USE OF RAMP TESTS TO OBTAIN INVERSE NEURAL NETWORK PROCESS MODELS

By

PAUL A. BELCHER

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1996

USE OF RAMP TESTS TO OBTAIN INVERSE

NEURAL NETWORK PROCESS MODELS

Thesis Approved:

James R Whiteley

Thesis Adviser

Arland H. Johannes

Karen A Hugh

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to thank my thesis adviser Dr. Rob Whiteley for his guidance, suggestions, patience, and encouragement throughout my research experience. I want to give special thanks to my wife Megan. She is understanding and always takes the time to listen and understand. She has been an instrumental part of my success and I greatly appreciate it. I am also very thankful for the support of my entire family, especially my parents, throughout college. I would also like to thank Dr. Karen High, Dr. Alan Tree, and Dr. Rob Whiteley for providing me with enjoyable experiences interacting with students and the opportunity to encourage them. In addition, I want to thank Dr. Alan Tree for his time and effort which made my transition to chemical engineering virtually seamless. I want to thank Dr. Karen High and Dr. A.H. Johannes for being on my committee and for their suggestions. I want to thank the entire School of Chemical Engineering faculty for providing me with an enriching and rewarding graduate school experience. The friendships I have made with some faculty members are invaluable. Finally, I want to thank the School of Chemical Engineering for financially supporting me during graduate school.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

NOMENCLATURE

A	Amplitude
C	Actual process output
$\tilde{C}$	Predicted process output
CHSF	Control Horizon Scaling Factor
CMAC	Cerebellar Model Arithmetic Computer
CSTR	Continuous Stirred-Tank Reactor
DMC	Dynamic Matrix Control
E	Error
EIR	Equivalent Inverse Ramp
G	Actual process
$\tilde{G}$	Process model
Gc	PID Controller
Gc*	IMC Controller
IMC	Internal Model Control
IR	Inverse Ramp
IS	Inverse Step
K_C	Controller gain
L	Process disturbance
MAC	Model Algorithmic Control

MIMO	Multiple Input, Multiple Output
MPC	Multi-step Predictive Control
MPHC	Model Predictive Heuristic Control
n	A harmonic frequency of a function f(t)
N	Number of sampling periods
NIMC	Nonlinear Internal Model Control
P	Input to process
$\overline{P}$	Bias value
PID	Proportional, Integral, Derivative
PRBS	Pseudo Random Binary Signal
R	Setpoint
$\widetilde{R}$	Adjusted setpoint
SISO	Single Input, Single Output
t	Time
τ_I	Reset time
τ_D	Derivative time
ω	Frequency

CHAPTER I. INTRODUCTION

Motivation

Problem

In recent years there has been an increasing interest in neural networks and their possible applications in process control. Process control systems have become increasingly more complex in order to meet the need for increased manufacturing productivity. While traditional proportional, integral, derivative (PID) control methods are adequate most of the time, there are a number of control problems in which the PID strategy fails, especially when the process is very nonlinear and has large delays. In most cases, classical control methods will work with these types of systems, but they usually yield poor results. In others, a robust control system with capabilities beyond those of traditional control techniques is necessary to maintain optimum performance. Achieving this optimum performance may require the precision control of a dynamic system for which there may be either no attainable model or any model that is obtained will be highly uncertain (Sheppard *et al.*, 1992). Thus, a more advanced control strategy is necessary.

One powerful and practical control technique that has been developed is internal model control (IMC). IMC is a generic term for a widely used class of process control algorithms. The general idea is simple. A process model is used on-line to adjust the available manipulated variables in response to disturbances, changing goals, *etc.* While it has been demonstrated that IMC can often satisfy these demanding requirements, there is one weak link in the strategy - the accuracy of the process model.

Solution

Neural networks are attracting interest as process models for IMC as well as other advanced control schemes, such as multi-step predictive control (MPC), dynamic matrix control (DMC), and adaptive control (Su *et al.*, 1992a). Selection of a reasonable model structure based on first principles usually requires many months of effort on the part of a modeling specialist (Ricker, 1991). Even then, there is no guarantee that the resulting model will describe a complex process with sufficient accuracy for use in IMC. One alternative to this classical approach is the formulation of what are essentially empirical nonlinear models through the use of neural networks. While neural networks have already been successfully implemented in many applications, their potential for use in IMC has not been fully exploited.

It is difficult to obtain an inverse process model capable of providing good overall performance using traditional step tests. As a result, there has been minimal research devoted to evaluating inverse neural network models for use in IMC. This study evaluates the use of ramp tests to obtain inverse neural network models. This represents an alternative approach, which has not been previously considered in the literature. The approach is appealing theoretically and practically. Results show the approach to be very promising.

CHAPTER II. BACKGROUND

Model-Based Control

Internal model control (IMC) is derived from multi-step predictive control (MPC). IMC is the base case of MPC, since IMC only makes one prediction of the controlled variable(s) to determine the best input for the manipulated variable(s), while MPC makes several future predictions to determine an optimal set of control moves for each manipulated variable that will minimize the error between the predicted output and a specified trajectory for each controlled variable. A comparison of the classical feedback control structure to the one used in IMC will help one better understand the IMC strategy.

Classical Feedback Control

Figure 1 is a block diagram of the classical feedback control strategy. In the figure, R is the setpoint, C is the output of the system, L represents a disturbance, G is the process, P is the input to the process, and E is the error (R-C) that is input to the controller Gc. Those familiar with PID controllers know that the error E is directly used to obtain a new input to the process P through the use of proportional, integral, and derivative contributions using the formula

$$P(t) = \overline{P} + K_C\left[E(t) + \frac{1}{\tau_I}\int_0^t E\left(t^*\right)dt^* + \tau_D\frac{dE}{dt}\right] \qquad (1)$$

where $\overline{P}$ is the bias value, K_C is the controller gain, τ_I is the reset time, and τ_D is the derivative time. This method does not consider future behavior of the process and, as a result, is shortsighted in its selection of control moves. This can often result in excessive overshoot, long settling times, and overaggressive control actions since one must make a

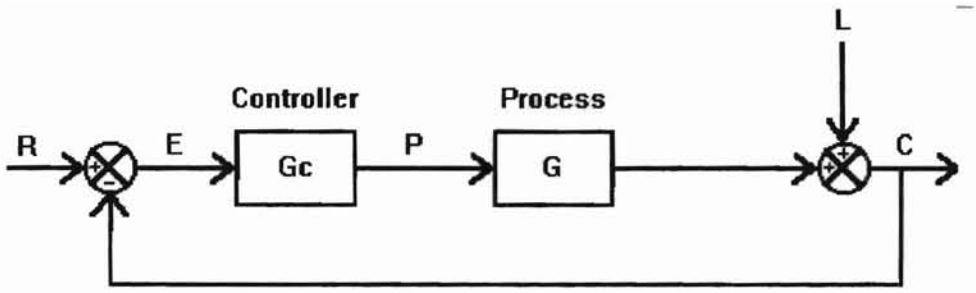

Figure 1. Classical Feedback Control Structure

tradeoff between disturbance rejection and setpoint change capabilities when tuning a PID controller.

Internal Model Control

The IMC design method is based on an assumed model of the plant that relates process inputs to process outputs. To better understand the fundamental differences between the way a classical PID controller and an IMC controller function, consider Figure 2. In Figure 2, R is the setpoint, C is the actual output of the system, $\tilde{C}$ is the predicted output from the model $\tilde{G}$, L represents a disturbance, G is the process, P is the input to the process and the model, the controller is Gc*, $C-\tilde{C}$ is the model mismatch, and $\tilde{R}$ is the adjusted setpoint $(R-(C-\tilde{C}))$. It should be noted that the model mismatch $C-\tilde{C}$ can occur as a result of modeling error, process changes, disturbances, or any combination of the three. The IMC control strategy has many advantages over the classic PID control strategy. Some advantages include inherent anti-windup, which can decrease overshoot and settling time, and the ability to compensate for process dead time by looking past it to determine a control action, which prevents the controller from being overaggressive. Seborg *et al.* (1989) state that there are two important advantages to using the IMC approach: it has the ability to explicitly account for any model uncertainty and it allows the designer to balance control system performance with control system robustness to process changes and modeling errors.

Development of Model-Based Control

The idea of model predictive heuristic control (MPHC), a type of MPC strategy that uses impulse responses to represent the system was introduced by Richalet *et al.* (1978). They successfully applied this strategy to three different industrial processes and

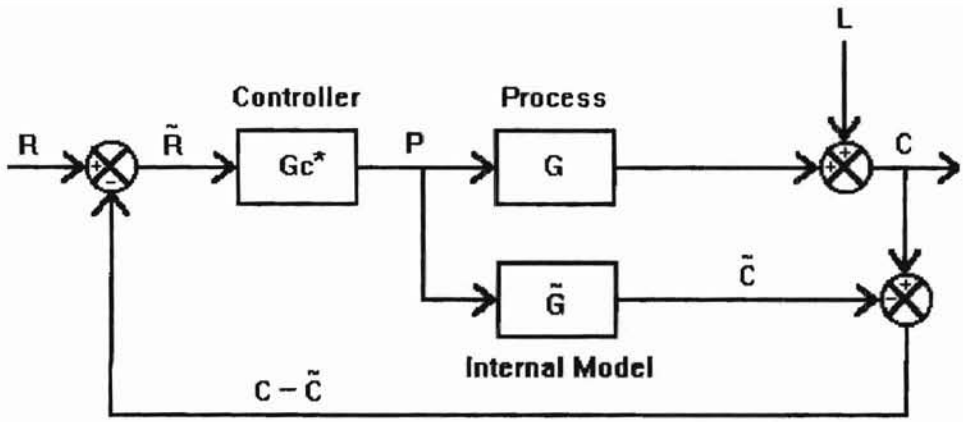

Figure 2. Internal Model Control Structure

were able to achieve significant cost savings for each process. Their success quickly established the idea of optimal and robust control as a very attractive control alternative. The initial success of MPC led to research on the stability and robustness of model algorithmic control (MAC) strategies by Rouhani and Mehra (1982).

An excellent paper that summarizes the various control strategies based on the MPC concept and how they compare and relate to each other, as well as to traditional methods, was presented by Garcia *et al.* (1989). The paper helped establish the idea that, for nonlinear multivariable process control problems, the MPC approach is very advantageous.

A good text on IMC has been provided by Morari and Zafiriou (1989). It covers single input, single output (SISO) and multiple input, multiple output (MIMO) IMC design in detail and includes a chapter on robust stability and performance.

Neural Networks in Control

Neural networks can be very powerful tools when used properly. However, there are a few rules that should be followed. Any good textbook on neural networks, such as Zurada (1992), should provide an adequate starting place for someone who wants to learn about neural networks. There are a number of articles in the literature that do a good job of summarizing the more important aspects of neural networks and can help determine whether or not using a neural network should be considered. An excellent paper summarizing various types of neural networks and how they function in general was presented by Hammerstrom (1993). Background discussion on neural network mechanics

is located in Appendix A. The following represents an overview of how neural network models have been employed in the process industries.

Neural Networks as Process Models

Neural networks have been tested in empirical data modeling applications and compared to conventional statistical techniques by Cheung *et al.* (1992). One advantage of having a good model of process data is that there are many process variables that cannot be measured frequently enough, if at all, while the system is on-line. Furthermore, if the modeling predictions are accurate enough, they can be used to control or monitor a process. The researchers carried out studies comparing the performance of neural networks and linear regression methods in estimating product stream properties on two different fractionators. They had hoped that the neural networks would extract the nonlinearities of the process data, but they discovered that one of the limitations of neural networks is that excessive noise will mask the nonlinearities in the data.

Karim and Rivera (1992) used neural networks to estimate bioprocess variables. They compared the performance of feed forward and recurrent neural networks in learning, recalling, and generalizing the nonlinear behavior of a fermentation process. While they found that both types of networks performed adequately as unmeasureable state estimators and had good recall abilities, the recurrent network did a much better job of generalizing.

The applications of radial basis function networks in process modeling and control have been investigated by Hofland *et al.* (1992). They compared the performance of radial basis function networks to sigmoidal activation function feed forward neural

networks using data from an industrial penicillin fermentation process. Their study showed that radial basis function networks are capable of representing data more accurately than feed forward networks when applied to biomass estimation.

Neural Networks for Fault Detection and Diagnosis

Use of neural networks for fault detection in heat exchangers was investigated by Himmelblau (1992). Deviations from normal states of measurement and internal faults were detected and diagnosed using neural networks, linear discrimination, and nearest neighbor classification. It was discovered that the neural networks and clustering methods were both very sensitive and superior to linear discrimination methods. Furthermore, using neural networks for classification does not require that any assumptions be made about the probability distribution characteristics of the data, which is a definite advantage.

Implementation of several types of neural networks in the role of process fault diagnosis was evaluated by Sorsa and Koivo (1993). Networks that were compared include the multilayer perceptron, radial basis function, nearest-neighbor rule, ART2, and Kohonen feature maps. They pointed out that, in practice, the definition of different fault situations is a difficult problem. As a result, neural networks trained in an unsupervised learning mode provide a promising method for fault detection and diagnosis and that the next step is to experiment using this technology on real processes.

Hsu and Yu (1992) have incorporated self-learning, based on the reinforcement learning feature of a neural network, into a qualitative/quantitative model-based diagnostic system. They noted that a self-learning feature makes the qualitative/quantitative model-

based diagnostic system more attractive in practical applications since it requires much less engineering effort.

Kavuri and Venkatasubramanian (1993) replaced the linear activation function typically used in feed forward neural networks with an ellipsoidal one and developed an algorithm that would generate or terminate hidden nodes. The use of an ellipsoidal activation function provides bounded regions and, as a result, the network is able to overcome the problem of generalization that is usually encountered when using linear activation functions. Making the network structure adaptive allows the network to increase in size so that it has sufficient representational capacity or to decrease in size by eliminating nodes that are not contributing to the representation.

Neural Networks in Advanced Control Strategies

Various types of neural networks have been employed for use in advanced process control strategies. Su *et al.* (1992a) used recurrent neural networks to obtain a process model for use with what they call a neural network model predictive control algorithm (NNMPC). They claim that it is a DMC-like, model-predictive approach. In this setup, the neural network is used to obtain multiple future predictions of plant behavior, based on a combination of present and past information, while also optimizing the future trajectory to the desired setpoint when selecting each control action. This strategy was implemented with great success using a simulation of a complex industrial reactor that consisted of more than forty coupled nonlinear differential equations that had been obtained from first principles modeling.

A neural network was used in a generalized predictive model-based control algorithm to control an experimental furnace by Sheppard *et al.* (1992). The neural network was trained with only a small amount of data from one furnace experiment, yet performed extremely well when used for control within the bounds of the networks training experience.

Ishida and Zhan (1993) have developed a control strategy for a MIMO process with time delay that incorporates four different neural networks consisting of two prediction networks and two control networks.

Use of neural networks to assure quality control in batch processes was investigated by Joseph and Hanratty (1993). They successfully demonstrated that a neural network model of a nonlinear batch process can be used as the process model in a MPC scheme to assure product quality.

The problem of how to use neural networks for control of a system without an objective function was addressed by Hoskins and Himmelblau (1992). They developed a control strategy, comprised of both an evaluation network and an action network, which relies on reinforcement learning. As a result, the control engineer must apply his/her knowledge to specific subgoals or additional appropriate criteria in order to obtain intelligent control. These criteria are crucial to the effectiveness of the reinforcement signal.

An integrated control architecture for complex systems was proposed by Lu (1992). This general strategy included a control and/or decision mechanism, which could be an explicit model, a neural network, use fuzzy logic, or any combination of these. It

also provided for learning and adaptation of the control and/or decision mechanism and included human input.

Hernandez and Arkun (1990) developed an extended DMC algorithm for control of nonlinear systems when the process model is specified by a neural network. One interesting aspect of their algorithm was the input/output structure of their neural network. They did not input values of the predicted variable ranging from oldest to most recent, as was done with the controlled variable. Instead, they made the window of the predicted variable smaller by moving the most recent input to the network back in time. The purpose for this was that, since multiple future predictions needed to be made at each control interval, the uncertainty in the predictions could be decreased by not allowing prediction error to compound with each future time step.

Control-relevant properties of neural network models were investigated by Hernandez and Arkun (1992). This included stability of equilibrium points and stability of the inverse model dynamics. Several examples were provided to support the theory that was developed. An extended horizon controller was evaluated, and it was found that it could provide stable control of a nonlinear system around a stable equilibrium point as long as the selected horizon was large enough.

Psichogios and Ungar (1991) presented an excellent paper studying both direct and indirect IMC and MPC control strategies for SISO systems using neural networks as both models and controllers. Performance was also compared to that when using a linear regression model in each indirect control strategy. The linear regression model was found

to be inferior, as was expected, since the process they were testing the strategies on was a nonlinear continuous stirred-tank reactor (CSTR).

Three different nonlinear controller strategies were compared by Piovoso *et al.* (1992) and included generic model control (GMC), global linearizing feedback (GLF), and IMC. The IMC controller used a neural network as the process model and was referred to as an IMC-NN scheme. The GMC was implemented using a neural network as a functional approximator, called GMC-NN, in addition to just using the equations that described the process.

Improving the Performance of Neural Networks

Kramer *et al.* (1992b) have studied the use of a hybrid network that incorporates both neural networks and first principles models. They developed a new hybrid network architecture that accounts for constraints that must be satisfied for all future network inputs. The performance of both backpropogation and radial basis function neural networks in this proposed strategy was evaluated. It was found that using a radial basis function network assures that the hybrid model predictions conform to prior knowledge in the absence of calibration data.

Su *et al.* (1992b) developed a method of training neural networks that can increase their accuracy when used in advanced control strategies requiring multiple future predictions, such as MPC. What they have essentially done is, rather than training the network as a one step ahead predictor and then simply chaining it to itself as many times as needed to obtain the desired future prediction, they chained the network to itself during training. They refer to this procedure as a parallel identification method. They evaluated

it using both feed forward and recurrent neural networks to obtain long-term and multiple-step predictions and found the recurrent network to be superior. Therefore, they recommended using a recurrent network for MPC applications.

The idea of hierarchical neural networks was investigated by Mavrovouniotis and Chang (1992). Their hierarchical network consisted of individual subnets combined to form the complete network, with the idea that each subnet will capture some particular aspect of the input data. They claim that organizing the variables into related sets and then structuring the network as a multiple hierarchy of subnets supplies the neural network with hints as to which directions are the most promising to find patterns. However, this approach requires some *a priori* knowledge of the structure and behavior of the system being modeled.

A discussion on combining expert systems and neural networks to form expert networks was given by Caudill (1991). Several different strategies were evaluated. These included divide and conquer, embedded neural networks, explanation by confabulation, and artificial expert.

Other Applications of Neural Networks

Rehbein *et al.* (1992) recommended various possible applications of neural networks in the process industry. These included the following: process models, process optimization, open-loop advisory systems, prediction of product quality values, predictive, multivariate, statistical process control, predictive maintenance scheduling, sensor validation, and closed-loop real-time control. They also noted that some chemical companies have had significant success using neural networks.

Autoassociative neural networks have been investigated by Kramer (1992a) for use in noise filtering, missing sensor replacement, and gross error detection and identification. A comparison was made with linear methods of noise filtering and gross error removal and it was determined that the autoassociative network performed much better.

Neural networks were used successfully by Osborne (1992) to obtain more accurate estimates of reservoir permeability. Use of neural networks as opposed to regression methods almost doubled a correlation coefficient when comparing core-derived permeability to predicted permeability.

Neural Networks as Inverse Process Models

A paper by Kasparian and Batur (1992) presents a neural network structure that employs two feed forward neural networks. One network learns the forward dynamics of the process to be controlled while the other network learns the inverse dynamics of the neural network process model. They evaluate the performance of the proposed neural network control structure on a dynamical second order simulated process.

An article by Moran and Nagai (1993) presents the method used to obtain a neuro-observer that identifies the inverse dynamics of the front suspension of a vehicle. This allows front road disturbances to be identified so that they can be used to improve the response of the rear suspension.

Kyung *et al.* (1994) present a nonlinear compensation method for trajectory control of robotic manipulators based on multi-layered neural networks. A simple acceleration based learning scheme has been proposed for the promotion of the adaptation capability of the neural network feedforward controller. The feasibility of the proposed

learning scheme was demonstrated through computer simulations compared with the conventional learning scheme.

Nikolaou and Hanagandi (1993) presented an integrated methodology for the modeling and controller design of nonlinear dynamical systems. Their three-step methodology was tested on a CSTR and shown to perform better than a linear, optimally-tuned controller.

Normandin *et al.* (1994) considered the control of a continuous stirred tank fermenter using a neural network model in a predictive control strategy. It was shown that a relatively simple neural network, developed as a one-step ahead predictor, can be used recursively to predict accurately the biomass and the substrate concentrations many sampling periods in the future.

Eskandarian *et al.* (1994) developed a hybrid dynamics-CMAC (Cerebellar Model Arithmetic Computer) algorithm which has the advantage of reduced memory requirements and improved computational speed over the previous application of CMAC as a trainable and learning robot controller. Test cases indicated a successful application of the developed hybrid dynamics-CMAC method for simulation as well as control of robotic manipulators.

Zhang *et al.* (1994) developed and implemented a prototype neural network-based supervisory control for *Bacillus thuringiensis* fermentation. The results from the simulation and experimental results of the neural network controller were compared. The technique was capable of improving the control performance of the fermentation process.

Morris *et al.* (1994) provide a good summary on the current status of neural networks and their role in process control.

Nahas *et al.* (1992) propose a strategy for neural network models in nonlinear internal model control (NIMC). The NIMC consists of a model inverse controller and a robustness filter with a single tuning parameter.

Thibault and Grandjean (1991) have provided an excellent survey paper on neural networks. The paper reviews the fundamentals of feedforward neural networks as well as the various neural network-based control strategies, making use of the plant and/or plant inverse neural models.

CHAPTER III. INVERSE PROCESS MODELS

What is an Inverse Process Model

With any process there are inputs and outputs, and, for any given set of inputs a unique set of outputs is generated. That is, there is a distinct mapping between the inputs and outputs of a dynamic system. A forward process model should be able to accurately describe this mapping. Figure 3 represents a generic forward process model where inputs A and B produce an output C. For many processes, this relationship is easy to obtain through a first-principles analysis of the system. For nonlinear systems, the system can be modeled using nonlinear equations that are eventually linearized for use in a control strategy. Other methods obtain process models through the use of open-loop response data. These models assume a general form, usually a first- or second-order model with time delay, for use in a control strategy. While all of these methods can be very powerful, there are systems where this type of model approximation will not provide adequate controller performance. This often occurs when a system has variable time delays that are difficult to model or when some of the processes are not fully understood, or both. Furthermore, some systems are very complex, and the interactions between various process input and output variables may be difficult or impossible to predict and model.

An inverse process model is one that is capable of providing the set of process inputs that will produce a given set of process outputs. Figure 4 represents a generic inverse process model, where the output C generates the inputs A and B that produced it. To obtain an inverse process model, also known as an inverse plant, one must know how a

Figure 3. Representation of a Forward Process Model

Figure 4. Representation of an Inverse Process Model

system behaves in reverse. While this backwards process is sometimes intuitive, it usually involves enough parameters to make the reverse reasoning process extremely difficult. For example, consider a process where two numbers are summed. Because one knows how the process works, one can predict the correct output given any two inputs. However, one might not understand the process well enough to predict the relationship in reverse, or, as in this case, it may be too complicated, with any given output being generated by more than one set of inputs. For example, if a given output was four, the inputs could be zero and four, one and three, two and two, three and one, or four and zero. The problem becomes even more complex if some of the combinations are physically impossible based on the previous behavior of the system.

Why Use an Inverse Process Model

The main advantage of an inverse process model is that it does not have to be inverted each time a control move is needed. It can be difficult and computationally inefficient to solve for inputs using a forward process model. Furthermore, convergence to a solution is not guaranteed. While this might only present a small difficulty when dealing with SISO systems or smaller MIMO systems, it can become virtually impossible to solve large MIMO systems using conventional numerical methods. While it is possible to obtain very good forward process models for multivariable control, the limiting factor is having the capability to mathematically solve for the inputs. This is where the inverse process model has a great advantage. The only fundamental differences between implementing an IMC controller using an inverse process model instead of a forward

process model are that the control inputs are calculated explicitly, removing the need to numerically solve for them, and model mismatch is accounted for by adjusting the controller output instead of the setpoint.

There are also many other reasons to use an inverse process model for control. First, a perfect inverse process model would allow one to implement a perfect servo controller. Figure 5 shows a servo controller, where R is the setpoint, C is the output from the process G, and P is the input to the process from the controller Gc*. Gc* is the inverse of the process G and usually includes a filter to insure that the transfer function is proper or that derivative action is disabled. Second, it would provide offset-free control in an IMC control strategy (see Figure 2) and have the ability to compensate for disturbances much better than traditional control methods. Another advantage to having an inverse process model is that it can look at past and current process information and predict future problems, whether they might be offsets from a setpoint change or some type of disturbance, before a conventional feedback controller. Furthermore, a model-based control strategy will minimize overshoot associated with setpoint changes since anti-windup effects are intrinsic to it. Finally, a model-based control system has the potential to be adaptive. The inverse model can be updated on-line as needed to insure that the process is always accurately modeled.

Figure 6 shows the inverse neural network process model architecture used in this study. It has eleven inputs, one output, and two hidden layers, with eight neurons in the first and three neurons in the second. In the figure, MV represents a coolant valve signal and CV represents the reactor temperature. The coolant valve signal is the manipulated variable and the reactor temperature is the controlled variable. The network outputs the

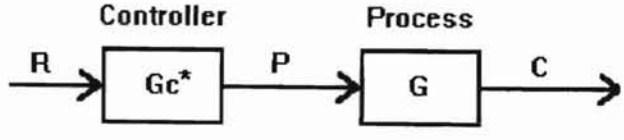

Figure 5. Servo Controller

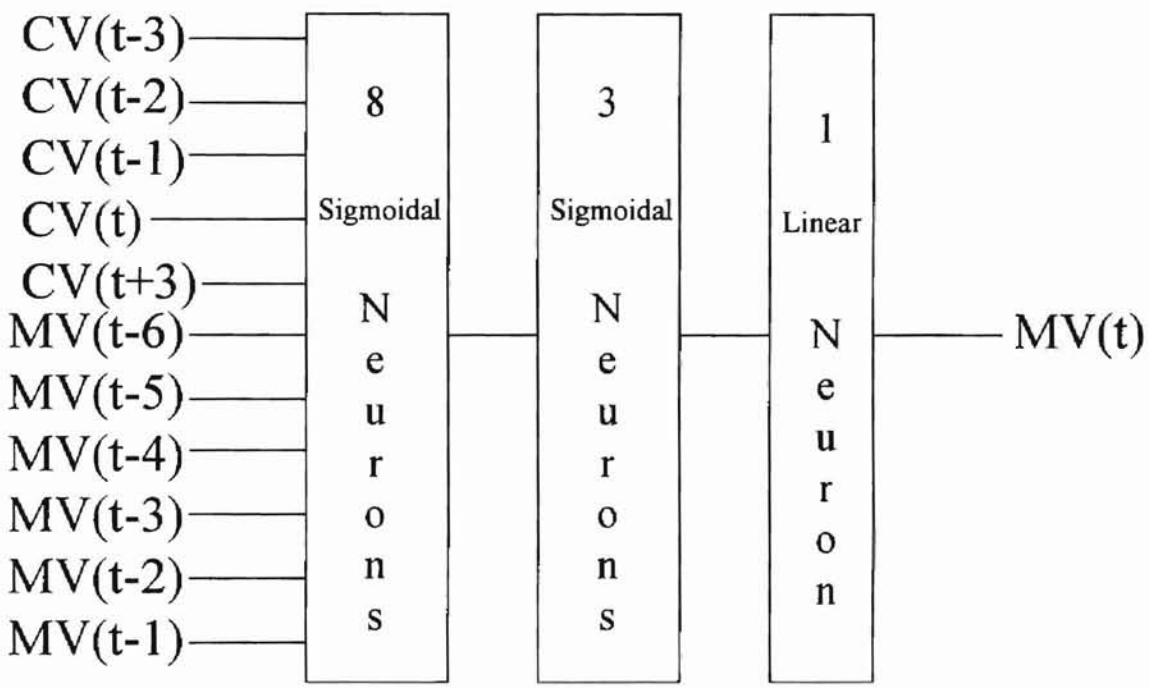

Figure 6. Inverse Neural Network Process Model Structure

coolant valve signal, MV(t), that will achieve a desired reactor temperature, CV(t+3). In addition to the desired reactor temperature, the network also has as inputs the current and past three values of the reactor temperature (CV(t), CV(t-1), CV(t-2) and CV(t-3), respectively) and the past six signals to the coolant valve (MV(t-1), MV(t-2), MV(t-3), MV(t-4), MV(t-5), MV(t-6)).

To further clarify how this model was used in an IMC control strategy, consider the inverse neural network model control structure shown in Figure 7. In the figure, TDL represents transmission delay lines that simply sample and hold, at various multiples of the sampling time, the coolant valve signal input to the process and the reactor temperature output from the process to produce the necessary time history profile for input to the control algorithm. The control algorithm contains the neural network model within it. There are two primary functions of the control algorithm. These are to calculate a value for the desired future reactor temperature input, CV(t+3), based on the setpoint and current reactor temperature, CV(t), and to correct for model mismatch. The formula used to obtain CV(t+3) is

$$CV(t+N)=CV(t)+(Setpoint-CV(t))/(CHSF) \qquad (2)$$

where CV(t+N) is the desired reactor temperature N sampling periods into the future that the neural network model was trained with (in this case N=3) and CHSF is the control horizon scaling factor. CV(t+N) will equal the setpoint when CHSF equals one. To effectively double the number of sampling periods used to achieve the desired future reactor temperature, one would set CHSF equal to two. This would change the effective control horizon from N to 2N sampling periods and approximate the behavior of a neural

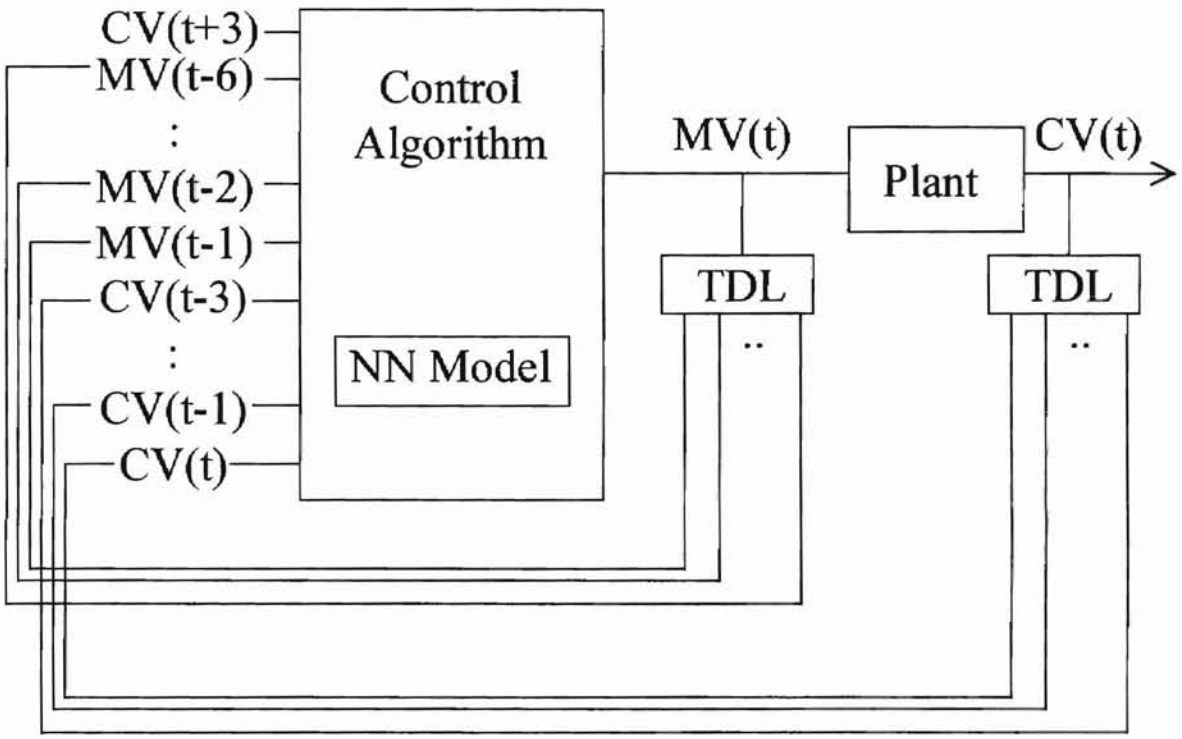

Figure 7. Inverse Neural Network Control Structure

network model trained using CV(t+2N) as the desired reactor temperature 2N sampling periods into the future. It should be noted that the control horizon must always be greater than the dead time of the system. The effects of adjusting the control horizon will be discussed later.

Model mismatch is compensated for by stepping back in time and calculating the control signal that would have been input to the process to produce the current output. The difference between the control signal that was actually input to the process and the one that would have been input to produce the current output is then added to the current control signal output.

The Function Approximation Problem

One is essentially approximating a function when training a neural network. For example, one could take a data set generated using an algebraic formula and then train a neural network to learn that formula. The resulting network would be able to accurately predict the output of any given input over the range of inputs used in training. While a neural network can extrapolate out of the region in which it has been trained, it is generally not a good idea to do this. Usually, the neurons in a network will saturate when it attempts to extrapolate too far from its trained region and, as a result, will output a constant value at some point.

