predict the plant outputs taking these lag times into account.

### 4.4.3 Time-Invariant Dynamic Inverse Models

An inverse model attempts to predict the inputs to the process that will result in a certain plant outputs being achieved (see section 3.1.1). Continuing the example from above the neural network model would attempt to predict the flow rate necessary to achieve a certain tank level (i.e. a setpoint).

The problem here is slightly different from the forward time-invariant dynamic model (section 4.4.1). The future input (control action) is less dependent on the past inputs (control actions) than the outputs are on their past trajectories. Though they can impact on future outputs, past inputs become of reducing importance.

The most important input for these neural network models is the desired setpoint of the plant output. This value gives the model an indication of what the control action should be. Obviously the most desirable situation is a future trajectory of setpoints. However, the training in this situation must also resolve the contributions of future inputs. In this study, the output was considered only one step ahead of the present.

As was the case previously some intelligent training techniques are required here. It is better to consider 1 average setpoint over the relevant interval of interest (3*t in this example) than just the future setpoint (T + t) during the model training. Continuing with this example the average level setpoint for training would be calculated as follows.

$$level_{average} = average\ [\ level(T + t) + level(T + 2t) + level(T + 3t) + level(T+ 4t)\ ]$$

*Equation 4.9*

This average level would be used with the future control action (flow rate (T + t) during training so that the model would learn to associate that input (average level) with that model output (flow rate (T + t)). With new data the neural network model would treat the level setpoint as an average (even though it may be not be) and predict the corresponding control action that would give that average setpoint. In this way the model is taking the time lags in the system into account and correcting for them. Obviously if there was an immediate output response to a plant input then the model would not need to trained to take the dead time and response time into account.

Since this is a dynamic model some values of previous control actions can be taken into account. As an example of how many values to use in the model, consider the current example. The control actions from two intervals ago could have an effect on the earliest level considered (level (T+ t)).

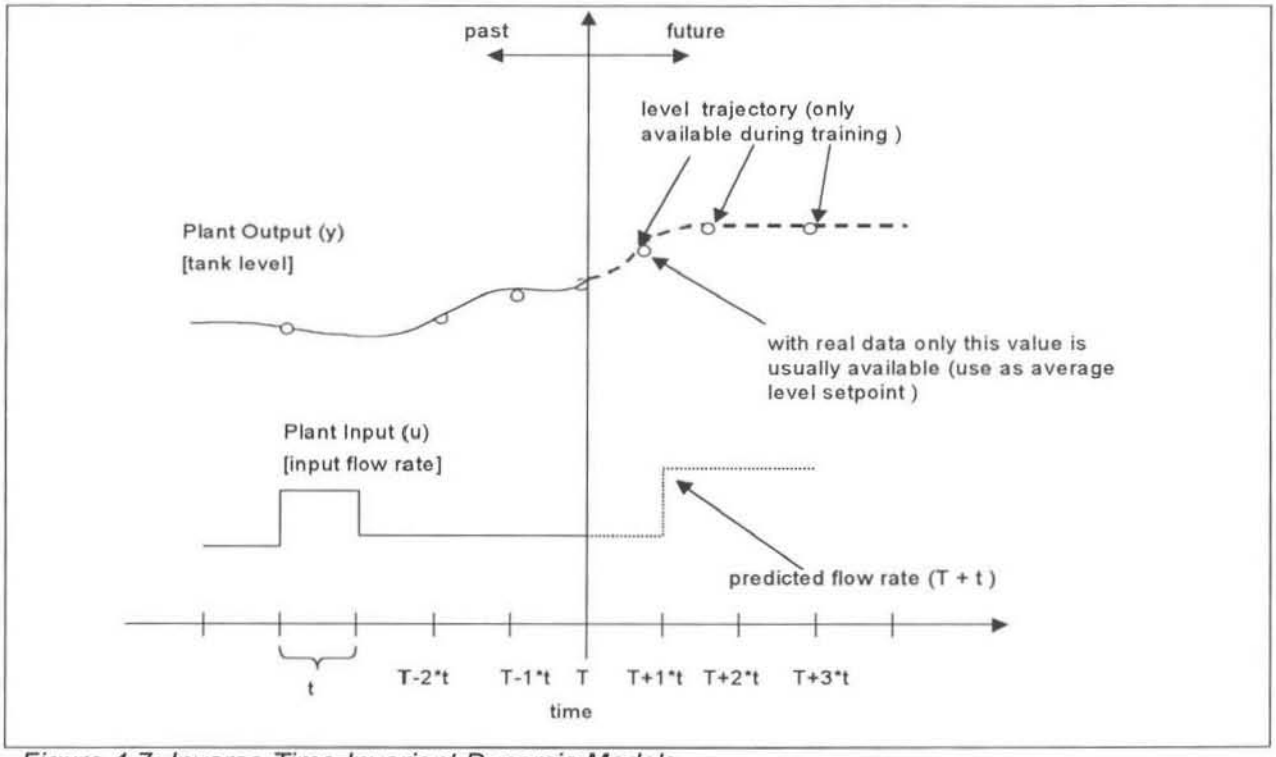


*Figure 4.7: Inverse Time-Invariant Dynamic Models*

Taking all of the above into account the model inputs would probably look something like this:

- The future plant output (i.e. average level setpoint)
- The present (T) and a few past (T- 1,2..*t) plant inputs (control actions)
- Some of the current (T) plant conditions (i.e. Temperature etc).

The predicted value would then be

- The future (T+ t) plant input (control action)

As was the case in section 4.4.1 the current process conditions may aid the model in distinguishing certain operating regions. If a past trajectory of these variables is given then the designer must remember the trade-off between perhaps a slight increase in model performance and creating too bulky a model.

### 4.4.4 Time-Invariant Algebraic Inverse Model

The inverse time-invariant algebraic model is basically the opposite of the forward time-invariant algebraic model (section 4.4.2). Figure 4.8 shows the tank level example. This 'lookup' table stores the relationship between the desired plant output and the plant input that would achieve it. Again by giving the model more information about other variables (such as temperature etc) in the system it will help it to distinguish different operating conditions. This is useful when these operating conditions have a large effect on the plant outputs, as in the case of the polymer reactor.

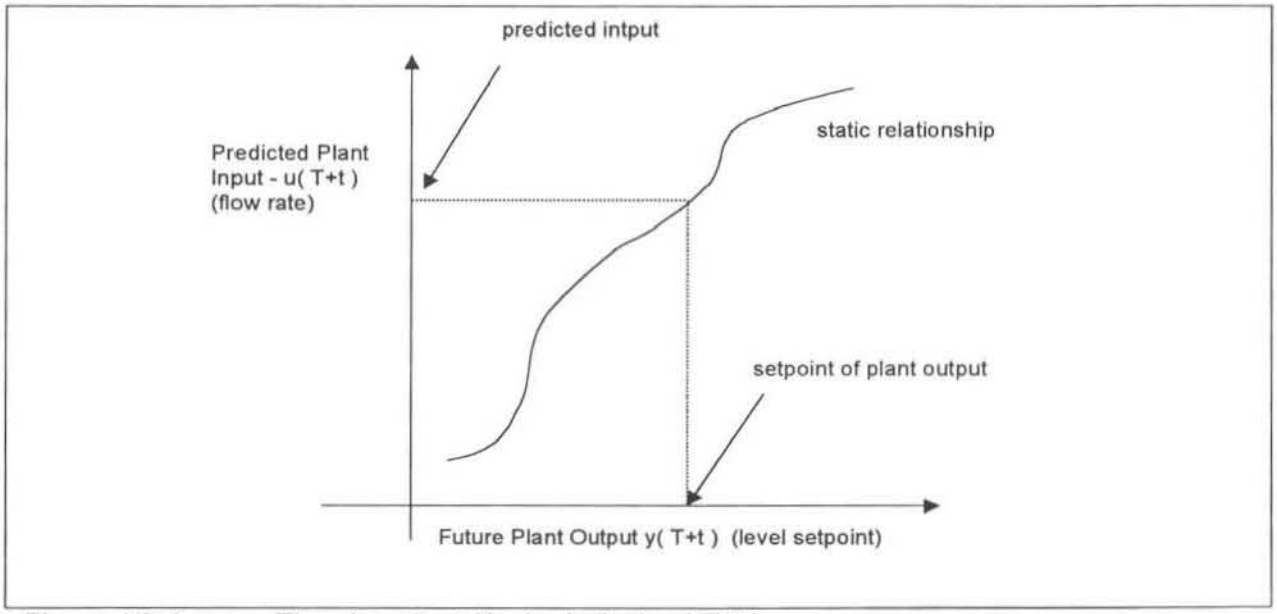


*Figure 4.8: Inverse Time-Invariant Algebraic 'lookup' Table*

As is the case with the time-invariant dynamic inverse model, the presentation of training data to the neural network is very important. The model needs to be trained with enough information about the future plant outputs (level) to create an association with the control action.

If the plant setpoint for the outputs (level) is only available for 1 time interval ahead then it is better to train the neural network on the average of these future levels. As in the previous section for the tanks example the average level is calculated from equation 4.9. This ensures that the model will account for the time lags in the system. This is only necessary if there is a significant dead time or response time in the system.

## *4.5 Testing Neural Network Models*

Once a model of a system has been trained and the weights obtained, this model needs to be tested to see whether it is suitable to be used or not. This testing will validate how well the model has learnt the system's behaviour. The testing should ideally check the model's response to input disturbances, the error between predicted and actual outputs and the long-term model prediction of outputs. There are several available methods for model testing. These programs have been written using the software program MATLAB and are included in the Appendices I to M.

### 4.5.1 Neural Network 1-Step Predictions

The simplest test is just to check the model prediction versus the actual outputs using unseen data, as shown in figure 4.9. Unseen data simply refers to data other than that with which the model was trained but obviously still within a similar operating range. The reason for using unseen data is to check whether the model has learnt the behaviour of the system or simply learnt the actual outputs of the data set due to overtraining or too many internal nodes being

used. In the latter of the two cases the model will perform poorly since it will not be able to accurately interpolate in the range of the new data.

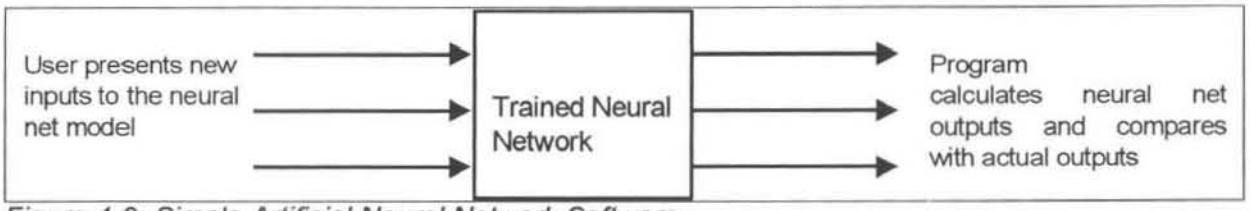


*Figure.4.9: Simple Artificial Neural Network Software*

If an accurate neural network model were tested the predictions of the outputs would closely match the actual outputs. For a poorly trained or an overtrained model the error between the actual and predicted values would be much larger and the predictions may follow an erratic path or may be heavily offset from the actual values.

One of the easiest ways of quantifying this error is by calculating the standard deviation (or RMS – Root Mean Squared error) between these two sets of values.

$$SD = \sqrt{\frac{\sum_{i=1}^{N} (y_{pred_i} - y_{obs_i})2}{N-1}} \qquad \textit{Equation 4.10}$$

where SD = the standard deviation of the error
$y_{pred}$ = predicted value of ANN model
$y_{obs}$ = the actual value from the training/testing set
N = number of records tested

Another method would be to make use of the squared correlation coefficient (equation 4.11), which can be assessed statistically by means of formal hypothesis testing. (This criterion was not considered for use in the work in this dissertation.)

$$R^2 = 1 - \sum_{i=1}^{N} (y_{pred_i} - y_{obs_i})^2 / \sum_{i=1}^{N} (y_{avg_i} - y_{obs_i})^2 \qquad \textit{Equation 4.11}$$

where R = squared correlation coefficient
$y_{avg}$ = the average value of the observed output data

In the case of a time-invariant dynamic neural network model which predicts a future output given the present and past history of the system inputs and outputs, it is very likely that the neural network prediction will be not be far off the actual value. This will be true especially when the time step between the predictions is not very large and the system does not move very quickly (i.e. slow dynamics). In this case it is more difficult to distinguish between good and poor models by this method.

For these 1-step models another method of evaluation would have been taking the autocorrelation of the outputs into account. This can be done by statistically comparing the correlation between the output of the model one time step ahead and the actual output ( $y_{obs\ i}$ vs $y_{pred\ i}$), with the correlation between the actual output one time step ahead and the current actual output of the model ( $y_{obs\ i}$ vs $y_{obs\ i-1}$ ). This method was not considered in this dissertation.

Other methods of checking model validity that are offered in the Basic ModelGen Package are, but that were not used in this work are:

- Model Residue
- Fast Fourier Transform
- Sensitivity Analysis

### 4.5.2 Artificial Neural Network Multi-Step Predictions – "Chaining"

Probably the best test of a time-invariant dynamic neural network model is to check the multi-step output predictions versus the actual values, in other words, to chain the neural network. The procedure of 'chaining' or 'iterating' a neural network is discussed by Saint-Donat, Bhat, McAvoy [1991] and Narotam [1999].

The purpose of the chaining algorithm is to see how well the model can replicate the plant behaviour over a certain prediction horizon, given only the new inputs to the system on each time step. The model prediction and the actual values over the time horizon can be plotted and compared. The amount of drift that occurs in the neural network predictions gives an indication of the accuracy of the model. For the same reasons as presented in the previous section it is better to present completely new input to the neural network models.

Chaining of a model is only useful when considering a forward model of a system. It is meaningless to chain an inverse model of a system, since it predicts process inputs. Effectively the model is predicting backwards in time, which is of little value in model predictive control. It can be used in testing inverse models, however it is not covered in the work in this dissertation.

Figure 4.10 shows the flow of information in the chaining algorithm. For every iteration the neural network model predicts the outputs from the given set of inputs. The input and output terms can be vectors or single values as the concept is the same for SISO or MIMO systems. This diagram shows the input and output terms being shifted back in time. For example the model inputs 'input(t)' and 'output(t)' terms in the first iteration become 'input(t-1)' and 'output(t-1)' in the next iteration. Similarly the prediction term 'output(t+1)' from the first iteration becomes 'output(t)' in the next iteration. This iteration is repeated N times until it is terminated.

The only new information entering the algorithm on each step is the current input term 'input(t)'. The model has no reference to the actual system outputs. All the 'output' terms used in the neural network models are model predictions. This is what makes this algorithm excellent for checking the model behaviour. If the model is able to follow a system in an unseen data set, then the model must be a good representation of the system. The standard deviation between the actual and predicted outputs provides a useful measure for quantifying the model accuracy.

This algorithm as shown in figure 4.10 was altered slightly for the examples of chaining presented in this work. The output prediction from the first iteration is 'attached' to the first actual output value, by adding the error between the two on to the artificial neural network model prediction. This error correction is held constant and added to all further iterations for the rest of the prediction horizon. This is done so that the model has a reference over the different prediction horizon and the accuracy can be evaluated relative to some starting point. At the end of the prediction horizon this error is recalculated and the new error value used to reattach the predicted level to the actual value. Chaining is also useful if the model is being considered for use in a controller where it is necessary to have a prediction of the outputs for a number of future time steps.

The programs written to simulate this algorithm are shown in Appendices I to M.

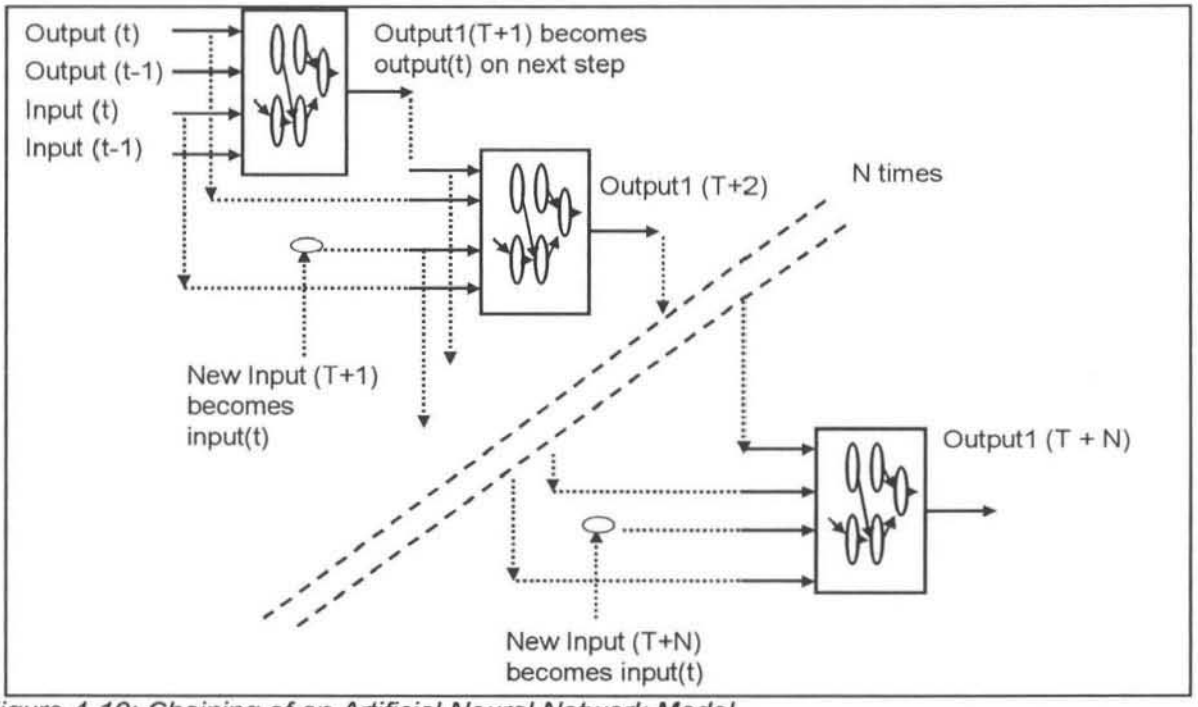


*Figure 4.10: Chaining of an Artificial Neural Network Model*

### 4.5.3 Artificial Neural Network Model Sensitivity Analysis

The purpose of a sensitivity curve is to establish how a model will respond to a disturbance to the inputs of that model. Obviously it is desired that the model will produce the same response as the real system does when subjected to the same disturbance. Studying these sensitivity curves is most probably the best method of determining whether an inverse model of a system is suitable to act as a controller or not. The MODELGEN PACKAGE performs a similar test however for the work in this dissertation all the code was written to suit the needs of this work.

These curves are calculated in the same way for both time-invariant algebraic and time-invariant dynamic models. Only the model input of interest is perturbed. In the case of the inverse controller models (discussed in detail in section 7.2) only the setpoint input to the model is moved. It is the model response to this new setpoint value that is of interest.

In figure 4.11 below an example of a sensitivity curve is shown. This example is of an inverse process model taken from the interacting tanks case study (section 5.2). The tanks system has 2 inputs (valve positions) and 2 outputs (tank levels). The inputs to the model that are perturbed are the future level setpoints. The outputs of the model are the suggested valve settings and it can be seen how they respond to the input disturbance.

These curves are based upon an average operating condition (0% : 0% co-ordinate). This condition is determined by visually examining the plant data. The individual inputs, level 1 setpoint in this case, are varied from between 85% to 115% of their normal operating condition and the output responses are plotted. The output responses are also based on a percentage change from their average operating condition. The curves are calculated in this way so that it is possible to see the direction and size of the model's response.

For example if the setpoint of level 1 was increased by 10% from the average operating condition the model would suggest closing valve 1 by 30% and valve 2 by 8%. From past experience with this system it is known that these valve changes will move level 1 in the correct direction. The magnitude of the respective changes also seems to be reasonable.

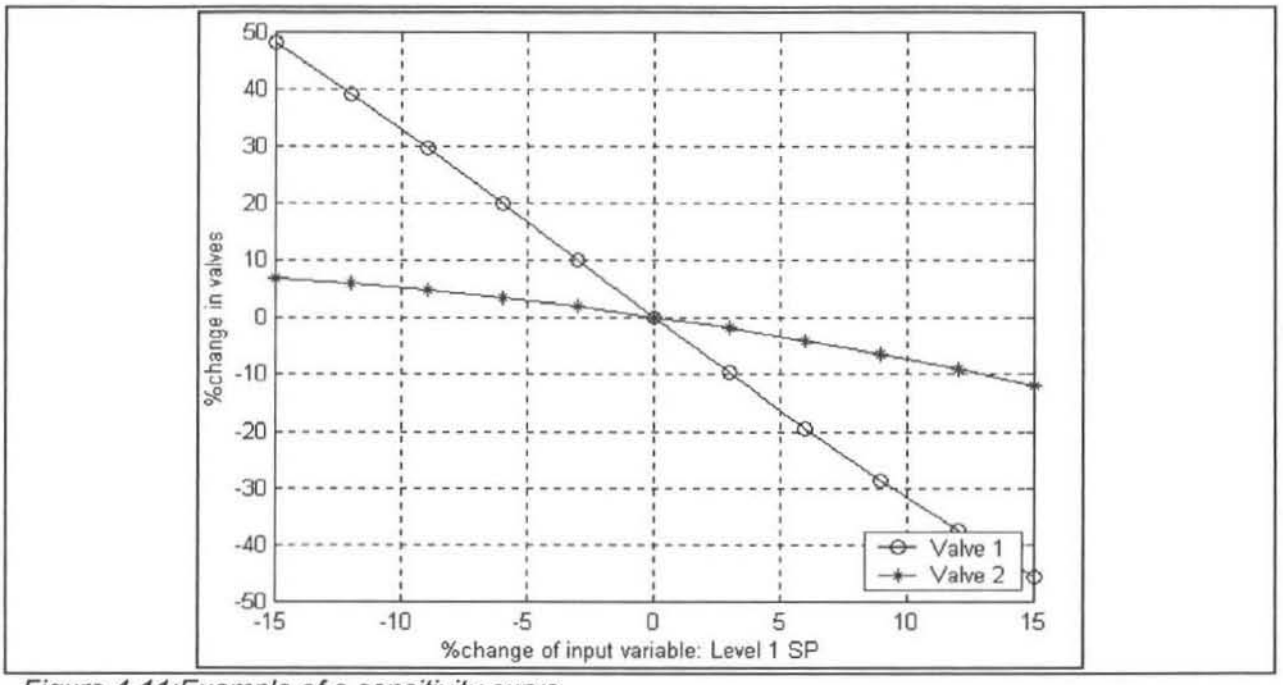


*Figure 4.11:Example of a sensitivity curve*

Another important result that is apparent from these sensitivity curves is whether a system is linear or non-linear. The system above is very close to being linear. In some of the sensitivity curves obtained for the SASOL POLYMERS plant the non-linearity between variables is clearly visible by the curvature of the responses.

Samples of this sensitivity curve software are shown in Appendix J.

# CHAPTER 5

## 5. ARTIFICIAL NEURAL NETWORKS IN CONTROL APPLICATIONS

### *5.1 General Literature Survey*

The use of neural network models in control algorithms is widely reported. In an article by Hussain [1999] a survey of recent papers on neural network control schemes is presented. The number of papers reviewed was close to 100. He attributes the recent 'explosion' in the number of neural network control applications to the following reasons:

- Advances in computing power. He sees their use increasing even more as parallel computing technology becomes more available.
- The ability of neural networks to learn complex process dynamics, especially non-linear systems. This modelling technique is in some cases more effective than developing first principle models. However the availability of rich plant data is a prerequisite.
- Their versatility of their structure enables them to be utilised in the middle ground between conventional model-based approaches and the strictly black box type approaches. The combination forms a hybrid type of solution that is attractive in the chemical industry.
- This versatility ultimately means that they can easily and effectively be incorporated into a multitude of control structures.

The paper groups the control schemes into 3 main groups; model predictive control (MPC), internal model based control (IMC) and adaptive control. Some of the controllers and techniques highlighted in this paper have been mentioned already and an application of one of a modified IMC controller is presented in section 6.2 and Chapter 8.The most notable conclusions from this survey are:

- Multi-layered feedforward neural networks with sigmoidal or hyperbolic activation functions are the most commonly used. They have demonstrated their suitability for most modelling and identification problems, including those with non-linear systems.
- Many control applications of neural networks have been used in on-line situations, however these are nearly all on smaller scale pilot or laboratory plants. The number of applications used in an on-line real plant situation is very small.
- The real test of the neural network control strategy will be to get these applications on-line, and he sees the greatest potential in the IMC scheme.

The paper by Bhat & McAvoy [1989] is another useful reference in the field of neural networks and their application in the control of chemical processes. The paper shows how the backpropagation technique is suited to detecting non-linear process characteristics when compared with traditional modelling techniques. An interesting point in this paper was the technique used to train the inverse plant models. As a result a similar technique was adopted

for this research, which was discussed in section 4.4. This involves training the neural network model with a number of future values of the desired variable to be controlled as inputs. In this case the number of future values was 5. When the neural network is used as a controller the plant setpoint is fed to these inputs.

## *5.2 Control Structures employing Artificial Neural Networks*

Once a reasonable neural network model of the system has been trained the question arises "How can this model be used to control the system better?" The versatility of neural networks allows for many interesting predictive controllers to be configured. These range from simple inverse model controllers to more sophisticated multi-step MPC optimisation controllers. As is the case with model predictive control techniques (MPC) the quality of the neural network controller will be dependent on the quality of the plant model that is developed.

In this section a few of these different controller algorithms will be examined. The main focus is on a modified traditional IMC controller, which is discussed in section 5.4.

### 5.2.1 Simplified Inverse Model Control

Probably the simplest type of neural network controller is an inverse process model of the system as shown in figure 5.1. In this case the inverse model also has a dependence on previous values of the output, which then constitute the feedback term. The "present" value of the output is supplied separately to the controller as the setpoint. As discussed in section 3.1.1 this model would suggest an input (u) to the process which aims to achieve the setpoint output on the next step (dead-beat control). This input could for example be a flow rate setting or the actual valve setting. The necessary control action could be taken and the plant would respond to this providing a new feedback value.

For the calculation of the next control action the current plant output (y) and previous plant input (u) would be presented to the model along with a selection of other inputs including the future setpoint for the plant output (r) as shown in figure 5.2. The number of previous inputs ($N_1$) and outputs ($N_2$) to use is up to the designer, as is the time step between the successive variables. The controller shown in the figures is for a SISO plant but the methodology is the same for a MIMO system.

This type of controller can work efficiently when the neural network model is almost an exact inverse of the process. It will be difficult to get good control of the system if a significant plant-model mismatch exists, since there is no feedback error (e) from an internal model as is the case in IMC. For this reason the controller finds it difficult to account for model error or steady plant disturbances (d). To make sure that the controller suggestions are within reason, excessively large or unreasonably high/low control actions may have to be clipped or constrained.

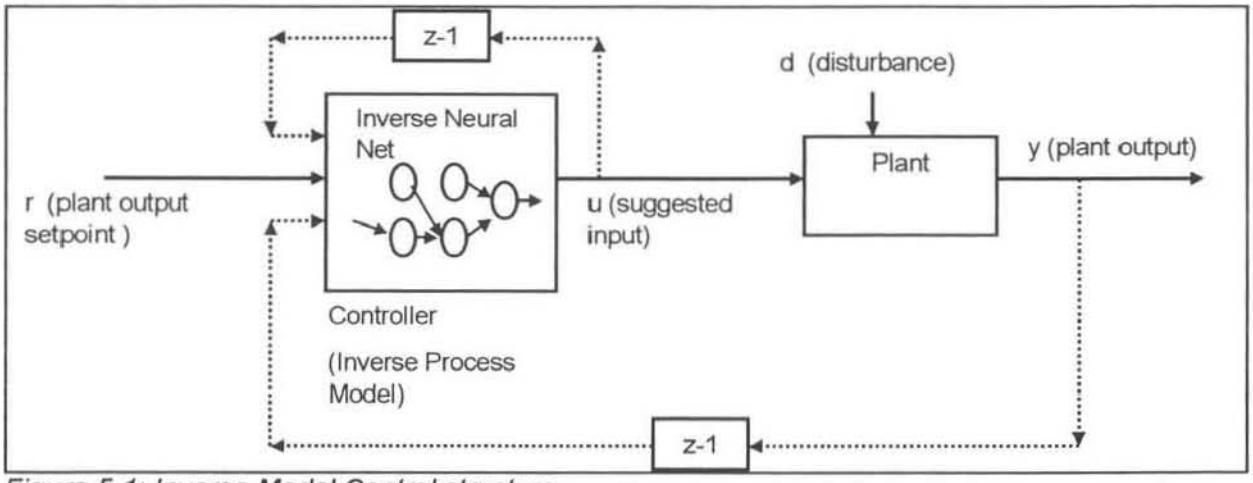


*Figure 5.1: Inverse Model Control structure*

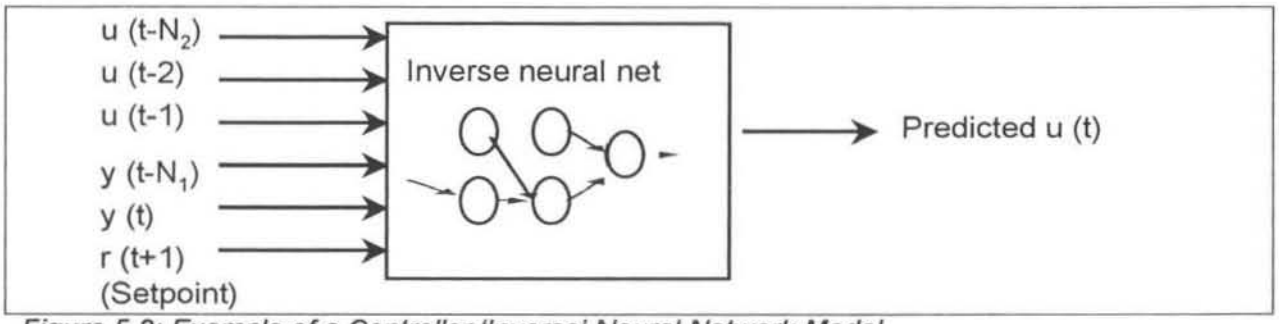


*Figure 5.2: Example of a Controller 'Inverse' Neural Network Model*

This type of controller has been used in off-line and on-line applications [Hussain, 1999] with mixed successes. In an offline application [Sbarbaro, Neumarkel & Hunt (1993)] this method was used to control the strip thickness in a steel rolling process. The inverse model was not used alone here but in several different combinations with PI controllers and an integrator and in an IMC configuration. These were compared with PI (proportional-derivative) controllers and it was found that the inverse model in parallel with the integrator gave the best results. However the IMC and model predictive controllers gave equally good results.

In an on-line application Dirion [1995] used a direct inverse artificial neural network plant model as a temperature controller in a bench scale semi-batch reactor. The reactor was a jacketed glass reactor that had a modified heating-cooling system. Good results were obtained for the temperature setpoint tracking, after experiments and simulations were done to validate the models.

Khalid and Omar [1992] trained an inverse artificial neural network to model the inverse of a bench-scale heated water bath. The inverse model was used directly to control the temperature in the bath. They tested the setpoint tracking, disturbance-rejection and the effect of dead time on the control. The results they obtained were better than those for a conventional PI controller.

VanCan and Bracke [1995] presented an interesting method of obtaining an inverse neural network model. They trained a forward model on a laboratory pressure vessel, the aim being to control the pressure by adjusting the inlet air flow rate. The forward model was numerically

inverted and implemented as a direct controller. Setpoint tracking comparisons were made using a PI and a linear model controller and it was found that the artificial neural network controller performed comparably to the PI controller and better than the linear model based controller.

An application of the controller shown in figure 5.1 was tested in section 6.2. The results showed that without the feedback error correction term (e) the control was indeed poor. When using the same controller model in an IMC structure (see the next section) the results were far better.

### 5.2.2 Traditional Internal Model Control Structure

A more commonly used technique is Internal Model Control (IMC). The general form of this was discussed in section 3.3. The neural network model modified version is shown in figure 5.3. The inputs to the neural network models have not been shown to keep the diagram simple, although the controller model would resemble the model shown in figure 5.2. The forward plant model would have a similar structure except it predicts the plant output ($y_{pred}$).

As mentioned in section 3.3 the main advantages of the IMC controller is that it explicitly takes into account model uncertainty and corrects for this via the error feedback term (e). It also provides reliable robust control and allows the designer to concentrate on the model accuracy without having to worry about the stability of the controller.

Another advantage of using neural network models is that independent forward and inverse process models can be developed. Recall that for stability the controller model must be an inverse model of the actual plant, and not the plant model simply inverted [de Vaal, 1999]. By using neural network models the relevant forward and inverse models are developed separately and are not direct inverses.

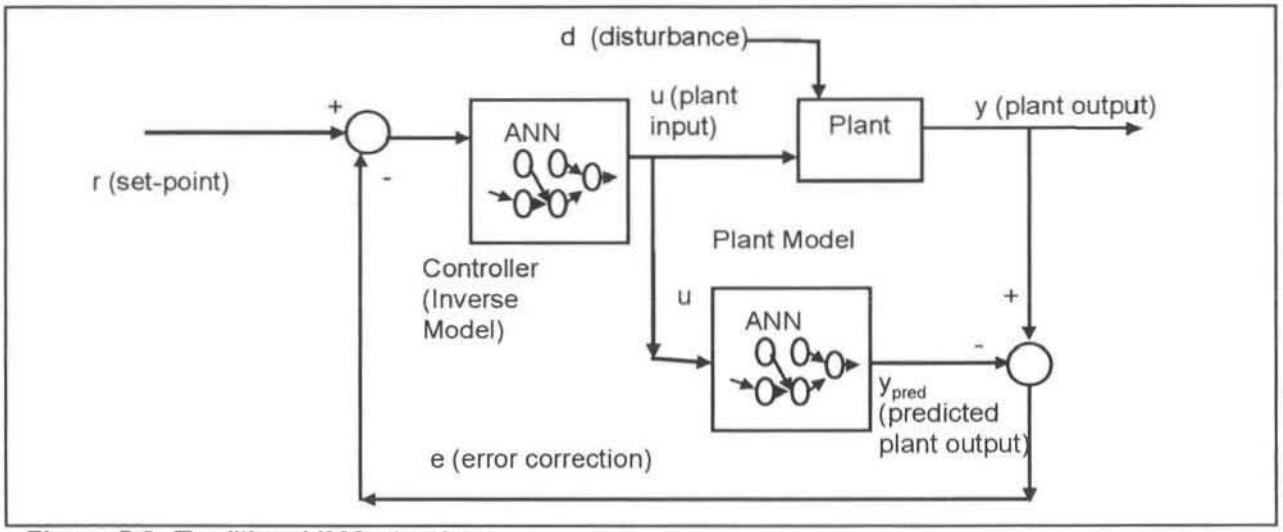


*Figure 5.3: Traditional IMC structure*

The IMC structure corrects the controller (inverse neural network model) prediction by having the feedback error correction (e). This structure will be especially useful when the neural network model accuracy is not ideal. The error term is used to correct for the offset between the plant and model.

In the paper by Hussain [1999] a whole section is devoted to reviews on IMC applications and covers over 100 papers. In general most of the papers showed that the results obtained using IMC were comparable or outperformed traditional PID control or other forms of MPC. The range of systems experimented on was very diverse and included CSTR's and neutralisation reactors, batch fed bioprocesses, distillation columns and a rolling mill.

Evans [1993] used artificial neural networks in a model predictive controller (MPC) on similar equipment to that used in section 6.2. It was a laboratory process consisting of 2 non-interacting tanks in series. The neural network model predicted the process outputs (u) for a number of time steps ahead. When compared to a traditional PID controller the neural network controller was found to give pleasing results. The speed of response of the controller was good and it was observed that the control actions suggested were very smooth. In chapter 6.2 of this work a very similar result was obtained.

Dubois, Nicolas & Billat [1994] tested an adaptive IMC controller to control the temperature in an oven system. The plant model (see figure 5.3) was updated online, by recalculating the weights for the radial bias function neural network. They could not find an accurate inverse model (controller model) by using plant data and instead trained it using data from the forward model. It was found to follow the given setpoint trajectories satisfactorily.

Several successful on-line applications of neural network IMC were also reviewed. Hussain & Kershenbaum [2000] reported good results for temperature control in a partially simulated pilot-plant reactor. After extensive testing using simulations of the reactor, the IMC controller was run online and showed that the controller performed well in adjusting to setpoint changes and load disturbances, despite the existence of a significant plant/model mismatch. Since the system was mildly non-linear the controller did exhibit slightly better performance than a classical PID loop that was tested.

Keeler and Martin [1997] implemented a non-linear, neural network based, predictive controller on an industrial polypropylene plant to control the melt flow index (MFI). They developed their controller in software called Process Perfector. The package utilised steady state and dynamic artificial neural network models of the process, combined with a dynamic optimisation program which calculated the control actions. The results showed that good setpoint tracking was possible, and it compared especially well with traditional linear model predictive method.

Seborg [1994] showed that the IMC controller outperformed a PI controller in a two tank neutralisation system. The radial basis function neural networks were modelled to regulate the pH in the system under disturbances in the acid and buffer flow rate. The structure of the controller was similar to that in figure 5.3, except that the suggested control action (u) was modified slightly. This was done to minimise a certain performance criterion, similar to the simple clipping and ramp-limiting that was done in section 6.2. As mentioned the controller performed better than the PI controller, mainly because of the artificial neural network's ability to model the non-linearity of the system.

An interesting point noted from several of the reviews [Nahas, Henson & Seborg (1992); Dayal, Taylor & MacGregor (1994)] was that the trained inverse neural network models performed better than the numerically inverted neural network models and required less computational time. In fact one of the main advantages of neural networks is that totally separate forward and inverse models can be developed, which are independent of each other, yet using the same data set. The option of using numerically inverted neural network models was not considered necessary for the work in this dissertation, since only trained inverse models were required.

Perhaps one of the disadvantages of the IMC structures is that they only predict the control action (input) one step ahead. Since a control action is taken on every step these actions may be quite big step changes and may tend to work against each. It would be desirable to modify this controller so that it considers several time steps in an optimisation cycle, such as in the DMC algorithm. However this type of predictive controller was not investigated in this dissertation.

### 5.2.3 Modified Internal Model Control Structure

The modified two-degrees-of-freedom IMC structure [Garcia et al, 1989] that was discussed in section 3.4 has the potential to be a useful controller, although no references using this particular configuration have been found. The neural network model based form of this structure including the feedback error (e) is shown in figure 5.4. This is a more detailed version of the diagram shown in figure 3.3. As mentioned previously this structure has been termed "Inverse Internal Model Control (IIMC) ". The main reasons for choosing to use this structure are recapped here:

- It was thought that the modified feedback error (e) would be more effective than the method used in traditional IMC. This is since two identical models are used, so there is no disparity between the two models.
- The modelling effort is reduced, since only one model is required.

This is similar to the controller in figures 5.2 and 5.3 except that some form of feedback is provided. This feedback error (e) is crucial in correcting the controller for plant-model mismatch and the effect of plant disturbances (d). The inverse plant models can be time-invariant algebraic or time-invariant dynamic neural network models. The length of the time delay that is

used for the 'old' plant input ($u_{t-1}$) depends on the process. This obviously shouldn't be long enough to allow the plant to radically change operating conditions, since the effectiveness of the feedback error would be lessened.

The actual inputs to the two models are very similar. The controller model (inverse model No. 1) would take the plant setpoints as inputs (i.e. $C_6/C_2$ composition SP) whereas the predictor model (inverse model No. 2) would take the actual value (i.e. actual $C_6/C_2$ ratio at time t). If the controller model used a process variable such as the current reactor temperature as an input, then the predictor model would use the 'old' temperature value (temperature at time t-1).

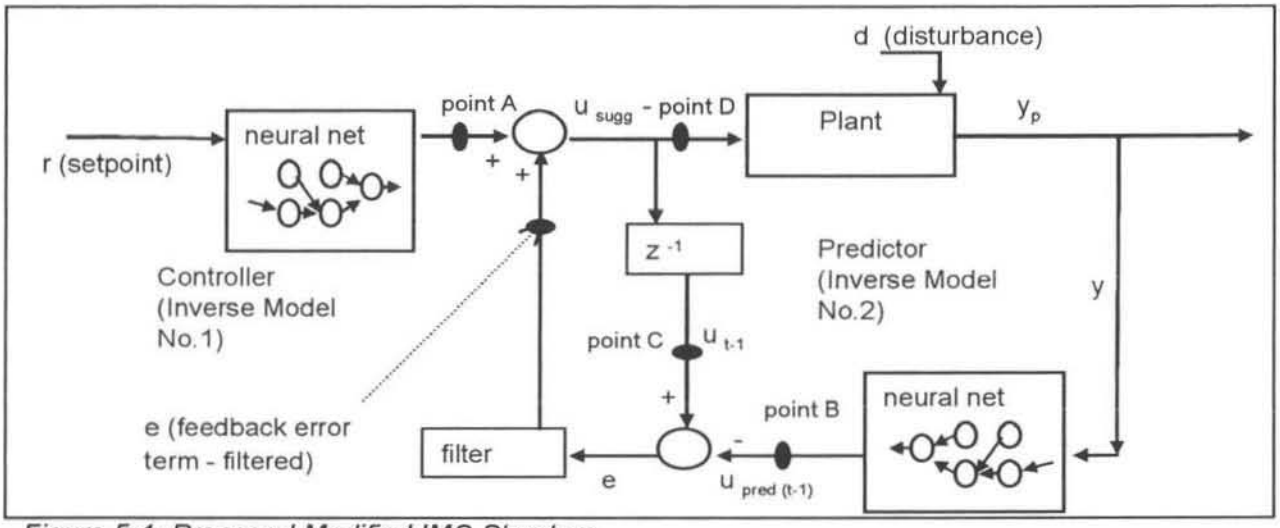


*Figure 5.4: Proposed Modified IMC Structure*

A single-exponential filter has been used to smooth the values of the feedback error term (e). The equation for this filter is given in below. The designer is left with the decision for the alpha values, which are always less than or equal to 1. These values may need to be tuned on-line depending on the plant conditions and responses to disturbances. A lower alpha value will result in a stronger smoothing effect on the signal by the filter. The effectiveness of filtering the feedback error (e) can be seen from the results in section 6.2.

$$y\_bar_t = (1 - \alpha)\, y\_bar_{t-1} + \alpha\, y_t \qquad \textit{Equation 5.1}$$

y_bar -> filtered variable

y -> raw variable.

$\alpha$ -> filter constant (always less than or equal to 1)

This controller was tested on a tanks system and is discussed in detail in section 6.2. The controller was not tested on-line in a closed loop on the polymer plant that was studied for this research. The open-loop predictions are discussed in chapter 8.

# CHAPTER 6

## 6. ARTIFICIAL NEURAL NETWORK CASE STUDIES

Several case study examples were considered to get familiar with the training of neural network models and using these models in different control algorithms.

### *6.1 Modelling a Two Tank Equation System*

#### 6.1.1 Introduction

The first system considered was a non-linear set of equations that represent a model of an interacting two-tank system in the Chemical Engineering Department at the University of Natal. This simple system is shown in figure 6.16. The simple models were trained on existing software and then tested using several different algorithms, which were written in MATLAB. The equations 6.1 and 6.2 represent a MIMO system comprising two inputs and two outputs as shown in figure 6.1 below.

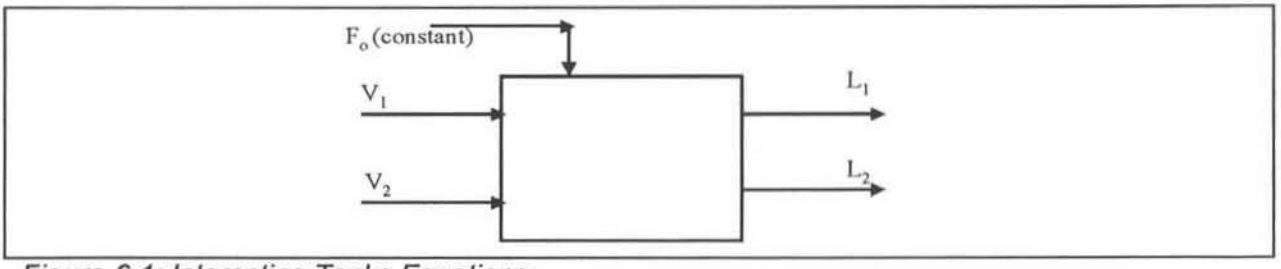


*Figure 6.1: Interacting Tanks Equations*

$$\frac{dL_1}{dt} = F_o - 2 * V_1 - 0.1 * V_2 * (L_1 - L_2) - L_1 \qquad \textit{Equation 6.1}$$

$$\frac{dL_2}{dt} = 0.1 * V_2 * (L_1 - L_2) - 2 * L_1 \qquad \textit{Equation 6.2}$$

where

Fo = 270 (units of flow)
$L_1$ = level 1
$dL_1$ = change in level 1
$V_1$ = valve 1
dt = change in time

The inputs to the system are the two valves, valve 1($V_1$) and valve 2 ($V_2$). The outputs from the system are the two tank levels, level 1($L_1$) and level 2 ($L_2$). The variable $F_o$ is the input flow to the tanks and has the value of 270 units. The actual tank system was used in a subsequent

case study. A more detailed description of the equipment configuration and control follow in the next section.

### 6.1.2 Training the Process Neural Network Models

The first step in training a neural network model is to obtain data from the system. This data must accurately reflect the nature of the process so that the neural network can learn this behaviour. Ideally this data should be relatively noise free and cover the entire expected operating conditions of the system. This data for training was prepared using MATLAB by using Euler integration on equations 6.1 and 6.2. The modified integration equations are shown in equations 6.3 and 6.4 below.

$$L_{1t+1} = L_{1t} + \Delta t * (270 - 2V_{1t} - 0.1V_{2t}(L_{1t} - L_{2t}) - L_{1t}) \qquad \textit{Equation 6.3}$$

$$L_{2t+1} = L_{2t} + \Delta t *( 0.1V_{2t}(L_{1t} - L_{2t}) - 2L_{1t}) \qquad \textit{Equation 6.4}$$

where

$L_{1t+1}$ = level 1 at time t+1

$L_{1t}$ = level 1 at time t

The equations had to be integrated on a very small time interval otherwise the system went unstable. A 0.1 second time step ($\Delta t$) was used. The valve positions were varied quite considerably so as to obtain level responses for the whole operating range of the system. An example of the data is shown in figure 6.2.

