Spring 2002

Process control of a laboratory combustor using neural networks

Thana Slanvetpan
New Jersey Institute of Technology

ABSTRACT

PROCESS CONTROL OF A LABORATORY COMBUSTOR USING NEURAL NETWORKS

by
Thana Slanvetpan

Active feedback and feedforward-feedback control systems based on static-trained feedforward multi-layer-perceptron (FMLP) neural networks were designed and demonstrated, by experiment and simulation, for selected species from a laboratory two-stage combustor. These virtual controllers functioned through a Visual Basic platform. A proportional neural network controller (PNNC) was developed for a monotonic control problem – the variation of outlet oxygen level with overall equivalence ratio (ϕ_o). The FMLP neural network maps the control variable to the manipulated variable. This information is in turn transferred to a proportional controller, through the variable control bias value. The proposed feedback control methodology is robust and effective to improve control performance of the conventional control system without drastic changes in the control structure. A detailed case study in which two clusters of FMLP neural networks were applied to a non-monotonic control problem – the variation of outlet nitric oxide level with first-stage equivalence ratio (ϕ_o) – was demonstrated. The two clusters were used in the feedforward-feedback control scheme. The key novelty is the functionalities of these two network clusters. The first cluster is a neural network-based model-predictive controller (NMPC). It identifies the process disturbance and adjusts the manipulated variables. The second cluster is a neural network-based Smith time-delay compensator (NSTC) and is used to reduce the impact of the long sampling/analysis lags in the process. Unlike other neural network controllers reported in the control field,

NMPC and NSTC are efficiently simple in terms of the network structure and training algorithm. With the pre-filtered steady-state training data, the neural networks converged rapidly. The network transient response was originally designed and enabled here using additional tools and mathematical functions in the Visual Basic program. The controller based on NMPC/NSTC showed a superior performance over the conventional proportional-integral-derivative (PID) controller. The control systems developed in this study are not limited to the combustion process. With sufficient steady-state training data, the proposed control systems can be applied to control applications in other engineering fields.

PROCESS CONTROL OF A LABORATORY COMBUSTOR USING NEURAL NETWORKS

by
Thana Slanvetpan

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Chemical Engineering

Department of Chemical Engineering

May 2002

APPROVAL PAGE

PROCESS CONTROL OF A LABORATORY COMBUSTOR USING NEURAL NETWORKS

Thana Slanvetpan

Dr. Robert B. Barat, Dissertation Advisor — Date
Associate Professor of Chemical Engineering, NJIT

Dr. Basil C. Baltzis, Committee Member — Date
Professor of Chemical Engineering, NJIT

Dr. Dennis L. Blackmore, Committee Member — Date
Professor of Mathematics, NJIT

Dr. Dana E. Knox, Committee Member — Date
Associate Professor of Chemical Engineering, NJIT

Dr. Norman W. Loney, Committee Member — Date
Associate Professor of Chemical Engineering, NJIT

Dr. John G. Stevens, Committee Member — Date
Professor of Mathematics, Montclair State University

BIOGRAPHICAL SKETCH

Author:	Thana Slanvetpan
Degree:	Doctor of Philosophy
Date:	May 2002

Undergraduate and Graduate Education:

- Doctor of Philosophy in Chemical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2002

- Master of Science in Engineering Management,
  New Jersey Institute of Technology, Newark, NJ, 2002

- Master of Science in Chemical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1997

- Bachelor of Science in Chemical Engineering,
  Chulalongkorn University, Bangkok, Thailand, 1995

Major: Chemical Engineering

Presentations and Publications:

Slanvetpan, T., Robert B. Barat, and John G. Stevens, "Process Control of a Laboratory Combustor Using Neural Networks," *Computers and Chemical Engineering* (submitted).

Brukh, R., T. Salem, T. Slanvetpan, R. B. Barat, and S. Mitra, "Process Modeling and On-Line Monitoring of Benzene and Other Species During the Two-Staged Combustion of Ethylene in Air," *Advances in Environmental Research* (accepted).

Slanvetpan, T., Robert B. Barat, and John G. Stevens, "Process Control of a Laboratory Combustor Using Neural Networks," *ICCIT 2001 Conference*, Montclair State University, Montclair, NJ, 2001.

Barat, R. B., T. Slanvetpan, and John G. Stevens, "Real-Time Monitoring and Feedback Control of Emissions from a Staged Combustor," *AIChE 2000 Annual Meeting*, Los Angeles, CA, 2000.

Slanvetpan, T. and Robert B. Barat, "Real-Time Monitoring and Feedback Control of Emissions from a Staged Combustor," *Uni-Tech Conference*, New Jersey Institute of Technology, Newark, NJ, 2000.

Slanvetpan, T., Robert B. Barat, John G. Stevens, and Arthur T. Poulos, "Demonstration of Feedback Control and Real-Time Monitoring of Organic Emissions from a Staged Combustor," *The Eastern States Combustion Institute*, George Washington University, Washington DC, 1999.

Slanvetpan, T., "Characteristics of Soot Particle in Electrocatalytic Reactor," *Master Thesis*, New Jersey Institute of Technology, 1997.

To my beloved parents

ACKNOWLEDGEMENT

I wish to express my deepest sincere gratitude to my dissertation advisor, Dr. Robert Barat, for his constant support, guidance, encouragement, and friendship throughout this study. Working with him has been one of the most precious experiences in my life. Appreciation is expressed to Dr. John Stevens for serving as a committee member and his guidance and enthusiasm in the field of artificial neural networks. I would like to also express my sincere thanks to Dr. Basil Baltzis, Dr. Dennis Blackmore, Dr. Dana Knox, and Dr. Norman Loney for their participation as members of the dissertation committee.

Through my working effort in the research lab, I would like to thank Dr. Joseph Bozzelli's research group for their kind assistance and support of the NO_x gas analyzer. I also send my thanks to Roman Brukh, Clint Brockway, and John Griscavage for their helpful assistance.

I wish to thank the Northeast Hazardous Substances Research Center and U.S. Environmental Protection Agency for their financial support of this research.

Lastly, I would like to thank my dearest family, especially my parents and my sister for their constant support and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Background

Minimization of both transient and steady state emissions through active process control of combustion systems such as waste incinerators, furnaces, turbines, automobile engines or power plants is an important research area. Classical controllers using the proportional-integral-derivative (PID) algorithm are still widely used. However, they can be challenged by process oscillations when large disturbances or setpoint changes are encountered. Another drawback is that tuning the PID can be time-consuming and requires a combination of operational experience and trial-and-error. The task becomes even more difficult for highly nonlinear processes especially in the presence of significant time delay, such as a sampling/analysis time lag.

To facilitate process control, a model-based approach is especially advantageous. An accurate process model can be used either as the basis for classical controller design methods or incorporated directly as the controller. Typically, the process model is based on conservation laws (mass, species, energy, and momentum). The model for a chemical process often turns out to be quite complex since most processes exhibit nonlinear characteristics and time-varying behavior.

A supplement or alternative to a traditional conservation law-based model is an approach using artificial neural networks (ANNs). The ability to learn a nonlinear governing relationship from a sample input/output data set enables the ANN to generate accurate nonlinear models for systems that can be difficult to model otherwise.

Furthermore, the ANN models are data-driven and computationally efficient, provided that their structure is not exceptionally complex. Therefore, they can be implemented practically in a model-based control approach. Neural networks hold great promise in modeling and control applications for complex dynamic processes in chemical engineering fields.

There are several approaches to incorporating ANNs in a model-based control structure. The neural networks used in most control schemes are feedforward multi-layer-perceptron (FMLP) architecture networks. In order to incorporate the process dynamics, lags, and other real factors into the process controller, one must carefully design a time-history window of the process inputs and outputs from a period of time equal to the delay in the process and then input them into the FMLP neural network (Bhat and McAvoy, 1990; Gomm et al., 1997; Nikravesh et al., 2000; Palancar et al., 1998; Syu and Chen, 1998; Tendulkar et al., 1998). This technique is possible when there are discrete-time training data available. Thus, the important issue of using the FMLP neural network for an active control problem is training-data acquisition. Typically, the model-based control structure prevents the network from operating independently from the actual process since the neural network model requires historical discrete-time data on both process inputs and outputs. As a result, network learning is often performed online in the presence of the real process. However, chemical processes are usually quite complex, so the online network weight updating (i.e. training) is often difficult and time-consuming. Therefore, in most cases, the neural networks are trained offline.

Recurrent neural networks offer a solution in dealing with a time-dependent control problem (Chovan et al., 1996; Palancar et al., 1998; Gomm et al., 1997).

Unlike the FMLP neural network, the recurrent network can develop an internal representation of time history through learning due to its internal feedback connections. Only current process inputs and outputs are required instead of a time history. As a result, the recurrent neural network can operate independently from the process and can be used in a generalized mode for offline simulation and development. Although the process dynamics and delay time can be mapped to a fully recurrent neural network, the network-training algorithm convergences slowly since it requires the use of all weights and feedback activities in the network for a weight updating process (Chovan et al., 1996). In addition, network-predicting performance can deteriorate with time since the recurrent neural network operates independently from the process. Hence, any error in the network output is also fed back into the network and can accumulate with time (Gomm et al., 1997).

1.2 Objectives

Although there have been numerous recent studies in modeling and control applications based on neural networks, the application of the neural networks in the case of emission control are very scarce. In this study, online gas analyzers are combined with computerized process control for a two-stage bench combustor. The objective is to develop and demonstrate, by experiment and simulation, control systems based on static back-propagation trained neural networks. The originality of this research can be mapped into three parts to address three different functionalities of the neural networks.

First, a simple feedforward neural network is used in a proportional neural network configuration. Based on the information it receives, the trained neural network serves as an intelligent tool to update the proportional controller's bias value. A comparison of the open-loop and some closed-loop runs is presented to illustrate the effectiveness of applying the neural network to the simple proportional controller.

Secondly, a model-based control configuration, consisting of two clusters of trained neural networks, is demonstrated. The objective here is to present the basic concept of using neural networks in a model-predictive control configuration for a highly nonlinear system with significant sampling/analysis time delay where the conventional PID controller is difficult to tune and has a potential to fail. The originality of the work here lies in the structure and the functionalities of the two clusters of FMLP neural networks. The first cluster serves as the process controller while the second serves as the time-delay compensator.

Lastly, the neural network is used to model the laboratory-scale combustion process. The trained neural network model for the combustor is embedded in the simulator interface constructed in the Visual Basic 6.0 environment. Although the neural network is trained by a steady state historical database, the time-dependent response of the real combustion system is enabled by the built-in tool and mathematical functions in the Visual Basic program. The simulation data are validated by results from existing experimental runs and conventional modeling results. Once equipped with the selected control application, the simulator serves as an actual combustion process replacement, and provides a more flexible and more reliable platform to study process control. Several of simulation runs under various control applications are made to gather more information about the process behavior and gain more insight into how to optimize the real process operation.

CHAPTER 2

LITERATURE SURVEY

2.1 Process Dynamics and Control

Chemical engineers are quite familiar with the virtues of feedback or feedforward process control, which allows continuous chemical processes to operate with little human intervention. Recently, the performance requirements for process plants as well as tough environmental and safety regulations have become increasingly difficult to satisfy. In order to meet plant quality standards and environmental regulations, modern chemical plants are usually equipped with many controllers.

Since the proportional-integral-derivative (PID) controller found widespread use in the process industries, great resistance often occurs to changing to other control methodologies in practical situations. The main reasons are the simplicity, robustness, and successful applications provided by PID-based control approaches (Hoskins and Himmelblau, 1992; Olsson et al., 2001; Seborg et al., 1989; Stephanopoulos, 1984; Martins and Coelho, 2000; Shu and Pi, 2000).

Olsson et al. (2001) presented a strategy for closed-loop control of a multi cylinder turbo-charged homogeneous charge compression ignition (HCCI) engine using PID controllers. The aim of the study was to demonstrate a functioning control system for optimizing a dual fuel HCCI engine by manipulating fuel mixing ratio and fuel flow rate. Although, the PID yielded promising control results in most cases, it was suggested that the PID settings had to be set conservatively in some ranges of operations to avoid process instabilities.

The task of tuning the PID becomes more difficult for highly nonlinear processes like combustion. In addition, time delay between sensor measurements and control computation and between control computation and final control element actuation can be a limiting factor when trying to achieve a fast process control response through the use of the PID.

To facilitate process control and overcome the problems associated with the traditional PID controller, a model-based control technique is an alternative approach that has received widespread attention. In the last decade, model-predictive control formalisms that employ an explicit model to predict the future process outputs and calculate optimal control movements have been extensively studied and used to control complex processes such as the nonlinear processes in chemical engineering fields.

Typically, a process model is based on conservation laws (mass, species, energy, and momentum). The model for a chemical process often turns out to be quite complex since most processes exhibit nonlinear characteristics and time-varying behavior. To simplify the problem, the control formalisms frequently utilize linearized models and some simplified assumptions even when systems behave nonlinearly. In the event that the non-linearity is not severe, the linearized approaches can provide adequate performance.

In the case where the process is a highly nonlinear one, or is forced from the region where the linearized model is acceptable, the linear controller often shows significant deterioration in control performance or in some cases fails to keep up with perturbations or process shifts. The situation becomes worse in the presence of process lags since it is more difficult to tune the linearized model-based controller to compensate for the lags. This traditional modeling approach can become vulnerable to modeling errors and thus, ultimately leads to control performance deterioration. In addition, the mathematical model representation for a chemical process often turns out to be too complex for practical implementation in a nonlinear model based controller. In such cases, an attractive alternative is to generate an empirical relationship based on input-output data.

2.2 Artificial Neural Networks

Neural computing is one of the fastest growing areas of artificial intelligence (AI). The ability to learn and generalize a nonlinear governing relationship from a sample input-output data set enables artificial neural networks (ANNs) to generate accurate nonlinear process models for systems that can be difficult to model otherwise. As a result, ANN holds great promise in modeling and control applications for complex dynamic processes in many engineering fields. It has been studied intensively with special regard to engineering applications.

Based on the nonlinear mapping capabilities, problems like identification, simulation, and control of chemical processes have been widely studied and solved by neural network methodologies. Allen et al. (1993) applied a neural network to process, identify, and analyze the resultant emission patterns in a flame image from a utility boiler. Booth et al. (1998) developed a neural-network-based program simulator to empirically model the nonlinear relationship between the operating modes, load conditions, and the emission NO_x from commercial boilers. Bhat and McAvoy (1990) used a back-propagation neural network for learning the dynamic model from plant input-output data, and outlined a possible scheme to use the neural network in place of a traditional model-approach control structure. Tendulkar et al. (1998) developed ANN-based nonlinear process identification and model-predictive control strategies for the phenol hydroxylation process in a pilot-scale fixed-bed reactor system. Nikrawesh et al. (2000) presented a model predictive control strategy, which used a neural network to model the process, and then applied the mathematical inverse of the process model as the controller. Chovan et al. (1996) demonstrated the concept of using a recurrent neural network for solving dynamic control problems.

Generally, neural networks are characterized by three basic parts: the network topology, the computational characteristic, and the training rule. Neural networks can be divided into two main classes by their topologies: feedforward networks and recurrent networks. Figure 2.1 shows a schematic of the feedforward multi-layer-perceptron (FMLP) topology, which is the most widely used and applied in this study. The network in Figure 2.1 has three layers, the left column is an input layer, the middle column is a hidden layer, and the right column is an output layer. Although the FMLP can have more than one hidden layer, it has been proven and is widely accepted that a single hidden layer is sufficient for most cases (Bhat and McAvoy, 1990; Chovan et al., 1996; Gomm et al., 1997; Martins and Coelho, 2000; Nascimento et al., 2000; Shu and Pi, 2000; Tendulkar et al., 1998).

The network consists of processing elements called neurons (nodes) and information flow channels between the neurons called interconnections. Each neuron carries out a local computation, which converts inputs to the neuron to an output. This computation is based on the activation function assigned to each neuron. The information flows in a forward direction, from the input layer to the hidden layer, and then the output layer. The FMLP neural network is useful for steady-state modeling. Unlike the recurrent network (Figure 2.2), the outputs of the FMLP network are not allowed to be included as inputs to any of the network's neurons. A more detailed description of the FMLP network computation can be found in Chapter 4.

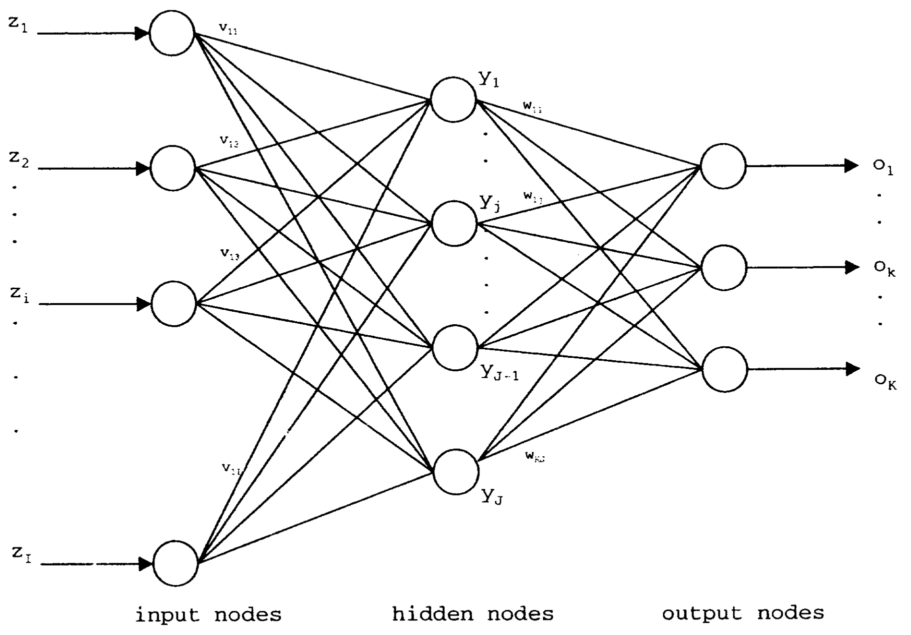


Figure 2.1 FMLP neural network architecture.

The final part, which is the key to most neural networks, is the training rule. Training the network means finding a set of parameters (interconnection weights) that produce the desired behavior. Training methods are generally divided into two classes: supervised and unsupervised trainings. The back-propagation trained networks used in this study are supervised networks. It means the networks need the input values and the corresponding desired output values to learn. At present, the supervised back-propagation has been applied to a wide variety of practical problem. It has proven very successful in its ability to model nonlinear relationships (Allen et al., 1990; Bhat and McAvoy, 1990; Gomm et al., 1997; Hernández and Arkun, 1992; Miller and Lemieux, 1998; Nascimento et al., 2000; Nikravesh et al., 2000; Palancar et al., 1998; Reinschmidt and Ling, 1994; Shu and Pi, 2000; Syu and Chen, 1998; Tendulkar et al., 1998). Unlike the supervised learning rule, unsupervised training systems develop their own rules by extracting information from examples without a complete set of the input-desired output pair. Only the supervised back-propagation training system is discussed and implemented in this study (Chapter 4). Detailed descriptions of the supervised and unsupervised training can be found elsewhere (Zurada, 1992).

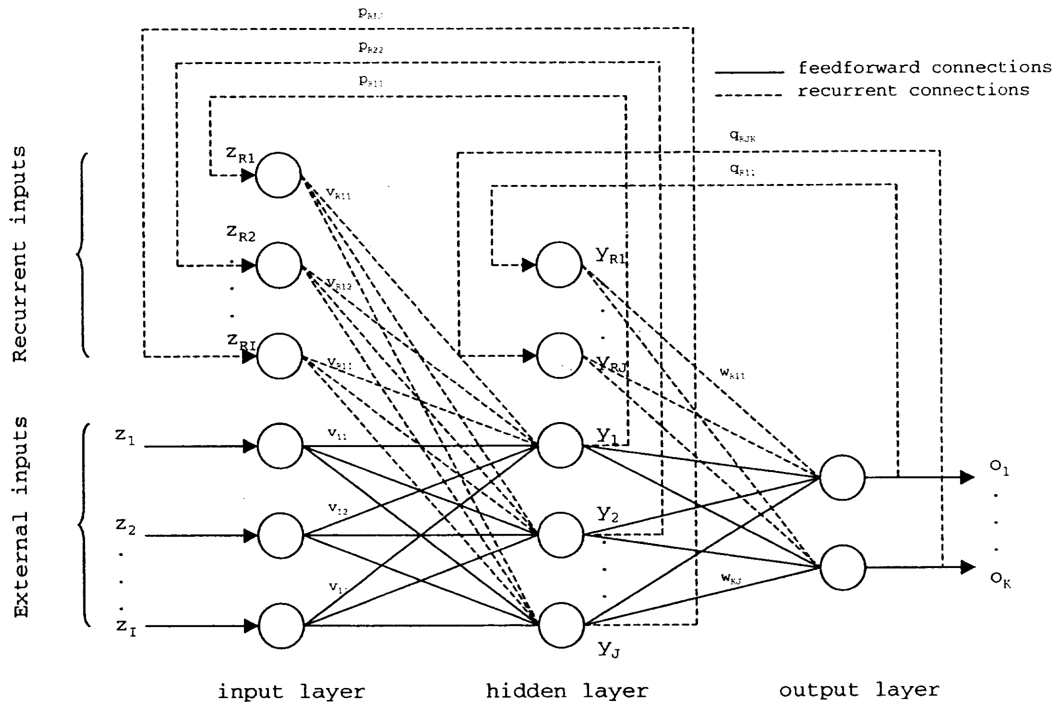


Figure 2.2 Recurrent neural network architecture.

2.2.1 Process Modeling Using Neural Networks

The use of ANNs for model development in recent years has overcome many limitations and problems associated with the traditional modeling methods. The use of the ANNs becomes even more attractive in the chemical engineering areas because of the non-linearity present in most chemical processes and a potential to implement the ANNs in a model-based control approach.

The simplest form when applying the feedforward neural network in modeling applications is to use it as a steady-state forward process model wherein the process input values serve as the network inputs. The network statically predicts as its output the process state based on the information it receives. Since the neural network can be statically trained by the steady-state training data prior to using it, the network usually converges rapidly during the offline training process.

Reinschmidt and Ling (1994) developed and demonstrated the use of feedforward neural networks to model the relationship of NO_x emissions from the utility boiler to various parameters. Their objective was to generate a computer representation of the response surface of NO_x production as a function of selected control variables. The network input layer consisted of 21 nodes, which represent controllable input variables (e.g. O_2 rate, coal feed rate, auxiliary air, and etc.) and load to the actual process. There are two hidden layers; each employs a sigmoid function as an activation function. The measured NO_x emission is the only network output. The bias signals were added to all neurons in the hidden layers only. The number of neurons used in the two hidden layers was determined by trial-and-error. All training and testing data were collected from a series of previous tests on an operating utility boiler at various loads and conditions. They were randomly partitioned into two sets: a training set and a testing set. Each data point has 22 data elements, i.e. 21 of them are used as inputs (e.g. controllable input variables), the other one as output (NO_x). Although many modified back-propagation learning algorithms were tried, including back-propagation with momentum, adaptive learning rate, etc., it was concluded that the basic back-propagation algorithm was sufficiently robust and led to an acceptable learning error reasonably fast after validating the neural network model against the 20 randomly chosen testing set.

A simple FMLP neural network in conjunction with some programming techniques can be used to simulate dynamic process responses. Booth et al. (1998) used the static-trained feedforward neural network to empirically model a nonlinear relationship between the operating modes, load conditions, and the emission NO_x from commercial boilers. Additional programs were utilized to incorporate process and equipment dynamics and instrumentation response time into the ANN-based model. The objective was to provide gradual transitions for the setpoint and bias adjustments as the model responds to changes in operating conditions or equipment performance. The ANN-based process simulator was used in many process control studies in trying to reduce NO_x emissions from commercially operating utility boilers (Booth et al., 1998; Radl and Roland, 1995).

2.2.2 Dynamic Process Control and Neural Networks

The links between neural networks, dynamic process models, and process control are provided by the concept of model-based control wherein the neural network is used in place of the traditional process model based on conservation equations. The neural network model-based control approach has become increasingly advantageous in chemical engineering for two major reasons. First, many chemical processes exhibit nonlinear behavior, where relationships between the controlled variables and the manipulated variables are complex. Second, the active control of most chemical processes is often complicated by the process dynamics, lags, and other real factors associated with process fluid, and time delays in sampling and species analyses. In trying to incorporate these time-varying behaviors into a classical control theory, the traditional Smith time-delay compensation method (Seborg et al., 1989; Stephanopoulos, 1984) can be used to construct controllers when the transfer function of the system is known. However the transfer functions of most chemical processes are too complex for practical implementations. Therefore, recent developments in the applications of nonlinear models based on neural networks are entering successfully into the fields of model-based control.

When using FMLP networks to model process dynamics and implementing them in a model-based control structure, a careful design of the history time-window of process inputs and outputs is necessary. The inputs of the network are typically formed by using past, present, and future process inputs and corresponding process outputs. A detailed assignment of inputs and outputs is determined according to the requirements of each individual control problem.