The step test data that is being approximated is simply a square wave with varying amplitude. A square wave can be approximated using an infinite series of sines or cosines,

also known as a Fourier series. Each neuron in a neural network is synonymous with one sine or cosine term in a Fourier series. A square wave is defined as,

$$f(t) = \frac{4A}{\pi} \sum_{n=1,3,5,}^{\infty} \frac{1}{n} \sin(n\omega t) \qquad (3)$$

where t is time, ω is the frequency, A is the amplitude, and n is a harmonic frequency of f(t). Thus, a neural network would need an infinite number of neurons to be able to approximate the square wave perfectly. Because it is not computationally feasible to use a very large number of neurons in a network, the accuracy that can be achieved with a finite number of neurons is not adequate to approximate the step test data. However, this is not the case with the ramp test data.

The ramp test data is essentially a triangular wave with varying amplitude. A triangular wave can also be approximated using a Fourier series. A triangular wave is defined as,

$$f(t) = \frac{8A}{\pi^2} \sum_{n=1,3,5,}^{\infty} \left[\frac{1}{n^2} \sin\left(\frac{n\pi}{2}\right) \right] \sin(n\omega t) \qquad (4)$$

where t is time, ω is the frequency, A is the amplitude, and n is a harmonic frequency of f(t). Like the square wave, the triangular wave will also need an infinite number of terms to make a perfect approximation. However, it will not need as many terms to achieve the same accuracy as the square wave since the changes in slope are not as radical. In other words, the square wave is a series of infinite slope changes and, as a result, is very difficult to approximate. The triangular wave, however, is a series of more gradual changes and therefore requires fewer terms to obtain a sufficient approximation. Figure 8 shows four different approximations of square and triangular waves. The number of terms used for

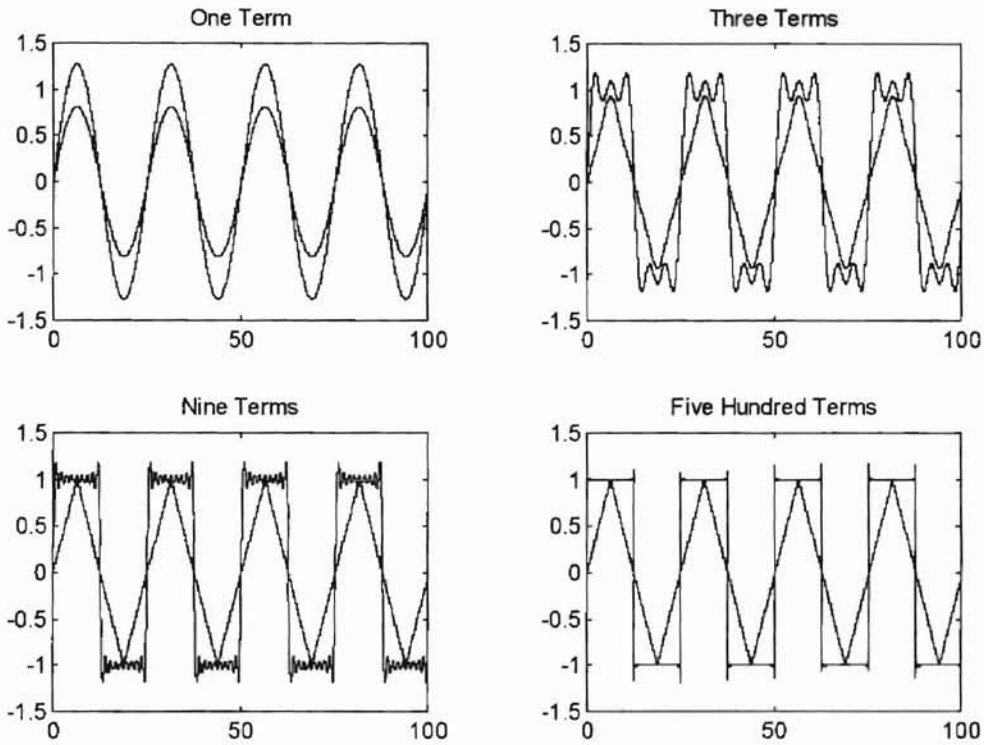

Figure 8. Effects of the Number of Terms Used in a Fourier Series on the Accuracy of Square and Triangular Waves

each (1, 3, 9, and 500, respectively) is shown in the figure. One can easily observe that a triangular wave is much easier to approximate than a square wave. The bottom line is, if the training function has many changes with infinite or almost infinite slopes, it will be very difficult to obtain a good function approximation without using a large network with many neurons. However, if the training function has slope changes that are more gradual and less abrupt, it will be possible to obtain a satisfactory network with fewer neurons.

A major consideration when designing a neural network is what type of training set is to be used. There are two types of data that are usually used for network training: simulation data and actual process data. When a simulator is being used to generate training data, it is very easy to control the level of excitation and make sure that all types of process behavior will be represented in the training set. However, one must also be careful to not over-excite the system, since this can cause many problems during training.

One problem that may arise if a highly excited training set is used is that the network will exhibit an inability to properly relate the process variables. That is, if the changes in process variables are too abrupt, the network will never get the opportunity to properly learn how they relate to one another. The other problem that can come about relates to the previously discussed problem concerning the type of signal used. Abrupt changes in process variables coincide with infinite slope approximation, which is a very unfavorable situation.

These problems are alleviated when actual process data can be obtained. However, new problems take their place. These include redundancy in the training data,

noisy data, and long periods of inactivity. All three of these problems can be handled by preprocessing the data so that noise is reduced or eliminated, and the inactive periods and repeated or similar data are removed from the training set. However, one must be careful not to remove too much from the training data or the data set will not be rich enough to provide an accurate process model. Furthermore, there may be relationships between the process variables within the data that are not evident to the person editing the data but may be crucial to the network performance.

There is a trade-off that must be made between how much to excite the network and what type of function will be used when using both types of training data. If you use a signal with infinite slopes, the network will have a hard time approximating them, but, if you use a signal that is less aggressive, you will run the risk of not producing a rich enough training set. If you use a signal that is highly excitatory, whether it has infinite slopes or not, you will run the risk of inhibiting the ability of the network to learn the proper relationships between process variables, but, if you use a signal that does not provide sufficient excitation, you will end up with a process model that is not very accurate and will not be sufficient for model-based control. The correct combination of these two factors is what must be ultimately determined by the user when generating an inverse process model.

Anticipated Improvements Using Ramp Inputs

There are two main improvements that should come about from using a ramped input training set, as opposed to one that is stepped, once the network is integrated into a model-based control strategy: less overshoot and decreased settling time.

The amount of overshoot should be reduced when a ramped input training set is used since a more accurate function approximation can be obtained. If one considers the development of an approximate function for a square wave, one knows that, as each successive term is added, the front of the wave becomes less spiked and the slope of the top of the wave approaches zero. The number of neurons in a neural network is fixed at some predetermined value; thus, there is already a limit on the accuracy that the network can achieve while trying to approximate the stepped function. After training a network with this type of input, one will usually observe the following phenomenon. The larger the step change, the more overshoot, in the form of a spike, there will be, and, consequently, for smaller step changes, the overshoot will be minimal. Thus, one can conclude that the network has inherently learned an improper behavior. That is, the larger the step change, the more it will overcompensate. Therefore, once the model is being used for control purposes, it will continue to overcompensate, whether during a setpoint change or when rejecting a disturbance.

Ramped inputs can virtually eliminate this problem, since this type of phenomenon is almost completely abolished while using the same number of neurons and network architecture. Because the ramped inputs are much easier to approximate, they allow a much more accurate process model, *i.e.*, one that more closely approximates the training

data, to be obtained. Thus, the network does not learn an improper behavior and, as a result, when implemented into the control strategy, will provide control with less overshoot than that observed when using stepped training data.

Another advantage that ramped inputs have over stepped ones is that the settling time will decrease. Just as the step-trained networks have a tendency to overshoot, they also tend to be oversensitive. This behavior is primarily a result of the way the model-based controller is implemented. The step-trained controller must use steps to make changes to the input. This is sometimes too aggressive and results in extended oscillation about the setpoint.

On the other hand, the ramp-trained controller uses a ramp to make changes to the input. The "ramp" is actually a discrete ramp in the sense that it is essentially made up of a series of equal, smaller steps. Overall, this type of signal is less aggressive and results in smoother transitions during setpoint changes and disturbance rejection. Furthermore, it has a much smaller settling time, since it is not as burdened with having to correct for overaggressive changes to the input.

The overshoot and settling time can be directly related to how sensitive the model is. If the model is too sensitive, the network will behave like the step-trained model. It should be noted that this type of behavior might be desirable if the network is to be used more for disturbance rejection than for setpoint changes. If the network is not sensitive enough, it will result in sluggish setpoint changes and poor disturbance rejection. The ramp-trained network has the ability to perform both tasks well.

Simulated Nonlinear CSTR with Time Delay was Implemented to Study Performance

In this study, a simulation of a nonlinear CSTR was used to evaluate the closed-loop performance with and without model mismatch correction of several inverse process model controllers. The reaction is exothermic and the exit concentration of the reactor needs to be controlled. The two manipulated variables in the system are the volume of the reactor, regulated by changing the level, and the flow rate of water through the cooling jacket. It was determined that, while the level in the tank does affect the outlet concentration of the reactor, the coolant flow has a much larger effect. This is because the temperature of the reactor, changed by adjusting the coolant flow, is more important than the residence time in determining the extent of conversion. Thus, it was determined that the controller would be SISO, with the coolant flow rate as the manipulated variable and the reactor temperature as the controlled variable, since temperature is much easier to control than concentration. Furthermore, it was determined that regulatory control would be adequate for control of the level. Figure 9 shows the CSTR used in this study. As illustrated, the coolant inlet flow rate will be manipulated to control the reactor temperature, which will consequently change the outlet concentration.

The training data was obtained by running the reactor temperature control in open loop mode. The training signal was input to the coolant flow valve, the manipulated variable used to regulate the temperature of the reactor. It was important to leave the reactor level in closed loop operation and maintain a constant reactor level since changes in the level would have affected the reactor temperature. Two types of signals, stepped and ramped, were utilized. Figure 10 contains examples of typical step and ramp signals

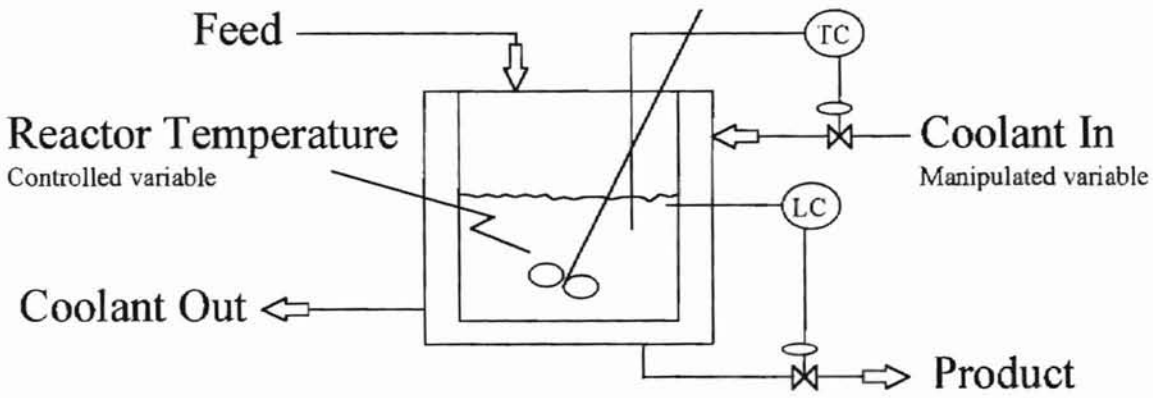

Figure 9. Nonlinear Continuous Stirred-Tank Reactor System Used in this Study

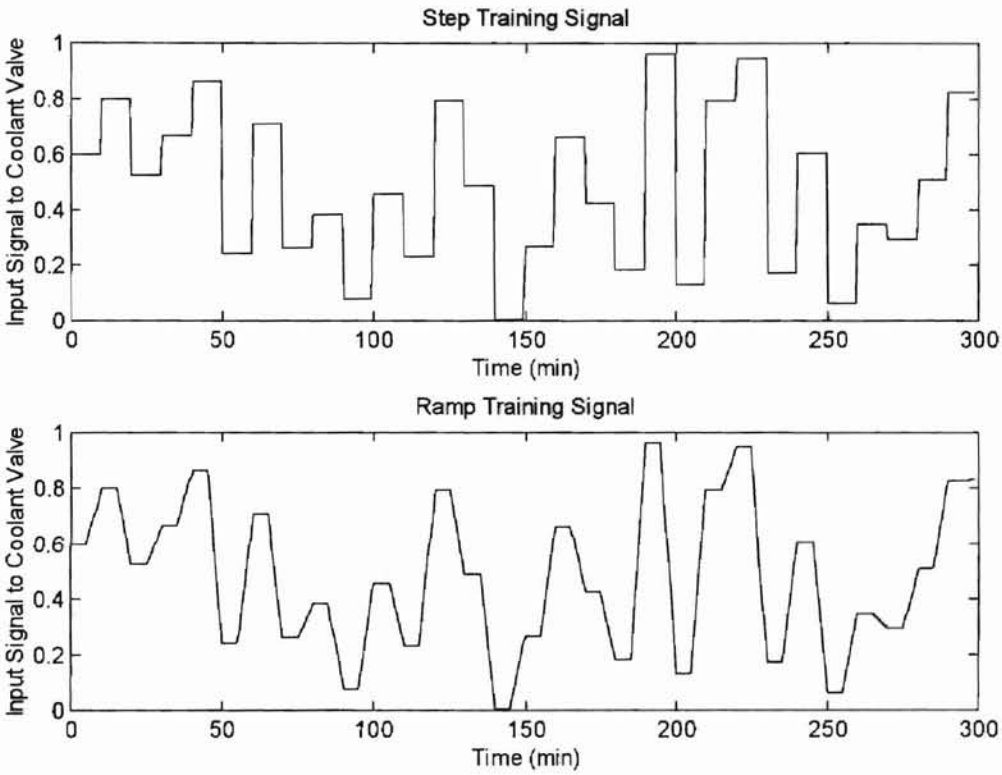

Figure 10. Examples of Step and Ramp Input Training Signals

used to generate training data. The number, period, and magnitude of the changes for both types of coolant flow input signals were the same. A total of 30 changes of varying magnitude and direction were made every 10 minutes. The step input signals reached each new value immediately whereas the ramp input signals required 5 minutes. This resulted in a 9 minute relaxation period for the step input signals and a 5 minute relaxation period for the ramp input signals. The relaxation periods between changes aided in the networks ability to learn the inverse process dynamics.

The step tests were performed and the resulting system output was saved. Ramp tests were then performed using the same parameters discussed above. These data files were then used to generate training sets for various neural network inverse process models. The same data sets were used in many different network architectures. All neural networks used in this study were feed forward networks and employed nonlinear logarithmic-sigmoidal activation functions defined as $f(x)=1/(1+\exp(-x))$. Networks were batch trained using the Levenberg-Marquardt algorithm (Marquardt, 1963) and were validated using an independent set of test data. The network structure used for all inverse process models was seen earlier in Figure 6. There are 8 neurons with logarithmic-sigmoidal activation functions in the first hidden layer, 3 neurons with logarithmic-sigmoidal activation functions in the second layer, and 1 neuron with a linear activation function in the output layer. While determining the size and structure of a neural network is not a straightforward task, there are some general guidelines that can be followed to obtain a network that will not simply memorize the data set or have poor generalizability. A discussion of the detailed procedure used to generate an inverse neural network process

model can be found in Appendix B. Most of the methods and suggestions are very empirical in nature. However, they all have one element in common: they assume that one has already chosen the network architecture.

For the CSTR process, the network inputs consisted of a time history of the 6 previous control actions and 3 past temperatures, the current temperature, and a future desired temperature. The output of the network is the control signal that would come closest to achieving the future desired temperature. Thus, the inverse process model controller had 11 inputs and 1 output.

The set of inputs was chosen after much empirical investigation. It was found that, because the network was predicting the manipulated variable, there needed to be more information on it than on the controlled variable supplied to the network. This sort of weighting of the inputs resulted in a much more accurate prediction of the necessary control action. Furthermore, the extra control inputs allowed the network to better learn the various dynamics associated with different short-term control histories.

Offset and Disturbance Rejection Capabilities

The classic IMC strategy requires feedback to complete the control strategy. This is because a forward process model prediction is subtracted from the actual output of the system and the difference is then added to the setpoint to drive the system to the desired value. This type of feedback serves two very important functions. First, it provides a method of compensating for any model mismatch errors. Second, it is the sole means in which disturbances are rejected by the controller. Without feedback, an IMC controller

would simply operate blindly in a servo controller mode of operation. The neural network equivalent of this type of controller would simply consist of a neural network forward process model of the system replacing a more conventional type of model, such as a transfer function or state-space model. However, the method in which the models are inverted is quite different.

Transfer functions can be readily inverted, made proper, and then filtered. On the other hand, inverting a neural network, especially one that has more than one manipulated variable, is much more complicated. As a result of this complex optimization problem, obtaining a solution becomes increasingly more difficult as the number of controlled variables increases. Consequently, this is where the power of an inverse process model becomes very evident.

Because an inverse process model only requires process information that is already available (see Figure 6) and generates required control actions explicitly, there is no optimization problem at each control period, which results in a more computationally efficient control algorithm. However, an inverse process model neural network controller also requires feedback for the same reasons as forward process model neural network controllers and classic IMC controllers do.

The best way to begin evaluating how well a neural network controller will perform is to allow the controller to run in closed-loop mode without model mismatch correction. This will provide one with an idea of how much model mismatch is present, how robust the controller might be, whether or not the model is stable, how sensitive it is, and how it will perform. It also provides a good basis to compare the performance of

inverse model controllers designed from two types of training data, stepped and ramped signals.

Closed-Loop Performance Without Model Mismatch Correction

Setpoint Changes

Performance of the inverse step (IS) and inverse ramp (IR) controllers was evaluated by performing numerous setpoint changes of varying magnitudes and directions from different operating points. Setpoint changes over a wide range of magnitudes were made in order to evaluate the performance of the neural network models in terms of overshoot and settling times. Setpoint changes in both a positive and negative direction were made in order to determine how well the model had learned the nonlinear dynamics of the system. This is an important aspect of the neural network controllers, since a conventional controller must be tuned to perform optimally for setpoint changes in both directions, even though there is a different set of optimal controller gains for each direction due to the nonlinearity of the system. Finally, setpoint changes were made from a variety of operating points above and below the one in which the model was trained in order to determine how robust the neural network controllers were.

The controllers could not be operated in an open-loop mode since the neural network models require feedback in the form of past temperatures and valve inputs. However, the controllers could operate in closed-loop mode without model mismatch correction by simply disabling that function of the control algorithm so that it would

calculate a value for the desired future reactor temperature, T(t+3), based only on the setpoint and the current reactor temperature, T(t).

IS and IR Model Error

All inverse models that were trained using ramp data achieved prediction errors an order of magnitude smaller than those trained with step data. Thus, one would expect these models, with the lowest errors, to provide the best performance. Equivalent inverse ramp (EIR) models were generated to demonstrate that better controller performance is linked to model error. To do this, an integral absolute error was calculated for all step models and the EIR models were then trained until the errors were approximately the same. It was virtually impossible to get them equal, since the integral absolute error could drop an order in magnitude in one training epoch.

Results show that EIR models do perform similar to IS models. Furthermore, the results to be presented demonstrate that IR models do provide performance superior to the both the IS and EIR process models. It should be noted that all networks were trained using data in which the magnitudes and durations of the steps and ramps were identical. In addition, all networks were trained using the same set of initial weights. These measures were taken to facilitate as direct a comparison as possible.

The training data used to generate all of the IS process models are show in Figure 11. The test data used to validate all IS process models are shown in Figure 12. Similarly, the training data used to generate all IR and EIR process models can be seen in Figure 13. Figure 14 shows the test data used to validate all IR and EIR process models.

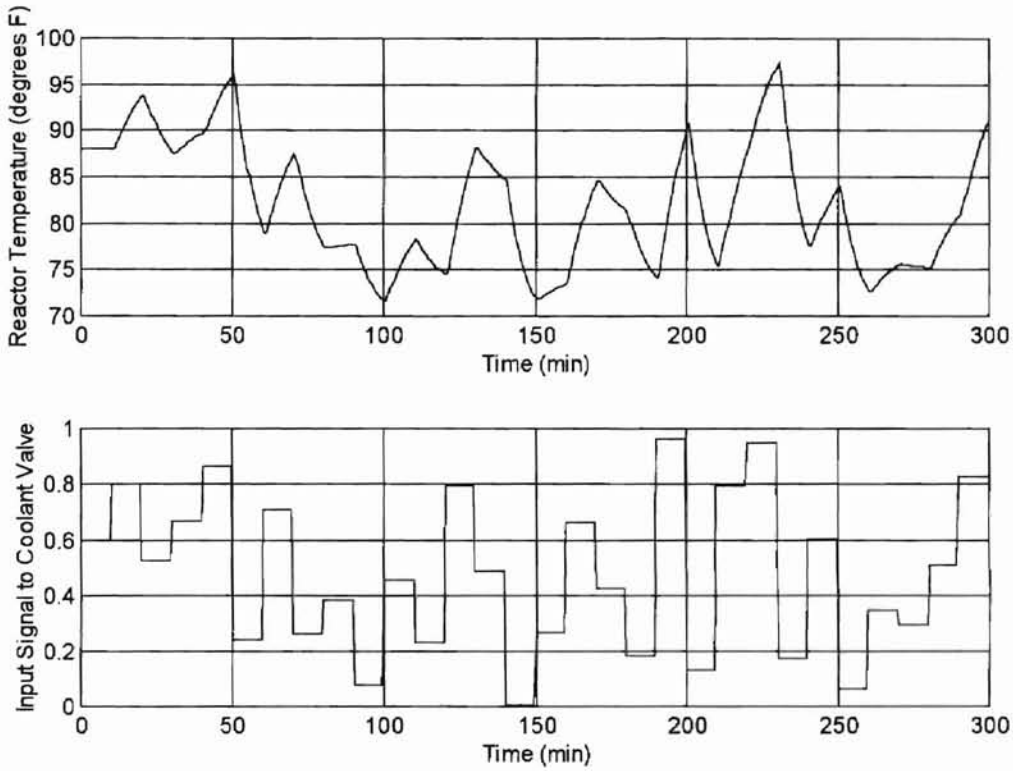

Figure 11. Step Test Data Used as Training Data for all Step-Trained Inverse Process Models

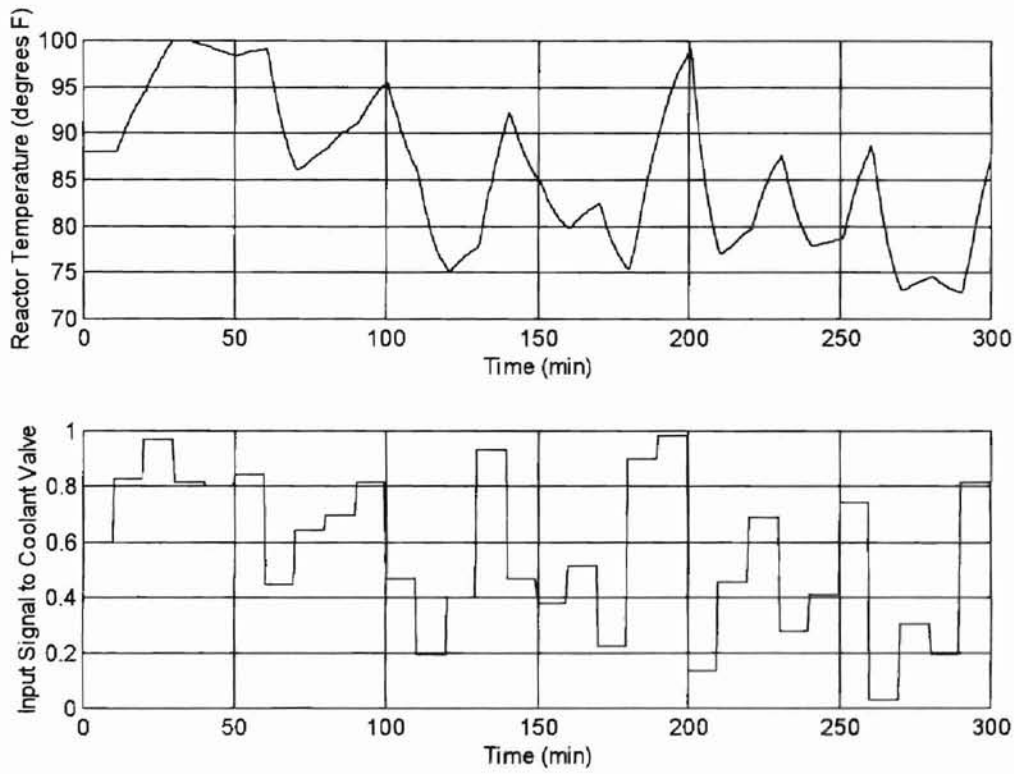

Figure 12. Step Test Data Used as Validation Data for all Step-Trained Inverse Process Models

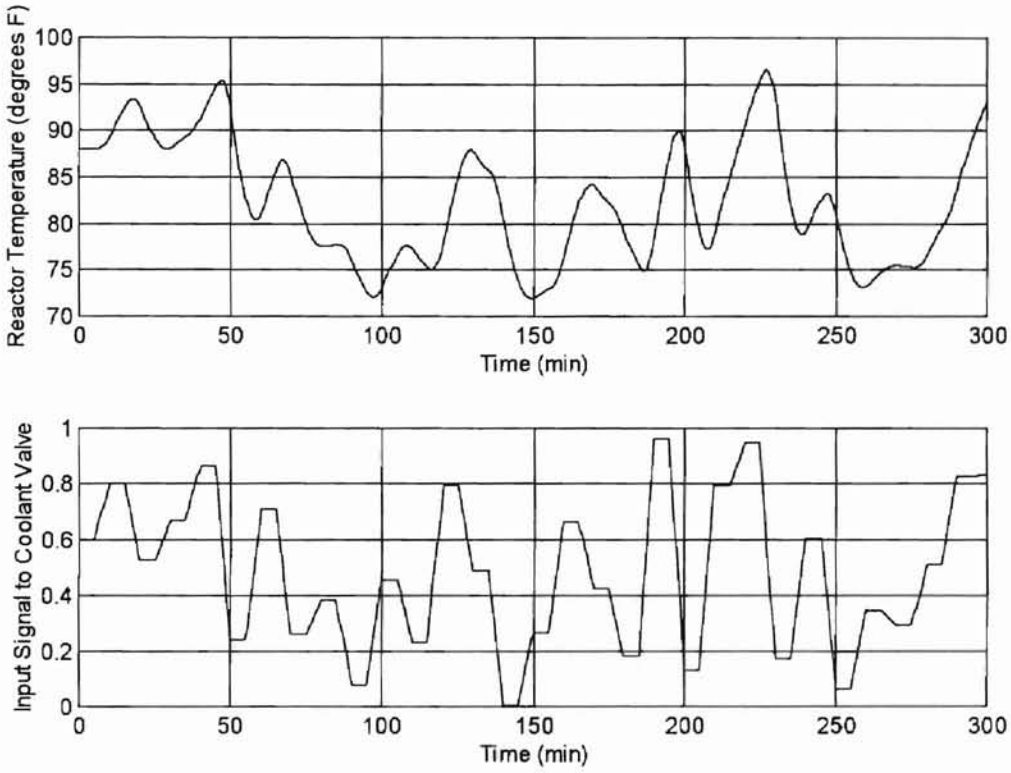

Figure 13. Ramp Test Data Used as Training Data for all Ramp-Trained Inverse Process Models

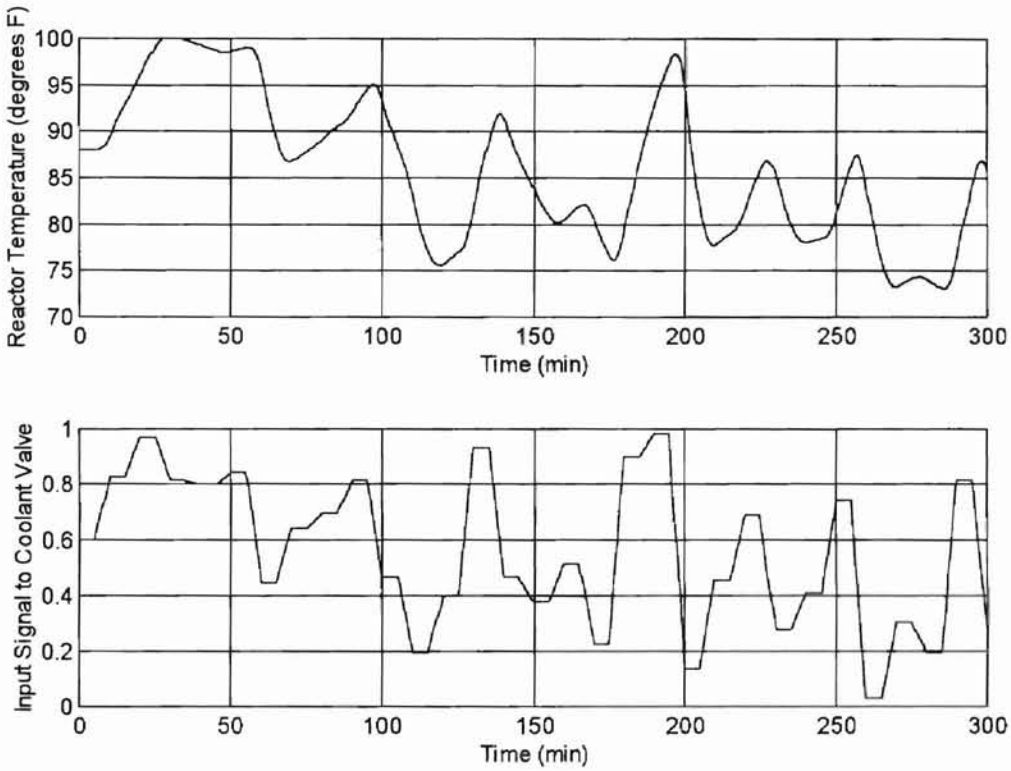

Figure 14. Ramp Test Data Used as Validation Data for all Ramp-Trained Inverse Process Models

Comparison of IS, IR, and EIR Models for Setpoint Changes

Model mismatch correction was not enabled during these tests. Disturbance rejection capabilities were evaluated in later tests, with model mismatch correction enabled.

IS Models

Two different IS controllers were generated. Both had 11 inputs and 1 output. The only difference was in one of the inputs - T(t+i) where i represents the control horizon. For this study, the horizons were chosen to be two and three times the dead time. The sampling time was equal to the dead time. Therefore, two times the delay, or two sampling periods, was the smallest possible horizon and provides the most aggressive control horizon. Three times the sampling period was chosen as the other horizon for this study, since it was found through empirical observations that controller performance became much less aggressive (*i.e.* more sluggish) and, as a result, less favorable, the farther out the horizon was set.