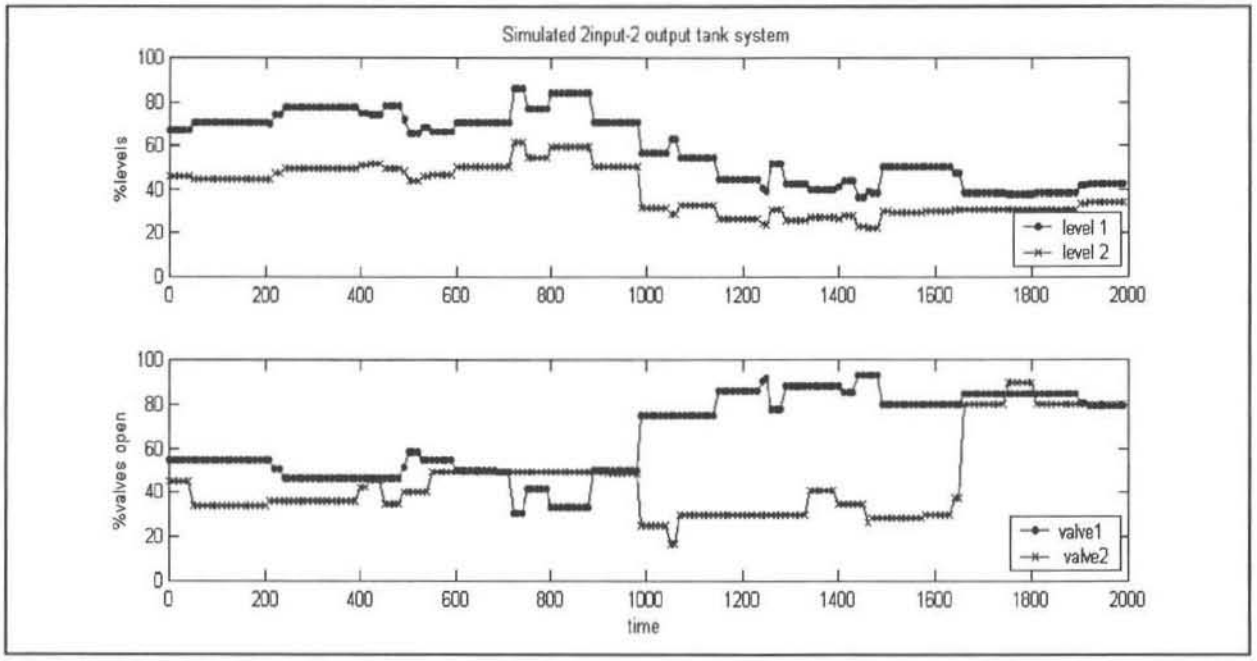


*Figure 6.2: Sample of Data from Tanks Equations*

The following neural network was trained as a forward model of the system. This neural network model should be able to replicate the behaviour of the system if it has been trained on good representative data and predict the tank levels one time step ahead.

This is a dynamic neural network model since previous valve and level positions have been used to predict the future level positions ($L_{1\,t+1}$ and $L_{2\,t+1}$). The inputs and outputs presented to the network during the training and application phases are as in figure 6.3 below. They include the current valve positions ($V_{i\,t}$) and levels ($L_{i\,t}$) and the 2 previous level and valve positions. The size of the time step between data points used in this model is 1 second. The neural network model had a single hidden layer with 6 hidden nodes, 12 input nodes and 2 output nodes.

| Inputs | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Level 1 | | | Level 2. | | | Valve 1 | | | Valve 2 | | |
| t–2 | t-1 | t | t–2 | t-1 | t | t–2 | t-1 | t | t–2 | t-1 | t |

| Outputs | |
|---|---|
| Level 1 | Level 2 |
| t + 1 | t + 1 |

*Figure 6.3: Structure of Forward model of Tanks equations*

### 6.1.3 Neural Network Model Results

Numerous different sets of weights were obtained. These weights were trained from different sets of data. Throughout this section two of these sets of weights will be presented to show how to distinguish accurate neural network models from poor models. The first model, call it Model 1, is an example of a good model - it was trained on data that included plenty of step responses and covered the entire operating range of the system. The second model, Model 2, was trained on very flat data and only covered a small portion of the operating range, however the same number of unique data points were used in the training set. As a result of this the model is not able to accurately predict levels in the certain operating regions. Also it is not able to cope with large changes in the valve positions.

#### 6.1.3.1 1-Step Model Predictions

As a first indication of whether the models had learnt to mimic the systems behaviour the predicted levels versus actual levels were plotted against each other. The differences between these two sets of curves is not large, which is as expected. The relative change between the current level ($L_{i\,(t)}$) and the level on the next time step ($L_{i\,(t+1)}$) will not be large since the time step of 1 second is quite short. From figures 6.4 and figures 6.5 it can clearly be seen which is the better model, *viz* Model 1. In figure 6.5, which is Model 2, the poorly trained model, it is

clear that the model cannot handle the high level 1 values. Although the model predicts the steps in level 1 correctly it is unable to anticipate the magnitude. This is since the data from which it was trained did not include these large steps. The level 2 predictions seem to be similar to the good model.

The predictions of level 1 and 2 in figure 6.4 are very close to the real values. Looking at these results it would be easy to assume that a fairly accurate model of the system had been found. There are however still a few more tests that can be performed on the models to make sure that they will behave as expected.

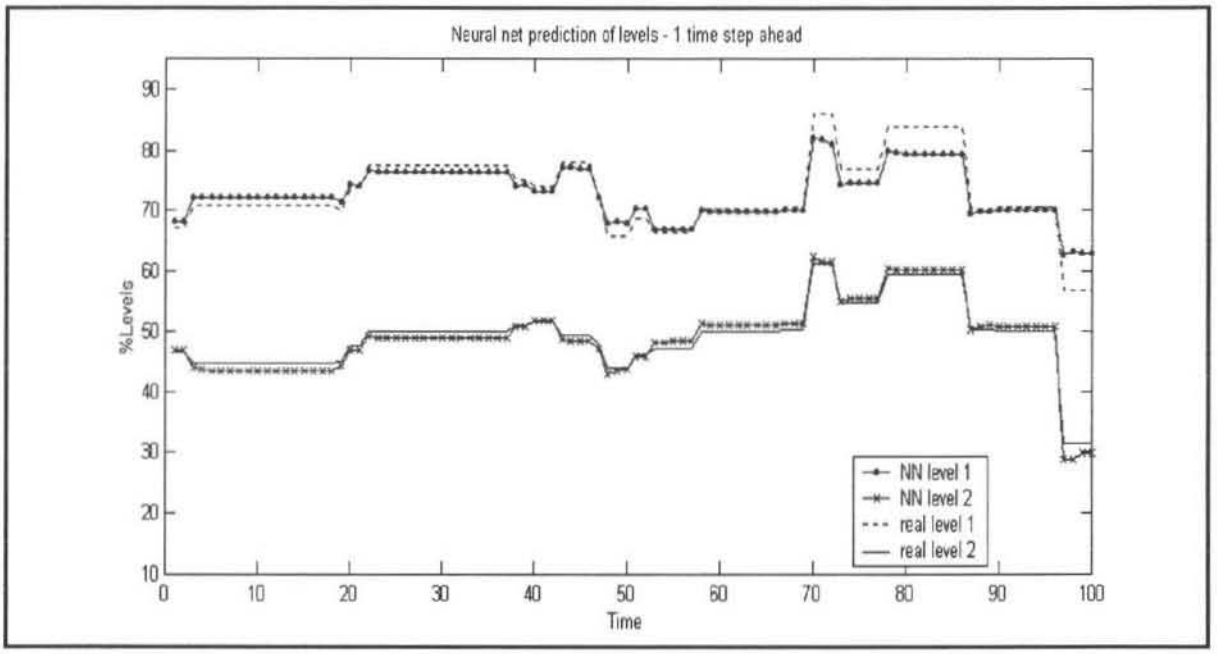


*Figure 6.4: 1-step predictions of Model 1*

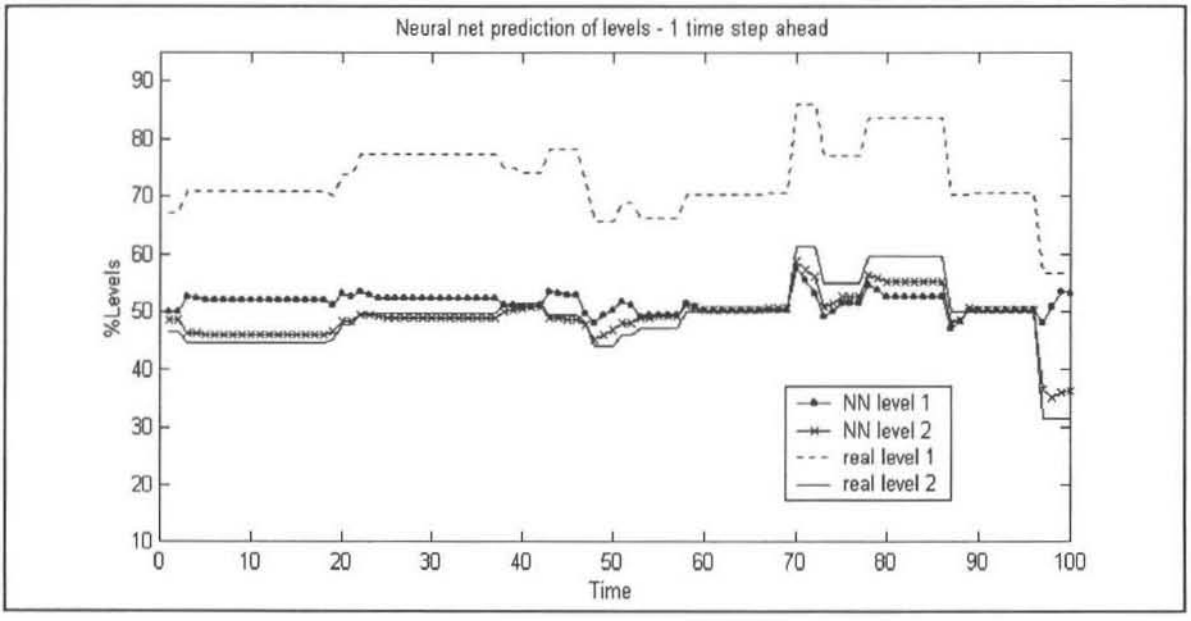


*Figure 6.5: 1 step predictions of Model 2*

### 6.3.1.2 Sensitivity Curves of the Models

It is important to know that the model will respond to a disturbance in the system in the same way that the real system does. This can be determined from sensitivity curves. In this case they give an indication of what the level prediction will do when the valve positions are changed, according to the models. Referring to figure 6.3 above showing the model structure, the current valve positions are perturbed individually so that the response of levels can be observed.

As an example, it is known that in the case of this tank system levels 1 and 2 will decrease if valve 1 is opened. Also if valve 2 is opened the level 1 will decrease while level 2 will rise. In figures 6.6 to 6.9 we can see this occurring. Figures 6.6 and 6.7 show the effect that changing valve 1 has on levels 1 and 2, according to Models 1 and 2 respectively. Figures 6.8 and 6.9 show the effect that changing valve 2 has on levels 1 and 2, according to Models 1 and 2 respectively. It is seen that both models predict the same behaviour, however the magnitude of the responses differs quite considerably. An interesting result from these curves can be seen in figures 6.6 and 6.7. The coupling between equations 6.1 and 6.2 is clearly represented. As valve 1 is opened level 2 will decrease. This is an example of why neural networks are so useful for modelling.

These sensitivity curves show that both models have learnt the system's behaviour correctly. The level responses are consistent with the known behaviour of the system as is established in the next case study. It is very important to have confidence in these curves since they are the primary way of determining how the models will react when the system is disturbed.

All this appears to confirm what was established above, that Model 1 is the more accurate model. Model 2, because it was trained on poor data, is unable to properly predict the size of the level responses when the valve positions are changed.

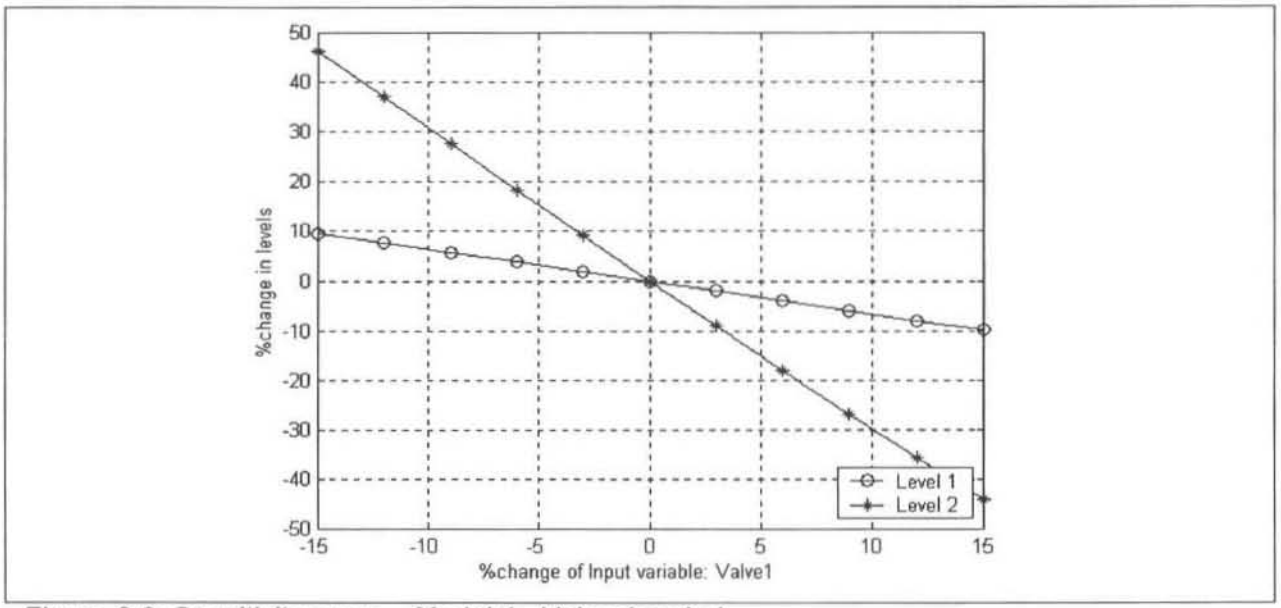


*Figure 6.6: Sensitivity curve – Model 1– Valve 1 varied*

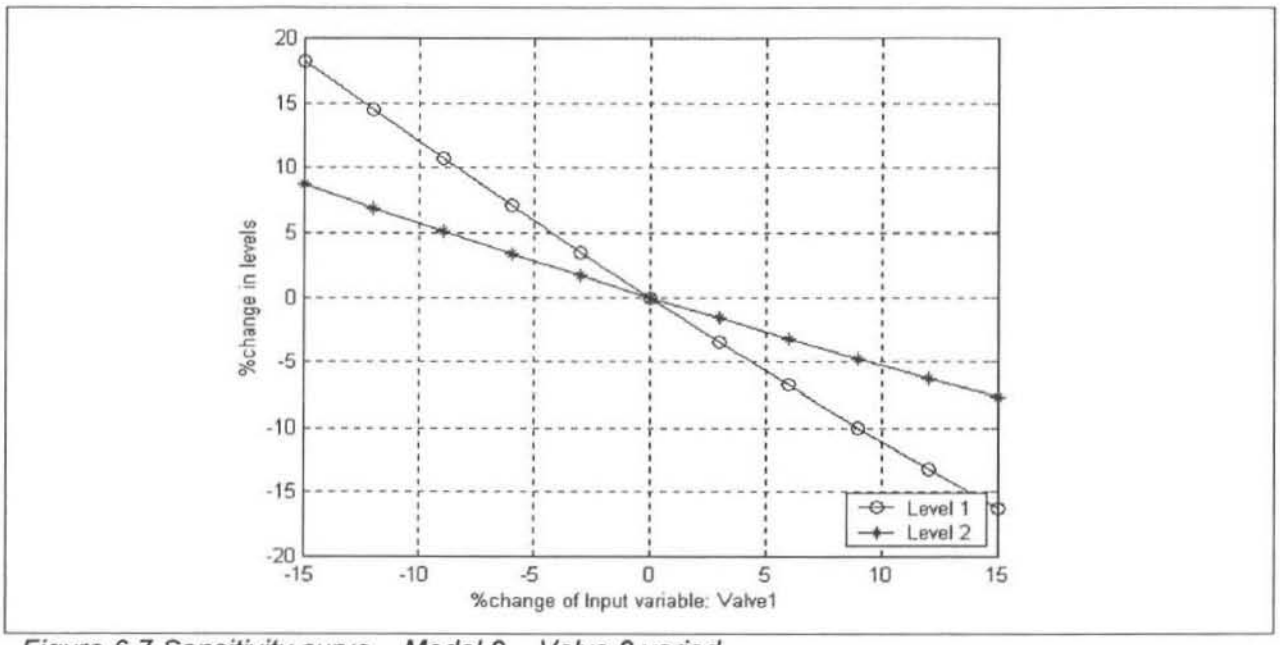


*Figure 6.7 Sensitivity curve – Model 2 – Valve 2 varied*

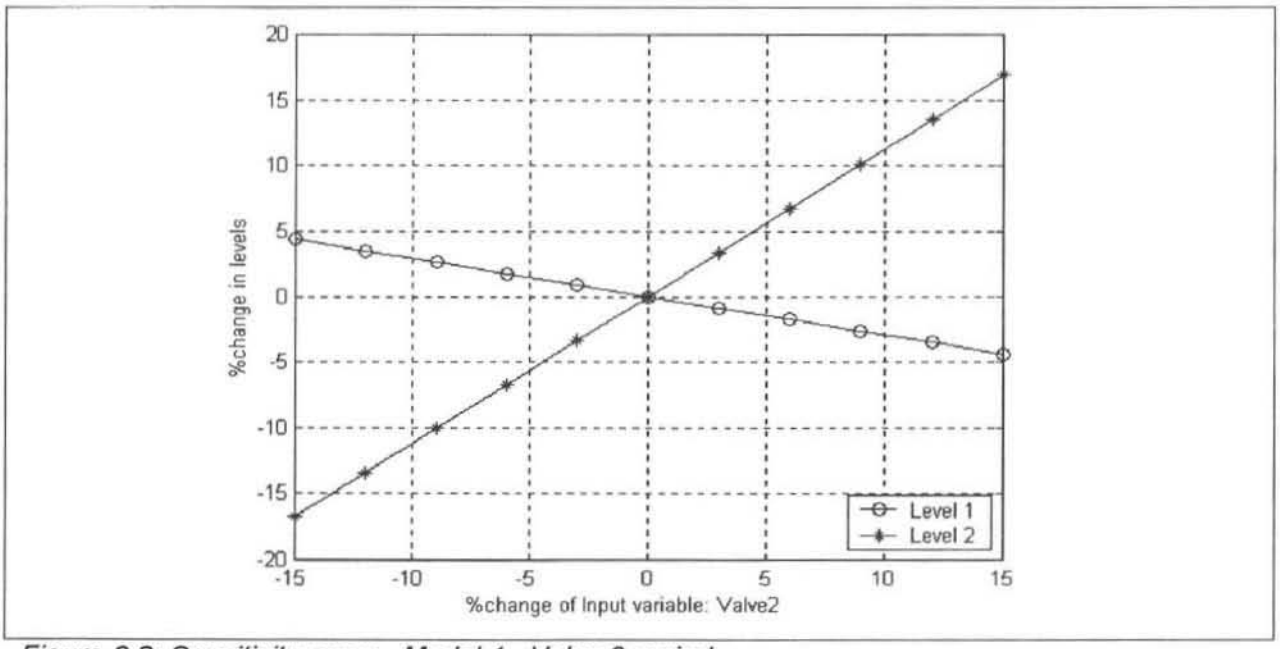


*Figure 6.8: Sensitivity curve –Model 1– Valve 2 varied*

### 6.1.3.3 Chaining of the Models

Probably the best tests of a neural network are its multi-step predictions versus the actual values, in other words, to chain the neural network. However this has only been considered for the forward model of the system, as discussed in section 4.5.2.

For these examples only the simplest type of chaining was applied. The first neural network prediction is 'attached' to the first actual level values, by subtracting the error difference from the neural network prediction. This error correction is held constant for the rest of the prediction horizon. At the end of the prediction horizon this error is recalculated and the new error value used to reattach the predicted level to the actual value. The amount of relative drift that occurs

in the neural network predictions gives an indication of the accuracy of the model fit. In all cases completely new input data were presented to the neural networks when testing this chaining.

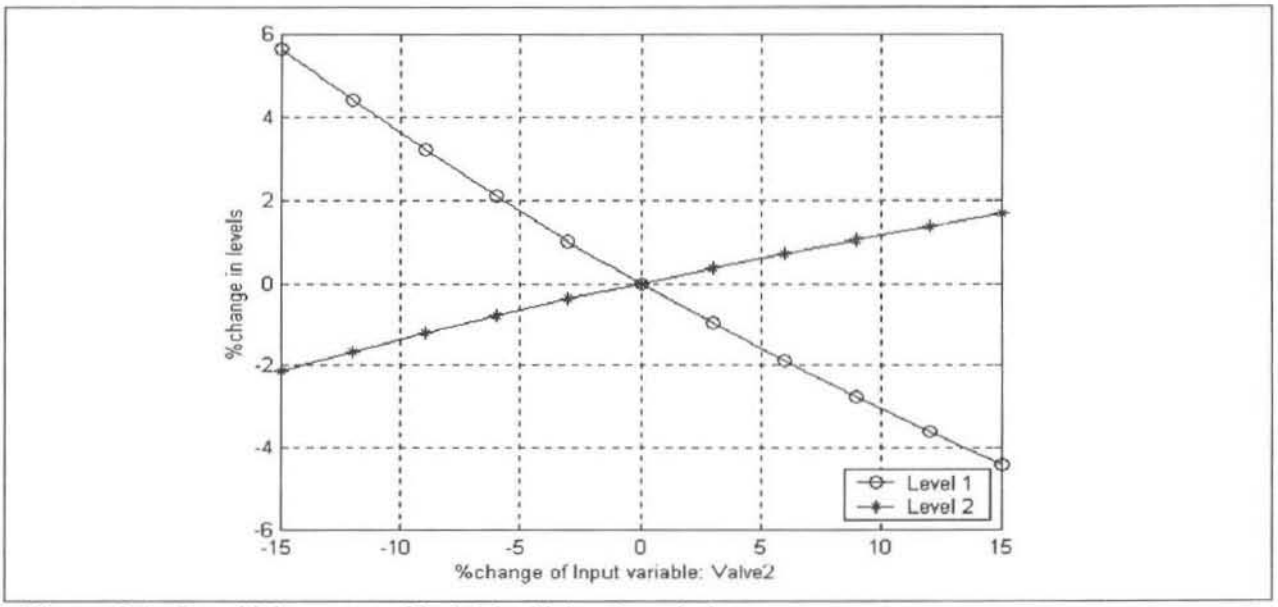


*Figure 6.9: Sensitivity curve –Model 2– Valve 2 varied*

In figures 6.10 and 6.11 Model 1 is chained over a prediction horizon of 197 time steps without re-attachment to the observations. It can be seen that both level predictions follow the actual outputs very closely over the chaining interval. The differences between the respective levels and level predictions are small as can be seen in the table of standard deviations, figure 6.14. This indicates that the network has learned the system's behaviour well. The results here demonstrate how effective neural networks can be at modelling. In this example the model has no reference to what the actual levels are doing. The 'old' model predictions are used for the past level neural network inputs. This means that apart from the first prediction, the neural network model has no idea what the actual levels are.

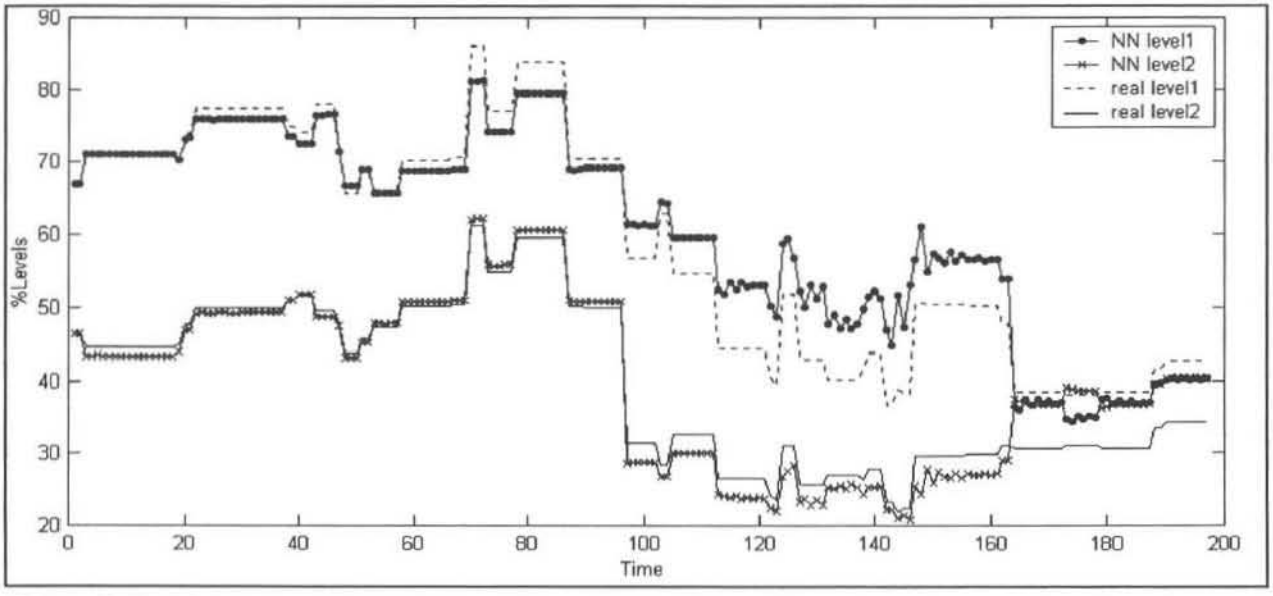


*Figure 6.10: Multi-Step (chaining) Predictions of Model 1*

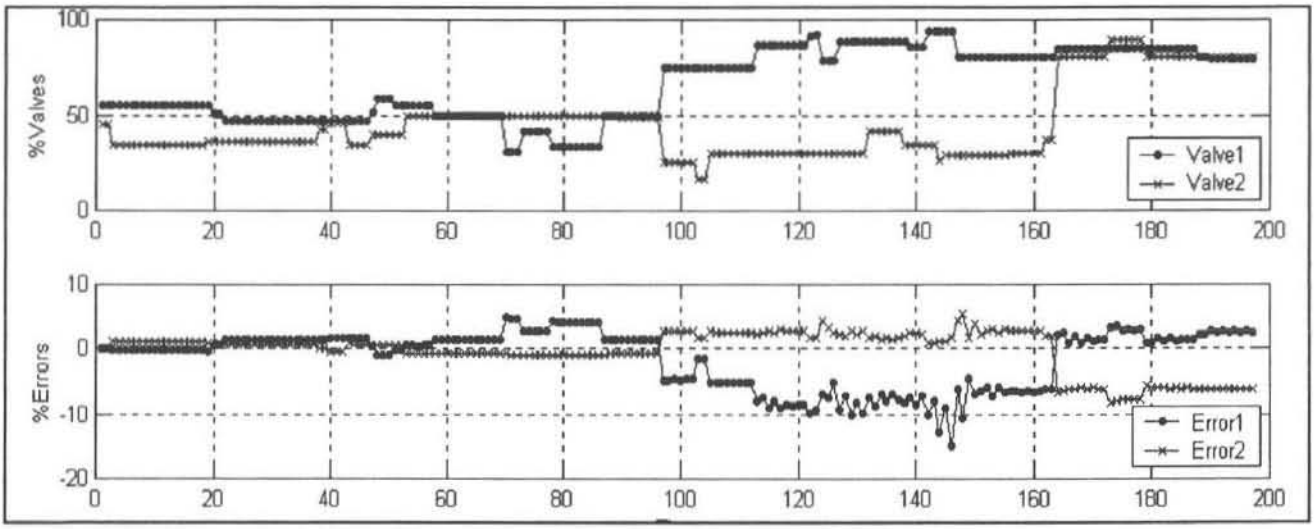


*Figure 6.11: Valve Positions and Prediction Errors – Model 1*

The results of chaining Model 2 over the same prediction horizon are shown in figure 6.12. This models predictions show that it is not as accurate as Model 1. The standard deviations of the errors are slightly larger than for Model 1. As in the above comparisons Model 2 is unable to predict the size of the level responses as well as Model 1.

After clearly establishing Model 1 as the better of the two models it was interesting to compare how Model 1 performed with different prediction horizons. It would be expected that as the prediction horizon decreased then so should the standard deviation of the errors. Figure 6.15 shows Model 1 chained over a 25 step time interval. It can be seen how the difference between the actual and predicted values is reset to zero every 25 time steps.

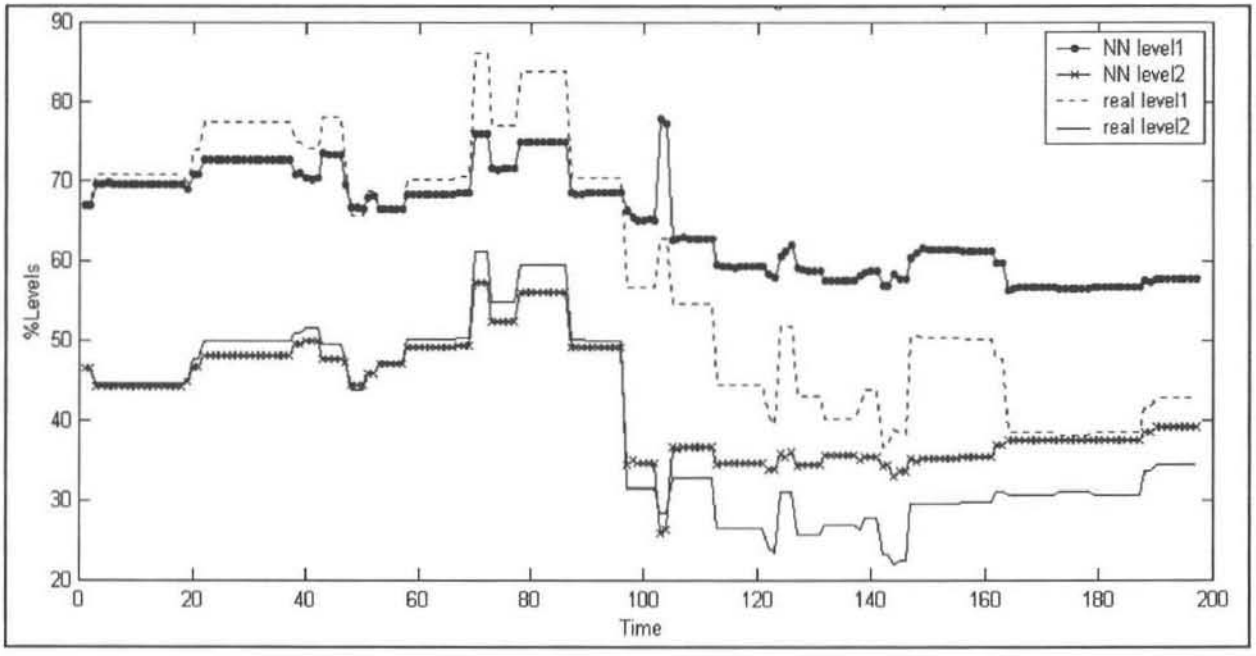


*Figure 6.12: Multi-Step (chaining) Predictions of Model 2*

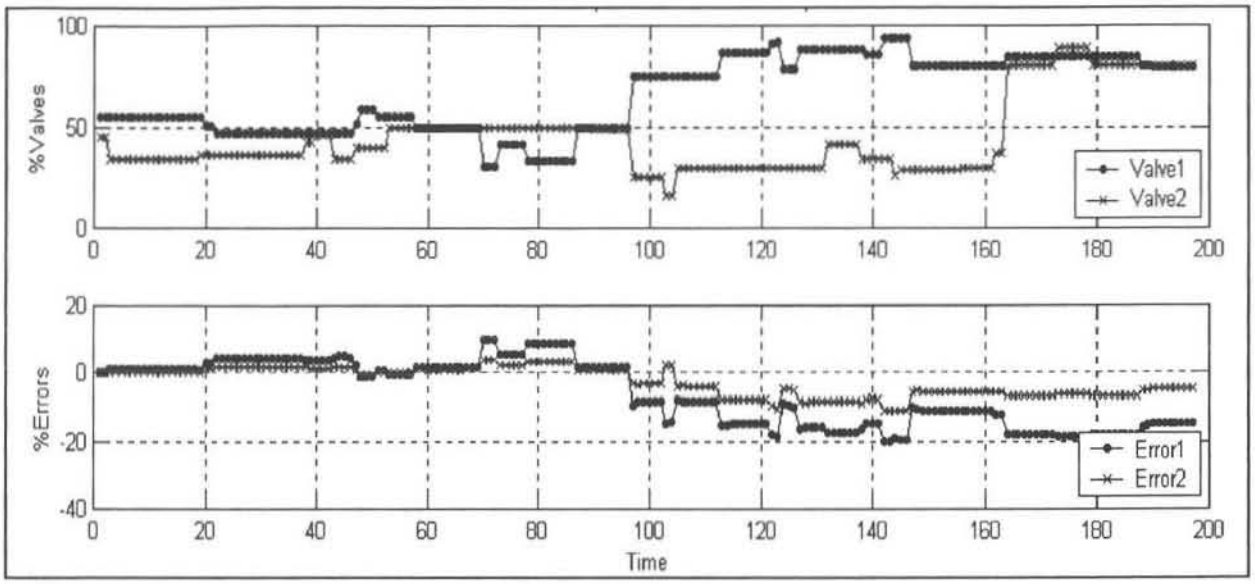


*Figure 6.13: Valve Positions and Prediction Errors – Model 2*

Similar tests were conducted for prediction horizons of 10, 50, 75, 100, and 150 time steps. The table below, figure 6.14, shows the respective standard deviations of the errors.

| Model and Prediction horizon | Standard deviation. Level 1 prediction | Standard deviation. Level 2 prediction | Average standard deviation |
|---|---|---|---|
| Model 1 – 197 | 4.629 | 3.108 | 3.868 |
| Model 2 – 197 | 11.191 | 4.935 | 8.063 |
| Model 1 – 150 | 5.672 | 4.400 | 5.036 |
| Model 1 – 100 | 3.708 | 3.998 | 3.853 |
| Model 1 – 75 | 6.678 | 4.742 | 5.710 |
| Model 1 – 50 | 4.866 | 4.336 | 4.601 |
| Model 1 – 25 | 3.484 | 2.726 | 3.105 |
| Model 1 – 10 | 2.418 | 1.970 | 2.194 |

*Figure 6.14: Table of Standard Deviations of Multi-Step Model Predictions*

The standard deviation of the errors of the small prediction horizons (10 and 25 steps) is clearly lower than the rest, which lie in the range [3.8 ; 5.8]. Above 50 time steps the standard deviations of the errors, for Model 1, do not follow a clear trend. The conclusion from this is the simple fact that the shorter prediction horizons (10 to 25 steps) give a lower standard deviation, and hence better predictability than the horizons greater than 25.

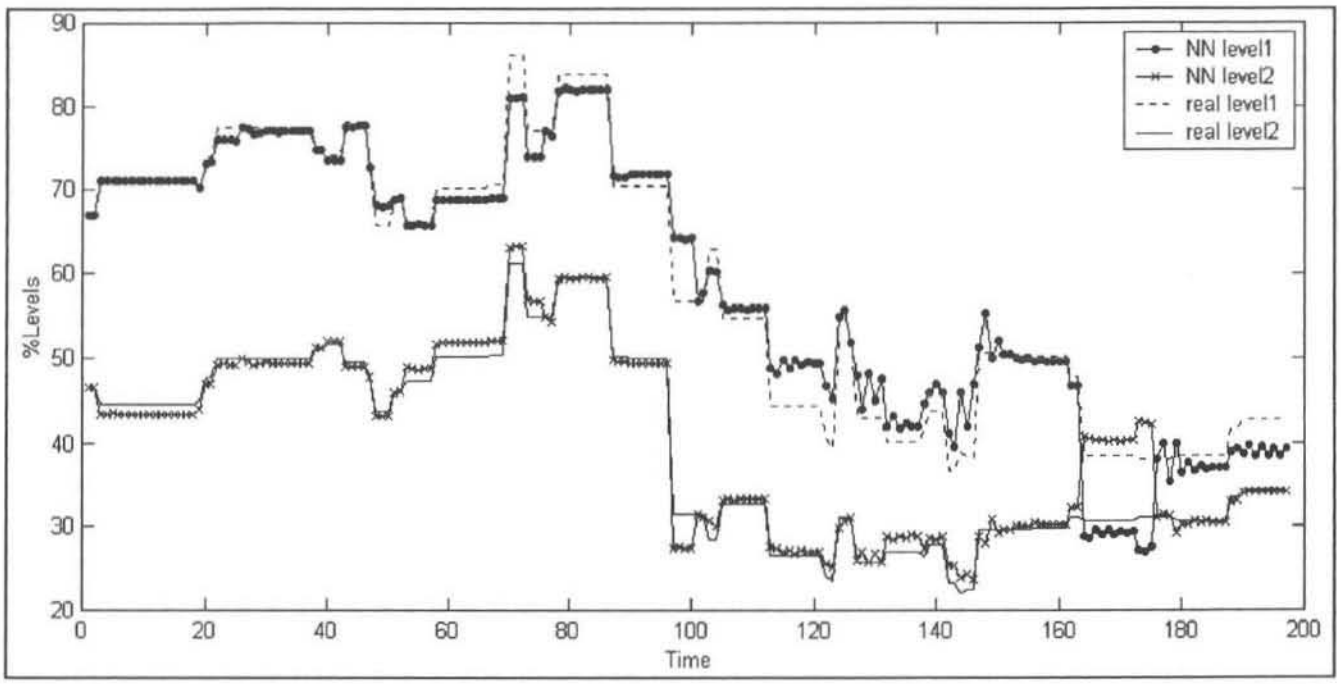


*Figure 6.15: 25-Step (chaining) Predictions of Model 1*

## 6.2 IIMC Control of a Real Tanks System

### 6.2.1 Introduction

The second documented case study that was performed on the actual interacting tanks system in the laboratory at the School of Chemical Engineering, University of Natal. This is the system that the equations used in the previous section were modelled on. The system is shown in figure 6.16 below. As was the case with the equations this is a 2 input 2 output coupled non-linear system.

The aim of this study was to attempt to construct and test the modified IIMC (Inverse Internal Model Control) structure that was discussed in section 5.2.3. This multivariable controller was designed to regulate the levels in the two tanks by manipulating the water flow to them through the two valves. The controller consists mainly of two inverse trained neural network models of the tank system and an error correction between the two models. The effectiveness of the controller was evaluated with different internal settings and compared to a traditional DMC (Dynamic Matrix Control) controller.

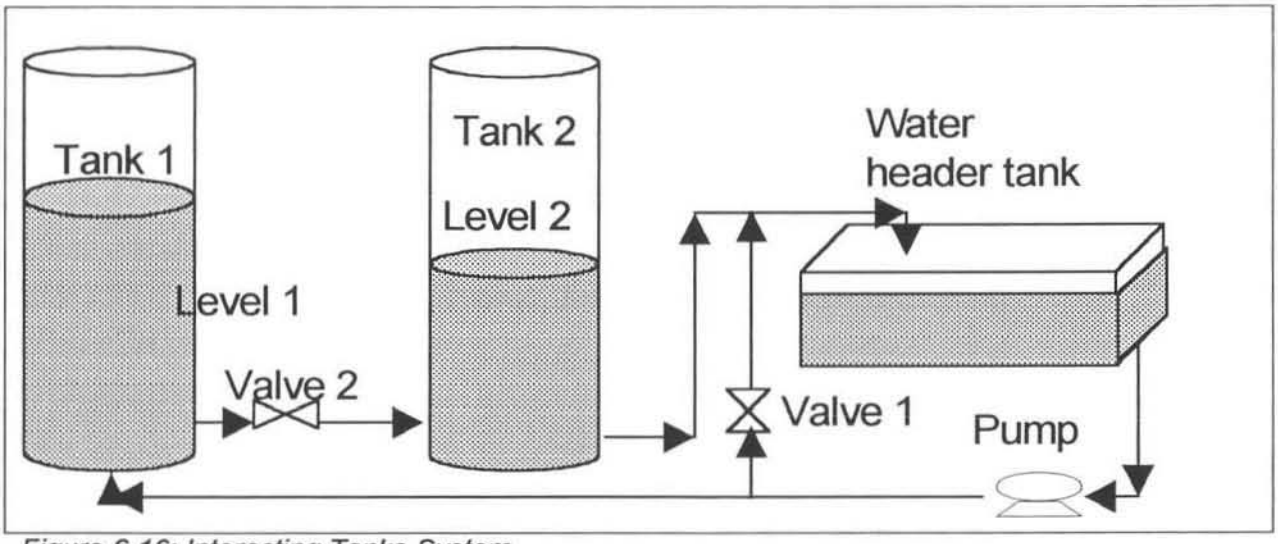


*Figure 6.16: Interacting Tanks System*

Tank 1 is actually composed of 3 tall linked tanks. This is to increase the volume of the 'first' tank so that the time-constant is not too dissimilar to that of the second tank, which deals with a much smaller flow. Water is pumped from the main header tank into the delivery line. A recycle branches from here and this is where Valve 1 is situated. The amount of water fed off to the recycle determines the volume of water delivered by the pump to the first tank. The water flow from Tank 1 to Tank 2 is controlled by Valve 2. As Valve 2 is opened, more water will flow from Tank 1 to Tank 2, reducing Level 1 and increasing Level 2. From Tank 2 the water drains by gravity to the supply tank. The settling time of the levels in response to valve changes is approximately 3 minutes for tank 1 and 6 minutes for tank 2., remembering that Tank 1 is actually 3 tall linked tanks. This is the time it takes for the response to approximately reach steady state.

Level sensors on Tanks 1 and 2 send signals to a control computer (PC). The C++ SCADA control software developed in the department is loaded on this PC. This SCADA software is responsible for controlling the movement of Valves 1 and 2. The neural network IIMC controller code had to be programmed in the SCADA software. In this SCADA software there is scope for the development for different types of control algorithms. On this and other experimental systems in the department, various control strategies such as Kalman filter's, EKF's (Extended Kalman Filters) with on-line RLS parameter estimation and DMC controllers have been implemented.

### 6.2.2 Obtaining an Inverse Neural Network Model of the System

For any model-based control to be successful it is essential to have an accurate representation of the system under consideration. The results in the previous case study prove how important it is for the neural network model of the system to be trained with characteristic plant data. This data must reveal the full range of expected operating conditions on the plant so that the controller is able to deal with all moods of the plant. As is the case with most model predictive controllers the quality of the controller will be dependent on the quality of the model.

There are several important time delays in the system to consider, as discussed in section 6.2.1 valve changes will not cause immediate level changes. These time delays have to be taken into account when developing the neural network model. It is important to carefully consider how far back in the data records to look when establishing relationships between the variables.

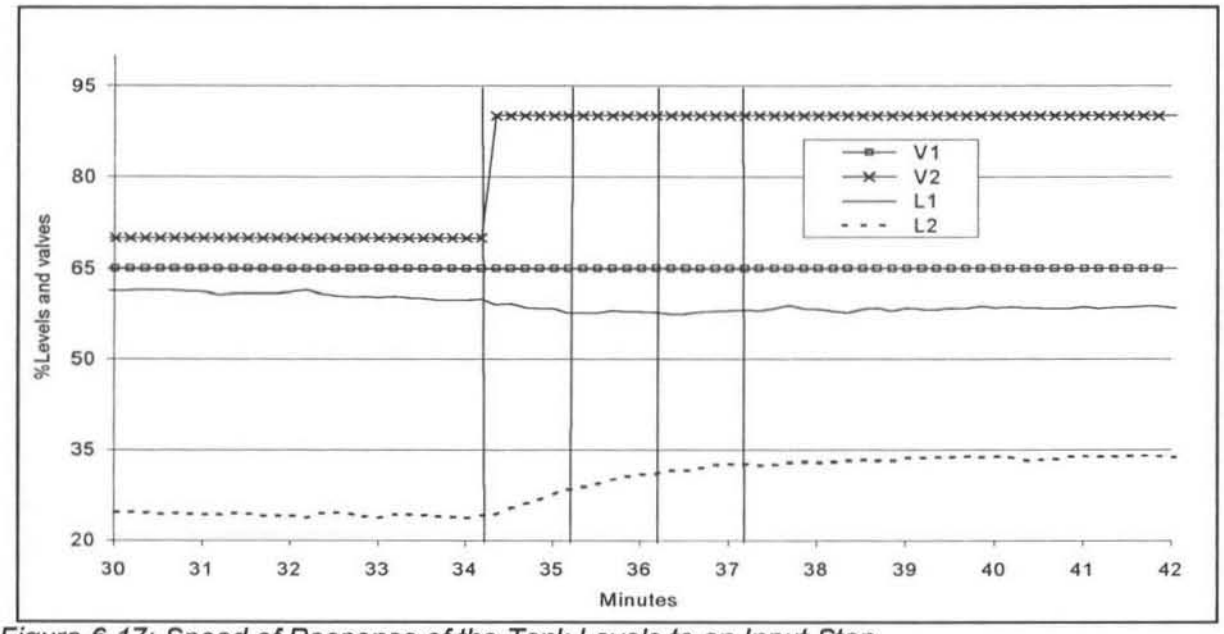


*Figure 6.17: Speed of Response of the Tank Levels to an Input Step*

Several step tests were performed to determine the time constants of the system. An example of these responses is shown in figure 6.17. As can be seen the bulk of the level responses were defined in the first 3 minutes following the step change. Thus for the neural network

models it was decided to use a 1 minute time interval between the consecutive past and present inputs. The desired operating region is shown in figure 6.18. It was decided that this would be the best region for effectively controlling the system.

| | Valve 1 | Valve 2 | Level 1 | Level 2 |
|---|---|---|---|---|
| Operating region | 0 – 100 | 50 – 100 | 25 - 90 | 5 – 50 |

*Figure 6.18: Desired Operating Regions of the Tank System*

Several inverse neural network model structures were experimented with. The model that was found to give the most accurate valve predictions is shown below. This model is relatively small, with only 10 inputs, compared to some of the other models tried. Nine hidden nodes were used. This number was found by a trial and error process of comparing the standard deviations of the training errors using different number of hidden nodes, as is done in section 7.2.2.2.

| Inputs | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Level 1 | | | Level 2. | | | Valve 1 | | Valve 2 | |
| t+1 | t | t-1 | t+1 | t | t-1 | t–1 | t | t-1 | t |
| (SP) | | | (SP) | | | | | | |

| Outputs | |
|---|---|
| Valve 1 | Valve 2 |
| t+1 | t+1 |

*Figure 6.19: Structure of the Inverse Neural Network Model*

### 6.2.3. Results from Training/Testing the Model

As is already known the 1 step ahead comparison of predicted versus actual values is not the most reliable method of verifying a model's accuracy. Figure 6.20 shows these results.