There are two sub-classes when applying the neural network into a model-based control structure: a direct control scheme and an indirect control scheme. In the direct control scheme, wherein the feedforward neural network serves as a controller, the neural network is trained to map the process inputs, outputs, and setpoint into the control action.

This kind of solution can be used to train neural network controllers for tasks that are not too complicated and usually can be successfully solved by human operators. Such a back-propagation feedforward neural network was presented by Syu and Chen (1998) for online control of a wastewater treatment system.

In coping with this time-dependent problem, the data from previous sampling times become relevant and have to be included into the neural network input nodes. The network structure, proposed by Syu and Chen (1998), is shown in Figure 2.3, where t, t-1, t-2, and t-3 refer to the process states in the present and past time steps. The network input nodes are present and past process outputs (e.g. measured chemical oxygen demand COD) and past process inputs (manipulated variable, e.g. amount of added reagent H_2O_2). The network output node is the predicted control action, the amount of reagent H_2O_2 (t) to be added.

Regarding the dynamic characteristics of the system, a moving window node of supplying data to the neural network for learning was used. The neural network was trained in a dynamic mode during the online operation when the control action was not called up. For each learning cycle, a fixed number of training data was provided to the network. That is, once new data were added as time moved on, the oldest data would be removed from the training set. When the network was switched to be a controller, the present system output COD (t) was replaced by the desired process setpoint. Such neural network adaptive control is notable for its ability in carrying out the online control of the continuous wastewater treatment system successfully as long as the network parameters and operation conditions are properly chosen.

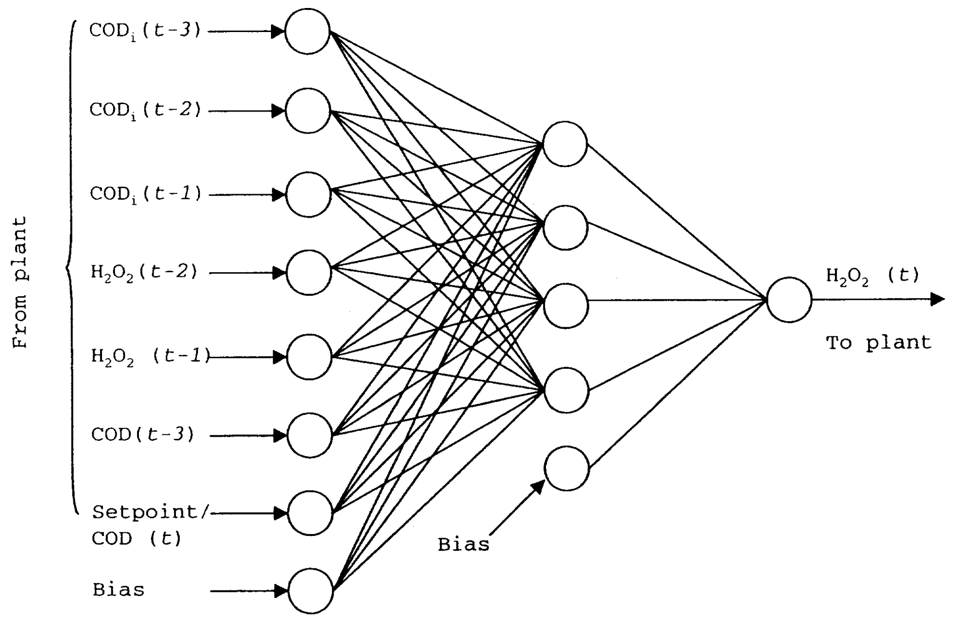


Figure 2.3 Neural network in a direct control scheme (Syu and Chen, 1998).

Another model-based control approach is to use the neural network as a process model in an indirect control scheme. The indirect control method uses an explicit process model to predict the future process outputs from past and present inputs and outputs. In this approach, the inverse of the model at each sampling time must be calculated via an optimization routine to calculate an optimal control action. Many computing techniques have been studied and incorporated into the indirect control scheme to calculate the control output. The concept of employing neural networks in the indirect control scheme has been successfully demonstrated by many researchers (Bhat and McAvoy, 1990; Braak et al., 1998; Evenson et al., 1998; Gomm et al., 1997; Johnson, 1998; Nascimento et al., 2000; Nikravesh et al., 2000; Palancar et al., 1998; Tendulkar et al., 1998).

Bhat and McAvoy (1990) demonstrated the use of a back-propagation neural network for dynamic modeling (BDM) and control of a nonlinear chemical process – pH in a continuous-stired-tank-reactor (CSTR). The CSTR has two input streams, one containing a base reagent (NaOH) and the other containing an acid reagent (HAC). There are 15, 5, and 5 neurons in the network's input, hidden, and output layers, respectively. The network inputs were formed from past and present values of process inputs and outputs, and future values of the process inputs (a moving time-window of process inputs and process outputs). It was noted that the future values of the process inputs were known during the network training process since the training set was developed from the historical time-dependent database. The network predicted the process output (pH) one to five steps into the future. After several cycles through the training process, the BDM converged and gave excellent predictions. Compared to a traditional modeling method (e.g. autoregressive and moving averages), the neural network technique showed its ability to learn more of the nonlinear characteristics of the process. Once trained, the network can be used in a model-based control structure. However, as demonstrated, the neural network only predicted the process output, but not the manipulated variable. Therefore, an additional control element is needed in the indirect ANN-based control scheme for computing an optimal control action, based on the neural network model outputs and the future desired process setpoint.

The use of a process optimizer to calculate the manipulated variables was proposed by many authors (Bhat and McAvoy, 1990; Braake et al., 1998; Nascimento et al., 2000; Tendulkar et al., 1998). Figure 2.4 presents a schematic diagram showing how the neural model and the process optimizer are constructed and used in the indirect model-based control structure. Typically, the process optimizer performs an iterative computation of an inverse problem to calculate the manipulated variable based on the neural network outputs and future desired setpoint. The optimizer suggested by Tendulkar et al. (1998) uses the steepest descent method for iteratively adjusting the neural network inputs representing the manipulated variables until the error between the network-computed process outputs and the setpoint falls below a pre-assigned small threshold. After convergence, the steepest descent search is terminated and the converged value of the manipulated variable is applied to the process. This procedure is repeatedly executed at every sampling instant. Since the neural network model is frequently complex, the iterative computation of the manipulated variable through the process optimizer often turns out to be difficult and time consuming. The task becomes too difficult to implement for highly nonlinear processes.

Gomm et al. (1997) developed process models and predictive controllers using FMLP neural networks. The capabilities of the neural network system were demonstrated in practical applications to modeling and control of a nonlinear process. The FMLP neural network was trained via back-propagation to model a nonlinear process. The trained neural network was implemented in conjunction with a nonlinear process optimizer in a model predictive control scheme. Improved setpoint tracking over the traditional tuned PI controller was achieved over a nonlinear operating range with significant reductions in the required movement of the process input.

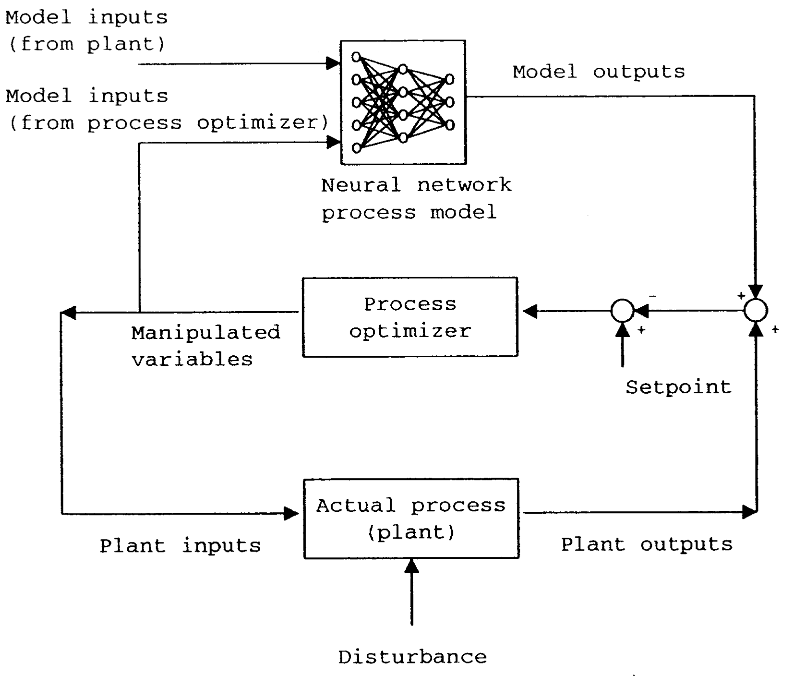


Figure 2.4 A FMLP neural network in an indirect control scheme.

Martins and Coelho (2000) reported a control methodology based on PID control algorithms conjugated with the FMLP neural network. The objective was to apply the neural network as a complementary tool to improve the PID-based control performance. The neural network was used to predict future values of the controlled variable as a function of process measurements and the process setpoint. This information was then incorporated in a conventional PID control system structure.

Basically, the techniques used to compute the control action in an indirect control scheme involve an iterative inverse adjustment of the neural network outputs and the actual process outputs. Such computation is a time-consuming process especially for a highly nonlinear process. To avoid time-consuming calculations of the inverse problems in the indirect control structure, a neural network of the process inverse model can be used in place of the process optimizer or some types of controllers described above. For example, referring to Bhat and McAvoy (1990), one could switch the future process outputs with the future process inputs and develop an inverse system model. That is, the inverse neural network model predicts what flow rates of NaOH (process inputs) are necessary to achieve a desired pH (process output). Once the inverse model is available, it can be used in many schemes of a model-based control approach such as an internal model control (IMC) structure as shown in Figure 2.5.

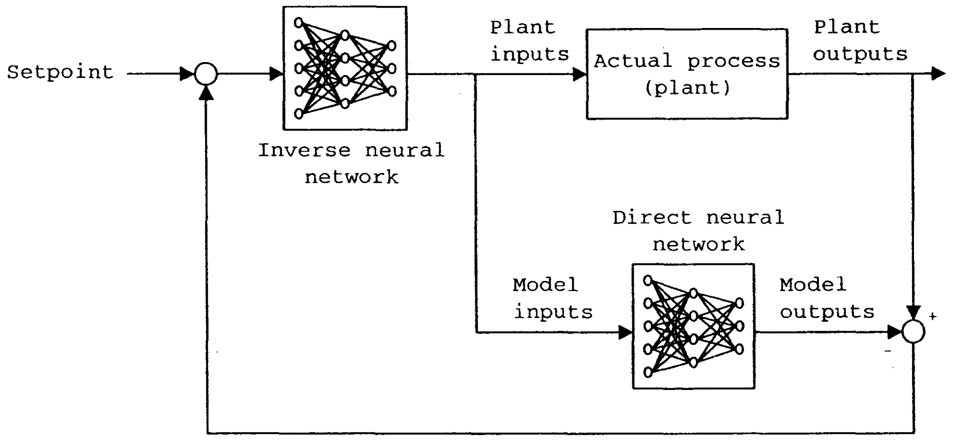


Figure 2.5 Neural networks in a model-predictive control scheme.

The strategies of using and combining the neural network model and its mathematical inverse for process control applications are diverse and have been investigated and demonstrated by several researchers (Palancar et al., 1998, Chovan et al., 1996, and Nikravesh et al., 2000).

Following the nomenclature for neural network study used in Bhat and McAvoy (1990) and the field of process control, the neural network process model usually refers to a direct network that predicts the outputs of the process based on the values of the input variables. The mathematical inverse of the process model refers to an inverse network. The inverse network predicts future process input variables based on current and past values of the process inputs, outputs, and desired outputs.

Palancar et al. (1998) designed a control system based on a combination of two artificial neural networks for a pH control process in a CSTR. The purpose of the controller was to maintain the pH of the exit stream of a neutralization tank as close as possible to a given setpoint value. The proposed control configuration is shown in Figure 2.6. The two neural networks are feedforward multi-layer-perceptron (FMLP) types with one hidden layer. Each network employs the sigmoid function as an activation function. The input and output values used by the neural networks were normalized into the range of 0 to 1. To incorporate the time delay in the system, the researchers included past data of a number of sampling periods corresponding to the system lag into the networks inputs and outputs.

The first neural network is a plant model that predicts future pH values (Y_k, Y_{k+1}, ..., Y_{k+5}) from past and present values of pH (Y_{k-3}, Y_{k-2}, ..., Y_k) and valve stem position (Xv_{k-3}, Xv_{k-2}, ..., Xv_k) and future values of valve stem position (Xv_{k+1}, Xv_{k+2}, ..., Xv_{k+5}). Note that the valve stem position controls the flow rate of a manipulated variable – the alkaline stream. The second neural network is a plant inverse model that provides a control action by calculating the future values of valve stem position (Xv_{k+1}, Xv_{k+2}, ..., Xv_{k+5}) from present and past values of pH (Y_{k-3}, Y_{k-2}, ..., Y_k) and the valve stem position (Xv_{k-3}, Xv_{k-2}, ..., Xv_k) and future values of pH setpoint (Ysp_{k+1}, Ysp_{k+2}, ..., Ysp_{k+5}).

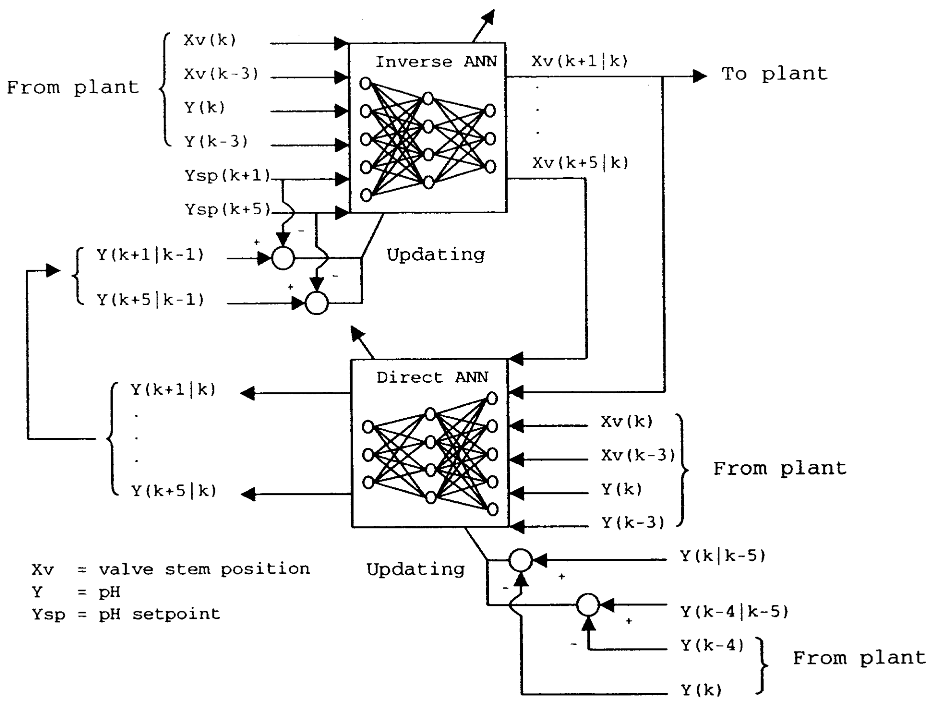


Figure 2.6 ANN-based control configuration (Palancar et al., 1998).

Back-propagation was implemented for the network training process. The direct network was first trained by using a historical discrete-time database (pH vs. time). The online weight updating of the direct network was also made once at each sampling time during the closed-loop operation. It served to adapt better networks for situations rather different from the ones used during the first rough offline training. Because of the structure of the control loop, the inverse network had to be always operated in a learning mode (online training) during the closed-loop operation. Although it was not clearly mentioned in the article, it would be assumed that the historical training data were not sufficient to span all the possible operation ranges of the neutralization tank. The experimental results show that the control efficiency is good only for perturbations that do not involve strong buffering changes (e.g. small perturbations of setpoint, flow and concentrations of acids). Under these conditions, the neural networks were able to adapt their weights online and effectively responded to the perturbations they had never seen before. Nevertheless, the neural networks could not learn adequately under frequent and large perturbations. Allowing for these limitations, the proposed controller could be useful only for processes in which the expected perturbations and buffering changes were small.

As demonstrated, the neural networks used in most control schemes are multi-layer-perceptron feedforward (FMLP). The most frequently used approach to incorporate the process dynamics, process lags, and other real factors into the process model is to carefully design a time-history window of the process inputs and outputs from a period of time equal to the delay in the process and then input them into the feedforward neural network (Bhat and McAvoy, 1990; Gomm et al., 1997; Nikravesh et al., 2000; Palancar et al., 1998; Syu and Chen, 1998; Tendulkar et al., 1998). This technique is possible when there are discrete-time training data available. Thus, the important issue of using the feedforward neural network for an active control problem is training-data acquisition.

Ideally, the training set should include enough training vectors to adequately describe the modeled system behavior. However, in the indirect control scheme, the output error of the neural network model is used in a standard back-propagation algorithm to pass the error back to the controller output. This indirect control structure inhibits the network from operating independently from the actual process since it requires discrete-time data of both process inputs and outputs to be used for the neural network inputs. As a result, network learning is essentially performed online in the presence of the real process. Since most chemical processes are quite complex, such online weight updating is often difficult and time-consuming. Therefore, in most cases, the neural networks cannot be adequately trained during the online learning process.

Recurrent neural networks offer a solution to cope with the difficulties and limitations associated with the feedforward neural network especially when dealing with a time-dependent control problem (Chovan et al., 1996; Palancar et al., 1998; Gomm et al., 1997). Unlike the feedforward neural network, the recurrent network can develop an internal representation of time history through learning due to their internal feedback connections. Only current process inputs and outputs are required instead of their time-history. As a result, the recurrent neural network can operate independently from the process and can be used in a generalized mode for offline simulation and development. Although the process dynamics and delay time can be mapped to a fully recurrent neural network as demonstrated by Chovan et al. (1996), the network-training algorithm convergences slowly since it requires the use of all weights and feedback activities in the network for a weight updating process. Another drawback of using the trained recurrent neural network, suggested by Gomm et al. (1997), is deterioration in the prediction accuracy as time advances. The reason for this is that, in the recurrent operating mode, no corrections are made to the network predictions – that is, the recurrent neural network operates independently from the process. Hence, any error in the network output is also fed back into the network and can accumulate with time.

CHAPTER 3

EXPERIMENTAL FACILITY

The overall experimental setup schematic is shown in Figure 3.1. This overall setup can be divided into three different parts: a two-staged combustion system, a gas sampling and analysis system, and a control system.

3.1 A Two-Stage Combustion System

An atmospheric pressure two-stage reactor presented in Figure 3.2 serves as the combustion facility. It has been well characterized elsewhere (Mao, 1995; Mao and Barat, 1996).

The first stage (primary zone) of the combustor is a 250-cm^3 well-mixed zone that can be modeled as a perfectly stirred reactor (PSR) under most conditions. It is custom cast from refractory alumina cement. An overhead cross section of the first stage is shown in Figure 3.3. The feed stream enters the first stage from an outer stainless steel circular manifold through a series of 32 jets positioned at the circumference of a torus. Each jet is angled 20° off the radius of the stainless steel circular manifold. The feed stream consists of metered C_2H_4, NH_3, air, and diluent N_2.

Polymer-grade purity (99.9%) ethylene (C_2H_4) is used as the primary fuel in this study. Two ethylene cylinders are connected in parallel to the feed line. Ethylene at cylinder pressure flows through a stainless steel coil bathed in hot water before the regulator in order to compensate for Joule-Thomson cooling during the significant pressure drop across the regulator. The hot water in the bath is continuously heated and controlled at 70 to 80 °C. After preheating and regulating to 40 psig, the ethylene gas temperature measured at the regulator outlet is typically 20 °C, which is a desirable feed temperature.

Anhydrous 99.99% purity ammonia (NH_3), which serves as the secondary fuel, is used as a model dopant to simulate fuel-bound nitrogen, which produces NO. The ammonia is regulated to 40 psig prior to its rotameter.

An in-house compressor provides pressurized air at 120 psig for the oxidant in both first and secondary stages (PSR and PFR). A knockout filter is mounted on the combustion air line to remove any oil particles and saturated moisture. The air is regulated to 40 psig, for both primary and secondary stages, prior to the rotameters. The primary air is delivered via two rotameters connected in parallel (Figure 3.1): one with a large flow, manual valve, and the other with a small flow, electronic valve. The total primary air flow rate is the combination of the flow from the two rotameters. The large flow, manual valve is very essential during the system start-up, shutdown, and changeover (fuel-lean to fuel-rich and vice versa) procedure. The secondary air is injected into the second stage through a ceramic tube located at the base of the second stage. One electronic control valve is installed for computerized control adjustment of the secondary air flow rate. The three-way valve, located downstream of the secondary air rotameter, provides a manual control option.

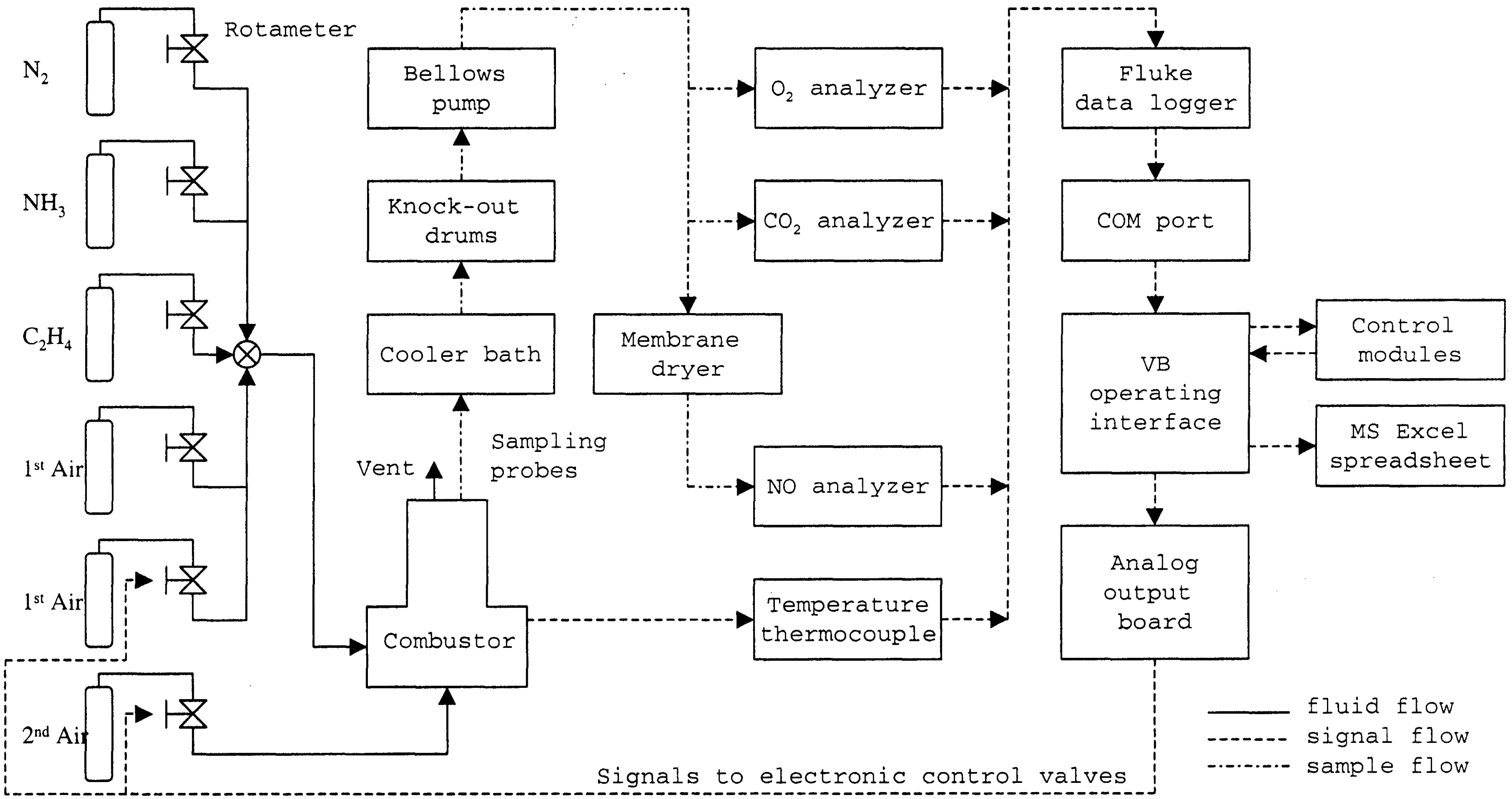


Figure 3.1 Overall experimental schematic.