Figure 15 shows the sum-squared prediction error for the T(t+2) IS model versus the number of epochs trained. All networks were trained until the sum-squared error of the test set began to increase. Training was always halted at this point, since any further training would degrade the networks ability to generalize. Figure 16 shows the T(t+2) IS controller predicted input versus the actual input for the training set. This figure is representative of the type of spiking that occurs when step data are used for training purposes. Figure 17 shows the networks predicted input versus the actual input for the test set. This prediction is not as good as the one for the training data and exhibits even more spiking behavior. Figure 18 shows the results of a series of setpoint changes for the

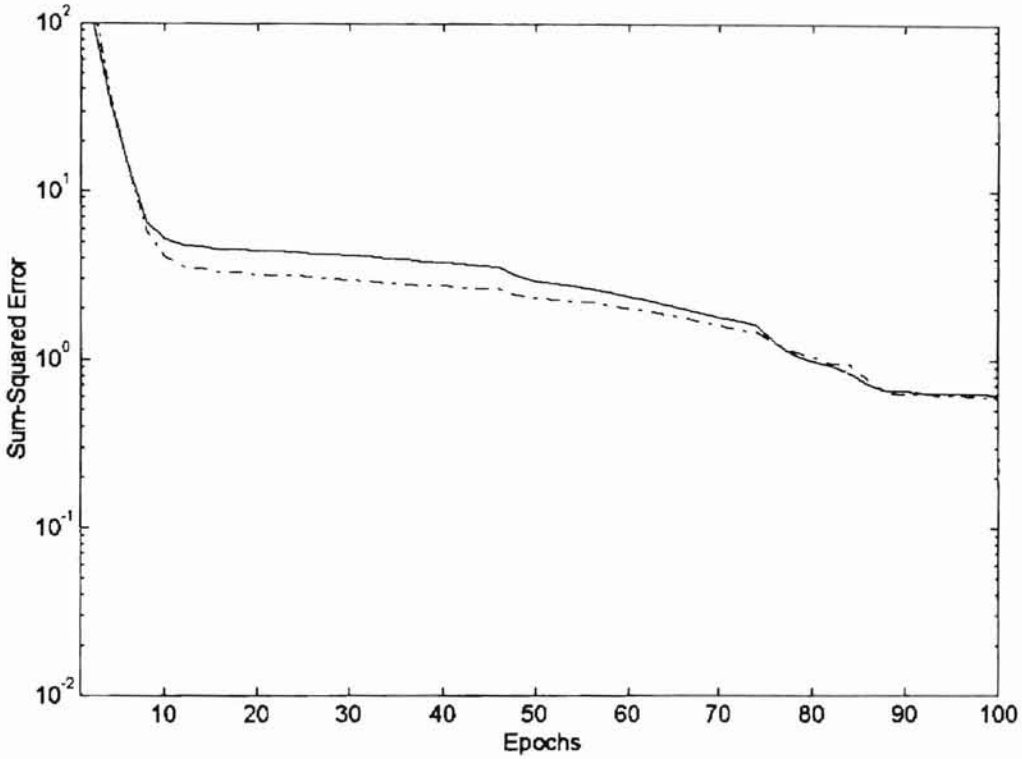

Figure 15. T(t+2) Step-Trained Inverse Process Model Sum-Squared Error as a Function of the Number of Epochs for the Training Set (—) and Test Set (— –)

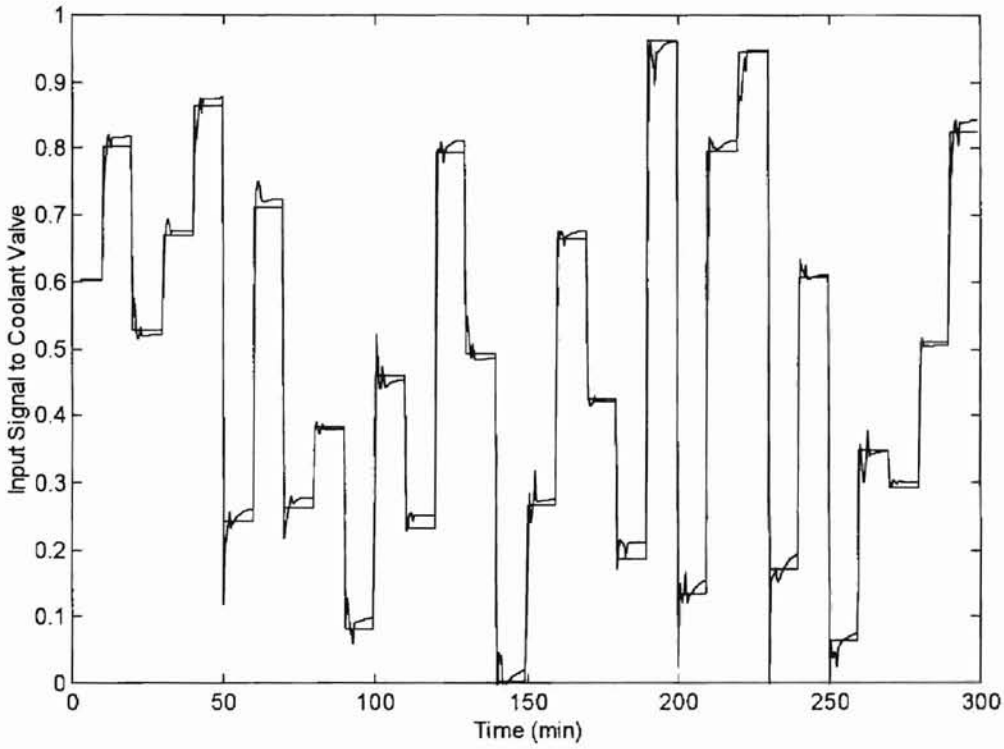

Figure 16. T(t+2) Step-Trained Inverse Process Model Training Set Prediction

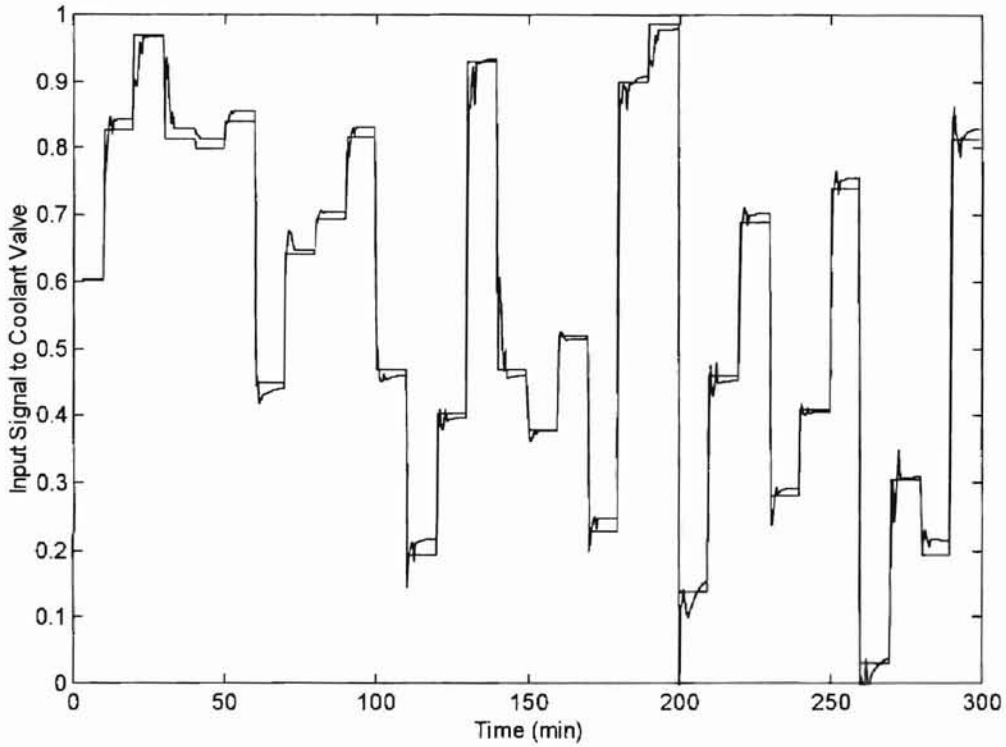

Figure 17. T(t+2) Step-Trained Inverse Process Model Test Set Prediction

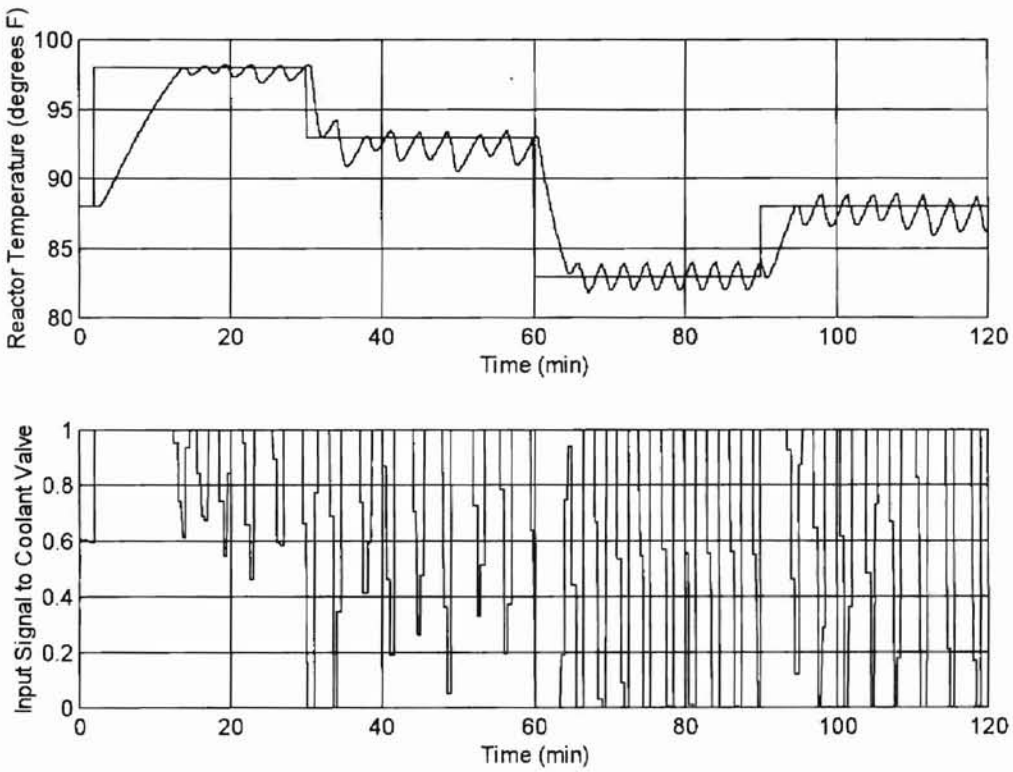

Figure 18. Setpoint Change Performance of the T(t+2) Step-Trained Inverse Process Model

T(t+2) IS controller. The network is clearly unstable at other operating points and has a high sensitivity, resulting in oscillation and poor tracking.

Figure 19 shows the sum-squared prediction error for the T(t+3) IS model versus the number of epochs trained. Figure 20 shows the networks predicted input versus the actual input for the training set. This figure shows the effects of extending the control horizon on the spiking in the networks predictions. The spiking is much less pronounced and a better overall prediction was obtained. These effects make sense, since extending the horizon should give the model a better opportunity to learn the dynamics of the system. Figure 21 shows the networks predicted input versus the actual input for the test set. Since it provides test set predictions that are much closer in accuracy to those of the training set, it can be seen that this model does a better job of generalizing. Figure 22 shows the results of a series of setpoint changes for the T(t+3) IS model. This network is more stable at other operating points, but is still more sensitive than desired.

Figure 23 shows the same series of setpoint changes with a 50% increase in the process dead time. While the overshoot and settling time for each setpoint change increased, the controller still performed well. Figure 24 shows the setpoint change series with the process dead time decreased by 50%. Three of the four setpoint changes had less overshoot and smaller settling times than those in Figure 22. However, the controller performance for the third setpoint change became unstable. The sensitivity problems of the step-trained models are again seen in the last two setpoint changes.

Figure 25 shows the results of the setpoint change series with the heat of reaction increased by 10%. This figure shows an increase in the overshoot and settling time for each step and excessive sensitivity during the last two setpoint changes, where the

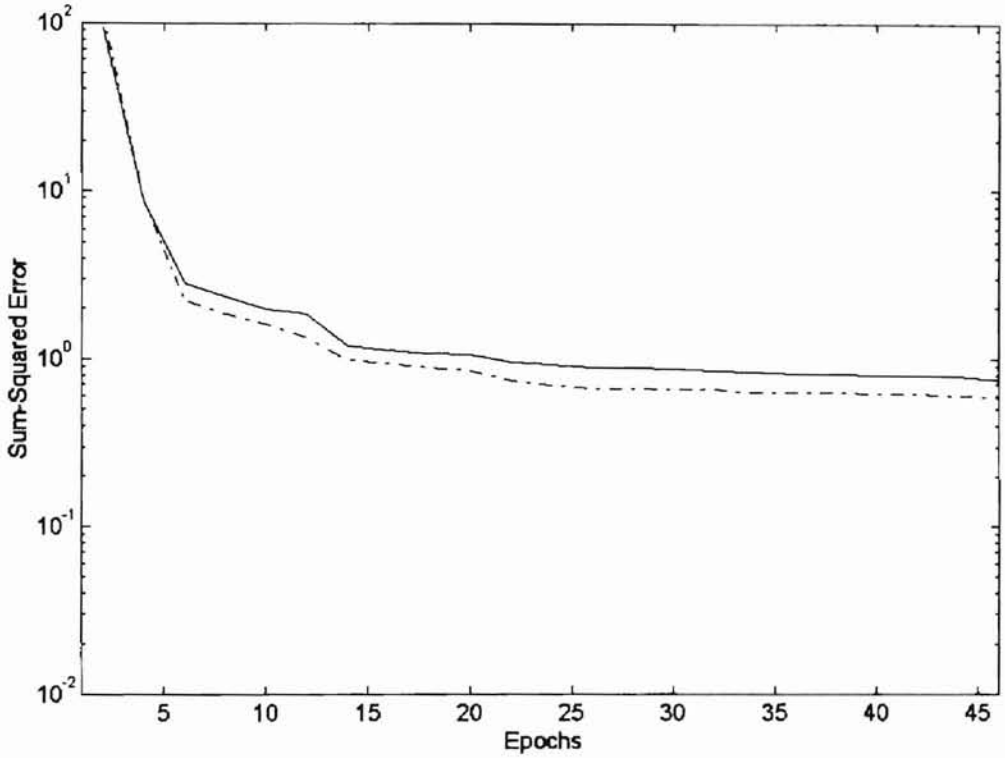

Figure 19. T(t+3) Step-Trained Inverse Process Model Sum-Squared Error as a Function of the Number of Epochs for the Training Set (—) and Test Set (— –)

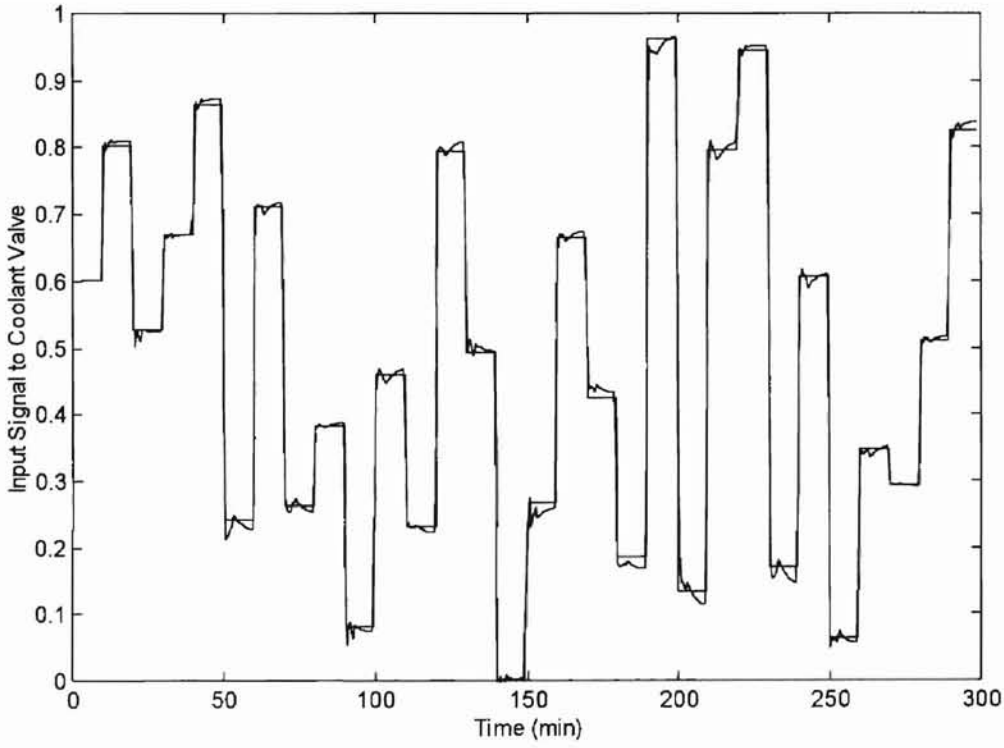

Figure 20. T(t+3) Step-Trained Inverse Process Model Training Set Prediction

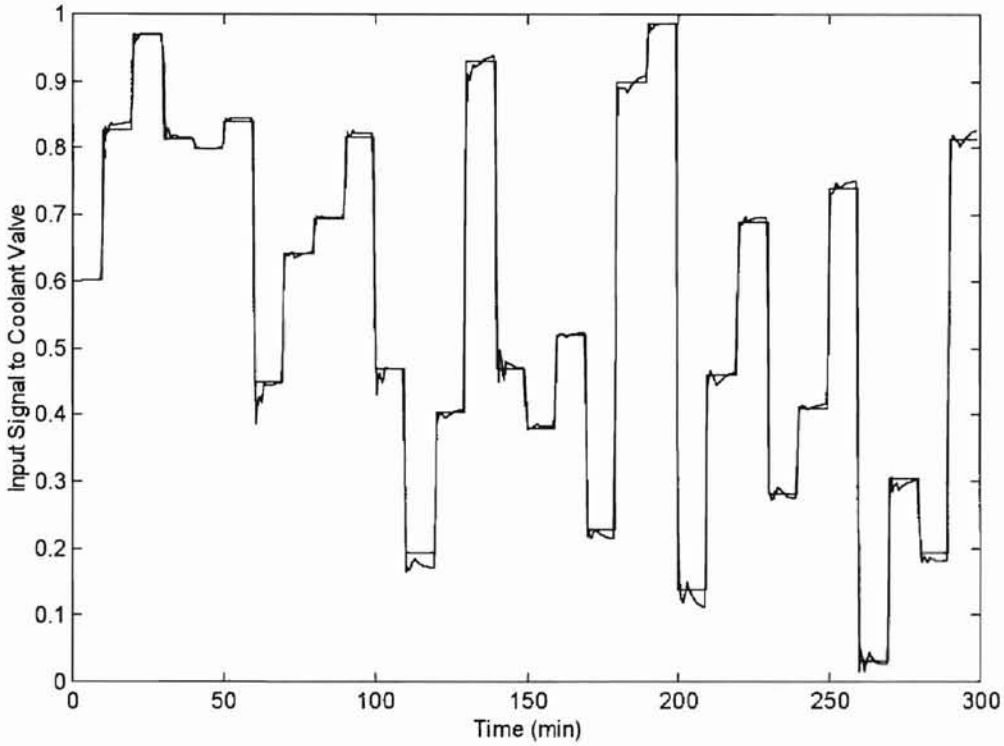

Figure 21. T(t+3) Step-Trained Inverse Process Model Test Set Prediction

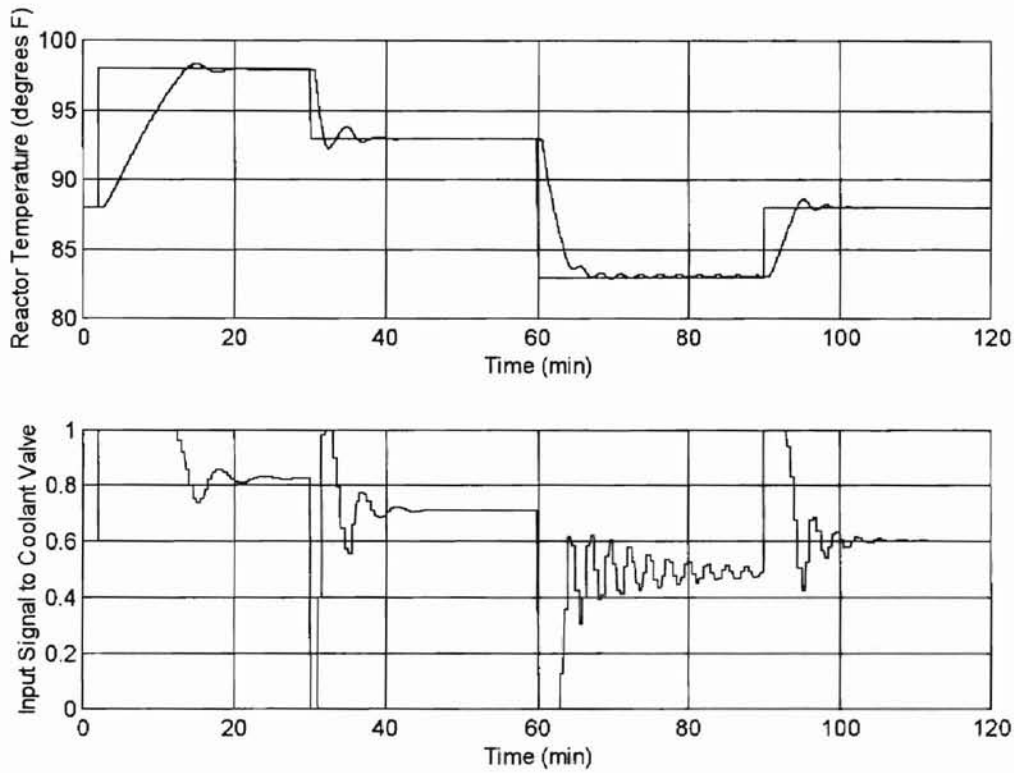

Figure 22. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model

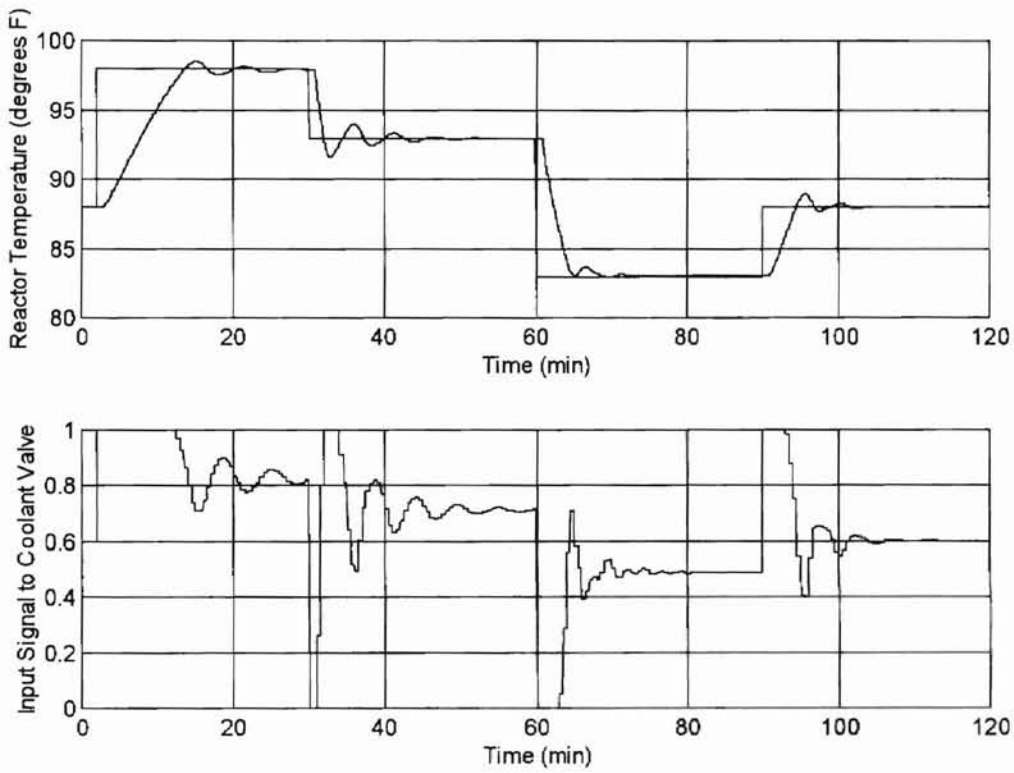

Figure 23. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model with a 50% Increase of the Process Dead Time

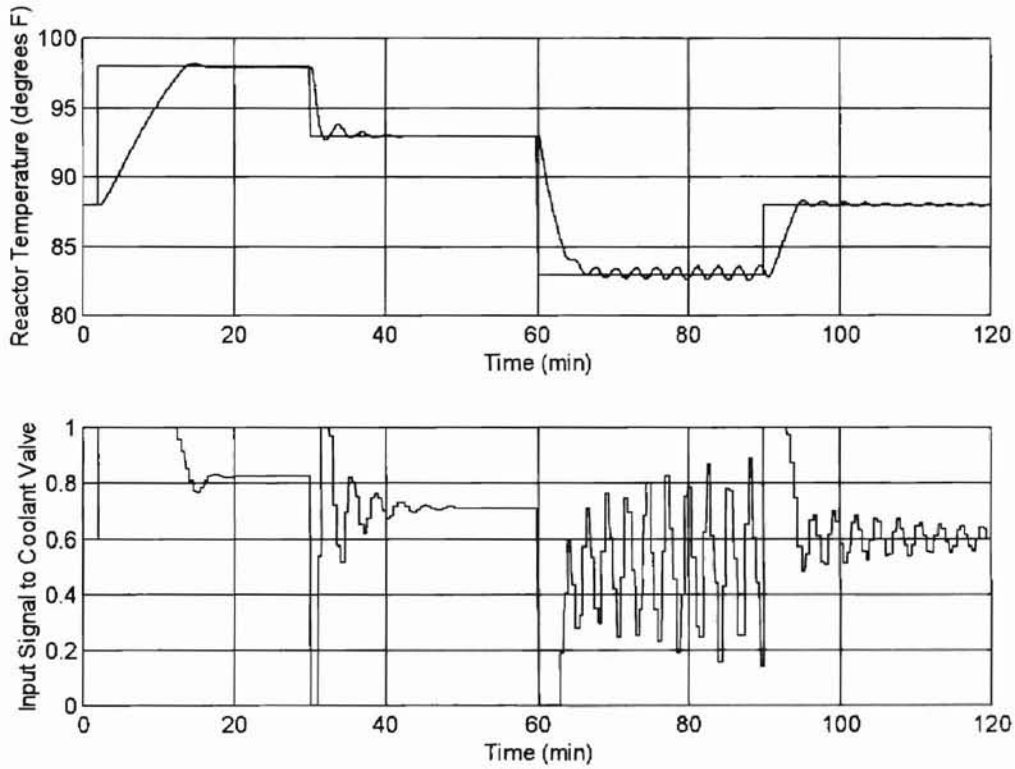

Figure 24. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model with a 50% Decrease of the Process Dead Time

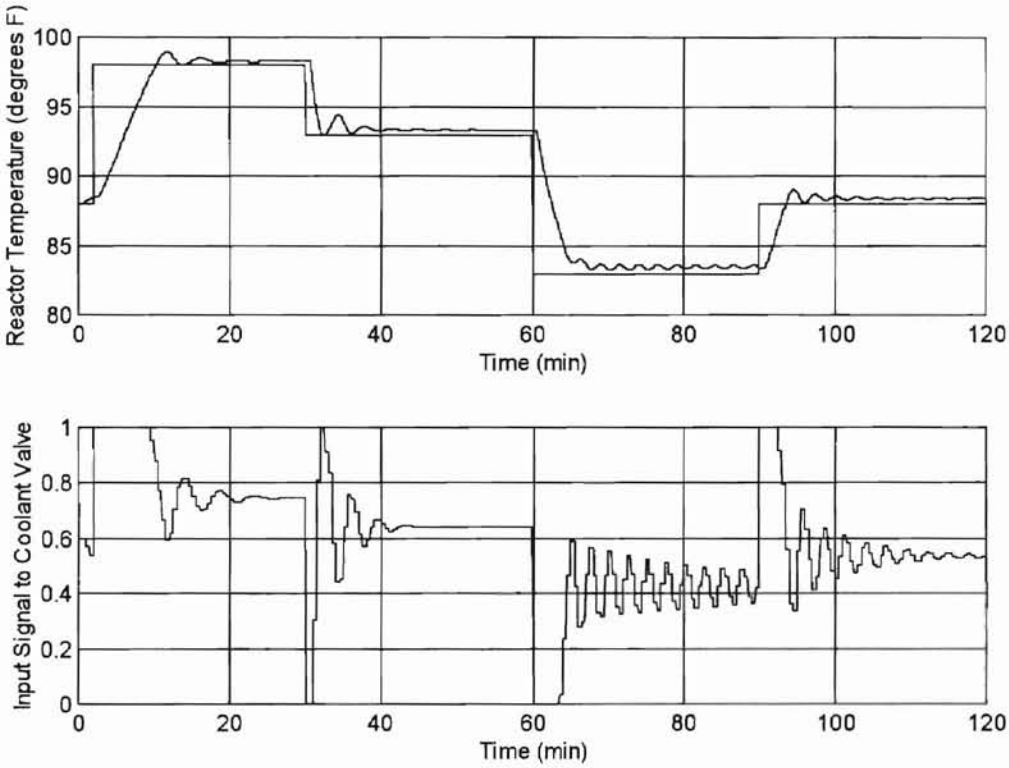

Figure 25. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model with a 10% Increase in the Heat of Reaction

controller is only marginally stable. Also, while there has been some model mismatch for all the setpoint changes so far, Figure 25 shows this mismatch much more profoundly.

Figure 26 shows the setpoint change series with the overall heat transfer coefficient of the cooling jacket decreased by 25%. The effects of changing the heat transfer coefficient were very similar to those obtained by changing the heat of reaction. As before, the controller shows an inability to fully compensate for model mismatch and remains marginally stable after the third setpoint change, where the process dynamics are the fastest.

IR Models

Two different IR models were also generated. Like the IS models, both had 11 inputs and 1 output with the only difference being the control horizons of two and three times the dead time. The series of setpoint change tests used to study the IR controllers performance were the same as those used to evaluate the IS controllers.

Figure 27 shows the sum-squared prediction error for the T(t+2) IR model versus the number of epochs trained. As before, training was halted when the sum-squared error of the test set began to increase. Note that the final sum-squared error of the ramp model was an order of magnitude less than that of the step model (0.074 *vs.* 0.62). Figure 28 shows the networks predicted input versus the actual input for the training set. While this figure does show some spiking behavior similar to that with the step-trained model, it almost always occurs immediately after the ramp levels out, whereas the spiking seen associated with step inputs is much more erratic and seen well into the level part of the steps. Figure 29 shows the networks predicted input versus the actual input for the test set. This test prediction has a sum-squared error less than that of the training set (0.061

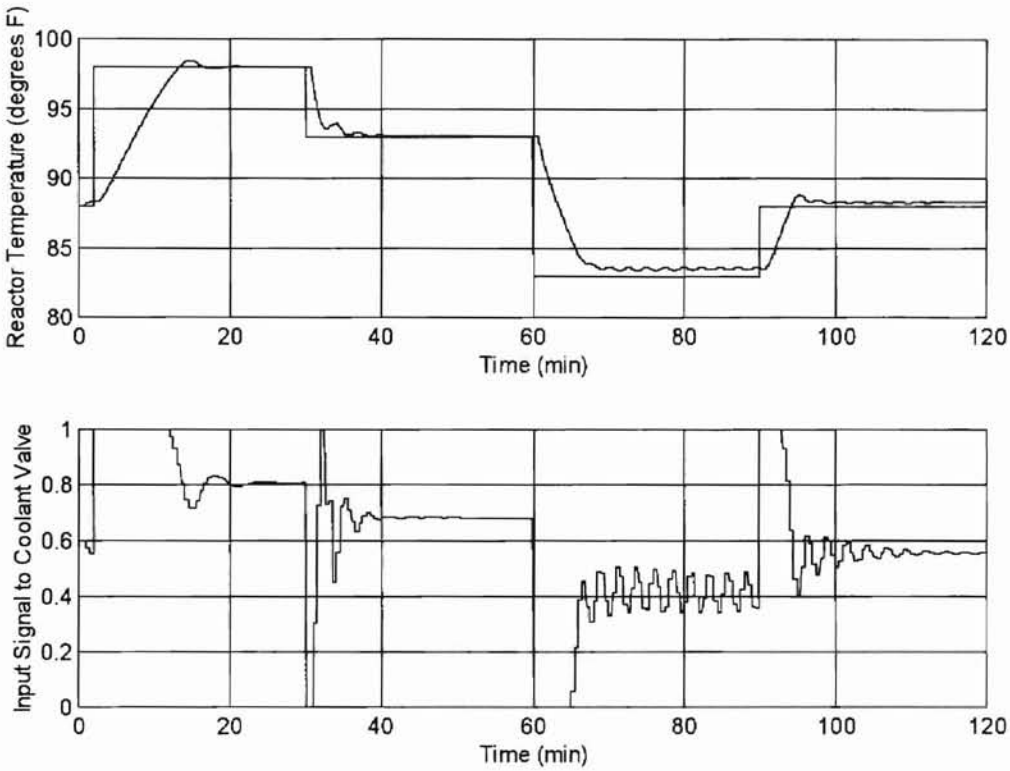

Figure 26. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model with a 25% Decrease in the Overall Heat Transfer Coefficient of the Cooling Jacket

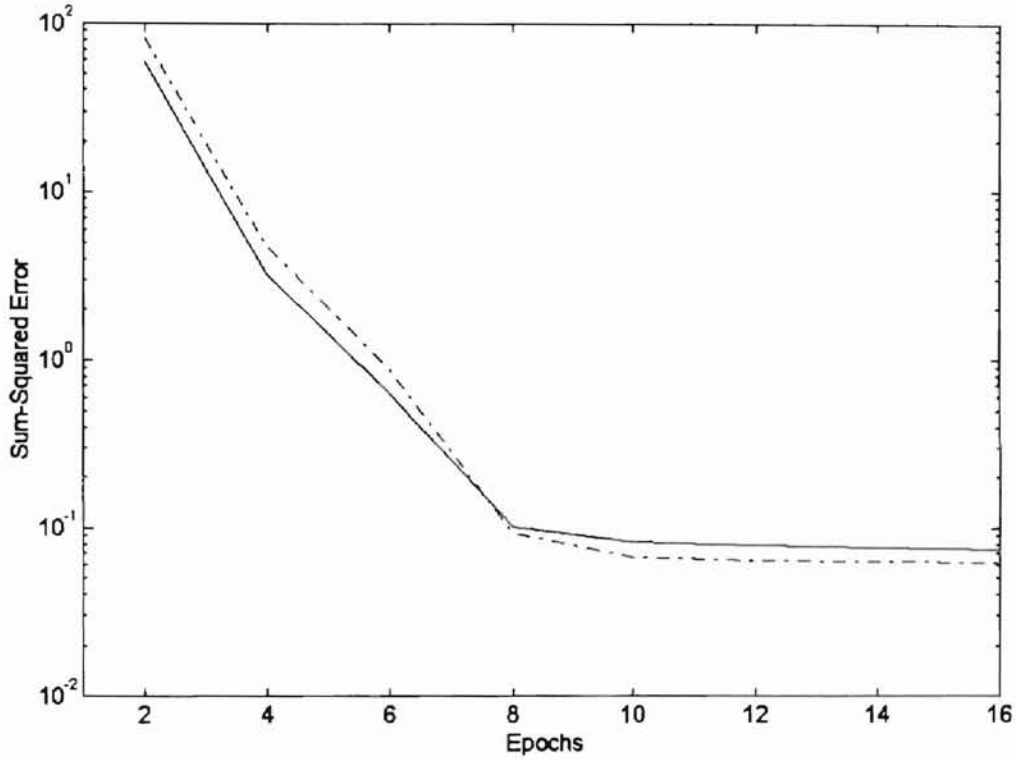

Figure 27. T(t+2) Ramp-Trained Inverse Process Model Sum-Squared Error as a Function of the Number of Epochs for the Training Set (—) and Test Set (— –)

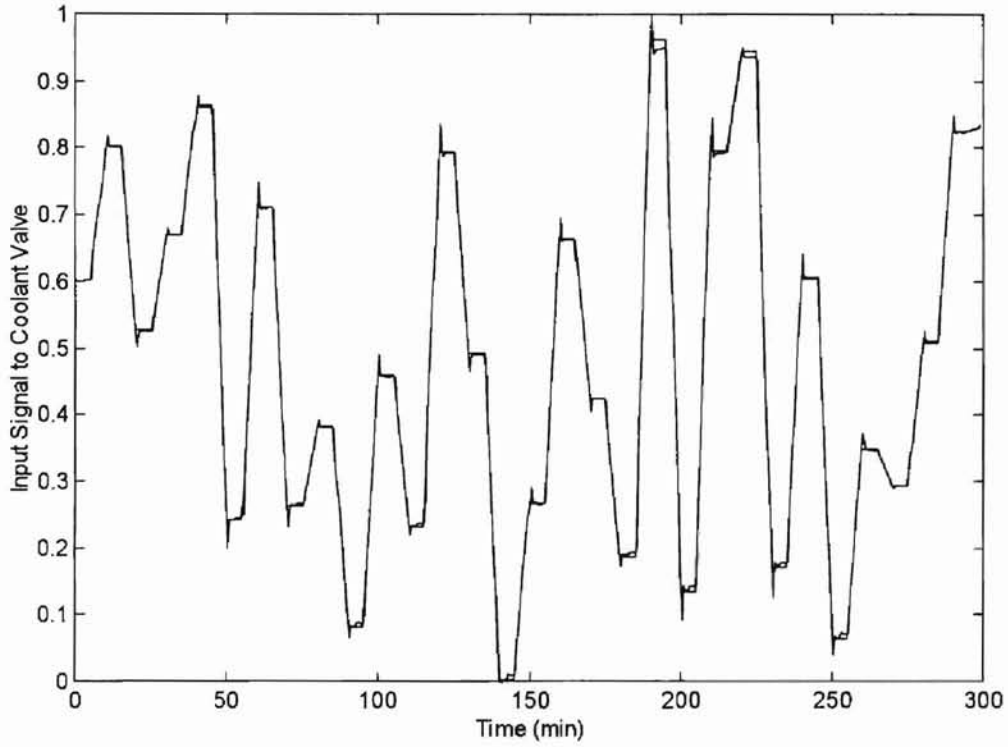

Figure 28. T(t+2) Ramp-Trained Inverse Process Model Training Set Prediction

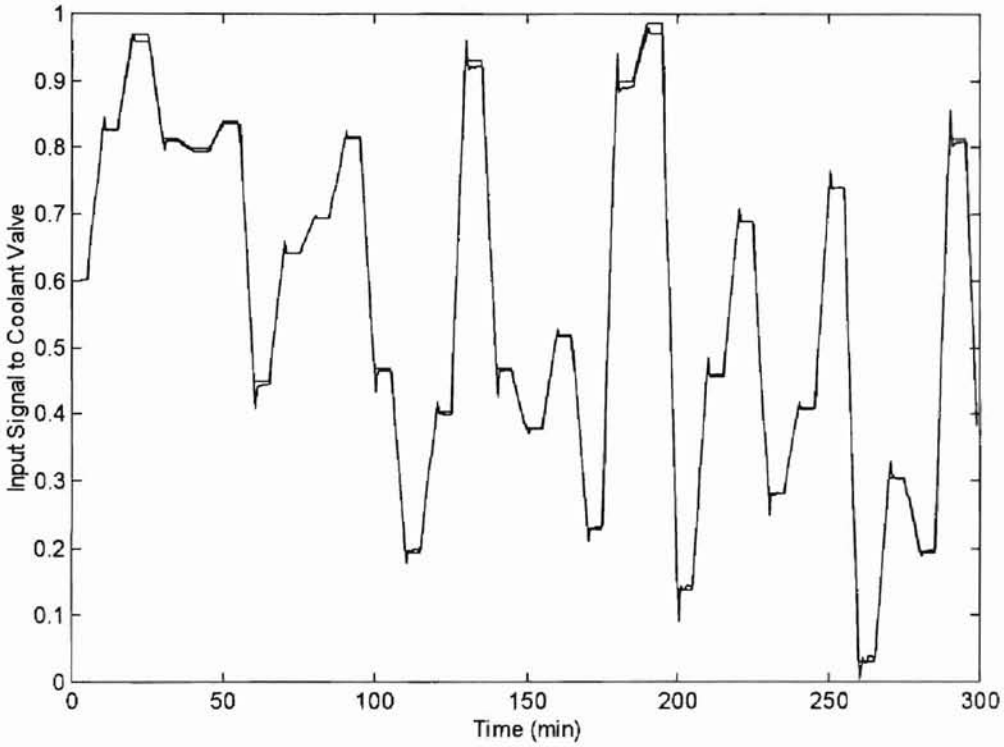

Figure 29. T(t+2) Ramp-Trained Inverse Process Model Test Set Prediction

vs. 0.074), whereas the sum-squared error of the test prediction for the step model was approximately the same as that of the training set (see Figure 15).