The sensitivity curves are most probably the best indicator of whether an inverse model of a system is suitable to act as a controller or not. They are shown in figures 6.21 and 6.22 below. The responses in these curves show that the model will suggest reasonable valve changes when level setpoints are changes. For example if the setpoint of level1 was increased by 10%, referring to figure 6.21, the model would suggest to the controller to close valve 1 by 30% and close valve 2 by 8%. From past experience with this system it is known that these valve changes will move level 1 in the correct direction. The same applies for level 2 setpoint responses.

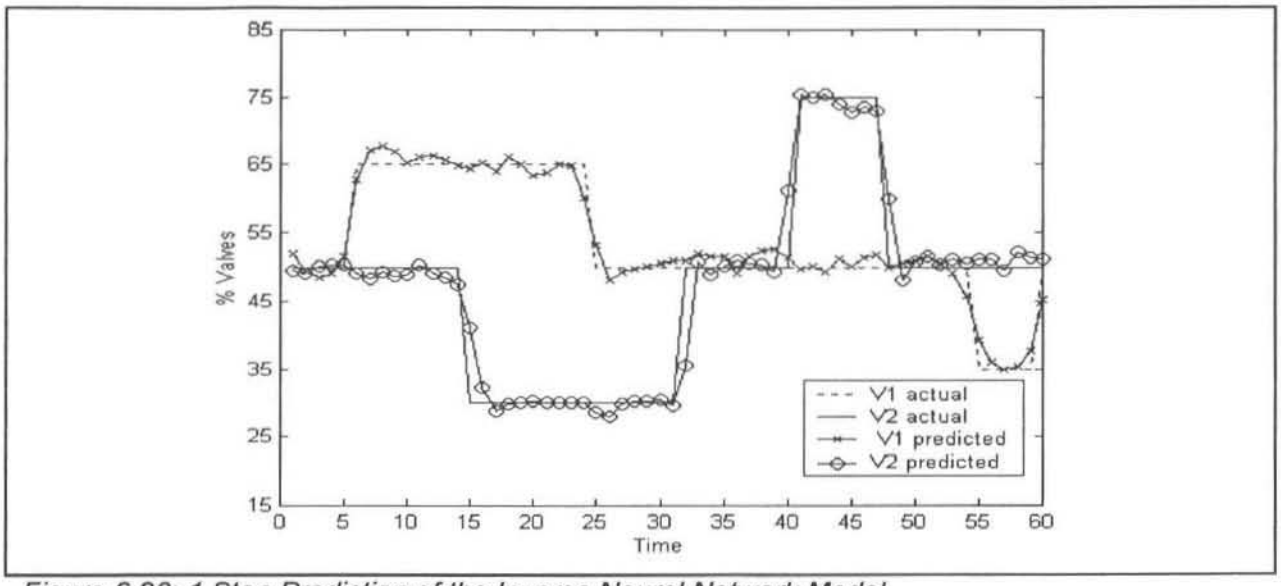


*Figure 6.20: 1 Step Prediction of the Inverse Neural Network Model*

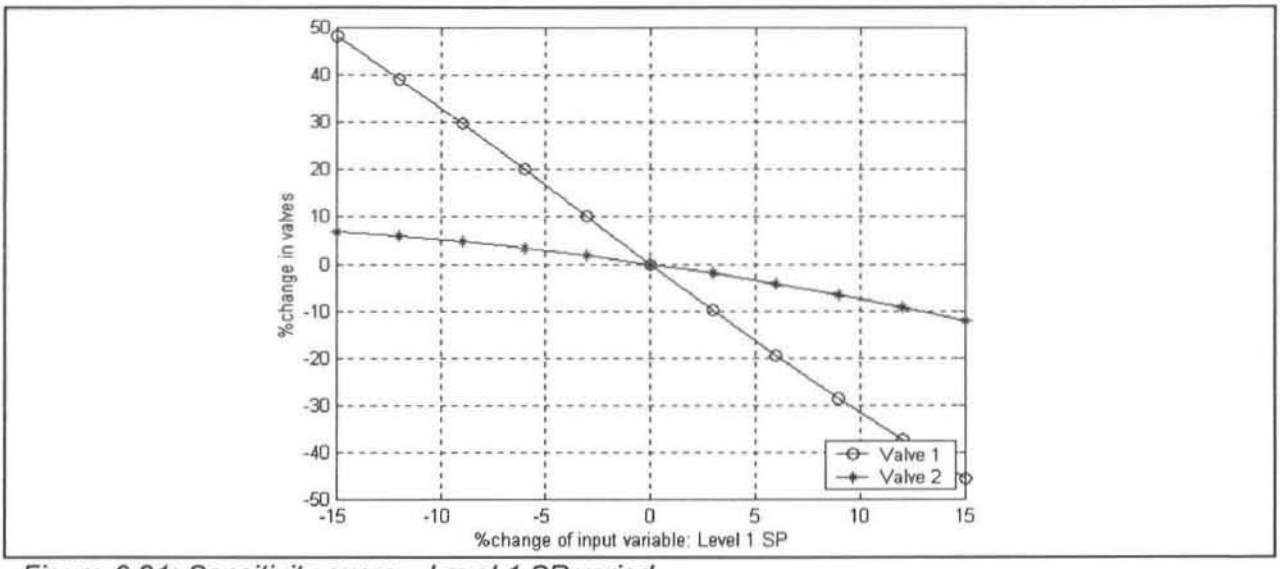


*Figure 6.21: Sensitivity curve – Level 1 SP varied*

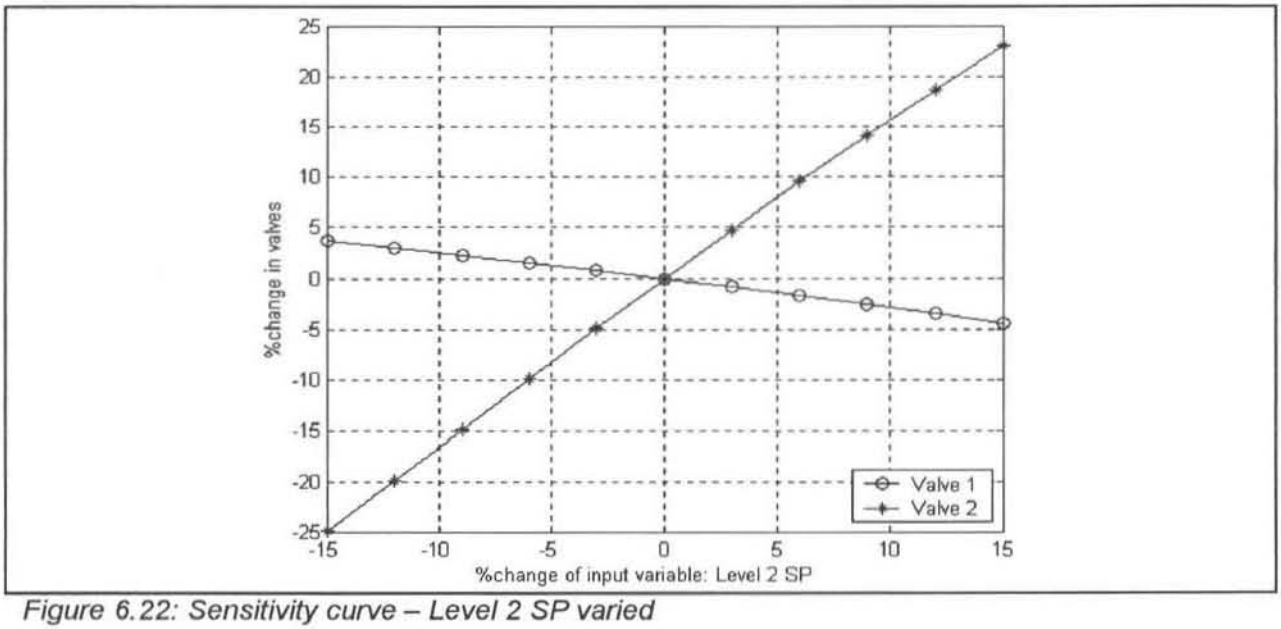


*Figure 6.22: Sensitivity curve – Level 2 SP varied*

### 6.2.4 IIMC Control Results

#### 6.2.4.1 Controller Structure

The multi-step chaining algorithm was not applied to inverse models (see section 4.5.2) which makes these models more difficult to validate for forward models. Going on the information above in section 6.2.3 it appears that a suitable inverse model of the system has been found for control purposes. The IIMC controller is designed to correct the model on every step by using the feedback error term (e), as is shown in figure 5.4 and discussed in section 5.2.3. Since the IIMC controller is suggesting positions for valves 1 and 2, most of the terms in the calculations are vectors. As such the filter settings $\alpha 1$ and $\alpha 2$ are the tuning variables for the corresponding valve position predictions (i.e. $\alpha 1$ is used to tune the controller for valve 1 predictions).

The IIMC controller adjusts the valve positions at 1-minute time intervals. The SCADA software stores and feeds the neural network model inputs, along with the level setpoints for the next time step. The controller model (inverse model No. 1) gives a prediction of the required valve positions, which are then corrected by the feedback error terms (e). These values ($u_{sugg}$) are first clipped to fit within the valve operating range, and then ramp-limited to prevent excessively quick changes. An arbitrary value for these ramp-values can be set, within reason. In most cases the next valve position was restricted to be within 20% of scale of the previous position These modified valve positions are then sent to the valves as the next control action. A minor disadvantage of this "safety" measure is that the original training data were not subject to this arbitrary restriction.

#### 6.2.4.2 Test 1 – No Feedback Error Correction

Refer to figure 5.4 in section 5.2.3, where the structure of the IIMC controller was discussed. If the feedback error term (e) is set to zero manually (by breaking the loop in the calculation algorithm) then only model 1 predictions are used for the control action. This is essentially the same type of controller as shown in figure 5.1. The purpose of this test was to check how effective the controller would be without the feedback error (e) correction.

From figure 6.23 it can be seen that the control of the levels is poor compared to some of the following tests. The controller was given the same level setpoints as in test 2 and it is seen that neither level 1 nor 2 are controlled with any accuracy. This is confirmed by the standard deviation (equation 4.10) results as shown in figure 6.30. The controller is unable to follow the setpoint trajectory because of the model-plant offset.

#### 6.2.4.3 Test 2 - No Filtering of Feedback Error

In this test the filter parameters $\alpha 1$ and $\alpha 2$ were set to 1. Recall the equation for a single exponential filter.

$$\bar{y}_t = (1 - \alpha)\,\bar{y}_{t-1} + \alpha\, y_t \qquad \textit{Equation 6.5}$$

$\bar{y}$ -> filtered variable

$y$ -> raw variable.

$\alpha$ -> filter constant (always less than or equal to 1)

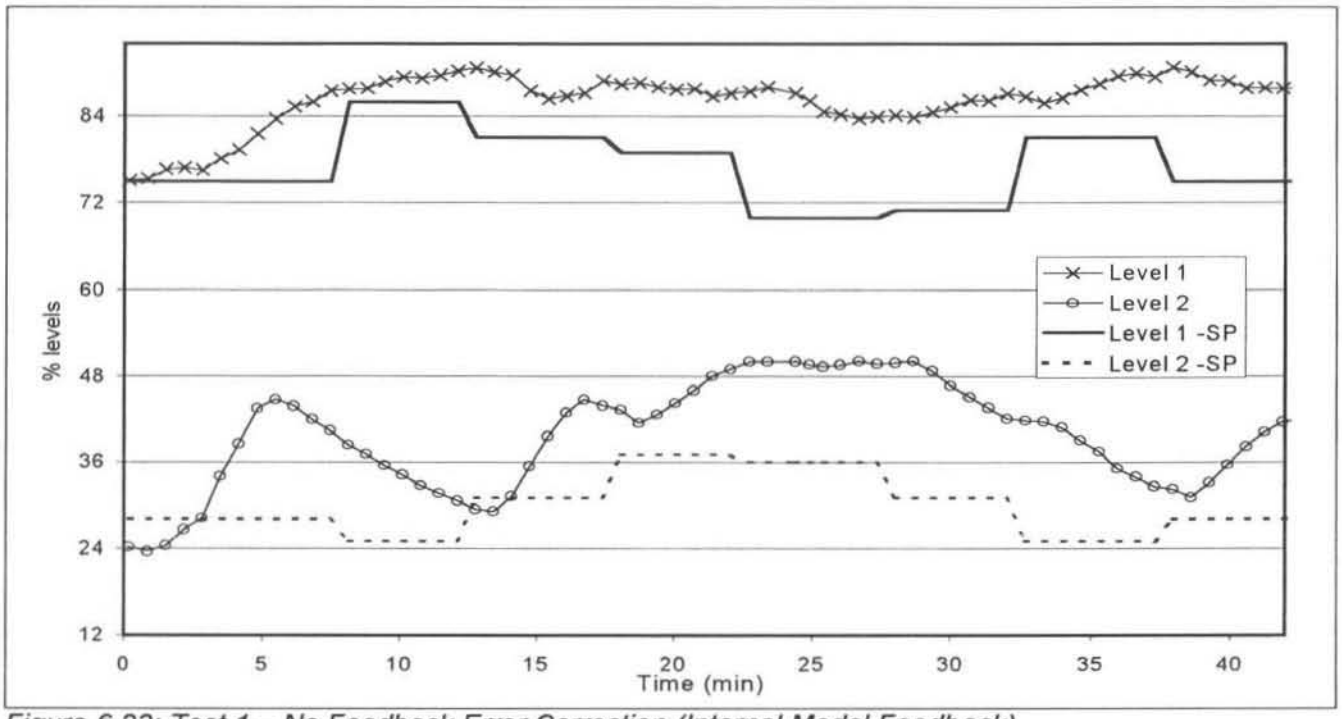


*Figure 6.23: Test 1 – No Feedback Error Correction (Internal Model Feedback)*

In this test there was no filtering ($\alpha_1 = \alpha_2 = 1$) of the feedback error (e). The actual values of the error vector were used and there was no smoothing of these values as normally occurs when using a single exponential filter. The results are shown in figures 6.24 and 6.25. As can be seen the controller does a fairly good job keeping the levels at their setpoints. The control of Level 1 appears better than that of Level 2. The standard deviation measurements for this test verify this. However it must be remembered that the level setpoint changes in this test were not as large as in the next three tests, hence the better standard deviation results.

Valve 1 movements are fairly smooth and this maybe suggests that only a small amount of filtering of error 1 ($\alpha_1$) may be required. Level 2 fluctuates erratically around the setpoint as a result of valve 2 moving wildly (i.e. opening and closing too much). Hence significant filtering of feedback error 2 ($\alpha_2$) was required.

#### 6.2.4.4 Test 3 – Best Filter Settings

In this test the optimal filter settings were used ($\alpha_1 = 0.75$ and $\alpha_2 = 0.10$). These were found by using a trial and error process of comparing the standard deviations of the controlled level and setpoints. The level setpoints for this test and the two DMC tests were made slightly more extreme to see how the controllers would handle larger setpoint changes.

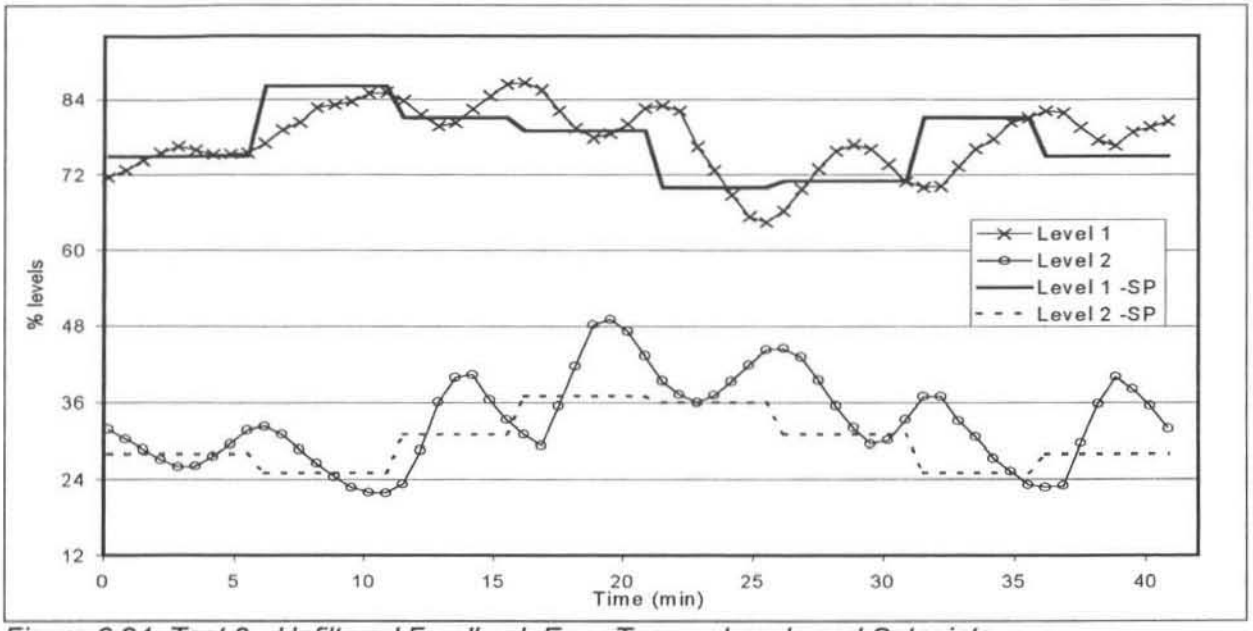


*Figure 6.24: Test 2 - Unfiltered Feedback Error Term – Levels and Setpoints*

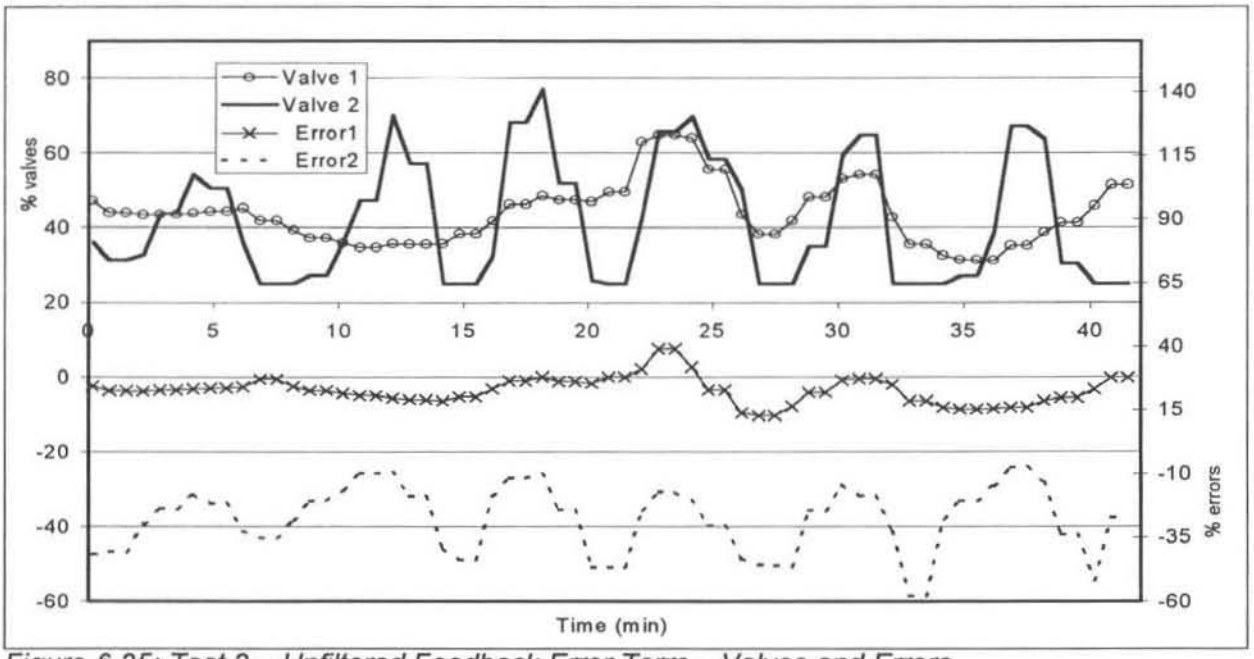


*Figure 6.25: Test 2 – Unfiltered Feedback Error Term – Valves and Errors*

The results are shown in figures 6.26 and 6.27. The controller was able to bring the levels to setpoint even on the largest setpoint steps within a reasonable amount of time. The valve movements were very smooth and organised with no wild jumping around or unnecessary movement. In figure 6.27 the filtered feedback errors (e) are shown. What is very evident is that error 2 ($\alpha_2$) is filtered more strongly than error 1 ($\alpha_1$), since its alpha value is significantly less than for error 1 (see section 6.2.4.3).

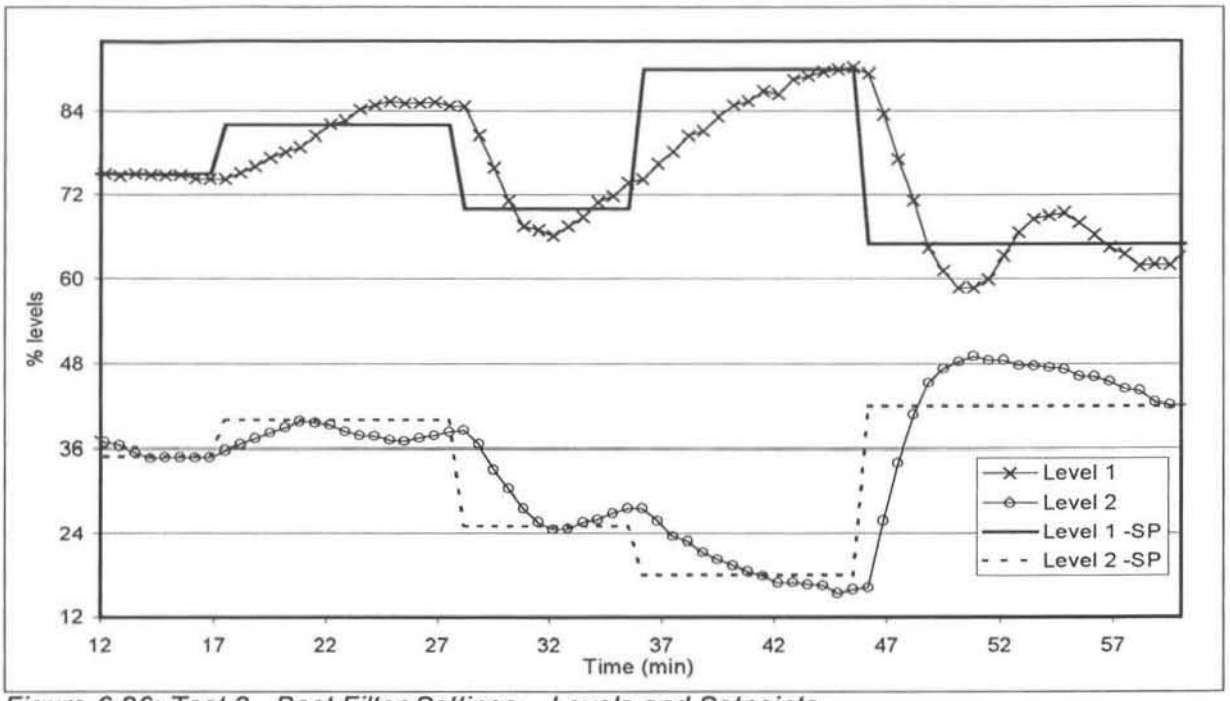


*Figure 6.26: Test 3 - Best Filter Settings – Levels and Setpoints*

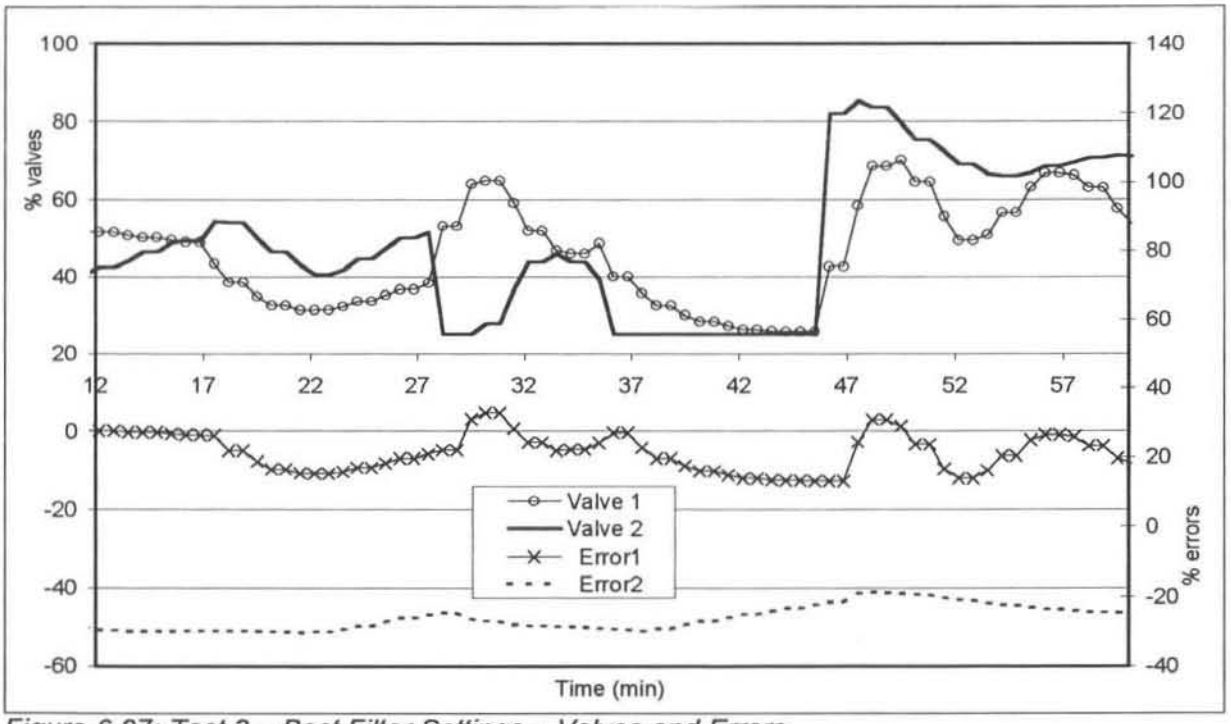


*Figure 6.27: Test 3 – Best Filter Settings – Valves and Errors*

In figure 6.30 it can be seen that the results from this controller and the 30-second interval DMC controller are quite similar. This is the best controller configuration tested in this case study. If more time was available the filter parameters could be fine-tuned, however the benefit gained may not be substantial. Improvements to this artificial neural network IIMC controller may possibly be achieved by a more suitable choice of the time step between control actions.

### 6.2.4.5 Tests 4 and 5 - DMC Results

The difference between two DMC (Dynamic Matrix Control) controllers and the neural network IIMC controller were compared. These optimal DMC controllers were found in a similar study performed on the tank system [Guaimba, 2000]. The difference between the DMC controllers was the time between control actions. The one controller updated the control action every 30 seconds while the other one worked off a 1-minute sample time cycle, the same as the neural network controller. In fact the normal sampling time for these DMC controllers is shorter than 30 seconds and is more like 5 to 10 second for this tank system.

The reason that the larger intervals were used with the DMC controllers was to test their performance when using a similar sampling interval to the neural network controller. These DMC controllers were given the same level setpoints as the neural network controller in Test 3.

The results of the 60-second DMC controller are shown in figure 6.28. The levels appear to follow their respective setpoints closely, and from the standard deviations in figure 6.30 it can be seen that the IIMC neural network controller performs very similarly. In figure 6.29 the results of the 30-second sample time DMC controller are shown. This controller is visibly better than it's 60 second counterpart. The performance is very similar to the IIMC neural network controller, based on the standard deviations in figure 6.30, however from visual inspection the DMC looks like a better controller. The responses to level setpoint changes are smooth and there is very little overshoot of the setpoint.

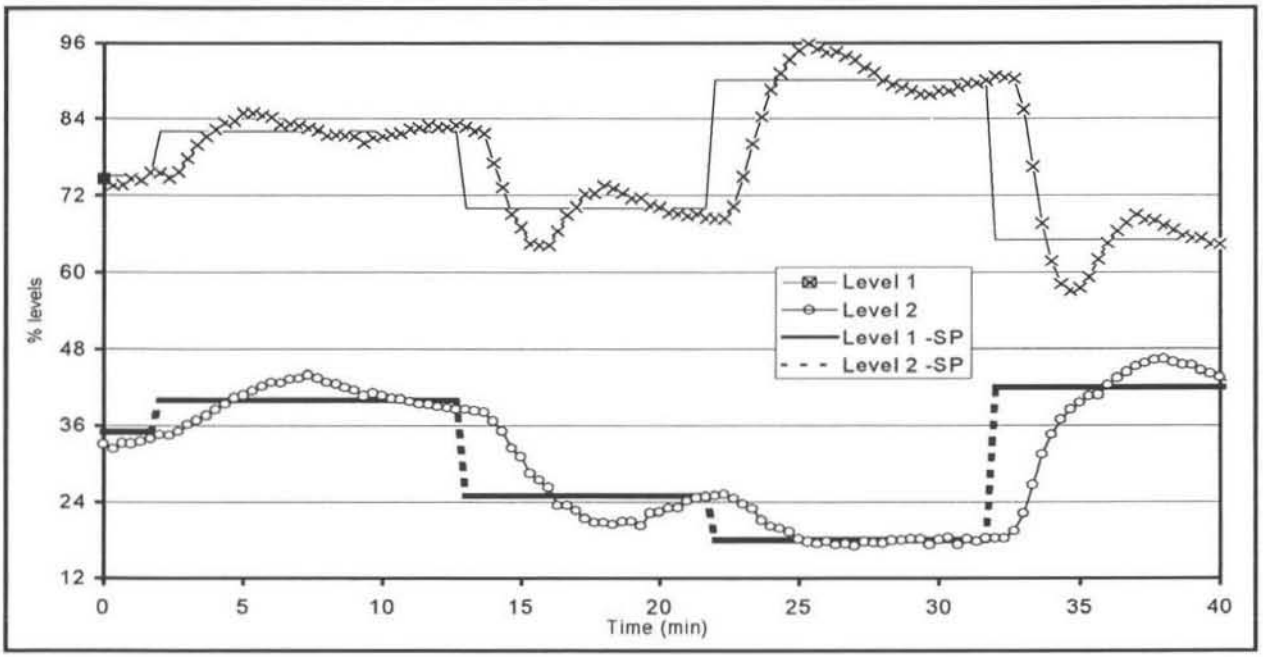


*Figure 6.28: Test 4 - DMC (60-second sample time)– Levels and Setpoints*

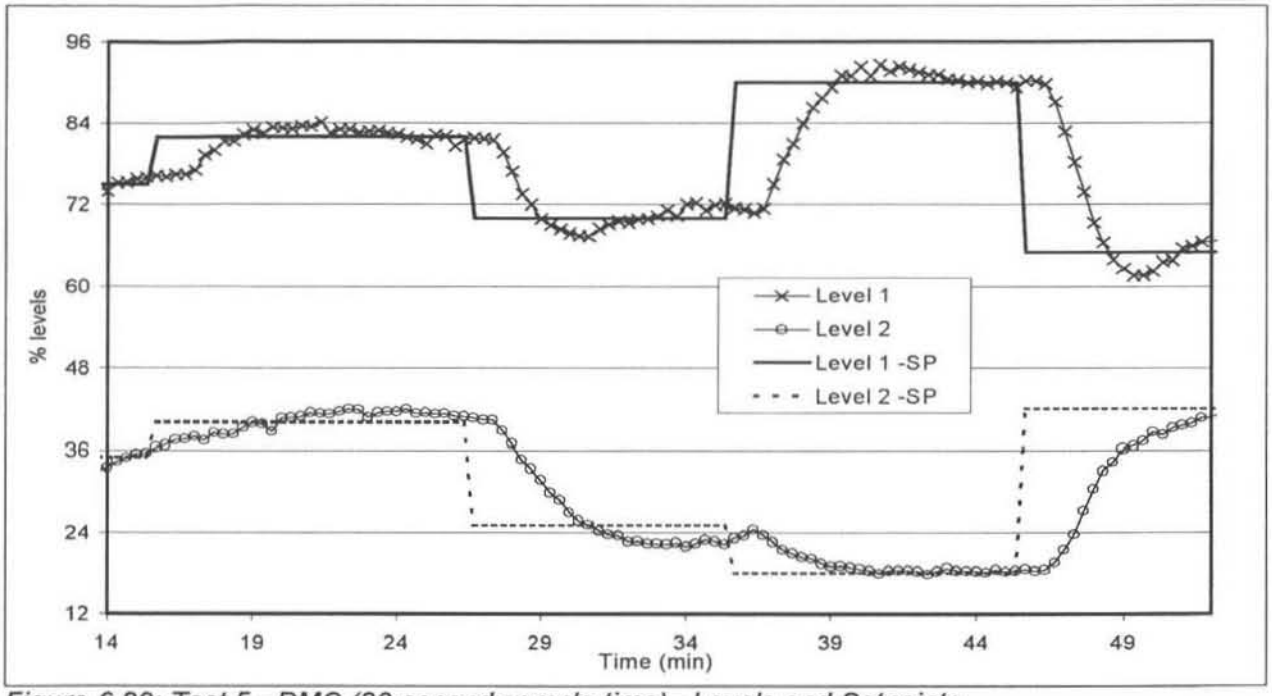


*Figure 6.29: Test 5 - DMC (30-second sample time)– Levels and Setpoints*

| | Standard deviation | | AVG S.D. |
|---|---|---|---|
| | Level 1 | Level 2 | |
| TEST 3<br>ANN IIMC - Best filter settings | 16.50 | 13.87 | **15.18** |
| | | | |
| TEST 5<br>30 sec DMC | 21.29 | 15.82 | **18.56** |
| | | | |
| TEST 4<br>60 sec DMC | 19.83 | 44.16 | **31.99** |
| | | | |
| TEST 2<br>ANN IIMC - No error filtering | 11.71 | 19.34 | **15.53** |
| | | | |
| TEST 1<br>ANN IIMC - No feedback error correction | 54.93 | 66.55 | **60.74** |

*Figure 6.30: Table of Standard Deviations for Level Control from IMC Tests*

#### 6.2.4.6 Discussion of Tests 1 to 5

The standard deviations between the respective controlled levels and their setpoints are summarised in figure 6.30. The individual level standard deviation and average level standard deviation are presented. The 'optimised' IIMC controller has lower individual and average standard deviations than both of the DMC controller tests. However the results are only numbers indicating the 'average' values of the difference between the two measured values over the given time period. It gives no further information about the differences between the qualities of the two controllers. One would need to look at factors such as speed of response

and whether the controlled variables oscillate around the setpoint. By visually comparing the respective graphs (figures 6.26, 6.28 7 6.29) it is possible to get a feel for some of these factors. Some may feel that these show that the speed of response of the DMC controllers is faster and the control tighter and that more tests should have been done to quantify these variables. However the aim of this case study was simply to see whether the IIMC controller would perform comparably with a DMC controller, which it did. Although the tests performed here are by no means conclusive or exhaustive they do show the potential of using this type of controller.

There are a few other interesting points worth taking note of in figure 6.30. Note how the IIMC controller in Tests 1 & 2 seemed to better control Level 1 to its setpoint (lower standard deviation), but in Test 3 the reverse in observed. By changing the filter settings on the IIMC controller it was possible to improve the control of both levels, to the extent that Level 2 had a lower standard deviation.

It is also interesting to note how increasing the resolution of the DMC controller from 60 to 30 seconds affected the control of the levels. At 60 seconds resolution (Test 4) the control of Level 2 was fairly poor (compared to Test 5), while Level 1 was reasonably good (compared to Test 3), which resulted in an average standard deviation of about 30 units. In Test 5 the control of Level 2 improved drastically while Level 1 results deteriorated slightly. However, because Level 2 results improved so much the average standard deviation decreased to 18.5 units. By visually examining the graphs (figure 6.28 & 6.29) it does seem to confirm that the DMC controller in Test 5 was the better of the two.

To summarise the results of the IIMC artificial neural network controller in this case study the following points need to be mentioned.

- The advantage of using the two inverse models is that only one model of the system needs to be trained. In the case of traditional IMC control two models of the system are required, one forward and one inverse model.
- A pleasing aspect of this case study was that the artificial neural network models were able to learn the system's behaviour despite the process and signal noise on the variables. This indicates that the training process was able to distinguish the relevant data and ignore the random noise. That was one of the reasons why artificial neural networks were chosen for modelling the polymer plant.
- The effect of using the feedback error (e) to correct the inverse model predictions greatly improved the controller capabilities (compare Tests 1 & 3).
- The issue of tuning the filter parameters is an important one. The benefits of properly tuning the filter are evident in Tests 2 and 3. Reducing the sampling interval could be an improvement to this IIMC controller.

# CHAPTER 7

# 7. POLYMER PLANT MODELLING

## *7.1 Introduction*

One of the overall objectives of this research was to obtain an inverse neural network model of the reactor. This model was to relate the main gas compositions in the cycle gas loop to the reactor feed flows. The feed flow to the reactor as mentioned previously consists of ethylene monomer, a co-monomer (hexene or butene) and other smaller components including hydrogen, nitrogen and iso-pentane.

The polymer plant reactor is far more complicated that the 2 input-2 output tank system studied in the case studies presented. The number of process variables in the reactor and cycle gas loop is very large, probably running into the hundreds. These variables include just about every imaginable measurement of the state of the reactor, cycle gas stream and product polymer.

Before artificial neural network models of the system could be trained the plant data needed to be analysed and graphed in order to obtain some idea of the structure that would be required for the models. Some of the key issues that arose from this analysis were:

- The number of relevant variables was narrowed down to about 20, nearly all relating to the state of the reactor. These were the main component flows in and out of the reactor, its operating conditions and the properties of the polymer product, and some of the gas compositions in the recycle loop. These variables are shown in figures 7.1 and 7.2 and figures A.1, A.2 and A.3.
- Data sets that showed potential for training neural network models were noted. These data sets preferably included dynamic changes in the key variables and were over a fairly long time period.
- The approximate times for the gas composition ratios to change after a flow ratio change were noted. This was important for determining the structure of the neural network models (i.e. how many past variables to include in the dynamic model inputs).
- From the rate of these changes it was also possible to determine an interval for logging the data. It is important to obtain an interval that is short enough to properly define changes in the response curves, but not too short so as to cause unnecessary calculations in the neural network models.

- The different operating regions of the various polymer grades were identified. It was decided that different models were needed for grades that were produced with greatly differing operating conditions.
- Sets of maximum and minimum values for the chosen variables are important for scaling during training and use of neural network models (see section 4.2.4). These values were obtained for all grades under consideration from the plants operating conditions. These chosen minima and maxima cannot be presented for confidentiality reasons.

### 7.1.1 Training and Testing Data

Many data sets were investigated and a few of these were selected for training and testing the neural network models. These sets were chosen on the basis of a few simple criteria:

- The grades being produced were similar and so the plant operating conditions were similar.
- The data contained some dynamic variations in the important variables. In most cases this included a small grade transition or an external load disturbance.

The plant data was scaled using the average operating conditions. As mentioned previously these values are not shown in order to protect the interests of the company operating the reactor. The data was scaled to values approximately between 0.1 and 0.9, however values outside this range have not been clipped. In the open loop on-line implementation of the controller all values are clipped between the range 0.01 and 0.99. The plots have been shifted on the vertical axis so that it is possible to distinguish the different trend lines.

These graphs are shown in figures 7.1 and 7.2 and figures A.1 and A.2 (in the Appendix). The data in figures 7.1 and 7.2 was used to train most of the neural network models used in this work. The models were then tested using the data in figures A.1 and A.2. The models trained were found to be effective on the grades such HFM2010, HFM2280, HFM2410 etc. Other models for different grades were developed but the results for these are not presented here.

Figures 7.1 and A.1 show the flow rates of the fresh feeds to the cycle loop and the gas compositions in the reactor. Figures 7.2 and A.2 show the flow and composition ratios, and the general plant operating conditions.

### 7.1.2 Determining the Dead Time of the System

Before the modelling of the reactor could begin it was important to determine the dead time and dynamic response time of the system. It was already known that the dead time of the gas analysers was approximately 3 minutes.

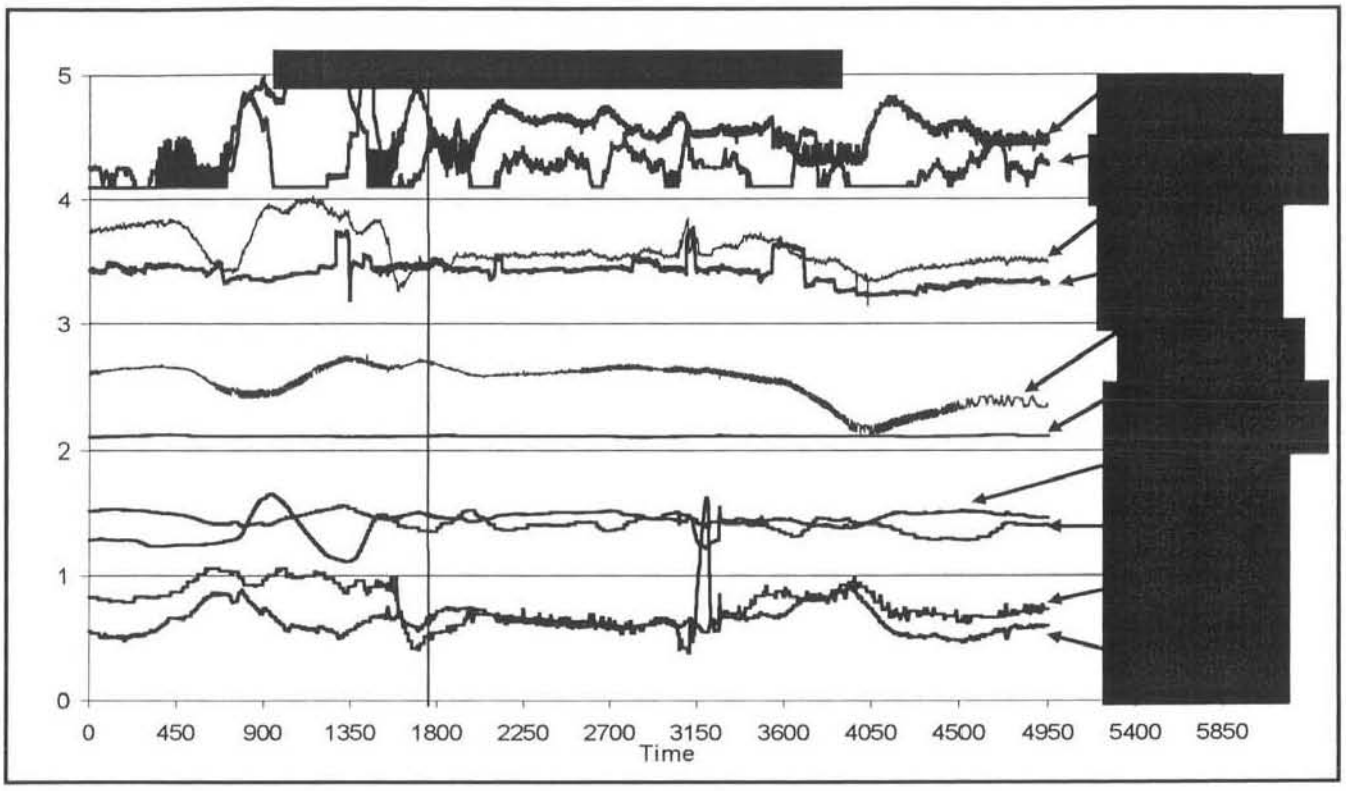


*Figure 7.1: Scaled Plant Data (A) used for Training Neural Network Models*

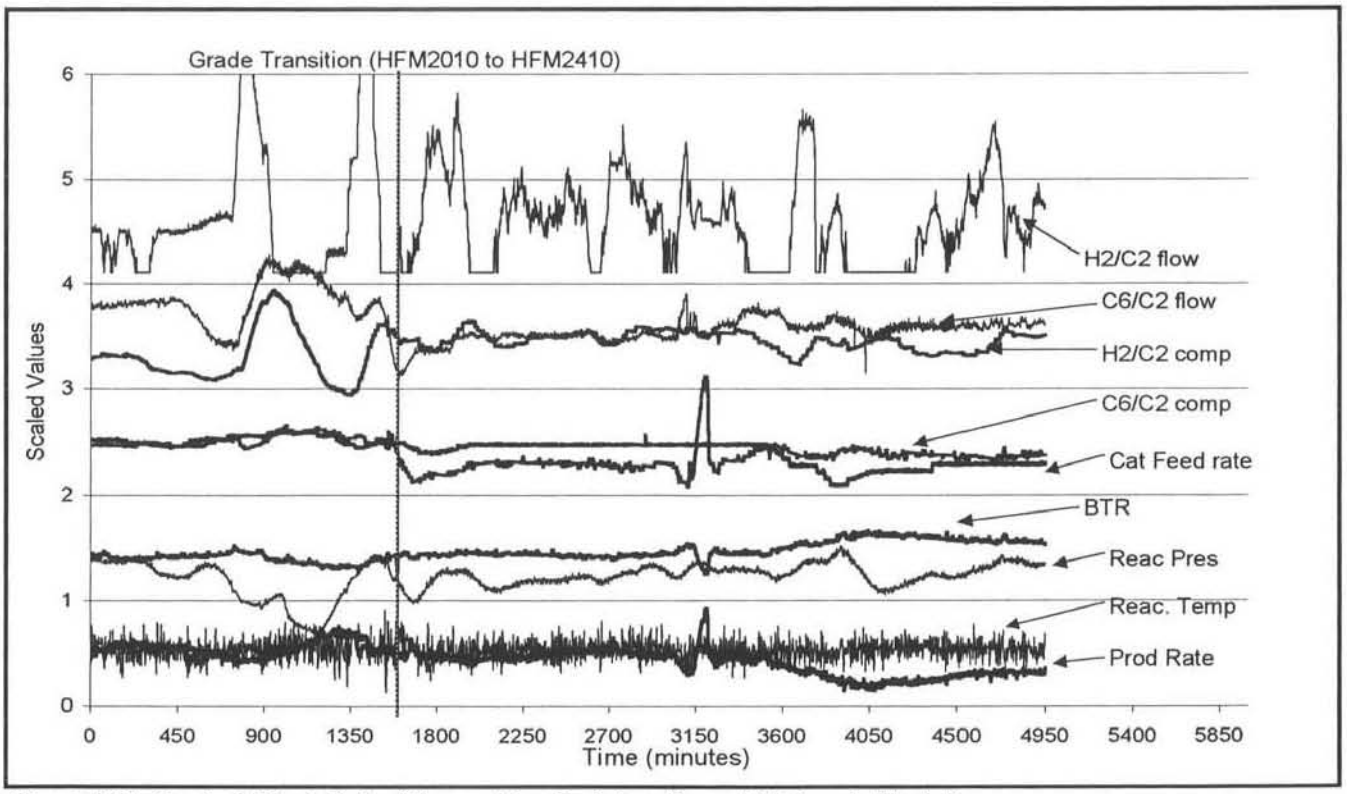


*Figure 7.2: Scaled Plant Data (B) used for Training Neural Network Models*

The other important time lag was between the feed flows into the reactor and the resultant gas compositions in the reactor. It was known that this was considerably longer than 3 minutes from operator knowledge. Although there is a portion of dead-time lag, this is largely a first-order

response resulting from the large volume of gas present. Hence the flow changes take a long time to affect the compositions in the reactor.