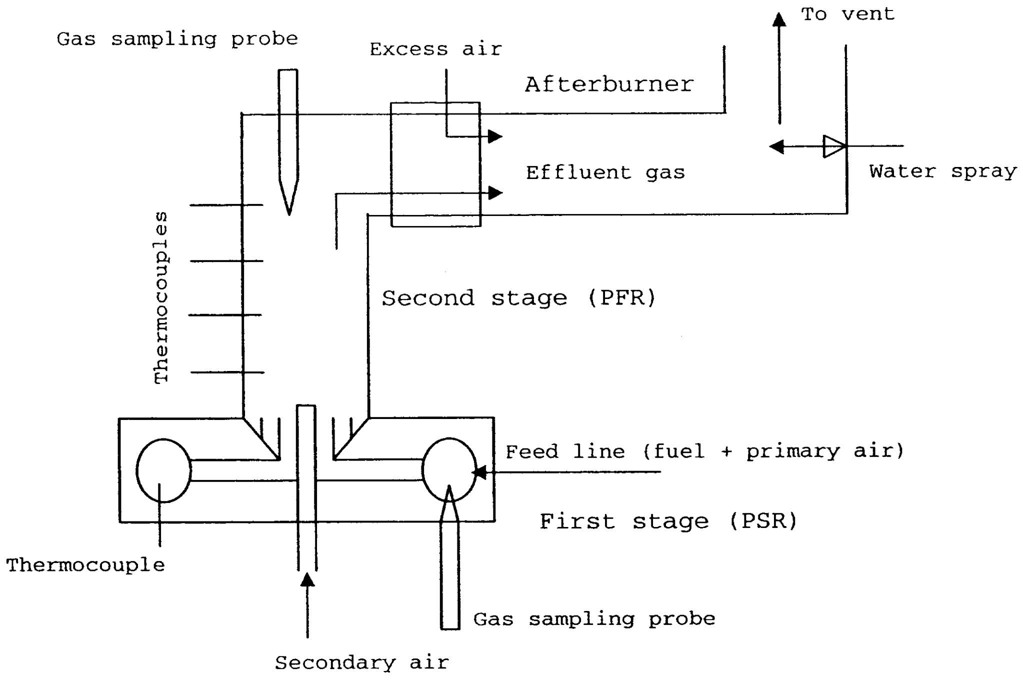


Figure 3.2 Two-stage combustion facility.

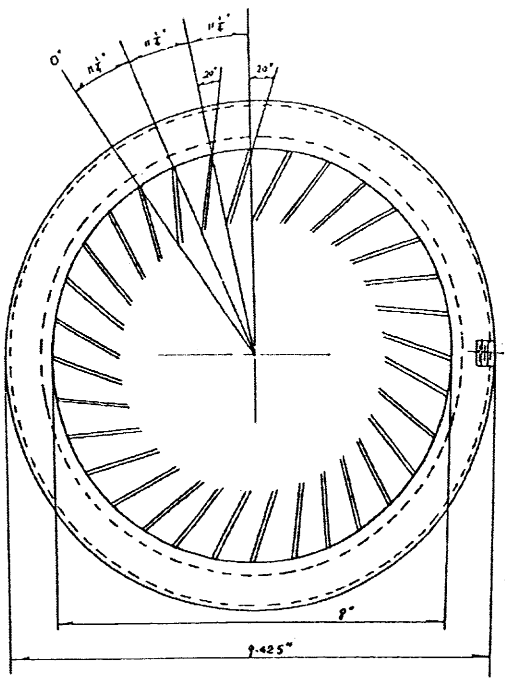


Figure 3.3 Top cross section of the first stage.

The fuel/air mixture composition is characterized by the equivalence ratio ϕ given by:

$$\phi = \frac{\left(\dfrac{Fuel}{Air}\right)_{actual}}{\left(\dfrac{Fuel}{Air}\right)_{stoichiometric}} \tag{3.1}$$

where $Fuel$ = volumetric or molar flow rate of C_2H_4

Air = volumetric or molar flow rate of primary air

Nitrogen gas from a high pressure refrigerated dewar is used for diluting the fuel-rich feed and controlling the reactor temperature. Additional nitrogen gas does not change the feed equivalence ratio, but does change the feed and product concentrations and reactor temperature. The diluent nitrogen gas from the dewar is regulated to 80 psig at room temperature prior to its rotameter.

Residence times in the primary zone are 5-10 milliseconds based on mass flow rate and actual temperature. A water-cooled extractive sampling probe and one uncoated type R micro-thermocouple are inserted into the first stage from the bottom for sample gas withdrawal and temperature measurement, respectively.

The hot effluent from the first stage passes over a flow straightener and then enters the second stage – a linear flow zone (50 mm in inner diameter × 330 mm long) that can be modeled as a plug flow reactor (PFR). The second stage reactor is made from a pre-cast alumina cylinder. The secondary air can be introduced through a ceramic tube installed at the base of the secondary zone. Second stage residence times are on the order of 20 milliseconds at combustion temperature. A variable position water-cooled probe is inserted axially into this secondary zone from the top for sample gas withdrawal.

Four type R micro-thermocouples are radially inserted into this linear flow zone approximately 6 mm from the wall for temperature measurement.

The exit gases from the secondary zone mix with large volumes of axially injected excess air in a bluff body-stabilized afterburner. The exhaust gases are then cooled down by water spray before venting.

3.2 Gas Sampling and Analysis System

A probe sampling system is shown in Figure 3.4. The sample gas stream passes successively through a chilled bath and two droplet knockout drums to reduce the water vapor mole fraction, before being drawn through a bellows pump that provides the suction for sample withdrawal and the pressure head for sample gas passage to the online continuous emission monitors.

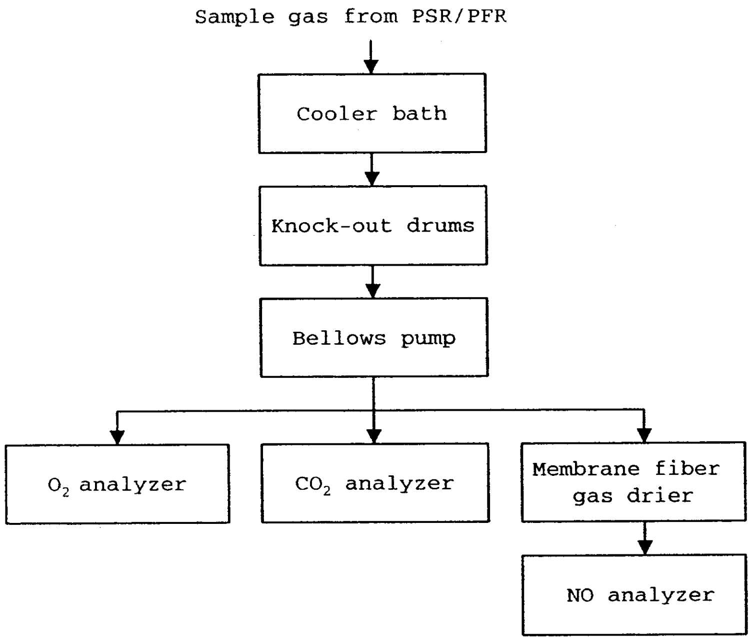


Figure 3.4 Gas sampling and analysis system.

The sample gases are split, with portions flowing to a paramagnetic O_2 analyzer and a non-dispersive infrared CO_2 analyzer. The remainder is dried using a membrane fiber bundle gas drier prior to passing to a chemiluminescence NO_x analyzer.

3.2.1 Paramagnetic O_2 Analyzer

The oxygen level is determined by a Beckman model 755 O_2 analyzer using a paramagnetic method based on the ability of the oxygen molecule to become a temporary magnet when placed in a magnetic field. Measurement is accomplished by a torque force balance system. The force produced in this system is proportional to the sample oxygen content. Variables that can influence the measurement precision are sample gas pressure and operating temperature. Therefore, it is necessary to calibrate the instrument at the same pressure as the actual sample, and to warm up and maintain the analyzer temperature at 60 °C.

3.2.2 Non-Dispersive Infrared CO_2 Analyzer

The Beckman model 880A is a non-dispersive infrared CO_2 analyzer. Within the analyzer, two equal energy infrared beams are directed through two parallel optical cells: a flow-through sample cell and a static reference cell. The detector continuously measures the difference in the amount of infrared energy absorbed within each of the two cells. This difference represents the concentration of CO_2 in the sample stream.

3.2.3 Chemiluminescence NO-NO_2-NOx Analyzer

The model 42C high-level chemiluminescence NO-NO_2-NO_x analyzer from Thermo Environmental Instrument, Inc. is used to monitor the NO concentration in this study. The model 42C uses the principle that nitric oxide (NO) and ozone (O_3) react to produce a characteristic luminescence with intensity linearly proportional to the NO concentration. Infrared light emission results when electronically excited NO_2^* molecules decay to lower energy states. The reactions involved are:

$$NO + O_3 \rightarrow NO_2^* + O_2 \tag{3.2}$$

$$NO_2^* \rightarrow NO + h\nu \tag{3.3}$$

where h = Planck's constant

ν = frequency, Hz.

From Equations 3.2 and 3.3, as NO (contained in the sample) and O_3 (produced by an ozone generator in the analyzer) are mixed in a small reaction chamber, the chemiluminscent reaction produces light emission that is directly proportional to the concentration of NO contained in the sample. This emission is measured by a detector and converted into an electric signal.

3.3 Control System

A schematic diagram for the control system is shown in Figure 3.5. The application of process control for the combustor involved several components and tasks. Two electronically proportional solenoid valves, one for primary air and the other for secondary air, are the final control elements.

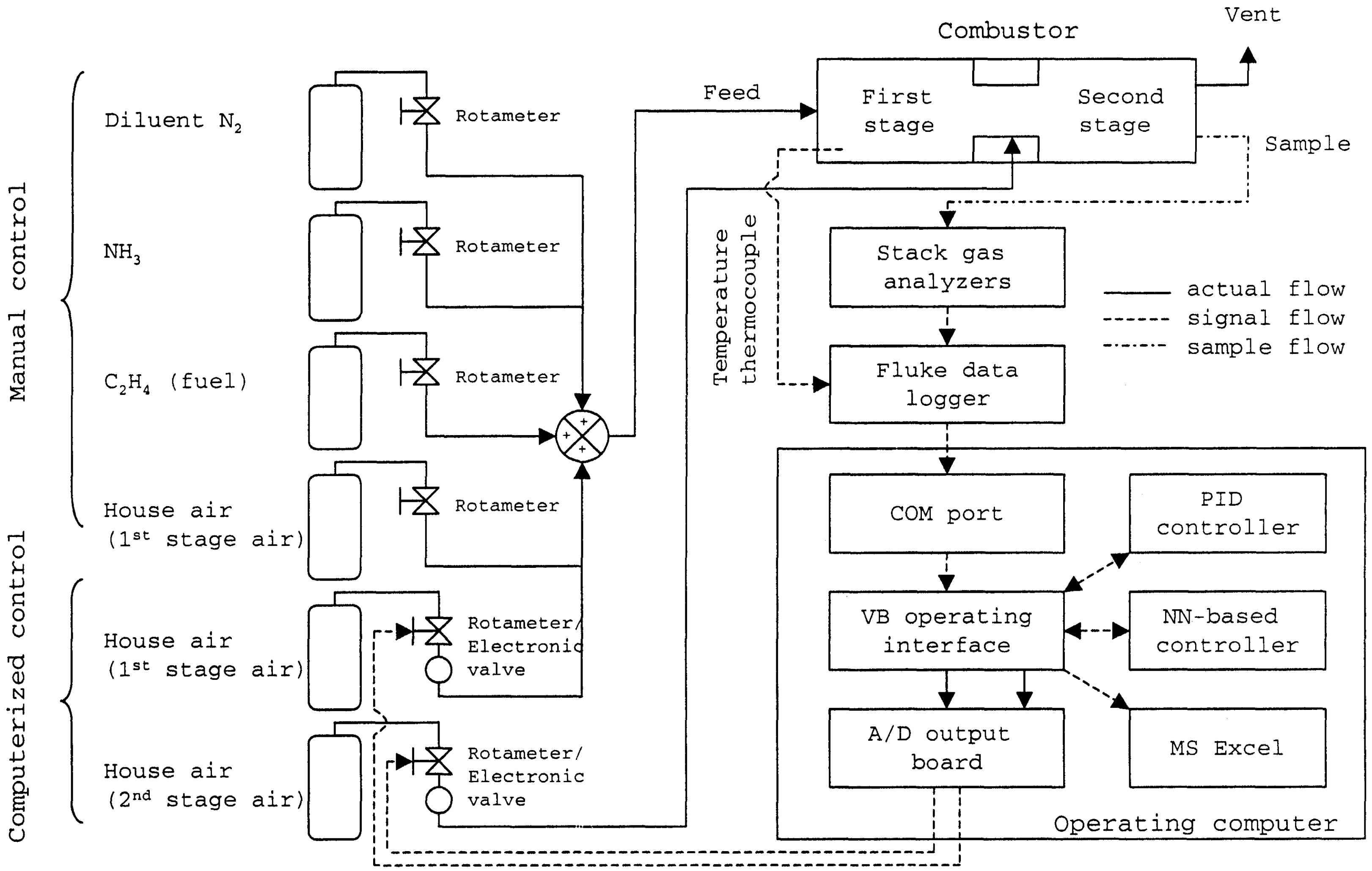
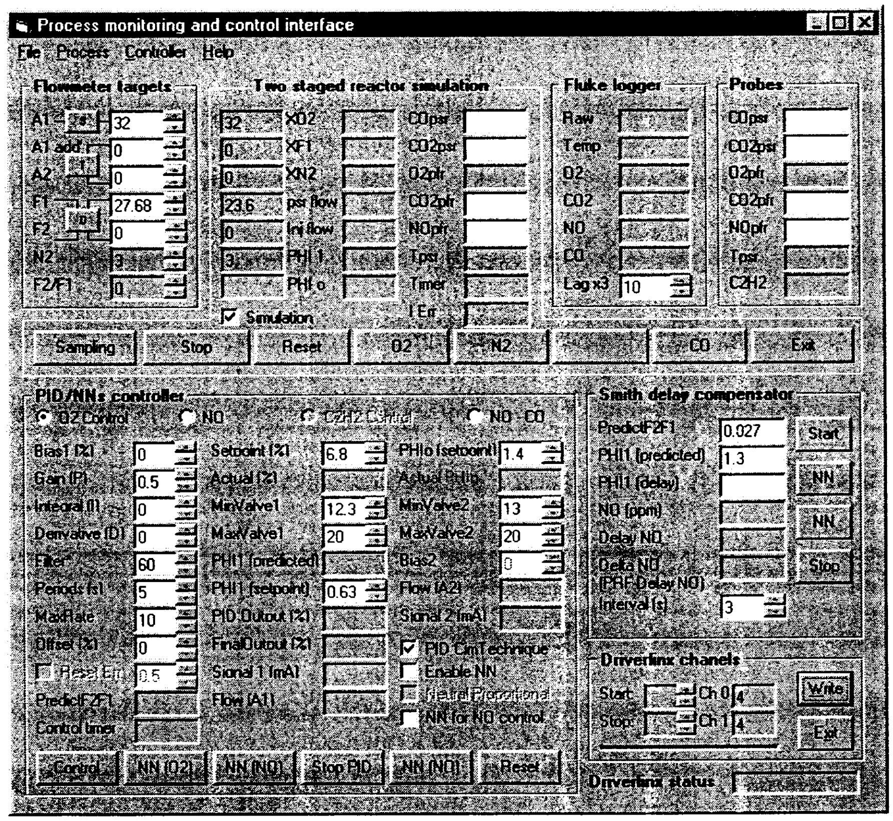


Figure 3.5 Process control schematic.

Analog signals from stack gas analyzers (O_2, CO_2, and NO) are continuously monitored and digitized by a Fluke data logger. All digital signals from the Fluke data logger are then sampled at the user's preset frequency rate to the COM port of an operating computer via an RS-232 interface.


Process monitoring and control interface

Flowmeter targets: A1 32; A1 add 0; A2 0; F1 27.68; F2 0; N2 3; F2/F1 0

Two staged reactor simulation: XO2 32; XF1 0; XN2 0; psr flow 23.6; Inj flow 0; PHI 1 3; PHI o; COpsr; CO2psr; O2pfr; CO2pfr; NOpfr; Tpsr; Timer; I Err; [✓ Simulation]

Fluke logger: Raw; Temp; O2; CO2; NO; CO; Lag x3 10

Probes: COpsr; CO2psr; O2pfr; CO2pfr; NOpfr; Tpsr; C2H2

PID/NNs controller: [✓ O2 Control] NO C2H2 Control NO - CO

Bias1 (%) 0; Gain (P) 0.5; Integral (I) 0; Derivative (D) 0; Filter 60; Periods (s) 5; MaxRate 10; Offset (%) 0; Reset Err 0.5; PredictF2F1; Control timer

Setpoint (%) 6.8; Actual (%); MinValve1 12.3; MaxValve1 20; PHI1 (predicted); PHI1 (setpoint) 0.63; PID Output (%); FinalOutput (%); Signal 1 (mA); Flow (A1)

PHIo (setpoint) 1.4; Actual PHIo; MinValve2 13; MaxValve2 20; Bias2 0; Flow (A2); Signal 2 (mA); [✓ PID CimTechnique]; Enable NN; Neutral Proportional; NN for NO control

Smith delay compensator: PredictF2F1 0.027; PHI1 (predicted) 1.3; PHI1 (delay); NO (ppm); Delay NO; Delta NO (PRF-Delay NO); Interval (s) 3

Driverlinx chanels: Start; Stop; Ch 0 4; Ch 1 4

Driverlinx status

Figure 3.6 Visual Basic operating interface.

The operating user interface, shown in Figure 3.6, is written in Visual Basic 6.0. The Visual Basic programming source code is listed in the Appendix. The software and hardware setup in the operating computer is shown in Figure 3.7. The operating interface enables the operating computer to accept and display the signals from the computer COM port. For a closed-loop experiment, the operating interface enables the operating computer to process the signals to and from selected control modules. A PID control algorithm is embedded as a module in the operating interface program, while the neural networks are embedded in the operating interface program as dynamic link library files (DLL). The operating interface allows manual control for an open-loop experiment as well.

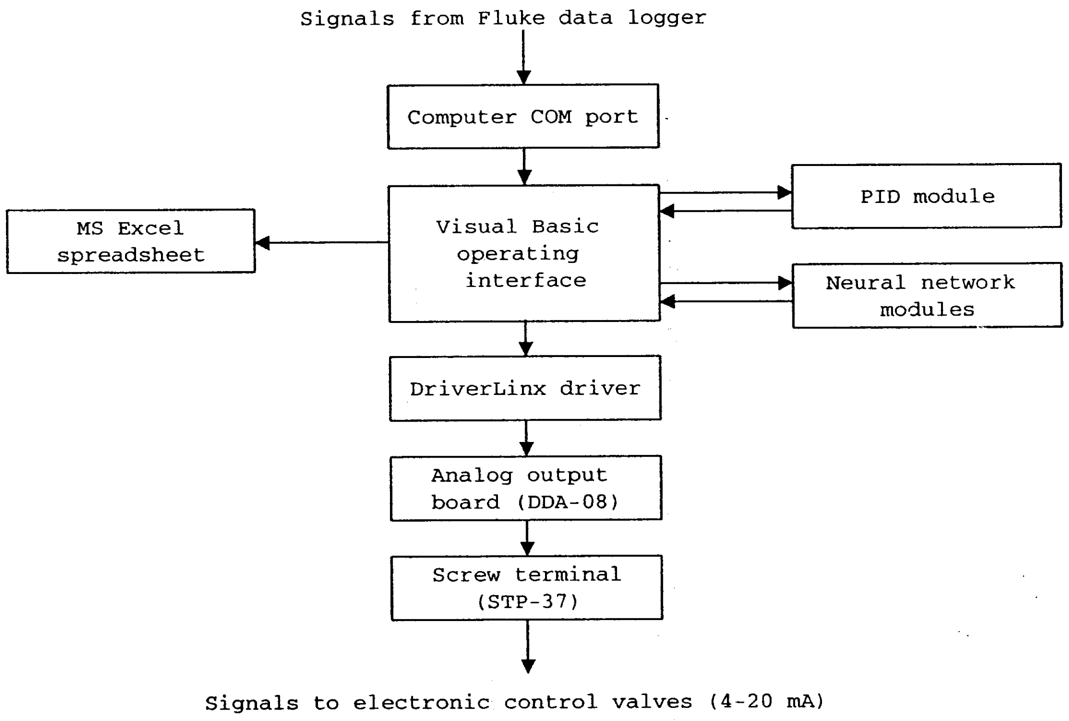

Signals from Fluke data logger
Computer COM port
Visual Basic operating interface
MS Excel spreadsheet
PID module
Neural network modules
DriverLinx driver
Analog output board (DDA-08)
Screw terminal (STP-37)
Signals to electronic control valves (4-20 mA)

Figure 3.7 Control software and hardware setup.

The operating computer is a 333 MHz Pentium II processor with 64 MB RAM and Windows 98 operating system. Based on instructions from the operating interface, the computer generates control signals to the final control elements, which are the two electronic control valves. The control signals are sent to the two electronic control valves through a Keithley Metrabyte 12 bit 8-channel analog output board (DDA-08) installed into the operating computer. The DriverLINX driver, which is the 32-bit application development device driver for custom data acquisition under the Windows environment, is used as the programming interface between the Visual Basic application and the DDA-08 analog output board. The DDA-08 analog output board provides selectable voltage or current outputs. Each output channel contains a buffer for storing data, thus enabling the board to achieve fast update rates. Two channels of the DDA-08 are utilized; one for primary air control, another one is for secondary air control. In this study, the DDA-08 board was configured and tested for 4-20 mA signal outputs by the DriverLINX utility program.

All experimental data from the Visual Basic operating interface are continuously stored into a pre-assigned Visual Basic memory buffer. Once the buffer is filled up, the data in the buffer are transferred and stored into a Microsoft Excel spreadsheet through the use of Visual Basic for Applications (VBA) and Microsoft Excel macro recorder.

To summarize the operation of the control system, the detailed sequence of measurements and calculations at each sampling time is as follows:

1. Measuring selected combustion effluent compounds by the stack gas analyzers.
2. Monitoring and digitizing the analog signals from the stack gas analyzers by the Fluke data logger.
3. Sampling digital signals from the Fluke data logger into the RS-232 COM port of the operating computer.
4. Accepting the digital signals from the operating computer COM port into the Visual Basic operating interface.

5. Accessing the selected control module.
6. Updating the control signals.
7. Outputting the control signals to the final control elements through the analog output board (DDA-08).
8. Storing selected experimental data into a pre-assigned buffer for transfering to the Microsoft Excel spreadsheet in batch.

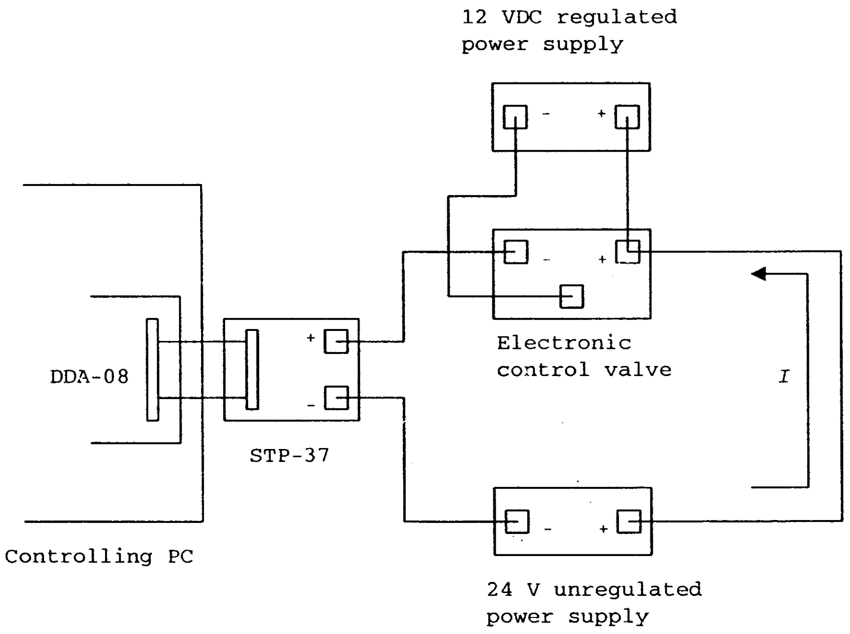


Figure 3.8 Electronic control valve circuit.

As mentioned, the two electronic solenoid valves serve as the final control elements. Both electronic valves are from Omega (model PV-104) and configured to receive 4-20 mA signals. Each electronic control valve is connected as a floating load to two power supplies; one is a 12 VDC regulated power supply for powering the valve, while the other is a 24 V unregulated power supplier (Omega model U24y101) for filtering signals to the valve. Higher current (control signal) results in more plunger movement of the electronic control valve and more flow. A Keithley external screw terminal panel (STP-37) allows easy access and wiring connections between the DDA-08 and the final control elements. A Diagram for the electrical circuit for the electronic control valve is shown in Figure 3.8.