Figure 30 shows the results of a series of setpoint changes for the T(t+2) IR model. While the results presented in Figures 27 through 29 made this network appear to have much better control potential, it was actually unstable and not suitable for control. During the course of this research, numerous IS and IR models using a control horizon of T(t+2) had been previously obtained. Most of these were successfully implemented into the control scheme. The behavior exhibited in Figure 30 is most likely explained by assuming that the set of initial weights used to obtain the models was not favorable for training models that used a control horizon of T(t+2).

Figure 31 shows the sum-squared error for the T(t+3) IR model versus the number of epochs trained. Figure 32 shows the networks predicted input versus the actual input for the training set. Extending the control horizon for the IR models results in a prediction with virtually no spiking. As before, this improvement can be attributed to the networks ability to better learn the process dynamics as a result of the control horizon being moved farther out. Similar to the T(t+2) IR model, the sum-squared error of the T(t+3) IR model was an order of magnitude less than that for the IS model. Figure 33 shows the networks predicted input versus the actual input for the test set. From this figure, it is evident that this model has learned to generalize well using the ramp inputs. The accuracy of the prediction for the test set is almost identical to that of the training set.

Figure 34 shows the results of a series of setpoint changes for the T(t+3) IR model. This model is stable at all operating points and does not exhibit excessive sensitivity characteristics, especially at the lower operating temperatures, where the

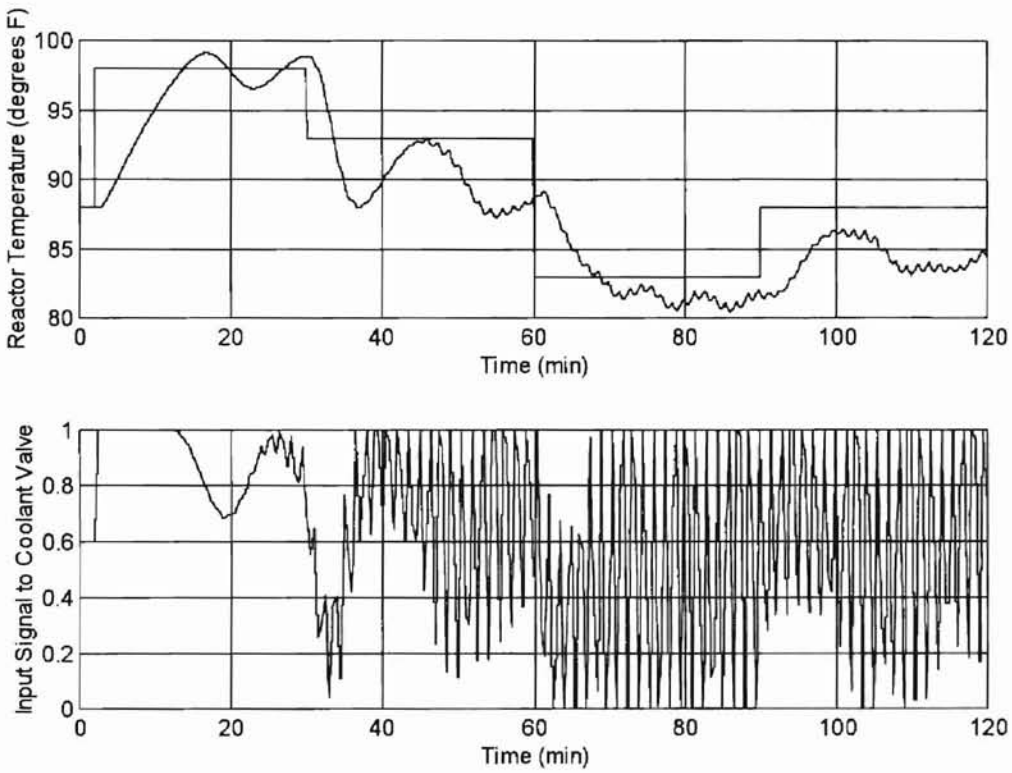

Figure 30. Setpoint Change Performance of the T(t+2) Ramp-Trained Inverse Process Model

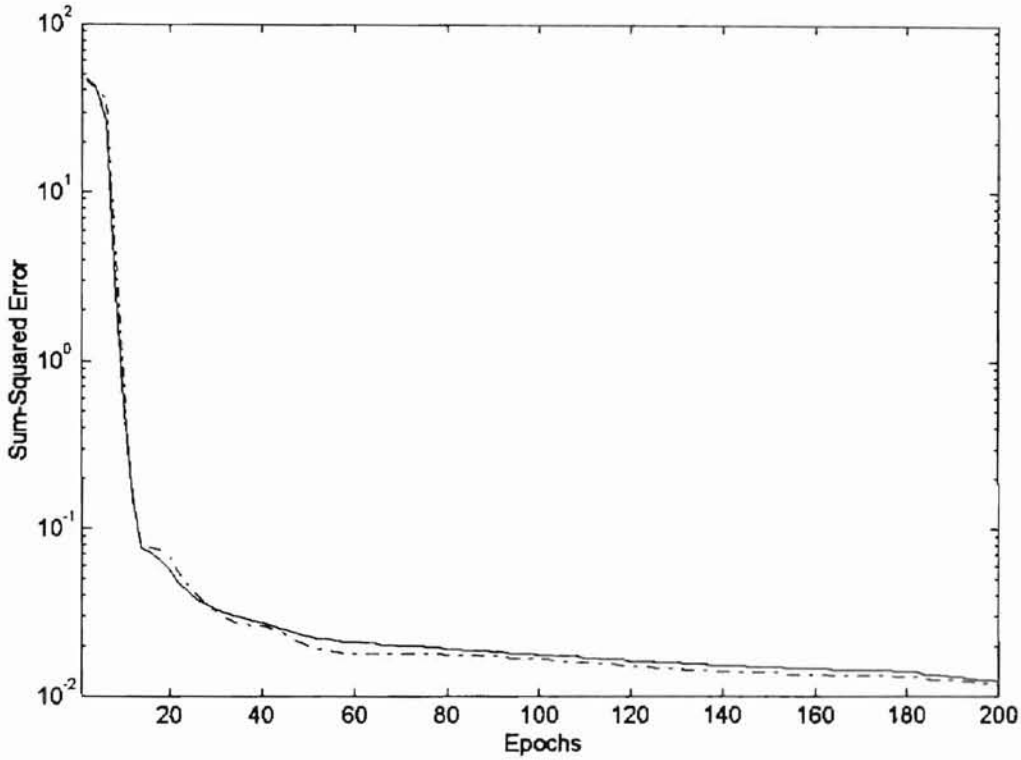

Figure 31. T(t+3) Ramp-Trained Inverse Process Model Sum-Squared Error as a Function of the Number of Epochs for the Training Set (—) and Test Set (— –)

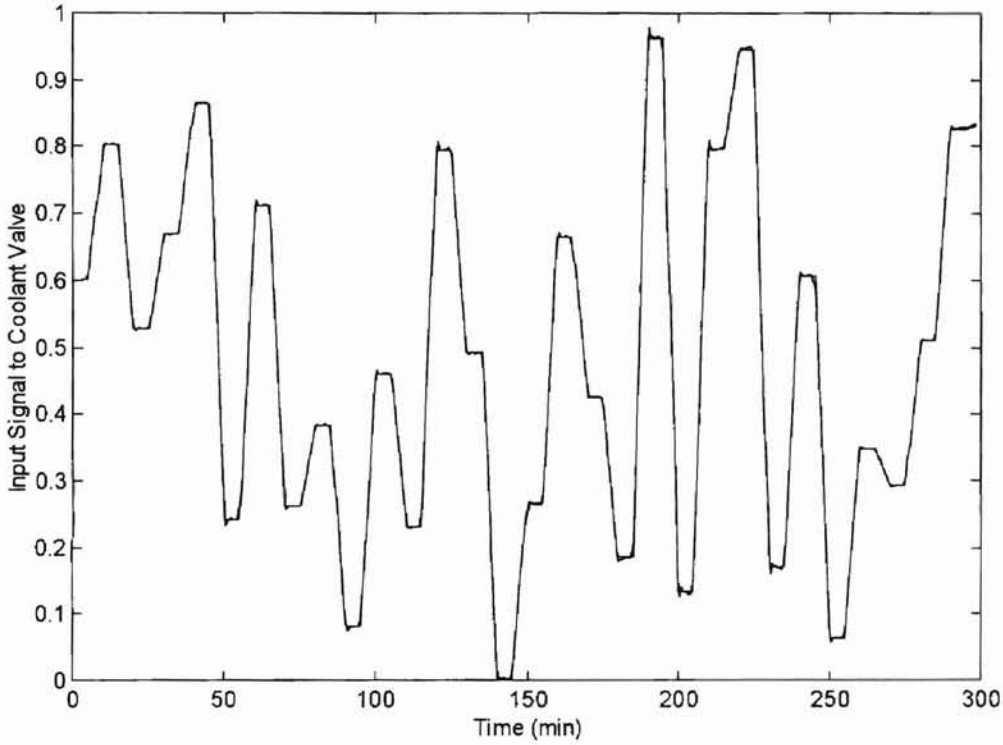

Figure 32. T(t+3) Ramp-Trained Inverse Process Model Training Set Prediction

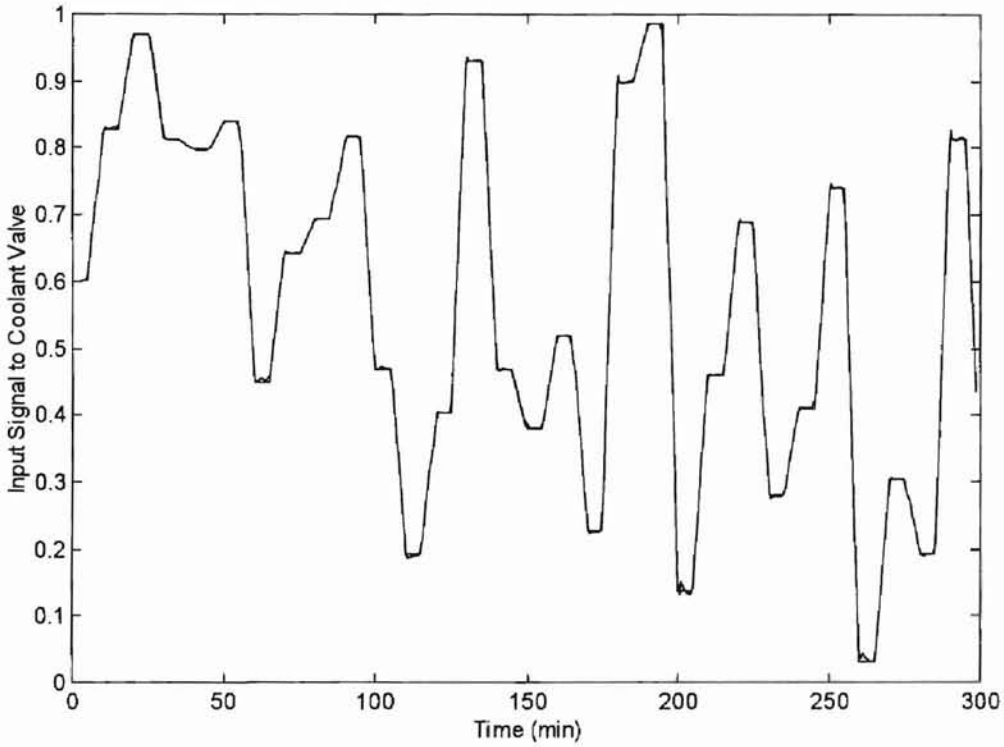

Figure 33. T(t+3) Ramp-Trained Inverse Process Model Test Set Prediction

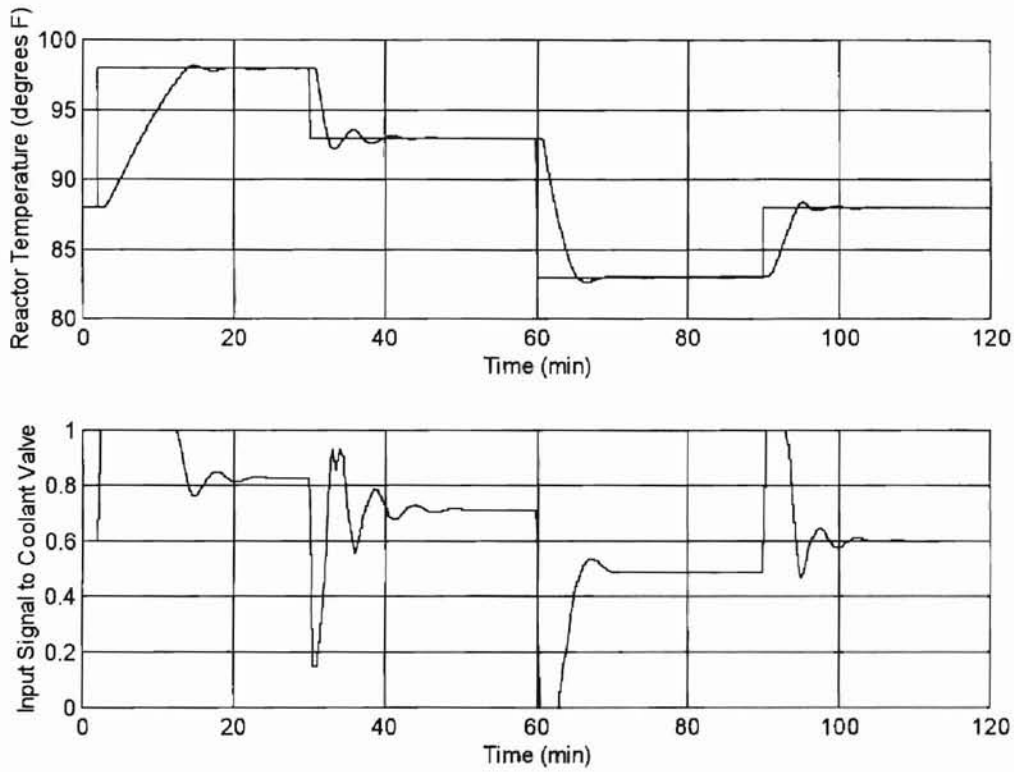

Figure 34. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model

process dynamics are faster. Figure 35 shows the same series of setpoint changes with a 50% increase in the process dead time. The settling time increased for each setpoint change. However, model sensitivity did not become a problem as it did with the step models. Figure 36 shows the setpoint change series with the process dead time decreased by 50%. The resulting control was superior to that of the base case. While this type of response makes sense - decrease the delay and overshoot and the settling time should correspondingly decrease - this was not the case when analyzing the IS model results. The IS model actually performed more poorly than both the base case and the case where the process dead time had been increased by 50%. This is because the IS model is too sensitive and becomes too aggressive, resulting in marginal stability at lower operating temperatures.

Figure 37 shows the results of the setpoint change series with the heat of reaction increased by 10%. This figure shows an almost negligible change in the performance of the controller. Only a slight increase in overshoot and offset for a few of the setpoint changes occurred. Furthermore, there was no evidence of any stability problems and the sensitivity of the model looked good. Finally, Figure 38 shows the setpoint change series for the case where the overall heat transfer coefficient of the cooling jacket has been decreased by 25%. Decreasing the heat transfer coefficient was similar to increasing the dead time of the process, since the dynamics were slowed down. Performance was very good, with only a slight increase in the rise times distinguishing these results from those of the base case. Again, model sensitivity and stability were good with no apparent problems.

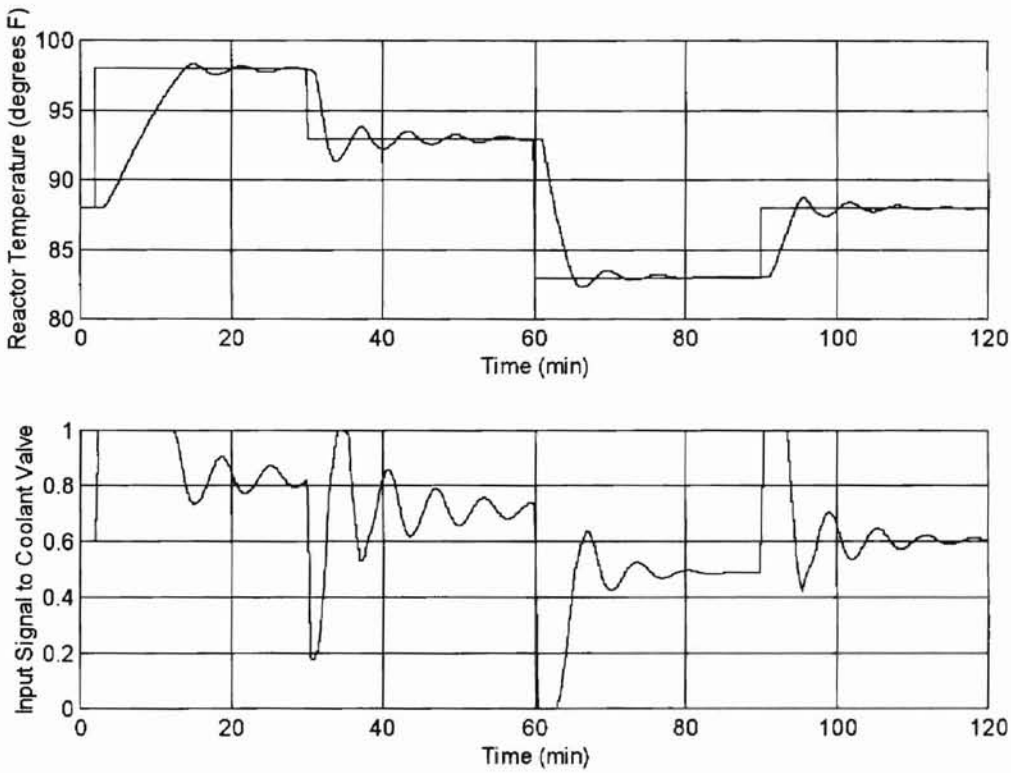

Figure 35. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model with a 50% Increase of the Process Dead Time

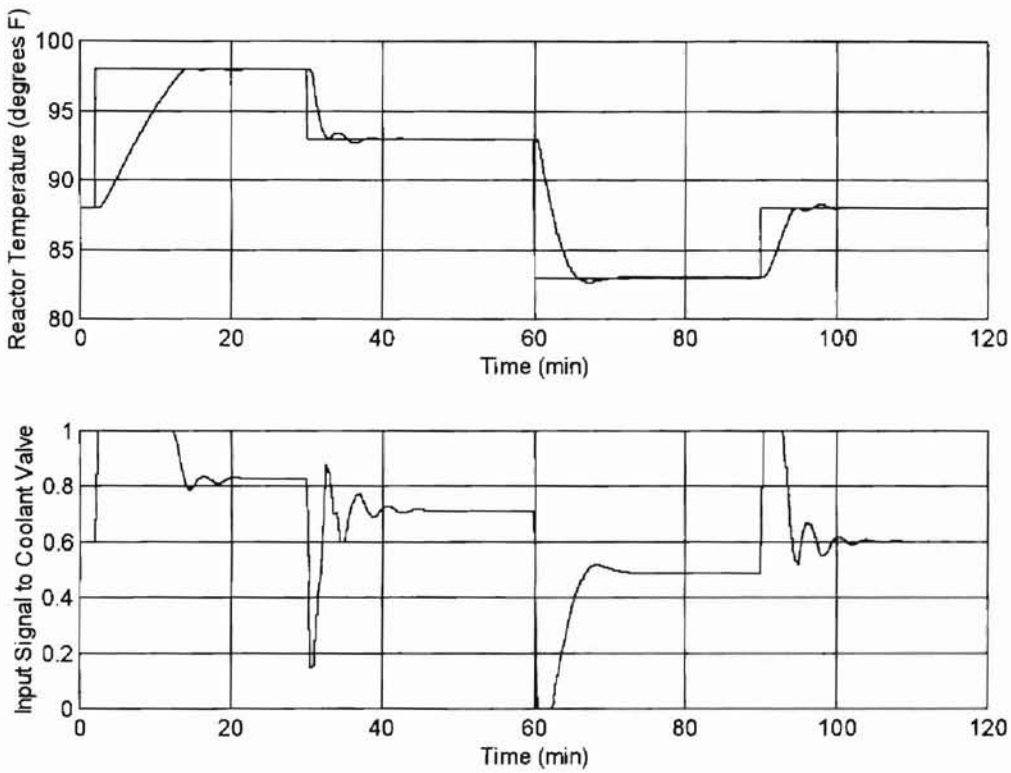

Figure 36. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model with a 50% Decrease of the Process Dead Time

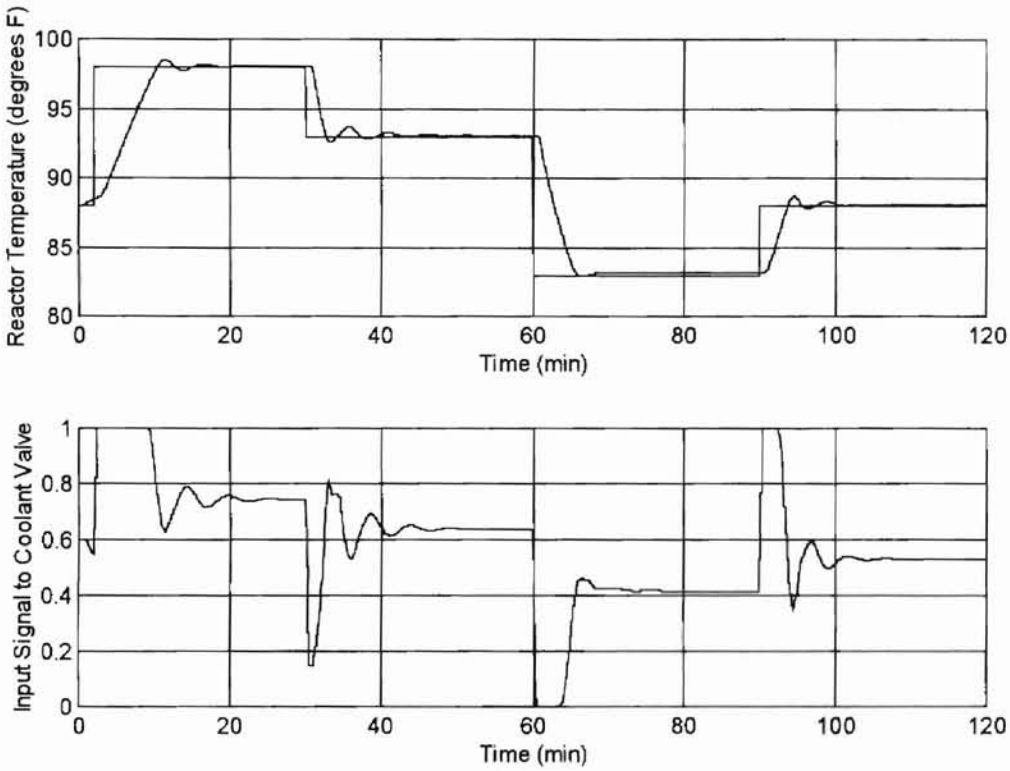

Figure 37. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model with a 10% Increase in the Heat of Reaction

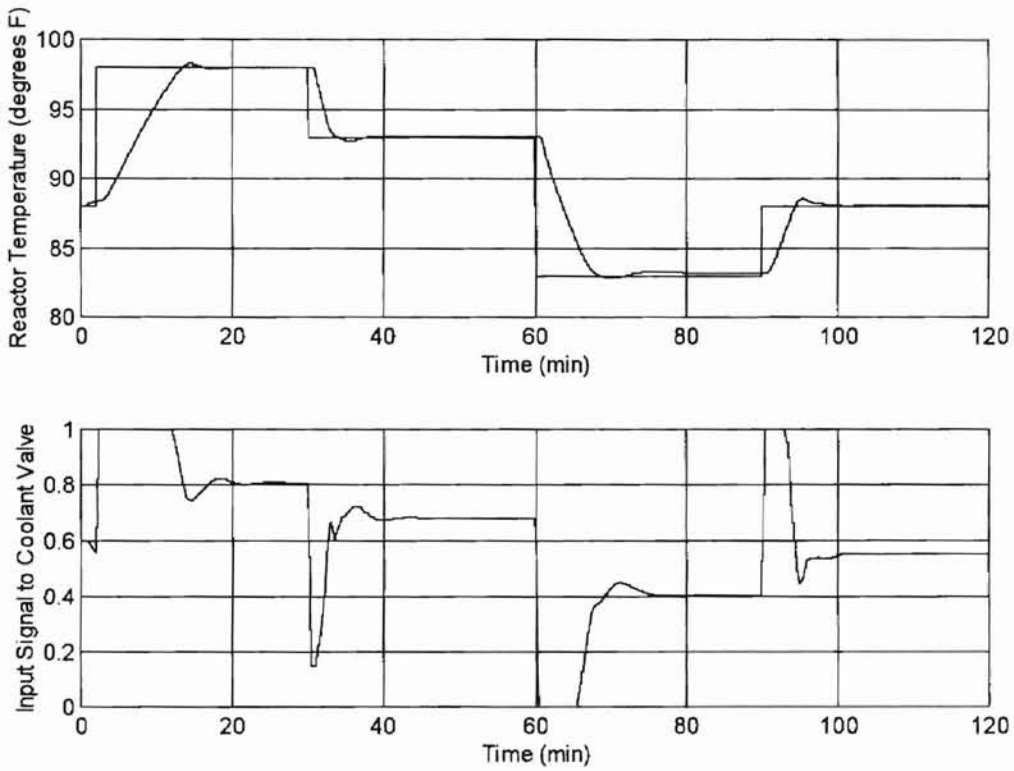

Figure 38. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model with a 25% Decrease in the Overall Heat Transfer Coefficient of the Cooling Jacket

EIR Models

As previously discussed, the EIR models were generated to demonstrate that better control performance is linked to model error. The two different EIR models were generated by training IR models until they had approximately the same integral absolute error as the corresponding IS models. Results should show that EIR models perform similar to IS models and that the IR models do provide better performance than both the IS and EIR models. Table I summarizes the integral absolute error of all of the models. Like the previous models, both EIR models had 11 inputs and 1 output, with the only difference being the control horizons of two and three times the dead time. The series of setpoint change tests used to study the EIR controllers performance were the same as those used to evaluate the IS and IR controllers. Figure 39 shows the sum-squared prediction error for the T(t+2) EIR model versus the number of epochs trained. One can see from the plot that network training was halted when the sum-squared error was still decreasing at a very high rate. This illustrates how the IR models are able to achieve much lower training errors in fewer epochs. Figure 40 shows the networks predicted input versus the actual input for the training set. While the prediction is not as accurate as that of the original ramp-trained inverse model, it is still much better at prediction than the step-trained inverse model with a similar integral absolute error.