The easiest way of checking this lag time was by visually examining trend curves that include dynamic changes of the composition and flow rate variables under consideration. Initially curves of the individual component compositions and flow rates were analysed but it was evident that no clear result could be determined from this method, as can be seen in Figure 7.3. There seems to be no clear trend to the response of the compositions to the changes in the respective flow rates.

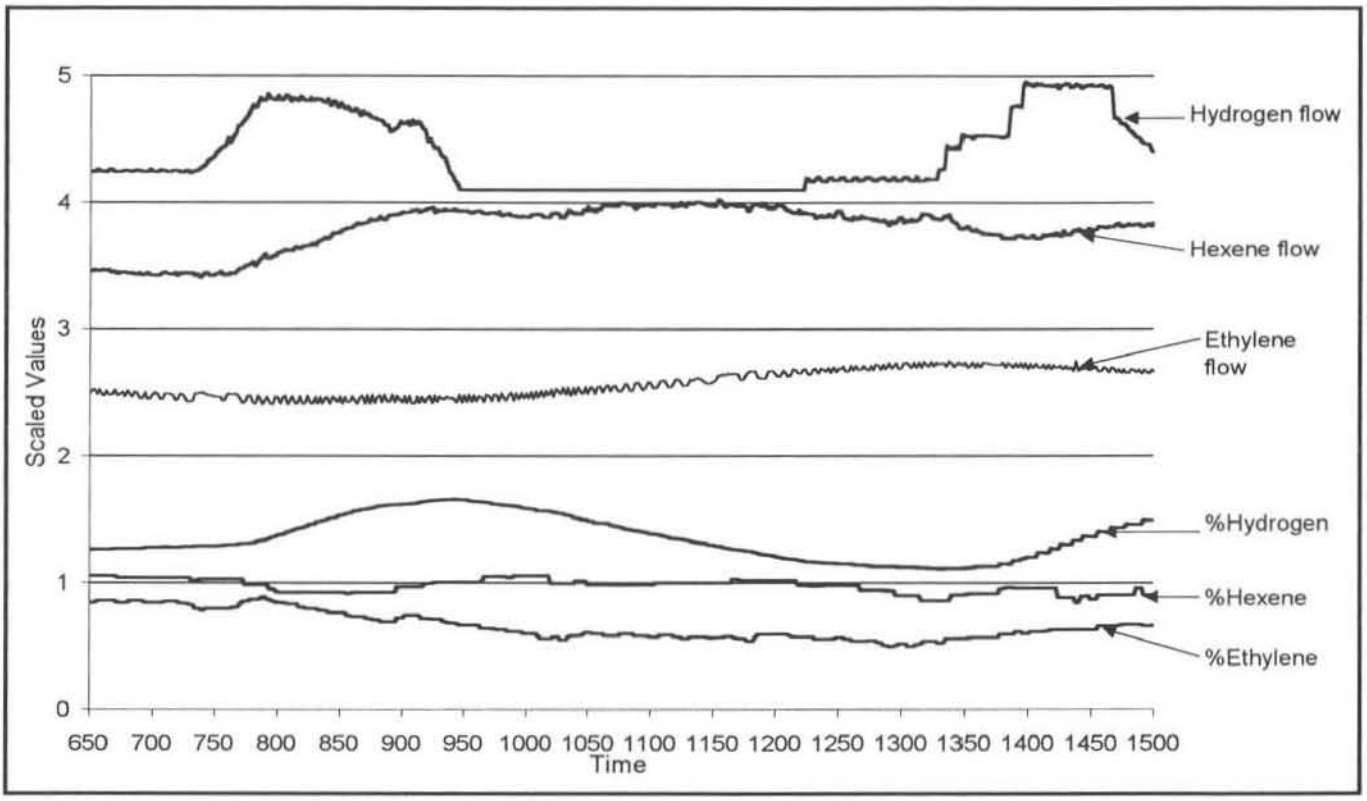


*Figure 7.3: Compositions and Flow Rates in the Reactor / Cycle Gas Loop*

However when the ratios of the compositions and flow rates were plotted, on the same data set as in figure 7.3 the time lag was immediately apparent. The response of the hexene ethylene composition ratio (solid black line) can be seen at about 800 minutes, approximately 50 minutes after the flow ratio was changed. Again when the flow ratio changes after 1375 minutes the composition ratio responds about 55 minutes later only.

The same response time is observed in the hydrogen ethylene ratios (figure 7.5). When the flow ratio increases at 730 minutes, it takes the composition ratio about 50 minutes to also begin changing. Again at 1320 minutes when the flow ratio increases it takes the composition about 45 minutes to respond. The lag time between the flow ratio peak and the composition peak is slightly longer. This is in the order of 100 minutes.

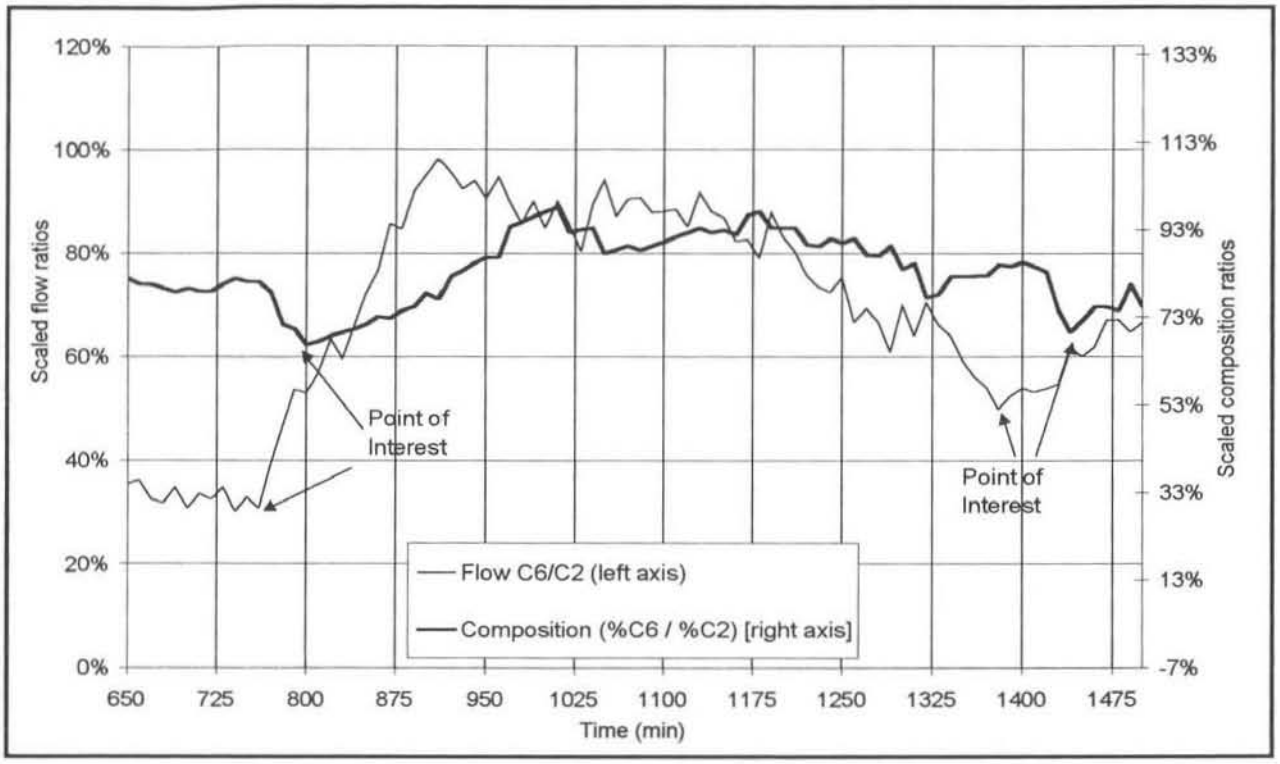


*Figure 7.4: Hexene Compositions and Flow Ratio in the Reactor / Cycle Gas Loop*

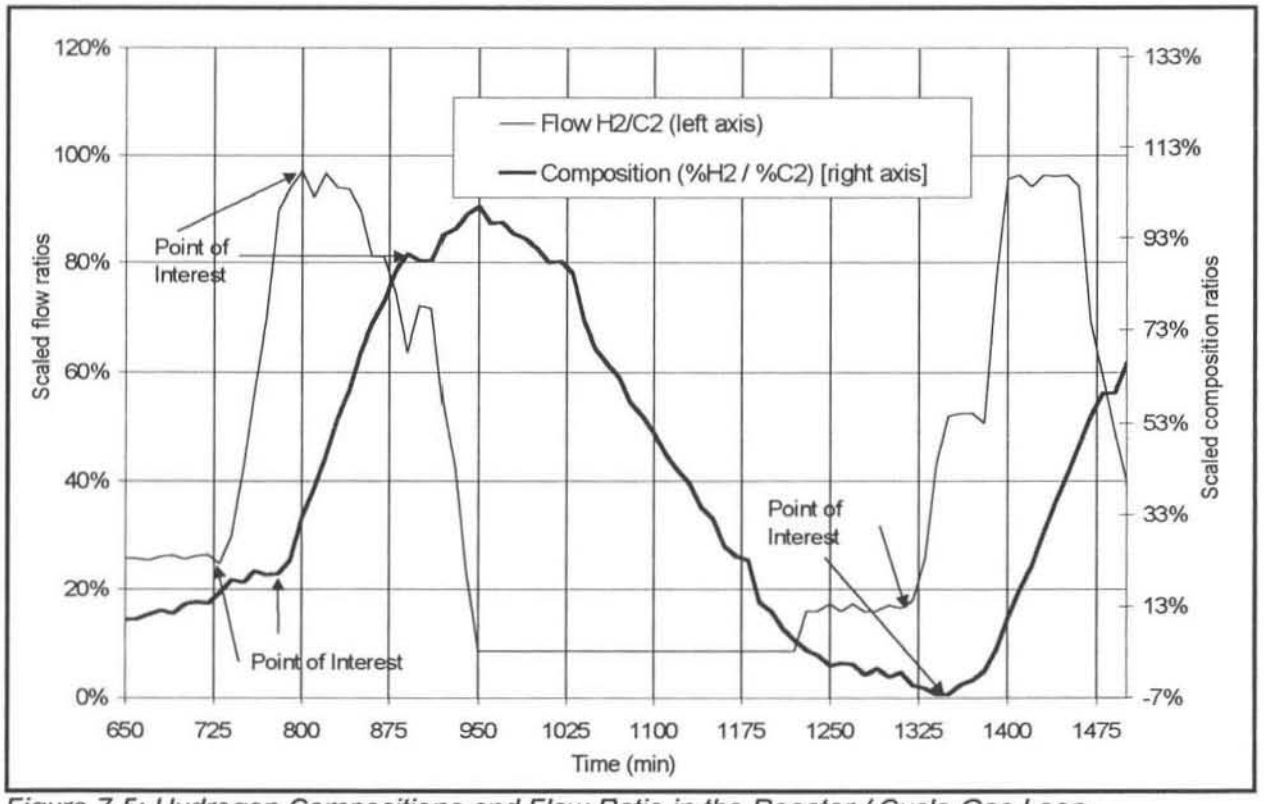


*Figure 7.5: Hydrogen Compositions and Flow Ratio in the Reactor / Cycle Gas Loop*

From these results it was assumed that the total lag in the system, including the gas analyser delay, was between 50 and 60 minutes. It was vital that this rather long delay was discovered so that the time-invariant dynamic neural network models could be designed to take it into account.

## 7.2 *Inverse Process Neural Network Models*

As discussed in section 5.2 inverse plant models can be used as plant controllers or in controller structures. In the case of the polymer reactor an inverse model was required that would predict the required flow rates of ethylene, co-monomer and hydrogen to the reactor to achieve a certain gas composition.

As mentioned in section 5.2.3 the IIMC structure uses two identical inverse process models. The inputs to these models are essentially the same, with the main differences being for the hydrogen and hexene composition ratio values. In the Controller Model (figure 5.4) the setpoints of these values are used while in the Predictor Model the process variables are used. Also, the other process variables used as inputs to the Predictor model are previous plant values. These differences are highlighted in Model 1 in section 7.2.1 below.

Several different modelling approaches were attempted as discussed in sections 4.4. The two main options available were to use time-invariant algebraic (time adjusted) models or the dynamic type of model. Any dynamic model used would need to take into account the long time lag.

Four of the most successful inverse neural network models that were developed are presented here although many other sizes and structures of models were tried. The first inverse models developed were very large, including all of the variables listed in figure A.2 as inputs. The inputs also included 3 past values of each variable sampled at 4-minute intervals. The number of inputs to these models was in the region of 40 to 70. These models were unsuccessful because of their large size and bulkiness and since they did not take into account the much longer time lag of the system. It was soon discovered that the smaller dynamic models could predict the behaviour of the system just as well as the larger models. Models 3 and 4 are examples of how these smaller dynamic models were used effectively.

Most of the models developed used the training technique as described in sections 4.4.3 and 4.4.4. This allows them to take into account the lags in the system. The models were trained by taking the average of 4 points at 30-minute intervals (T, T+30, T+60, T+90) as the value of the setpoint, for subsequent input of the composition setpoints in controller mode.

### 7.2.1 Inverse Model 1 (Time-Invariant Algebraic Model)

#### 7.2.1.1 Structure of Model 1

The structure of the model is shown in figure 7.6. The neural network has 10 inputs and 5 outputs. This model was developed to use the actual component compositions and flow rates,

and not the ratios. The model predicts the flow rates of the 5 main inflows to the reactor. The nitrogen and TEAL flow rates were included at this stage. The nitrogen and TEAL flows were subsequently omitted to allow concentration on the hydrogen and co-monomer additions. The prediction of the hydrogen flow rate proves more difficult than the rest of the flows and is discussed later.

The model inputs 2 and 3 are setpoints of co-monomer (hexene) and Hydrogen composition in the reactor. These come from the proprietary Poly Server Model. The values used for these during training were the average of 4 future values, as discussed in section 4.4 and 7.2. The current ethylene and nitrogen compositions are taken from plant measurements. All of the other plant measurements are also the current values.

Model Inputs for the Controller Model (See figure 5.4)

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Ethylene % | Hexene % | Hydrogen % | Nitrogen % | Prod Rate | Reac Temp |
| t | t+1 (SP) | t+1 (SP) | t | t | t |

| 7 | 8 | 9 | 10 |
|---|---|---|---|
| Reac Pres | B.T.R. | Vent MR | Tot. Cat Fdr |
| t | T | t | t |

Model Outputs the Controller Model

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Ethylene Flow | Hexene Flow | Hydrogen Flow | Nitrogen Flow | TEAL Flow |
| t+1 | t+1 | t+1 | t+1 | t+1 |

Model Inputs for the Predictor Model (See figure 5.4)

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Ethylene % | Hexene % | Hydrogen % | Nitrogen % | Prod Rate | Reac Temp |
| t-1 | t | t-1 | t | t-1 | t-1 |

| 7 | 8 | 9 | 10 |
|---|---|---|---|
| Reac Pres | B.T.R. | Vent MR | Tot. Cat Fdr |
| t-1 | t-1 | t-1 | t-1 |

Model Outputs the Predictor Model

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Ethylene Flow | Hexene Flow | Hydrogen Flow | Nitrogen Flow | TEAL Flow |
| t | t | t | t | t |

*Figure 7.6: Structure of Inverse Model 1 (Time-invariant algebraic Model)*

From training, the optimum number of hidden nodes was found to be 13. This was done by comparing the standard deviations of the predicted and actual values. The values shown in figure 7.7 are the error calculated from the normalised or scaled outputs of the neural network model.

#### 7.2.1.2 Results from Model 1

##### 7.2.1.2.1 Training and Testing Standard Deviations

As can be seen from the standard deviations, the hydrogen flow is the most inaccurate prediction. It was concluded that it would be very difficult to model the reactor using the absolute hydrogen flow rate. This is because the hydrogen flow is not a constant flow but rather an on-off flow, as can be seen in figure 7.1. It is sometimes added in batches and between these batches the flow is zero. The neural network model cannot predict this type of value and so it tends to give an average value between the on (high flow) and off (low flow value) positions.

Individual Component Flow Rates (Training)

| | Ethylene Flow | Hexene Flow | Hydrogen Flow | Nitrogen Flow | TEAL Flow |
|---|---|---|---|---|---|
| SD (RMS) | 0.05342 | 0.07423 | 0.12207 | 0.08892 | 0.07608 |

Overall results

| | Training | Testing |
|---|---|---|
| SD (RMS) | 0.0822 | 0.0896 |

*Figure 7.7: Results from Training and Testing Model 1*

The ratio of hydrogen to ethylene flow also exhibits this on-off behaviour. However it was decided to use the ratios of components in future models as the relationships in the data seemed to be more apparent and it was hoped that the neural network models would recognise these patterns more accurately.

##### 7.2.1.2.2 Sensitivity Analysis

As mentioned in section 4.5.3 testing the sensitivity of time-invariant algebraic models is a useful indication of whether the neural network models have learned the behaviour of the system. Shown below are some examples of these curves. Only the ethylene, hexene and hydrogen flow rates were considered of interest here.

Figure 7.8 shows how the three flow rates react when the hexene setpoint (model input No. 2) is varied. The results make sense. A striking feature of the curves is the non-linearity of the relationships between the variables. If the setpoint is lowered by 10%, the hexene flow rate decreases while the other two flows increase. This should result in a lower hexene composition

in the reactor. Similarly if a higher composition of hexene is required if the hexene flow rate increases while the other two decrease.

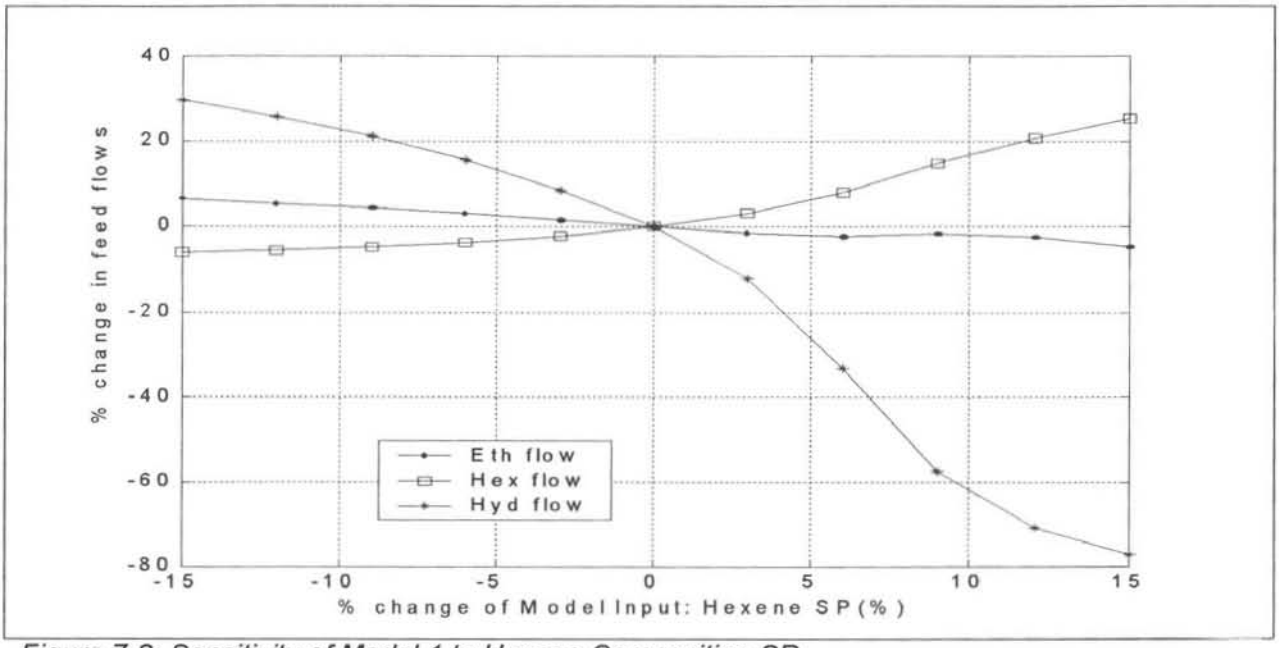


*Figure 7.8: Sensitivity of Model 1 to Hexene Composition SP*

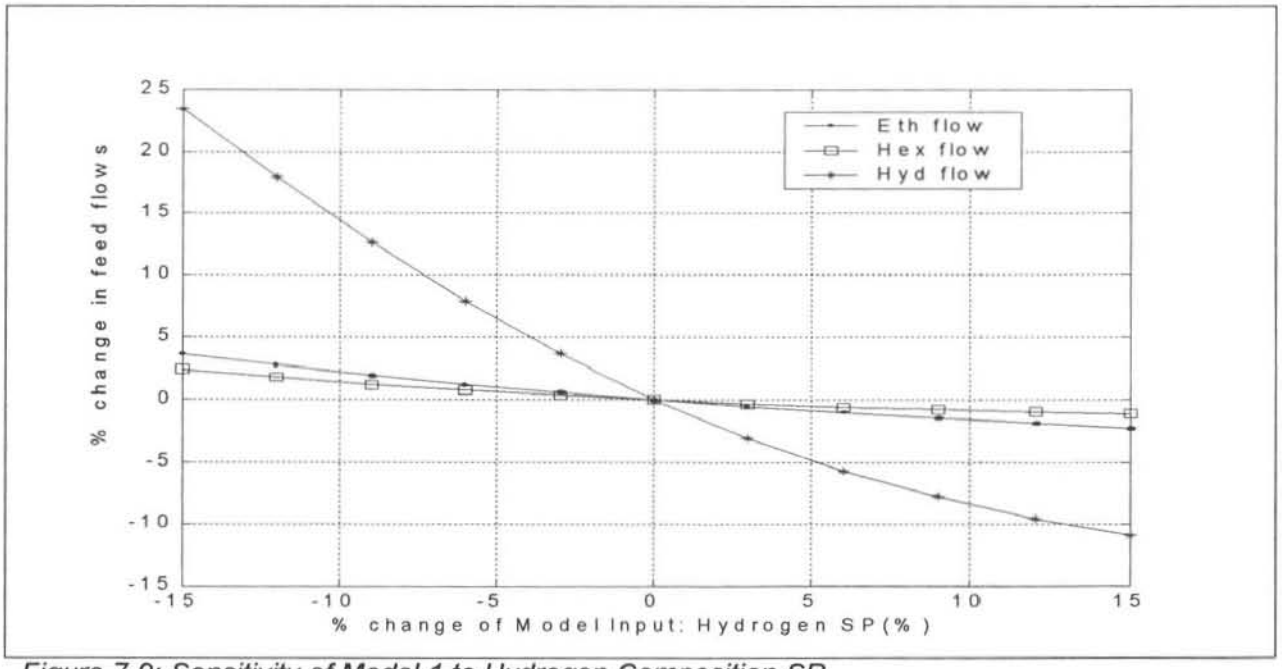


*Figure 7.9: Sensitivity of Model 1 to Hydrogen Composition SP*

Figure 7.9 shows how the three flow rates react when the hydrogen setpoint (model input No. 3) is varied. The responses here are different to the hexene. A decreased hydrogen setpoint will result in an increase in the hydrogen flow rate. At first this doesn't make sense, however the large increase in the ethylene flow rates means that the relative amount of hydrogen will decrease. These graphs clearly indicate the advantages of considering the relationships in terms of the ratios of the variables, since they are so much simpler to understand. This is confirmed in the following sections.

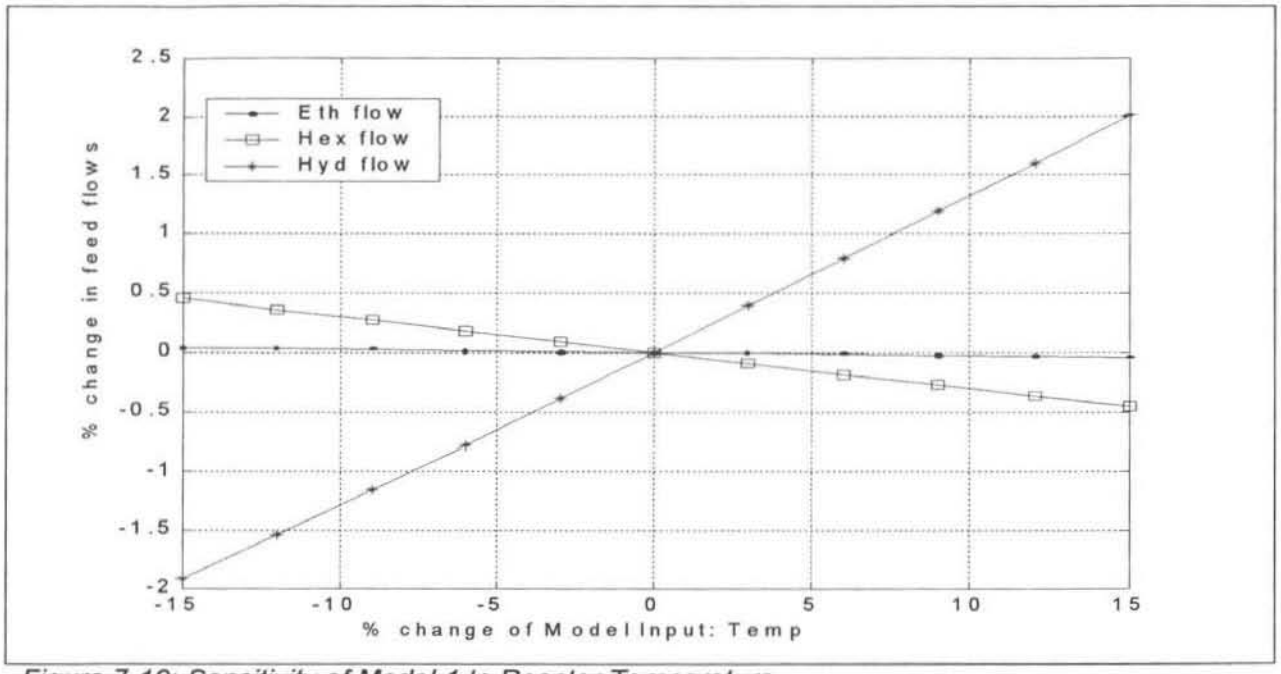


*Figure 7.10: Sensitivity of Model 1 to Reactor Temperature*

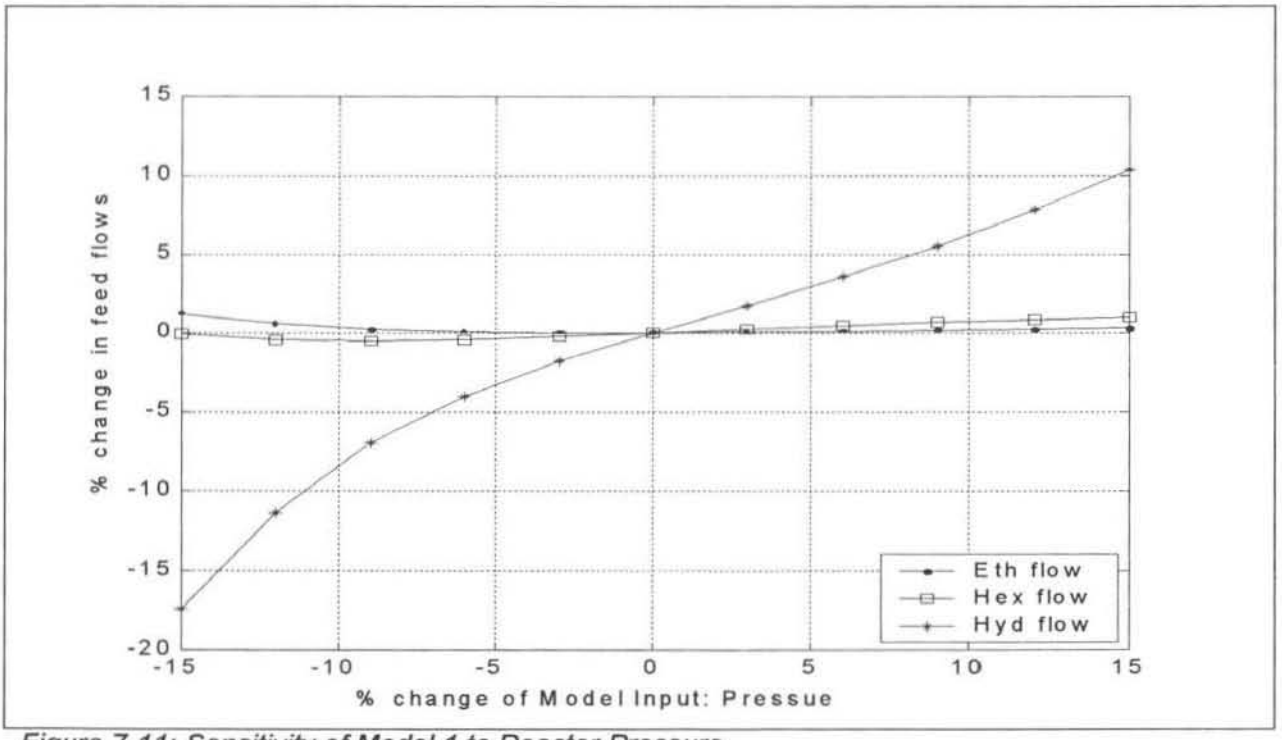


*Figure 7.11: Sensitivity of Model 1 to Reactor Pressure*

Figures 7.10 and 7.11 are given to show how the model responds to a different reactor temperature and pressure. The relationships here are far more linear. Interesting results from these graphs show the temperature and pressure in the reactor are quite dependent on the hydrogen flow. This is probably since the hydrogen has a very small part to play in the reaction so increasing amount of hydrogen in the reactor increases the pressure. The effect on the temperature, although small, may have something to do with the extra gas volume that needs cooling in the cycle gas cooler.

## 7.2.2 Inverse Model 2 (Time-Invariant Algebraic Model)

### 7.2.2.1 Structure of Model 2

This model, shown in figure 7.12 (where times refer to the actual control sequence), was the most successful inverse time-invariant algebraic model of the reactor that was developed. It is relatively small with only 6 inputs and 2 outputs. However, based on the knowledge acquired so far, most of the important variables are represented here. As mentioned in section 7.1.2 the composition and flow variables are best modelled in terms of their ratios. The current ethylene flow rate (model input No. 3) was included in the neural network model. It was thought that this would have the effect of aiding the model to predict the future flow ratios based on the current ethylene flow rate and the future composition setpoints. The other three process variables used describe adequately the state of the reactor.

Model Inputs

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| $\%C_6/\ \%C_2$ | $\%H_2/\ \%C_2$ | Ethylene flow rate | Reac Pres | Reac. Temp | Prod Rate |
| t+1 (SP) | t+1 (SP) | t | t | t | t |

Model Outputs

| 1 | 2 |
|---|---|
| $C_6/\ C_2$ flow ratio | $H_2/\ C_2$ flow ratio |
| t+1 | t+1 |

*Figure 7.12: Structure of Inverse Model 2 (Time-invariant algebraic Model)*

### 7.2.2.2 Results from Model 2

#### 7.2.2.2.1 Training and Testing Standard Deviations

The most appropriate number of hidden nodes to use in this model was found to be 16. The method used for arriving at this figure can be seen in figure 7.13. More than 16 hidden nodes were not considered since it was thought that it might lead to overtraining of the model. Again the accuracy of the predictions for the hydrogen ratios was poorer than for those of the hexene ratio. This seemed to be inevitable with this type of modelling. As mentioned in section 7.2 the value used for model inputs 1 and 2 during training was the average of 4 time future steps.

| | No. Hidden nodes | $C_6$/ $C_2$ flow ratio | $H_2$/ $C_2$ flow ratio | Average value (training) | Average value (testing) |
|---|---|---|---|---|---|
| SD | 10 | 0.0701 | 0.1011 | 0.0856 | 0.0960 |
| SD | 7 | 0.0684 | 0.1112 | 0.0898 | 0.0940 |
| SD | 13 | 0.0686 | 0.0996 | 0.0841 | 0.0971 |
| SD | 16 | 0.0534 | 0.0850 | 0.0692 | 0.0830 |

*Figure 7.13: Results from Training and Testing Model 2*

#### 7.2.2.2.2 Sensitivity Analysis

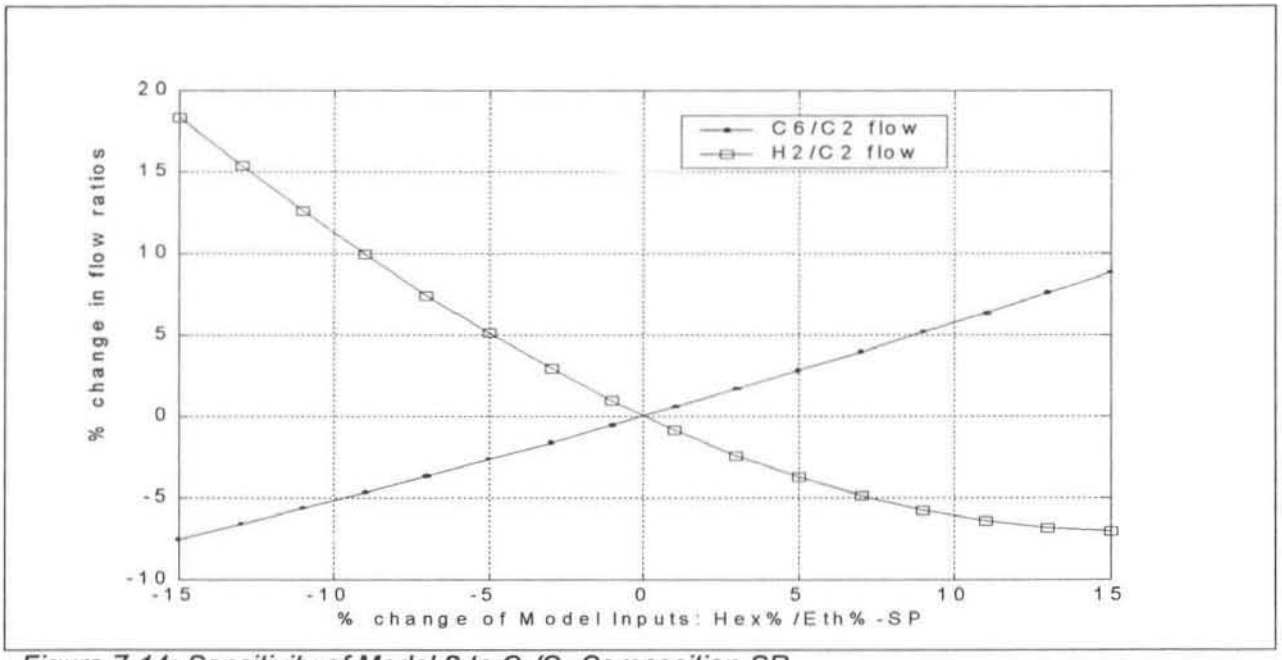


*Figure 7.14: Sensitivity of Model 2 to $C_6/C_2$ Composition SP*

The sensitivity curves proved useful for validating the neural network models. They also showed some interesting results and were far simpler to understand. The relationships appeared more linear after converting to ratios. In figure 7.14 the model's response to a change in the hexene to ethylene ratio is shown. An increase in setpoint results in the hexene ratio increasing and the hydrogen ratio decreasing.

A lower hydrogen to ethylene composition setpoint (figure 7.15) results in both ratios increasing, however since the increase of the hexene to ethylene flow ratio is much greater, the overall effect should be obtained. In general these diagrams are simpler to understand than those of the individual components.

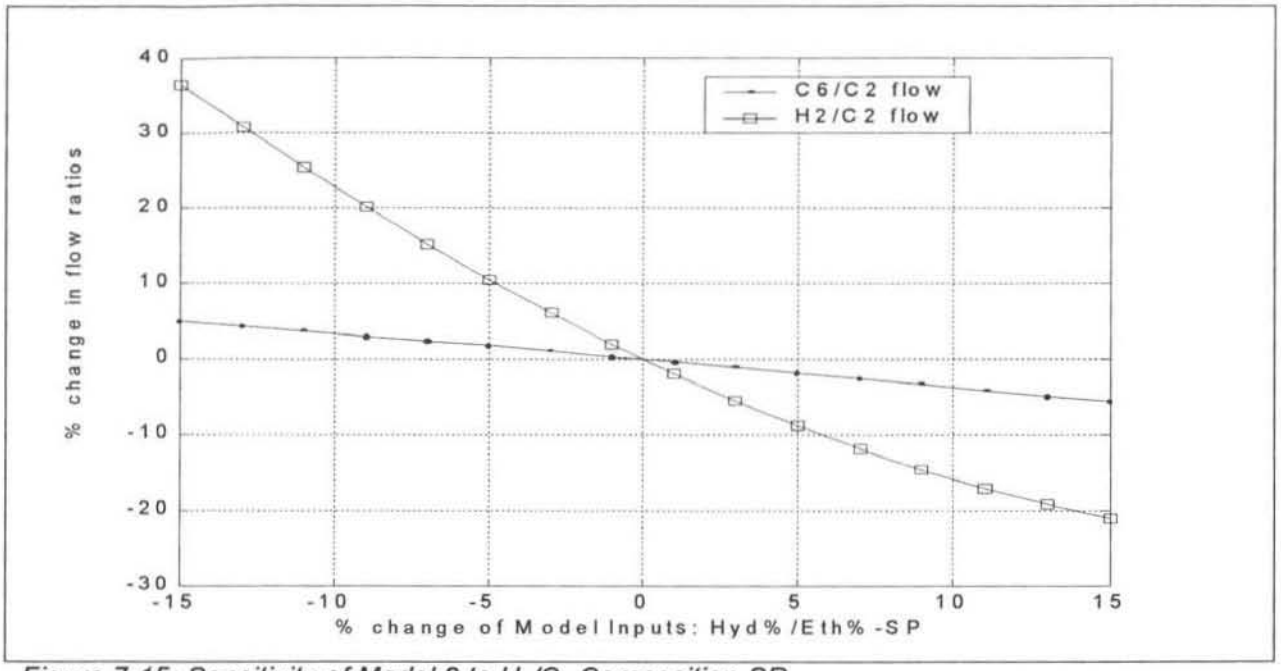


*Figure 7.15: Sensitivity of Model 2 to $H_2/C_2$ Composition SP*

Just to show that Model 2 behaves similarly to Model 1 with respect to the other common inputs consider figure 7.16. It is observed that an increase in the hydrogen to ethylene ratio and a decrease in the hexene to ethylene ratio results in the pressure rising. This is the same result as observed in figure 7.11.

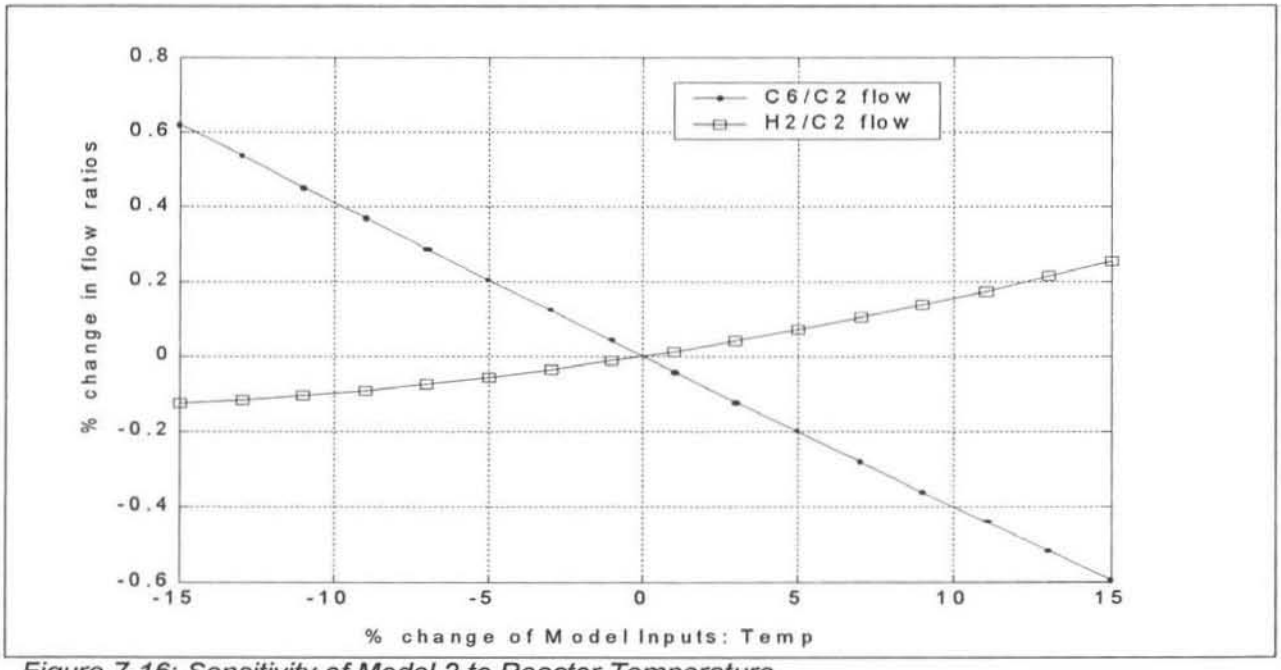


*Figure 7.16: Sensitivity of Model 2 to Reactor Temperature*

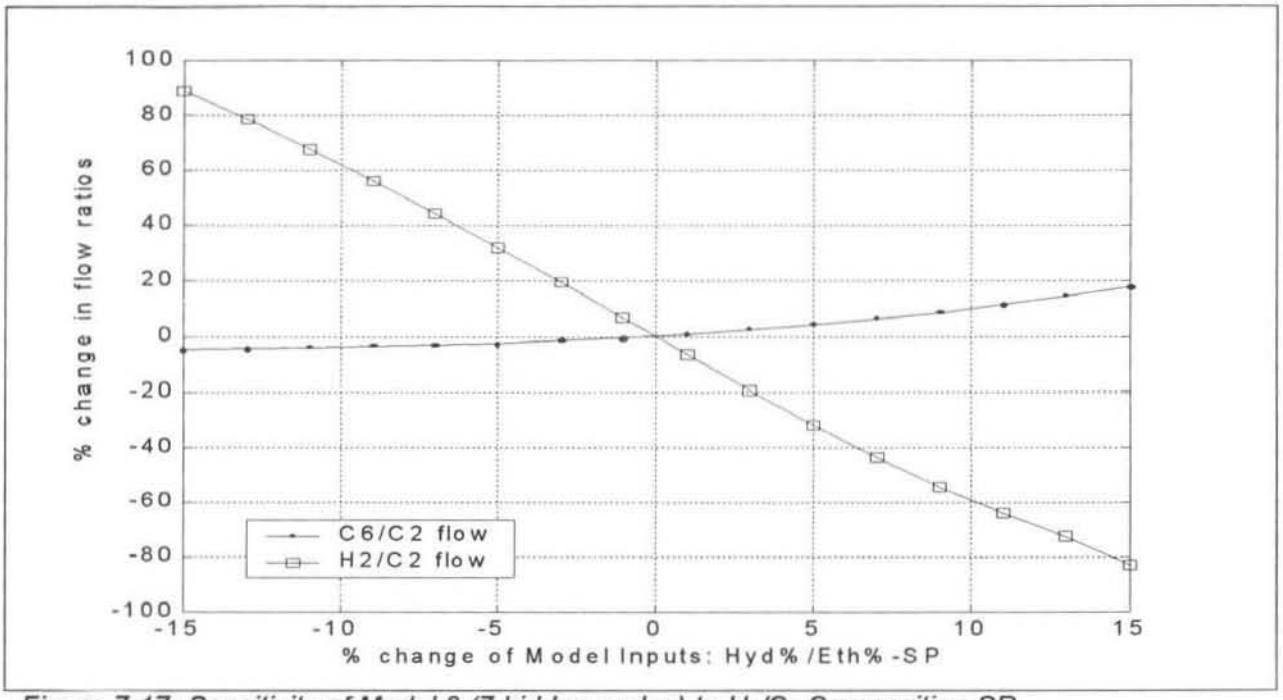


*Figure 7.17: Sensitivity of Model 2 (7 hidden nodes) to $H_2/C_2$ Composition SP*

The behaviour of the neural network models with different numbers of hidden nodes are all very similar. Figure 7.17 shows how a model with 7 hidden nodes reacts to hydrogen to ethylene ratio setpoint changes. The results are very similar to those in figure 7.15.

The performance of these time-invariant algebraic neural network models in the IIMC structure is discussed in chapter 8.

### 7.2.3 Inverse Model 3 (Time-invariant dynamic model)

#### 7.2.3.1 Structure of Model 3

This model is the first of two dynamic inverse neural network models presented. The task of creating a dynamic model of this system is quite difficult, as has been discussed previously. The large time lag in the system has to be accounted for. This has been done by including past values of the hydrogen and hexene ratios, as well as the future setpoint and the present value. Also the same technique of training the model with the average of a future trajectory of plant outputs for inputs 1 and 5 has been followed, as in the previous models. The past ethylene flow rate has also been included rather than the current flow rate since this value has more effect on the considered compositions in the reactor. The same reactor process variables as in Model 2 were included. It was felt that these were necessary to give the model some indication of the present plant conditions.

This model, shown in figure 7.18, was the most successful dynamic model of the reactor that was developed. It had 13 inputs and only predicts the hexene and hydrogen flow ratios. The best number of hidden nodes used was determined to be 16.