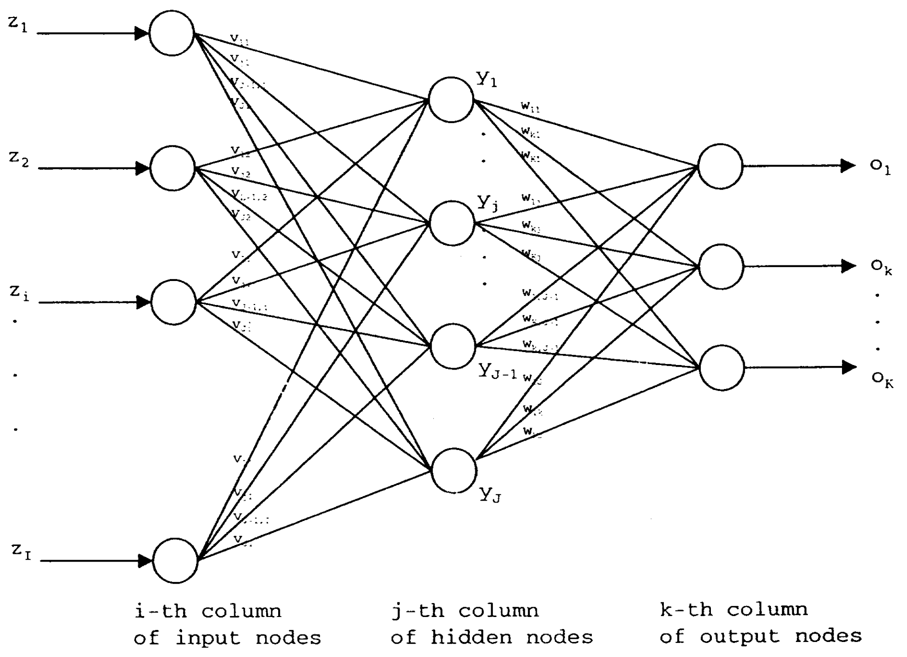
CHAPTER 4

NEURAL NETWORK TOPOLOGIES AND LEARNING ALGORITHM

4.1 Network Topologies

Feedforward multi-layer-perceptron (FMLP) networks were constructed and used in this study. This architecture has been successfully used in several ANN-based control applications (Allen et al., 1993; Bhat and McAvoy, 1990; Braak et al., 1998; Gomm et al., 1997; Hernández et al., 1992; Martins and Coelho, 2000; Nascimento et al., 2000; Nikravesh et al., 2000; Palancar et al., 1998; Reinschmidt and Ling, 1994; Syu and Chen, 1998; Tendulkar et al., 1998; Tao and Burkhardt, 1994). The FMLP network architecture is shown in Figure 4.1. The network consists of layers of interconnected parallel processing elements called nodes or neurons. "Feedforward" implies that network nodes have to be connected to one another in a forward fashion, and the output of the network is not allowed to be included as input to any of the network's nodes.

As shown in Figure 4.1, the left column is the input layer, the middle column is the hidden layer, and the right column is the output layer. The overall input to the feedforward neural network is the vector $\bar{z}$ that enters through the network input layer. The circles represent the network nodes. The lines joining the circles represent the network synapses or weighted information passed from one layer of nodes to the next layer of nodes. Each input node is connected to each node in the hidden layer. The signal from each input node is multiplied by a corresponding weight. The sums of each weighted signal become the inputs to the node in the hidden layer. Each of the nodes in the hidden layer is, in turn, connected to each node in the output layer. A similar algorithm is applied to each connection between the hidden nodes and the output nodes. The overall network output is the vector $\bar{o}$. The power of neural computation comes from the massive interconnection among the nodes that share the load of the overall processing task, and from the adaptive nature of the parameters (weights) that interconnect the nodes.



z_1
z_2
z_i
z_I
y_1
y_j
y_{J-1}
y_J
o_1
o_k
o_K
i-th column of input nodes
j-th column of hidden nodes
k-th column of output nodes

Figure 4.1 FMLP neural network architecture.

Figure 4.2 shows a closer look at a particular node k. Every node model consists of a processing element with synaptic input connections and a single output. The processing element consists of two computing units called a summing node and a threshold logic unit.

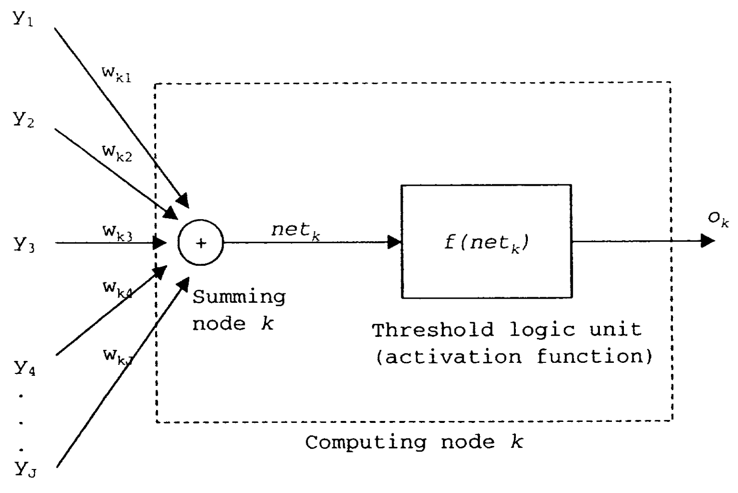


Figure 4.2 Neural network computing node.

The inputs to the summing node are denoted by the vector $\bar{y}$ where $\bar{y} = [y_1 \quad y_2 \quad \ldots \quad y_J]^t$. The output from the summing node k is called an activation value, given by:

$$net_k = \sum_{j=1}^{J} w_{kj} y_j \tag{4.1}$$

where w_{kj} are weights to be determined during the training process. Once the activation value is obtained, the node's output can be computed based on the selected mapping function:

$$o_k = f(net_k) \tag{4.2}$$

The function $f(net_k)$ is often referred to as an activation function. The activation function used in this study is a hyperbolic tangent function that can be expressed as follow:

$$f(net_k) = \tanh(net_k) \tag{4.3}$$

Normally, a neural network can have several layers of nodes or processing elements. In this study, a simple network structure with one hidden layer is used.

4.2 Network Learning Algorithm

In order to perform any processing tasks, the neural network must "learn" an input-output mapping from a set of known input-output pairs. Back-propagation is by far the most common form of learning (Allen et al., 1990; Bhat and McAvoy, 1990; Gomm et al., 1997; Hernández and Arkun, 1992; Miller and Lemieux 1998; Nascimento et al., 2000; Nikravesh et al., 2000; Palancar et al., 1998; Reinschmidt and Ling, 1994; Shu and Pi, 2000; Syu and Chen, 1998; Tendulkar et al., 1998). In this study, each neural network was trained offline with a static error back-propagation training algorithm in a supervised learning mode. The objective of the training is to find the series of weights that provides the best possible approximation of the network outputs, based on the training set of input-output pairs.

4.2.1 Supervised Learning Process

Network training involves minimization of an error function using a steepest descent strategy known as the generalized delta rule wherein the network outputs are compared with their desired values (known target value). The difference between the network outputs and their desired values is then used to modify the interlayer connection weights. The mapping error is propagated backward from the output layer through the hidden layer toward the input layer. This mechanism of backward error transmission is used to modify the network synaptic weights. The input-output mapping, comparison of target and actual values, and weight adjustment continue until all mapping examples from the training set are learned with in an acceptable error. Usually, mapping error is cumulative and computed over the full training set. The detailed description of the error back-propagation training algorithm and delta rule learning can be found elsewhere (Zurada, 1992).

As shown in Figure 4.1, the input and output values of the network are denoted z_i and o_k, respectively, for $i = 1, 2, \ldots, I$ and $k = 1, 2, \ldots, K$. The signal values from the hidden layer are denoted y_j, for $j = 1, 2, \ldots, J$. The weight w_{kj} connects the j^{th} neuron in the hidden layer to the k^{th} neuron in the output layer. The weight v_{ji} connects the i^{th} neuron in the input layer to the j^{th} neuron in the hidden layer. During the classification, usually called the recall phase, the trained neural network operates in a feedforward manner as described in the following expressions:

$$\bar{o} = \Gamma\left[\overline{W}\bar{y}\right] \tag{4.4}$$

$$\bar{y} = \Gamma\left[\overline{V}\bar{z}\right] \tag{4.5}$$

where the input, hidden, and output vectors and the weight matrices are, respectively

$$\bar{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_I \end{bmatrix}, \quad \bar{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_J \end{bmatrix}, \quad \bar{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_K \end{bmatrix}$$

$$\overline{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1I} \\ v_{21} & v_{22} & \cdots & v_{2I} \\ \vdots & \vdots & \cdots & \vdots \\ v_{J1} & v_{J2} & \cdots & v_{JI} \end{bmatrix}, \quad \overline{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1J} \\ w_{21} & w_{22} & \cdots & w_{2J} \\ \vdots & \vdots & \cdots & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KJ} \end{bmatrix}$$

A nonlinear diagonal operator $\Gamma[\cdot]$ is,

$$\Gamma[\cdot]=\begin{bmatrix} f(\cdot) & 0 & \cdots & 0 \\ & f(\cdot) & & 0 \\ \vdots & \vdots & \cdots & \\ 0 & 0 & \cdots & f(\cdot) \end{bmatrix}$$

where f is the network activation function as described earlier.

In the supervised learning process, the desired (target) output vector has to be provided from the set of known input-output (training set) data.

$$\bar{d}=\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_K \end{bmatrix}$$

4.2.2 Output Layer Weight Adjustment Algorithm

To formulate the network-learning algorithm, the adjustment of the synaptic weights between the network output layer and the hidden layer is first to be considered.

The delta-learning rule is simply derived from the condition of least squared error between network output o_k and desired output d_k. The error expression for a neuron in the output layer can be expressed as follows:

$$E=\frac{1}{2}\sum_{k=1}^{K}(d_k-o_k)^2 \tag{4.6}$$

Minimization of the error based on the delta-learning rule requires the weight changes Δw_{kj} to be in the negative gradient direction (where η is a positive constant):

$$\Delta w_{kj}=-\eta\frac{\partial E}{\partial w_{kj}} \tag{4.7}$$

The computation at each node in the output layer is

$$net_k = \sum_{j=1}^{J} w_{kj} y_j \tag{4.8}$$

The k^{th} neuron's output is

$$o_k = f(net_k) \tag{4.9}$$

The error signal term δ (called delta) produced by the k^{th} neuron in the output layer is defined as follows:

$$\delta_{ok} = -\frac{\partial E}{\partial(net_k)} \tag{4.10}$$

Using the chain rule, the term on the right-hand side in Equation 4.7 can be written as follows:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial(net_k)} \cdot \frac{\partial(net_k)}{\partial w_{kj}} \tag{4.11}$$

The second term of the product of Equation 4.11 is the derivative of the sum of products of weights and patterns $w_{k1}y_1 + w_{k2}y_2 + w_{k3}y_3 + \ldots + w_{kJ}y_J$. Thus, it can be written as follows:

$$\frac{\partial(net_k)}{\partial w_{kj}} = y_j \tag{4.12}$$

Combining Equations 4.10 and 4.12 leads to the following form for Equation 4.11:

$$\frac{\partial E}{\partial w_{kj}} = -\delta_{ok} y_j \tag{4.13}$$

From Equations 4.7 and 4.13, the weight change can be obtained

$$\Delta w_{kj} = \eta \delta_{ok} y_j \tag{4.14}$$

Equation 4.14 represents the general formula for delta-learning weight adjustments. In order to obtain the Δw_{kj}, the error signal term delta δ_{ok} needs to be computed. From Equation 4.10, it can be expressed as follows:

$$\delta_{ok} = -\frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial (net_k)} \tag{4.15}$$

Denoting the second term in Equation 4.15 as a derivative of the activation function

$$f_k'(net_k) = \frac{\partial o_k}{\partial (net_k)} \tag{4.16}$$

From Equation 4.6, we know that

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k) \tag{4.17}$$

Equation 4.15 can be rewritten as follows:

$$\delta_{ok} = (d_k - o_k) f_k'(net_k) \tag{4.18}$$

Hence, the final formula for the output layer weight adjustment can now be obtained from Equation 4.14 as

$$\Delta w_{kj} = \eta (d_k - o_k) f_k'(net_k) y_j \tag{4.19}$$

The update weight values w_{kj}' become

$$w_{kj}' = w_{kj} + \Delta w_{kj} \tag{4.20}$$

Thus, the updated individual weights under the delta training rule can be expressed for $k = 1, 2, \ldots, K$, and $j = 1, 2, \ldots, J$, as follows:

$$w_{kj}' = w_{kj} + \eta (d_k - o_k) f_k'(net_k) y_j \tag{4.21}$$

The updated weights for the output layer can be expressed using the vector notation

$$\overline{W}' = \overline{W} + \eta \overline{\delta}_o \overline{y}^t \tag{4.22}$$

where the error signal vector $\bar{\delta}_o$ is defined as a column vector consisting of the individual error signal terms:

$$\bar{\delta}_o = \begin{bmatrix} \delta_{o1} \\ \delta_{o2} \\ \vdots \\ \vdots \\ \delta_{ok} \end{bmatrix}$$

4.2.3 Hidden Layer Weight Adjustment Algorithm

Layers with neurons whose outputs are not directly accessible are called internal or hidden layers. In this study, all networks are two-layer networks with one hidden layer as shown in Figure 4.1. The network shown in Figure 4.1 can be called a single hidden-layer network. The delta-training rule, previously used for the output layer, is now generalized for the weight increment Δv_{ji} for the hidden layer.

The negative gradient descent formula for the hidden layer is

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}} \tag{4.23}$$

The error signal term δ_{yj} is now defined as

$$\delta_{yj} = -\frac{\partial E}{\partial (net_j)} \tag{4.24}$$

With the chain rule,

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial (net_j)} \cdot \frac{\partial (net_j)}{\partial v_{ji}} \tag{4.25}$$

Similar to Equation 4.14, the second term of the product of Equation 4.25 is equal to z_i. Thus, the general expression for delta-learning weight adjustment in the hidden layer can be expressed as follows:

$$\Delta v_{ji} = \eta \delta_{yj} z_i \tag{4.26}$$

The error signed term δ_{yj} is computed as:

$$\delta_{yj} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial (net_j)} \tag{4.27}$$

From the definition of Equation 4.6, the first term of the product of Equation 4.27 becomes

$$\frac{\partial E}{\partial y_j} = -\left(\sum_{k=1}^{K} (d_k - o_k) f'(net_k) \frac{\partial (net_k)}{\partial y_j} \right) \tag{4.28}$$

The second term of the product of Equation 4.27 is equal to

$$f'_j(net_j) = \frac{\partial y_j}{\partial (net_j)} \tag{4.29}$$

Equation 4.28 can be simplified to a compact form by using Equations 4.8 and 4.18

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K} \delta_{ok} w_{kj} \tag{4.30}$$

Combining Equations 4.29 and 4.30 and substituting back into Equation 4.27

$$\delta_{yj} = f'_j(net_j) \sum_{k=1}^{K} \delta_{ok} w_{kj} \tag{4.31}$$

The weight adjustment Equation 4.26 in the hidden layer now becomes

$$\Delta v_{ji} = \eta f'_j(net_j) z_i \sum_{k=1}^{K} \delta_{ok} w_{kj} \tag{4.32}$$

The modified weights of the hidden layer can be expressed as

$$v'_{ji} = v_{ji} + \Delta v_{ji} \tag{4.33}$$

$$v'_{ji} = v_{ji} + \eta f'_j\left(net_j\right) z_i \sum_{k=1}^{K} \delta_{ok} w_{kj} \tag{4.34}$$

The updated weights for the hidden layer can be expressed using the vector notation as follows:

$$\overline{V}' = \overline{V} + \eta \overline{\delta}_y \overline{z}^t \tag{4.35}$$

where

$$\overline{\delta}_y = \begin{bmatrix} \delta_{y1} \\ \delta_{y2} \\ \vdots \\ \vdots \\ \delta_{yJ} \end{bmatrix}, \quad \overline{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1I} \\ v_{21} & v_{22} & \cdots & v_{2I} \\ \vdots & \vdots & \cdots & \vdots \\ v_{J1} & v_{J2} & \cdots & v_{JI} \end{bmatrix}, \quad \overline{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_I \end{bmatrix}$$

4.2.4 Summary of the Supervised Error Back-propagation Training

The flow chart summarizing the supervised error back-propagation training algorithm for a two-layer network is shown in Figure 4.3. The weighting of the connections between various nodes can be arbitrarily set at the beginning of the training process. With initial weights $\overline{W}$ and $\overline{V}$, the learning begins with the feedforward recall process of a single known input-output training pattern ($\overline{z}_p$ and $\overline{d}_p$). It should be noted here that there are *P* training patterns (*P* known input-output pairs) in the training set (i.e. there are *P* input patterns of $\overline{z}_1, \overline{z}_2, \ldots, \overline{z}_P$ and *P* desired output patterns of $\overline{d}_1, \overline{d}_2, \ldots, \overline{d}_P$).

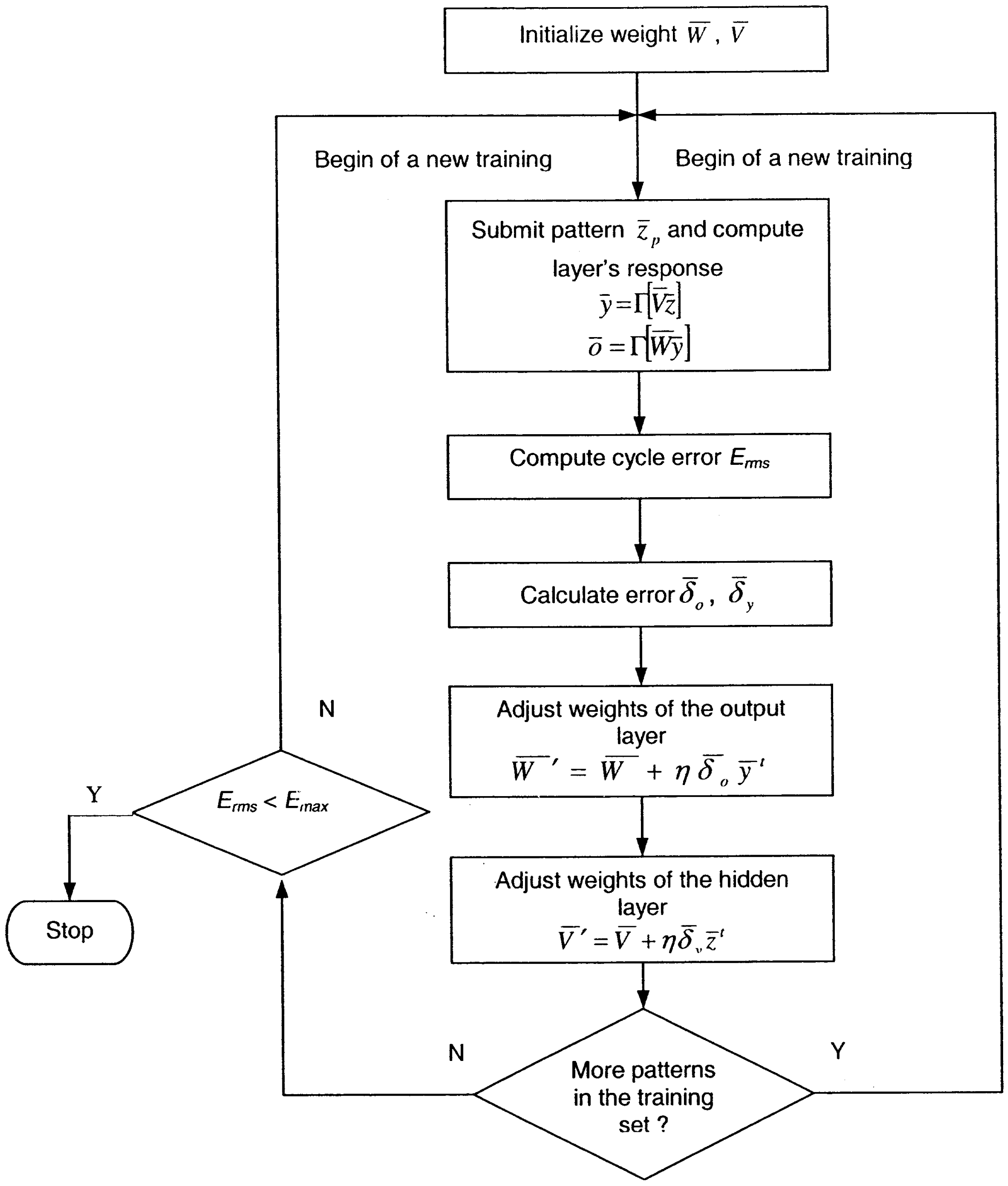

Initialize weight $\overline{W}$, $\overline{V}$
Begin of a new training
Begin of a new training
Submit pattern $\overline{z}_p$ and compute layer's response
$\overline{y}=\Gamma[\overline{V}\overline{z}]$
$\overline{o}=\Gamma[\overline{W}\overline{y}]$
Compute cycle error E_{rms}
Calculate error $\overline{\delta}_o$, $\overline{\delta}_y$
Adjust weights of the output layer
$\overline{W}' = \overline{W} + \eta\overline{\delta}_o\overline{y}^t$
Adjust weights of the hidden layer
$\overline{V}' = \overline{V} + \eta\overline{\delta}_v\overline{z}^t$
More patterns in the training set ?
N
Y
$E_{rms} < E_{max}$
N
Y
Stop

Figure 4.3 Back-propagation learning flowchart.

After submitting the vector $\bar{z}_p$ at the input layer, the hidden layer responses $\bar{y}_p$ and output layer responses $\bar{o}_p$ are computed. The cumulative error is then computed over the training cycle. The most commonly used measure of cumulative error, the root-mean-square normalized error (E_{rms}), is employed here. It is given by

$$E_{rms} = \frac{1}{PK}\sqrt{\sum_{p=1}^{P}\sum_{k=1}^{K}\left(d_{pk} - o_{pk}\right)^2} \tag{4.36}$$

The next step in the training process is the computation of the error signals in the output and hidden layer, respectively. Once the error signal vectors ($\bar{\delta}_o$ and $\bar{\delta}_y$) are obtained, the $K{\times}J$ weight values in the output layer can be adjusted with in the matrix $\overline{W}$. After the updating of the matrix $\overline{W}$, the $J{\times}I$ weight values in the hidden layer are adjusted in the matrix $\overline{V}$. This computation is repeated to complete all of the P training patterns in the training set. After each complete training set, the final error value (based on Equation 4.36) is calculated for the entire training cycle. Typically, as the network learned, the error E_{rms} exponentially dropped toward zero. The learning procedure stops when the final error value drops below the preset error limit E_{max}. The final weight metrics $\overline{W}$ and $\overline{V}$ are then stored in the network as a result of the learning process.

Once the network is trained, it can utilize the experiential training patterns ($\overline{W}$, $\overline{V}$) to process the actual (non-training) input, and map them to the optimal output values. The more relevant the training set is, the more efficiently the network can be generalized. Therefore, the training set must be carefully selected to span the whole range of the possible combustion behavior conditions.

It must be noted that the lower error E_{rms} does not always mean a better network. It is possible to overtrain the network. Overtraining is a phenomenon where the network tends to memorize the training patterns rather than generalize or interpolate them. It was suggested by Hecht-Nielson (1989) that the overtraining is typically illustrated by the learning-curve behavior exhibited in Figure 4.4. The network's performance is measured using two different data sets: the training set and the testing set. While the training set error constantly decreases and levels out, the testing set error decreases for a while, but then begins to increase again. The challenge is to keep monitoring both the training error, and testing error and then stop the training when the testing set error is at its minimum, while at the same time, the training set error reaches its minimum.