Figure 41 shows the networks predicted input versus the actual input for the test set. This prediction spikes some, much like the prediction using the ramp-trained inverse model, with the main difference being the prediction offset at some points. The prediction is still much more accurate than the one provided by the IS model, since it not only has less spiking but is also not as erratic during the sustained part of each step. However, it

Table I. Summary of IS, IR, and EIR Integral Absolute Errors

	Integral Absolute Error
T(t+2) IS	4.7686
T(t+2) IR	1.5719
T(t+2) EIR	2.4774
T(t+3) IS	4.4139
T(t+3) IR	0.6491
T(t+3) EIR	1.4321

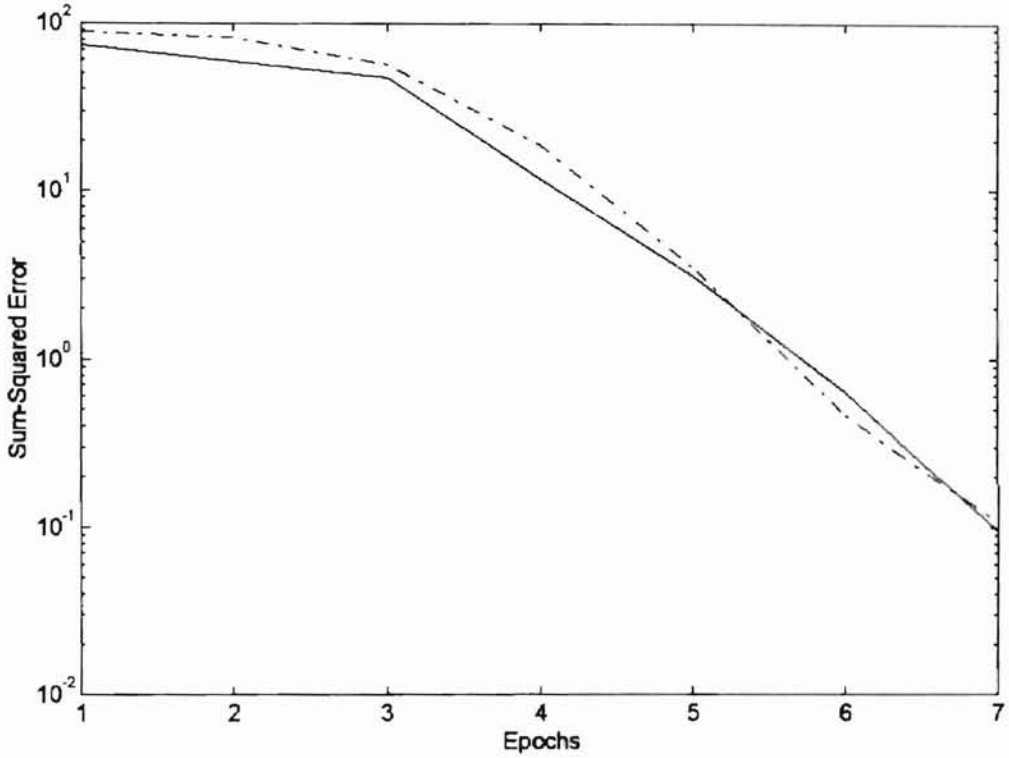

Figure 39. T(t+2) Equivalent Ramp-Trained Inverse Process Model Sum-Squared Error as a Function of the Number of Epochs for the Training Set (—) and Test Set (— –)

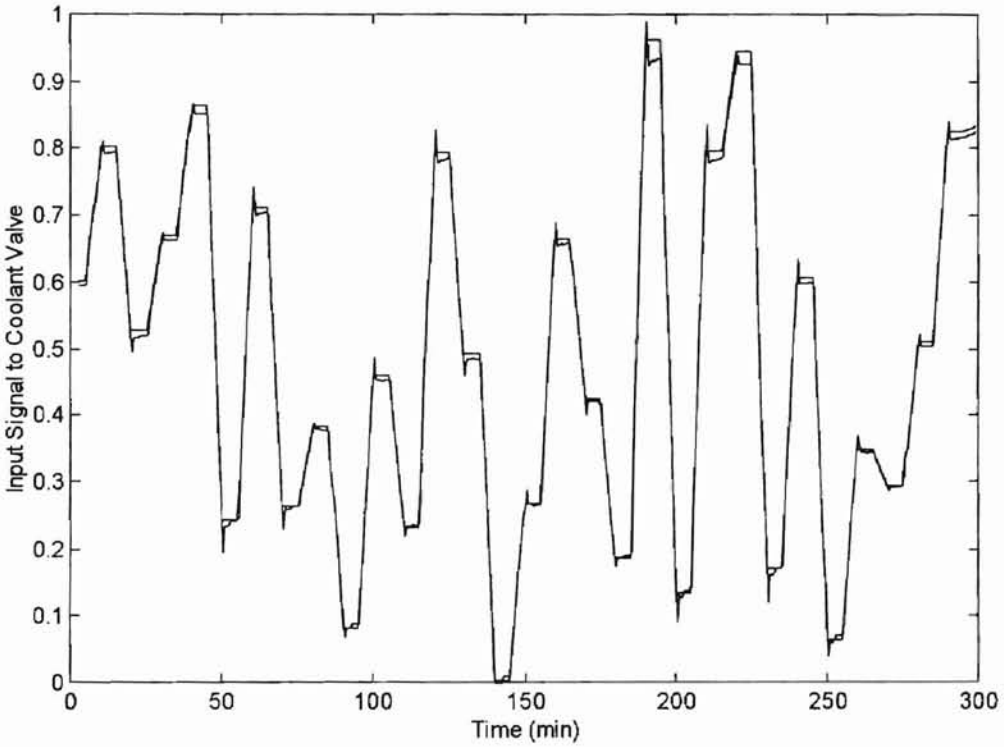

Figure 40. T(t+2) Equivalent Ramp-Trained Inverse Process Model Training Set Prediction

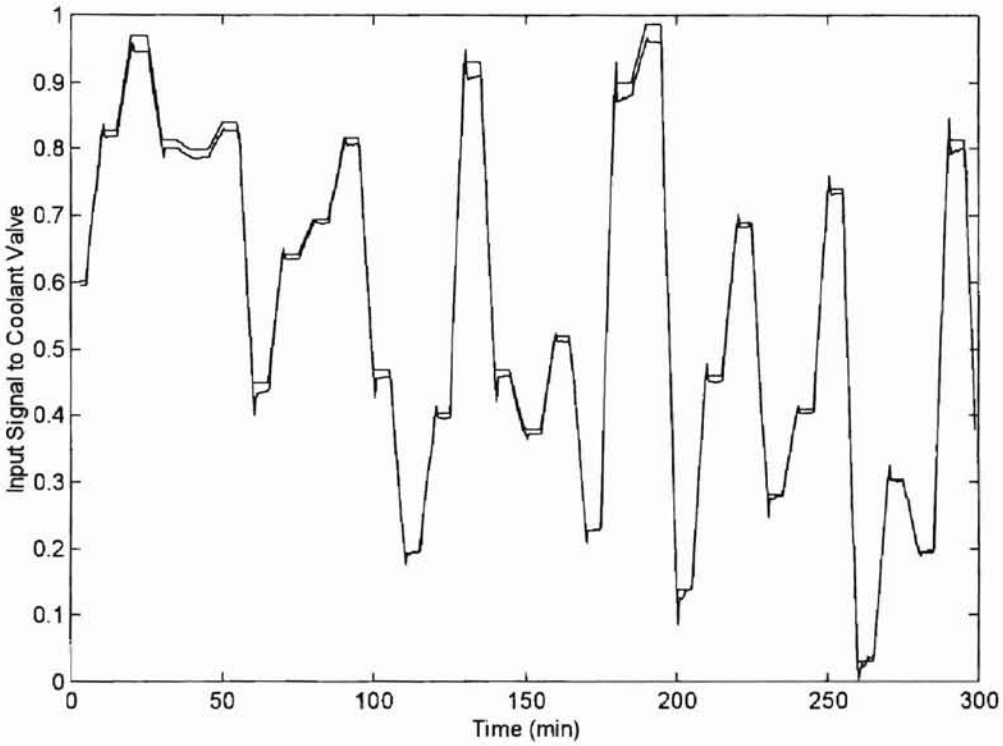

Figure 41. T(t+2) Equivalent Ramp-Trained Inverse Process Model Test Set Prediction

should again be noted that two different types of predictions with equivalent integral absolute errors should appear to have the same error, whereas two models might have very different sum-squared errors and still look very similar in their ability to provide accurate predictions.

Figure 42 shows the results of setpoint changes for the T(t+2) EIR model using the same series of setpoint changes that were used to evaluate the IS and IR models. The performance of the network was not as good as that of the IS model and was very similar to that of the IR model. Again, this behavior is attributed to bad initial weight selection. The network is too sensitive and results in unstable control, especially at the lower operating temperatures.

Figure 43 shows the sum-squared error for the T(t+3) EIR model versus the number of epochs trained. As was the case with the T(t+2) EIR model, training was halted while the error was still decreasing at a high rate in order to achieve an integral absolute error as close as possible to that of the IS model. Figure 44 shows the networks predicted input versus the actual input for the training set. Much like the IR model, this prediction has very little spiking in it and is very accurate. Also, when compared to the IS, the overall prediction is much better.

Figure 45 shows the networks predicted input versus the actual input for the test set. It can be seen from this figure that this model does an adequate job of generalizing. The most noticeable difference between the EIR and the IR models test set predictions is the amount of prediction offset. Figure 46 shows the results of the setpoint series for the T(t+3) EIR model. The controller exhibits some sensitivity problems at the lower

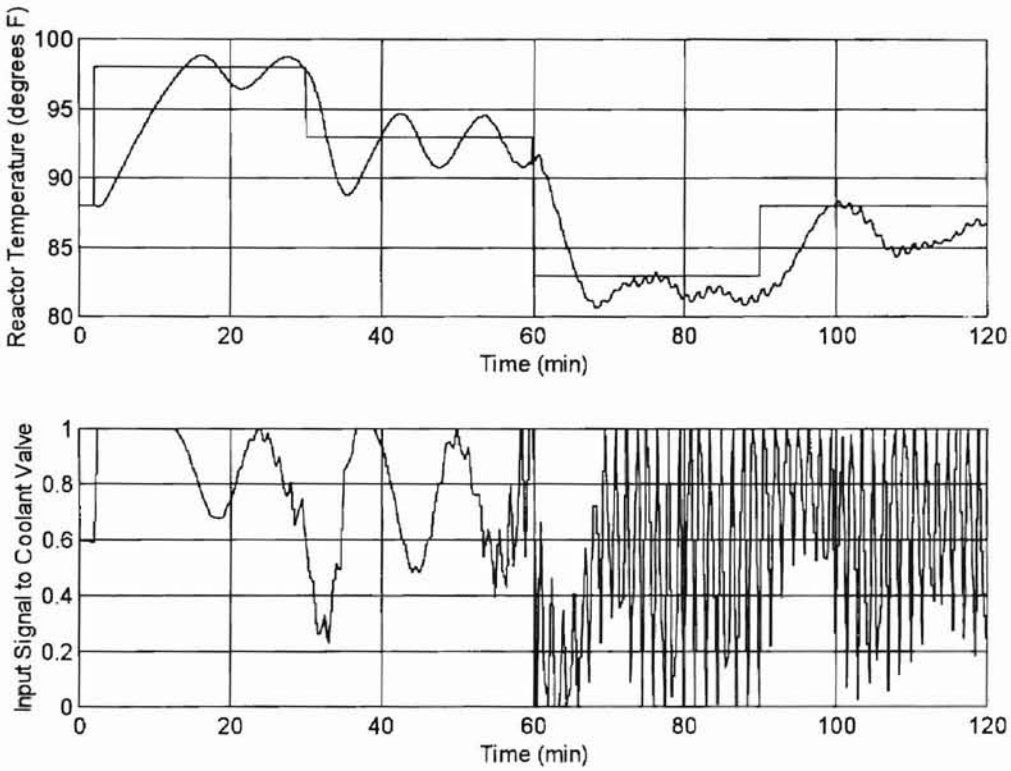

Figure 42. Setpoint Change Performance of the T(t+2) Equivalent Ramp-Trained Inverse Process Model

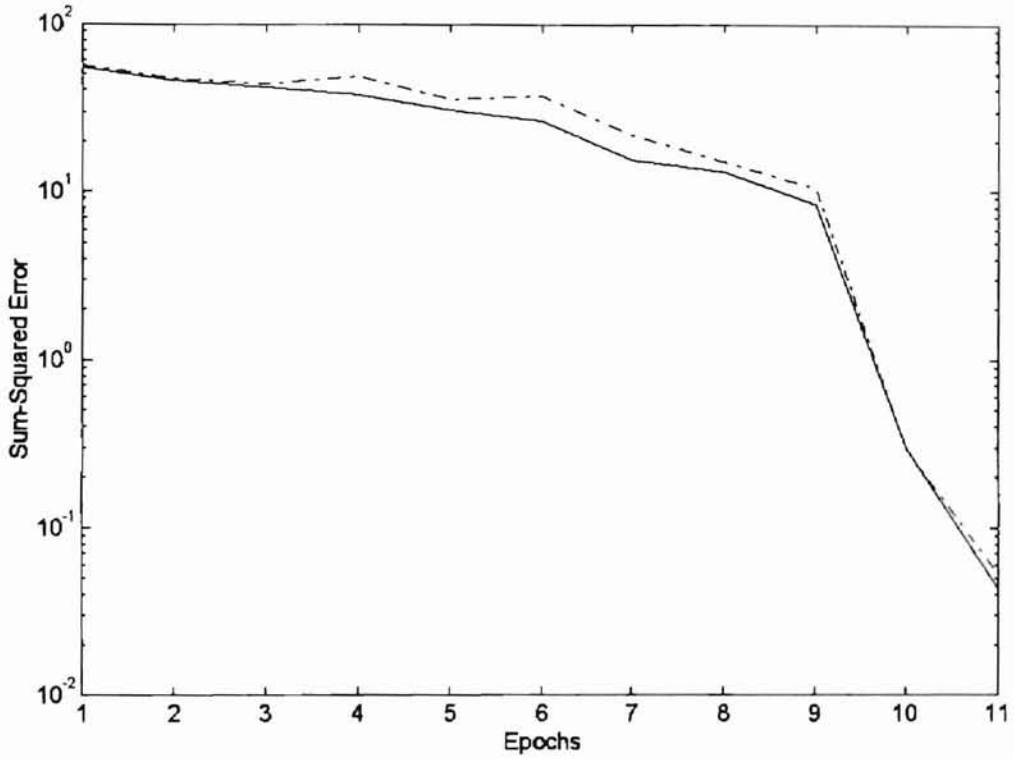

Figure 43. T(t+3) Equivalent Ramp-Trained Inverse Process Model Sum-Squared Error as a Function of the Number of Epochs for the Training Set (—) and Test Set (— –)

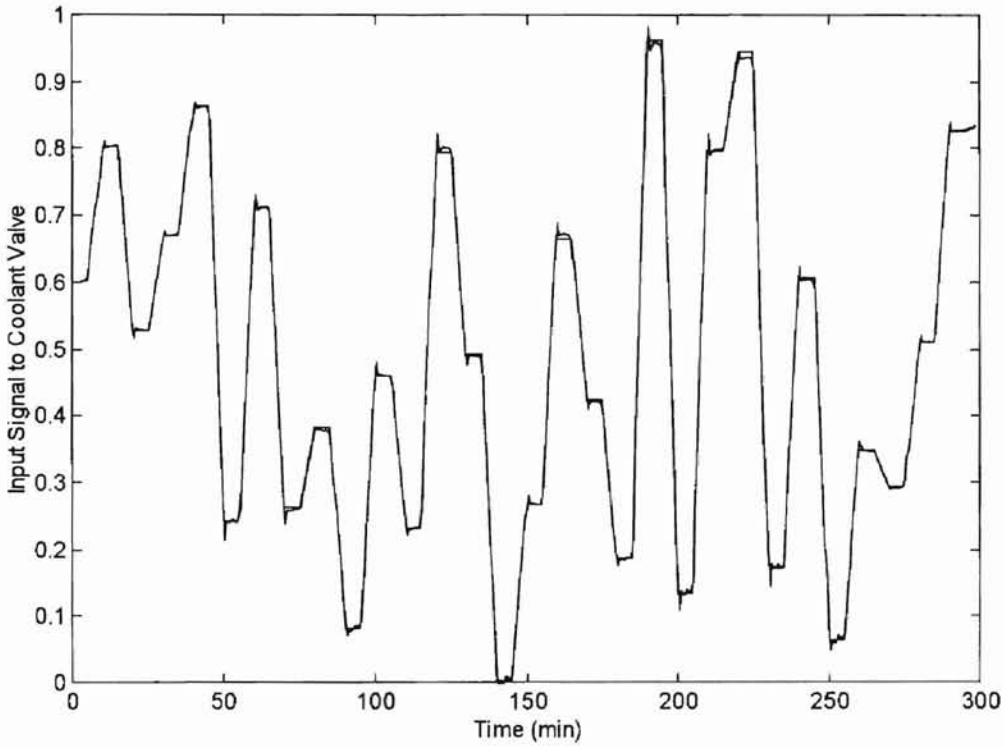

Figure 44. T(t+3) Equivalent Ramp-Trained Inverse Process Model Training Set Prediction

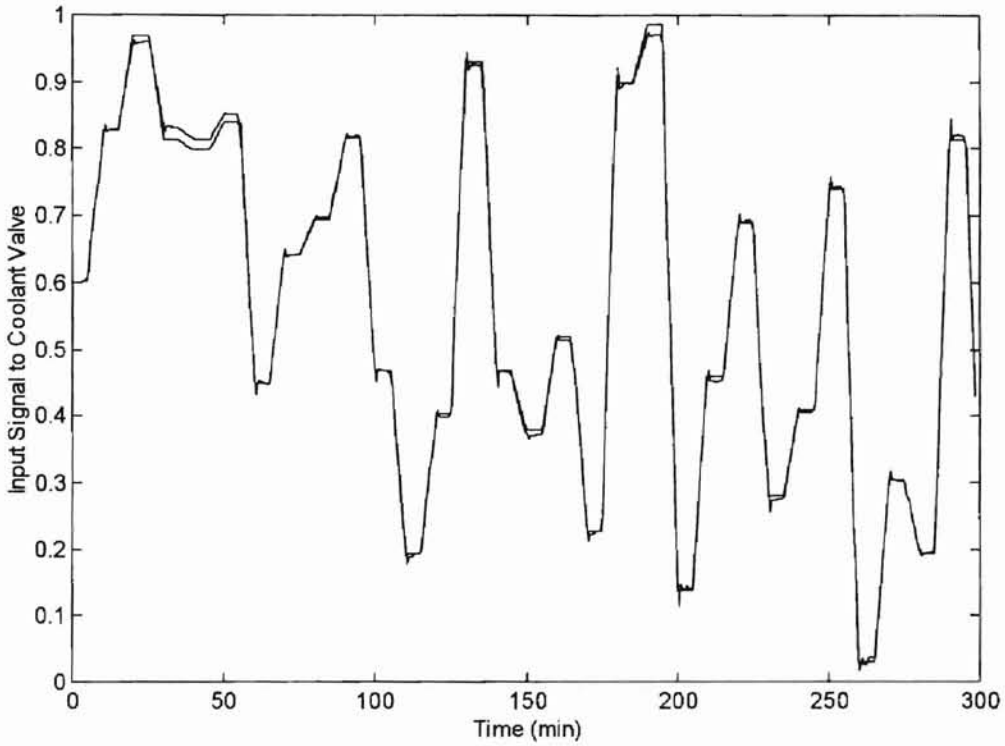

Figure 45. T(t+3) Equivalent Ramp-Trained Inverse Process Model Test Set Prediction

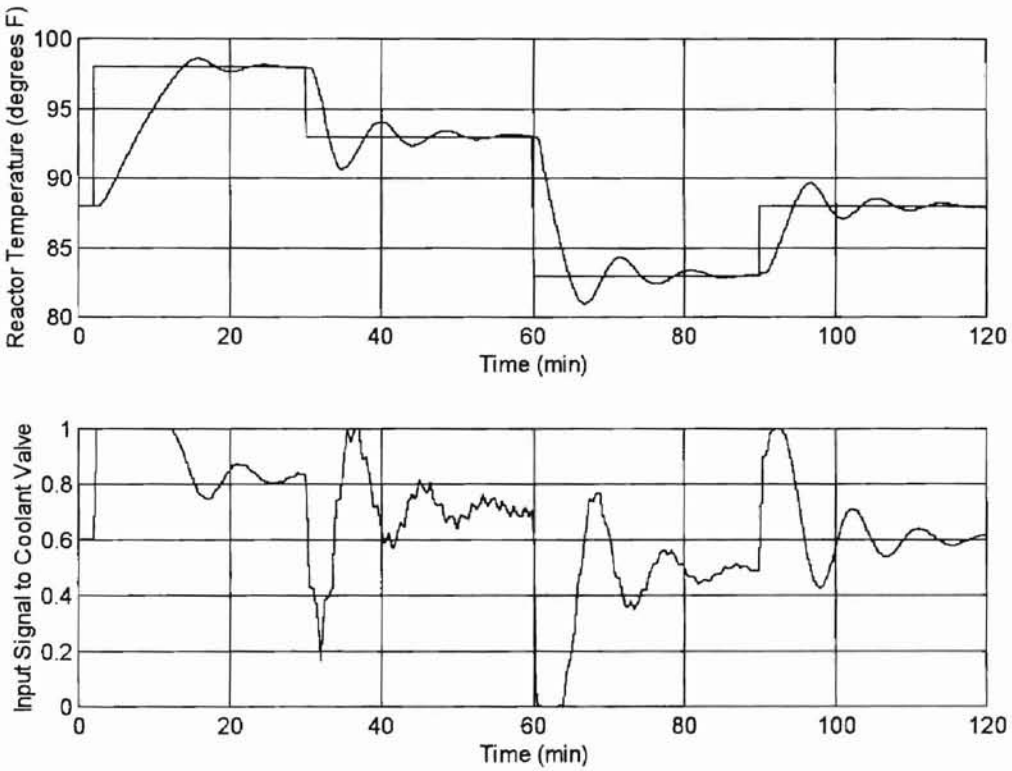

Figure 46. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model

operating temperatures and overshoots more with a larger settling time than both the IS and IR models.

Figure 47 shows the setpoint change series for the EIR model with the process dead time increased by 50%. As one would expect, the response is more sluggish resulting in increased overshoot and settling time. Also, because of the delay increase, the controller is effectively detuned and consequently does not appear to have any significant problems with model sensitivity. Figure 48 shows the series of setpoint changes with the process dead time decreased by 50%. The controller is able to decrease the overshoot and reduce the settling time for each setpoint change. However, as can be seen in the second setpoint change, the model is now too sensitive for the decreased dead time and results in an overaggressive controller at lower temperatures.

Figure 49 shows the results of setpoint changes with the heat of reaction increased by 10%. The controller performed similar to the base case, with a much more noticeable offset due to the increased model mismatch. Again, sensitivity problems were observed. Finally, Figure 50 shows the setpoint change series performance with the overall heat transfer coefficient of the cooling jacket decreased by 25%. Increased offset due to the added model mismatch is apparent, while the general aggressiveness of the controller is similar to that of the base case. However, one should note that the controller performed much better at the lower operating temperatures. This is because the decreased heat transfer coefficient helped the stability of the controller when the process dynamics were much faster. However, the controller exhibited sensitivity problems during the second step where the dynamics are slower.

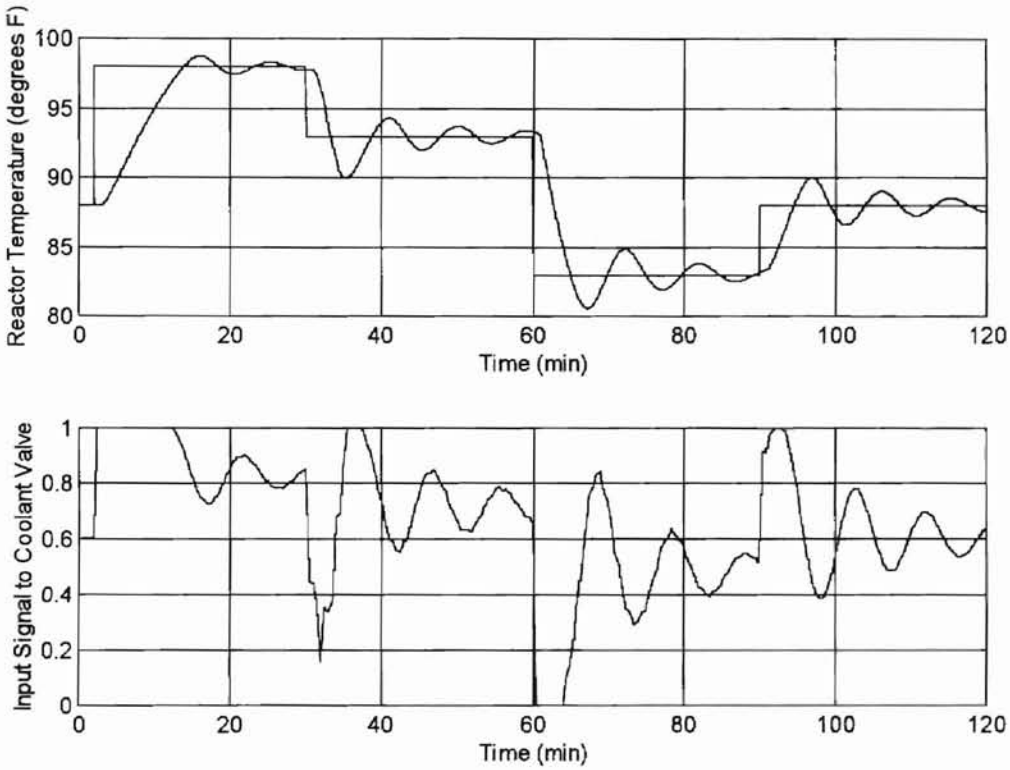

Figure 47. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model with a 50% Increase of the Process Dead Time

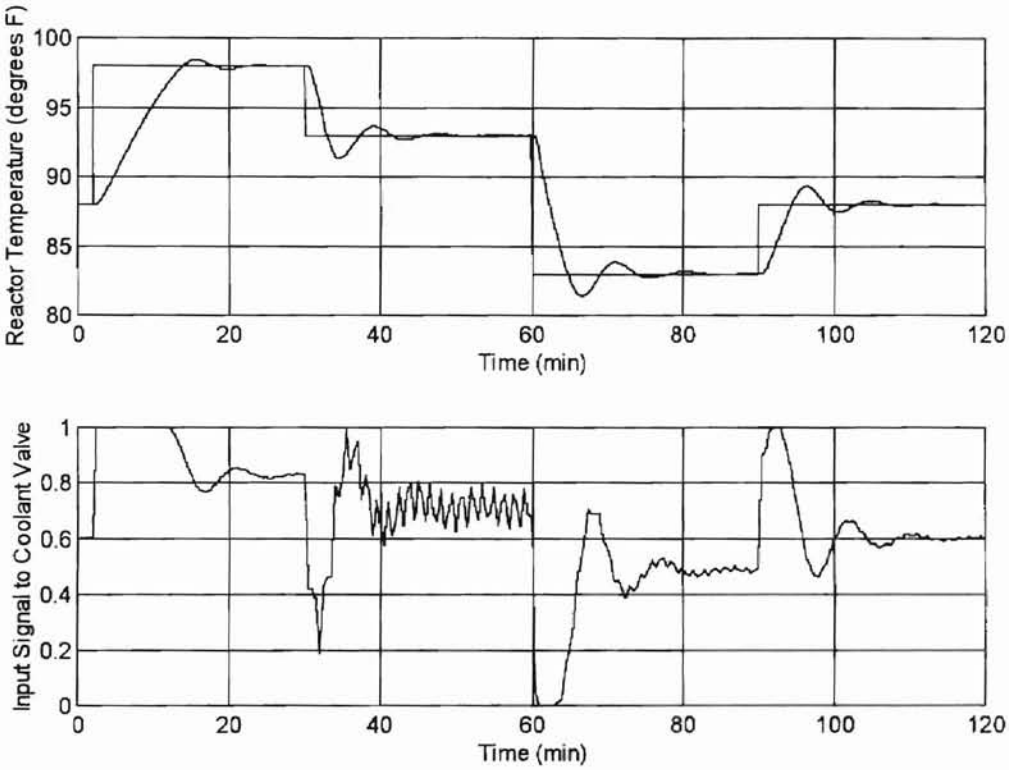

Figure 48. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model with a 50% Decrease of the Process Dead Time

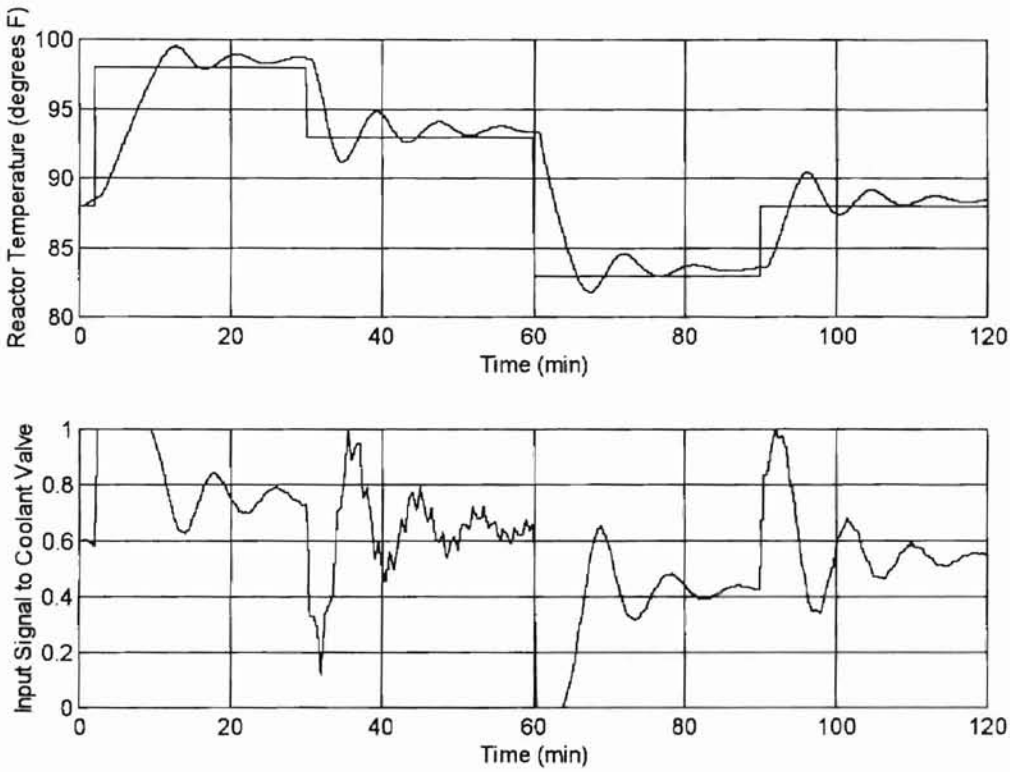

Figure 49. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model with a 10% Increase in the Heat of Reaction

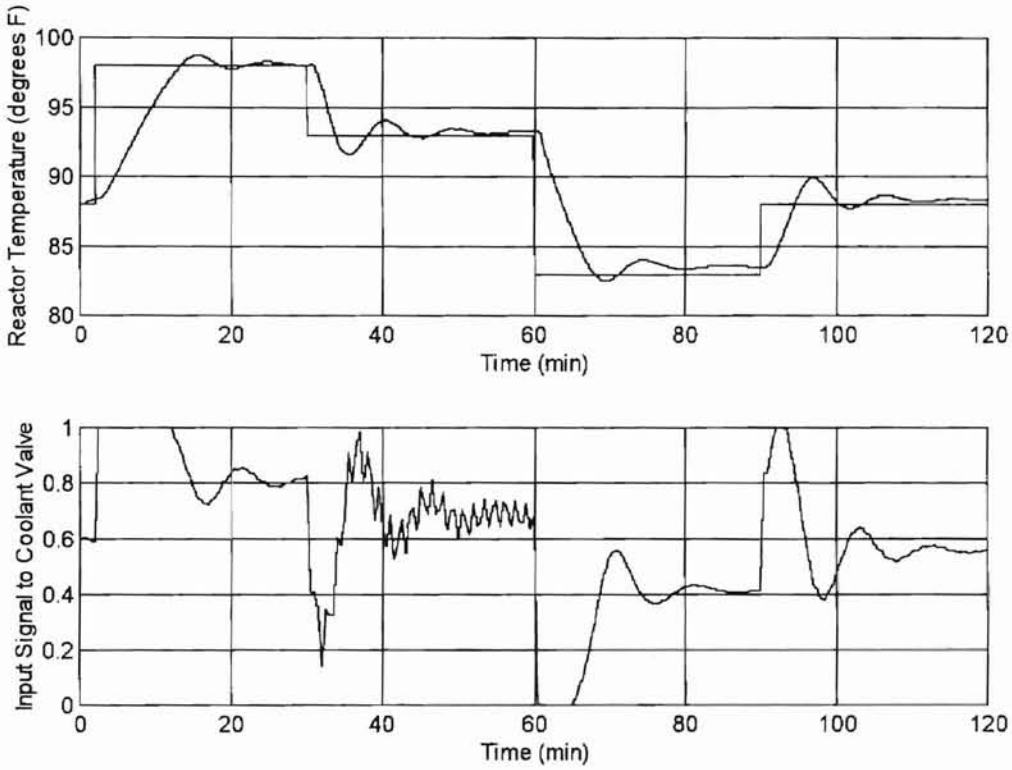

Figure 50. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model with a 25% Decrease in the Overall Heat Transfer Coefficient of the Cooling Jacket

Comparison of IS, IR, and EIR Results

Overall, both the T(t+2) and T(t+3) IS controllers exhibited some instability, were highly sensitive, and demonstrated poor performance in many instances. This could be seen in many of the setpoint changes in which the controller became unstable and never settled to the new setpoint, oscillating about it instead. It was more evident in the T(t+2) controller than the T(t+3) controller. This is because the model using the T(t+2) control horizon was closer to the dynamics of the process, resulting in much more aggressive control actions. Furthermore, it becomes more difficult to learn the process dynamics as the control horizon approaches the process dead time. Overall, the T(t+3) IS controller performed much better than the T(t+2) IS controller. The sensitivity of the IS models was calculated for comparison with the IR and EIR models. This was done by simulating a step change in the setpoint for each network. The resulting change in the network output was then divided into this setpoint change to obtain the sensitivity of the particular network. The sensitivity of the step models was much higher than that of both ramp-trained models. This was expected due to the nature of the function being approximated.

The T(t+2) IR and T(t+2) EIR controllers were not able to provide stable control. This unstable behavior for both of the T(t+2) ramp models and the T(t+2) step controller was assumed to be the result of bad initial weight selection. However, the T(t+3) IR and T(t+3) EIR controllers were able to virtually eliminate these problems and provide adequate control.

Overall, the T(t+3) IR controller exhibited desirable performance for all setpoint changes. The IR model also demonstrated a low sensitivity, with good performance even

in conditions of significant model mismatch. The T(t+3) IR controller did not oscillate about the new setpoints and did not exhibit any significant sensitivity problems. Compared to the T(t+3) IS controller, less overshoot and smaller settling times were observed. Finally, as was the case with the IS controllers, the T(t+3) controller was less aggressive and, overall, performed much better. However, one should be aware of the fact that the future goal can be moved forward or backward to an optimum number of sampling times for both positive and negative setpoint changes. In this sense, the control horizon is like a tuning parameter for setpoint changes. One should keep in mind, however, that any disturbance rejection capabilities will degrade the farther out the control horizon is moved due to the decrease in control action aggressiveness.

A comparison of the two types of controllers, IS and IR, operating in a closed-loop mode without model mismatch correction allows one to observe three things. First, the T(t+3) IR model appeared inherently stable whereas the T(t+3) IS model did not. Second, the IR is less sensitive due to the nature of the type of control action it has learned and is implementing. This results in less overshoot, smaller settling times, and better tracking about setpoints. Third, the ramp-trained models are able to achieve much lower errors than the step-trained models, which results in less offset due to model mismatch. The combined effect of these improvements allows the IR to provide overall better performance.

A summary of the IS and IR neural network models used in this study is provided in Table II. The table includes the number of epochs that the network trained, the sum-squared and integral absolute errors for the network, and the network sensitivity.