Model Inputs

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| $\%C_6/\ \%C_2$ | $\%C_6/\ \%C_2$ | $\%C_6/\ \%C_2$ | $C_6$ flow/ $C_2$ flow | $\%H_2/\ \%C_2$ | $\%H_2/\ \%C_2$ |
| SP | t | t-50 | t-50 | SP | t |

| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|
| $\%H_2/\ \%C_2$ | $H_2$ flow/ $C_2$ flow | $C_2$ flow | Reac Temp | Reac Pres | Prod Rate | Catalyst feed rate |
| t-50 | t-50 | t-50 | t | t | t | t |

Model Outputs

| 1 | 2 |
|---|---|
| $C_6/\ C_2$ flow ratio | $H_2/\ C_2$ flow ratio |
| t+1 | t+1 |

*Figure 7.18: Structure of Inverse Model 3 (Time-invariant dynamic Model)*

### 7.2.3.2 Results from Model 3

#### 7.2.3.2.1 Training and Testing Standard Deviations

Individual Component Ratios (Training)

| | $C_6/\ C_2$ flow ratio | $H_2/\ C_2$ flow ratio |
|---|---|---|
| SD (RMS) | 0.05187 | 0.05619 |

Overall results

| | Training | Testing |
|---|---|---|
| SD (RMS) | 0.05403 | 0.05626 |

*Figure 7.19: Results from Training and Testing Model 3*

The standard deviation of the training and testing errors are lower than for any previous model obtained. Not only is the training error low, the error from the testing set is also low. This could indicate that the model has created a good representation of the process. Another pleasing factor is the improvement in the prediction of the hydrogen to ethylene flow ratio. This could be as a result of the model receiving the past value of the flow ratio as an input, hence it has a reference value when predicting the future value.

#### 7.2.3.2.2 Sensitivity Analysis

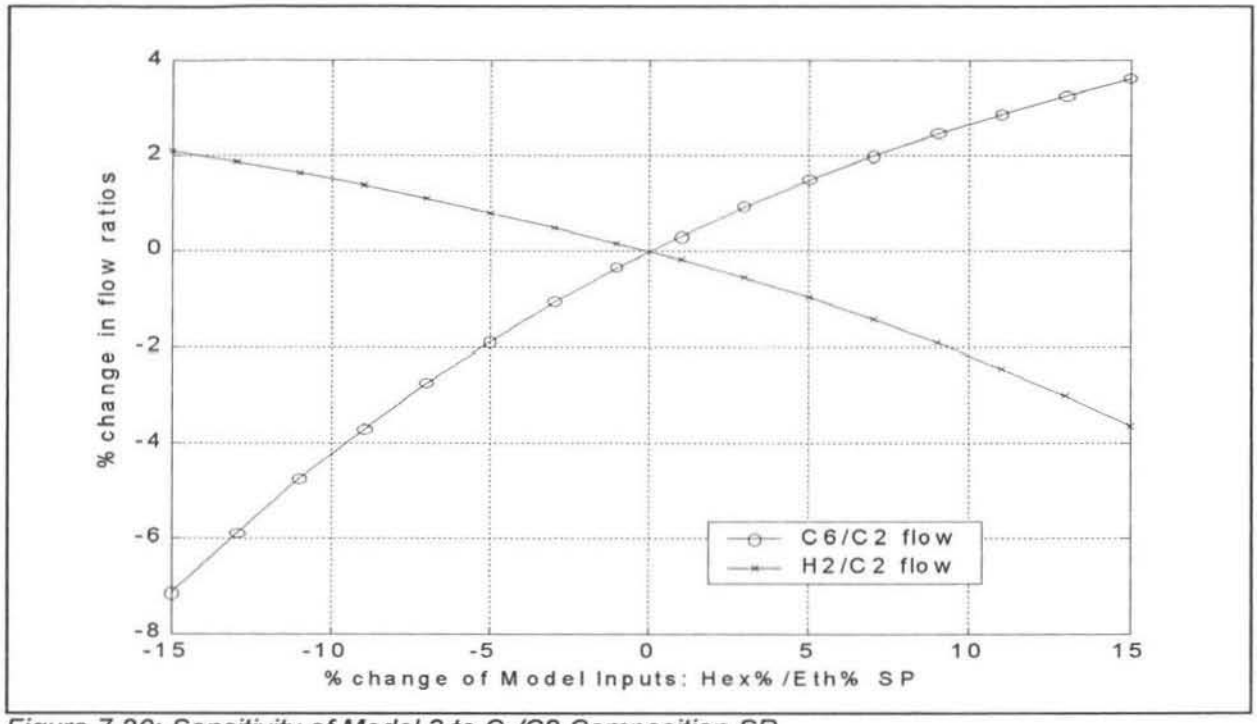


*Figure 7.20: Sensitivity of Model 3 to $C_6$/C2 Composition SP*

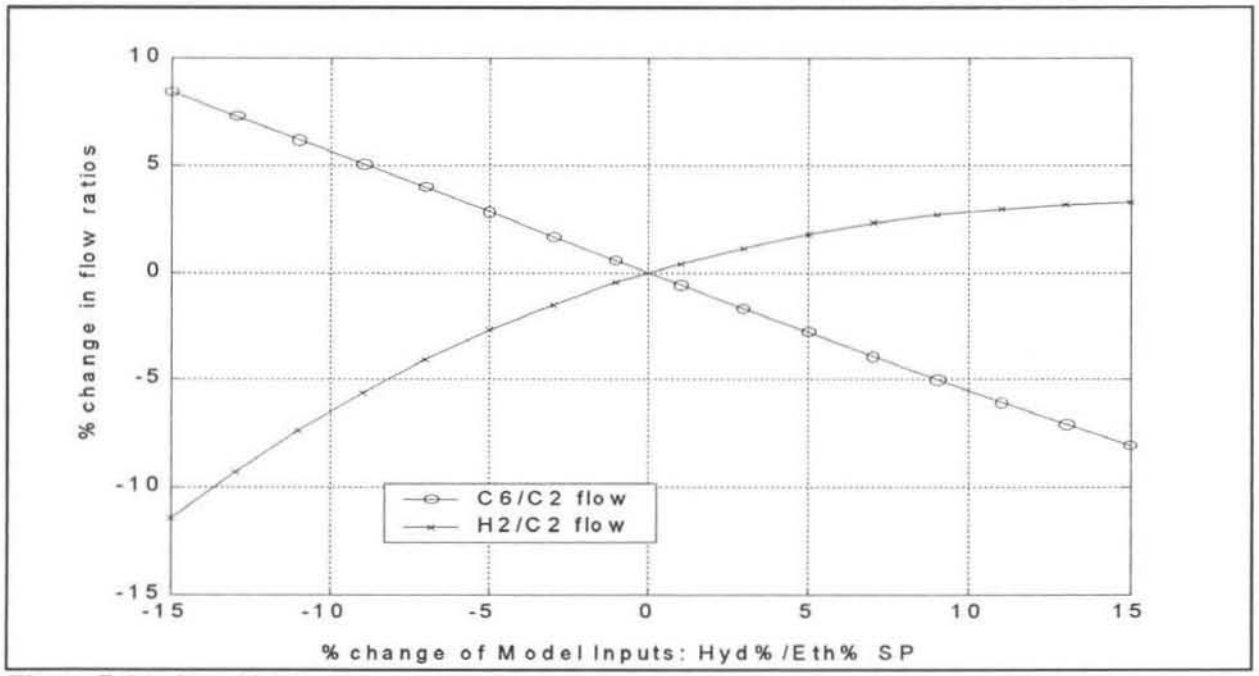


*Figure 7.21: Sensitivity of Model 3 to $H_2/C_2$ Composition SP*

The sensitivity of this model is shown in figures 7.20 and 7.21. The inputs that were varied to the model are the composition ratio setpoints. The model's response to a change in the hexene to ethylene ratio setpoints is shown in figure 7.20. Again the results are what would be expected intuitively and also agree with the results in figure 7.14. A higher ratio setpoint would result in the model suggesting an increase in the hexene flow ratio, while decreasing the hydrogen flow ratio. It is however interesting to note that the magnitudes of the suggested

changes are smaller than those suggested in figure 7.14. This is most probably since the model is dynamic, hence it knows how the ratios have changes in the past 50 minutes, and is less likely to suggest a value that differs too greatly to what the previous values have been. This is a result of it being trained on actual plant data, where there are very few large sudden setpoint changes. The results in figure 7.21 are in agreement with those shown in figure 7.15, which show the model's response to an input change in the hydrogen to ethylene flow ratio.

This particular model performed very well as a controller and is discussed in more detail in chapter 8.

### 7.2.4 Inverse Model 4 (Time-Invariant Dynamic Model)

#### 7.2.4.1 Structure of Model 4

This last example has been included as an illustration of some of the novel dynamic models that were tested. A very interesting study was done using this model and the data that is was trained on. This is discussed a bit later in section 7.2.4.2.3. In this model only the composition and flow ratios have been modelled. None of the process operating conditions has been included. The desired result is to see how effective the neural network model is at learning the relationship between these variables, and if these relationships are truly multi-variable or not.

Model inputs 1 and 2 are the current process values of the ratios. The model input 3 ($\Delta$% $C_6/C_2$) when the controller is on-line is the difference between the setpoint of the hexene flow ratio and the current ratio. The same applies for the model input 4. During training of this model the value used for the composition setpoint was the process value at a time 50 minutes into the future. This ensures that during training the model learns to take the time lag of the process into account. The outputs of the model are the same as in the previous model, *viz* the suggested flow ratios.

Model Inputs

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| %$C_6$/ %$C_2$ | %$H_2$/ %$C_2$ | $\Delta$ %$C_6$/ %$C_2$ | $\Delta$ %$H_2$/ %$C_2$ |
| t | t | $\Delta t = 50$ | $\Delta t = 50$ |

Model Outputs

| 1 | 2 |
|---|---|
| $C_6$/ $C_2$ flow ratio | $H_2$/ $C_2$ flow ratio |
| t+1 | t+1 |

*Figure 7.22: Structure of Inverse Model 4 (Time-Invariant Dynamic Model)*

### 7.2.4.2 Results from Model 4

#### 7.2.4.2.1 Training and Testing Standard Deviations

| | No. Hidden nodes | $C_6$/ $C_2$ flow ratio | $H_2$/ $C_2$ flow ratio | Average value (training) | Average value (testing) |
|---|---|---|---|---|---|
| SD | 9 | 0.0812 | 0.1027 | 0.0920 | 0.1108 |
| SD | 6 | 0.0872 | 0.1144 | 0.1008 | 0.1214 |
| SD | 12 | 0.0661 | 0.0993 | 0.0827 | 0.0996 |

*Figure 7.23: Results from Training and Testing Model 4*

As can be seen in figure 7.23 the best number of hidden nodes was determined to be 12. Here again more than 12 hidden nodes were not considered since it was feared that it might lead to over training of the model. The standard deviations of the errors during training are larger here than in Model 3. This may be an indication of the benefit of including information relating to the current state of the reactor as model inputs (i.e. temperature, pressure etc).

#### 7.2.4.2.2. Sensitivity Analysis

The model response to changing the inputs 3 and 4 is documented here. Before examining the figures consider what the delta term represents. If it is a positive number then the process variable value (composition ratios in this case) is less than the setpoint. As this number increases so the process variable moves further away from the setpoint. In order to bring the process variable closer to the setpoint the flow ratio of that composition should increase, to increase the gas composition ratio.

This can be seen in figure 7.24. As the delta term increases so the hexene flow ratio increases to compensate for the larger difference between the setpoint and the process variable. Similarly if the delta term increases in the negative direction, the model accordingly reduces the gas flow ratios. The same trends are observed when the delta hydrogen composition is varied in figure 7.25.

#### 7.2.4.2.3 Comparison with Best Fit Polynomial Equations

An interesting study was performed on the training data, using this particular model as a benchmark. It was desired to see whether in fact the neural network was a multi-variable model of the system and gave more accurate predictions than a simple polynomial relationship between the corresponding gas compositions and flow ratios. This would indicate whether the neural network was in fact accounting for the coupling between the hydrogen and hexene composition and flow ratios.

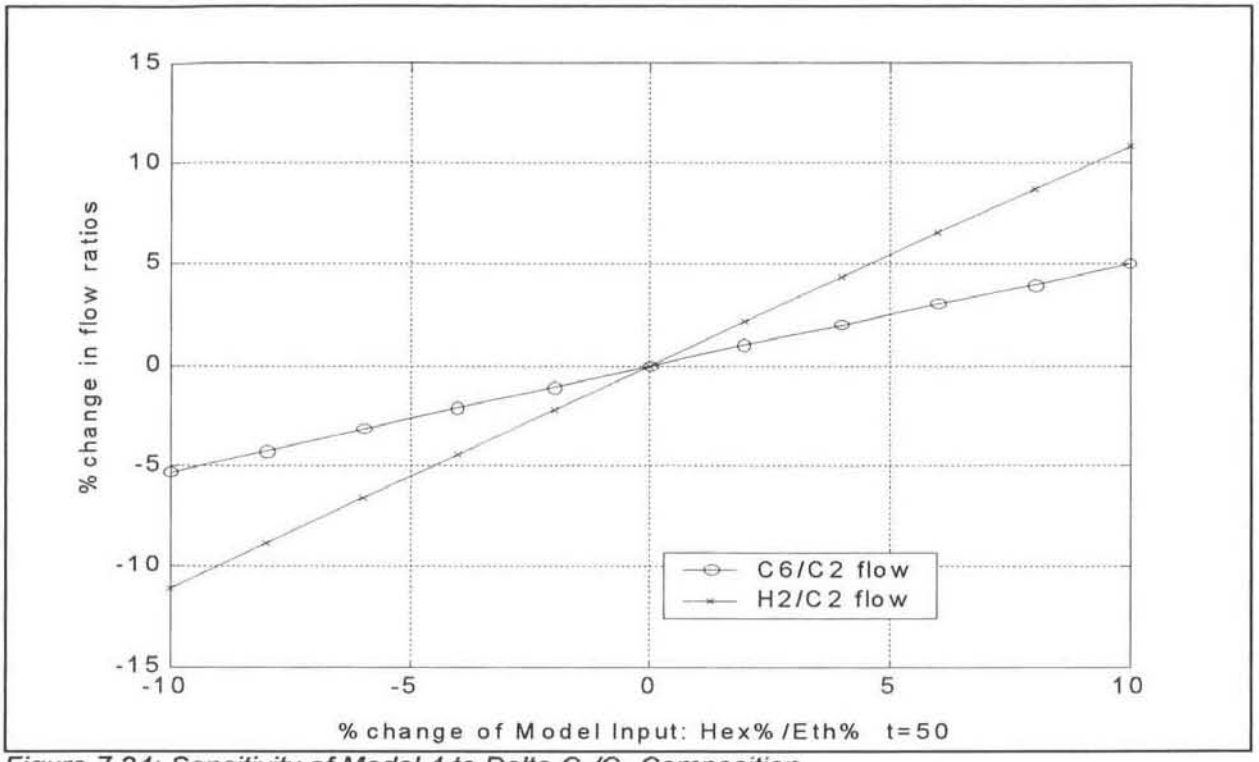


*Figure 7.24: Sensitivity of Model 4 to Delta $C_6/C_2$ Composition*

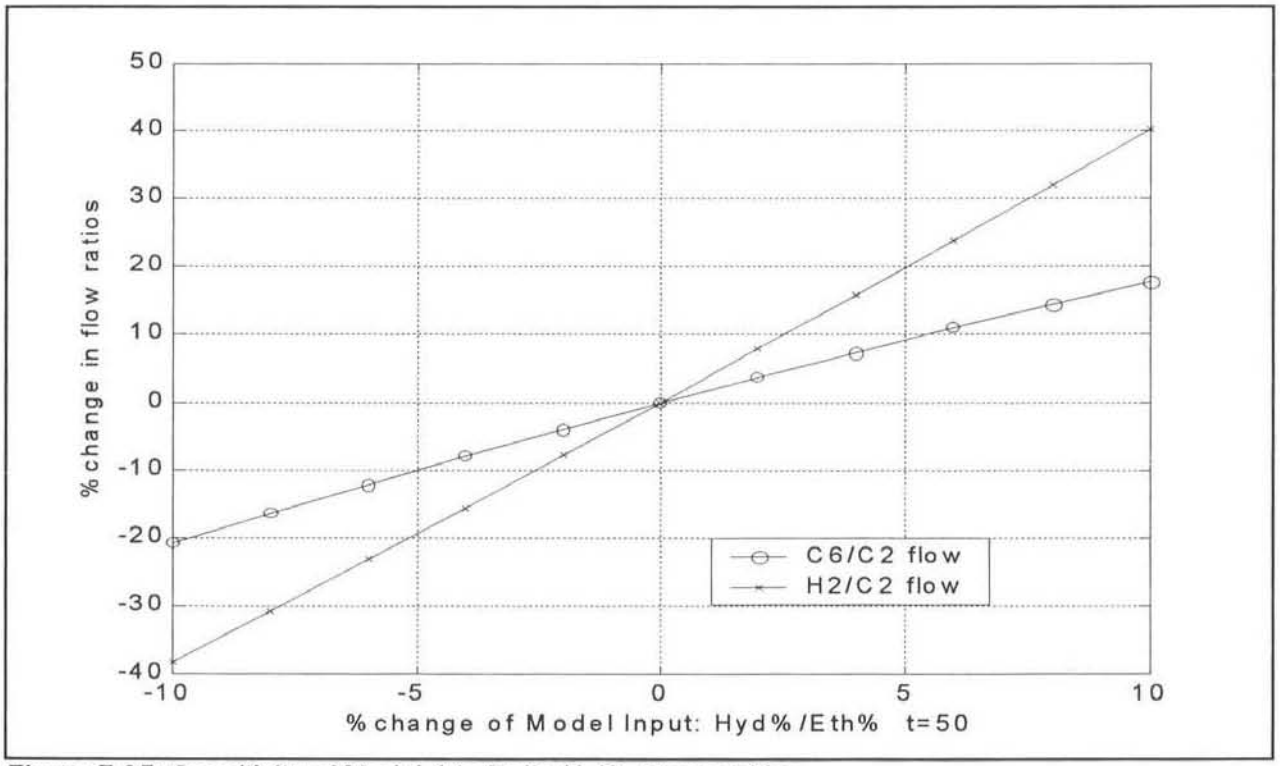


*Figure 7.25: Sensitivity of Model 4 to Delta $H_2/C_2$ Composition*

As was clearly evident in figures 7.4 and 7.5 the composition ratios lagged behind the flow ratios by about 50 minutes. The composition ratios were shifted forward in time by 50 minutes (to remove the lag) and the individual flow ratios and compositions ratios plotted. These plots are shown in figures 7.26 and 7.27. A general best polynomial equation was fitted through these

points to obtain a relationship between the respective flow and composition ratios. These equations, which are not given for confidentiality reasons, could then be solved for the flow ratio.

Flow ratio $(_{C6 \text{ or } H2 / C2}) \approx$ f (Composition ratio $_{C6 \text{ or } H2 / C2}$) *Equation 7.1*

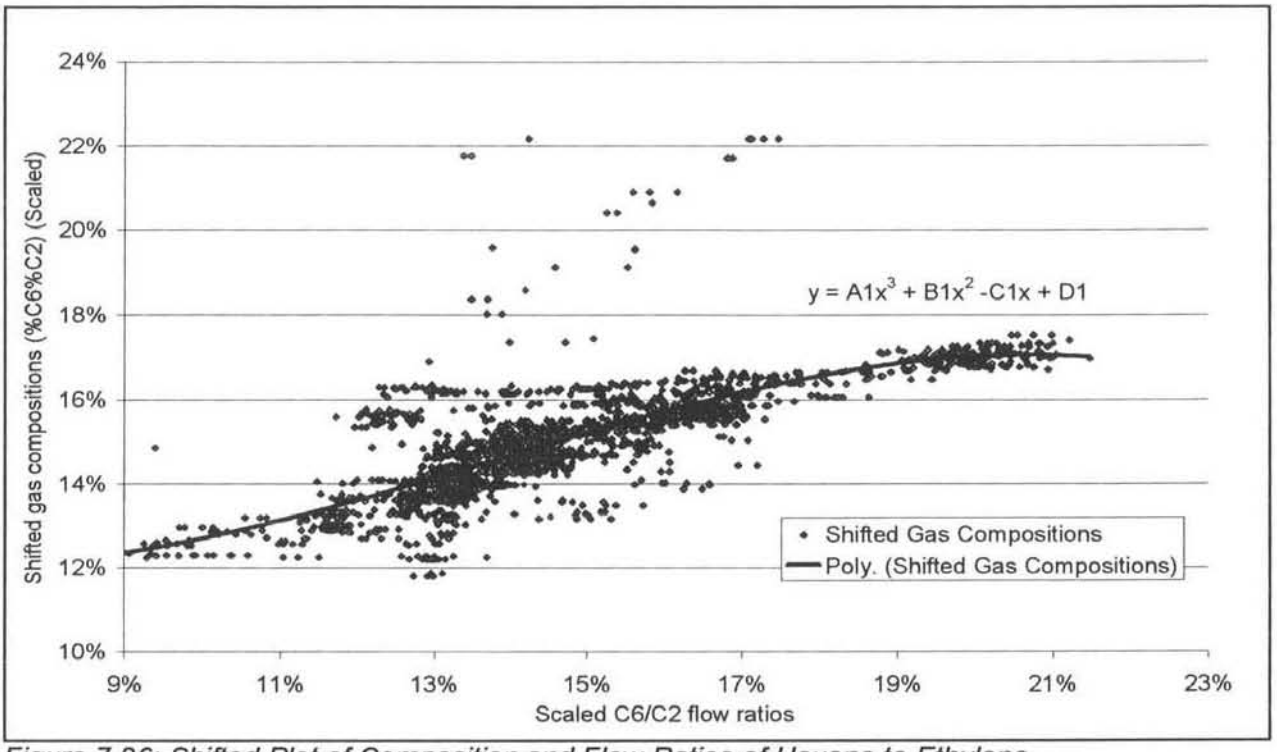


*Figure 7.26: Shifted Plot of Composition and Flow Ratios of Hexene to Ethylene*

Using the testing data set, shown in figures A.1 and A.2, these flow ratios were calculated from the composition ratios using the equations. The predictions of neural network model 4 (with 12 hidden nodes) of the flow ratios were compared with those of the equations and plotted on the scatter-graphs in figures 7.28 and 7.29.

It is apparent from examining the graphs that the neural network predictions (the crosses) fall within an area closer to the actual values than do the predictions of the equations (the circles). The standard deviation of these errors was not calculated since it was evident that the neural network predictions were better. The areas on the boundaries of the scaling limits are the areas where the neural network models have problems (i.e. the on-off hydrogen flow rate). Only the predictions of the neural network model with 12 hidden nodes are shown, since they were grouped closer to the actual compositions than those of the models with 6 and 9 hidden nodes.

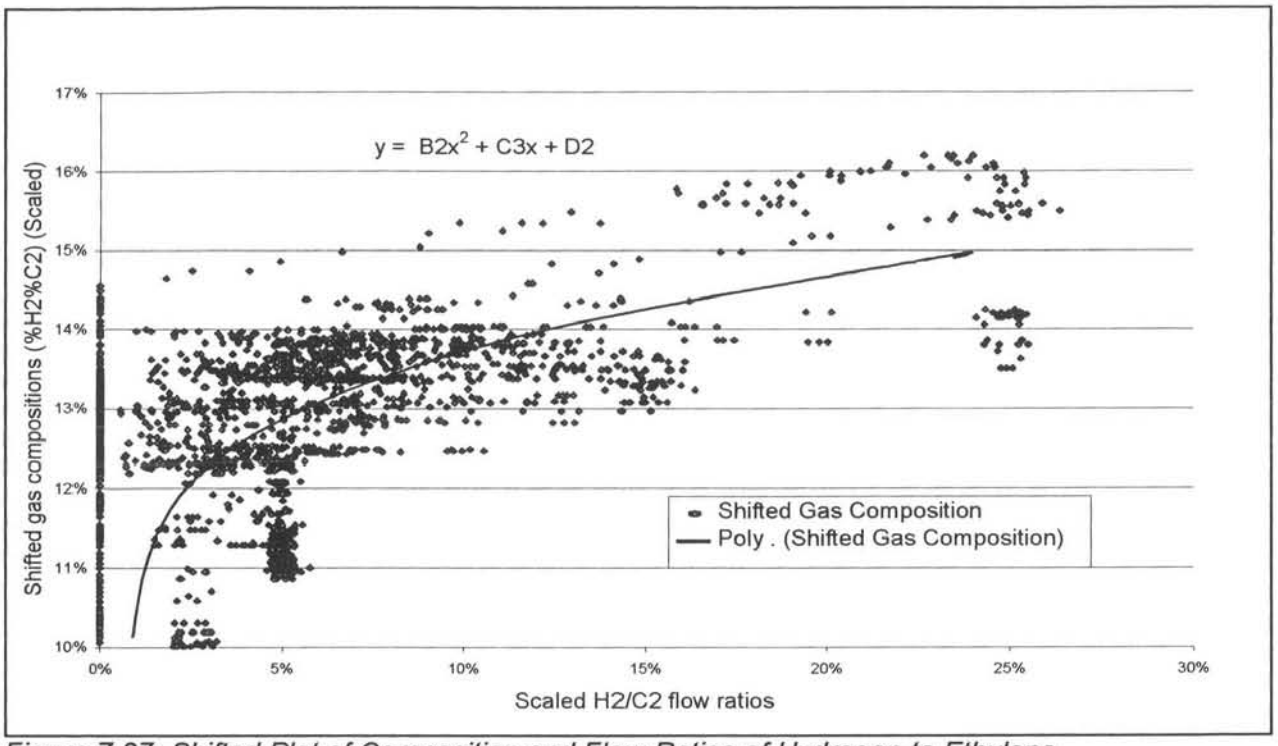


*Figure 7.27: Shifted Plot of Composition and Flow Ratios of Hydrogen to Ethylene*

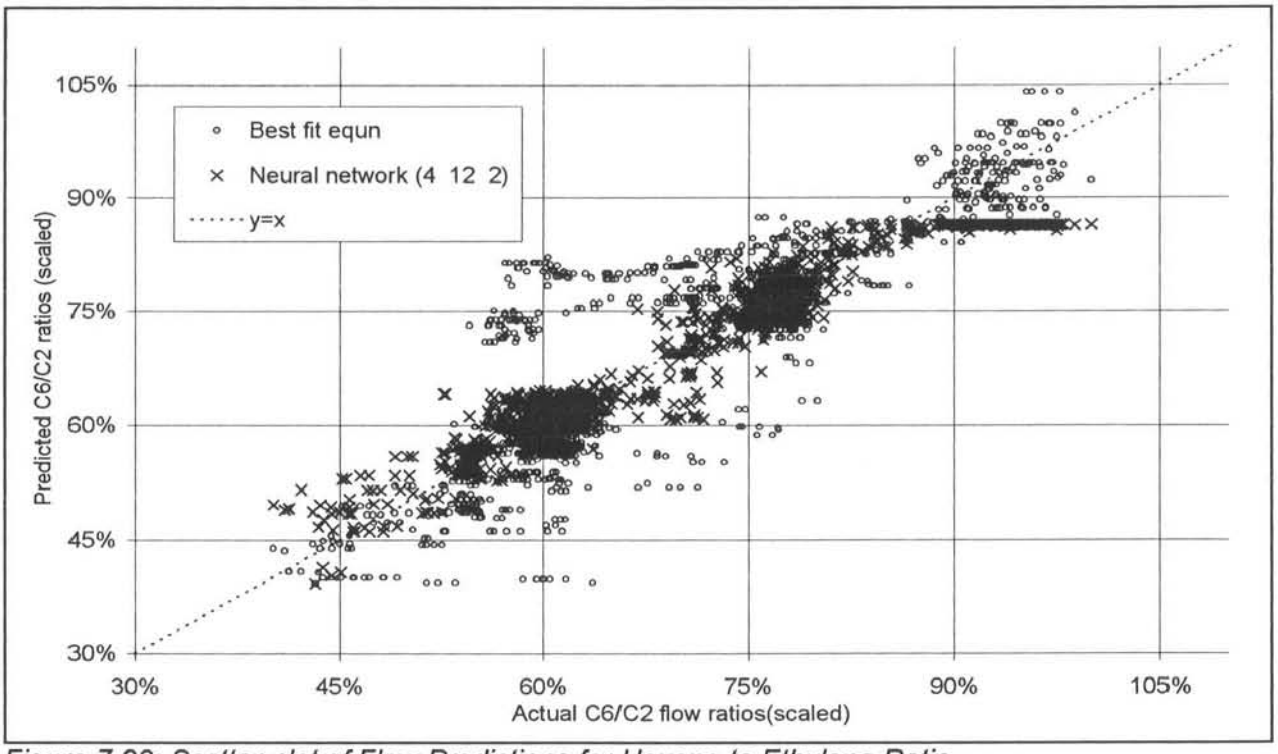


*Figure 7.28: Scatter-plot of Flow Predictions for Hexene to Ethylene Ratio*

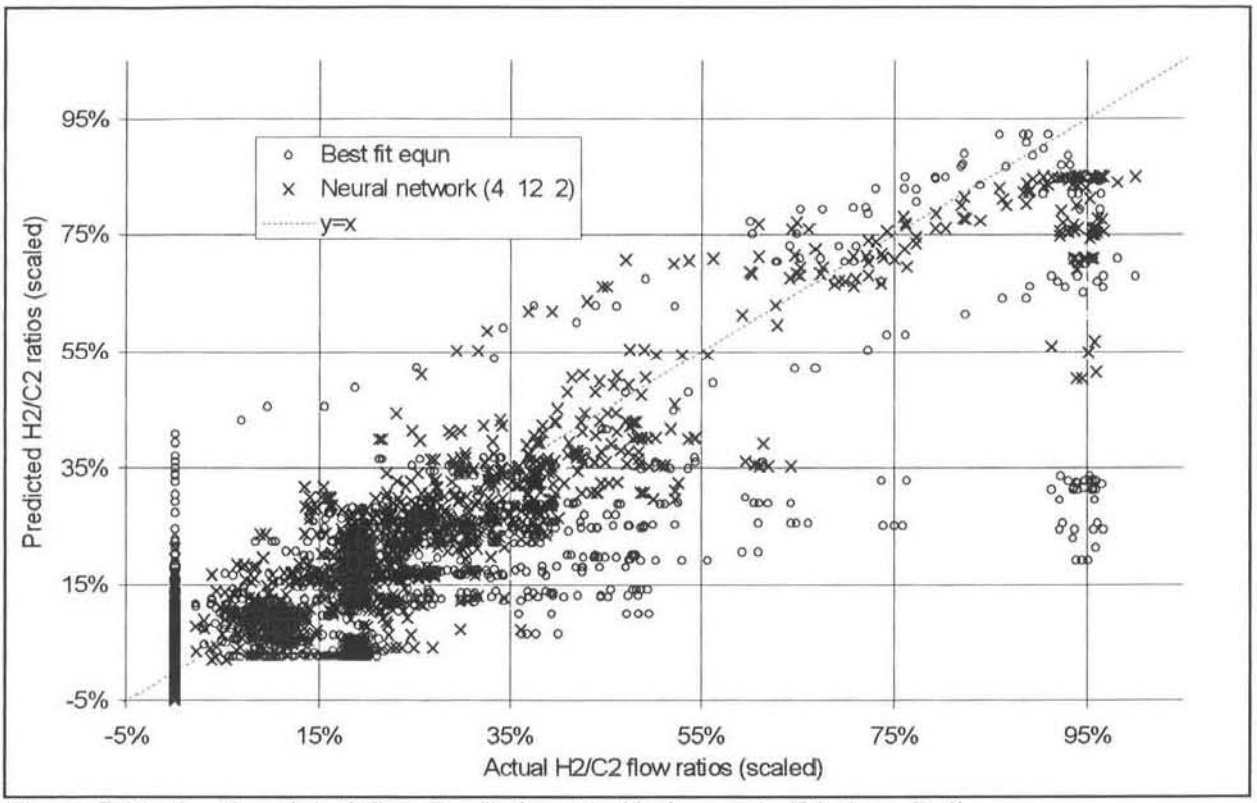


*Figure 7.29: Scatter-plot of Flow Predictions for Hydrogen to Ethylene Ratio*

What can be concluded from this small exercise is that these simple neural network models appear to more accurately model the process than the non-multivariable equations. The significance of this is fully understood if one considers how the PID controllers that set the flow ratios to the polymer reactor operate, as shown in figure 2.11. These are not multivariable controllers since they adjust the flow ratios by considering only that respective composition ratio. This justifies testing these neural network models for use in a multivariable controller to replace or aid these PID controllers.

# CHAPTER 8

# 8. ONLINE POLYMER PLANT CONTROL

## *8.1 The IIMC Neural Network Controller On-line*

One of the main reasons for initiating the research and for proceeding this far with the work was to test the proposed controller structure on an industrial application. The SASOL POLYMERS polymer reactor has been mentioned in chapters 1, 2 and 7. In chapter 7 the reactor neural network models that were developed for this control application were discussed.

The exercise of programming the controller into the polymer plant DCS was one that took a considerable amount of time, effort and co-ordination with the onsite engineers. The amount of code is fairly large for a relatively small application. It is explained in detail in Appendices C to J.

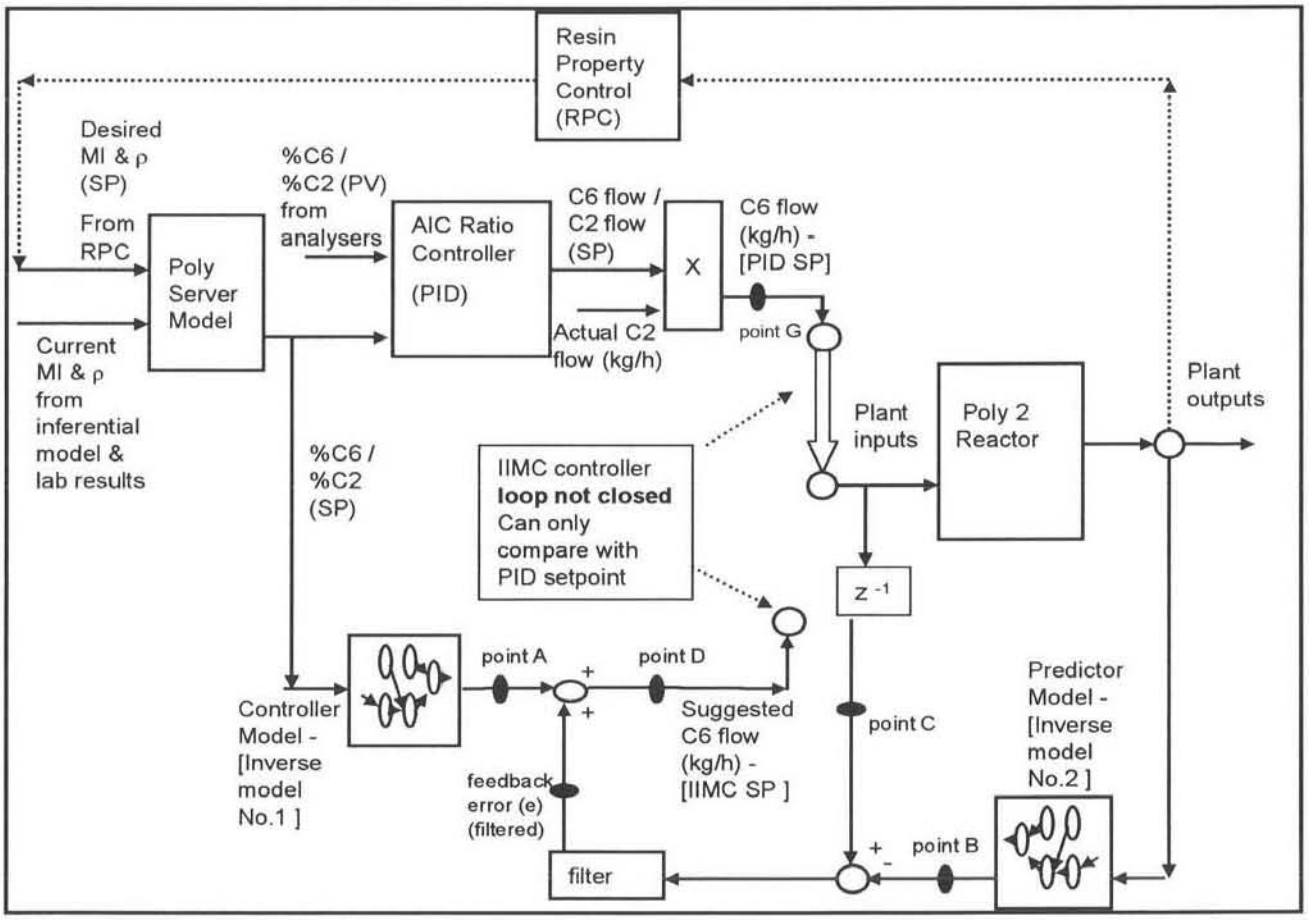


*Figure 8.1: The IIMC and PID controllers in the Polymer Plant Control Scheme*

Figure 8.1 shows how the IIMC neural network controller fits into the current DCS control system. This figure is a combination of the diagrams in figures 2.10, 2.11, 3.3 and 5.4. It can be seen how the IIMC structure from figure 5.4 has been incorporated into the overall control

system. Not all of the interconnections for the neural network model inputs have been shown, as these are dependent on the different model structures.

The diagram also shows how the inputs and outputs of the current PID controllers fit into the overall plant control. The figure only indicates how the flow rate setpoint for the co-monomer (hexene) is obtained from the PID controllers (point G in figure 8.1). A similar PID loop is also set up for the hydrogen flow rate. As mentioned these PID controllers are not multi-variable and do not account for the inter-relationship between the co-monomer and hydrogen flow rates and the respective composition ratios to ethylene in the reactor.

The IIMC controller's suggested setpoints (point D on figure 8.1) are presented on an output interface, but, as is seen, the loop has not yet been closed. For the results presented in this chapter (and dissertation) the loop remains open. As an initial indication of the IIMC neural network controller performance the suggested setpoints (point D) can be compared with the actual setpoints given by the PID loops (point G) and used on the plant. If the setpoints from the two controllers are similar then it indicates that the IIMC controller performs at least comparably with the PID controllers. To determine in fact which controller scheme works best, extensive testing would have to be done to see how the different schemes handle different plant conditions, input disturbances and grade transitions.

The decision to test the proposed controller's closed loop performance depends on the open-loop results, as presented here. If further action is to be taken it will be at the discretion of the polymer plant management based upon recommendations from this dissertation and the polymer plant engineers involved in the project.

The reasons for not closing the loop are quite simple. A multi-million Rand plant is not going to jeopardise its polymer production on a relatively untested new controller structure. The normal channels to follow here would be to test the controller's performance against the PID controller's performance on the small-scale pilot plant that is on the same site. This would be an interesting study and one that could warrant further detailed research.

## *8.2 Results*

As is mentioned in the discussion on the controller software developed (Appendix C1) it was possible to test 3 different neural network models in the IIMC structure simultaneously. This proved to be very useful in identifying the better neural network models and showing that tuning the filter settings ($\alpha$) for the feedback error term (e) affected the controller performance. The results shown here are not very extensive since only a limited amount of time was available during which the plant was producing the grades on which the models could be tested.

The results shown here are from a similar hexene grade to what is shown in the testing data, presented in figure A.1. The results from the individual models are shown in the following sections. The controllers were all initially started running on-line (open loop) with the alpha filter value set to one ($\alpha$ = 1), there was no filtering of the feedback error (e). The controller predictions (points A & B on figure 8.1) and feedback errors (e) were plotted and a decision was made as to what the next alpha value to be tested should be.

Again it must be stressed here that the IIMC neural network controller's predictions (point D) are being compared with the actual setpoints suggested by the current PID controllers (point F). In most of the graphs shown the thick bold line represents the PID setpoint suggestions. These results give no indication of whether the neural network predictions are better than those of the PID controllers. They simply show whether the neural network controller is providing sensible suggestions.

The reader is also reminded here how the feedback error term (e) in the IIMC controller is calculated. As is shown in figure 8.1 the feedback error is the difference between the suggested PID setpoint (point C) and the prediction (point B) from the internal neural network model (predictor model). A large error term shows that a significant plant-model mismatch exists. The feedback error corrects the controller for this mismatch and this is the reason for incorporating it into the controller.

### 8.2.1 Model 1

The structure of neural network model 1 was shown in figure 7.6. This time-invariant algebraic model predicted the actual flow rates of hexene and hydrogen required. The results shown in figures 8.2 to 8.7 were obtained by using Model 1 in the IIMC controller as shown in figure 8.1. For the results shown in figures 8.2 and 8.3 the feedback error term (e) was not filtered, i.e. the alpha ($\alpha$) values were equal to 1.

From figure 8.2 it can be seen how the IIMC suggested flow rates (thin black line) compares with the setpoint of the PID controller for the hexene flow (solid black line). Although the suggestions are a bit "jumpy" one can imagine that a smoothed trend line through the points would probably be quite similar to the PID setpoints. In figure 8.3 the outputs from the two separate neural network models (points A & B on figure 8.1) and the feedback error (e) are shown. The predictions of the two neural networks are very similar. The reason for this is that the plant is running at near steady state conditions, hence the inputs to the two models are very similar. The variables do not change much with the relatively small sampling rate being used (4 minutes). The main difference between the inputs to the two models is that the controller model

(inverse model No. 1) takes the composition setpoints as inputs, while the predictor model (inverse model No. 2) takes the actual plant compositions.

It is also interesting that the values of the predictions are not close to the actual PID suggested value (compare figure 8.2 and 8.3). The predictions are in general larger than the actual values, with the rather large negative feedback error (e) showing the definite existence of a plant-model mismatch. Another striking feature of figure 8.3 is that the feedback error (e) curve appears to have an inverse shape to that of the neural network model predictions. The reason for this is twofold; firstly, the more erratic neural network predictions (point B) are subtracted from the more stable actual setpoints (point C) when the feedback error (e) is calculated, so they have a larger effect on the values. Secondly the feedback error (e) is not filtered so the curve is not smoothed, hence the large spikes are still evident.

This shows how important filtering the feedback error is to these types of controllers. The error curve is clearly responding to small plant disturbances. It was decided to set the alpha filter value to 0.5 for future tests ($\alpha$ = 0.5). It can also be noted that for this data set used above the plant operating conditions were very steady. There were no grade transitions and no major abnormal disturbances.

The results in figure 8.4 and 8.5 show the hexene flow rate suggestions (point D) with this new filter setting. The suggestions are visibly better than previously and even when the plant started undergoing a small grade transition at about 5am in the morning the suggestions were still comparable to the PID controller setpoints (point G) with the feedback error (e) staying at about -20%.

From figure 8.5 the smoothing effect of the filter on the feedback error can be seen. The two neural network model predictions (points A & B) are again similar and higher than the actual PID setpoints. It can be seen how they adjust to the changing plant conditions during the grade transition. Again it is noted the feedback error (e) curve has an inverse shape to that of the two model predictions. The effect of filtering the error can be seen since it does not respond to all the small fluctuations that the predictions are prone to, as was the case previously.