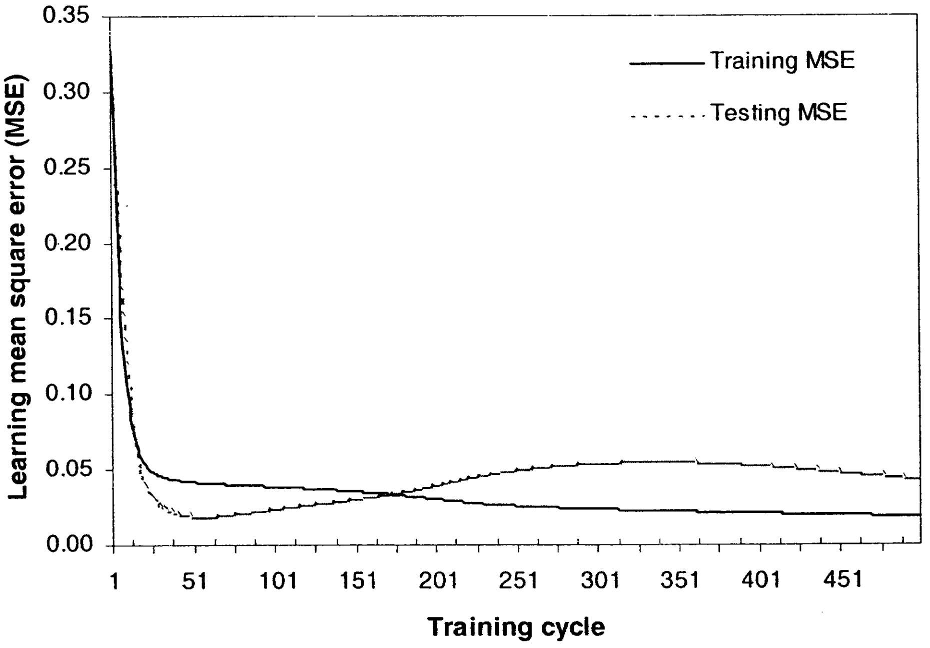


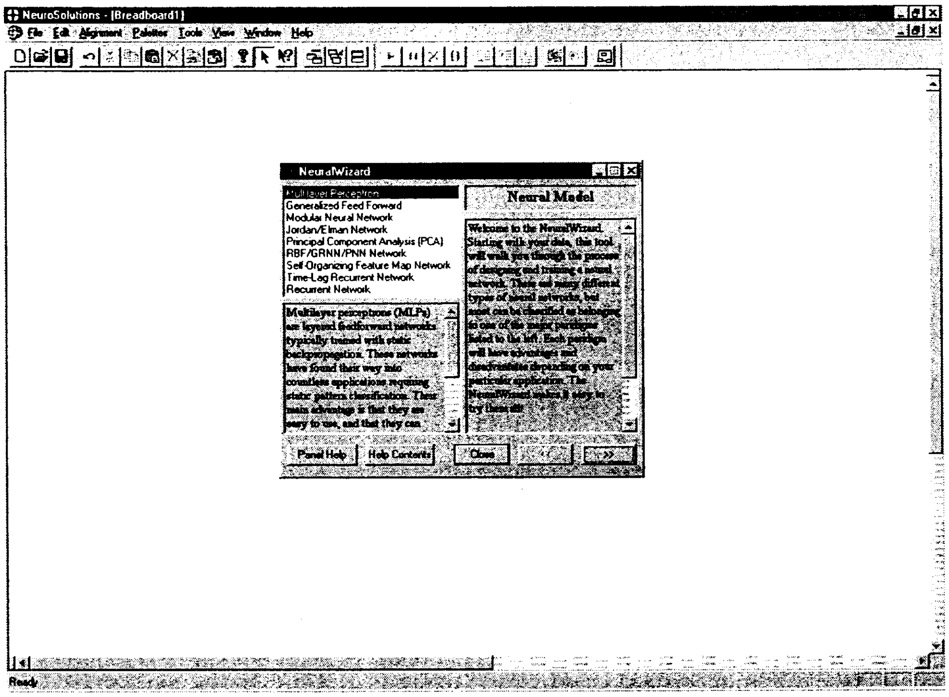
Figure 4.4 Overtrained neural network learning curves.

4.3 Neural Network Construction and Implementation

The NeuroSolutions software package from NeuroDimension, Inc. was used to construct all neural networks in this study. The software combines a modular, icon-based network design interface with a built-in custom wizard. These allow the user to customize networks, generate, compile executable dynamic link library (DLL) files, and embed them into the existing Visual Basic controlling interface.

The NeuralWizard is an application that comes with the NeuroSolutions software package. It aids in the design and construction of the neural network. The NeuralWizard presents a series of user-interface panels that represent logical steps in the network design process. At each panel, the user can specify preference choices of parameters in constructing the desired neural network. A sample of the user-interface panel for specifying the network topology is shown in Figure 4.5. It should be noted that although a number of network nodes and layers configuration were tried in the following chapters, it was found that an optimal network configuration can be well guided by the NeuralWizard.

Each neural network is built on the NeuroSolutions worksheet. A worksheet is called a breadboard. A sample of the 2-4-4 neural network architecture is shown in Figure 4.6. For each network component (represented by an icon on the breadboard), there is a one-to-one correspondence between the icons on the breadboard and the numerical computations going on behind the graphic user interface (GUI). Each network component also has a corresponding parameter set that the user can edit or specify through a dialog box on the breadboard.

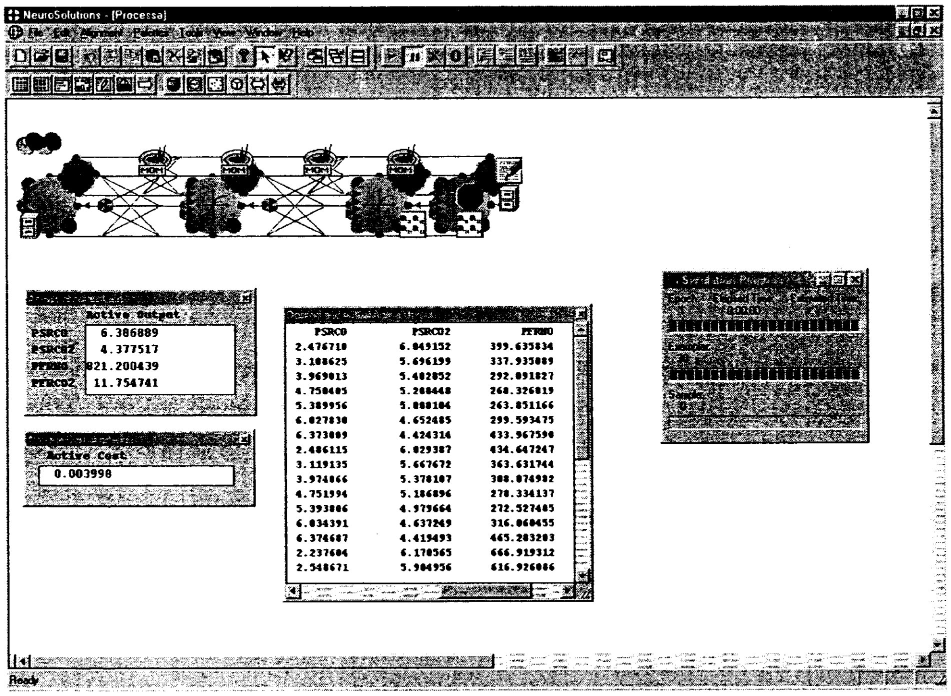

NeuroSolutions - [Breadboard1]
File Edit Alignment Palettes Tools View Window Help
NeuralWizard
Neural Model
Panel Help
Help Contents
Close
<<
>>
Ready

Multilayer Perceptron
Generalized Feed Forward
Modular Neural Network
Jordan/Elman Network
Principal Component Analysis (PCA)
RBF/GRNN/PNN Network
Self-Organizing Feature Map Network
Time-Lag Recurrent Network
Recurrent Network

Multilayer perceptrons (MLPs) are layered feedforward networks typically trained with static backpropagation. These networks have found their way into countless applications requiring static pattern classification. Their main advantage is that they are easy to use, and that they can

Welcome to the NeuralWizard. Starting with your data, this tool will walk you through the process of designing and training a neural network. There are many different types of neural networks, but most can be classified as belonging to one of the major paradigms listed to the left. Each paradigm will have advantages and disadvantages depending on your particular application. The NeuralWizard makes it easy to try them all!

Figure 4.5 NeuralWizard user interface.


NeuroSolutions - [Processa]
File Edit Alignment Palettes Tools View Window Help
Ready

Active Output

PSRC0	6.386889
PSRC02	4.377517
PFRN0	821.200439
PFRC02	11.754741

Active Cost

0.003998

PSRC0	PSRC02	PFRN0
2.476710	6.049152	399.635834
3.108625	5.696199	337.935089
3.969013	5.402852	292.091827
4.750405	5.200448	268.326019
5.389956	5.000104	263.851166
6.027030	4.652405	299.593475
6.373009	4.424314	433.967590
2.486115	6.029387	434.647247
3.119135	5.667672	363.631744
3.974066	5.378107	308.074982
4.751994	5.186896	278.334137
5.393006	4.979664	272.527405
6.034391	4.637249	316.060455
6.374607	4.419493	465.283203
2.237604	6.170565	666.919312
2.548671	5.904956	616.926086

Figure 4.6 A complete-built neural network on the NeuroSolutions breadboard.

Data collection is crucial in the network training process. The training data should cover conditions that the network may encounter during the feedforward recall process. In this study, the training data were collected from an existing experimental and CHEMKIN simulation run database at various steady states. An approach used to minimize the overtraining was to visually examine and pre-filter the training with raw data. Noisy data-points were purposely removed from the training set in order to prevent the network from learning the noisy pattern that could potentially cause deterioration of the network generalization performance.

The training data are coded and written in a column-formatted ASCII file before being fed into the neural network. Each column of a column-formatted ASCII file represents one channel of data. Each channel may be tagged as the input or desired output.

Prior to the network training, the input-output data are partitioned into two sets: the training set and the testing set. The former is used to update the network weights while the testing set is used to evaluate the generalization ability of the network model. All of the training and testing data are presented to the networks in random order. This was proven to break up any serial correlations in the data, and lead to significant improvements in convergence speed and performance of a trained network (Gomm et al., 1997). Since the generalization ability of the network model was identified offline using the testing set, the testing set had to be carefully selected to contain essentially every possible case that the network would encounter in the actual experimental environment.

During the training process, the input and desired data from the training and testing sets are repeatedly fed to the complete-built network shown on the breadboard

(Figure 4.6). To monitor how well the network has learned, the cost function (E_{rms}) of the training and testing sets is observed. Typically, as the network learns, the E_{rms} exponentially drops toward zero. The typical learning curves for a well-trained neural network are shown in Figure 4.7, wherein the errors of both training and testing sets level out simultaneously. The term MSE, shown in Figure 4.7, stands for mean-square-error and is used in the NeuroSolutions software package. It is equivalent to E_{rms}.

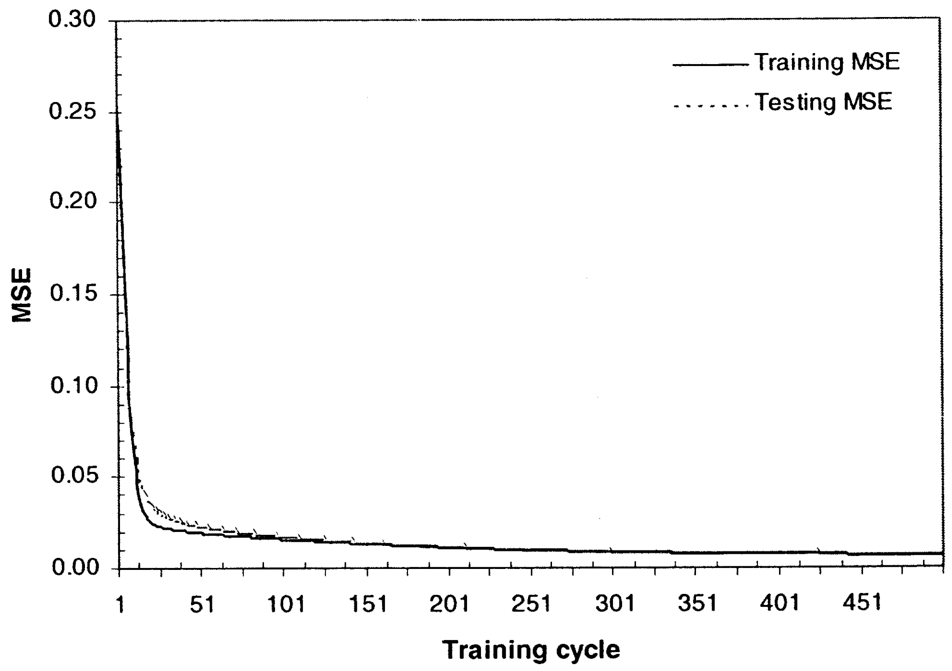


Figure 4.7 Well-trained neural network learning curves.

The NeuroSolutions software package allows the user to specify the number of training cycles or the learning error limit. In the former case, the learning process stops when the number of training cycles reaches the specified number. In the latter case, the learning process stops when the learning error drops below the user-input error limit ($E_{rms} < E_{max}$). In this study, the software package offers a cross validation function that allows the user to monitor the training and testing learning curves simultaneously (as shown in Figure 4.7). Thus, the optimal number of training cycles can be found and the overtraining effect is minimized.

Each well-trained neural networks is converted into a dynamic link library (DLL) file that is compatible with the Visual Basic language by the Custom Solution Wizard. The Custom Solution Wizard is an application, provided with the NeuroSolutions software package, that takes a neural network created in the NeuroSolutions breadboard and automatically generates and compiles a DLL file. The DLL file is an executable module, integrated into the Visual Basic operating interface, that can be called from the Visual Basic operating interface to perform the mapping input-output function (recall process). All measurable combustor parameters are displayed and preprocessed in the Visual Basic operating interface before being fed into the neural networks. The outputs from the trained neural networks are displayed on the Visual Basic operating interface. The outputs are then directed to computation modules for calculating the signals to the final control elements (proportional valves).

CHAPTER 5

FEEDBACK PROCESS CONTROL

5.1 Introduction

The proportional-integral-derivative (PID) controller has been widely accepted as one of the robust control techniques in the industry. However, the traditional PID controller can be challenged by process oscillations when large disturbances or setpoint changes are encountered. Another drawback of implementing the PID controller is that tuning can be time-consuming and can require a combination of operational experience and trial-and-error. The task becomes even more difficult for highly nonlinear processes especially in the presence of significant time delay because of the many possible combinations of PID parameter settings (see Equation 5.2).

A proportional controller is one of the simplest control configurations. Tuning the proportional controller is simple and requires a lot less skill and time than tuning the PID. However, the proportional controller has the inherent disadvantage of its inability to totally eliminate the sustained error or offset when it encounters a process disturbance. Theoretically, the offset can be eliminated by manually resetting the process setpoint or the control bias value (refer to Equation 5.2).

An alternative control strategy that suppresses the offset, but with less tuning complications, uses a neural network in conjunction with the conventional proportional controller. This approach is introduced and tested in this chapter. The proposed control strategy falls into one category of the model-based control methodologies. The goal is to let the neural network represent the experiential operator's knowledge to update the proportional controller's bias value, so that the process offset can be minimized.

5.2 Experimental Outlines

The stated control objective in this chapter is to control the O_2 level in the combustor outlet at the desired setpoint. Primary air is the oxidizer and only ethylene (C_2H_4) serves as the fuel. The fuel/air mixture composition is characterized by the equivalence ratio ϕ given by:

$$\phi = \frac{\left(\frac{Fuel}{Air}\right)_{actual}}{\left(\frac{Fuel}{Air}\right)_{stoichiometric}} \tag{5.1}$$

where Fuel = volumetric or molar flow rate of C_2H_4

Air = volumetric or molar flow rate of primary air

There is no air injection into the second stage of the combustor; that is, the first stage and overall equivalence ratios are equal ($\phi_1 = \phi_o = \phi$). At a selected steady state, a step-increase disturbance of the fuel (C_2H_4) is manually made. This step disturbance causes changes in the equivalence ratio ϕ, consequently changing the O_2 levels in the combustor outlet. After a step disturbance, the controller, which receives signals from a paramagnetic O_2 analyzer, takes immediate action by computing and sending control signals to a manipulated variable (primary air flow rate) and brings the process to the new steady state. Time delay in the gas sampling and analysis system is not significant (less than 45 seconds), so it does not have a severe impact on the control performance. The control signals are based on the deviation of the measured O_2 from the user's preset setpoint.

In order to establish a control experimental baseline, a series of open-loop runs were made to examine the effects of the feed composition on an outlet O_2 level. The feed equivalence ratio was varied from 0.60 to 1.35 by changing the flow rate of the primary air and C_2H_4. The first-stage reactor temperature was kept constant at 1673 K with diluent N_2. A monotonic relationship between the equivalence ratios ϕ and the observed O_2 level is shown in Figure 5.1. The result is consistent with the previous study by Mao (1995).

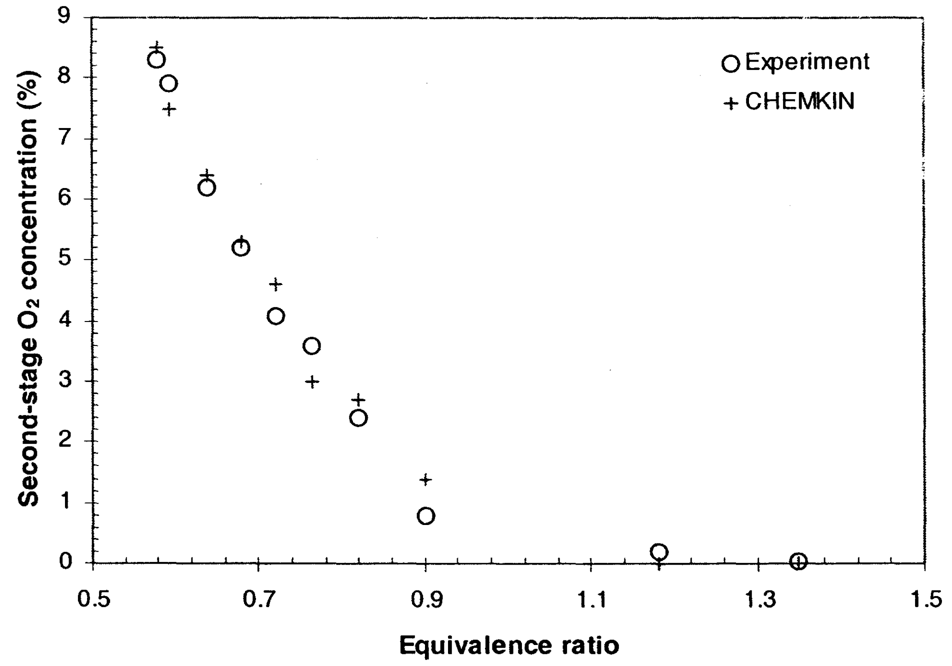


Figure 5.1 Second-stage O_2 concentrations at various steady states.

A control baseline is to first set the equivalence ratio at 0.63, which yields the O_2 level at about 6.8%. A step disturbance is then manually applied to the fuel flow rate

(C_2H_4) that causes ϕ to rise to 0.76. From Figure 5.1, this would yield about 3.4% of O_2 concentration in the combustor outlet for the open-loop run. The control objective is to bring the O_2 level back 6.8% by manipulating the primary air flow rate. The performance of the PID controller will be considered first, followed by the implementation of the proportional-neural-network controller (PNNC).

5.3 Experimental Setup

5.3.1 A PID Controller

The experimental setup for the PID controller is shown in Figure 5.2. The computer source code for the PID algorithm is obtained from CIMTechniques, Inc. The source code was originally written in Basic. It is attached as a module in the Visual Basic operating interface. The PID function is described by the following equation:

$$p(t) = \overline{p} + K_c \left[e(t) + \frac{1}{\tau_I} \int e(t)dt + \tau_D \frac{de(t)}{dt} \right] \tag{5.2}$$

where $p(t)$ = controller output

$\overline{p}$ = bias value

K_c = controller gain

τ_I = integral time

τ_D = derivative time

$e(t)$ = error signal between setpoint and measured control variable

The PID controller is set to execute every sampling time period, chosed by an operator. All PID parameters can be adjusted through the Visual Basic operating interface by the operator. The adjustable parameters are summarized below:

- Proportional gain (K_c)
- Integral gain (τ_I)
- Derivative gain (τ_D)
- Period or control action frequency

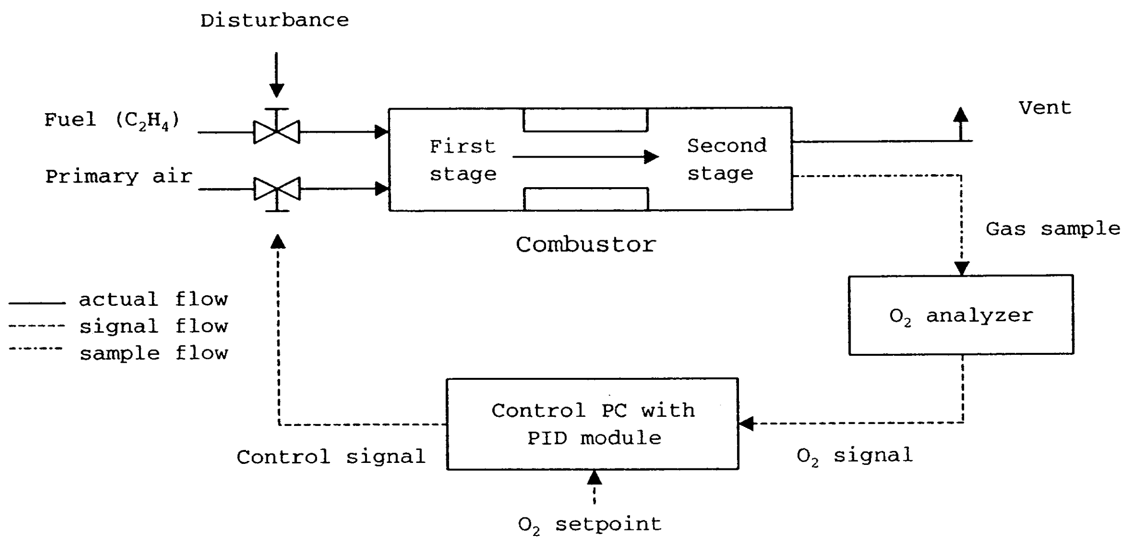


Figure 5.2 PID experimental schematic.

As the name implies, the proportional gain K_c directly adds a contribution to the control output proportional to the difference between the process setpoint and the measured value of the control variable. If the proportional gain is high, the control variable can reach the setpoint more quickly. However, if K_c is too high, the system will not have time to react as the setpoint is approached and an overshoot or oscillation will occur. The proportional gain can be reduced to minimize the overshoot and the oscillation. A small K_c may lead the system to reach equilibrium before the process variable approaches the process setpoint. The difference between the measured control variable and the setpoint at the new steady state is known as the offset.

The integral term integrates the process error and adds a contribution proportional to the integrated sum. The integral term causes the controller output to change as long as an error signal $e(t) \neq 0$. One disadvantage of applying the integral control action is a phenomenon called reset windup. This results from a large integral term. The reset windup can also occur as a consequence of a large sustained load disturbance that is beyond the range of a manipulated variable. In both situations, the integral term in the PID function continues to build up after the controller output saturates. This is because the controller is already doing all it can to reduce the error, but a physical limitation of the final control element (i.e. the maximum flow rate through the electronic control valve) prevents the controller from reducing the error signal to zero. To reduce the reset windup effect, in this study, the control signal is limited to between 0 – 100% control output.

Derivative control takes action according to the speed and direction of the change of the process variable. It tends to improve the dynamic response of the controlled variables, especially if there are some time lags in process response, by anticipating and correcting the future process behavior. Ideally, this decreases the process settling time, i.e., the time it takes the process to reach steady state.

The control time interval (period) or frequency of control actions (1/period) is another important parameter. Several time intervals were tried in the preliminary test. Small time intervals tend to make the process response faster, which in turn, minimize the process settling time. However, the process might exhibit an undesirable process response (e.g. setpoint overshoot or process oscillation) if the time interval is too small. Factors used to determine the time intervals in this study are the actual process residence time, sampling and analysis time, and the speed of the operating computer. The time interval of 5 seconds yields the best response to control action. It is also the shortest interval during which the operating computer can still effectively process all programming tasks.

Tuning the PID is time-consuming because of many possible combinations of the parameter settings. In this study, a process reaction curve tuning method, originally proposed by Ziegler and Nichols, was used to develop a process model. With the controller in the manual mode, a step change in the controller output (primary air flow rate) was introduced and the measured process response (O_2 concentration) was recorded.

The process reaction curve and the controller output are shown in Figures 5.3 and 5.4, respectively. The initial ϕ was 0.80 and the controller output was 12.30 mA. The controller output was then manually increased to 17.69 mA and thus caused ϕ to drop to 0.64. From Figure 5.3, the second-stage O_2 concentration rose from 4% to 7%. Based on the process reaction curve and the transfer functions for the individual instrument in the control loop, the PID control setting can be calculated from the Cohen and Coon controller design relations (Seborg et al., 1989) as $K_c = 1.04$, $\tau_I = 7.6$, and $\tau_D = 1.2$.

The control setting, derived through the process reaction curve, is based on the assumption that the process is a first-order system with a short time-delay. Therefore, it is possible that the approximation may be poor and may need further adjustments. In this study, the final tuning process was made experimentally (online) wherein the Cohen and Coon setting was used as an initial guess.