Table II. Summary of IS, IR, and EIR Neural Network Models Studied

	Epochs	Sum-Squared Error	Integral Absolute Error	Sensitivity
T(t+2) IS	100	0.6162	4.7686	64.7
T(t+2) IR	16	0.0739	1.5719	4.7
T(t+2) EIR	7	0.0961	2.4774	6.4
T(t+3) IS	46	0.7489	4.4139	22.4
T(t+3) IR	200	0.0123	0.6491	13.8
T(t+3) EIR	11	0.0439	1.4321	8.2

Sensitivity is in units of (% change of input signal to coolant valve per °F change in setpoint) at steady state conditions.

CHAPTER IV. APPLICATION OF INVERSE PROCESS MODELS FOR CONTROL

Integration of an Inverse Model into an IMC Control Strategy

Control Algorithm

The neural network control algorithm serves two purposes. First, it calculates a value for the future temperature as defined by Equation (2). Second, it takes process information and calculates the adjustment necessary to correct for model mismatch. More precisely, the algorithm takes the actual temperature that was achieved, goes back the corresponding number of sampling periods, calculates the control action that the neural network model would have specified to achieve that temperature, takes the difference between that value and the control input that was actually used, and then adds or subtracts this difference, accordingly, to the current control action. This will drive the controlled variable to the setpoint through the addition of the correction for model mismatch.

Tuning Parameters and Model Mismatch

There are two ways that one can go about tuning an inverse neural network model controller: modification of the control horizon and implementation of a filter. When implementing a classic IMC controller, a filter is always added to the inverse of the plant model after it has been made proper in order to avoid derivative control. This helps ensure stability. However, in the case of a neural network process model, whether it is a forward or an inverse one, addition of a filter is not as straightforward. One cannot look at a neural network model like one can the inverse of a transfer function and tell what order of filter, if any, is necessary to help ensure stability.

While the IS neural network models showed some instability, especially the one that utilized the shorter control horizon of two sampling periods, filtering them would still be less desirable than adjusting the control horizon. This is because the filter could have a more adverse effect on the controllers ability to reject disturbances and make setpoint changes than simply adjusting the control horizon. It is always preferable to manipulate the control horizon rather than filter the controller. For example, if one wanted to make a setpoint change large enough that it could not be physically achieved over a period of one control horizon, and one had employed a filter, the rise time would be longer than that of the same controller without the filter and any control horizon with a length that is smaller than or equal to the actual settling time. Therefore, the controller would have a less favorable response.

A discussion on the control horizon and the method used to manipulate it (defined by Equation (2)) can be found in the earlier section "Why Use an Inverse Process Model". Unlike implementing a filter, changing the control horizon does not affect a neural network models inherent anti-windup capability, and it allows the model to provide better performance through faster rise times, less overshoot, and smaller settling times.

One might ask how a controller that is using a modified control horizon could ever reach the actual setpoint if it is always moving only towards to the setpoint instead of to the setpoint. The answer is that because the controller corrects for model mismatch several moves after the correction was actually needed, the controller never really maintains the setpoint, but instead tracks closely around it. In other words, this correction delay makes the controller bypass the setpoint by overcompensating too much at the

current sampling period. While this correction scheme was not the most desirable one due to the delayed model mismatch correction that involved tracking of the setpoint rather than actually maintaining it, it should be understood that this control strategy was still valid for evaluation and comparison of the IS and IR neural network process models.

Closed-loop Performance with Model Mismatch Correction

Setpoint Changes

The IR models performed much better during setpoint changes than the IS models. This was expected, since the ramp-trained models exhibited better performance during closed-loop tests without model mismatch. Using IR models resulted in less overshoot, decreased sensitivity, and better performance at various operating points. While the addition of model mismatch correction capabilities provided offset-free control and allowed for disturbance rejection some additional tuning parameters were employed.

The tuning parameters can make the controllers more or less aggressive, whether through the use of a filter or through the modification of the control horizon. As stated before, it is always preferable to avoid using a filter unless absolutely necessary. It was observed that the controllers with models using a larger number of sampling periods for the horizon performed better than those using fewer. It was also observed that the controllers had to be tuned by adjusting the control horizon in order to achieve the best response, that which would minimize rise time, overshoot, and settling time.

Providing feedback to correct for model mismatch in order to eliminate offset after a setpoint change was effective but did have one unfavorable consequence. While there

was no steady offset, the resulting controller would track closely around the setpoint. As previously discussed, this behavior can be attributed to the control algorithm that was implemented. It occurred because the controller was not able to compensate for the model error until the actual value of the controlled variable could be obtained for comparison. Thus, the controller drove the controlled variable to the setpoint, where it eventually calculated that there was no model mismatch, which then allowed the controlled variable to wander from setpoint, after which it began correcting for model mismatch again, thus completing the cycle.

Comparison of IS, IR, and EIR Models for Setpoint Changes

The closed loop performance with model mismatch of the T(t+3) IS, IR, and EIR models was evaluated by activating the control algorithms model mismatch correction feature. Adding the model mismatch correction allowed the controller to reach steady state without offset and to reject disturbances. Preliminary tests using the T(t+2) IS, IR, and EIR models showed that they were not stable enough to enable mismatch correction. When enabled, the controllers were adversely affected. Before turning on the model mismatch, it was important to detune the controllers by adjusting the control horizon. Increasing the control horizon caused the controller to act more sluggish but did provide help in keeping the effects of model sensitivity, once the model mismatch correction was added, minimized.

The same setpoint change series used in the previous tests was again utilized. After some trial and error, it was determined that for these T(t+3) models, a control horizon scaling factor (CHSF) of 2 would be best. This effectively makes the models

behave as if they were trained with a control horizon twice that of the one they actually were. In this case, the effective horizon would be T(t+6).

Figure 51 shows the setpoint change performance of the T(t+3) IS model with a CHSF of 2. Comparing Figure 51 to Figure 22, one can see that the controllers response is more sluggish. Also, the sensitivity problem seen earlier after the third step has been alleviated. To gain a better understanding of how the control horizon adjustment affects controller performance with more model mismatch, the setpoint changes were again made with a 50% increase in the process dead time. Figure 52 shows the results of this test. The same phenomena are again observed: controller response is more sluggish and previous sensitivity problems have diminished. A comparison of Figure 52 and Figure 51 shows that the controller has handled the dead time increase very effectively, with only the slightest decrease in performance. Now that the effects of the control horizon adjustment were better understood, the model mismatch was turned on. Figure 53 shows the setpoint change performance of the T(t+3) IS model with both the horizon adjustment and model mismatch correction implemented. One can easily see that turning on the correction resulted in the controller becoming unstable. This behavior is attributed to the IS models high sensitivity, which becomes even more pronounced with the model mismatch enabled.

Figure 54 shows the setpoint change performance of the T(t+3) IR model with a CHSF of 2. As was the case with the IS model, the controllers response is now more sluggish. Figure 55 shows the results of the setpoint change series with a 50% increase in the process dead time. Comparing the controller responses of Figures 55 and 54, one sees that, much like the T(t+3) IS controller, the only noticeable effect is a slight decrease in

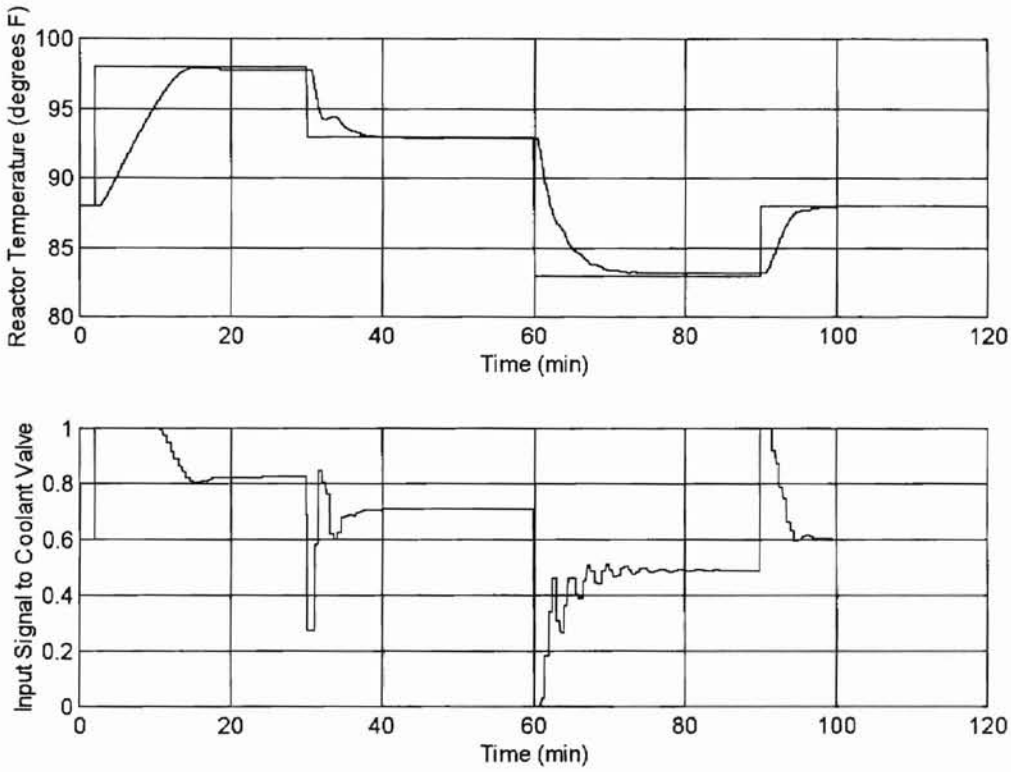

Figure 51. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model Using a CHSF of 2 without Model Mismatch Correction Enabled

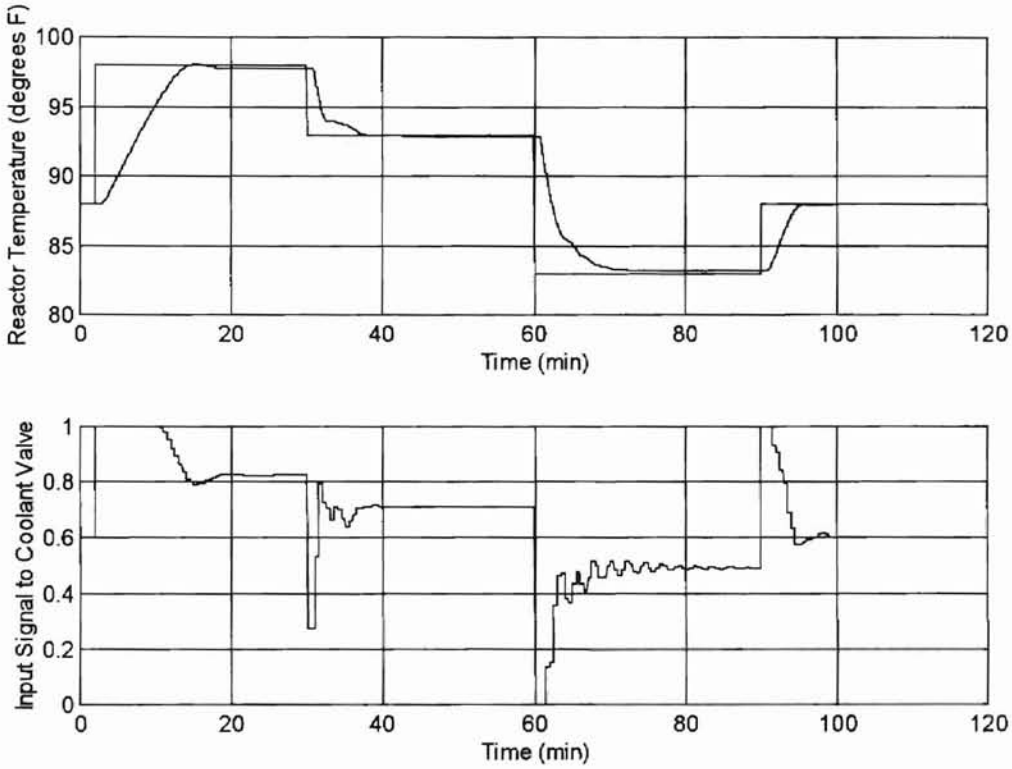

Figure 52. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model Using a CHSF of 2 with a 50% Increase of the Process Dead Time without Model Mismatch Correction Enabled

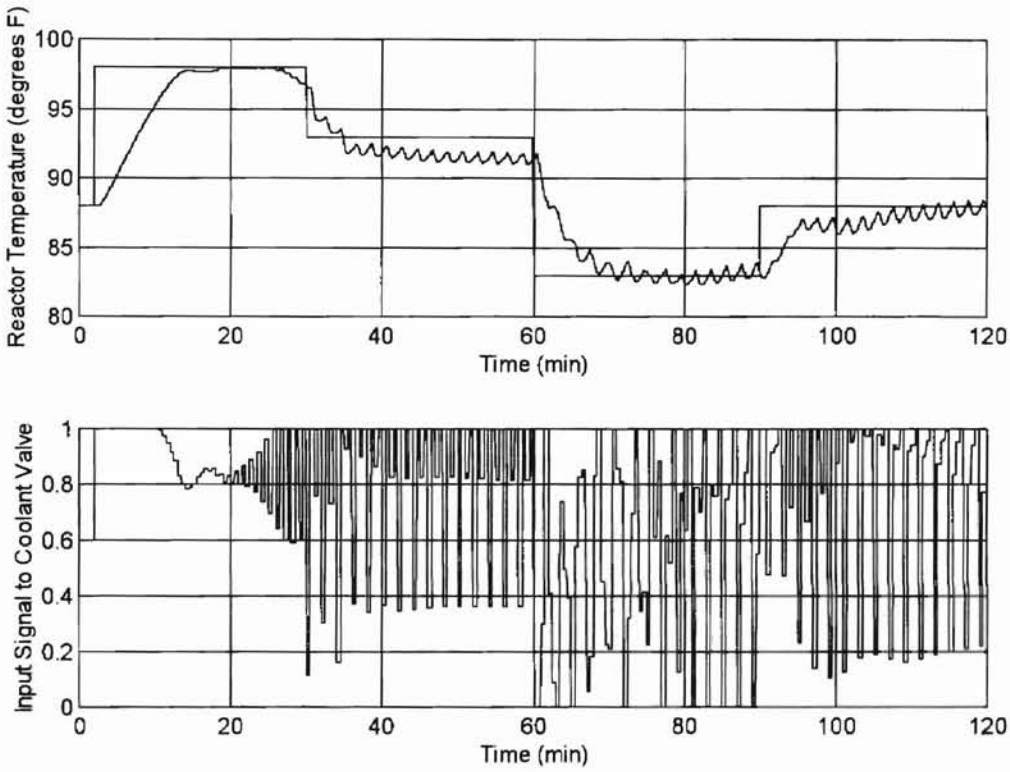

Figure 53. Setpoint Change Performance of the T(t+3) Step-Trained Inverse Process Model Using a CHSF of 2 with Model Mismatch Correction Enabled

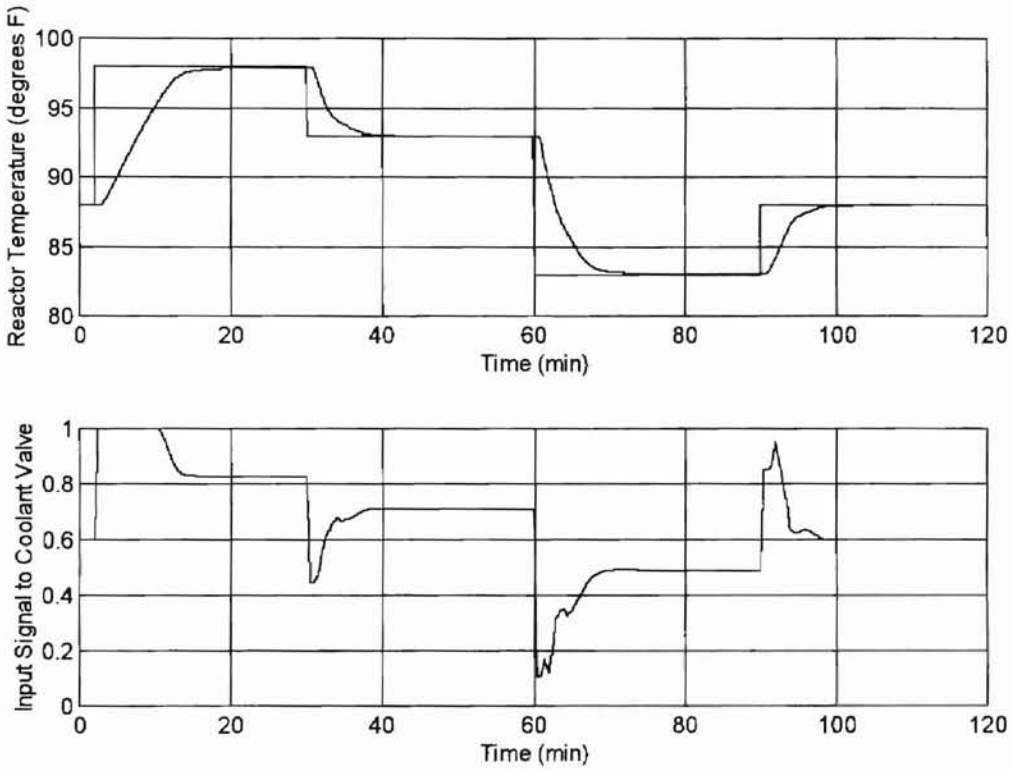

Figure 54. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model Using a CHSF of 2 without Model Mismatch Correction Enabled

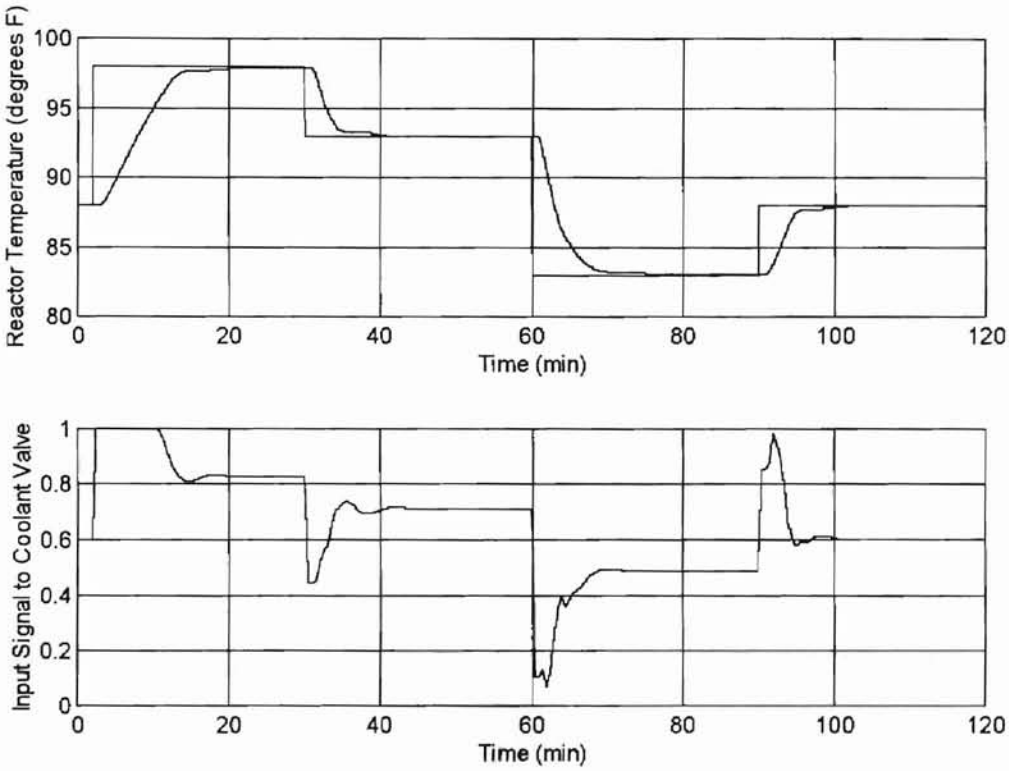

Figure 55. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model Using a CHSF of 2 with a 50% Increase of the Process Dead Time without Model Mismatch Correction Enabled

the controllers performance. Figure 56 shows the setpoint change performance of the T(t+3) IR, where both the control horizon adjustment and model mismatch correction were active. Unlike the T(t+3) IS model, the IR model has a much smaller sensitivity that allows it to provide offset-free control while still remaining stable. The model sensitivity effects can be seen in the last three setpoint changes. While there is some excessive valve movement, it should be noted that the IR model remains stable, unlike the T(t+3) IS model.

Figure 57 shows the setpoint change performance of the T(t+3) EIR model with a CHSF of 2. Comparing Figure 57 to Figure 46, one can observe that the controller provides a much more favorable response. In this case, the overshoot is minimal and the settling times are small, whereas before the overshoot was bigger for each setpoint change and the settling times were much longer. Figure 58 shows the setpoint change performance with the process dead time increased by 50%. As with the IS and IR controllers, the increase in dead time resulted in slightly more overshoot and longer settling times. Figure 59 shows the result of performing the setpoint change series with both the control horizon adjustment and model mismatch correction in use. Much like the IS controller, the EIR model is too sensitive for this mode of control and becomes very unstable.

Disturbance Rejection

The previously discussed closed-loop controller with model mismatch correction can also provide disturbance rejection capabilities. Because both model mismatch correction also imparts disturbance-rejection capability, one must also consider, when

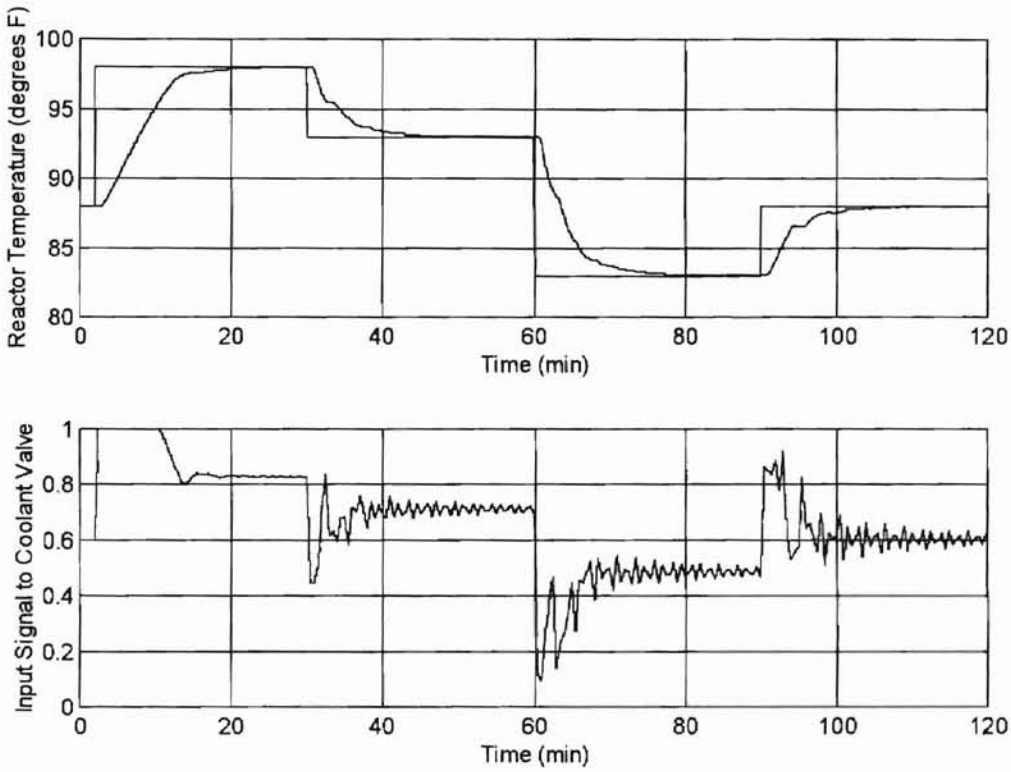

Figure 56. Setpoint Change Performance of the T(t+3) Ramp-Trained Inverse Process Model Using a CHSF of 2 with Model Mismatch Correction Enabled

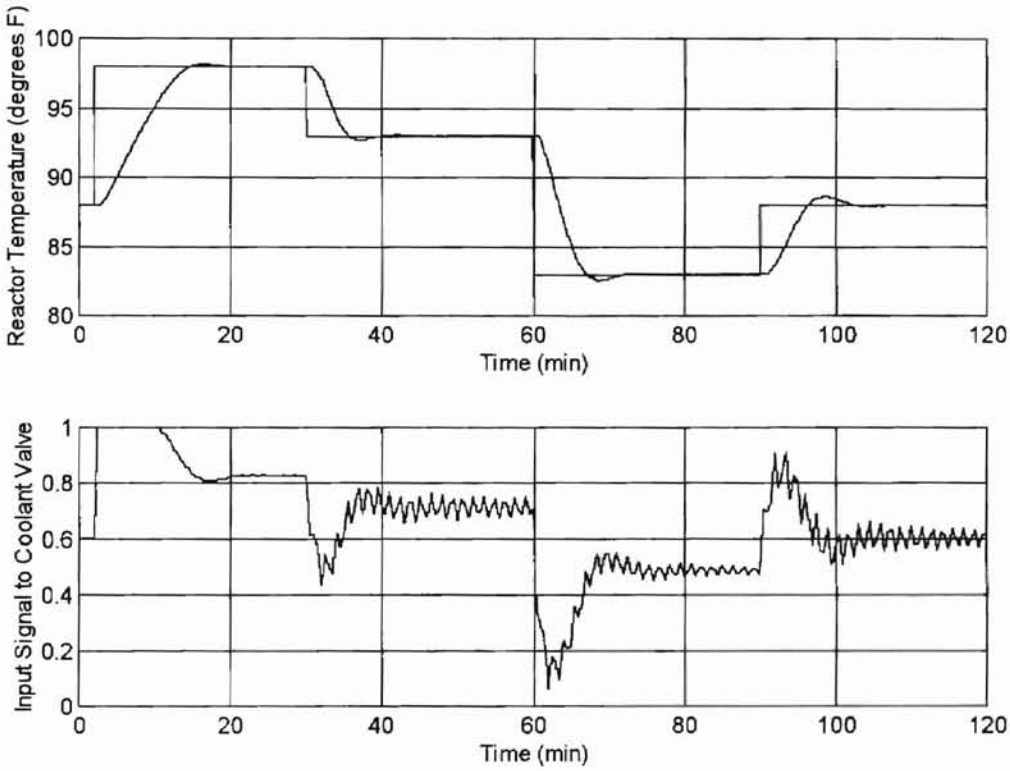

Figure 57. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model Using a CHSF of 2 without Model Mismatch Correction Enabled

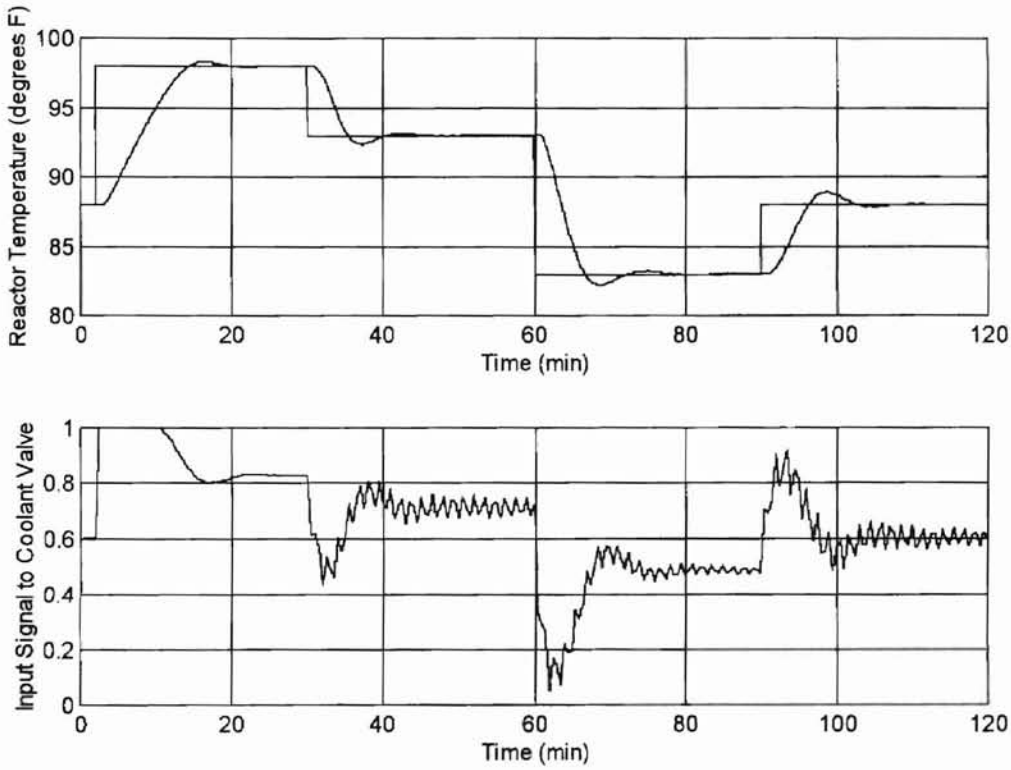

Figure 58. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model Using a CHSF of 2 with a 50% Increase of the Process Dead Time without Model Mismatch Correction Enabled

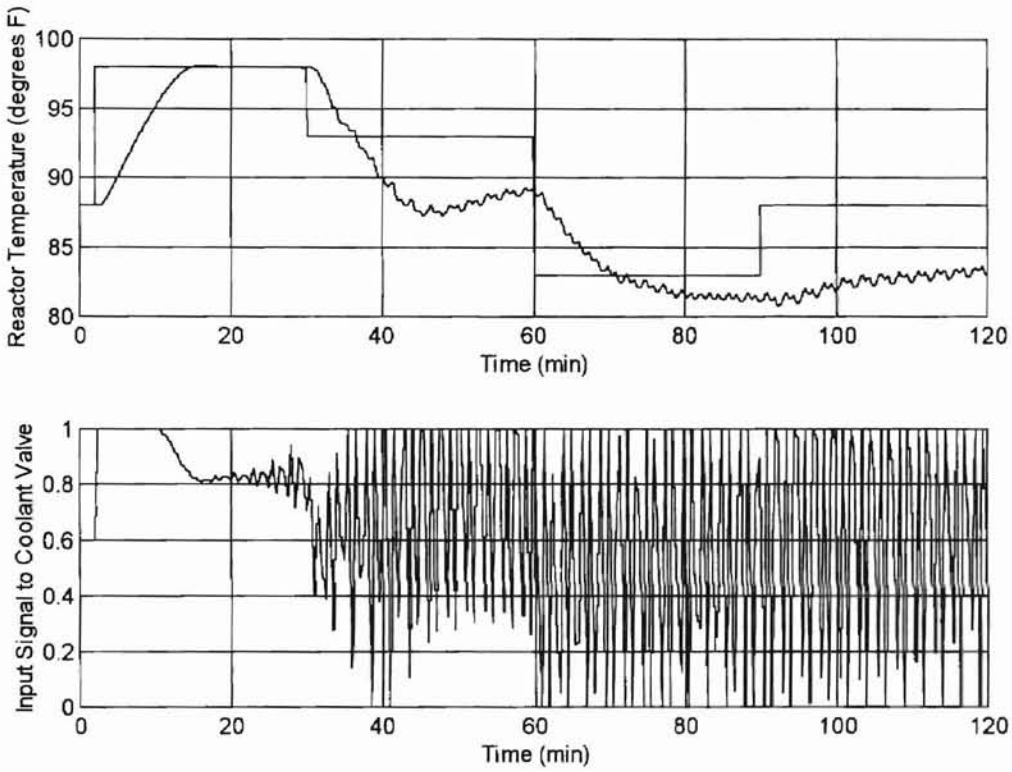

Figure 59. Setpoint Change Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model Using a CHSF of 2 with Model Mismatch Correction Enabled

designing and tuning a neural network controller, how well it will be able to handle disturbances.

When designing an inverse process model neural network controller, the selection of how many sampling periods to set the future goal at is quite crucial to the disturbance rejection capabilities of the controller. The farther out the future goal is set, the less aggressive the controller will be, and, as a result, the disturbance rejection capabilities will degrade. It should also be noted that the selection of the number of sampling periods associated with the future goal is permanent once the model is generated and can only be effectively changed by adjusting the control horizon. One can never use a control horizon any shorter than that for which the model was trained. Thus, one should try to choose the number of sampling periods associated with the future goal as close to the system delay as possible for the best, or most aggressive, disturbance rejection. However, one must also consider how well the controller will perform setpoint changes, which usually results in the future goal being set farther out.

Once the architecture of the inverse process model is set, the controller can be tuned by changing the control horizon or adding a filter. As previously discussed, it is always preferred to adjust the control horizon over using a filter. While both can make the controller less aggressive, only adjusting the control horizon still utilizes the full potential of the dynamics in the neural network model. However, as was the case when tuning for setpoint changes, one must try to find a set of tuning parameters that will achieve both control objectives, setpoint changes and disturbance rejection, well.

Comparison of IS, IR, and EIR Models for Disturbance Rejection

The T(t+2) inverse process models were not considered due to their instability problems. However, previous experience has shown that T(t+2) inverse process models are much more aggressive at rejecting disturbances than those using three sampling periods.

The IS models were more responsive to disturbances than the IR models. This type of behavior was anticipated, since the IS models were more sensitive than IR ones and should therefore have responded more vigorously. However, while this might sound more desirable, it is not. If the controller overreacts to the disturbance, it will simply take longer to achieve steady state again. This is usually the case with the IS models, especially the one using two sampling periods for the future goal. Also, the controller can become unstable if the disturbance is large enough and, as a result, will oscillate and possibly become unstable.

Three different disturbances were used to test each controller. The first, a three degree increase in the inlet temperature of the feed, occurred at 0 minutes. The second, a three degree increase in the cooling water inlet temperature, occurred at 10 minutes. The third and final disturbance, a 3% increase in the inlet concentration, occurred at 20 minutes. All of the disturbances accumulated throughout the simulation so that, at the end, the controller was rejecting three disturbances simultaneously.