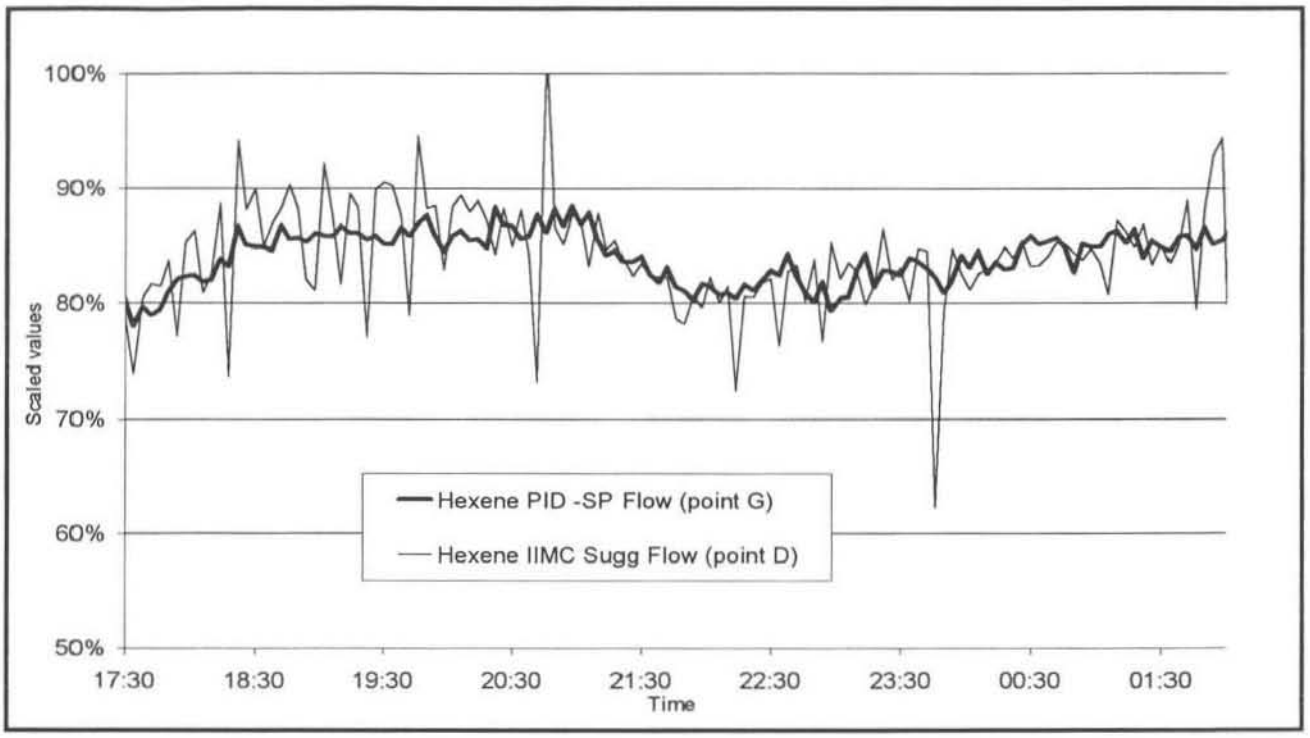


*Figure 8.2: 1-Hexene Flow Rate SP vs IIMC Suggestions (Feedback Error - $\alpha$ = 1)*

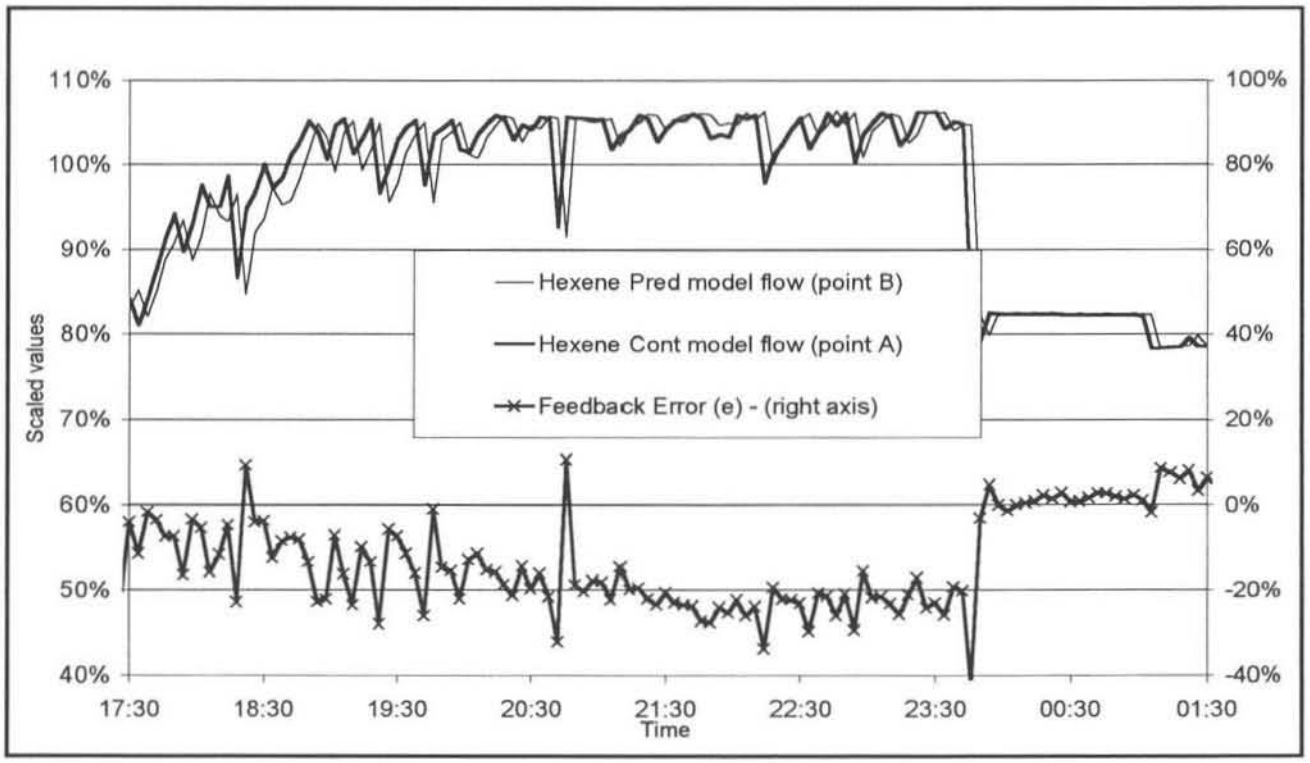


*Figure 8.3: 1-Hexene IIMC Internal Values and Feedback Error ($\alpha$ = 1)*

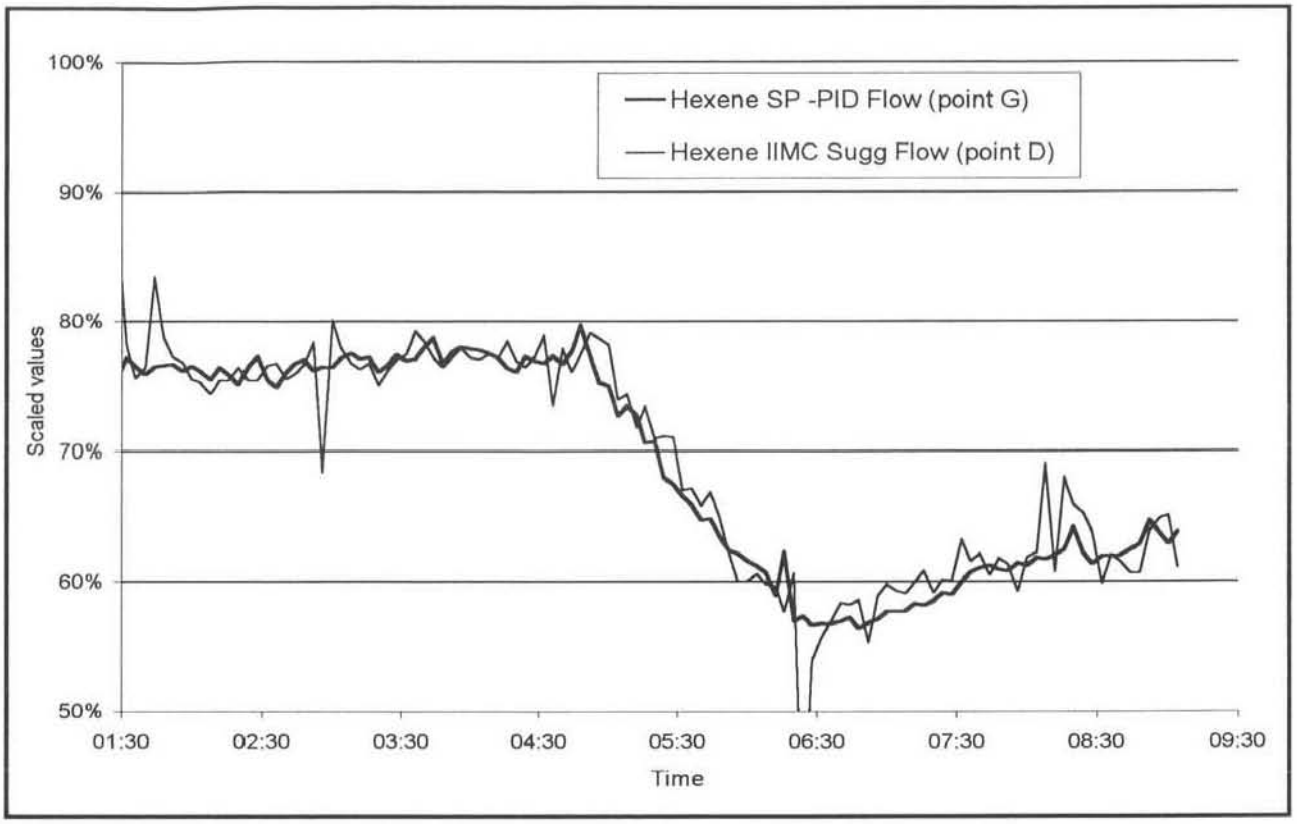


*Figure 8.4: 1-Hexene Flow Rate SP vs Suggestions (Feedback Error - $\alpha$ = 0.5)*

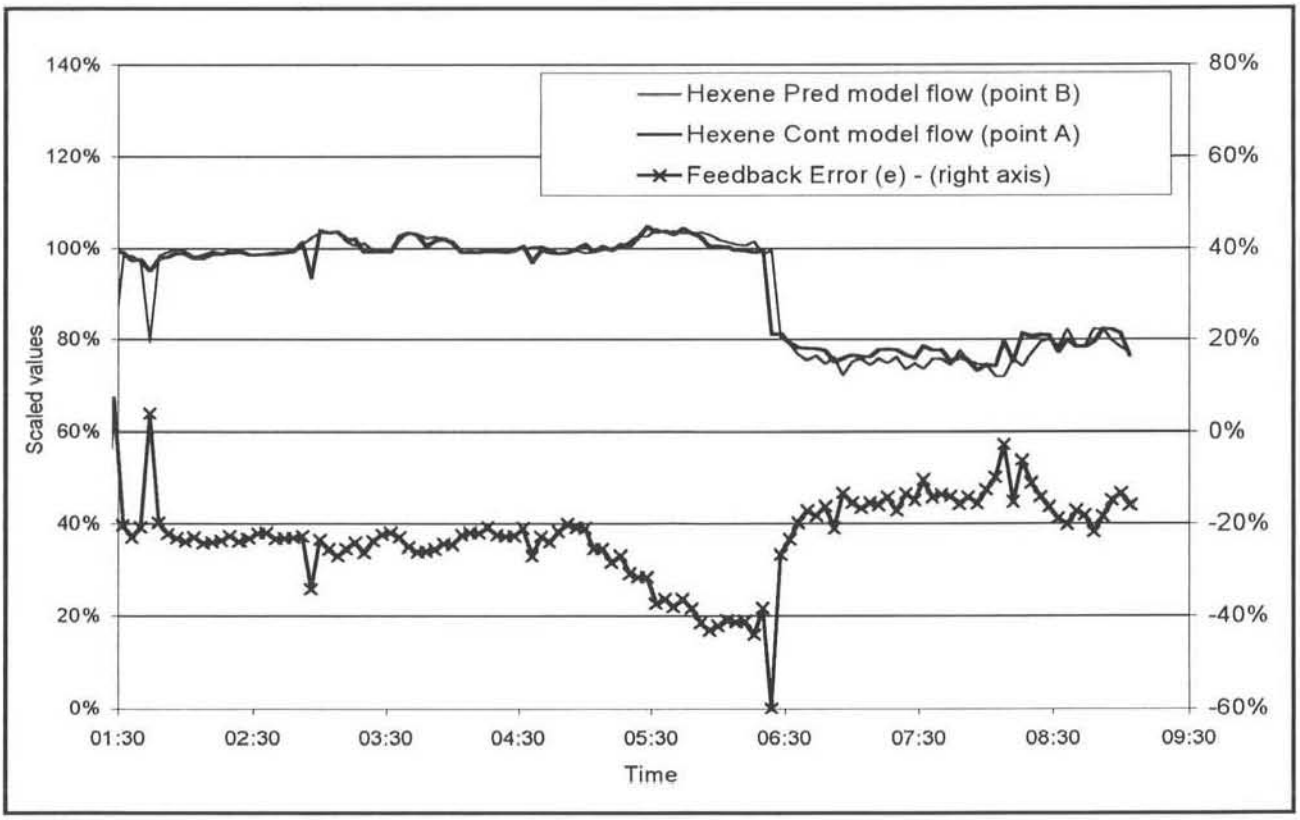


*Figure 8.5: 1-Hexene IIMC Internal Values and Feedback Error ($\alpha$ = 0.5)*

The IIMC Model 1 suggestions of the hydrogen flow rate are shown in figures 8.6 and 8.7. In figure 8.6 the IIMC suggestions (thin black line), using the unfiltered ($\alpha$ = 1) feedback error (e),

are compared to the PID hydrogen flow rates. As mentioned in section 7.2.1.2 the on-off hydrogen flow is difficult for the neural network models to predict. The IIMC suggestions, using the filtered ($\alpha$ = 0.5) feedback error, versus the PID flows are shown in figure 8.7. These do show an improvement from figure 8.6. However, they are still somewhat inferior to the hexene predictions. It must also be noted here that although provision was made in the code to limit the maximum step a suggested value may take, this feature had not yet been put into use. With this limit in place the controller would not be able to suggest such large step changes and the suggested values would be smoother. The neural network model predictions (points A & B) and feedback error (e) are not shown. The plant-model mismatch was even greater than in the corresponding hexene suggestions.

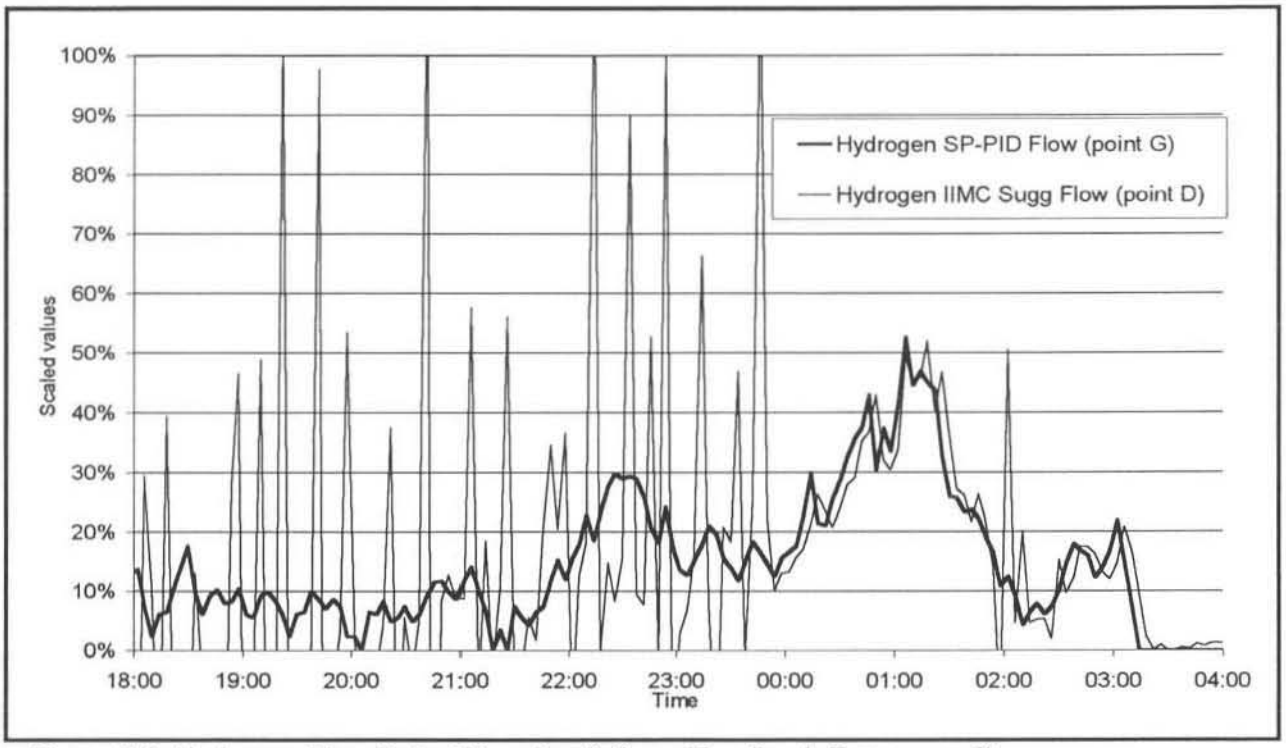


*Figure 8.6: Hydrogen Flow Rate SP vs Predictions (Feedback Error - $\alpha$ = 1)*

### 8.2.2 Model 2

The structure of Model 2 was shown in figure 7.12. This neural network model showed good results during training and testing (section 7.2.2.2), and it was expected to perform well as a controller. The results using Model 2 in the IIMC controller (as shown in figure 8.1) are presented in figures 8.8 to 8.11. They confirm that the model did give good results.

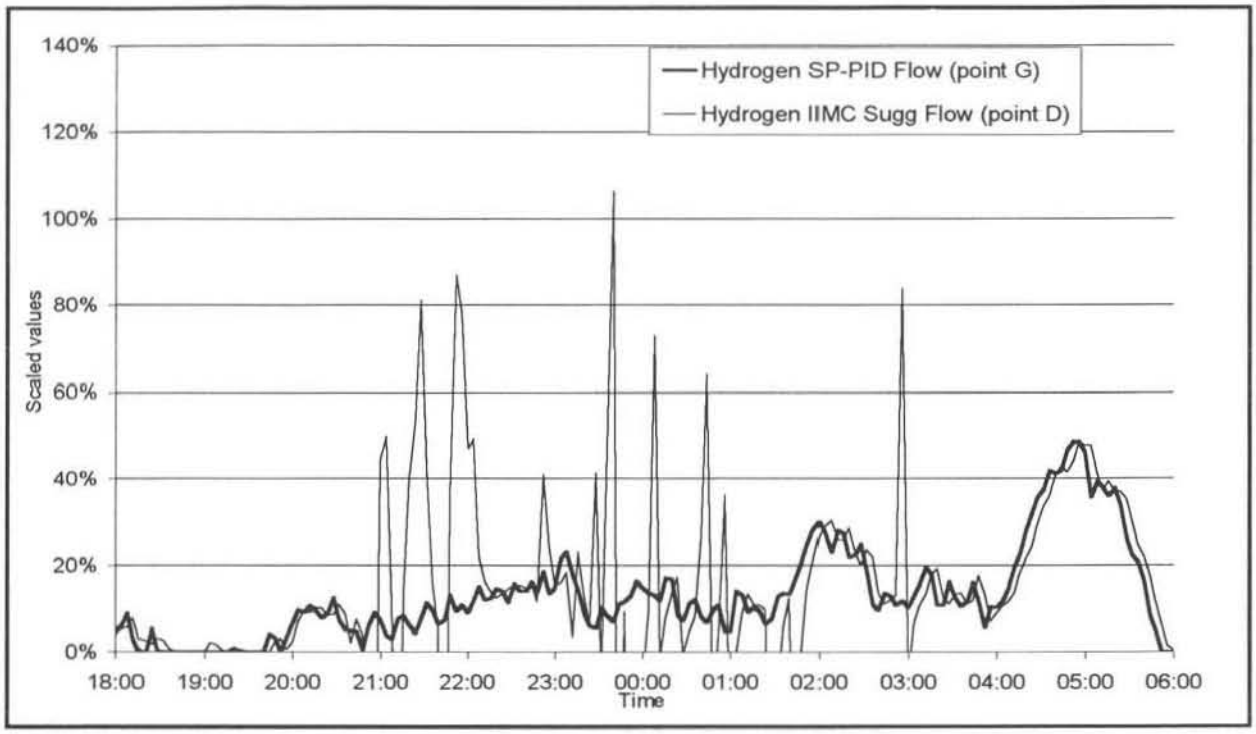


*Figure 8.7: Hydrogen Flow Rate SP vs Predictions (Feedback Error - $\alpha$ = 0.5)*

The IIMC suggestions (point D on figure 8.1) for the hexene : ethylene flow ratio, using the unfiltered ($\alpha$ = 1) feedback error (e), are shown in figure 8.8. The controller gives suggestions that are comparable to that of the PID ratio controller (point G on figure 8.1). The plant-model mismatch was considerably less for Model 2 as can be seen by the smaller values of the feedback error (e) in figure 8.9. This error however needs filtering to remove the effect of small plant disturbances. From figure 8.9 it is clear that the value from the predictor model (point B) is having the greatest influence on the feedback error (e), since the same inverse shape in the feedback error curve is observed.

A plant disturbance is apparent at about midnight (figure 8.9), when both the neural network models react to a change in the plant conditions, however little change occurred to the PID setpoint.

The IIMC suggestions for the hexene : ethylene ratio shown in figure 8.10 showed a drastic improvement from figure 8.8. This is due to the feedback error (e) being filtered with an alpha value of 0.5 ($\alpha$ = 0.5). The IIMC suggestions (point D) are very similar to the PID controllers' (point G) and appear to have been smoothed the right amount. There are no large spikes. The only anomaly occurs when a large plant disturbance occurs at about 6:00 am. This same disturbance is evident in all of the remaining figures in this chapter. After examining the plant data it was concluded that this disturbance was caused by quite a sudden large decrease in the ethylene flow rate to the reactor.

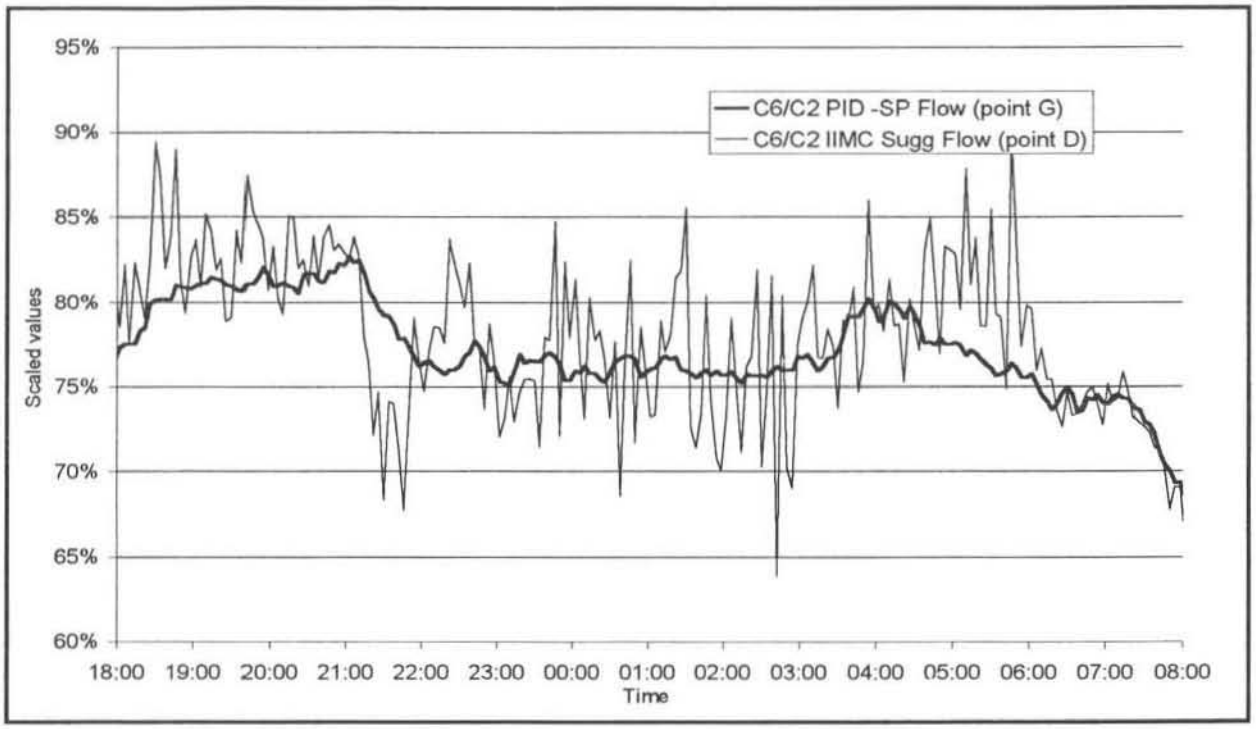


*Figure 8.8: $C_6/C_2$ Flow Ratio SP vs Suggestions (Feedback Error - $\alpha$ = 1)*

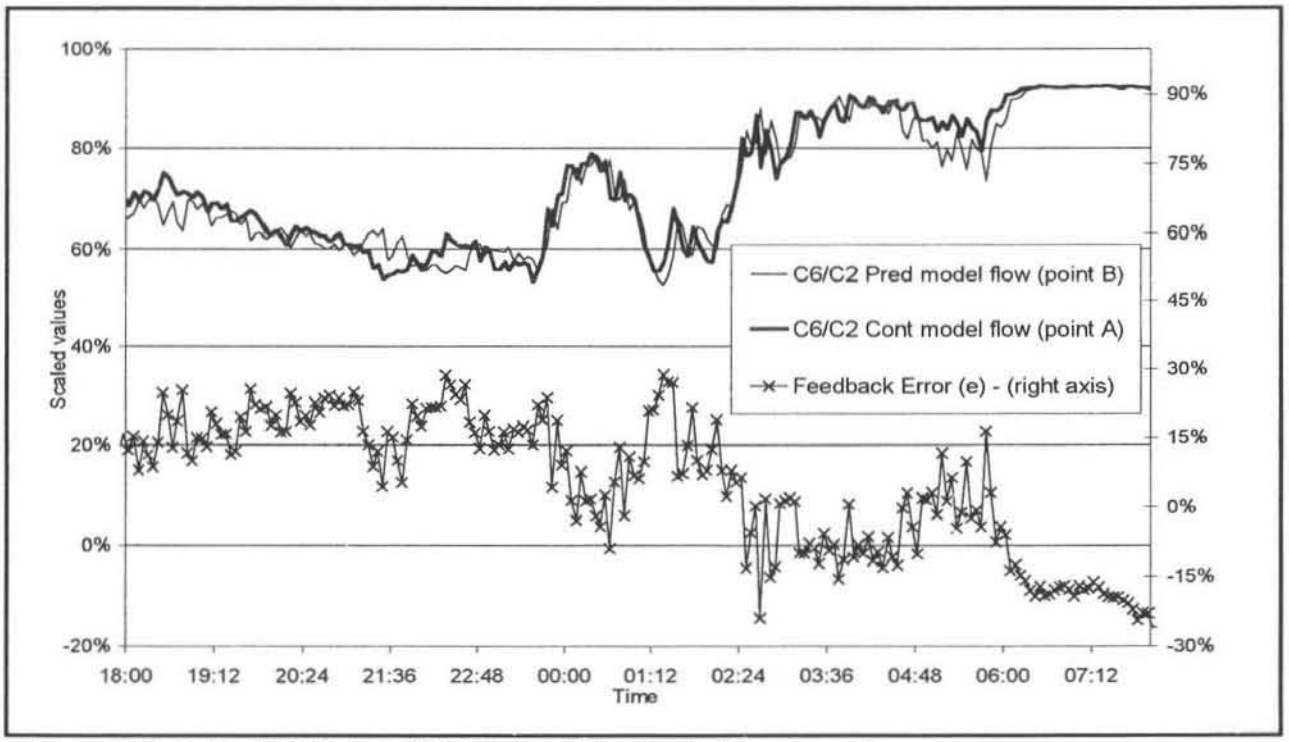


*Figure 8.9: $C_6/C_2$ IIMC Internal Values and Feedback Error ( $\alpha$ = 1)*

However this is an example where the PID controller did not react to the disturbance, because it has no direct reference to it. It is plausible that the suggestion from the IIMC controller would in fact have resulted in the plant rejecting the effect of the disturbance in a shorter time than the PID controller. This is why there is a need to test the IIMC controller (figure 8.1) in a closed loop control scheme.

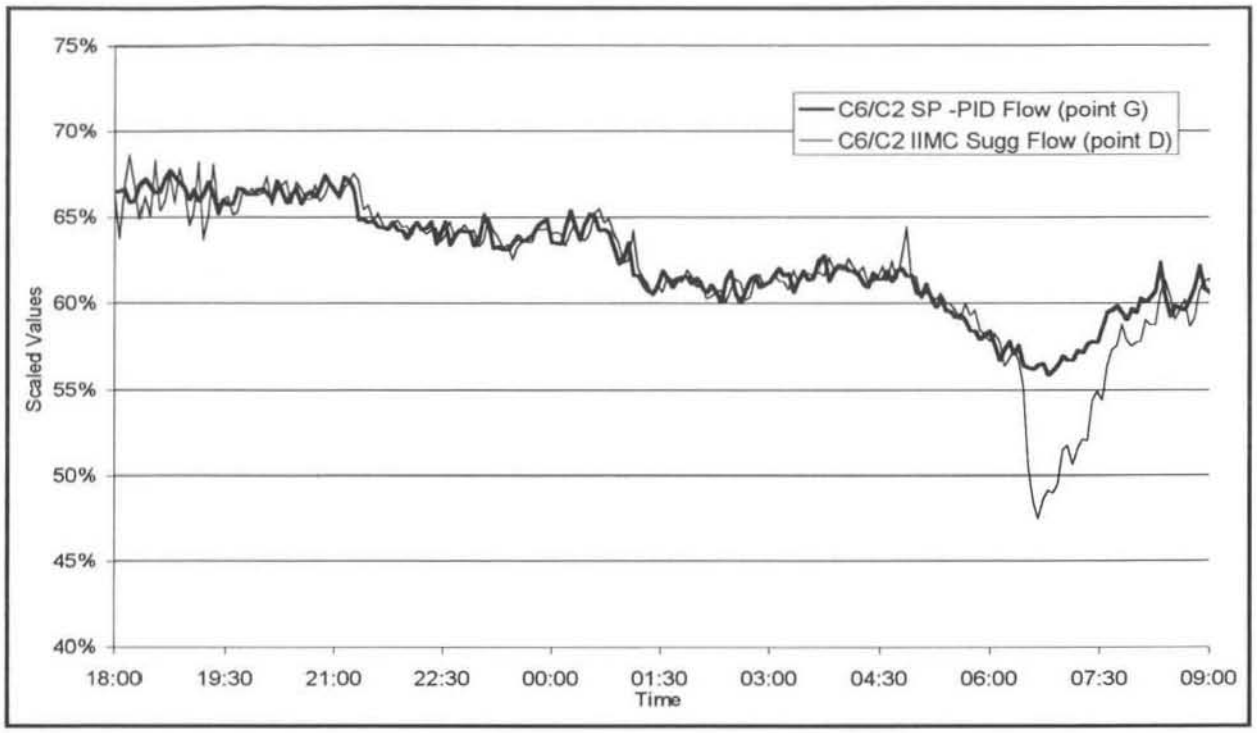


*Figure 8.10: $C_6/C_2$ Flow Ratio SP vs Suggestions (Feedback Error - $\alpha$ = 0.5)*

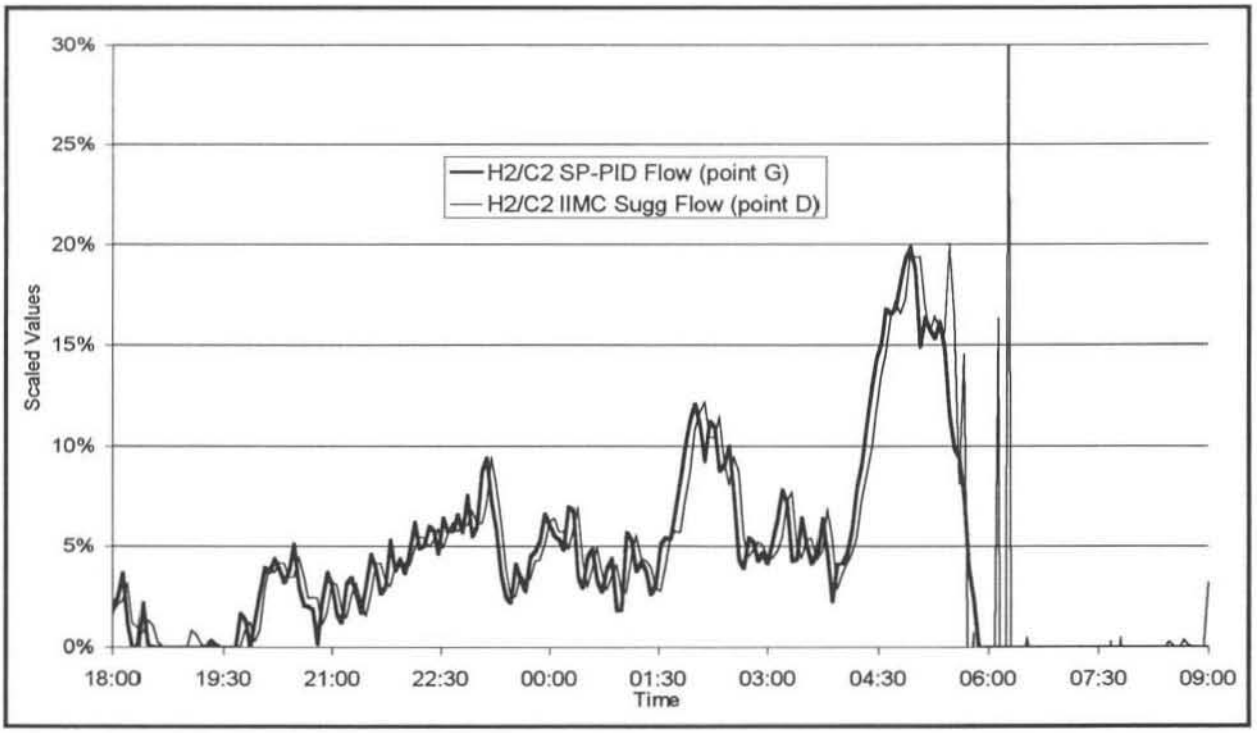


*Figure 8.11: $H_2/C_2$ Flow Ratio SP vs Suggestions (Feedback Error - $\alpha$ = 0.5)*

The IIMC suggestions of the hydrogen : ethylene flow ratio, with no filtering ($\alpha$ = 1) of the feedback error, is not shown. The IIMC suggestions (point D on figure 8.1), using the filtered ($\alpha$=0.5) feedback error, are shown in figure 8.11. The suggestions are quite similar to the PID setpoints (point G) until the same plant disturbance at 6:00am disrupts the IIMC controller, although perhaps these were better suggestions than those of the respective PID controller. The plant-model mismatch for the hydrogen ratio was significantly larger than that of the hexene

ratio (hydrogen : ethylene feedback error (e) curves not shown). This result is the same as in Model 1 where difficulty was encountered in accurately suggesting reasonable hydrogen flow rates. However these results on a whole show that Model 2 does hold potential as a possible neural network controller model.

### 8.2.3 Model 3

Model 3 was the time-invariant dynamic neural network model that showed the most promise during training and testing (section 7.2.3.2). The structure of Model 3 is shown in figure 7.18. The tests using this model in the IIMC controller were however not completed. Before all the tests with various settings for the filtering of feedback error could be completed the plant was shut down. Upon start-up a completely different grade was manufactured that could not be tested and time constraints for this dissertation unfortunately meant that it was not possible to obtain these results. The results that were obtained are shown in figures 8.12 and 8.13.

Figure 8.12 shows the IIMC controller suggestions (point D in figure 8.1) compared with the PID setpoint (point G). The feedback error (e) is unfiltered ($\alpha$ = 1) here and that is why the suggestions are "jumpy". Previously, when the feedback error was filtered the IIMC suggestions became much smoother. Thus it is expected that filtering here would improve the quality of the IIMC suggestions. The same plant disturbance that was evident in figures 8.10 and 8.11 is visible here at about 6:00am. Again the IIMC controller's suggestion differs from that of the PID controller. It is not possible to determine which is the better prediction. However there may be a need to limit the maximum step size that the controller can take to suppress sudden large changes that may arise from time phasing errors.

From figure 8.13 it can be seen that the plant-model mismatch, seen from the feedback error (e), is the smallest of the 3 models tested thus far. The feedback error was close to zero up until the plant disturbance at 6:00am. The feedback error (e) curve again had the same inverse shape as that of the neural network predictor model values (point B on figure 8.1). The feedback error (e) curve is fairly smooth already and the predictions would improve by assigning a value (less than 1) to the alpha ($\alpha$) value. As an initial estimate alpha would be made equal to 0.5.

The IIMC suggestions for the hydrogen : ethylene ratio (figures not shown) were similar to those for Model 2. The suggestion of accurate control actions for the hydrogen ratio proved to be more difficult than that of the hexene ratio. By filtering the feedback error (e) term it is hoped that the IIMC suggestions would improve. This dynamic model (Model 3) also showed potential for use in a neural network controller, although the tests were inconclusive.

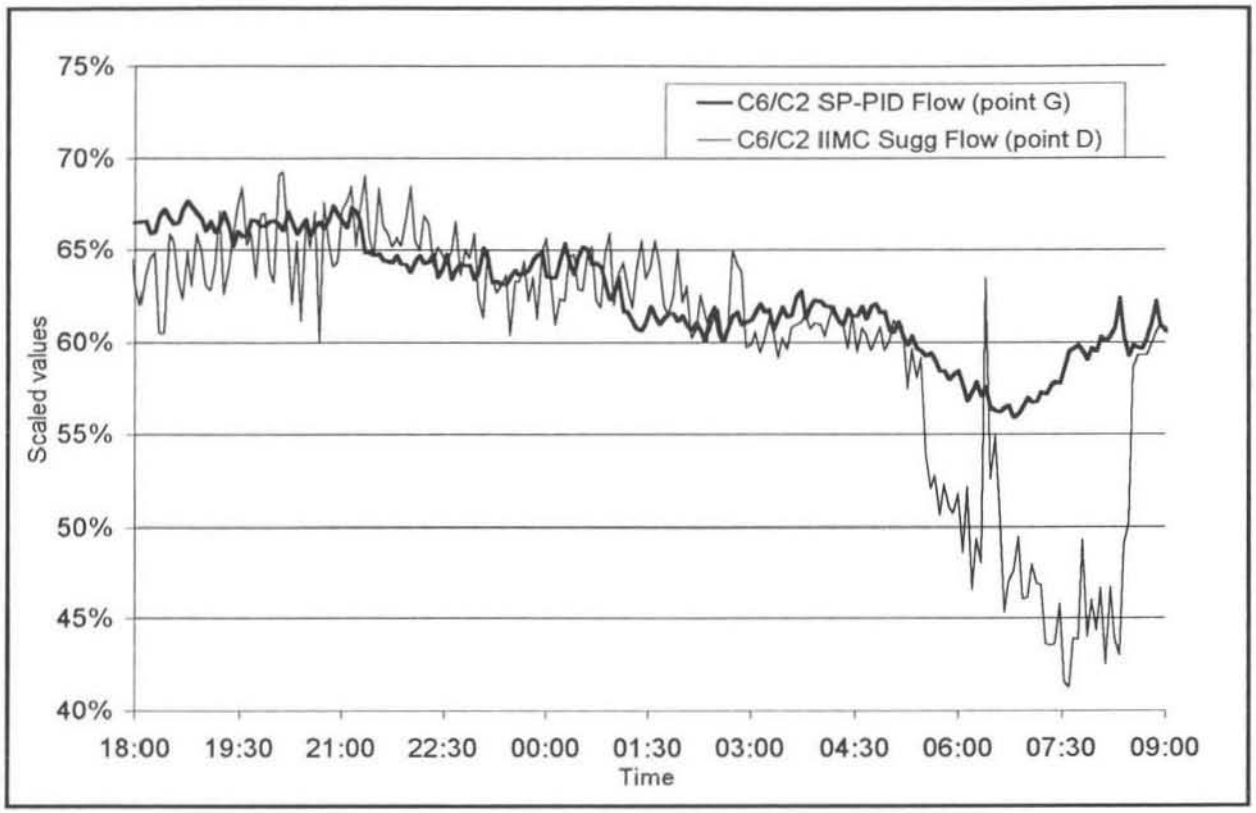


*Figure 8.12: $C_6/C_2$ Flow Ratio SP vs Suggestions (Feedback Error - $\alpha$ = 1)*

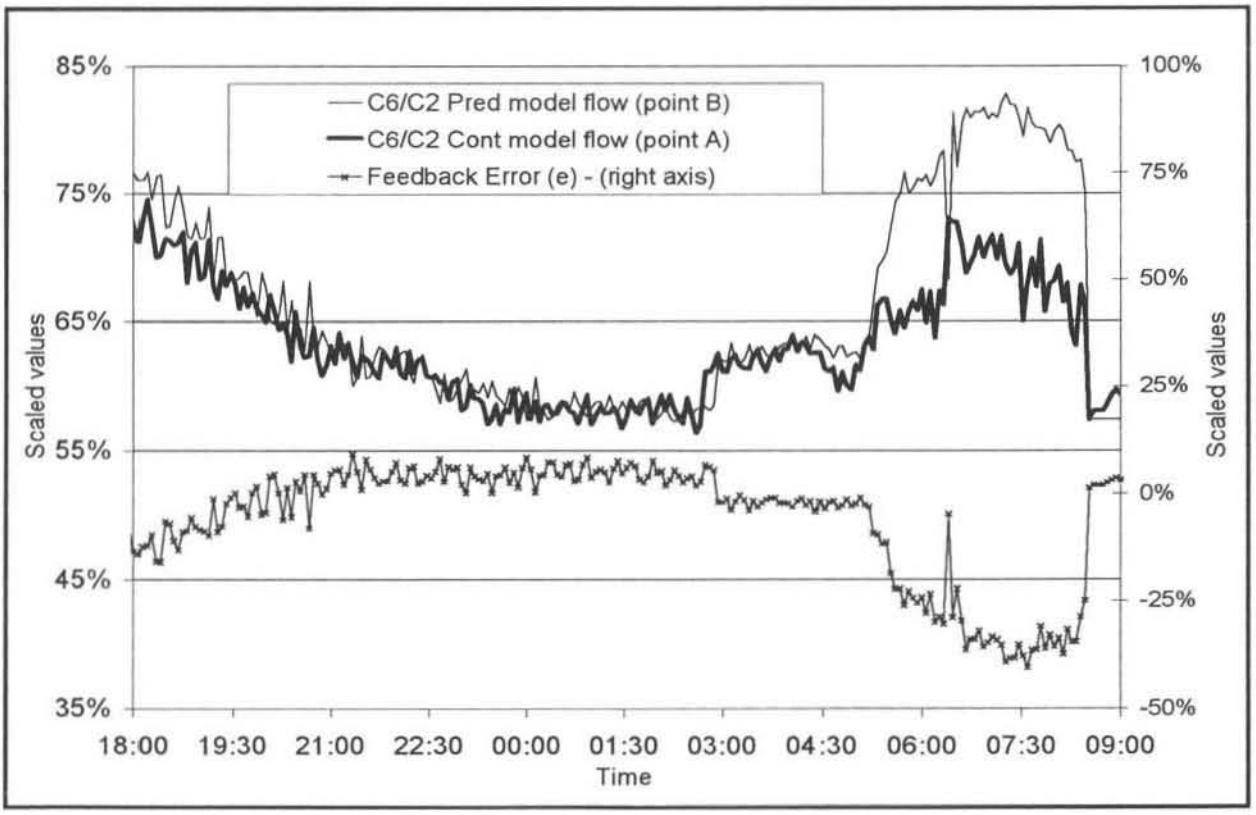


*Figure 8.13: $C_6/C_2$ IIMC Internal Values and Feedback Error ($\alpha$ = 1)*

## 8.3 Discussion of Tests

Model 4, as shown in figure 7.22, was tested on-line (open loop), although the results have not been presented here. It was found that this model's performance was the poorest of the 4 models tested. It was concluded that this was because not enough information was given to the model regarding the state of the reactor. The plant-model mismatch was significant and the feedback error could not account for this large discrepancy, as it did the other models.

To choose the best model is difficult from the results obtained. Model 1, although having excellent suggestions of the hexene flow rate, fell very short in its hydrogen flow rate suggestions. Model 3 showed promise but the results did not allow for any definite conclusions. The plant-model mismatch for the hexene monomer flow rates was the smallest for this particular model. The results from these tests show that Model 2 did produce the most stable and reasonable flow ratio suggestions, although Model 3 may have the same potential.

One conclusion that can be drawn from these results is that including information on the current plant conditions when training and using the neural network models increases the model's chances of giving accurate predictions. Although Model 2 was a time-invariant algebraic model and Model 3 a time-invariant dynamic model, the common feature between the 2 models was that they included variables such as temperature, production rate and pressure as inputs to the neural network models. It is evident that to obtain an accurate model of the desired relationship between the gas compositions and the input feed rates of monomer and co-monomer, some of the important operating conditions in the reactor need to be considered. This is even more clear if one considers the poor results obtained from model 4, which considered only the flow : composition relationships.

# CONCLUSIONS

This final summary of the work presented in this dissertation is divided up into the three main objectives that were mentioned in the introduction:

1) Familiarisation with the identification and modelling techniques employing Artificial Neural Networks (ANN).

2) Testing the performance of the Inverse Internal Model Control (IIMC) scheme in a separate case study.

3) Modelling of the polymer reactor using artificial neural networks, and applying these models in an open loop on-line IIMC controller.

1) An artificial neural network is a mathematical "black-box" non-linear modelling tool. The theory regarding one of the basic types of these neural networks, back-propagation or feedforward multi-layer networks, was covered. From this theory several models representing different processes were developed. Both forward and inverse process models were considered. These models included both time-invariant dynamic and time-invariant algebraic systems. The dynamic model is one in which the past trajectory of the plant variables is considered to enable the model to learn the system time dynamics. A time-invariant algebraic model does not include these time dynamics and behaves more as a sort of multi-dimensional reference table of the steady-state process operating conditions.

   Several software programs were also developed that aided in validating the neural network models. These included producing graphs which showed the model's response to input perturbations - these were called "sensitivity plots". Another program produced multi-step predictions from forward process neural network models. This procedure was known as "chaining". These techniques were successfully demonstrated by creating a model of a pump-tank system that was represented by a non-linear set of equations.

2) The theory of Model Predictive Control (MPC) was introduced and briefly discussed in the early chapters. This led up to the presentation of a novel form of the traditional Internal Model Control structure which was termed IIMC. This IIMC structure included an 'inverse' internal process model of the system. The benefits of using this IIMC structure when compared to traditional IMC seemed to be very favourable. The main reason for this was that only 1 model of the process was necessary and the possibility for error between the models was zero since they were identical. This controller was tested on a real pump-tank system. The controller was able to accurately control two

interacting tank levels to their setpoints. The filtering of the feedback error in the control loop was the main tuning variable available and semi-optimum settings were found for these parameters. With more time these filter settings could possibly be improved. Two Dynamic Matrix Controllers (DMC) were also tested on the system, and the results for the IIMC and DMC controllers were very similar.

3) Several inverse models of the reactor were developed. These ranged from simple time-invariant dynamic models to large time-invariant algebraic models. The most successful time-invariant algebraic (Model 2) and time-invariant dynamic (Model 3) models gave very similar results when compared using the sensitivity plots. These models appeared to have learnt the behaviour of the system well and were able to suggest control actions that would result in process moving towards the relevant setpoint and eliminating process disturbances. The most important result from this modelling was that the models needed to take the process dead time of 50 minutes into account. In another test it was found that a simple "time-invariant algebraic" neural network model of the reactor took the process non-linearities and coupling into account effectively. This was compared to two non-coupled equations representing an approximation of the relationship between the respective flow ratios and compositions in the reactor and it was clear that the neural network model of the system was better.

The neural network model based IIMC structure was programmed into the on-site Distributed Control System (DCS) at the polymer plant. The different controller models developed were tested on-line with real plant data. However the control loop was not closed (open loop). The actual reactor feed flow rates, as suggested by existing PID controllers, were compared with the IIMC controllers' suggested values. In the case of the most successful models these two sets of values were quite similar. However without extensive testing, including closed loop trials, it is difficult to determine if in fact the IIMC suggestions are better.

# RECOMMENDATIONS

- The neural network based IIMC definitely holds promise. It is simple, relatively easy to implement, and should be researched further. It is thought that the performance of this controller in the pump-tank case study could be improved using a shorter sampling interval.
- Before any further action can be taken to close the loop on the polymer reactor a series of planned tests needs to be performed. The small-scale pilot plant would be an ideal place to test the IIMC controller's performance and compare it with the existing PID controllers.

# REFERENCES


Basic ModelGen ™ 1.06, Data Analysis and Modelling Tool that uses the Latest Artificial Intelligence Technology, Crusader Systems, 1998.

Bhat, N.& McAvoy, T.J. (1989), "Use of Neural Nets for Dynamic Modelling and Control of Chemical Process Systems", *Computers and Chemical Engineering,* 14(4/5), 573-583.

Chouai , A (1999), "Application Des Reseaux De Neurones A La Modelisation Et A La Commande Multivariable Des Collones D'extraction Liquide-Liquide"., Toulouse.

Cybenko, G. (1989), "Approximation by Superposition of a Sigmoidal Function." *Math. Of Cont., Sig. and Syst.* , Volume J(11): 1501-1515.

de Vaal, PL (1999), "An overview of advanced control techniques", Department Of Chemical Engineering, University of Pretoria, Chemical Technology Sept/Oct 1999, p13 – 19.

Davidovici, G. (2000), "The World Of Polyethylene and its Competitive Position", *Presented to a Knowledge Sharing Workshop at Sasol Polymers Polythene Division*, July 2000, Sasolburg, South Africa.

Dayal, B.S., Taylor, P.A., MacGregor, J.F (1994), "The design of experiments, training and implementation of non-linear controllers based on neural networks", *Can J Chem. Engng*, 1994; 72, p 1067 – 1079.