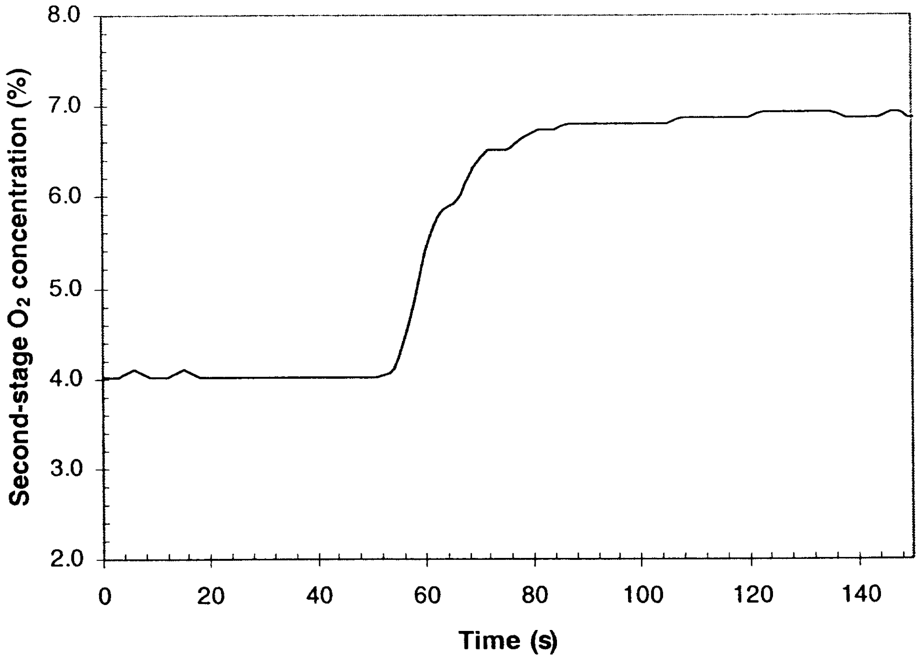


Figure 5.3 Process reaction curve.

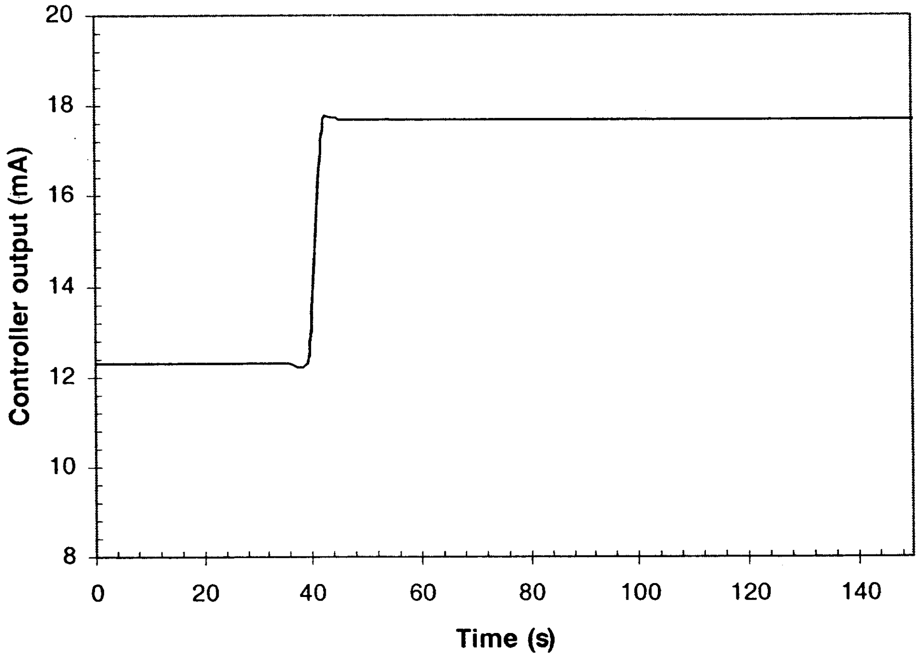


Figure 5.4 Controller output.

The control baseline case presented in section 5.2 was used during the online tuning process. The process responses under various control settings are shown in Figure 5.5. It is noted that P, I, and D are equivalent to K_c, $1/\tau_I$, and τ_D, respectively. The plots in Figure 5.5 especially illustrate the effect an increase in the integral term ($1/\tau_I$) has on minimizing the process offset. The proportional gain K_c of the PI controller was set lower than that of the P controller because the integral control mode makes the system more sensitive which potentially leads to instability (Stephanopoulos, 1984). It was also found that the derivative term τ_D has no significant impact on the controller performance. This is due to the fact that the combustion system is stable and the time delay in the sampling and analysis process is not too long.

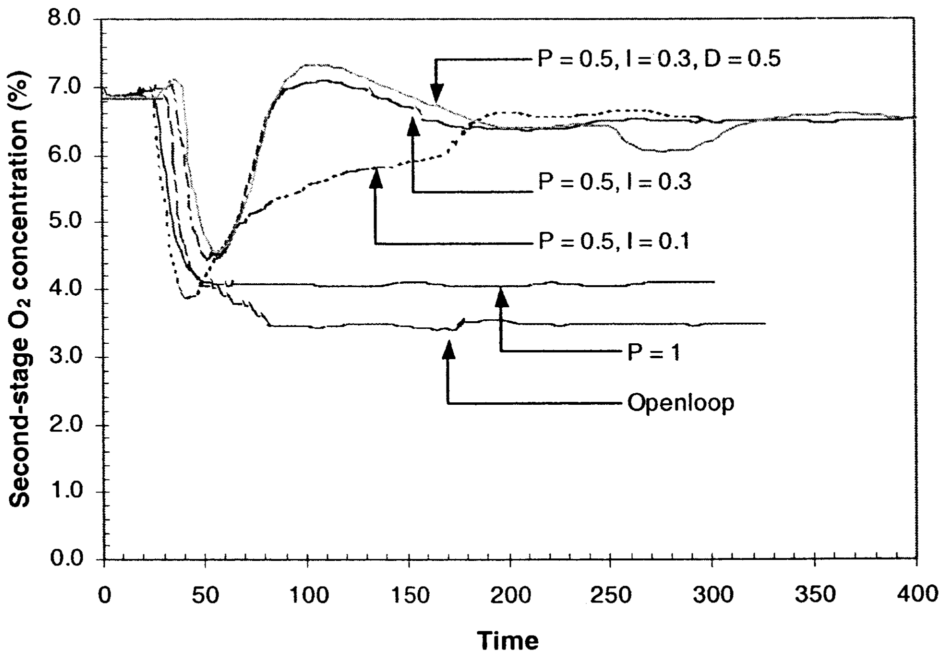


Figure 5.4 Second-stage O_2 responses under various PID settings.

From Figure 5.2, one sees that two inputs are fed into the PID controller. One is a process setpoint, which is the desired O_2 level; the other is the measured O_2 level from the outlet of the second stage of the combustion chamber. In trying to maintain the process at the desired process setpoint, the PID controller computed the control signal that corresponds to the flow of the manipulated variable (primary air). The control signal is based on the information the PID receives and the PID parameter setting. The control signal is then sent to the electronic control valve that adjusts the flow of the manipulated variable (primary air). The performance of the system with the PID control will be illustrated later in this chapter.

5.3.2 A Proportional Neural Network Controller (PNNC)

A proportional neural network controller (PNNC) is a result of combining a trained neural network with a simple proportional controller. The schematic of the PNNC is shown in Figure 5.6. The proportional controller can be described as a function in Equation 5.2, with the integral and derivative terms set to zero (i.e. $1/\tau_I = \tau_D = 0$). Unlike the traditional PID controller, the control bias value $\bar{p}$ in the proposed PNNC configuration can be updated by the trained neural network at every sampling time step, thus the process offset can be minimized.

Because of the monotonic relationship between the second-stage O_2 level and the feed composition (ϕ), the neural network has a simple architecture with one node in the input layer, two nodes in the hidden layer, and a single node in the output layer. The node in the input layer is the measured O_2 concentration. The output node is the process equivalence ratio ϕ.

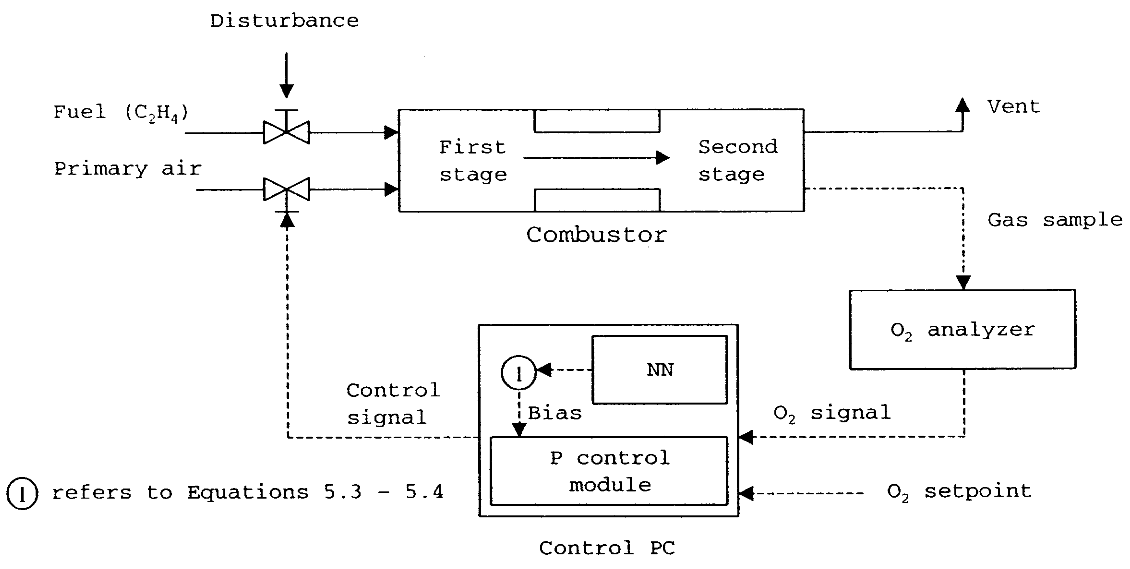


Figure 5.6 PNNC experimental schematic.

The neural network was trained and tested by training and testing data sets, respectively. These consisted of 30 known pairs of input-output values from existing experimental data and CHEMKIN simulation data. The training data is shown in Table 5.1 The first column represents the network input, which is the measured O_2 level. The second column represents the network output, which is the process equivalence ratio ϕ. The last six pairs of samples are used as the testing set. All data were randomized and normalized to –1 to +1 by subroutines in the NeuroSolutions software package before being presented to the neural network. The learning curves for both training and testing sets are shown in Figure 5.7. The learning cost functions (MSE) for both training and testing sets exponentially level out and reach their minimums. Thus, there is no evidence of overtraining. This can be attributed to the fact that the relationship between the O_2 concentration and the equivalence ratio ϕ is fairly simple.

Table 5.1 Neural Network Training and Testing Data (PNNC)

Sample No.	Second-stage O_2 (%)	Equivalence ratio ϕ
1	8.30	0.58
2	7.90	0.58
3	6.20	0.64
4	5.20	0.68
5	4.10	0.72
6	3.60	0.76
7	2.40	0.82
8	0.80	0.90
9	0.20	1.18
10	0.03	1.35
11	8.50	0.58
12	6.40	0.64
13	5.30	0.68
14	4.60	0.72
15	3.00	0.76
16	2.70	0.82
17	1.40	0.90
18	7.50	0.59
19	0.08	1.18
20	3.35	0.76
21	0.00	1.18
22	0.00	1.35
23	7.90	0.58
24	7.69	0.59
25(test)	6.60	0.64
26(test)	5.30	0.68
27(test)	4.23	0.72
28(test)	2.43	0.82
29(test)	1.20	0.90
30(test)	0.07	1.35

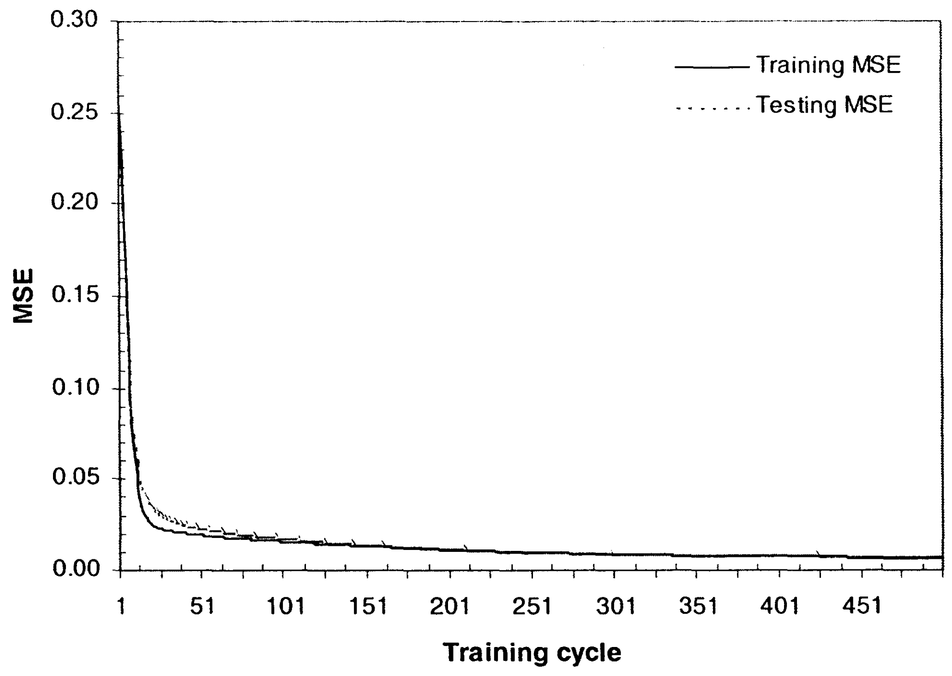


Figure 5.7 Neural network learning curves (PNNC).

From Figure 5.6, one sees that at each sampling time step, two inputs are fed into the proportional controller. These two inputs are the O_2 setpoint and the measured O_2 level from the second stage combustion chamber. The proportional control function adjusts the manipulated variable, in this case the primary air flow rate, in response to the error in the control variable, the O_2 level.

At the same sampling time step, the trained neural network takes part in the control action by identifying the current process equivalence ratio ϕ based on the information it receives, in this case the measured O_2 level. Once the current equivalence ratio ϕ is identified by the neural network, a calculation for the new proportional control bias value is done in the Visual Basic computation routine as described by the following steps:

1. A current fuel flow rate is computed, based on the following modified equivalence ratio relationship:

$$Fuel = \phi \cdot Air \cdot \left(\frac{Fuel}{Air}\right)_{stoichiometric} \tag{5.3}$$

where Air = current known primary air flow rate

ϕ = current equivalence ratio obtained from the neural network.

2. A new control bias value, corresponded to a new primary air flow rate, is computed, based on the following modified equivalence relationship:

$$Air = \frac{Fuel}{\phi_{setpoint} \cdot \left(\frac{Fuel}{Air}\right)_{stoichiometric}} \tag{5.4}$$

where Fuel = current fuel flow rate obtained in the previous step

$\phi_{setpoint}$ = desired equivalence ratio that yields the desired O_2 level.

Once the new proportional control bias value is obtained and updated, the final control signal output can be calculated using Equation 5.2. The control signal output is then sent to the electronic control valve, which adjusts the primary air flow rate.

Since there are no integral and derivative terms involved in the tuning process, tuning the PNNC is much simpler than tuning the PID controller. The optimal proportional gain K_c, obtained from the previous section, is used in the PNNC structure. Here, the K_c value becomes less detrimental to the control performance. This is due to the fact that the variable control bias value $\overline{p}$ dictates the controller output.

Unlike the setting in the traditional PID controller, the control time interval for the PNNC has to be set equal to the average time-delay in the sampling and analysis process

(40 seconds) at the very minimum, in order to ensure that the neural network receives the true feedback value from the combustion system.

5.4 Experimental Runs and Results

As mentioned earlier, the control baseline was to first set ϕ at 0.63, which yielded the O_2 level at about 6.8%. This value was then used as the process setpoint. A step disturbance was then manually applied to the fuel flow rate (C_2H_4) and caused the equivalence ratio to change to 0.76. From Figure 5.1, this would yield about 3.4% of O_2 in the combustor outlet for the open-loop run. The control objective is to bring the O_2 level back to where it was before the disturbance.

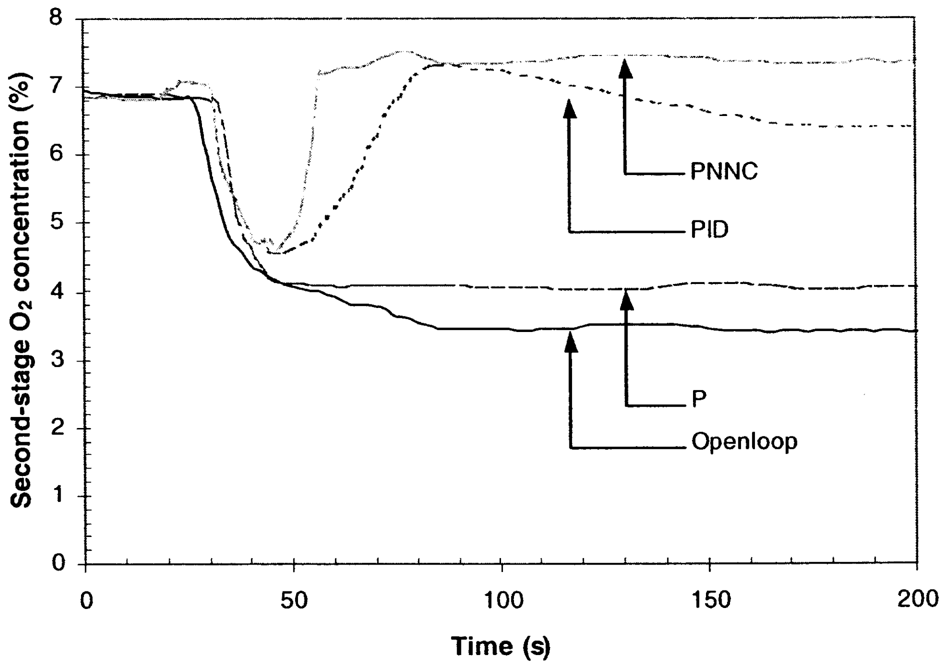


Figure 5.8 Open-loop and closed-loop second-stage O_2 response.

The second-stage O_2 profile under the PNNC is shown in Figure 5.8. As a basis of comparison, the open-loop experimental results as well as the process responses under P and PID controller, obtained earlier in section 5.3.1, are illustrated here as well. Although the proportional controller is the simplest control configuration, Figure 5.8 once again confirms the inherent disadvantage of the proportional controller – its inability to eliminate the sustained error or offset. The PID and PNNC controllers yield a promising result in this study. The PNNC was able to identify the changes in the fuel feed flow rate and effectively took part in the control action by performing the simple O_2-to-ϕ mapping and updating the control bias value. Although the offset was not completely eliminated, the PNNC shows a considerable improvement of setpoint tracking and process settling time over the conventional proportional controller. The process offset can be attributed to several sources, most prominent of which is the error caused by the electronic control valve. The electronic control valve is very sensitive to the pressure and becomes inaccurate over time especially when the control signal changes direction. Such phenomena severely affected the controller performance. Nevertheless, the neural network approach is proven to be effective in improving the simple-to-tune proportional controller performance without any drastic change in the control configuration.

5.5 Conclusions

Although tuning the PID controller is a time-consuming process, the PID controller is shown to be a robust technique for a monotonic control problem – the variation of outlet oxygen level with overall equivalence ratio (ϕ). In this study, the PID controller was initially tuned by the process reaction curve method, followed by the online fine-tuning process. The inherent disadvantage of the proportional controller – its inability to eliminate the process offset, is shown experimentally. The PNNC represents the use of the FMLP neural network in conjunction with the simple proportional controller. Without the tuning complication and drastically changing the control configuration, the experimental results show significant performance improvement of the proportional controller when the neural network is applied as an intelligent tool for updating the proportional controller bias value.

CHAPTER 6

FEEDFORWARD-FEEDBACK PROCESS CONTROL

6.1 Introduction

To facilitate process control and overcome the problems associated with the traditional PID controller, a model-based control technique is an alternative control strategy that has received widespread attention. In the last decade, schemes that use an explicit predictive model to predict future process outputs and calculate optimal control movements have been widely studied. They have been used to control complex nonlinear processes in chemical engineering fields.

Typically, a process model is based on conservation laws (mass, species, energy, and momentum). Such a model often turns out to be too complex for practical implementation as a controller. To simplify such nonlinear problems, linearized models and some simplifying assumptions are often used. In the event that the nonlinearity is not severe, the linearized approaches can provide adequate performance. However, if the process is a highly nonlinear, or is driven away from the linearized region, the linear controller performance can be poor or even fail to keep up with perturbations or process shifts.

An alternative to a traditional conservation law-based model is an approach using artificial neural networks (ANNs). The links between neural networks, dynamic process models, and process control are provided by the concept of model-based control. Here, the neural network is used in place of the traditional process model. The neural network model-based control approach has become increasingly popular in chemical engineering.

Many chemical processes exhibit nonlinear behavior, where relationships between the controlled variable and the manipulated variable are too complicated to model by the traditional modeling methods. Also, the active control of most chemical processes is often complicated by the process dynamics, lags, and other real factors associated with fluid mechanics, and time delay in sampling and species analyses. In trying to incorporate these time-varying behaviors into a classical control theory, the traditional Smith time-delay compensation method can be used to construct controllers when the transfer function of the system is known. However, the transfer functions of most chemical processes are too complex to be incorporated into a traditional control algorithm.

The neural networks used in most control schemes are feedforward multi-layer-perceptron (FMLP). The most common approach to incorporating the process dynamics, process lags, and other real factors into the process controller, by far, is to carefully design a time-history window of the process inputs and outputs from a period of time equal to the delay in the process and then input them into the feedforward neural network (Bhat and McAvoy, 1990; Gomm et al., 1997; Nikravesh et al., 2000; Palancar et al. 1998; Syu and Chen, 1998; Tendulkar et al. 1998). This technique is only possible when there are discrete-time training data available.

In this chapter, feedforward multi-layer-perceptron (FMLP) neural networks are used in a model-based control approach to control a two-stage laboratory combustor operating in a mode wherein the process variable under control is a nonlinear function of a key variable. A process model, based on trained neural networks, is used as (1) a controller in a conventional feedback control configuration and (2) a process emulator in a Smith time-delay configuration to compensate for a significant sampling/analysis time delay. The objective of this study is to experimentally demonstrate a control system based on static back-propagation trained neural networks. The originality of this work lies in the structure and the functionalities of the two clusters of FMLP neural networks. The control configuration based on the two clusters of FMLP neural networks represents a feedforward-feedback control scheme. It combines advantages of the conventional feedforward and feedback control (Stephanopoulos, 1984). The first cluster serves as the process controller in the conventional feedback control configuration while the second facilitates the feedforward process control by serving as the time-delay compensator. The proposed neural-network-based control structure enables the networks to be statically trained offline with existing steady state experimental data. A Visual Basic control interface enables the statically trained neural network to transiently respond to the process.

6.2 Experimental Outlines

The stated control objective is to control NO emission from the two-staged combustor. Ethylene (C_2H_4) serves as the main fuel. Ammonia (NH_3), which serves as the secondary fuel, is used as a model dopant to simulate fuel-bound nitrogen, which produces NO. Primary and secondary airs are the oxidizer. The fuel/air mixture compositions in the primary stage and overall combustor are characterized by the equivalence ratio ϕ_1 and ϕ_o, given as follows:

$$\phi_1 = \frac{\left(\dfrac{F_1 + F_2}{A_1}\right)_{actual}}{\left(\dfrac{F_1 + F_2}{\frac{3}{0.21}F_1 + \frac{0.75}{0.21}F_2}\right)_{stoichiometric}} \tag{6.1}$$

$$\phi_o = \frac{\left(\dfrac{F_1 + F_2}{A_1 + A_2}\right)_{actual}}{\left(\dfrac{F_1 + F_2}{\frac{3}{0.21}F_1 + \frac{0.75}{0.21}F_2}\right)_{stoichiometric}} \tag{6.2}$$

where ϕ_1 = first-stage equivalence ratio

ϕ_o = overall equivalence ratio

F_1 = volumetric or molar flow rate of C_2H_4

F_2 = volumetric or molar flow rate of NH_3

A_1 = volumetric or molar flow rate of primary air

A_2 = volumetric or molar flow rate of secondary air

The fuel-bound nitrogen in the feed can be characterized by the ammonia dopant ratio, given by:

$$Ammonia \quad dopant \quad ratio = \frac{F_2}{F_1} \tag{6.3}$$

where F_1 = volumetric or molar flow rate of C_2H_4

F_2 = volumetric or molar flow rate of NH_3

The purpose of the controller is to adjust the first-stage equivalence ratio ϕ_1 and maintain the overall equivalence ratio ϕ_o in order to keep the emission levels of NO and CO below the desired levels. The detailed reaction mechanism and the relationship between NO, CO, and first and overall equivalence ratios (ϕ_1 and ϕ_o) are described elsewhere (Mao, 1995; Mao and Barat, 1996).

At a selected steady state, a step increase disturbance of the ammonia (NH_3) is manually made. This step disturbance causes changes in the NO levels in the combustor outlet. After a step disturbance, the controller, which receives signals from an online chemilumisnscence NO analyzer, takes immediate action by computing and sending control signals to the manipulated variables – the primary and secondary air flows, and brings the control variable (NO) to the setpoint. The long time delay in the NO gas sampling and analysis system is long, so it complicates the control problem and has a severe impact on the control performance.