Consider Figure 60, which shows the performance of the T(t+3) IR model with a CHSF of 1 and without model mismatch correction, as an example of why model mismatch correction is necessary to reject disturbances. The controller can only partially

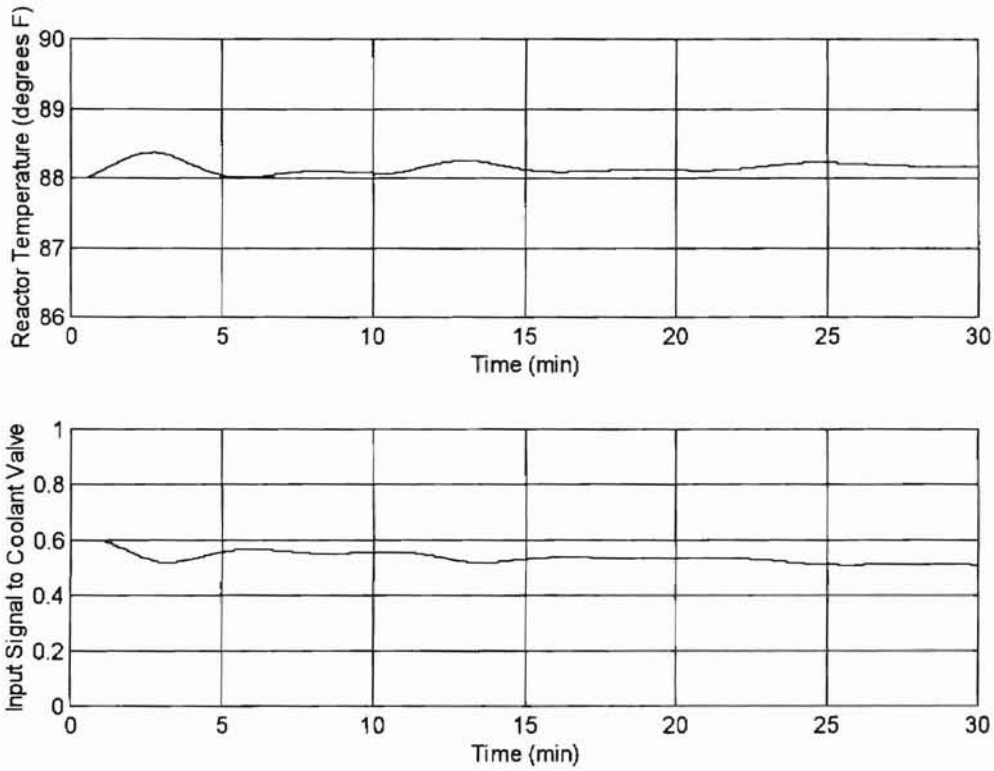

Figure 60. Disturbance Rejection Performance of the T(t+3) Ramp-Trained Inverse Process Model Using a CHSF of 1 with Model Mismatch Correction Disabled

compensate for each disturbance and, as a result, the amount of offset increases as each new disturbance is encountered. However, the remaining offset can be eliminated, and the disturbance completely rejected, through the use of model mismatch correction, in which the controller will attribute the remaining offset to model mismatch.

Figure 61 shows the disturbance rejection performance of the T(t+3) IS model at the same operating temperature that was used during training. The controller used a CHSF of 2 and model mismatch correction to reject the disturbances. The controller was already becoming unstable from trying to compensate for the first disturbance when the second one started. The second and third disturbances simply caused the controller to become unstable.

The T(t+3) IR model reacted less aggressively and was able to provide a much smoother rejection of disturbances while also minimizing excessive valve movement. Figure 62 shows the disturbance rejection performance of the IR controller. As with the IS model, a CHSF of 2 and model mismatch correction were used. The controller was able to successfully reject all three disturbances without excessive valve movement. To illustrate the IR models robustness, the same test was performed at 10 degrees above and below the operating point at which the model was trained. Figure 63 shows the results of the disturbance rejection test 10 degrees above the original operating point. The controller was again able to compensate for all three disturbances without becoming too aggressive or unstable. Figure 64 shows the results of the disturbance rejection test 10 degrees below the original operating point. While the controller does successfully drive

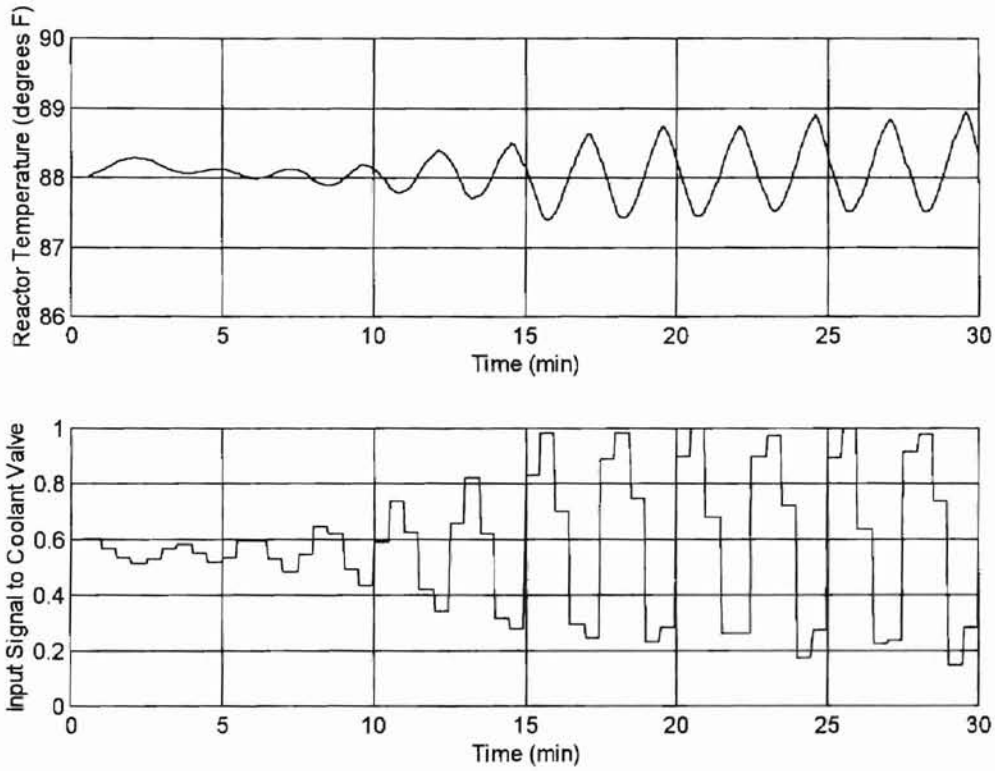

Figure 61. Disturbance Rejection Performance of the T(t+3) Step-Trained Inverse Process Model Using a CHSF of 2 with Model Mismatch Correction Enabled

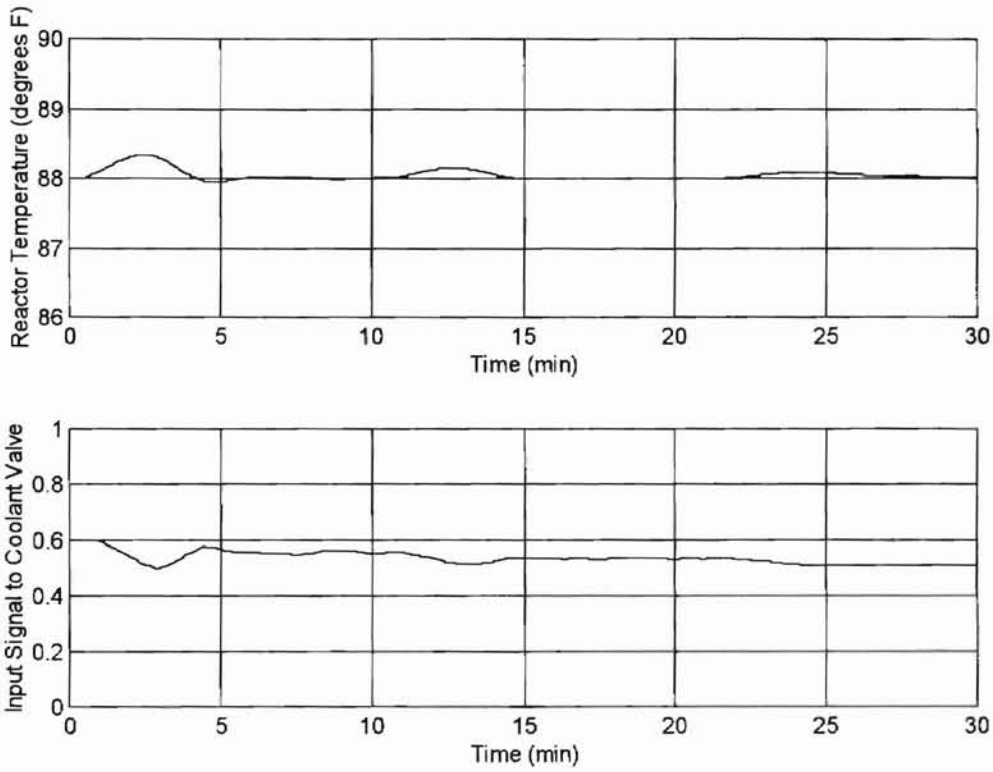

Figure 62. Disturbance Rejection Performance of the T(t+3) Ramp-Trained Inverse Process Model Using a CHSF of 2 with Model Mismatch Correction Enabled

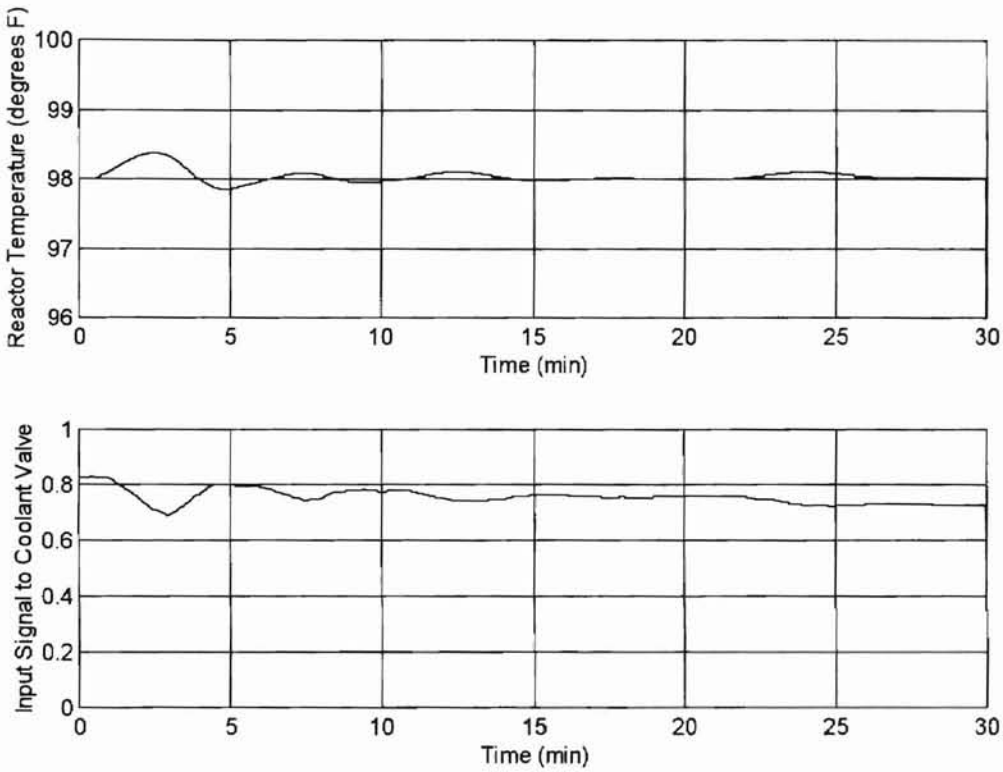

Figure 63. Disturbance Rejection Performance of the T(t+3) Ramp-Trained Inverse Process Model 10 Degrees Above the Operating Point of the Training Data Using a CHSF of 2 with Model Mismatch Correction Enabled

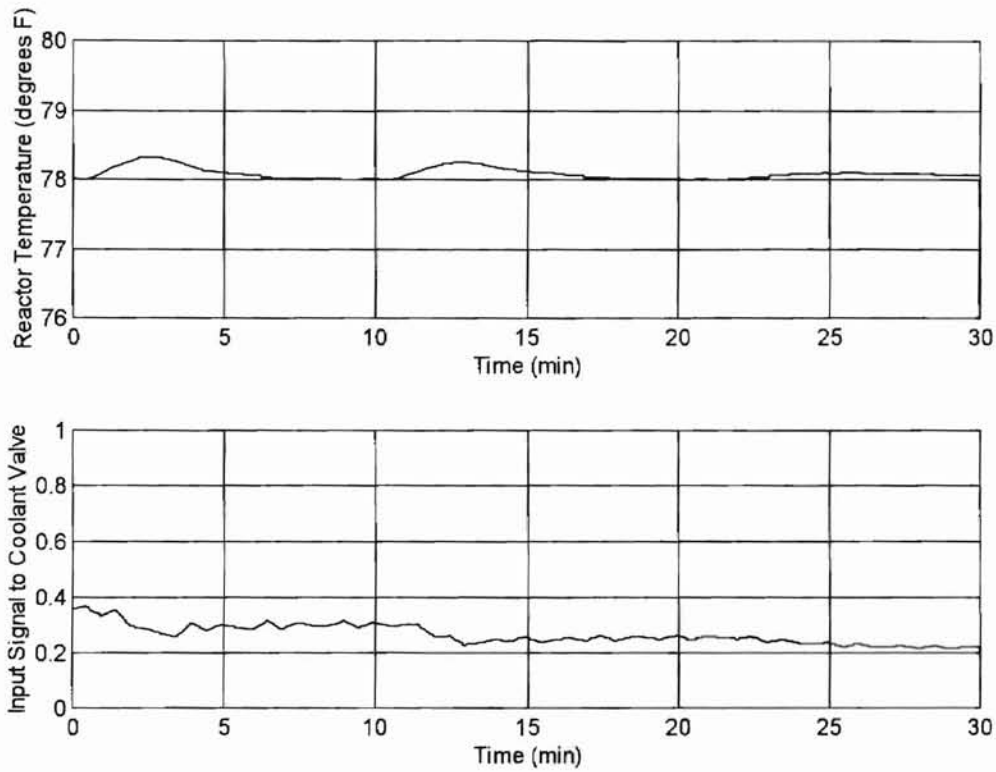

Figure 64. Disturbance Rejection Performance of the T(t+3) Ramp-Trained Inverse Process Model 10 Degrees Below the Operating Point of the Training Data Using a CHSF of 2 with Model Mismatch Correction Enabled

the temperature back to the setpoint, the control actions are a little more aggressive in this case due to the faster process dynamics.

Finally, the T(t+3) EIR controller was evaluated to determine its disturbance rejection capabilities. Figure 65 shows the disturbance rejection performance of the equivalent ramp controller at the original operating point. Initially, the controller appeared to be capable of compensating for the first disturbance. However, one can see that, as the temperature moved back to the setpoint, the controller action became more aggressive. When the second disturbance started, the controller showed a further increase in its over-manipulation of the coolant valve. By the time the third disturbance started, the controller was already appearing to go unstable. The third disturbance resulted in the controller becoming completely unstable.

Overall, only the T(t+3) IR controller was able to provide any disturbance rejection capabilities. Furthermore, it was able to reject disturbance at operating points 10 degrees above and 10 degrees below the operating point at which the model was trained. The poor performance of the T(t+3) IS and T(t+3) EIR controllers can be attributed to their high sensitivities.

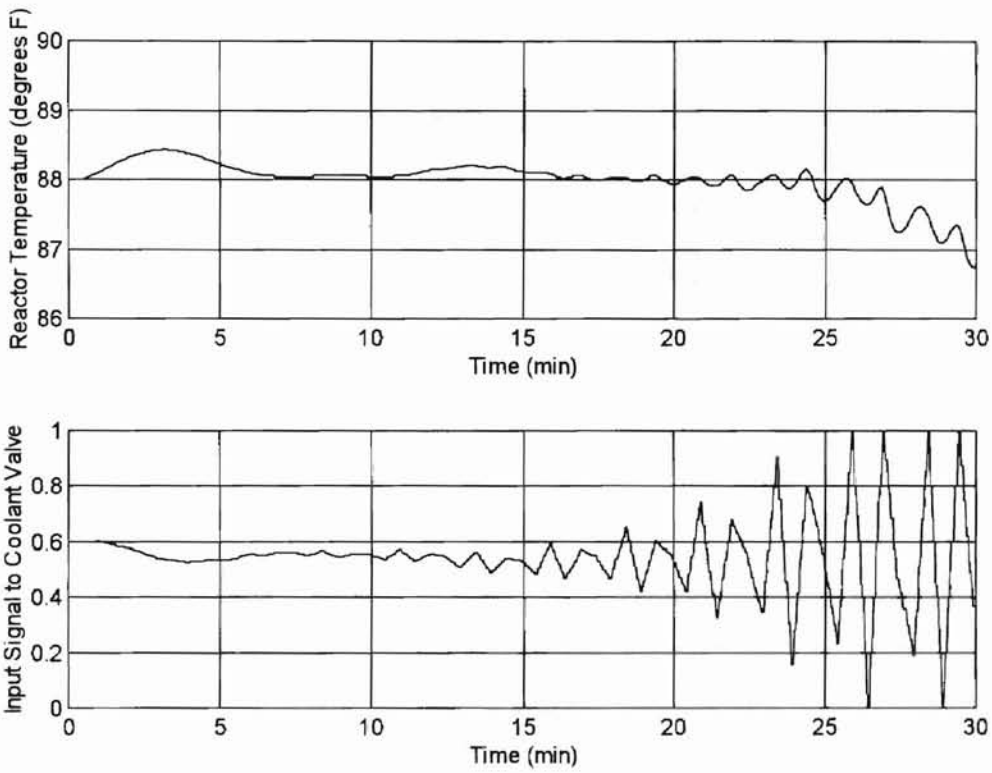

Figure 65. Disturbance Rejection Performance of the T(t+3) Equivalent Ramp-Trained Inverse Process Model Using a CHSF of 2 with Model Mismatch Correction Enabled

CHAPTER V. CONCLUSIONS

This study has shown that inverse neural network controller training sets that use ramp inputs instead of step inputs to obtain inverse process models have three important advantages: better modeling capability, decreased model sensitivity to system changes, and increased overall performance of the controller. Furthermore, this study demonstrated that decreased model prediction error results in better control system performance.

Based on the literature review, this is the first study to evaluate the use of ramp inputs to generate neural network training data. Direct inverse model neural network controllers have not been widely employed due to the problems encountered when trying to approximate a highly excited input signal. Much of the difficulty in approximating an input signal can be attributed to the fact that a typical input signal, whether it is step inputs, PRBS, or even random noise, contains many points at which the slope is infinite. This type of function is extremely difficult to approximate. On the other hand, a signal that uses ramp inputs is easier to approximate since it contains no infinite slopes. As a result, inverse neural network models trained using ramp input signals can achieve much lower prediction errors, typically an order of magnitude less, than models generated using step input data. The improved modeling capability, which is due to smaller model error, results in superior controller performance for setpoint changes and disturbance rejection in the form of decreased overshoot and settling times.

Due to the nature of the ramp input training signal, which requires the network to learn and use a less aggressive control behavior, the ramp-trained inverse process models

are less sensitive to system changes and are more stable when rejecting disturbances and making setpoint changes. In other words, a neural network can more accurately learn a less aggressive control behavior while simultaneously reaching a lower model error that results in better tracking and disturbance rejection capabilities.

Using ramp tests to generate inverse neural network process model training data provides one with a powerful tool for obtaining more accurate and better behaved inverse neural network controllers. The benefits of using ramp tests were demonstrated in this study by making a direct comparison with neural network models obtained using traditional step test data.

CHAPTER VI. RECOMMENDATIONS

Three recommendations to improve this work are as follows: improve the network structure, produce a more sophisticated model mismatch algorithm to complement the inverse neural network controller, and better quantify the relationship between model error and overall performance.

While the effects on the selection and type of inputs and how much information for each type of input should be provided to obtain a good inverse process model was considered in this study, improvement of the network structure needs to be further evaluated. A rigorous analysis of the effects of varying network parameters, such as the number of neurons, the number of layers, *etc.*, on controller performance would help establish guidelines that would aide in generalizing the process to obtain an inverse neural network controller. Investigation concerning the type and amount of process information necessary to allow the model to generalize well would further contribute to the development of a framework of inverse model controller development. In addition, studying how different desired future reactor temperatures (*e.g.* T(t+2), T(t+3), *etc.*) effect controller performance and tuning would be useful.

In this study, the algorithm used to correct for model mismatch was straightforward. While it has its shortcomings, it did allow the two types of inverse models to be compared in a disturbance rejection mode of operation. However, some tracking problems were evident when there was a significant amount of model mismatch.

A relationship between model error and network sensitivity was observed and analyzed quantitatively while the influence of model error on network performance was

only analyzed qualitatively. The theoretical development of a framework, consistent with the results of this study, that quantitatively relates specific effects of model error on aspects of controller performance such as overshoot and settling time would be useful.

REFERENCES

Caudill, M. Expert Networks. *Byte* **16,** 108-116 (1991).

Cheung, T. F., Kwapong, O., and J. I. Elsey. Building Empirical Models of Process Plant Data by Regression or Neural Network. *Proceedings of the American Control Conference*, Chicago, IL, 3-1922-1925 (1992).

Demuth, H., and M. Beale. *Neural Network Toolbox User's Guide*. The MathWorks, Inc., Natick, MA (1992).

Downs, J. J., and E. F. Vogel. A Plant-Wide Industrial Process Control Problem. *Computers Chem. Engng* **17,** 245-255 (1993).

Eskandarian, A., Bedewi, N. E., Kramer, B. M., and A. J. Barbera. Dynamics Modeling of Robotic Manipulators Using an Artificial Neural Network. *Journal of Robotic Systems* **11,** 41-56 (1994).

Farotimi, O., Dembo, A., and T. Kailath. A General Weight Matrix Formulation Using Optimal Control. *IEEE Trans. Neural Netw.* **2,** 378-394 (1991).

Garcia, C. E., Prett, D. M., and M. Morari. Model Predictive Control: Theory and Practice--a Survey. *Automatica* **25,** 335-348 (1989).

Garcia, C. E., Ramaker, B. and J. Pollard. Total Process Control - Beyond the Design of Predictive Controllers. *Chemical Process Control - CPCIV*, Padre Island, Texas, 335-361 (1991).

Hammerstrom, D. Working with Neural Networks. *IEEE Spectrum* **30,** 46-53 (1993).

Hernandez, E., and Y. Arkun. Neural Network Modeling and an Extended DMC Algorithm to Control Nonlinear Systems. *Proceedings of the American Control Conference*, Chicago, IL, 3-2454-2459 (1990).

Hernandez, E., and Y. Arkun. Study of the Control-Relevant Properties of Backpropagation Neural Network Models of Nonlinear Dynamical Systems. *Computers Chem. Engng* **16,** 227-240 (1992).

Himmelblau, D. M. Fault Detection in Heat Exchangers. *Proceedings of the American Control Conference*, Chicago, IL, 3-2369-2372 (1992).

Hofland, A. G., Morris, A. J., and G. A. Montague. Radial Basis Function Networks Applied to Process Control. *Proceedings of the American Control Conference*, Chicago, IL, 1-480-484 (1992).

Hoskins, J. C., and D. M. Himmelblau. Process Control Via Artificial Neural Networks and Reinforcement Learning. *Computers Chem. Engng* **16,** 241-251 (1992).

Hsu, Y., and C. Yu. A Self-Learning Fault Diagnosis System Based on Reinforcement Learning. *Ind. Eng. Chem. Res.* **31,** 1937-1946 (1992).

Ishida, M., and J. Zhan. Characteristics of Policy-and-Experience-Driven Neural Network when Applied to Level Control. *Journal of Chemical Engineering of Japan* **25,** 485-489 (1992a).

Ishida, M., and J. Zhan. Control of a Process with Time Delay by Policy-and-Experience-Driven Neural Networks. *Journal of Chemical Engineering of Japan* **25,** 763-766 (1992b).

Ishida, M., and J. Zhan. Neural Network Control for a MIMO Process with Time Delay. *Journal of Chemical Engineering of Japan* **26,** 337-339 (1993).

Joseph, B., and F. W. Hanratty. Predictive Control of Quality in a Batch Manufacturing Process Using Artificial Neural Network Models. *Ind. Eng. Chem. Res.* **32,** 1951-1961 (1993).

Karim, M. N., and S. L. Rivera. Application of Neural Networks in Bioprocess State Estimation. *Proceedings of the American Control Conference*, Chicago, IL, 1-495-499 (1992).

Kasparian, V., and C. Batur. Neural Network Structure for Process Control Using Direct and Inverse Process Model. *Proceedings of the American Control Conference*, Chicago, IL, 1-562-566 (1992).

Kavuri, S. N., and V. Venkatasubramanian. Representing Bounded Fault Classes Using Neural Networks with Ellipsoidal Activation Functions. *Computers Chem. Engng* **17,** 139-163 (1993).

Kramer, M. A. Autoassociative Neural Networks. *Computers Chem. Engng* **16,** 313-328 (1992a).

Kramer, M. A., Thompson, M. L., and P. M. Bhagat. Embedding Theoretical Models in Neural Networks. *Proceedings of the American Control Conference*, Chicago, IL, 1-475-479 (1992b).

Krieger, J. Chemical Engineering Seeks to Address Worldwide Concerns. *Chemical & Engineering News* **69,** 16-22 (1991).

Kyung, K. H., Lee, B. H., and M. S. Ko. Acceleration Based Learning Control of Robotic Manipulators Using a Multi-Layered Neural Network. *IEEE Transactions on Systems Man and Cybernetics* **24,** 1265-1272 (1994).

Levin, A. U., and K. S. Narendra. Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization. *IEEE Trans. Neural Netw.* **4,** 192-206 (1993).

Lu, Y. The New Generation of Advanced Process Control. *Control Engineering* **39,** 21-23 (1992).

Marquardt, D. An Algorithm for Least Squares Estimation of Non-Linear Parameters. *J. Soc. Ind. Appl. Math.* 431-441 (1963).

Mavrovouniotis, M. L., and S. Chang. Hierarchical Neural Networks. *Computers Chem. Engng* **16,** 347-369 (1992).

Moran, A., and M. Nagai. Optimal Preview Control of Rear Suspension Using Nonlinear Neural Networks. *Vehicle System Dynamics* **22,** 321-334 (1993).

Morari, M., and E. Zafiriou. *Robust Process Control.* Prentice Hall, Englewood Cliffs, NJ (1989).

Morris, A. J., Montague, G. A., and M. J. Willis. Artificial Neural Networks: Studies in Process Modeling and Control. *Chemical Engineering Research & Design* **72,** 3-19 (1994).

Nahas, E. P., Henson, M. A., and D. E. Seborg. Nonlinear Internal Model Control Strategy for Neural Network Models. *Computers Chem. Engng* **16,** 1039-1057 (1991).

Nikolaou, M., and V. Hanagandi. Control of Nonlinear Dynamical Systems Modeled by Recurrent Neural Networks. *AIChE J.* **39,** 1890-1894 (1993).

Normandin, A., Thibault, J., and B. P. A. Grandjean. Optimizing Control of a Continuous Stirred Tank Fermenter Using a Neural Network. *Bioprocess Engineering* **10,** 109-113 (1994).

Osborne, D. A. Neural Networks Provide More Accurate Reservoir Permeability. *Oil & Gas Journal* **90,** 80-83 (1992).

Piovoso, M. J., Kosanovich, K. A., Rokhlenko, V., and A. Guez. A Comparison of Three Nonlinear Controller Designs Applied to a Non-Adiabatic First-Order Exothermic Reaction in a CSTR. *Proceedings of the American Control Conference*, Chicago, IL, 1-490-494 (1992).

Psichogios, D. C., and L. H. Ungar. Direct and Indirect Model Based Control Using Artificial Neural Networks. *Ind. Eng. Chem. Res.* **30,** 2564-2573 (1991).

Ray, W. H. *Advanced Process Control.* McGraw-Hill, New York (1981).

Rehbein, D. A., Maze, S. M., and J. P. Havener. The Application of Neural Networks in the Process Industry. *ISA Transactions* **31,** 7-13 (1992).

Richalet, J., Rault, A., Testud, J. L., and J. Papon. Model Predictive Heuristic Control: Applications to Industrial Processes. *Automatica* **14,** 413-428 (1978).

Ricker, N. L. Model-Predictive Control: State of the Art. *Chemical Process Control - CPCIV*, Padre Island, Texas, 271-296 (1991).

Rouhani, R. and R. K. Mehra. Model Algorithmic Control (MAC); Basic Theoretical Properties. *Automatica* **18,** 401-414 (1982).

Rudd, J. B. Using a Neural Network System for Advanced Process Control. *Tappi Journal* **74,** 153-159 (1991).

Seborg, D. E., Edgar, T. F., and D. A. Mellichamp. *Process Dynamics and Control.* John Wiley & Sons, New York, NY (1989).

Sheppard, C. P., Gent, C. R., and R. M. Ward. A Neural Network Based Furnace Control System. *Proceedings of the American Control Conference*, Chicago, IL, 1-500-504 (1992).

Sorsa, T., and H. N. Koivo. Application of Artificial Neural Networks in Process Fault Diagnosis. *Automatica* **29,** 843-849 (1993).

Su, H., Minderman, Jr., P. A., and T. J. McAvoy. Control and System Identification Using Elements of Neural Network Computation Engineering. *Proceedings of the American Control Conference*, Chicago, IL, 1-485-489 (1992a).

Su, H., McAvoy, T. J., and P. Werbos. Long-Term Predictions of Chemical Processes Using Recurrent Neural Networks: A Parallel Training Approach. *Ind. Eng. Chem. Res.* **31,** 1338-1352 (1992b).

Thibault, J. and B. P. A. Grandjean. Neural Networks in Process Control - a Survey. *Advanced Control of Chemical Processes - ADCHEM'91*, Toulouse, FR, 8-251-260 (1991).

Zhang, Q., Reid, J. F., Litchfield, J. B., Ren, J. and S. Chang. A Prototype Neural Network Supervised Control System for Bacillus thuringiensis Fermentations. *Biotechnology and Bioengineering* **43,** 483-489 (1994).

Zurada, J. M. *Introduction to Artificial Neural Systems*. West Publishing Company, New York, NY (1992).

APPENDIX A

Neural Network Mechanics

Neural network terminology, structure and function are discussed here. These are important concepts necessary for understanding how a neural network inverse process model is obtained and implemented in an IMC control strategy. A neural network performs a transformation on a set of inputs to produce a set of outputs. A neural networks ability to utilize nonlinear monotonic functions allows these transformations to be nonlinear and, as a result, gives them a great advantage over classical modeling methods. The function associated with a neuron is referred to as its activation function. This function can be linear and as simple as $f(x)=x$, which results in an output equal to the input. However, activation functions are usually nonlinear, such as $f(x)=1/(1+\exp(-x))$, which is commonly referred to as a logarithmic-sigmoidal function.

A neural network consists of three types of layers, these are the input layer, hidden layer(s), and an output layer. The input layer merely propagates the inputs to the first hidden layer. Hidden layers contain the neurons where intermediate inputs and outputs are processed. The output layer is simply the outputs of the last hidden layer. The inputs are usually normalized to aide the networks ability to learn, while the range of values of the outputs is dependent on the type of activation function used in the last hidden layer. Typically, a feedforward network will have no more than two hidden layers of neurons. In fact, it is recommended that no more than three hidden layers of neurons be used, since there is no real gain in generalizability seen when using more than three hidden layers. All neurons in a hidden or output layer are connected to all of the inputs from the previous layer. Each connection has a weighting factor associated with it. The value of these

weights is changed during the learning phase until the network has properly learned the desired relationship between the inputs and corresponding outputs. It should be noted that the initial values of the weights are randomly selected values between negative one and plus one. This, in combination with normalization of the inputs, keeps the network from having to excessively change the weights and, as a result, greatly reduces training time. Also, each neuron has its own bias, which is also changed during the training phase and further enhances the networks ability to learn.