Dirion, L (1995), "Design of a neural controller by inverse modelling.", *Comput Chem Engng*, 1995; 19S:S797 – S802.

Dubois, O. (1994), "Adaptive neural network control of the temperature in an oven", Colloquium on Advances in neural networks for control and systems, Vol. 8, 1994:1 – 3.

Evans, J.T. (1993), "An online application of a neural network based predictive control strategy", Proc. IEE Colloquium Nonlinear Contr., 1993; 12:1 – 4.

Foster, G. (1991). Polymer Reaction Engineering Course. Hamilton, Ontario, McMaster University.

Garcia, C.E., Prett, D.M., Morari, M (1998), "Model Predictive Control: Theory and Practice-A Survey", *Automation,* 1989,Vol. 25, No. 3, p335 –348.

Guimba, I., (2000), " Application of Dynamic Matrix Control to an Interacting Tanks System", M.Sc. Eng. Dissertation, University of Natal, Durban, 2000.

Hornik, K., M. Stinchcombe (1989). "Multilayer Feedforward Networks are Universal Approximators." *Neural Networks* 2: 359.

Hussain, M.A. (1999), "Review of the Applications of Neural Networks in Chemical Process Control – simulation and on-line implementation", *Artificial Intelligence in Engineering*, 1999, Vol 13, p255 – 69.

Hussain, M.A., Kershenbaum , L.S (2000), "Implementation of an Inverse-Model-Based Control Strategy using Neural Networks on a Partially Simulated Exothermic Reactor", *Trans IChemE*, Vol 78, Part A, March 2000, p299 – 311.

James, D. E. (1986). Encyclopaedia of Polymer Science and Engineering. J. I. Kroschwitz. New York, John Wiley & Sons. V:6 p 429.

Keeler, J., Martin, G. (1997), "The Process Perfector – the next step in multivariable control and optimisation", Technical Report, Pavilion Technologies Inc., Austin, TX.

Khalid, M., Omatu, S. (1992), "A neural network controller for a temperature control system", IEEE Contr. Syst. Mag., 2000, p58 – 64.

McAuley, K.B., MacGregor, J.F., Hamielec, A.E. (1990) "A kinetic model for industrial gas-phase ethylene copolymerisation", *AIChE Journal* , 36(6), 1990, p837-850.

Michalovic, M., K. Anderson, et al. (1996). "The Macrogalleria - a cyber wonderland of polymer fun ".

Nahas, E.P. , Henson, M.A., Seborg, D.E. (1992), "Non-linear internal model control strategy for neural networks models. ", *Comput. Chem. Engng*, 1992; 16(2), p1039 – 1057.

Narotam N. K. (1999), "Development of Artificial Neural Networks for the Modelling and Control of Chemical Processes" , M.Sc. Eng. Dissertation, University of Natal, Durban, 1999.

Pike, A.W. , Grimble, M.J. , Johnson, M.A. , Ordys, A.W. , Shakoor, S (1996)," CRC Control Handbook" , Chapter 51 – Predicitve Control, Edited by Levine, W.S. , Industrial Control Centre, University of Strathclyde, Glasgow, Scotland, U.K.

Royle, B., Rackham, B. (1998),"Polifin's Hexene-based PE-LLD" , Technical report , *Plastics SA*, 1998, October, p. 16-20.

Saint-Donat, J., Bhat, N., McAvoy, T.J. (1991) "Neural network based model predictive control", *International Journal of Control*, 1991, Vol. 54, p. 1453-1468.

Sbarbaro, H.D. , Neumarkel, D. , Hunt, K. (1993), "Neural control of a steel rolling mill" , *IEEE Control Systems* , 1993 ; 13(3) :631-640.

Seborg, D.E. (1994), "Experience with nonlinear control and identification strategies.", IEE Control Conference, 1994, p879 – 886.

Schumacher, J.W. (1996), "LINEAR LOW-DENSITY POLYETHYLENE (LLDPE) RESIN",*CEH ABSTRACT* , http://www.cmrc.sri.com/CIN/MayJune96/Article07.html

Thomason, R. (2000), "Real-time Observer Modelling of a Gas-phase Ethylene Polymerisation Reactor" , M.Sc. Eng. Dissertation, University of Natal, Durban, 2000.

VanCan, H.J., Bracke, H.A. (1995), "Design and real-time testing of a neural network predictive controller for a non-linear system" , *Chem. Engng. Sci.*, 1995, Vol. 50, p. 2419-2430.

# APPENDIX A

## TESTING/VALIDATION PLANT DATA

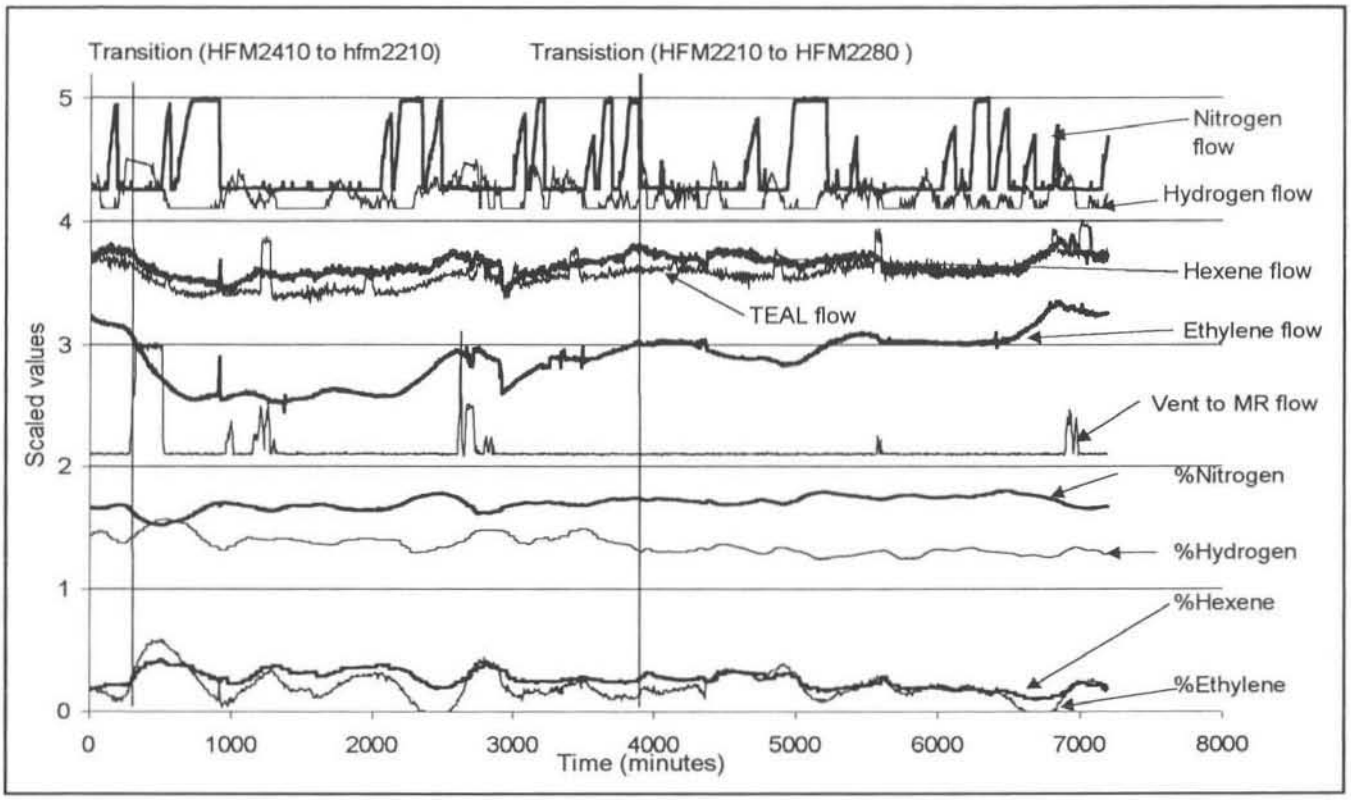


*Figure A.1: Scaled Plant Data (A) used for Testing Neural Network Models*

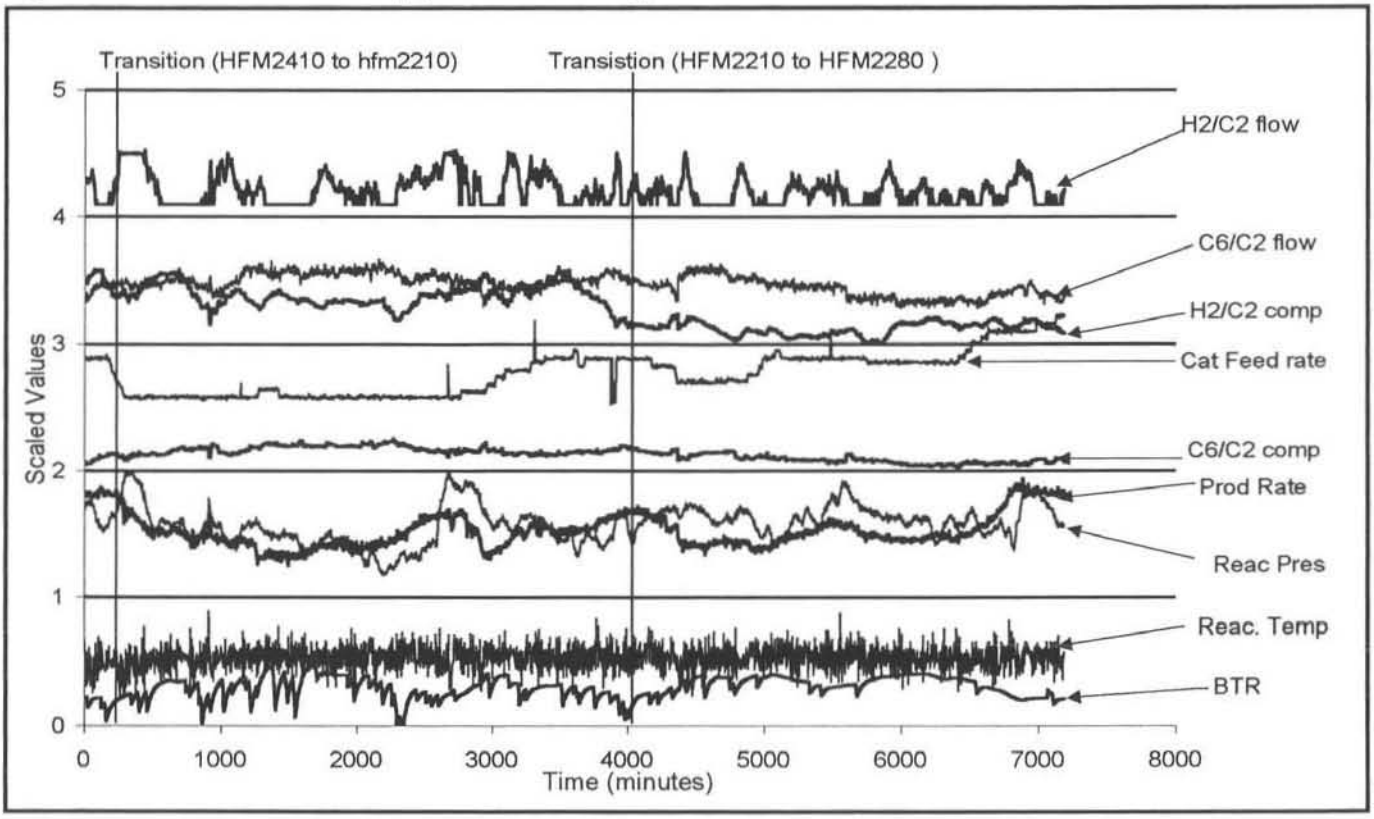


*Figure A.2: Scaled Plant Data (B) used for Testing Neural Network Models*

# APPENDIX B

## PLANT VARIABLES USED FOR MODELLING

| | Name | Units |
|---|---|---|
| 1 | Reactor Temp | Deg |
| 2 | Reactor Pres | kPa (g) |
| 3 | BTR | $h^{-1}$ |
| 4 | Vent Flow | $Nm^3$ / h |
| 5 | Vent to MR | $Nm^3$ / h |
| 6 | Combined Catalyst feed Rate | Kg / h |
| 7 | Production rate | t/h |
| | | |
| 8 | Ethylene Composition | % |
| 9 | 1-Hexene Composition | % |
| 10 | 1-Butene Composition | % |
| 11 | Hydrogen Composition | % |
| 12 | Nitrogen Composition | % |
| | | |
| 13 | Ethylene Flow | Kg/h (gas) |
| 14 | 1-Hexene Flow | Kg/h |
| 15 | 1-Butene Flow | Kg/h |
| 16 | Hydrogen Flow | Kg/h (gas) |
| 17 | Nitrogen Flow | $Nm^3$ /h |
| 18 | TEAL Flow | Kg/h |
| | | |
| 19 | $C_6/C_2$ Flow ratio | |
| 20 | $C_4/C_2$ Flow ratio | |
| 21 | $H_2/C_2$ Flow ratio | |
| | | |
| 22 | $C_6/C_2$ Composition | |
| 23 | $C_4/C_2$ Composition | |
| 24 | $H_2/C_2$ Composition | |

*Figure B.1: Plant Variables used for Modelling*

# APPENDIX C

## APPENDIX C1 - EXPLANATION OF THE ON-LINE CODE

The code for the IIMC controller using artificial neural network models was originally developed and tested in MATLAB. The code was then translated into C++ for use in the Visual C package MSDEV (Microsoft Developer Studio). It was used in this form for the on-line tests of the interacting tanks system, section 6.2.

The process of programming the code into the polymer plant DCS was non-trivial. The author is extremely grateful to the polymer plant control engineers for their help and expertise in this matter.

The layout of the DCS was shown in figure 2.8. The operating platform used in this system was UNIX. The code was written on one of the Application Programming Interfaces and stored in files in a project folder. As can be seen from figure 2.8 all the plant data goes to an RTAP database. For the controller to use on-line 'live' plant data the code had to be able to access this database. Most of the code written actually deals with getting this information from the database and storing it in local memory so that it can be used and manipulated during the calculations. Since the controller loop was not closed the proposed flow ratio setpoints were never written back to the Plant database.

Provision was made to run three controllers simultaneously incorporating different neural network models in them. This was done as it was thought that it might help with validating the different neural network models. The suggestions of the three controllers can be compared with the setpoint from the PID controllers.

As is clearly evident from figure C.1 the program was written in an Object Orientated fashion. The use of structures and special classes of functions was common. The majority of the code dealing with accessing the databases and the timing functions was written by the polymer plant control engineers. The code dealing with the IIMC neural network controller was written by the author. Since time constraints were quite severe most of the code written was not done so in the most efficient manner. From a computer science point of view this code would be regarded as a bit bulky. However from an engineering point of view the code works fine and is easily understood.

# APPENDIX C2 – EXPLANATION OF THE CODE IN THE DIFFERENT FILES

The large combined size of these files has necessitated that only the most relevant sections of the code are included in the Appendices. Brief descriptions of the function of each of the files shown in figure C.1 follow.

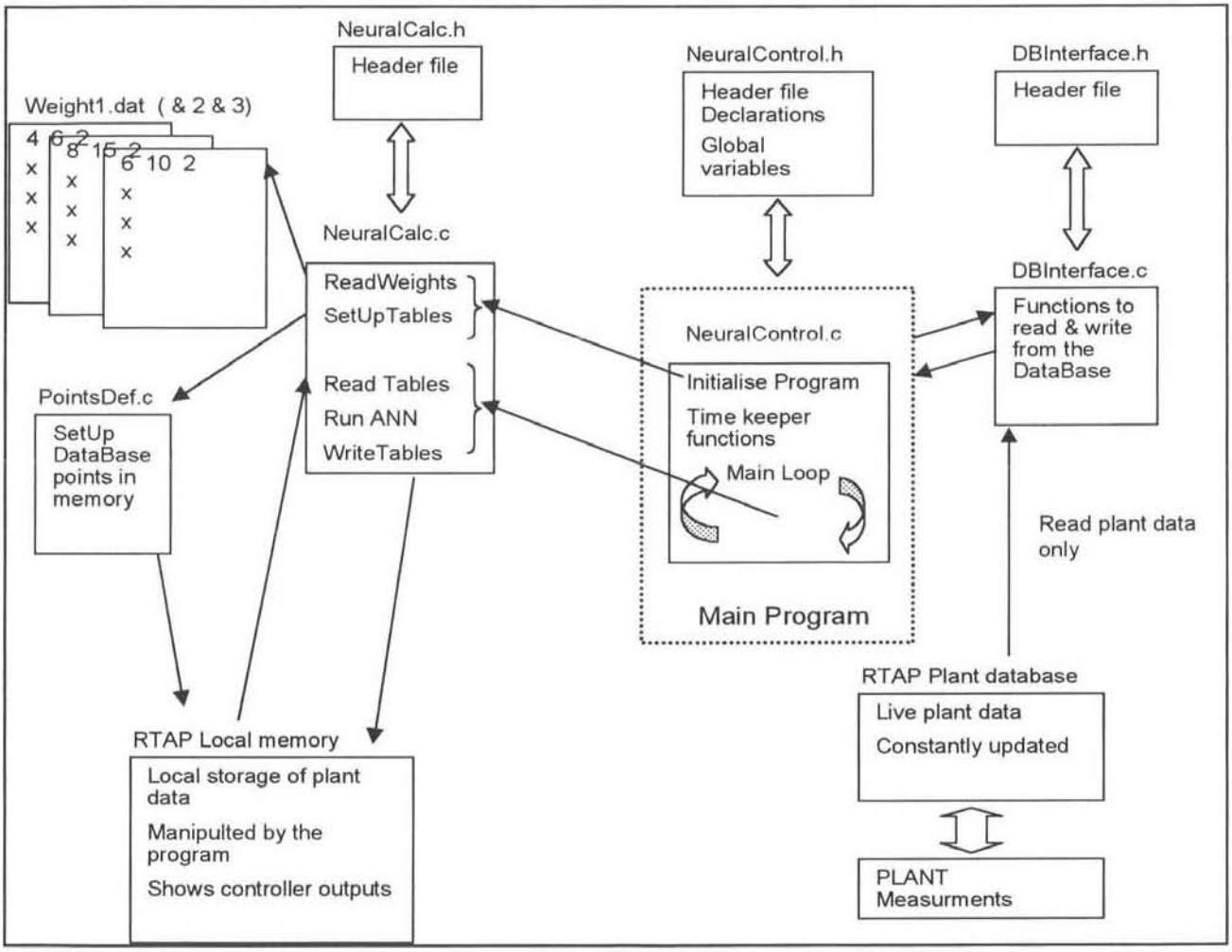


*Figure C.1: Structure of the IMC code programmed in the DCS*

### NeuralControl.c (Appendix D)

This is the main file in the program. The compiling and running of programs in this UNIX environment is slightly different from a Windows Language Compiler (eg. MSVC++) or Matlab. A separate makefile (NeuralControl.o) which tells the compiler what to use as the main function code is necessary and to set the level of debug. This level of debug allows the user to compile the code in different ways in order to pick up errors in the syntax, structures etc.

The main purposes of this file are the following:

- To initiate the program and the relevant timekeeper functions. These functions ensure that the program runs smoothly and is kept connected to all the relevant databases and drives. The timer for the main loop is set here, on line 168.
- The program initialisation is done here (line 157).

  This function is discussed in the under the NeuralCalc.c file.
- The contents of the main loop are shown in lines 187 – 202.

  Basically the loop consists of the following functions:
  - ReadTables: Read the current values from the tables (opens the tables for further use)
  - SetUpTables: Write the new plant data to the tables and cascade the old values down
  - RunNeuralNet: Run the neural network controllers
  - UpdateTables: Update the tables with the controller suggestions

The rest of this code in this file was written by the Polymer plant control engineers and works fine, which is enough incentive to leave it alone.

### NeuralControl.h (Appendix E)

The header file in this UNIX environment has the same functions as the normal header file in the Windows C language. The main areas of interest in this file are the structure declarations for the local database points, which are discussed later in Appendix A3.

### NeuralCalc.c (Appendix F)

This is by far the largest of the files. Its total length is exceeds some 2000 lines (2213 to be exact). This is not however a reflection on its complexity. The file primarily contains all of the functions, which are called from the NeuralCalc.c file, and a few other local functions. The reason for it being so long is simply that all the functions are in triplicate, since there are three distinct types of local database points, see Appendix C3. Also three different controllers were designed to accommodate different neural network model structures and these were also in triplicate. The file could have been made a lot shorter but since this was an exercise in advanced control and not computer programming the file remains as is. Time constraints also played a large part here.

The important areas of code and functions are summarised below. Remember that most of the functions shown actually have three different versions for the different database structures.

- Lines 44 – 70: The local structures for the database points are instantiated here and given specific local names. These are the points that are in the local memory.

- Lines 91 – 135: This is the definition of the Initialisation function called in NeuralControl.c. This function reads the weights from text files to the local memory space. It also inserts the relevant current plant values into the local tables (i.e. all the rows). This is so that there are no abnormal values when the controller starts up. Obviously one has to take this into account when considering the controller results. With dynamic models it will be a while before the model is seeing 'true' past values that have cascaded down.

- Lines 137 – 146: This is an example of a cascade function. This is where the current values are moved down the rows at the beginning of each loop.

- Lines 148 – 241: This is an example of a Set Up Function that reads the current plant values and writes them to the local variable structure. It can also be seen how these plant values are filtered.

- Lines 245 – 269: The Read and Update table functions are fairly straightforward. They use Read and Write functions that were developed by the Polymer plant engineers for reading from and writing to database points.
- Lines 275 – 314: This is an example of a GetWeights function. It reads the weights from a text file into the local memory.
- Lines 316 – 813: this is an example of a RunNeuralNetwork function. Again this is not complicated, it is just long due to inefficient programming on the part of the author. It can be broken down into simple parts. It is also easier to understand when compared with the diagram of the controller in figure 5.4.

- Lines 316 – 357: Declarations of local vectors for the function.

- Lines 390 – 445 & 526 – 579: This is where the inputs to the two neural network models are set up. The raw plant values are scaled according to the scaling factors in the tables (see Appendix A3) and clipped so that they do not exceed the range [0.01 to 0.99].

- Lines 449 – 498 & 582 – 628: This is the actual neural network calculation, node by node. It uses several nested for loops to calculate the summed value on each node and then uses

the sigmoidal transfer function. This code has been tested and validated dozens of times. The last section of these two bits of code is where the outputs from the neural network are assigned to the relevant local memory positions and re-scaled.

- Lines 636 – 808: This is where the IMC error and filtering calculations are performed. The 'predval' (see Appendix A3) neural network values are compared with old plant values to calculate the error term (e), [line 738]. This is filtered and added to the 'contval' prediction to obtain the 'suggval' value (lines 741 –746). The suggested values are then clipped and ramp-limited if excessive changes are suggested. Finally the values are assigned to the local database points.

### NeuralCalc.h (Appendix G)

This file is quite small. The main function of this file is for the prototypes of functions declared in the NeuralCalc.c file that need to be external (extern) so that they can be called in NeuralControl.c.

### PointsDef.c (Appendix H)

All variables that are called from or written to the databases (local and plant) need to have a point defined in this file that tells the Read and Write functions what the structure of the information will be. There are 71 points in this file. Examples of 3 of them are given. The first point is a flag that indicates if the connection with the plant database is still alive.

The second point is the reactor bed temperature that is read from the plant database.

The third point is the local variable of the reactor bed temperature. The differences between the two are subtle, but confuse the two in the code and many hours of debugging will be necessary!

### DBInterface.c & DBInterface.h

These files were not edited at all. They were created by the polymer plant control engineers. They contain the functions to Read and Write values from the databases and were working fine. As such they were wisely left alone.

# APPENDIX C3 –THE STRUCTURE OF THE DATABASE POINTS

As shown in figure C.1 above local memory points for some variables were created which could be manipulated by the program, to test the controller.

These points were divided up into three main categories.

- Simple **Process** Variables
- Gas Compositions in the Reactor – Analyser values
- **Manipulated** variables – Flows into the reactor

A total of about 17 variables are read in from the actual plant data. These variables are coming straight from the plant and as such may contain sensor and other types of noise signals. The facility exists in the program to use a single exponential filter on these values before they are read into the local variable tables. The smoothed signals may help the neural network models make more accurate predictions and the choice of to use the filter or not is discussed more in chapter 8.

The equation is as follows where $\alpha$ is the filtering constant.

$$plant\_value\_filt(t) = plant\_value(t) * \alpha + (1 - \alpha) * plant\_value\_filt(t - 1)$$

*Equation C.1*

### Process Variables (Figure C.2)

The variables that were grouped as Process Variables were:

- Temperature in the Reactor (oC)
- Pressure in the Reactor (kPa)
- Production Rate (tons/h)
- Bed Turnover Rate (1/ hour)
- Reactor Vent Flow Rate to Monomer Recovery (Nm3/h)
- Reactor Vent Flow Rate to Flare (Nm3/h)
- Combined Catalyst Feed Rate (kg/h)

The basic structure of these variables is shown in figure C.2.

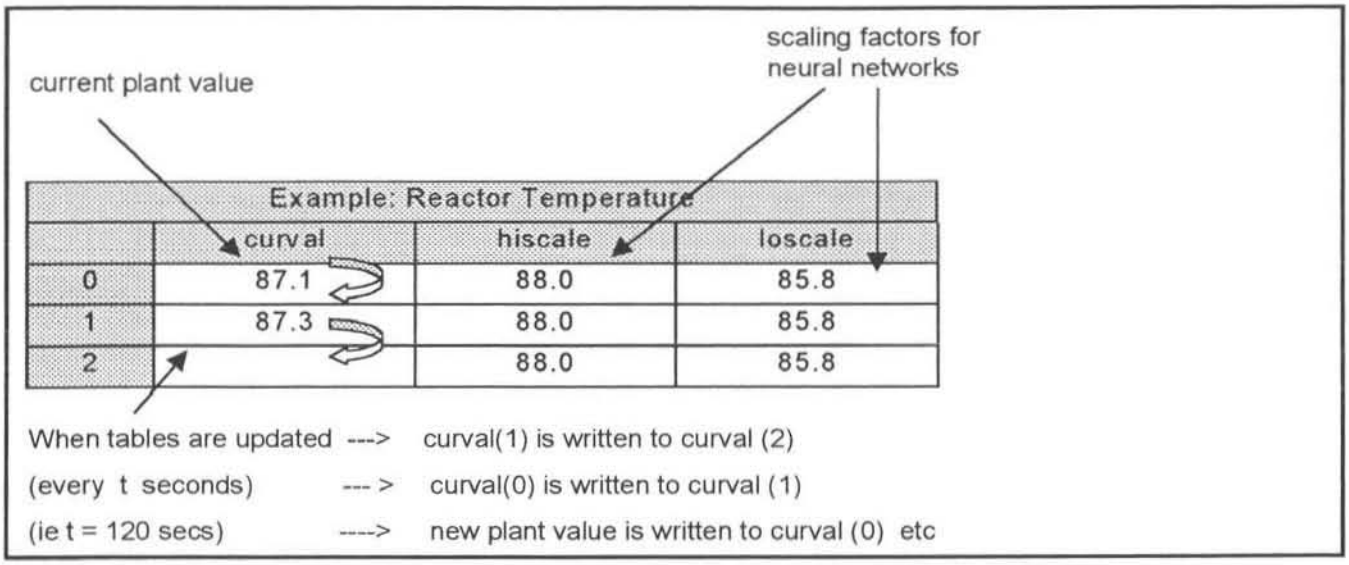


Figure C.2: Structure of the Process Variable Table

Not all the columns are shown. These are the main ones that are important for the calculations. Similarly only the first three rows are shown. The number of these rows went up to 6 for this type of variable.

The 'curval' field is read directly from the plant value every time the main loop in the NeuralControl.c file is executed (every 120 seconds). At this time the other 'curval' fields are cascaded down as shown. Thus the value in the $6^{th}$ row is the plant value from 12 minutes ago (120seconds x 6 rows).

The values in the columns 'hiscale' and 'loscale' do not change. They are entered by the user and are not overwritten. These are the scaling factors that correspond to the values xmax and xmin which are used in equation 4.6. It is important that these scaling factors match the factors used during training of the neural network model so that a normalised range of data is used.

### Gas Compositions in the Reactor (Figure C.3)

These variables were known as the 'controlled' variables since it was their compositions that were being controlled. The variables that were grouped like this were:

Ethylene Gas Composition (%)

1-Butene Gas Composition (%)

1-Hexene Gas Composition (%)

Hydrogen Gas Composition (%)

Nitrogen Gas Composition (%)

1-Butene / Ethylene Gas Composition

1-Hexene / Ethylene Gas Composition

Hydrogen / Ethylene Gas Composition

XL_Butene / Ethylene Gas Composition

XL_Hexene / Ethylene Gas Composition

XL_Hydrogen / Ethylene Gas Composition

The basic structure of these variables is shown in figure C.3.

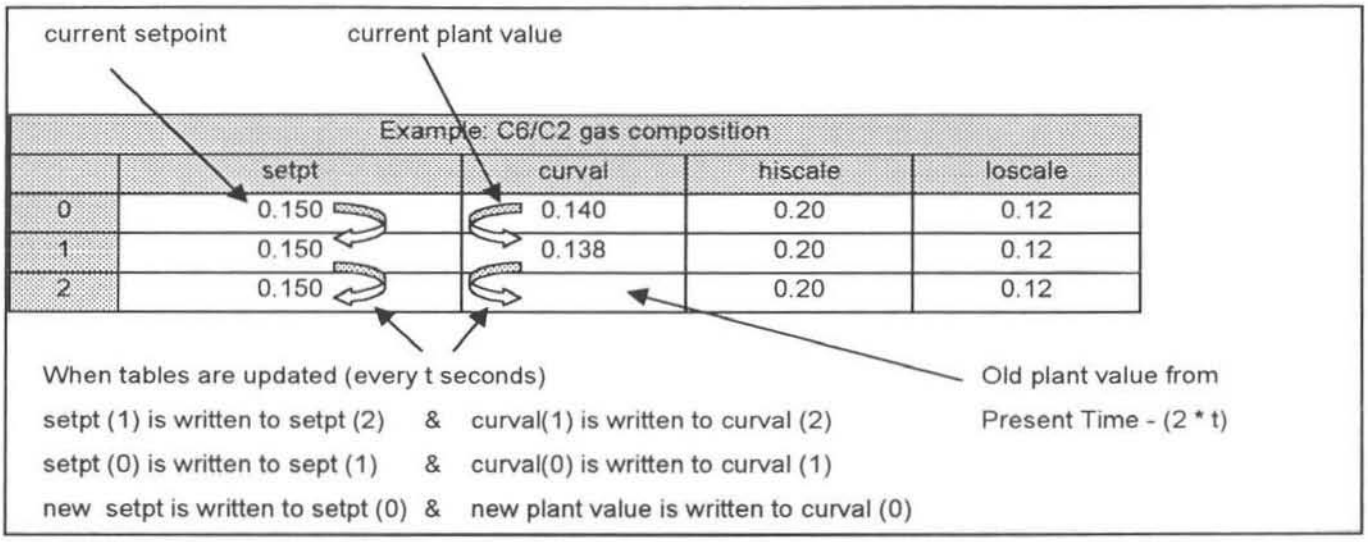


*Figure C.3: Structure of the Gas Composition Table*

Again not all the columns are shown. Also the number of these rows went up to 6 for this type of variable, except for the last three variables mentioned. These variables have the pre-script 'XL' on their names that indicate that the have 50 rows in them. Hence they can store plant data to up to 100 minutes in the past (2 minutes x 50 rows). This was needed for some of the time-invariant dynamic models.

The 'setpt field is the value that the Poly Server Model is predicting, figure 2.10. Most of the neural network models use this setpoint as the 'average' setpoint as discussed in section 4.4.3 and 4.4.4.

The two columns 'setpt' and 'curval' fields are also cascaded down every time the main program loop is executed in the NeuralControl.c file.

### Manipulated variables (Figure C.4)

The variables that were grouped under this name were the flows into the reactor. These variables were known as the 'manipulated' variables since it was by changing them that the compositions could be controlled. These variables were:

| | |
|---|---|
| Ethylene Flow Rate | (kg/h) |
| 1-Butene Flow Rate | (kg/h) |
| 1-Hexene Flow Rate | (kg/h) |
| Hydrogen Flow Rate | (kg/h) |
| Nitrogen Flow Rate | (Nm3/h) |
| TEAL Flow Rate | (kg/h) |

1-Butene / Ethylene Flow Rate (Ratio)

1-Hexene / Ethylene Flow Rate (Ratio)

Hydrogen / Ethylene Flow Rate (Ratio)

XL_Butene / Ethylene Flow Rate (Ratio)

XL_Hexene / Ethylene Flow Rate (Ratio)

XL_Hydrogen / Ethylene Flow Rate (Ratio)

The basic structure of these variables is shown in figure C.4.

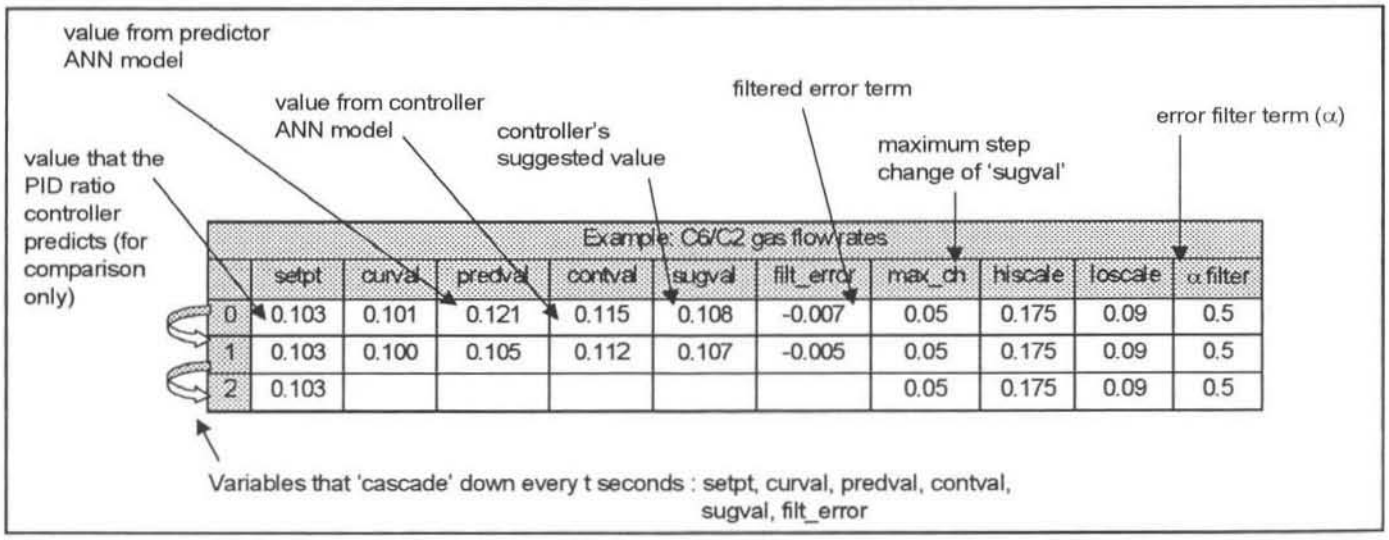


*Figure C.4: Structure of the Manipulated Variable Table*

This structure is slightly different from the previous two table structures. It has all the same columns and the values are cascaded down on every loop execution.

The 'setpt' field in this case is the prediction of the PID ratio controller in figure 2.10 and is only used for comparison with the controller suggested value, 'sugval'.

The main difference is that values from the program are written to this table. The fields 'predval', 'contval', 'sugval' and 'filterror' are written to the table at the end of each main loop execution. The values written to the fields can be seen figure 5.4 to give an idea of what the meaning of each value is.

'predval' – This is the output predicted by the Predictor inverse neural network model (No. 2). It is supposed to represent the prediction of what the plant output should have been 1 time interval ago.

'contval' – This is the value of the output from the Controller inverse neural network model ( No.1). It represents the control action predicted by that model given the plant setpoints.

'filterror' – This is the error term (e) that has been filtered using a single exponential filter, equation 5.1. This error term is the difference between the actual past control action ($u_{t-1}$) and the 'predval' of the past control action. The filter setting ($\alpha$) that is used in this filter is set in the last column '$\alpha$ -filter'.

'sugval' - This is the sum of the filtered error ($e_{filt}$) and the 'contval' prediction. This is the value of the flow rate (or ratio) that the controller is suggesting for the plant. The filtered error term is used to account for the model mismatch between the plant and inverse neural network models. This is the value that has been compared with the PID control setpoint for the flow rates in the On-Line results section, chapter 8.

As mentioned these values are written from the program to the table and cascaded down every loop cycle. These values are also logged to a text file so that the results can be analysed, graphed, and used for improving the filter settings.

The column 'maxch' is the maximum step change that the controller is allowed to make from the previous suggested value. This is a value that the user can set and is not changed by the program.

The last three variables listed with the 'XL' (extra large) prescripts have 50 rows, whereas the other only have 6. This is since some of the time-invariant dynamic models need to have reference to more past flow rates and compositions- in some cases up to 60 minutes previously. All that happens in these 'XL' tables is that there is essentially a larger stack of stored values.

# APPENDIX D (NeuralControl.c)

```
Extracts from NeuralControl.c

/*******************************************************/
/* sources/NeuralControl.c 2000/03/06 N Narotam        */
/*                                                     */
/*  Copyright (c) N Narotam 1999, 2000                 */
/*  All Rights Reserved                                */
/*                                                     */
/* DESCRIPTION                                         */
/*  Source file - This is the main program for the     */
/*  NeuralControl control package.  It cantains all    */
/*  the timing, logging and general housekeeping       */
/*  functions needed by the program.  It also          */
/*  schedules the activities of the program            */
/*******************************************************/
/* COMPILITATION CONTROL                               */
/* 2000/03/30  N Narotam  Ver 1.01 concieved           */
/*                                                     */
/*******************************************************/
#include <cr/crStandards.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include <locale.h>
#include <string.h>

#include <rtap/timekeeper.h>
#include <rtapClasses/rtIgloo.h>
#include <rtapClasses/rtMessages.h>

#include "nrutil.h"
#include "NeuralControl.h"
#include "DbInterface.h"
#include "NeuralCalc.h"

/*************************************************************/
/*              Start of main routines                       */
/*************************************************************/
void main(int argc,char *argv[])
{
int ier;        /* error flag                          */
int i,j;        /* loop counters                       */
int Debug;              /* stores debug level          */
double dummy;           /* What do U think this is?
        */
ApsPV  TestPID;         /* Test Pt for the APACS write utility
        */
rtProcessId *myProcId;  /* process details
        */
char            env[8]; /* enviroment to be used       */
char            mdb[8];     /* master db name
        */
char            sdb[8]; /* Slave DB name               */

/*************************************************************/
/* the following variables are used by the rtap event monitoring*/
/* system the event monitoring system handles all incomming
        */
/* messages and is equavalent to a while(1) loop.  It is however*/
/* more robust.                                        */
/*************************************************************/
rtInt           rtapFd;
rtInstance      MainEventMonitor;
```

# APPENDIX D (NeuralControl.c)

```
65  rtHandler       MainMsgHandler;
66  rtInstance      rtapMsgHandler;
67
68  rtHPCompatibility( argc, argv );        /* make runtime look like
69  HP-UX*/
70  (void) setlocale( LC_CTYPE,    "" );  /* Set up the locales. */
71  (void) setlocale( LC_COLLATE,  "" );
72  (void) setlocale( LC_TIME,     "" );
73  (void) setlocale( LC_MONETARY, "" );
74  #if defined(crIBM_COMP)
75     (void) setlocale( LC_MESSAGES, "" );
76  #endif
77
78  /* set up default command line parameters */
79  Debug = 0;              /* set to limited debug statements    */
80  OnLine = 0;             /* Set to off line for now            */
81  strcpy (env,"\0");      /* Set to current enviroment          */
82  strcpy (mdb,"XXX");     /* Default Master is no Database      */
83  strcpy (sdb,"XXX");     /* Default Slave is no Database
84          */
85
86  /* Now read the command line options */
87  for (i=0;i<argc;i++) {
88    if (strcmp (argv[i],"-e") == 0)
89      strcpy (env,argv[i+1]);
90    else if (strcmp (argv[i],"-d") == 0)
91      Debug = atoi (argv[i+1]);
92    else if (strcmp (argv[i],"-online") == 0)
93      OnLine = 1;
94    else if (strcmp (argv[i],"-m") == 0)
95      strcpy (mdb,argv[i+1]);
96    else if (strcmp (argv[i],"-s") == 0)
97      strcpy (sdb,argv[i+1]);
98  }
99
100 /** First, register for debugging                          */
101 if ((myProcId = rtRegisterForDebug(env, argv[0])) == (rtProcessId
102 *) NULL ) {
103   printf ("%s can't attach to environment to debug\n",argv[0]);
104   exit(1);
105 }
106
107 CurStatus.Debug = Debug;
108 CurStatus.OnLine = OnLine;
109 sprintf (CurStatus.env,"%s",env);
110
111 printf("NeuralControl (c) Predictive Control Application Starting on
112 host %s.....\n",env);
113 printf("All rights reserved - programmed by N Narotam and
114 University of Natal \n");
115 printf("No part of this application may be resold without
116 permission from N Narotam\n");
117 if (OnLine)
118   printf ("System is in online mode\n");
119 else
120   printf ("System is offline (open loop) mode\n");
121
122 /* set up debug level for all the functions               */
123 InitNeuralControl(Debug);
124 InitDbInterface(Debug);
125 InitNeuralCalc(Debug);
126
127
128 /* initialise the database */
```

```
129 ier = InitDatabase (mdb, sdb);
130
131 switch (ier) {
132   case 0: printf ("Successful log on to both databases \n");
133           break;
134   case 1: printf ("Slave DB log on failed, Master Success \n");
135           break;
136   case 2: printf ("Master DB log on failed, Slave Success \n");
137           break;
138   case 3: printf ("Master and Slave log on failed, exiting \n");
139           exit (1);
140   case 4: printf ("Master connected but failed query \n");
141           break;
142   case 5: printf ("Slave Connected but failed query \n");
143           break;
144 }
145
146 SingleWrite ( (rtUInt8 *) &CurStatus,1);
147
148 if (InitTimekeeper (myProcId) > 0) {
149   printf ("Timer system failed to start -- exiting to system\n");
150   exit(1);
151 }
152
153 /*****************************************************************/
154 /* ask functions to perform once off initialisations */
155 /*NeuralCalcInit ();              */
156 /*****************************************************************/
157 NeuralCalcInit_1 ();
158 printf ("Back in the main program after initialization...\n");
159 |
160
```

```
161 nt InitTimekeeper (rtProcessId *myProcId)
162 {
163 rtStartTime          start_1        = {rtWILD,rtWILD,rtWILD,-
164 1,-1,-1,0};
165
166 /* is this where the time interval for the logging of database points
167 happens */
168 rtInterval           interval_1     = {0,0,0,120,0};
169 struct timeval       next;
170 unsigned char        *msg;
171
172 if (rtStartTimer(NULL,myProcId,&start_1,&interval_1,msg,2,
173     (rtInt32) 10,(rtInt32) 0,&next) == (rtTimerId *) NULL) {
174   printf ("timer initialisation failed - cannot continue\n");
175   return (1);
176 } else {
177   if (thisDebug)
178     printf ("Rtap Timer initialised successfully\n");
179 }
180 return (0);
181 }
182
183   /* main processing loop
184         */
185   if (thisDebug) printf ("Start of main processing loop\n");
186
187   ReadProcVarTables ();
188   ReadManVarTables ();
189   ReadConVarTables ();
190
191   SetUpProcVarTables ();
192   SetUpManVarTables ();
```

```
193    SetUpContVarTables ();
194
195    RunNeuralNet1();
196    RunNeuralNet2();
197    RunNeuralNet3();
198
199    UpdateProcVarTables ();
200    UpdateManVarTables ();
201    UpdateConVarTables ();
202
203  } else if (msgHeader->msgType == rtASCII_MESSAGE) {
204
205    if (msgHeader->responseFlag == rtRESPOND) {
206      PleaseRespond    = rtTRUE;
207      RplyMsg.hdr.dest  = msgHeader->source;
208      RplyMsg.hdr.msgId        = msgHeader->msgId;
209    } else {
210      PleaseRespond = rtFALSE;
211    }
212    msgBody = (rtChar *) (msgHeader+1);
213    OperatorAction (msgBody);
214  /* send a response if required */
215    if (PleaseRespond) {
216      if (thisDebug)
217        printf ("Response requested \n");
218      strcpy (RplyMsg.bdy,"success");
219      RplyMsg.hdr.msgBodySize = sizeof(RplyMsg.bdy);
220      RplyMsg.hdr.priority   = 10;
221      RplyMsg.hdr.responseFlag = rtRESPONSE;
222      RplyMsg.hdr.typeHdrSize = 0;
223      RplyMsgSize      = sizeof(RplyMsg);
224        if (rtMsgSend (rtBLOCKING,&RplyMsg,RplyMsgSize) !=
225  rtSUCCESS)
226          printf ("Could not send reply\n");
227      }
228
229  } else {
230      printf ("Unknown Message received \n");
231  }
232  }
```

# APPENDIX E (NeuralControl.h)

```
1  Extracts from NeuralControl.h
2
3  /*********************************************************/
4  /* sources/NeuralControl.h 2000/03/06 N Narotam          */
5  /*  Copyright (c) N Narotam 1999, 2000                   */
6  /*  All Rights Reserved                                  */
7  /* DESCRIPTION                                           */
8  /*   Header file - contains general structure and        */
9  /*   function definitions that are needed by all the     */
10 /*   other modules                                       */
11 /*********************************************************/
12 /* COMPILITATION CONTROL                                 */
13 /* 2000/03/30  N Narotam  Ver 1.01 concieved             */
14 /*                                                       */
15 /*********************************************************/
16 #include <rtap/database.h>
17 #define TMHIST 6
18 #define ALPHA 1
19 #define MAXWEIGHTS1 1000
20 /*****************************************************************/
21 /*       General Control parameters structure definition         */
22 /*****************************************************************/
23 typedef struct {
24   rtInt32      Auto;
25   rtInt32      Ext;
26   rtDouble     setpt;
27   rtDouble     curval;
28   rtDouble     extsp;
29   rtDouble     hiscale;
30   rtDouble     loscale;
31 } SetPoint;
32
33 typedef struct {
34   rtInt32      active;
35   rtFloat      pad;
36   rtDouble     curvalue;
37   rtDouble     predvalue;
38   rtDouble     contvalue;
39   rtDouble     sugvalue;
40   rtDouble     error;
41
42
43   rtDouble     filterror;
44
45
46   rtDouble     maxchup;
47   rtDouble     maxchdn;
48   rtDouble     min;
49   rtDouble     max;
50   rtDouble     hiscale;
51   rtDouble     loscale;
52   rtDouble     filter;
53 } ManVar;
54
55 typedef struct {
56   rtInt32      active;
57   rtFloat      pad;
58   rtDouble     curvalue;
59   rtDouble     hiscale;
60   rtDouble     loscale;
61 } ProcVar;
62
63
64 typedef struct {
```

```
65    rtDouble set1[MAXWEIGHTS1];
66    rtDouble set2[MAXWEIGHTS1];
67    rtDouble set3[MAXWEIGHTS1];
68  } WeightsVector;
69
70  /*****************************************************************/
71  /*      Control Variable Parameters structure
72          */
73  /*****************************************************************/
74
75  /*  application status structure        */
76  typedef struct {
77          rtBytes8 env;
78          rtInt16  Debug;
79          rtInt16  OnLine;
80  } AppStatus;
81  extern int OnLine;
82
```