In order to establish a control experimental baseline, a series of open-loop runs were first made to examine the effects of the ammonia dopant and the first-stage equivalence ratio (ϕ_1) on the NO level. An overall fuel-lean baseline (ϕ_o=0.9) was set. The feed equivalence ratio ϕ_1 was sequentially increased from 1.15 to 1.45. The fuel-bound nitrogen in the feed was characterized by the feed molar ratio NH_3/C_2H_4 at values of 0.022, 0.027, 0.057, and 0.078. Diluent N_2 was used to keep the first-stage temperature within the safety-operating limit (less than 1673 K). Figure 6.1 shows the nonlinear relationship between the fuel-bound nitrogen dopant in the feed (NH_3/C_2H_4 on a molar basis); the first-stage equivalence ratio (ϕ_1); and the second-stage NO levels at constant ϕ_o = 0.9. Consistent with Mao and Barat (1996), emission of NO from the second stage outlet is minimized if ϕ_1 is kept at about 1.35. However, this minimum lies at the bottom of an approximately parabolic well. In trying to control ϕ_1, the classical PID controller can fail. For example, the integral term (refer to Equation 5.2) can give rise to a build-up of process error, causing the controller to fail in the saturated mode when the process reaches the minimum point of the response curve. In other words, the control signal could continue to change monotonically due to the accumulation of error in the integral term, even though it should "reverse direction," i.e., the sign of its derivative should change.

A control baseline is to initially set the first-stage equivalence ratio ϕ_I at 1.14 and overall equivalence ratio ϕ_o at 0.9. With the ammonia dopant of 0.027, the measured NO concentration from the second stage is 460 ppm. This number becomes the process setpoint. A step disturbance is then manually applied to the NH_3 flow rate to raise the dopant ratio to 0.057. From Figure 6.1, one sees that the open-loop NO level rose to 620 ppm. The control objective is to bring the NO level back to the setpoint by manipulating the primary air and secondary air flow rates.

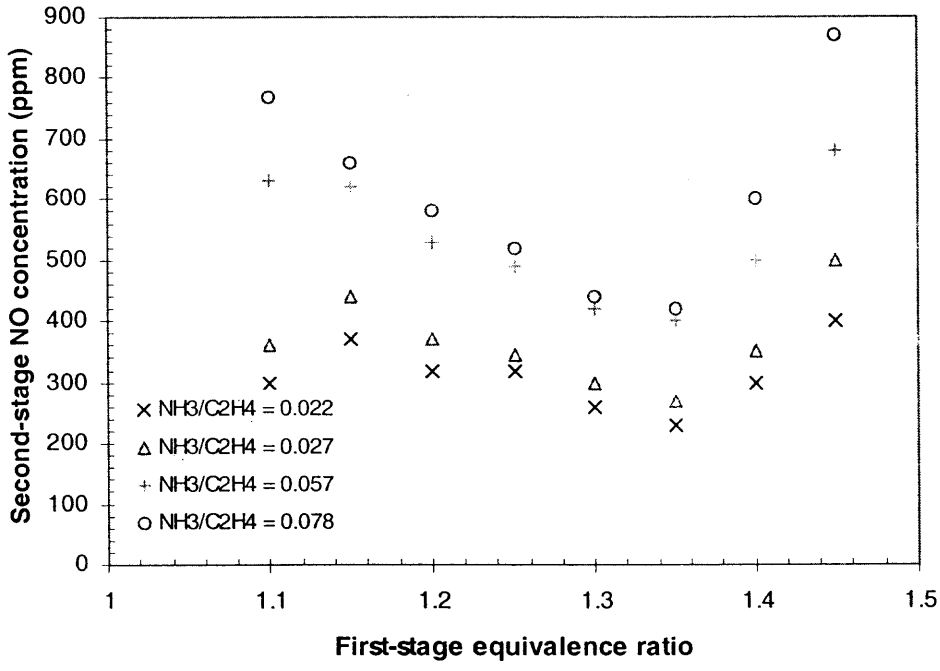


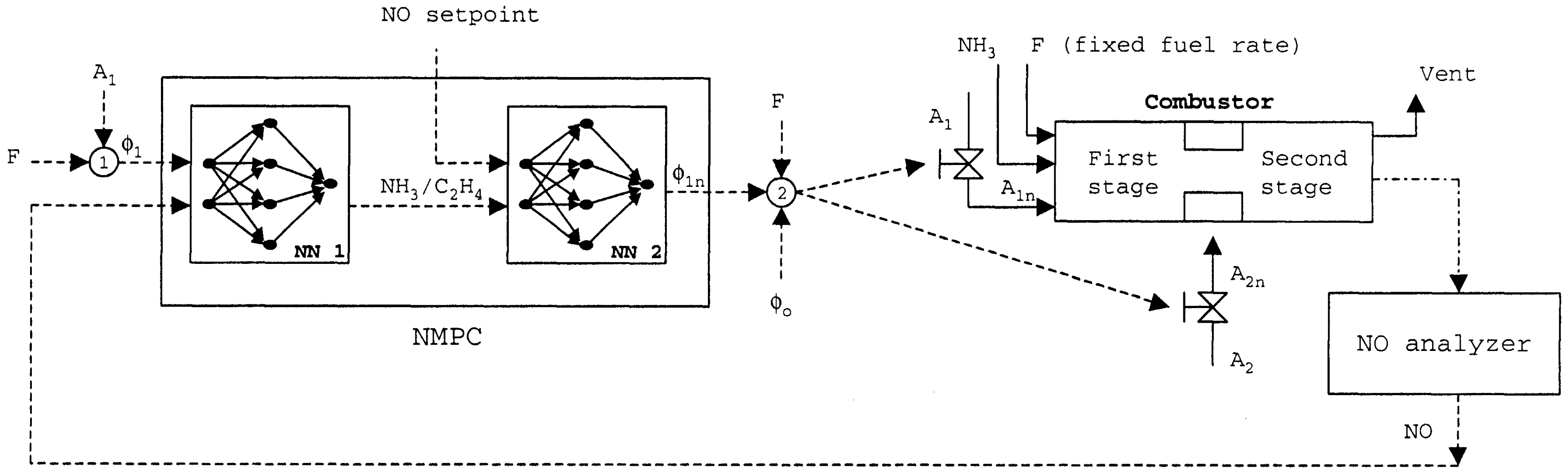
Figure 6.1 Second-stage NO concentrations at various steady states ($\phi_o = 0.9$).

6.3 Experimental Setup

6.3.1 Neural Network-Based Model-Predictive Controller (NMPC)

The experimental setup for the neural network-based model-predictive controller (NMPC) is shown in Figure 6.2. The NMPC is based on a cluster of two FMLP neural networks connected in series. Both neural networks have the same architecture, two nodes in the input layer, four nodes in the hidden layer, and one node in the output layer.

The first neural network serves as the process identifier. It identifies the relative amount of unknown fuel-bound nitrogen dopant in the feed stream based on the information it receives – the current ϕ_1 and the measured NO. The neural network was trained and tested using 30 known input-output pairs of existing experimental data (Figure 6.1). It should be noted that two data points in Figure 6.1 were purposely removed and not included in the neural network training set. Based on the knowledge gained from the previous study (Mao, 1995), the reported second-stage NO concentrations of 300 and 360 ppm at $\phi_1 = 1.1$ and ammonia dopant ratios of 0.022 and 0.027 respectively, were not accurate. This can be attributed to systematic error in the sampling and analysis system. These inaccurate data were considered as the noisy data patterns that could potentially deteriorate the network generalization performance as earlier described in Chapter 4. In this study, a full advantage of knowing the process was taken, which in turn, allowed the raw training data to be visually examined and pre-filtered. This approach however cannot be applied for systems that are inherently more complicated. In particular, the system that has more than a three-dimensional input-output relationship or the system that requires many training patterns that beyond the capability of the network user to visually characterize (pre-filter) based on the previously gained knowledge.



A_1, A_{1n} = primary air flow rate
A_2, A_{2n} = secondary air flow rate
F = C_2H_4 flow rate
ϕ_1, ϕ_{1n} = 1st stage equivalence ratio
ϕ_o = overall equivalence ratio
NH_3 = ammonia flow rate
NO = concentration of measured NO
NH_3/C_2H_4 = ammonia dopant ratio

——————— actual flow (gas)
- - - - - - - - - signal flow
-·-·-·-·-·-·- sample flow (gas)

(1) refers to Equation 6.1 (2) refers to Equations 6.1 and 6.2

Figure 6.2 NMPC experimental schematic.

The training data is shown in Table 6.1. The second and third columns represent the network inputs, which are the first-stage equivalence ratio ϕ_1 and the measured NO level. The fourth column represents the network output, which is the ammonia dopant ratio in the feed. The last five rows serve as a testing set. All data are normalized by the subroutine in the NeuroSolutions software package before presentation to the neural network breadboard.

As mentioned earlier, the diluent N_2 was used to keep the first-stage temperature below 1673 K. The use of the diluent N_2 could affect the combustion-related effluent concentrations. However, it appeared in this study that the diluent N_2 did not have any significant impact on the control variables (NO and CO_2) or the control performance. This can be attributed to the fact that the first-stage equivalence ratio (ϕ_1) was exercised in a small range ($1.15 < \phi_1 < 1.3$) throughout the control experimental program. As a result, the recorded first-stage temperature only fluctuated in a narrow range (less than 20 K), which in fact did not pose any potentially hazardous situation nor require much adjustment in the diluent N_2 flow rate. In contrast, if the control variables become strongly first-stage temperature dependent, the reactor temperature information (i.e. diluent N_2 flow rate vs. first-stage equivalence ratio ϕ_1) should be added into the neural network input as an additional input vector.

Table 6.1 Neural Network Training and Testing Data (NN#1 in NMPC)

Sample No.	ϕ_1	Second-stage NO (ppm)	NH_3 dopant ratio
1	1.15	440	0.027
2	1.20	530	0.057
3	1.20	318	0.022
4	1.15	660	0.078
5	1.30	440	0.078
6	1.25	490	0.057
7	1.40	650	0.078
8	1.35	270	0.027
9	1.20	580	0.078
10	1.30	300	0.027
11	1.40	530	0.057
12	1.10	770	0.078
13	1.25	320	0.022
14	1.45	914	0.078
15	1.15	620	0.057
16	1.30	420	0.057
17	1.45	683	0.057
18	1.35	420	0.078
19	1.25	345	0.027
20	1.30	260	0.022
21	1.35	230	0.022
22	1.35	400	0.057
23	1.45	400	0.022
24	1.45	500	0.027
25	1.40	300	0.022
26 (test)	1.40	350	0.027
27 (test)	1.20	370	0.027
28 (test)	1.15	370	0.022
29 (test)	1.10	630	0.057
30 (test)	1.25	520	0.078

The learning curves are shown in Figure 6.3. The minimum mean square errors (MSE) for both training and testing sets constantly decrease and reach their minimums. Hence there is no overtraining effect. This outcome can be attributed to the fact that the training and testing data used in this study were visually examined and pre-filtered such that any process noise that might cause detrimental impacts to the network generalization performance was purposely removed from the training set. As a result the network training converges readily with no evidence of the overtraining.

The first neural network output is fed into the second neural network (NN#2) along with the desired NO setpoint. The second network is a predictive controller. It takes control action by computing the new ϕ_1 for the next time step aimed to yield the desired NO setpoint and corresponding to the predicted fuel-bound nitrogen dopant level in the feed.

The second neural network was trained and tested using 13 known input-output pairs from existing experimental data and some interpolations of the existing experimental data. The training data are shown in Table 6.2. The second and third columns represent the network inputs, which are the desired NO setpoint and the relative nitrogen-bound dopant level. The fourth column represents the network output, which is the first-stage equivalence ratio ϕ_1. Again, all data are normalized to -1 to $+1$. The training procedure is fairly simple since only two NO target setpoints were included in the training set. Three out of thirteen pairs of known input-output were held out for using as a testing set. The same conclusion for the learning curves shown in Figure 6.4 can be drawn from the previous sections.

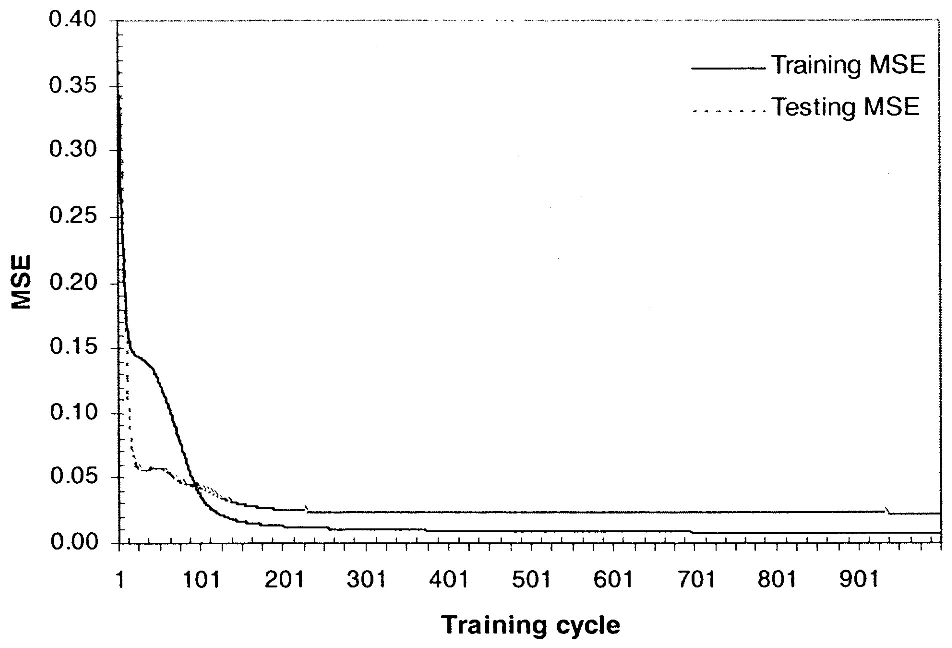


Figure 6.3 Neural network learning curves (NN#1 in NMPC).

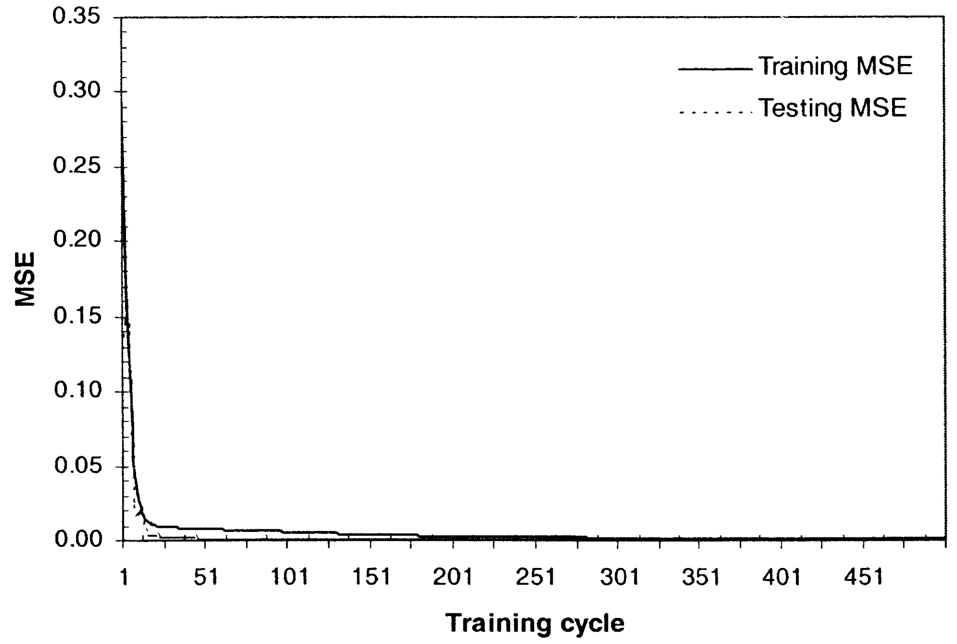


Figure 6.4 Neural network learning curves (NN#2 in NMPC).

Table 6.2 Neural Network Training and Testing Data (NN#2 in NMPC)

Sample No.	NO Setpoint (ppm)	NH_3 dopant ratio	ϕ_1
1	440	0.032	1.18
2	500	0.058	1.24
3	500	0.078	1.26
4	500	0.048	1.20
5	440	0.035	1.20
6	440	0.041	1.24
7	440	0.057	1.28
8	500	0.043	1.16
9	440	0.027	1.15
10	500	0.038	1.14
11 (test)	500	0.045	1.18
12 (test)	440	0.045	1.26
13 (test)	440	0.035	1.22

The two neural networks function in series in the Visual Basic platform application. All the data inputted into and outputted from the neural networks are displayed on the Visual Basic operating interface. Once the next first-stage equivalence ratio ϕ_1 is found from the second neural network, the primary and secondary air flow rates are computed in the Visual Basic computation routine. A detailed-sequence of the calculations can be described by the following steps:

1. A relative nitrogen-bound dopant is identified by the first neural network, based on the current ϕ_1 and the measured second-stage NO concentration.
2. The relative nitrogen-bound dopant, identified by the first neural network, and the user-input NO setpoint, are fed into the second neural network.
3. The new ϕ_1, corresponding to the predicted relative ammonia dopant ratio and the desired NO setpoint, is computed by the second neural network.

4. The ammonia flow rate (F_2) is computed, based on the nitrogen-bound dopant relationship (Equation 6.3).
5. The new primary air flow rate (A_1) is computed, based on the first-stage equivalence ratio relationship (Equation 6.1), where F_1, F_2 and ϕ_1 are known.
6. The secondary air flow rate (A_2) is computed, based on the overall equivalence ratio relationship (Equation 6.2), where F_1, F_2, ϕ_o, and A_1 are known.
7. The primary and secondary air flow rates are normalized to the current scale (4-20 mA) and outputted to the electronic control valves.

6.3.2 Neural Network-Based Smith Time-Delay Compensator (NSTC)

In this current work, there is a significant time lag (~240 seconds) between a change in the process and the reported change in the second-stage NO concentration. This is due to a long sampling path and a slow NO analysis process. An effective strategy to reduce the impact on control process of such a time lag is the Smith predictor technique (Seborg et al., 1989; Stephanopoulos, 1985).

The block diagram of a "generic" Smith predictor is shown in Figure 6.5. The actual process model is divided into two parts: the process model without a time delay (PM), and the time delay term (DM). At each cycle of the control output computation, the process model (PM) is used to predict the effect of the control action on the process output. The output C_1 from the process model (PM) is fed into the time delay term (DM) to incorporate the process time delay into the C_2. The delay process model response C_2 is compared with the actual process output C. The difference (C - C_2) is the correction term used to incorporate any modeling errors and load disturbances into the controller output. The controller utilizes the predicted response C_1 and the correction term (C - C_2) to calculate its output for the next time step. That is, if the process model were perfect with no load disturbance, $C = C_2$ and $E' = R - C_1$, where R is the process setpoint. In this case, the controller output is based on the error signal that would occur if no time delay were present. In classical control theory the Smith technique can be used to design controllers when the transfer function of the system is known. But the transfer function of a practical system is not easy to measure or model.

A cluster of two feedforward neural networks with a pre-assigned memory buffer in the Visual Basic operating interface serves as the actual process model in this study. It represents the Smith time-delay compensator, and is incorporated into the existing feedback control loop. The Smith time-delay compensator enabled the feedforward process control. The feedforward control is especially advantageous for slow systems or systems with significant dead time (Stephanopoulos, 1984). A modified NMPC incorporating the neural network-based Smith time-delay compensator (NSTC) is shown in Figure 6.6. The combined NMPC-NSTC falls into the feedforward-feedback control configuration. The NSTC is divided into a process model without time delay (NN#3), referred to as the current network, and a process model with the time delay (NN#4 plus the pre-assigned memory buffer), called the delay network. Both neural networks are identical, with two nodes in the input layer, four nodes in the hidden layer, and one node in the output layer. The network inputs are ϕ_1 and fuel-bound nitrogen ratio in the feed. Both networks output the second-stage NO emission. The only difference between the two networks is that the current neural network (NN#3) utilizes the current $\phi_{1,t}$ (predicted by the NMPC cluster, ϕ_{1n}) while the delay neural network (NN#4) utilizes the delayed $\phi_{1,t-d}$ from the memory buffer (ϕ_1'). This approach allows the delay network to detect the variation in the load disturbance (NH_3 dopant). The pre-assigned Visual Basic buffer is filled by an array of the values of ϕ_1 for the past d seconds ($\phi_{1,t}$, $\phi_{1,t-1}$, $\phi_{1,t-2}$, ..., $\phi_{1,t-d}$) to emulate a transfer function of the actual average dead time in the sampling line. For each time step of control output computation, the delayed $\phi_{1,\ t-d}$ is utilized by the delayed network (NN#4). Finally, the buffer is updated by time shifting, i.e., the value of $\phi_{1,t-i}$ becomes the new value for $\phi_{1,t-i-1}$ and the current value of ϕ_1 is entered for $\phi_{1,t}$.

Both neural networks were trained and tested using 30 known pairs of input-output data from existing experimental data shown in Table 6.3. The second and third columns represent the network inputs, which are the first-stage equivalence ratio ϕ_1 and the relative nitrogen-bound dopant ratio (NH_3/C_2H_4). The fourth column represents the network output, which is the second-stage NO concentration. The last five rows serves as the testing set. All data are normalized to –1 to +1 before presentation to the neural networks.

The learning curves are shown in Figure 6.7. The minimum mean square errors (MSE) for both training and testing sets constantly decrease and level out. The network converges easily with no overtraining effect. The same conclusion can be drawn from previous network training process and the discussion in Chapter 4.

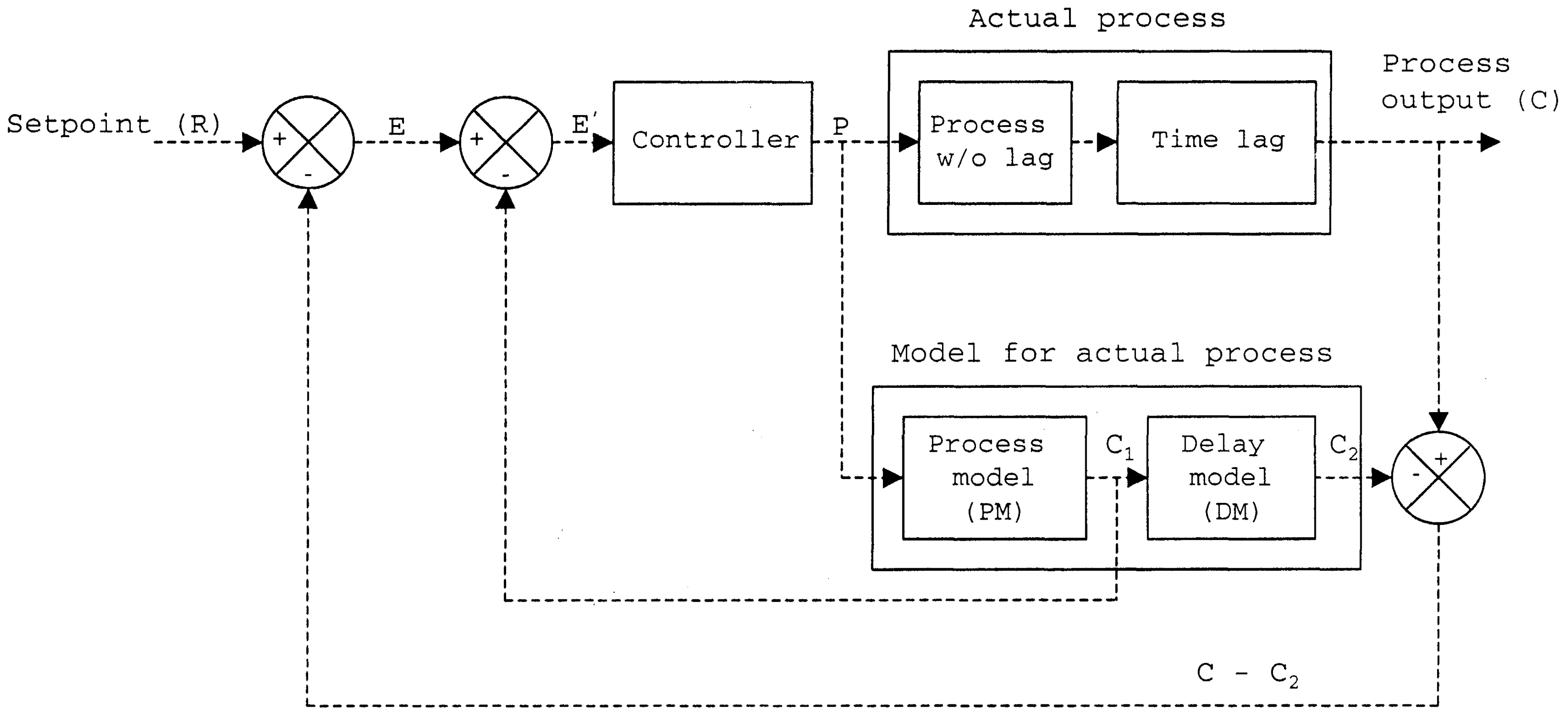


Figure 6.5 Smith time-delay compensator configuration.