A Single Neuron

Figure A-1 shows a graphical representation of a single neuron with multiple inputs. The neuron takes the input N, which is equal to the sum of input P(1) multiplied by weight W(1,1) through input P(R) multiplied by weight W(1,R) plus a bias B, and calculates an output A using an activation function F. The inputs, P(1) through P(R) can be outputs from another hidden layer or from the input layer. This output is then propagated to the next layer of the network whether it is another hidden layer or the output layer. When dealing with complex networks it is often helpful to quantify the network using matrices and vectors. For example, as shown in Figure A-1, the output of the neuron could be written as A=F(W*P+B) where F is the activation function, A is a 1 by 1 vector containing the output, B is a 1 by 1 vector with the bias, P is a R by 1 vector of inputs, and W is a 1 by R vector whose values represent the weights for each connection between the inputs and the neuron. This is the notation used in the Matlab Neural Network Toolbox (Demuth and Beale, 1992). Figure A-2 shows an equivalent Matlab representation of Figure A-1. The equivalent Matlab notation is being included for

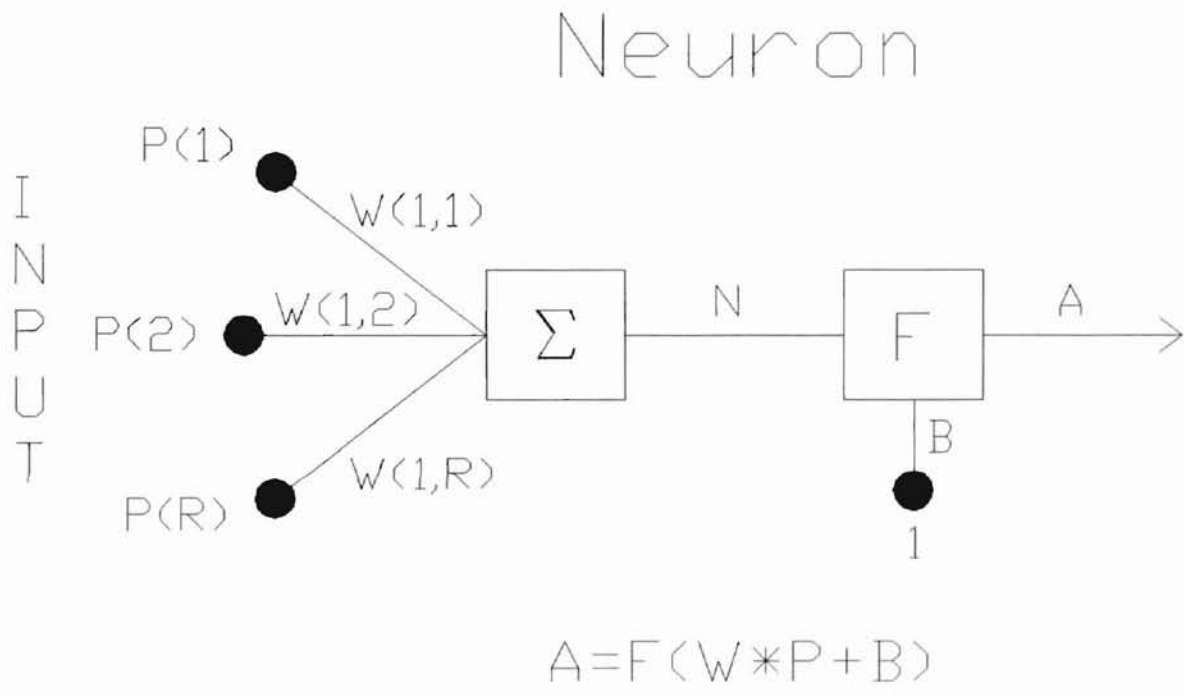

Figure A-1. A Multiple Input Neuron

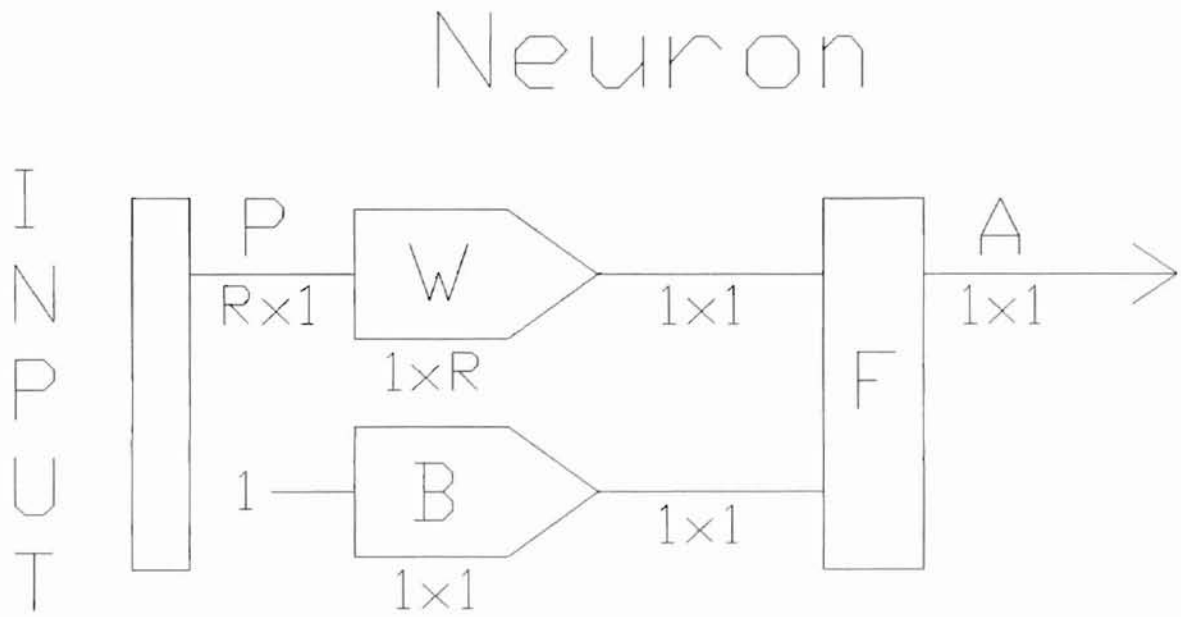

Figure A-2. Matlab Representation of a Multiple Input Neuron

two reasons. First, it provides an efficient method of defining a neural network. All weights, biases, and outputs can be easily put into matrices or vectors for analysis. Second, it allows one to focus more on how the network functions without getting caught up in detailed calculations. The Matlab neural network toolbox was used exclusively in this study. Now that we have a better understanding of neural network basics, the next logical steps are to look at how a layer of multiple input neurons functions and how multiple layers of multiple input neurons function.

A Layer of Neurons

A neuron in a layer of neurons functions just like a single multiple input neuron. The primary difference, as can be seen in Figure A-3, is that there are more outputs generated, one from each neuron. Also, there are many more weights due to the increased number of connections between the inputs and the neurons. As previously shown, for a layer with S neurons in it, the output of the layer could be written as A=F(W*P+B) where F is the activation function, A is now a S by 1 vector containing the outputs, B is now a S by 1 vector with the biases, P is a R by 1 vector of inputs, and W is now a S by R vector whose values represent the weights for each connection between the inputs and the neuron. Figure A-4 shows an equivalent Matlab representation of Figure A-3.

Multiple Layers of Neurons

Multiple layers of multiple input neurons are typically used to form complex networks capable of learning very nonlinear relationships. Each layer of neurons processes its inputs using the method previously discussed and then propagates its outputs to the next layer, whether it be a hidden or an output layer. One should understand that a

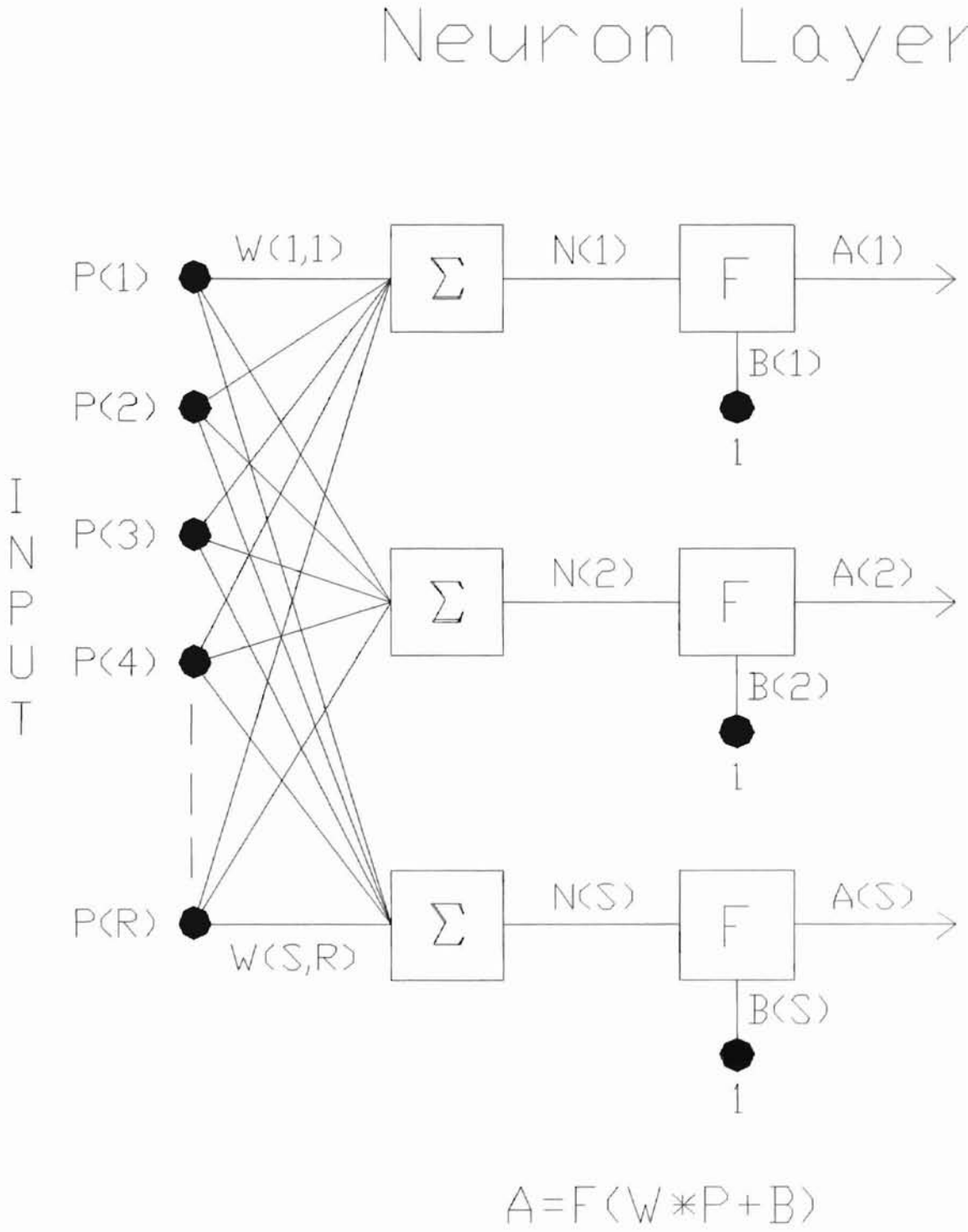

Figure A-3. A Layer of Multiple Input Neurons

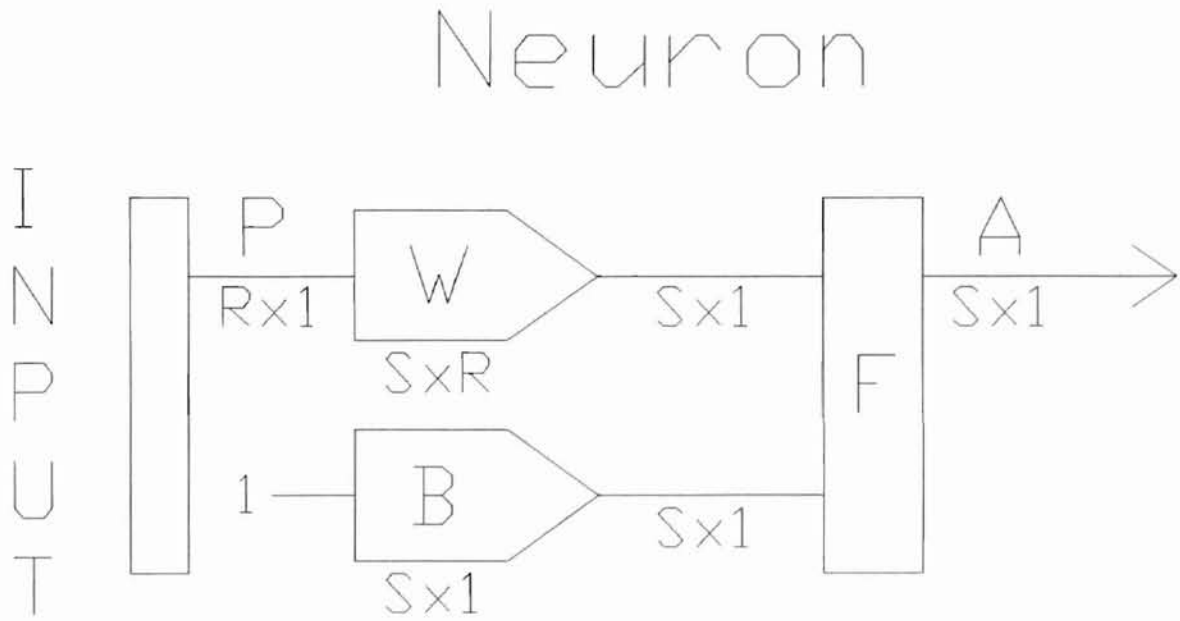

Figure A-4. Matlab Representation of a Layer of Multiple Input Neurons

neural network can be as simple as a single input to a single neuron with a single output. On the other hand, a neural network can be as complex as the one shown in Figure A-5. The network in Figure A-5 has one input layer, three hidden layers, and one output layer. Figure A-6 shows an equivalent Matlab representation of Figure A-5.

Characteristics of Neural Networks

Neural networks exhibit many characteristics that make them desirable for use in IMC as well as other applications and advanced control strategies. Neural networks, unlike conventional data processing techniques, are very powerful when applied to problems whose solution requires knowledge that is difficult to specify, but for which there is an abundance of examples. Many complex process control problems fit this criterion, as examples of system response to different control system stimuli are readily available and most industrial process information is saved continuously. Another capability of neural networks is the finding of solutions to complex nonlinear problems without the need for any *a priori* knowledge as to the nature of the solution. This is especially important when supervised learning or the aid of an expert is either not feasible or impossible. Neural networks also have the ability to generalize from examples, thus giving them the capability to interpolate and extrapolate. However, great caution should be taken when using a neural network for extrapolation. Neural networks are also capable of extracting essential information from noisy data and performing gross error removal.

These are just a few of the many abilities of neural networks. They have great potential when used carefully and intelligently. The section "Neural Networks in Control" in Chapter 2 provides a sample of how they have already been studied and utilized in various applications.

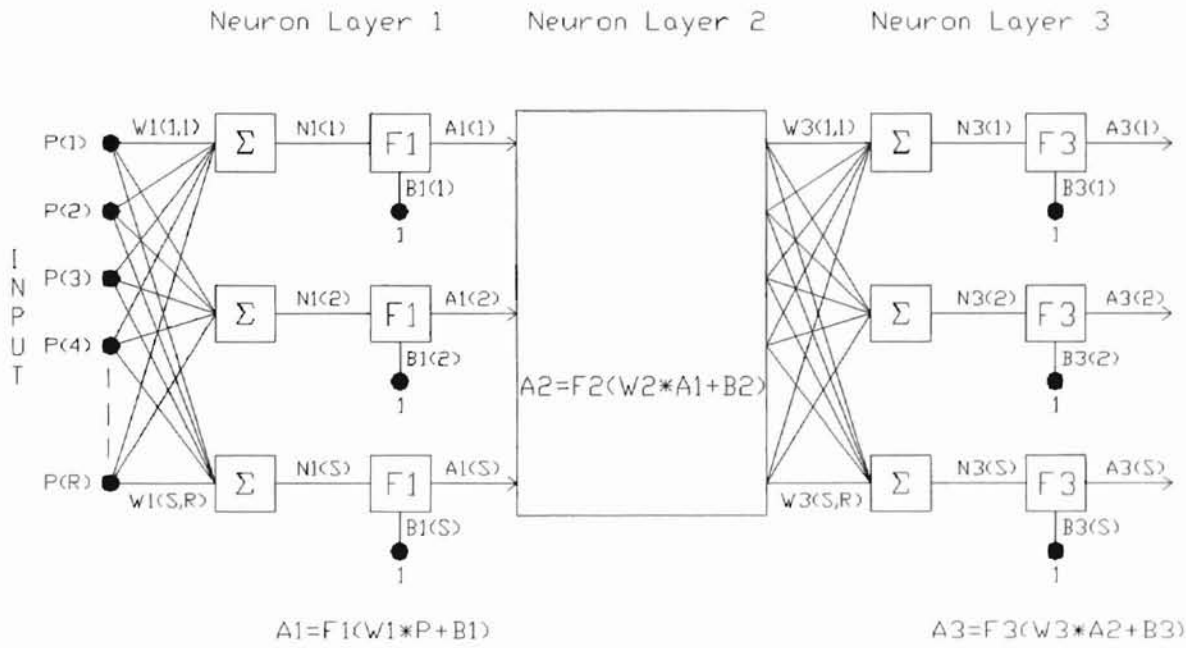

Figure A-5. Multiple Layers of Neurons

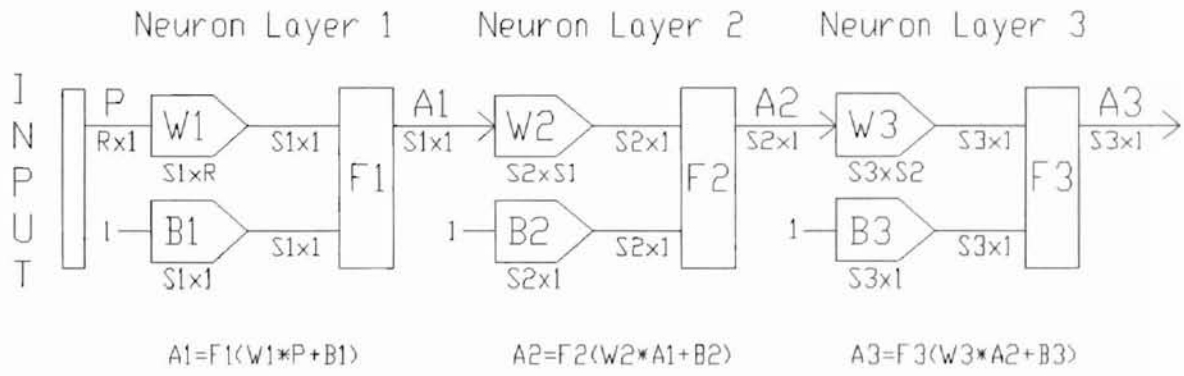

Figure A-6. Matlab Representation of Multiple Layers of Neurons

APPENDIX B

How to Obtain an Inverse Process Model Using Neural Networks

The method used to obtain an inverse process model is not straightforward. While each step in the process is not difficult, there are many decisions along the way that must be made without any general rules of guidance. Some of these include the type of neural network, the architecture of the network, the type of activation functions in the network, proper initialization of the network weights and biases, the training method to be used, the training algorithm to be used, what process variables will be network inputs, what process variables will be network outputs, the structure of the network inputs and outputs, and where and how to obtain training data. Each of these items is very crucial to the overall success of the model development. While there might be many different combinations that would work for one process, some processes are much less flexible.

Neural Network Type

The neural networks used in this study are all feed forward networks. In a feed forward network, all inputs propagate through the first layer of the network, with the outputs from the first layer propagating through each of the remaining layers until the output layer is reached. This is the most popular type of neural network and is commonly used with a variety of training algorithms. While there are an almost unlimited number of neural network types, such as radial basis function networks, Hamming networks, perceptrons, ART, *etc.*, the basic idea is the same: to perform a nonlinear transformation on a given set of inputs and map them to a corresponding set of outputs.

Neural Network Architecture

There is no set of rules for choosing the architecture of the network. However, one typically begins by considering the number of network inputs and outputs. There are two things that must be considered: the number of layers in the network and the number of neurons in each layer. Usually the number of hidden layers, which are those between the input and output layers, will not exceed three. This is because using more than three hidden layers becomes computationally inefficient during the training stage, with minimal benefits in terms of generalization. If the network is too big, it will simply memorize the training set. This phenomenon is checked for by verifying that none of the values of the weights have become approximately zero after the training process. The goal is to use as few hidden layers and neurons as possible. This will maximize the ability of the network to generalize and be more computationally efficient. However, care should be taken that the network is not reduced too much in size, since this could result in poor performance and high training errors.

Neural Network Activation Functions

Care should also be taken in the selection of the type of activation function(s) that will be used in the network. Activation functions are usually chosen so that all neurons in a given layer have the same type of activation function. While linear activation functions, such as hard-limits and pure-linears can be used, and are sometimes ideal for a particular application, nonlinear functions, such as logarithmic-sigmoidal and tangential-sigmoidal, are more commonly used.

The type of inputs and desired outputs is important in the selection of the activation function. For example, if one desires outputs between zero and one, a logarithmic-sigmoidal function would be ideal, since it provides only values in this range, whereas a tangential-sigmoidal function provide values between negative one and one. If tangential-sigmoidal activation functions were used, the network would not train as efficiently and would probably require more neurons than one using logarithmic-sigmoidal ones. The shape of the activation function used also becomes very important during training, since the derivative of the activation function is utilized. For example, if a network has only linear activation functions, it will only be capable of learning a linear mapping, and will therefore not be very useful for nonlinear applications.

It should be noted that the use of nonlinear activation functions in a parallel processing environment is where neural networks derive their power. For example, a neural network with any number of hidden layers utilizing linear neurons can do no better than a neural network with only one hidden layer containing a single linear neuron. Since most systems are nonlinear, and one of the motivations to use a neural network as a process model is its ability to model nonlinearities, the activation functions used in this study were logarithmic-sigmoidal.

Neural Network Initialization

One of the most important steps in the process of training a neural network is the proper initialization of the network weights and biases. If this is not done properly, it can greatly increase the training time or, in some cases, keep the network from converging at all. For example, if a given neuron has a logarithmic-sigmoidal activation function, the

derivative is very small at values close to zero and one. Thus, the initial bias and weight values for the neuron should be chosen so that the initial output of the neuron is close to 0.5, where the derivative is greatest and the neuron can be more easily adjusted during training. While there are numerous methods for initializing weights and biases, there is no "best" method that can be used in every case. This is because many of the initialization procedures are tailored to certain types of networks, network architectures, and activation functions. The procedure used to initialize the weights and biases for the networks used in this study was very simple. All weight matrices and bias vectors were randomly generated to be symmetric, with values between one and negative one.

Neural Network Training Method

There two ways to present training data to a neural network during the learning phase. If the network weights and biases are adjusted after each data set has been presented to the network, the training method is termed incremental training. If the entire set of data is presented to the network before any updates to the network weights and biases are made, the training method is referred to as batch training. In batch training, the changes for each data set are stored and then summed up to determine a net change for all weights and biases. Batch training will usually result in a much more stable and quicker network convergence than incremental training, since it is less susceptible to bad data sets and allows the network to learn in a more generalized fashion. The batch training method was used for training all networks in this study.

Neural Network Training Algorithms

There is a great variety of training algorithms to choose from for feed forward networks. Training algorithms simply decrease the error of the training data by adjusting the values of the network weights and biases until the error reaches some preset value, referred to as the convergence criterion. There are a number of ways to look at the error of a network: root-mean-squared error, sum-squared error, integral absolute error, *etc.* The sum-squared error provides an accurate representation of how well the network will perform based on the magnitude of the final error. The integral absolute error was used in this study to obtain equivalently-trained step and ramp inverse neural network models for a baseline comparison of performance between the two types of training signals. The training data for each model consisted of two sets of test data. One set, the training data, was used to actually obtain the neural network model while the other, the test data, was used to validate the model. Training of the network continued until the sum-squared error of the test data began to increase, since further training would only decrease the networks ability to generalize from the training data.

Selection of a training algorithm is very dependent upon the type of network, the activation functions and the mode of use intended for the trained network. Training algorithms, much like initialization techniques, are better suited to some combinations of the above items than others. Furthermore, the way training algorithms work also varies. For example, one might use a steepest descent method while another uses a conjugate gradient descent method. The networks in this study were trained using the Levenberg-

Marquardt algorithm, a conjugate gradient descent method, and were validated using an independent set of test data.

What to use as Network Inputs and Outputs

After all of these network parameters, the type of neural network, the architecture of the network, the type of activation functions in the network, proper initialization of the network weights and biases, the training method to be used, and the training algorithm to be used have been determined, one must decide what process variables will be used as network inputs and outputs. Before this can be done, one must know which variables are to be manipulated and which variables are to be controlled. This discussion will consider a single input, single output (SISO) system. Thus, there will be only one manipulated variable and one controlled variable. However, it should be kept in mind that the same ideas apply to systems with multiple inputs, multiple outputs, or both.

While there might be a wealth of process information available for input into the network, one does not want to burden the learning process with redundant information that will simply waste network resources. For example, consider digital control of the CSTR with time delay shown in Figure B-1, where one has process information on the height (H), volume (V), and exit concentration (C_{out}) of the tank readily available. If one were interested in controlling the exit concentration, one might regulate the height of the tank. Thus, one would want to input information on the height of the tank. However, one would not want to also input the volume of the tank, since this would simply require the network to learn the relationship between the height and diameter of the tank, which is fixed. Furthermore, one would not want to use only the volume of the tank, even though

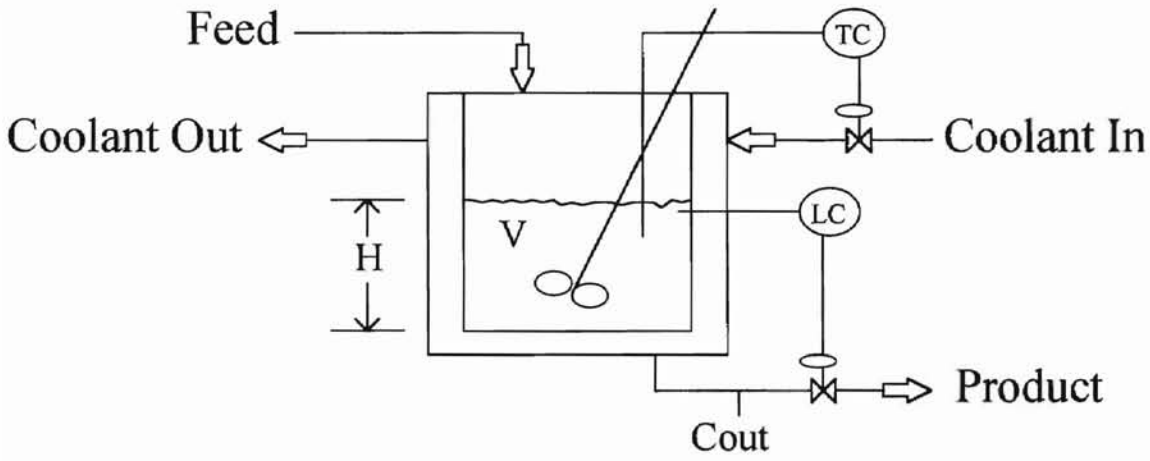

Figure B-1. Nonlinear Continuous Stirred-Tank Reactor

that is what directly effects the exit concentration, because it is not the manipulated variable. The output of the network would simply be the control move necessary to keep the controlled variable, the exit concentration, at the setpoint or move it back to the setpoint. This then raises the question of how the network knows what the setpoint is. The network must have a goal so that it has a basis to determine what control move must be made in an effort to either achieve the setpoint or at least decrease the error between the setpoint and the controlled variable. Thus, the setpoint must also be input to the network. However, the input/output determination process does not end there.

Neural Network Structure

If the setpoint and controlled variable were the only network inputs, the manipulated variable, or output, would be very erratic. This is because the network would only know that there is either no error or an error of some magnitude. The greater the magnitude of the error, the more aggressive the output would change. This type of behavior would lead to instability. Thus, something else needs to be input to make the network function more smoothly. This is where consideration of the structure of the inputs and outputs comes into play. It is not enough to simply determine what variables are being controlled and what variables going to be manipulated. In other words, one cannot simply input the value of the controlled variable and the setpoint and output the manipulated variable and expect the network to function properly. This is where one must think about what is making the network behave erratically and what can be done to remedy this behavior. There is no set of rules for how this is done. However, common

sense and a knowledge of the process, as with virtually all steps of the model development process, will provide some answers as to how one should proceed.

The first thing that can be done to improve the output of the network is to provide it with some past values, as well as the current one, of the controlled variable. This will make the network more stable, since it will learn the relationship between the rate of change of the controlled variable and the necessary aggressiveness of the control move relative to the last control action. However, this brings another parameter into play: how much past information about the controlled variable needs to be supplied. Since this example has time delay, as most chemical engineering processes do, one will need to provide at least enough information so that the effects of any given control action can be seen. For example, if the digital control system was using a sampling time of six seconds and had a time delay of 30 seconds, one would have to provide at least five sampling times worth of process information. While this modification to the input side of the network will result in a more stable model, one can still do much more to improve the performance and stability of the network.

As with the controlled variable, a history of the manipulated variable can also be supplied to the network. This will allow the network to learn the effect of the rate of change of the manipulated variable on the rate of change of the controlled variable. As with the controlled variable history, the number of sampling periods worth of manipulated variable information should be provided to the network is not arbitrary. The amount of information supplied should exceed the time delay window so that the network can see the effects of all control actions. At this point, it should be emphasized that the number of

inputs should be minimized, since too many inputs simply burden the network with redundant information and make it computationally less efficient. It has been this researcher's experience that one should input more information on the manipulated variable than the controlled variable in order to enhance the ability of the network to learn the relationships described above. In other words, one does not want a network that has learned how the controlled variable effects the manipulated variable. It should be apparent by now that there are no rules of thumb or simple and straightforward answers concerning the development of an inverse process model, but that common sense, intuition, and a sound understanding of neural network fundamentals are one's most powerful tools.

The output of the neural network model for this example is simply the manipulated variable, which is the control signal to the valve that regulates the height of the tank. As previously mentioned, for a multiple output system, the network would generate values for all manipulated process variables. Unlike the inputs to the network, the outputs can be directly determined.

How to Obtain Neural Network Training Data

A good data set for use in training a neural network should meet the following two criteria. First, it must provide sufficient excitation of the system so that the network can learn the process dynamics. If the training set is not rich enough, the resulting network will be inadequate for robust control. Second, the size of the training set should be kept as small as possible because, the larger the training set, the longer the time it will take to train the network due to the increased computational requirements. As a training set gets

larger, it begins to acquire redundant information as to how the process behaves in certain situations and at various operating conditions.

One must also determine what type of open-loop tests will be performed to obtain a rich data set. Traditionally, pulsed and stepped inputs have been used to excite a system. However, this study has considered the use of ramped inputs as an alternative. Figure B-2 shows pulsed, stepped, and ramped input signals. Before we can discuss the benefits of using ramped inputs, we must first understand what the process is trying to achieve.

As previously discussed, a neural network is usually utilized to either map inputs from one space to another or to perform what is known as function approximation. When using a neural network in a model-based control strategy, one is concerned with its ability to approximate functions. These functions are simply the outputs of the system during open-loop testing. Consider the same SISO system already discussed, since it is a base case for more complex systems. Because one wants the neural network process model to provide one with the control signal at each sampling period, the manipulated variable will be the network output. The step and ramp test procedures will be the same whether the system actually exists or is simply a computer simulation.

To begin, all controllers that are to receive their signal from the process model must be taken off-line. This should be done once the system has reached steady-state at the desired operating point. The remaining controllers should stay at the steady-state signal during the testing period. This will allow the controllers that are to be neural network-regulated to be stimulated as necessary to perform the step tests. This allows the system to respond in an open-loop mode. It is important that the step test provide a rich

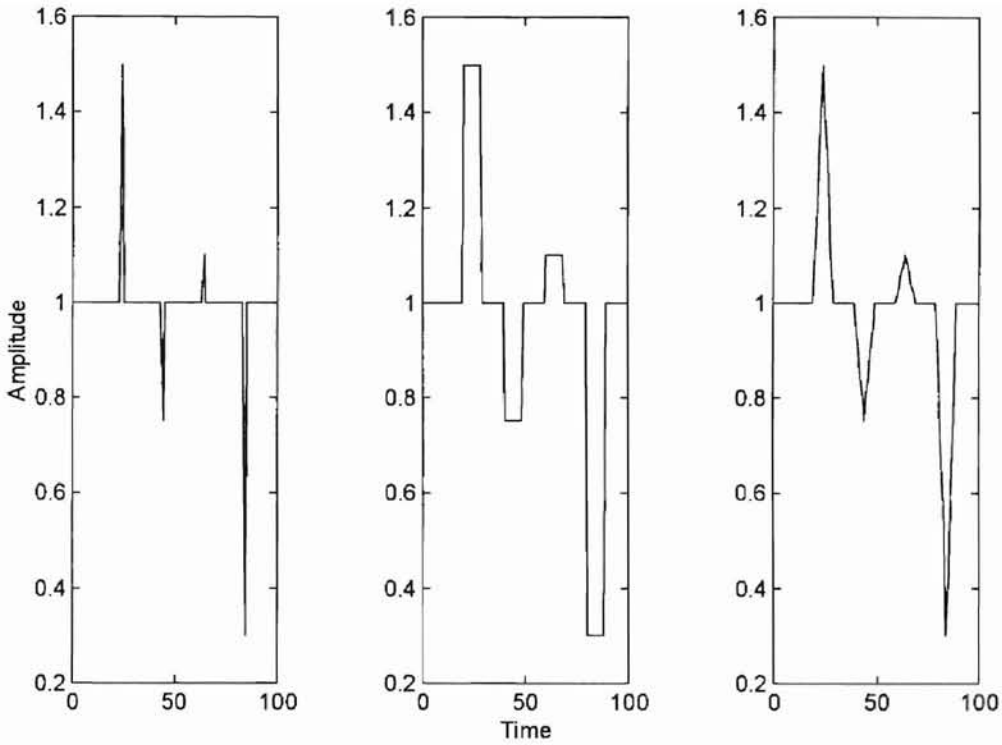

Figure B-2. A Sample of Pulsed, Stepped, and Ramped Signals

enough training set so that the neural network can learn the process dynamics. However, there is no standard method in which the steps should be carried out. Important parameters in the step tests include the frequency and magnitude of the steps. As before, one should know what the delay of the system is so that a proper step time can be chosen. The frequency should not be less than the delay, since it will make it impossible for the network to learn how the input affects the output. The magnitudes should vary over the full range of possible control signals in order to provide a more robust network. The procedure for performing the ramp tests is the same except that the signal is discretely changed in five equal steps instead of a single step.

VITA

Paul A. Belcher

Candidate for the Degree of

Master of Science

Thesis: USE OF RAMP TESTS TO OBTAIN INVERSE NEURAL NETWORK PROCESS MODELS

Major Field: Chemical Engineering

Biographical:

Personal Data: Born in Jacksonville, North Carolina, on June 5, 1970, the son of Gerald and Clara Belcher.

Education: Graduated from Pioneer High School, Waukomis, Oklahoma in May 1988; received Bachelor of Science degree in Mechanical Engineering from Oklahoma State University, Stillwater, Oklahoma in May 1993. Completed the requirements for the Master of Science degree with a major in Chemical Engineering at Oklahoma State University in May 1996.

Experience: Advanced Process Control Engineer for Conoco Inc.

Professional Memberships: American Society of Mechanical Engineers, American Institute of Chemical Engineers, Omega Chi Epsilon