# APPENDIX F (NeuralCalc.c)

```
1  Extracts from NeuralCalc.c
2
3  /*******************************************************/
4  /* sources/NeuralCalc.c 2000/04/11 N Narotam,R Dunwoodie*/
5  /*                                                     */
6  /*  Copyright (c) N Narotam 1999, 2000                 */
7  /*  All Rights Reserved                                */
8  /*                                                     */
9  /* DESCRIPTION                                         */
10 /*   Source file - ?????????????                       */
11 /*******************************************************/
12 /* COMPILITATION CONTROL                               */
13 /* 2000/04/11  N Narotam Conceived                     */
14 /*                                                     */
15 /*******************************************************/
16 /*******************************************************/
17 /* Combined Controllers 1,2 & 3                        */
18 /* Types 6 , 9  & 10 ANN Models                        */
19 /*                                                     */
20 /*******************************************************/
21
22 #include <cr/crStandards.h>
23 #include <rtap/rtap.h>
24 #include <math.h>
25
26 #include "NeuralControl.h"
27 #include "DbInterface.h"
28 #include "nrutil.h"
29 #include "NeuralCalc.h"
30
31 /* local function prototypes */
32 void PrintSetPointValues (SetPoint *);
33 void CascadeProcVar (ProcVar **pv);
34 void CascadeManVar_1 (ManVar **mv);
35 void CascadeContVar_1 (SetPoint **sp);
36
37 void GetWeights1 (void);
38 void GetWeights2 (void);
39 void GetWeights3 (void);
40
41 double Sigmoid(double);
42
43 /* set points data storage area */
44 static SetPoint SPEth[TMHIST];
45 static SetPoint SPBut[TMHIST];
46 static SetPoint SPHex[TMHIST];
47 static SetPoint SPHyd[TMHIST];
48 static SetPoint SPNit[TMHIST];
49 static SetPoint SPC4C2[TMHIST];
50 static SetPoint SPC6C2[TMHIST];
51
52 /* manipulated variables data storage */
53 static ManVar EthFlo[TMHIST];
54 static ManVar ButFlo[TMHIST];
55 static ManVar HexFlo[TMHIST];
56 static ManVar HydFlo[TMHIST];
57 static ManVar NitFlo[TMHIST];
58 static ManVar TealFlo[TMHIST];
59 static ManVar C4C2Flo[TMHIST];
60 static ManVar C6C2Flo[TMHIST];
61 static ManVar H2C2Flo[TMHIST];
62
63 /* process variables data storage */
64 static ProcVar R1Temp[TMHIST];
```

# APPENDIX F (NeuralCalc.c)

```
65  static ProcVar R1Press[TMHIST];
66  static ProcVar R1Prod[TMHIST];
67  static ProcVar R1BTR[TMHIST];
68  static ProcVar R1VentMR[TMHIST];
69  static ProcVar R1Vent[TMHIST];
70  static ProcVar R1CatFdr[TMHIST];
71
72  static WeightsVector Weights;
73
74  /* local data storage */
75  static int thisDebug;
76
77  /*Variables for Neural Net Calc  */
78  /*for RunNeuralNet 1          */
79  /* NNin  --> no. of input nodes in the ANN
80                  */
81  /* NNhid --> no. of 1st layer hidden nodes in the ANN
82                          */
83  /* NNout --> no. of output nodes in the ANN
84                  */
85
86  static int NNin_1;
87  static int NNhid_1;
88  static int NNout_1;
89
90
91  /*********************************************************/
92  /*      InitNeuralCalc:        Initialisation routine for this   */
93  /*                      function.  Currenly sets up    */
94  /*                      debug status only.             */
95  /*      type:          external                       */
96  /*********************************************************/
97  void InitNeuralCalc (int GlobDebug)
98  {
99  thisDebug = GlobDebug & 0x04;
100 if (thisDebug)
101    printf ("NeuralCalc is in Debug mode\n");
102 }
103
104 void NeuralCalcInit_1 (void)
105 {
106    int i;
107
108    ReadProcVarTables ();
109    ReadManVarTables ();
110    ReadConVarTables ();
111
112
113    /*this is to read in all 50 rows for the new large flow and comp
114 tables */
115    /* plus the normal 6 rows for each of the other variables */
116
117    for (i = 0; i < TMHIST ; i++)
118          SetUpProcVarTables ();
119
120    for (i = 0 ; i < TMHIST_1 ; i++)
121    {
122     SetUpManVarTables ();
123     SetUpContVarTables ();
124    }
125
126    GetWeights1();
127    GetWeights2();
128    GetWeights3();
```

```
129
130   UpdateProcVarTables ();
131   UpdateManVarTables ();
132   UpdateConVarTables ();
133
134   if (thisDebug) printf ("End of initialization step\n");
135 }
136
137 void CascadeContVar (SetPoint **sp)
138 {
139 int i;      /* loop counter */
140
141 for (i = (TMHIST-1);i>0;i--) {
142   (*sp+i)->curval = (*sp+(i-1))->curval;
143   (*sp+i)->setpt = (*sp+(i-1))->setpt;
144   (*sp+i)->extsp = (*sp+(i-1))->extsp;
145         }
146 }
147
148 int SetUpContVarTables (void) {
149
150 int offset;
151 SetPoint *ptr;
152 ApsPV tempSP;
153 UniPV tempUni;
154 ApsBOOL AnaFlag;
155 ApsAnalog tempAn01,tempAn02;
156
157
158 /* Read ratio controller setpoints */
159 SingleRead ((rtUInt8 *) &tempSP,68);
160 SPH2C2[0].extsp = SPXLH2C2[0].extsp = tempSP.value;
161
162 SingleRead ((rtUInt8 *) &tempSP,69);
163 SPC4C2[0].extsp = SPXLC4C2[0].extsp = tempSP.value;
164
165 SingleRead ((rtUInt8 *) &tempSP,70);
166 SPC6C2[0].extsp = SPXLC6C2[0].extsp = tempSP.value;
167
168 /*be sure to know what is happening here .....*/
169 /*the code must know which analyser reading to take */
170
171 SingleRead ((rtUInt8 *) &AnaFlag,13);
172
173 offset = 1 - AnaFlag.value;
174
175 ptr = &SPEth[0];
176 CascadeContVar (&ptr);
177 if (SPEth[0].Auto < 255) {
178   SingleRead ((rtUInt8 *) &tempAn01,14+offset);
179
180   SPEth[0].curval = (ALPHA) * (tempAn01.value) + (1- ALPHA) *
181 (SPEth[1].curval);
182   /*SPEth[0].curval = tempAn01.value;*/
183
184   /* Calculate analyzer sp values using the ratio sp values and
185 the ethylene analyzer value */
186   SPBut[0].extsp = tempAn01.value * SPC4C2[0].extsp;
187   SPHex[0].extsp = tempAn01.value * SPC6C2[0].extsp;
188   SPHyd[0].extsp = tempAn01.value * SPH2C2[0].extsp;
189
190 }
191
192 ptr = &SPBut[0];
```

# APPENDIX F (NeuralCalc.c)

```
193  CascadeContVar (&ptr);
194  if (SPBut[0].Auto < 255) {
195          SingleRead ((rtUInt8 *) &tempAn01,16+offset);
196
197          SPBut[0].curval = (ALPHA) * (tempAn01.value) + (1-
198  ALPHA) * (SPBut[1].curval);
199    /*SPBut[0].curval = tempAn01.value;*/
200  }
201
202  ptr = &SPHex[0];
203  CascadeContVar (&ptr);
204  if (SPHex[0].Auto < 255) {
205          SingleRead ((rtUInt8 *) &tempAn01,18+offset);
206
207          SPHex[0].curval = (ALPHA) * (tempAn01.value) + (1-
208  ALPHA) * (SPHex[1].curval);
209
210          /*SPHex[0].curval = tempAn01.value;*/
211  }
212
213  ptr = &SPHyd[0];
214
215  ptr = &SPC4C2[0];
216  CascadeContVar (&ptr);
217
218  if (SPC4C2[0].Auto < 255) {
219    SingleRead ((rtUInt8 *) &tempAn01,14+offset);
220          SingleRead ((rtUInt8 *) &tempAn02,16+offset);
221
222    SPC4C2[0].curval = (ALPHA) * (tempAn02.value /
223  tempAn01.value) + (1- ALPHA) * (SPC4C2[1].curval);
224
225          /*SPC4C2[0].curval = SPBut[0].curval / SPEth[0].curval ;*/
226  }
227
228  /* Read ratio controller setpoints */
229  SingleRead ((rtUInt8 *) &tempSP,68);
230  SPH2C2[0].extsp = SPXLH2C2[0].extsp = tempSP.value;
231
232  SingleRead ((rtUInt8 *) &tempSP,69);
233  SPC4C2[0].extsp = SPXLC4C2[0].extsp = tempSP.value;
234
235  SingleRead ((rtUInt8 *) &tempSP,70);
236  SPC6C2[0].extsp = SPXLC6C2[0].extsp = tempSP.value;
237
238
239
240
241  return(0);
242
243  }
244
245  int ReadConVarTables (void) {
246  SingleRead ((rtUInt8 *) &SPEth,32);
247  SingleRead ((rtUInt8 *) &SPBut,33);
248  SingleRead ((rtUInt8 *) &SPHex,34);
249  SingleRead ((rtUInt8 *) &SPHyd,35);
250  SingleRead ((rtUInt8 *) &SPNit,36);
251  SingleRead ((rtUInt8 *) &SPC4C2,37);
252  SingleRead ((rtUInt8 *) &SPC6C2,38);
253  SingleRead ((rtUInt8 *) &SPH2C2,39);
254
255  return (0);
256  }
```

```
257
258  int UpdateConVarTables (void) {
259
260  SingleWrite ((rtUInt8 *) &SPEth,32);
261  SingleWrite ((rtUInt8 *) &SPBut,33);
262  SingleWrite ((rtUInt8 *) &SPHex,34);
263  SingleWrite ((rtUInt8 *) &SPHyd,35);
264  SingleWrite ((rtUInt8 *) &SPNit,36);
265  SingleWrite ((rtUInt8 *) &SPC4C2,37);
266  SingleWrite ((rtUInt8 *) &SPC6C2,38);
267  SingleWrite ((rtUInt8 *) &SPH2C2,39);
268  return (0);
269  }
270
271  int ReadVectorTables (void) {
272  SingleRead ((rtUInt8 *) &Weights.set1,65);
273  SingleRead ((rtUInt8 *) &Weights.set2,66);
274  SingleRead ((rtUInt8 *) &Weights.set3,67);
275
276  return (0);
277  }
278
279  void GetWeights1 ()
280  {
281  int i,iweights;                    /*local counter
282  interger              */
283  int Nweights;                      /*local integer for the
284  number of weights     */
285
286  FILE *fp;
287
288  if ( (fp=fopen
289  ("/usr/users/nirmal/projects/UndNeural/sources/weights1.dat","rt")
290  ) == NULL) {
291    printf ("unable to read weights file\n");
292    exit(0);
293    }
294  fscanf(fp , "%d\n%d\n%d\n" , &NNin_1, &NNhid_1, &NNout_1);
295
296  Nweights = (NNin_1+1)*(NNhid_1) + (NNhid_1+1)*(NNout_1);
297
298
299  /* Initialize local weights array to zero */
300  for (i=0;i<MAXWEIGHTS1;i++){
301    Weights.set1[i]=0;
302  }
303
304  for (iweights=0 ; iweights<Nweights ; iweights++)
305          {
306          fscanf(fp , "%lf\n" , &(Weights.set1[iweights]) );
307  /*      printf ("weight%d %lf\n", iweights, Weights.set1[iweights]
308  );          */
309          }
310
311  SingleWrite ((rtUInt8 *) &Weights.set1,65);
312
313  fp=0;
314  }
315
316  int RunNeuralNet2(void)
317  {
318  /* all the basic interger etc declarations
319          */
```

```
int             i,
                ii,
                j,
                jj;
double          v;

rtDouble        *NNinputs,
                *S2,
                *Contr_Outputs,
                *Pred_Outputs,
                *mv_sugg,
                *mv_tm1,
                *mvMin,
                *mvMax,
                *mvMaxChup,
                *mvMaxChdn,
                *mv_er,
                *mv_er_filt,
                *mv_er_filt_tm1,
                *alpha;

/* declaring the size of the vectors */
NNinputs =          dvector(0,NNin_2 -1) ;
S2 =                dvector(0,NNhid_2 -1) ;
Contr_Outputs =     dvector(0,NNout_2 -1) ;
Pred_Outputs =          dvector(0,NNout_2 -1) ;
mv_sugg =           dvector(0,NNout_2 -1) ;
mv_tm1 =            dvector(0,NNout_2 -1) ;
mvMin =             dvector(0,NNout_2 -1) ;
mvMax =             dvector(0,NNout_2 -1) ;
mvMaxChup =             dvector(0,NNout_2 -1) ;
mvMaxChdn =             dvector(0,NNout_2 -1) ;
mv_er =             dvector(0,NNout_2 -1) ;
mv_er_filt =        dvector(0,NNout_2 -1) ;
mv_er_filt_tm1 =    dvector(0,NNout_2 -1);
alpha =             dvector(0,NNout_2 -1) ;


/*-------------------------------------------------------------
        */
/* CONTROLLER NETWORK
                                                              */
/* Type ANN Model 9

                        */
/*-------------------------------------------------------------
        */
/*-------------------------------------------------------------------------
-------------------------------*/
/*set points of levels for next time step             */
/*-------------------------------------------------------------------------
-------------------------------*/
/* We do the normalisation here so that all the inputs to the net
are scaled between 0.1 and 0.9 */
/* using the scaling factors that are stored in the RTAP tables


/*-------------------------------------------------------------------------
-------------------------------*/
/*be careful to get the indices right, as they represent the previous
time values                 */
/*-------------------------------------------------------------------------
-------------------------------*/
```

```
/*although hi and lo -scales are the same for all rows


/*-------------------------------------------------------------------------
----------------------------------*/
/* Desired Hexene/Ethylene composition setpoint (t+1)
        */
NNinputs[0] = (0.9-0.1)*(SPC6C2[0].curval-
SPC6C2[0].loscale)/(SPC6C2[0].hiscale-SPC6C2[0].loscale)
+0.1;

/*NNinputs[0] = (0.9-0.1)*(SPC6C2[0].extsp-
SPC6C2[0].loscale)/(SPC6C2[0].hiscale-SPC6C2[0].loscale)
+0.1;          */

/* Desired Hydrogen/Ethylene composition setpoint (t+1)
        */
NNinputs[1] = (0.9-0.1)*(SPH2C2[0].curval-
SPH2C2[0].loscale)/(SPH2C2[0].hiscale-SPH2C2[0].loscale)
+0.1;

/*NNinputs[1] = (0.9-0.1)*(SPH2C2[0].extsp-
SPH2C2[0].loscale)/(SPH2C2[0].hiscale-SPH2C2[0].loscale)
+0.1;*/

/* These variables are all taken from the first row of the tables ie
[0]                             */
/*since they are the most recent value


/* Ethylene current flow rate
                                */
NNinputs[2] = (0.9-0.1)*(EthFlo[0].curvalue-
EthFlo[0].loscale)/(EthFlo[0].hiscale-EthFlo[0].loscale) +0.1;


/* Reactor Pressure (t)                            */
NNinputs[3] = (0.9-0.1)*(R1Press[0].curvalue-
R1Press[0].loscale)/(R1Press[0].hiscale-R1Press[0].loscale)
+0.1;

 /* Reactor Temp (t)                               */
NNinputs[4] = (0.9-0.1)*(R1Temp[0].curvalue-
R1Temp[0].loscale)/(R1Temp[0].hiscale-R1Temp[0].loscale)
+0.1;

/* Production rate (t)        */
NNinputs[5] = (0.9-0.1)*(R1Prod[0].curvalue-
R1Prod[0].loscale)/(R1Prod[0].hiscale-R1Prod[0].loscale) +0.1;



for (i=0;i<NNin_2;i++){
   if (NNinputs[i] < 0.01) {
      NNinputs[i] = 0.01;
/*      printf ("nninputs_C21 %d %lf\n", i, NNinputs[i] );          */
       }
    if (NNinputs[i] > 0.99) {
      NNinputs[i] = 0.99;
/*      printf ("nninputs_C21 %d %lf\n", i, NNinputs[i] );          */
       }
    }
```

```
446
447 /*-------------------------------------------------------------------
448         */
449 /* this section of code calculates the Neural Net Contr_Outputs
450                 */
451 /* using the weights and max/min that were read in earlier  */
452 /*-------------------------------------------------------------------
453         */
454
455 /* Computation of outputs hidden layer
456                                                 */
457 for (i=0; i< NNhid_2; i++)
458         {
459         double ST = 0.0;
460         int j = (i)*(NNin_2  +1);
461         for (jj = 0; jj < NNin_2 ; jj++)
462         {
463                 ST += NNinputs[jj]*Weights.set2[jj+j];
464         }
465         ST += Weights.set2[j+NNin_2 ] ;
466         /* Sigmoid Function */
467         S2[i]=Sigmoid(ST);
468         }
469 /* Computation of outputs third layer
470                         */
471 for (ii=0; ii<NNout_2; ii++)
472 {
473         double ST = 0.0;
474         j = (ii)*(NNhid_2 + 1)+(NNhid_2)*(NNin_2 +1);
475         for (jj=0; jj<NNhid_2 ; jj++)
476         {
477                 ST += S2[jj]*Weights.set2[jj+j]  ;
478         }
479         ST += Weights.set2[j+NNhid_2] ;
480         /* Sigmoid Function */
481         Contr_Outputs[ii] = Sigmoid(ST);
482 }
483
484
485
486 /* here we need to re-scale the outputs of the neural network */
487 /* and re-assign the actual value to the Contr_Outputs so that it
488 can be used later */
489
490 C6C2Flo[0].contvalue= (Contr_Outputs[0] -
491 0.1)*(C6C2Flo[0].hiscale-C6C2Flo[0].loscale)/(0.9-0.1)  +
492 C6C2Flo[0].loscale;
493 Contr_Outputs[0] = C6C2Flo[0].contvalue ;
494
495 H2C2Flo[0].contvalue= (Contr_Outputs[1] -
496 0.1)*(H2C2Flo[0].hiscale-H2C2Flo[0].loscale)/(0.9-0.1)  +
497 H2C2Flo[0].loscale;
498 Contr_Outputs[1] = H2C2Flo[0].contvalue ;
499
500 /*-------------------------------------------------------------------
501         */
502 /* PREDICTOR NETWORK
503                                                                 */
504 /*-------------------------------------------------------------------
505         */
506
507 /* We do the normalisation here so that all the inputs to the net
508 are scaled between 0.1 and 0.9 */
```

```
/* using the scaling factors that are stored in the RTAP tables

/*-------------------------------------------------------------------------
----------------------------------*/
/*be careful to get the indices right, as they represent the previous
time values                    */
/*-------------------------------------------------------------------------
----------------------------------*/
/*-------------------------------------------------------------------------
----------------------------------*/
/*MUST BE 1 TIME STEP BEHIND THE CONTROLLER MODEL
ABOVE                                    */
/*-------------------------------------------------------------------------
----------------------------------*/
/*the Composition SP's come from actual plant values */

/* Desired Hexene/Ethylene composition setpoint (t)        */
NNinputs[0] = (0.9-0.1)*(SPC6C2[1].curval- SPC6C2[0].loscale)/(
SPC6C2[0].hiscale- SPC6C2[0].loscale) +0.1;


/*NNinputs[0] = (0.9-0.1)*(SPC6C2[0].curval-
SPC6C2[0].loscale)/( SPC6C2[0].hiscale- SPC6C2[0].loscale)
+0.1;   */


/* Desired Hydrogen/Ethylene composition setpoint (t)
        */
NNinputs[1] = (0.9-0.1)*(SPH2C2[1].curval -
SPH2C2[0].loscale)/(SPH2C2[0].hiscale-SPH2C2[0].loscale)
+0.1;

/*NNinputs[1] = (0.9-0.1)*(SPH2C2[0].curval -
SPH2C2[0].loscale)/(SPH2C2[0].hiscale-SPH2C2[0].loscale)
+0.1;*/

/* These variables are all taken from the second row of the tables
ie [1]                          */
/* since they are 1 time step behind the inputs to the controller
network                                  */

/* Ethylene  flow rate  (t-1)
                        */
NNinputs[2] = (0.9-0.1)*(EthFlo[1].curvalue-
EthFlo[1].loscale)/(EthFlo[0].hiscale-EthFlo[0].loscale) +0.1;


/* Reactor Pressure (t-1)                                  */
NNinputs[3] = (0.9-0.1)*(R1Press[1].curvalue-
R1Press[1].loscale)/(R1Press[1].hiscale-R1Press[1].loscale)
+0.1;

 /* Reactor Temp (t-1)                                     */
NNinputs[4] = (0.9-0.1)*(R1Temp[1].curvalue-
R1Temp[1].loscale)/(R1Temp[1].hiscale-R1Temp[1].loscale)
+0.1;

/* Prod rate  (t-1)              */
NNinputs[5] = (0.9-0.1)*(R1Prod[1].curvalue-
R1Prod[1].loscale)/(R1Prod[1].hiscale-R1Prod[1].loscale) +0.1;
```

```
for (i=0;i<NNin_2;i++){
  if (NNinputs[i] < 0.01) {
    NNinputs[i] = 0.01;
/*    printf ("nninputs_C22 %d %lf\n", i, NNinputs[i] );          */
     }
  if (NNinputs[i] > 0.99) {
    NNinputs[i] = 0.99;
/*-------------------------------------------------------------------
     */
/*     this section of code calculates the predictor Neural Net
Pred_Outputs                */
/* using the weights  that were read in earlier
                                 */

/* Computation of outputs hidden layer
                                 */
for(i=0; i< NNhid_2; i++)
     {
     double ST = 0.0;
     int j = (i)*(NNin_2  +1);
     for (jj = 0; jj < NNin_2 ; jj++)
     {
          ST += NNinputs[jj]*Weights.set2[jj+j]  ;
     }
     ST += Weights.set2[j+NNin_2 ]  ;
     /* Sigmoid Function                                  */
          S2[i]=Sigmoid(ST);
}
/* Computation of outputs third layer
     */
for(ii=0; ii<NNout_2; ii++)
{
     double ST = 0.0;
     j = (ii)*(NNhid_2+1)+(NNhid_2)*(NNin_2 +1);
     for (jj=0; jj<NNhid_2; jj++)
     {
          ST += S2[jj]*Weights.set2[jj+j] ;
     }
     ST += Weights.set2[j+NNhid_2] ;
     /* Sigmoid Function                                  */
     Pred_Outputs[ii] = Sigmoid(ST);
}

/* here we need to re-scale the outputs of the neural network */
/* and re-assign the actual value to the Pred_Outputs so that it
can be used later */

C6C2Flo[0].predvalue= (Pred_Outputs[0] -
0.1)*(C6C2Flo[0].hiscale-C6C2Flo[0].loscale)/(0.9-0.1)  +
C6C2Flo[0].loscale;
Pred_Outputs[0] = C6C2Flo[0].predvalue ;

H2C2Flo[0].predvalue= (Pred_Outputs[1] -
0.1)*(H2C2Flo[0].hiscale-H2C2Flo[0].loscale)/(0.9-0.1)  +
H2C2Flo[0].loscale;
Pred_Outputs[1] = H2C2Flo[0].predvalue ;


 for (i=0;i<5;i++){
/*     if( Pred_Outputs[i] < 0) Pred_Outputs[i]=0;*/
     }
/*-------------------------------------------------------------------
     */
```

```
636 /* the variables that are loaded below here are temporary vectors
637 */
638 /* that make the calculations simpler
639                                                              */
640 /*-----------------------------------------------------------
641         */
642
643 /*-----------------------------------------------------------
644         */
645 /* this is the previous control action vector , ie time minus 1
646                 */
647 /*      and is 1 TIME STEP BEHIND THE CURRENT ie row [1]
648                                         */
649 /*-----------------------------------------------------------
650         */
651 mv_tm1[0] = C6C2Flo[1].curvalue ;
652
653 mv_tm1[1] = H2C2Flo[1].curvalue ;
654 /*-----------------------------------------------------------
655         */
656
657 /*  need to load the mv_er_filt array
658                                                          */
659 /*      and 1 TIME STEP BEHIND THE CURRENT ie row [1]
660                                         */
661 /*-----------------------------------------------------------
662         */
663 mv_er_filt_tm1[0] = C6C2Flo[1].filterror ;
664
665 mv_er_filt_tm1[1] = H2C2Flo[1].filterror ;
666 /*-----------------------------------------------------------
667         */
668 /*-----------------------------------------------------------
669         */
670 /*  need to load the mvMin     array
671 /*-----------------------------------------------------------
672         */
673 mvMin[0] = C6C2Flo[0].min ;
674
675 mvMin[1] = H2C2Flo[0].min ;
676 /*-----------------------------------------------------------
677         */
678
679 /*-----------------------------------------------------------
680         */
681 /*  need to load the mvMax     array
682                                                          */
683 /*-----------------------------------------------------------
684         */
685 mvMax[0] = C6C2Flo[0].max ;
686
687 mvMax[1] = H2C2Flo[0].max ;
688 /*-----------------------------------------------------------
689         */
690
691 /*-----------------------------------------------------------
692         */
693 /*  need to load the mvMaxChup array
694                                                              */
695 /*-----------------------------------------------------------
696         */
697 mvMaxChup[0] = C6C2Flo[0].maxchup ;
698
699 mvMaxChup[1] = H2C2Flo[0].maxchup ;
```

```
/*-------------------------------------------------------------------
        */
/*-------------------------------------------------------------------
        */
/*  need to load the mvMinChdn array
                                                                */
/*-------------------------------------------------------------------
        */
mvMaxChdn[0] = C6C2Flo[0].maxchdn ;

mvMaxChdn[1] = H2C2Flo[0].maxchdn ;
/*-------------------------------------------------------------------
        */

/*-------------------------------------------------------------------
        */
/*  need to load the alpha - filter constant array

         */
/*-------------------------------------------------------------------
        */
alpha[0] = C6C2Flo[0].filter ;

alpha[1] = H2C2Flo[0].filter ;
/*-------------------------------------------------------------------
        */
/*-------------------------------------------------------------------
        */
/*       IMC PREDICTION ERROR FILTERING, CORRECTION
& CONSTRAINTS            */
/* these calculations have been left like this for simplicity     */
/*-------------------------------------------------------------------
        */
/*-------------------------------------------------------------------
        */

for (j=0; j<NNout_2; j++)
        {

        /* error fom 1 time step ago */
        mv_er[j]=mv_tm1[j] - Pred_Outputs[j];

        /* filtered error from the last cycle                      */
        mv_er_filt[j]=alpha[j]*mv_er[j] + (1 -
alpha[j])*mv_er_filt_tm1[j] ;

        /* Correct present output using (filtered) past observed
error                   */
        v  = Contr_Outputs [j]  +  mv_er_filt[j];

        /* The outputs are clipped here to their respective min and
max's          */
/*      if (v < mvMin[j]) v = mvMin[j];
        if (v > mvMax[j]) v = mvMax[j];


        /* The outputs here are ramp limited
                        */

/*      if ( (v - mv_tm1[j]) > mvMaxChup[j]) v = mv_tm1[j] +
mvMaxChup[j];

        if ( (v - mv_tm1[j]) < mvMaxChdn[j]) v = mv_tm1[j] +
mvMaxChdn[j];
*/
```

```
763         mv_sugg[j] = v;   /* final suggested control  value
764                                 */
765         }
766         }
767 */
768
769 /*THESE ARE VARIABLES THAT WILL BE WRITTEN BACK TO
770 THE */
771 /*TABLES SO THAT WE CAN MONITOR THEM
772                                                     */
773
774 /*-----------------------------------------------------------------
775         */
776 /*  the suggested control actions are written back to the table from
777 the local variable                */
778 /*-----------------------------------------------------------------
779         */
780 C6C2Flo[0].sugvalue  = mv_sugg[0] ;
781
782 H2C2Flo[0].sugvalue = mv_sugg[1] ;
783
784 /*-----------------------------------------------------------------
785         */
786 /*-----------------------------------------------------------------
787         */
788 /*  the control action errors are written back to the table from the
789 local variable            */
790 /*-----------------------------------------------------------------
791         */
792 C6C2Flo[0].error  = mv_er[0] ;
793
794 H2C2Flo[0].error = mv_er[1] ;
795
796 /*-----------------------------------------------------------------
797         */
798
799 /*-----------------------------------------------------------------
800         */
801 /*  the filtered control action errors are written back to the table
802 from the local variable           */
803 /*-----------------------------------------------------------------
804         */
805 C6C2Flo[0].filterror  = mv_er_filt[0] ;
806
807 H2C2Flo[0].filterror = mv_er_filt[1] ;
808 /*-----------------------------------------------------------------
809         */
810
811 return(0);
812
813 }
814 double Sigmoid(double st)
815 {
816 if (st >= 15.935773)
817   st = 0.99999988;
818 else
819   if (st < -15.935773)
820     st = 0.00000012;
821   else
822     st = 1.0/(1.0+exp(-1.0*st));
823 return (st);
824 }
```

# APPENDIX G (NeuralCalc.h)

```
1  Extracts from NeuralCalc.h
2
3  /*********************************************************/
4  /* sources/NeuralCalc.h 2000/04/11 N Narotam             */
5  /*                                                       */
6  /*  Copyright (c) N Narotam 1999, 2000                   */
7  /*  All Rights Reserved                                  */
8  /*                                                       */
9  /* DESCRIPTION                                           */
10 /*   Header file for controller program                  */
11 /*********************************************************/
12 /* COMPILITATION CONTROL                                 */
13 /* 2000/04/11  N Narotam  conceived                      */
14 /*                                                       */
15 /*********************************************************/
16 /* global  function prototypes */
17 extern void NeuralCalcInit_1 (void);
18
19 extern int SetUpProcVarTables (void);
20 extern int UpdateProcVarTables (void);
21 extern int ReadProcVarTables (void);
22 extern int RunNeuralNet1(void);
23 extern int ReadVectorTables (void) ;
24
25
```

# APPENDIX H (PointsDef.c)

```
1  Extracts from PointsDef.c
2
3  /*********************************************************/
4  /* sources/PointsDef.c 2000/03/06 N Narotam              */
5  /*  Copyright (c) N Narotam 1999, 2000                   */
6  /*  All Rights Reserved                                  */
7  /* DESCRIPTION                                           */
8  /*   Source file where RTAP database points are defined*/
9  /*********************************************************/
10 /* COMPILITATION CONTROL                                 */
11 /* 2000/03/30  N Narotam  Ver 1.01 concieved             */
12 /*********************************************************/
13
14 #define NUMPOINTS 71
15 static DbInfo          PtInfo[NUMPOINTS];
16
17 /*****************************************************************/
18 /*      InitPoints:     The user sets up this function with the */
19 /*                      details of the all the Database points  */
20 /*                      that may be needed to be read
21        */
22 /*                      called on startup initialisation ONLY   */
23 /*      Type:           local                                   */
24 /*      return codes:   NONE                                    */
25 /*****************************************************************/
26 void InitPoints (void) {
27
28 /*****************************************************************/
29 /* local declarations to cope with odd DB reads
30        */
31 /* these objects carry no data - they are just used to establish*/
32 /* the size during reads and writes.  This means that once off
33        */
34 /* variables do not need to be globalised                        */
35 /*****************************************************************/
36 SetPoint t_setpoint[TMHIST];
37 ManVar t_manvar[TMHIST];
38 ProcVar t_procvar[TMHIST];
39 rtDouble t_vector[MAXWEIGHTS1];
40
41 /* point 0 setup information   */
42 PtInfo[0].alias
43        ="NeuAppInfo.DBStatus(0,0:5)";
44 PtInfo[0].PtType                       = "UNIPOL";
45 PtInfo[0].DbInfo .addr.type            = rtDB_DIRECT;
46 PtInfo[0].DbInfo .addr.ds.direct.type  = rtDIR_FIELD;
47 PtInfo[0].DbInfo .addr.ds.direct.pointId =
48 malloc(sizeof(rtPointId) );
49 PtInfo[0].DbInfo .attrType             = rtTABLE;
50 PtInfo[0].DbInfo .recordCnt            = 1;
51 PtInfo[0].DbInfo .size                 = sizeof (DBStatus);
52 PtInfo[0].DbInfo .deTypes              =
53 calloc(7,sizeof(rtUInt8) );
54 PtInfo[0].DbInfo .deTypes[0]           = rtBYTES8;
55 PtInfo[0].DbInfo .deTypes[1]           = rtBYTES8;
56 PtInfo[0].DbInfo .deTypes[2]           = rtLOGICAL;
57 PtInfo[0].DbInfo .deTypes[3]           = rtLOGICAL;
58 PtInfo[0].DbInfo .deTypes[4]           = rtLOGICAL;
59 PtInfo[0].DbInfo .deTypes[5]           = rtLOGICAL;
60 PtInfo[0].DbInfo .deTypes[6]           = rtUNDEFINED;
61
62 /* point 11 setup information  */
```

```
63 PtInfo[11].alias                          =
64 "TC_4019.PV(0,0:7)";
65 PtInfo[11].PtType                         = "MOORE";
66 PtInfo[11].DbInfo.addr.type               = rtDB_DIRECT;
67 PtInfo[11].DbInfo.addr.ds.direct.type     = rtDIR_FIELD;
68 PtInfo[11].DbInfo.addr.ds.direct.pointId  =
69 malloc(sizeof(rtPointId) );
70 PtInfo[11].DbInfo.attrType                = rtTABLE;
71 PtInfo[11].DbInfo.recordCnt               = 1;
72 PtInfo[11].DbInfo.size                    = sizeof (ApsPV);
73 PtInfo[11].DbInfo.deTypes                 =
74 calloc(9,sizeof(rtUInt8) );
75
76 /* point 55 setup information  */
77 PtInfo[55].alias                          =
78 "NeuPVVars.PVR1Temp(0:5,0:4)";
79 PtInfo[55].PtType                         = "CUSTOM";
80 PtInfo[55].DbInfo .addr.type              = rtDB_DIRECT;
81 PtInfo[55].DbInfo .addr.ds.direct.type    = rtDIR_FIELD;
82 PtInfo[55].DbInfo .addr.ds.direct.pointId =
83 malloc(sizeof(rtPointId) );
84 PtInfo[55].DbInfo .attrType               = rtTABLE;
85 PtInfo[55].DbInfo .recordCnt              = TMHIST;
86 PtInfo[55].DbInfo .size                          = sizeof
87 (t_procvar);
88 PtInfo[55].DbInfo .deTypes                =
89 calloc(6,sizeof(rtUInt8) );
90
91
```

# APPENDIX I (weigths.txt)

```
1  Extract from a weight file
2
3
4  6       10      2
5  6.998
6  -7.662
7  -36.341
8  15.729
9  0.134
10 0.211
11 23.036
12 18.79
13 -0.001
14 0.166
15 3.974
16 -0.056
17 6.32
18 -11.84
19 21.506
20 -14.555
21 -18.838
22 12.274
23 -0.047
24 1.895
25
26 etc
27 etc
```

# APPENDIX J (sensitivity.m)

```
Extract from sensitivity.m

This is an example of a file that produces the sensitivity curves
that are used in the dissertation.

%Test the sensitivity of the models
%-----------------------------------------
-
%clear all the previous values
clc;
clear all;
%-----------------------------------------
-
%global functions
global No_inputs;
global inputs;
global outputs;
global act_outs;      % the actual outputs
global No_outputs;
global Nhl;             %no of hidden layers
global Nhu1;          %no of hidden units in
layer 1
global wih;             %weights from input to
hidden layer
global who;             %weights from hidden
layer to output layer
global upper_lim  low_lim;
global max_inputs min_inputs;
global max_outputs min_outputs;
%-----------------------------------------
-
%number of hidden layers
Nhl=1;
%-----------------------------------------
-
%reads the input data from a text file
open_inputs111a
%take the mean of the inputs
loop_inputs_abs = mean(inputs);
inputs=0;
act_outs=0;
%-----------------------------------------
-
%calls a file to open and format the 'weights'
file
%fine RD991020
column_weights_open111a
%this finds the absolute values of the outputs
so that the % change from this
%value can be calculated
[outputs_abs]=compute(loop_inputs_abs);
% this is the inputs we want to perturb,
starting at the 1st input and to the number
check = [1; 5;];
for i = 1:size(check,1),
   kk=1;
   for k = -10:2:10,

      loop_inputs = loop_inputs_abs;


loop_inputs(check(i,:))=loop_inputs(check(i,:))+
loop_inputs(check(i,:))*(k/100);
      %---------------------------------------
-------
```

# APPENDIX J (sensitivity.m)

```
      %checks the min's and max's of the data
set and then scales the inputs
      %between an upper limit and lower limit
      %maxima12a;          %checks out fine with
ModelGen
      %---------------------------------------
------

      [outputs]=compute(loop_inputs);

      %---------------------------------------
-------
      %this rescales the outputs,
      %re_scale12a;
      %---------------------------------------
-------------------
      %this is a storage vector for later
plotting
      %  the outputs
      %      outputs1(:,kk,i) = 100*(outputs_abs
- outputs)./outputs_abs;
      outputs1(:,kk,i) = 100*(outputs -
outputs_abs)./outputs_abs;
      %---------------------------------------
-------------------
      %just a variable to plot on the x-axis
      tp(kk,:)=k;

      kk=kk+1;
   end
end
%-----------------------------------------
-
% plot some graphs to see the actual outputs
versus the predicted ones
a = ['ko-';'kx-';'k+-';'k*-';'gs-';'bd-';'wv-
';'k<-';'y>-';'m^-';'cp-';'rh-';'go-';'bx-';'w+-
';'k*-'];
b=['Hex%/Eth% SP     ';
   'Hyd%/Eth% SP     ';
   'Hex%/Eth%  t=50  ';
   'Hyd%/Eth%  t=50  '];
for i=1:size(check,1),
   figure(i)
   clf;
   for j=1:No_outputs,
      plot(  tp, (outputs1(j,:,i) ),
num2str(a(j,:))  )
      hold on
   end
   grid on
   %   title('Inverse Sensitivity')
   legend('C6/C2 flow','H2/C2 flow',4);
   xlabel(['%change of Input variable:
',num2str(b(i,:))]);
   ylabel('%change in flow ratios');
   hold on

end
```

# APPENDIX K (read_weights.m)

```
Extract from read_weights.m

%this file reads in the value of the weights
from the modelgen text file
%-----------------------------------------
-
fid = fopen('C:\dunwoodie\stuff from
Pc\data\Hfm2010\Model11\assia\resultFX_1.txt');
%-----------------------------------------
-
%read in weights from the two columns from
assia's file
weights=zeros(Nhu1   * (  No_inputs+ 1 )  +
No_outputs *(Nhu1+1 ) ,  2);
weights = fscanf(fid,'%g %g',[2 (Nhu1   * (
No_inputs+ 1 )  +  No_outputs *(Nhu1+1 ) )  ]);
weights=weights(2,:);
column=1;                  % the column number
in the weights array
row =1;                    % the row number in
the weights array

for i=1:(Nhu1 * (No_inputs + 1)),  % the no. of
weights for the inputs to hidden layer
   wih(row,column) = weights(i);

   if column == (No_inputs+1)
      column=0;
      row=row+1;
   end

   i=i+1;
   column=column+1;
end

column=1;                  % the column number
in the weights array
row =1;                    % the row number in
the weights array

for i=(Nhu1 * (No_inputs + 1) + 1):(Nhu1 *
(No_inputs + 1) +  No_outputs *(Nhu1+1 ) ) ,  %
the no. of weights for the hidden layer to the
output
   who(row,column) = weights(i);

   if column == (Nhu1+1)
      column=0;
      row=row+1;
   end

   i=i+1;
   column=column+1;
end
fclose(fid);
```

# APPENDIX L (compute.m)

```
Extract from compute.m
% the computation of the outputs froma neural
net using the weights
function[outputs_local] =
compute(inputs_local);
inputs_local;
   %---------------------------------------
   %this initialises the value of the nodes
'node_hidden' in the hidden layer to zero
   node_hidden = zeros(1,Nhu1+1);
   %---------------------------------------
   %'for' loops to calculate values of the
hidden layer nodes
   for i=1:(Nhu1),  %start of for loop 1

      for j=1:(No_inputs+1),     %start of for
loop 2
         node_hidden(i)=node_hidden(i) +
inputs_local(j)*wih(i,j);  %8/9RD need to step
through inputs
      end     % end of for loop 2
      %value after it has been passed through
the output function
      if node_hidden(i)> 15.935773
         node_hid_func(i)=0.99999988;
      else
         if node_hidden(i)<-15.935773
            node_hid_func(i) = 0.00000012;
         else
            node_hid_func(i)= 1.0 /(1.0+exp(-
1.0*node_hidden(i)) );
         end
      end
   end     %end of for loop 1
   %---------------------------------------
   %Bias value for the hidden row
   node_hid_func(Nhu1+1)=1;
   %---------------------------------------
   %initialise the values of the output node/s
to zero
   node_out = zeros(No_outputs);
   %---------------------------------------
   %'for' loops to calculate values of the
output layer nodes
   for i=1:(No_outputs),   %start of for loop 3

      for j=1:(Nhu1+1),       %start of for
loop4
         node_out(i)=node_out(i) +
node_hid_func(j)*who(i,j);
      end     %end of for loop 4
      if node_out(i)> 15.935773
         outputs_local(i)=0.99999988;
      else
         if node_out<-15.935773
            outputs_local(i) = 0.00000012;
         else
            outputs_local(i)= 1.0 /(1.0+exp(-
1.0*node_out(i)) );
         end
      end
   end  %end of for loop 3

```

# APPENDIX M (Chaining.m)

```
Extract from chaining.m

%Test to see whether 'chaining algorithm ' works
%-----------------------------------------
%clear all the previous values
clc;
clear all;
%-----------------------------------------
-
%global functions
global No_inputs;
global inputs;
global outputs;
global act_outs;      % the actual outputs
global No_outputs;
global Nhl;             %no of hidden layers
global Nhu1;          %no of hidden units in
layer 1
global wih;             %weights from input to
hidden layer
global who;             %weights from hidden
layer to output layer
global upper_lim  low_lim;
global max_inputs min_inputs;
global max_outputs min_outputs;
%initialization
sum = zeros(1,2);
diff=zeros(1,2);
%number of times the net will be chained
No_chains=197;
NOW=197;
chain_break=NOW;
%-----------------------------------------
-
%number of hidden layers
Nhl=1;
%-----------------------------------------
-
%reads the input data from a text file
open_inputs2ch;
%scale the actual outputs so that it will be the
same size as the nn outs
act_outs = act_outs(1:No_chains,:);
act_inputs=inputs(:,7:12);

%only consider the first line of the inputs file
inputs=inputs(1,:);
%-----------------------------------------
-
%calls a file to open and format the 'weights'
file
%fine RD991020
open_weights2ch;

% tihs is the number of times that the chaining
will be repeated
for chain=1:No_chains,

   inputs(:,7:12) =  act_inputs(chain,:);

%-----------------------------------------
-
%checks the min's and max's of the data set and
then scales the inputs
%between an upper limit and lower limit
```

# APPENDIX M (Chaining.m)

```
66
67 maxima2ch;          %checks out fine with
68 ModelGen
69 %-----------------------------------------
70 -
71 [outputs]=compute(inputs);
72 %-----------------------------------------
73 -
74 %converts the outputs back to the original scale
75 of the data
76 %this rescales the outputs,
77 re_scale2ch;
78
79 %try to correct for offset, only do this on the
80 first step,
81 if chain_break ==NOW,
82    diff=act_outs(chain,:) - outputs;
83    chain_break =0;
84 end
85
86 % correct all the future predictions by this
87 contstant offfset
88 outputs=outputs+diff;
89 %Standard deviation calcs
90 sum = sum + (outputs - act_outs(chain,:)).^2;
91
92 outputs1(chain,:)=outputs;
93 inputs1(chain,:) = inputs(1,11:12);
94 tp(chain,:)=chain;
95
96 %shift the inputs file time wise back one step
97 inputs(1,1) = inputs(1,3);
98 inputs(1,2) = inputs(1,4);
99 inputs(1,3) = inputs(1,5);
100 inputs(1,4) = inputs(1,6);
101 inputs(1,5) = outputs(1,1);
102 inputs(1,6) = outputs(1,2);
103
104 chain_break=chain_break+1;
105
106 end
107
108 %----------------------------------------
109 sum = (sum./ (chain - 1)).^(1/2)
110
111 error=act_outs-outputs1;
112 %----------------------------------------
113 -
114 % plot some graphs to see the actual outputs
115 versus the predicted ones
116 figure(1)
117 clf;
118 plot(tp,outputs1(:,1),'k.-')
119 hold on
120 plot(tp,outputs1(:,2),'kx-')
121 hold on
122 plot(tp,act_outs(:,1),'k:');
123 hold on
124 plot(tp,act_outs(:,2),'k-');
125 hold on
126 legend('NN level1','NN level2','real
127 level1','real level2',1);
128 title('Good ''chained'' neural net predictions
129 of tank levels (pred. horizon - 10)');
130 xlabel('Time');
131 ylabel('%Levels')
```

```
132
133 figure(2)
134 clf;
135 subplot(2,1,1)
136 plot(tp,inputs1(:,1),'k.-')
137 hold on
138 plot(tp,inputs1(:,2),'kx-')
139 hold on
140 grid on
141 legend('Valve1','Valve2',4)
142 ylabel('%Valves')
143
144 title('Valves and errors from good ''chained''
145 neural net predictions');
146
147 subplot(2,1,2)
148 plot(tp,error(:,1),'k.-');
149 hold on
150 plot(tp,error(:,2),'kx-');
151 hold on
152 grid on
153 legend('Error1','Error2',4)
154 ylabel('%Errors')
155 xlabel('Time');
```