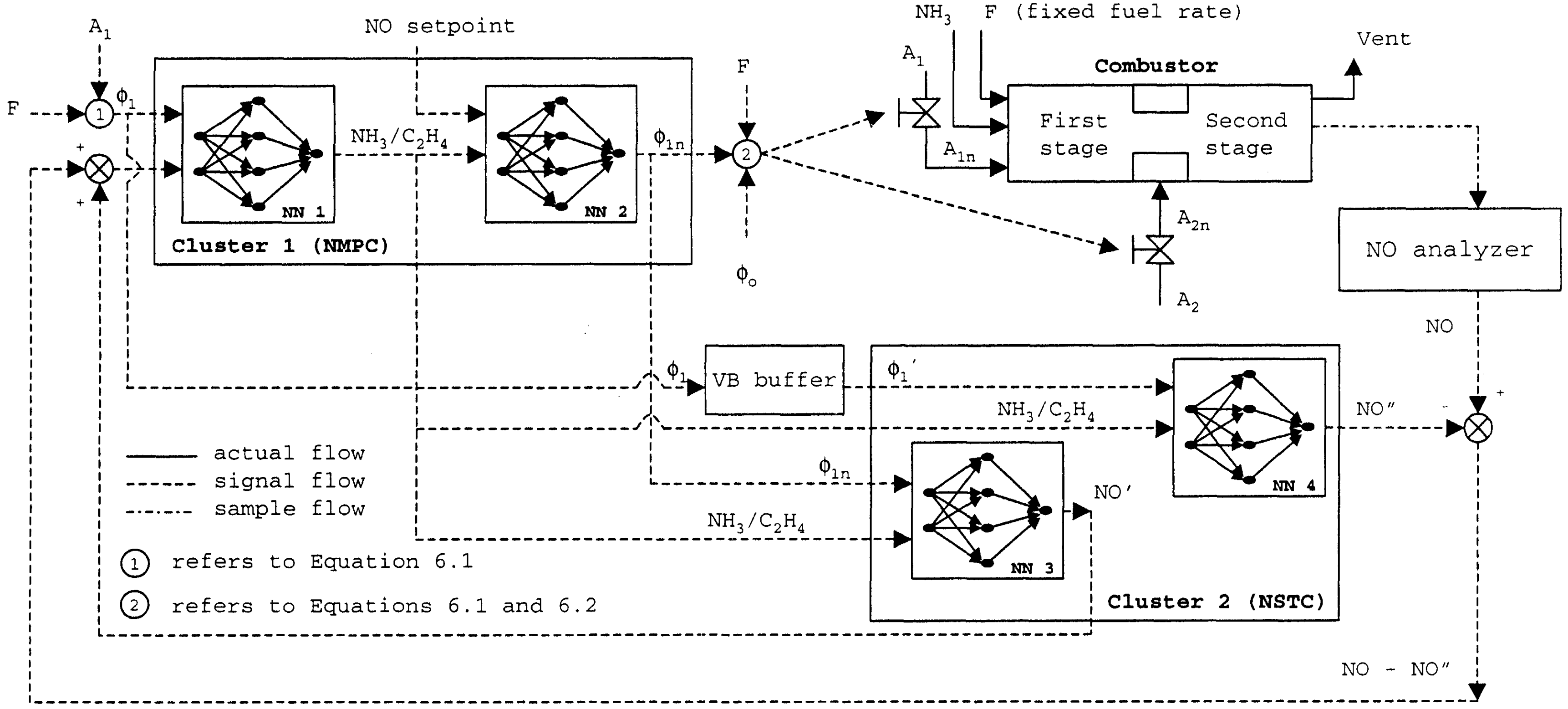


Figure 6.6 NMPC/NSTC experimental schematic.

From Figure 6.6, the NMPC cluster uses the predicted NO concentration (NO′) from NN#3 to ultimately calculate the manipulated variable, which is the primary air flow rate (A_1). Since the predicted NO concentration (NO″) from NN#4 is already delayed by the memory buffer, it is compared to the measured NO concentration from the actual process (NO). The difference between NO″ and the actual NO is directed to the controller in order to incorporate any modeling errors into the controller. That is, if the process model were perfect and there were no load disturbance, then NO - NO″ = 0. It is noted that the network weights for the networks in the NMPC cluster (NN#1 and NN#2) remain unchanged while incorporating the NSTC cluster (NN#3 and NN#4).

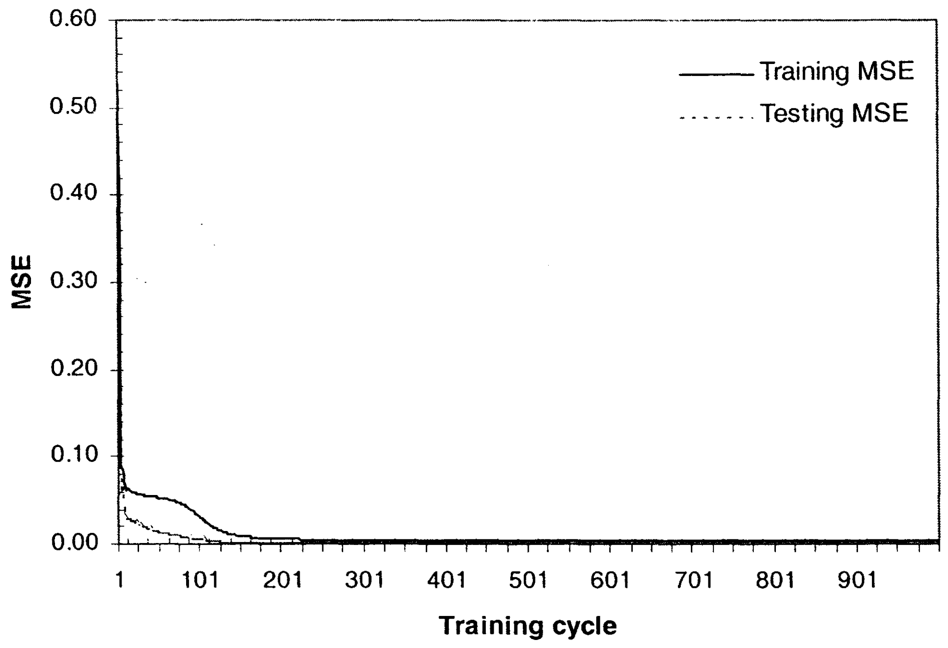


Figure 6.7 Neural network learning curves (NN#3 and NN#4 in NSTC).

Table 6.3 Neural Network Training and Testing Data (NN#3 and NN#4 in NSTC)

Sample No.	ϕ_1	NH_3 dopant ratio	Second-stage NO (ppm)
1	1.15	0.027	440
2	1.20	0.057	530
3	1.20	0.022	318
4	1.15	0.078	660
5	1.30	0.078	440
6	1.25	0.057	490
7	1.40	0.078	650
8	1.35	0.027	270
9	1.20	0.078	580
10	1.30	0.027	300
11	1.40	0.057	530
12	1.10	0.078	770
13	1.25	0.022	320
14	1.45	0.078	914
15	1.15	0.057	620
16	1.30	0.057	420
17	1.45	0.057	683
18	1.35	0.078	420
19	1.25	0.027	345
20	1.30	0.022	260
21	1.35	0.022	230
22	1.35	0.057	400
23	1.45	0.022	400
24	1.45	0.027	500
25	1.40	0.022	300
26	1.40	0.027	350
27	1.20	0.027	370
28	1.15	0.022	370
29	1.10	0.057	630
30	1.25	0.078	520

6.4 Experimental Runs and Results

The two experimental programs included both open-loop and closed-loop runs, exercised with two different sets of setpoints and disturbances. The desired second-stage NO concentration was set at 460 ppm for the first experimental program. Figures 6.8 and 6.9 show the first experimental results. The initial ϕ_l was 1.14, while ϕ_o was set at 0.9. The initial dopant/fuel ratio in the feed was 0.027. The measured NO concentration from the second stage was 460 ppm. Based on Figure 6.1, it should be noted that, at this NH_3 dopant level, the second-stage NO concentration could be made lower than 460 ppm by increasing the first-stage equivalence ratio ϕ_l. The reason for not doing so is that the CO burnout rate in the second stage will not be sufficient to achieve the low level of CO emissions if ϕ_l is too high.

A step disturbance was then applied to the NH_3 flow rate ($t = 0$ in Figure 6.8) to raise the dopant ratio to 0.057. From Figure 6.8, the open-loop NO level rose to 620 ppm. The closed-loop NO level is shown in Figure 6.9. The total time lag between process change on recorded NO is about 240 seconds. In closed-loop response without lag compensation (Figure 6.2 scheme), the NMPC ultimately increased ϕ_l to 1.3 by reducing the primary air flow rate (A_1) in order to bring the NO back to the setpoint (460 ppm). The secondary air flow rate (A_2) was simultaneously increased in order to maintain ϕ_o. The control action was set to execute every 240 seconds to facilitate the observed time delay due to the long gas sampling line and the slow NO analyzer response. This action ensured that the NMPC received the true value of the feedback signal from the NO analyzer. The NO level achieved the setpoint after about 1500 seconds.

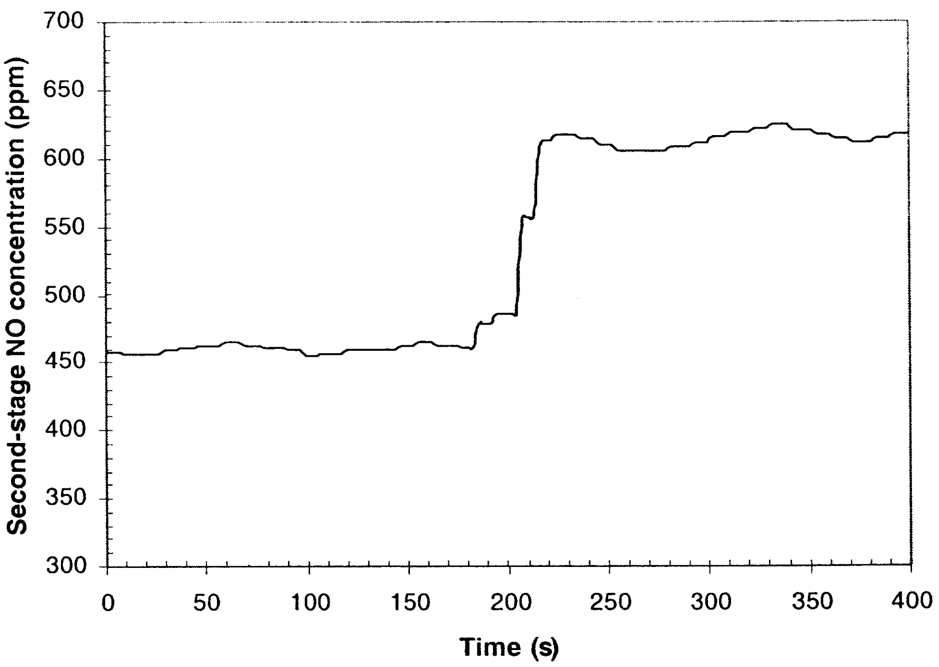


Figure 6.8 Open-loop second-stage NO response (experiment 1).

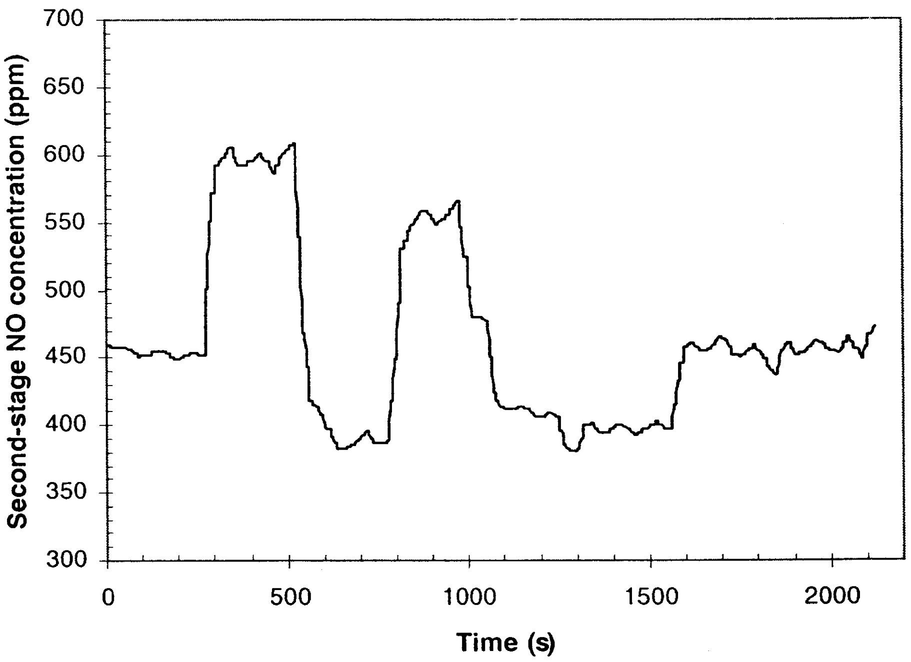


Figure 6.9 Closed-loop second-stage NO response (experiment 1).

To confirm that the overall equivalence ratio ϕ_o was simultaneously brought back to the setpoint ($\phi_o = 0.9$) by adjusting the secondary airflow rate (A_2), the CO_2 level from the second stage was measured. Before the disturbance, the second-stage CO_2 level at the second stage outlet was 8.8%. As shown in Figure 6.10, the CO_2 level varied in a small range and settled slightly higher (about 9.25%) after some fluctuation caused by the changes in the primary and secondary airflow rates.

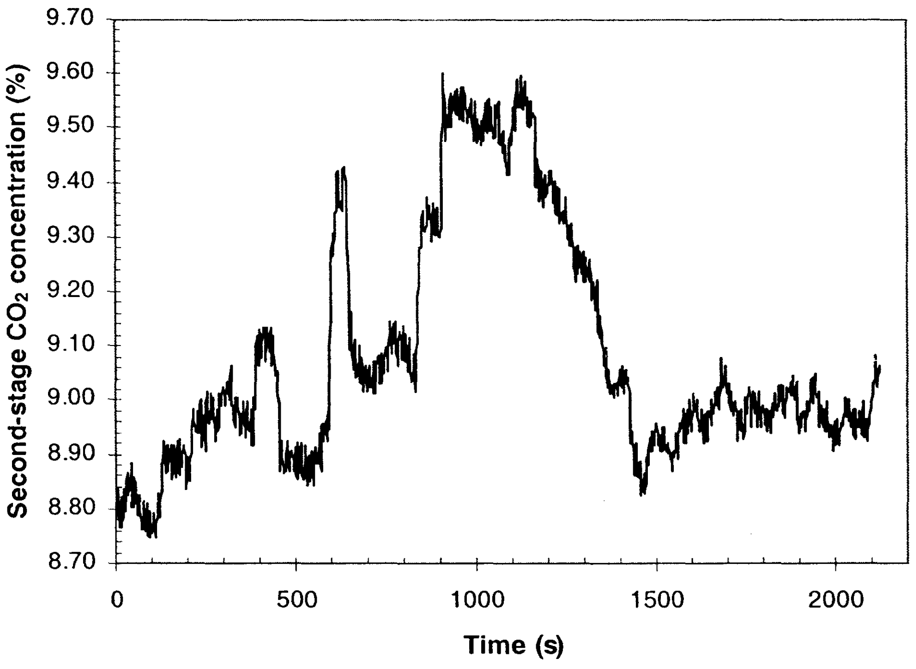


Figure 6.10 Closed-loop second-stage CO_2 response (experiment 1).

The second experimental program was to set the desired second-stage NO concentration at 500 ppm. Figures 6.11 – 6.13 show the experimental results. The overall equivalence ratio remained unchanged at 0.9. The initial ϕ_1 was set at 1.15, with initial dopant/fuel ratio at 0.035. This setting result in 500 ppm of second-stage NO concentration. This value was then used as the process setpoint. A step disturbance was then applied to the NH_3 flow rate ($t = 0$ in Figure 6.11) to raise the dopant ratio to 0.078. From Figure 6.11, the open-loop NO level rose to 660 ppm. The closed-loop NO level is shown in Figure 6.12. Unlike the first experimental program, the second-stage NO concentration settled at 440 ppm, 12% lower than the setpoint. Based on the observation during the experiment, the most prominent factor that caused the process offset was an inaccuracy of the two electronic control valves. The electronic control valves were sensitive to the pressure and became inaccurate over time especially when the control signal reversed direction. This phenomenon severely affected the controller performance as clearly shown by the second-stage NO responses in Figure 6.12. While NMPC provided control signals to adjust the manipulated variable, the electronic control valves did not response to the control signal precisely. The errors between the control signals and the control valve movement were continuously accumulated and fed into the NMPC, thus deteriorating the control performance as time advanced. The second-stage CO_2 level was measured and shown in Figure 6.13. Consistent with the first experimental program, the second-stage CO_2 profile has an upward trend (over time), which confirms that the overall burnout rate in the reactor was achieved.

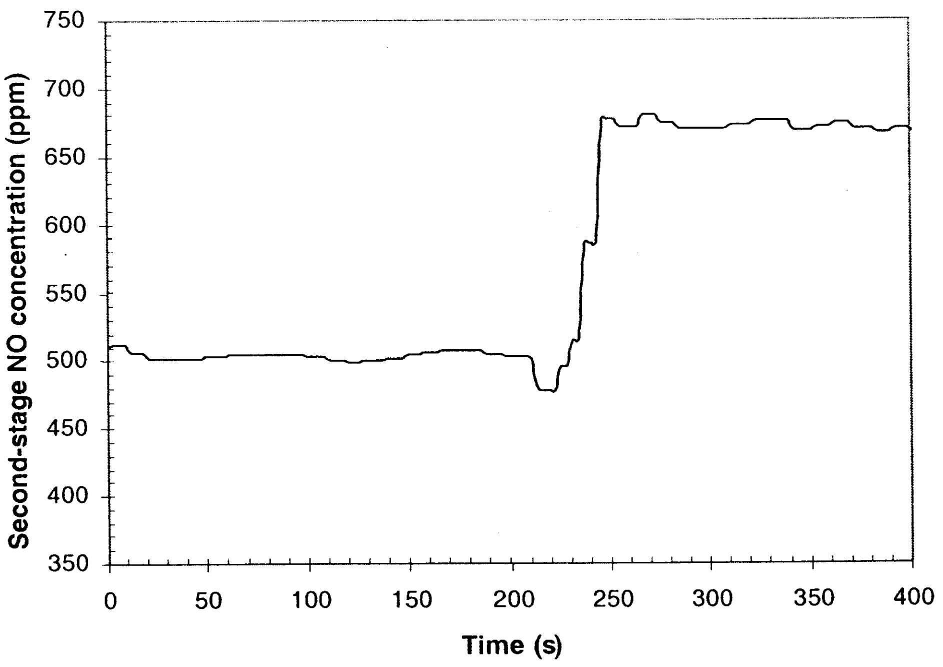


Figure 6.11 Open-loop second-stage NO response (experiment 2).

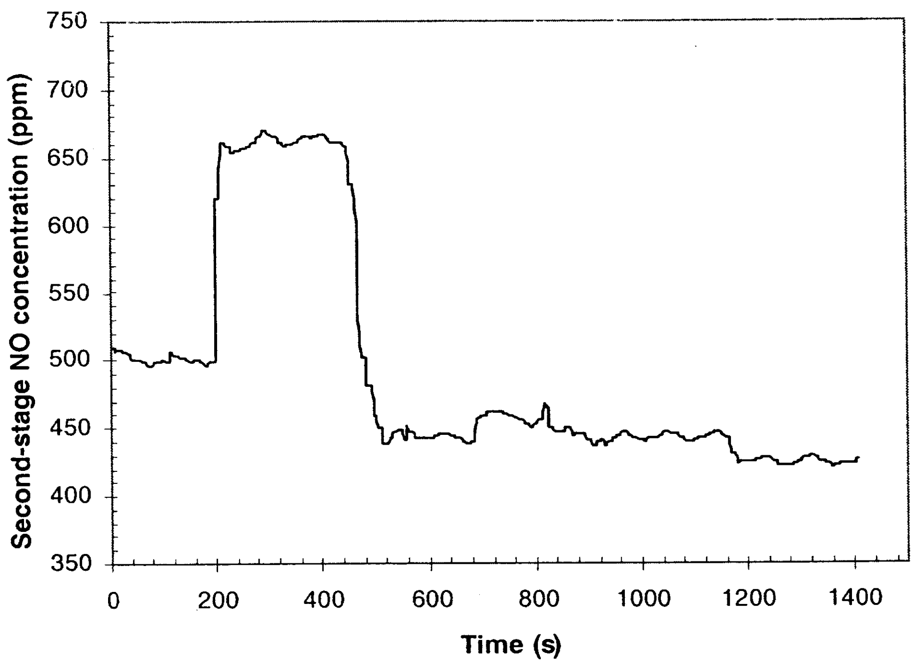


Figure 6.12 Closed-loop second-stage NO response (experiment 2).

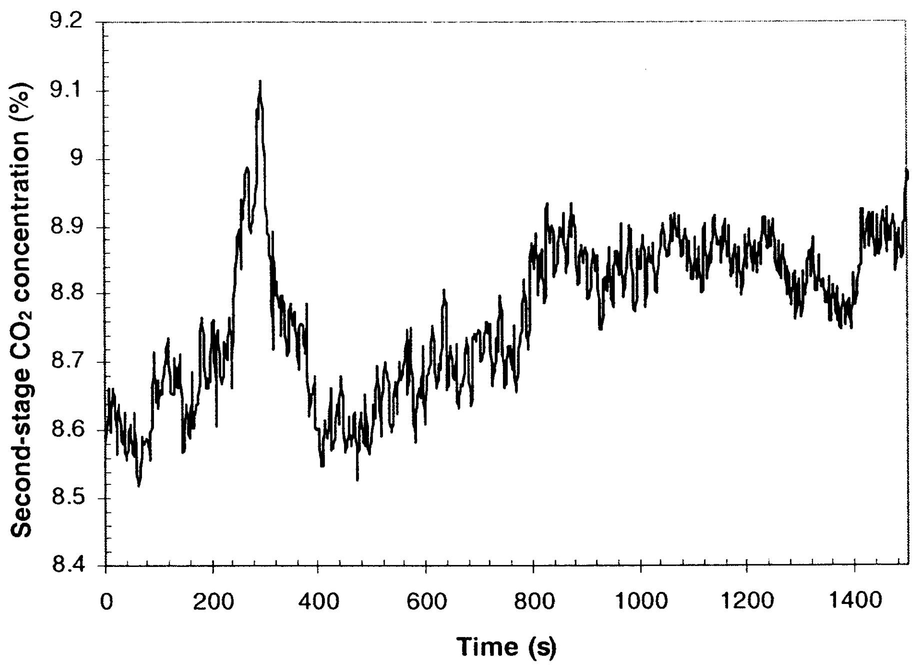


Figure 6.13 Closed-loop second-stage CO_2 response (experiment 2).

Dead time in the sampling process introduced a phase lag between the system output and the input signal. To reduce the impact of the large time-delay, the Smith predictor technique is applied here (Figure 6.6 scheme). Figure 6.14 shows the closed-loop response for the NMPC with the Smith time-delay compensator (NSTC) for the same step change in the nitrogen-bound dopant in the feed. The controller settings and process conditions are the same as those used in the previous experiment except for the frequency of the control action. With the NSTC, the controller was set to execute every 5 seconds. A comparison of Figure 6.14 and 6.9 shows the improvement in performance (process settling time of 400 seconds vs. 1500 seconds) obtained using the Smith predictor algorithm. The NSTC enables the control system to respond more dynamically to process changes. The second-stage CO_2 concentration profile is shown in Figure 6.15 to verify that the overall equivalence ratio (ϕ_o) was brought close to the setpoint.

Although the experimental results with the lag compensation show the average model errors (NO - NO″) are sometimes as much as ±15%, the Smith time-delay compensator, based on the cluster of neural networks, is robust enough to provide improvement over the uncompensated control configuration. This result is consistent with Seborg et al. (1989), that such compensation is effective as long as model errors are no greater than approximately ±30%. It should be noted that the model errors observed in these experiments were primarily caused by a variation in the time-delay in the sampling process and not due to the neural network process emulator. It was observed during the experiment that the NO sampling and analysis times fluctuated in the range of 200 to 240 seconds.

Lastly, regardless of the robustness of the controller, a small process offset after the system settled at the new steady state cannot be completely eliminated. The offset observed in this study can be primarily attributed to the imprecision of the electronic control valves. It was observed during the experiment that the electronic control valves are very sensitive to the pressure and become inaccurate over time especially when the control signal changes direction. The valve movement errors to the control signals were sometimes as high as ±20%. This phenomenon severely affected the controller performance.

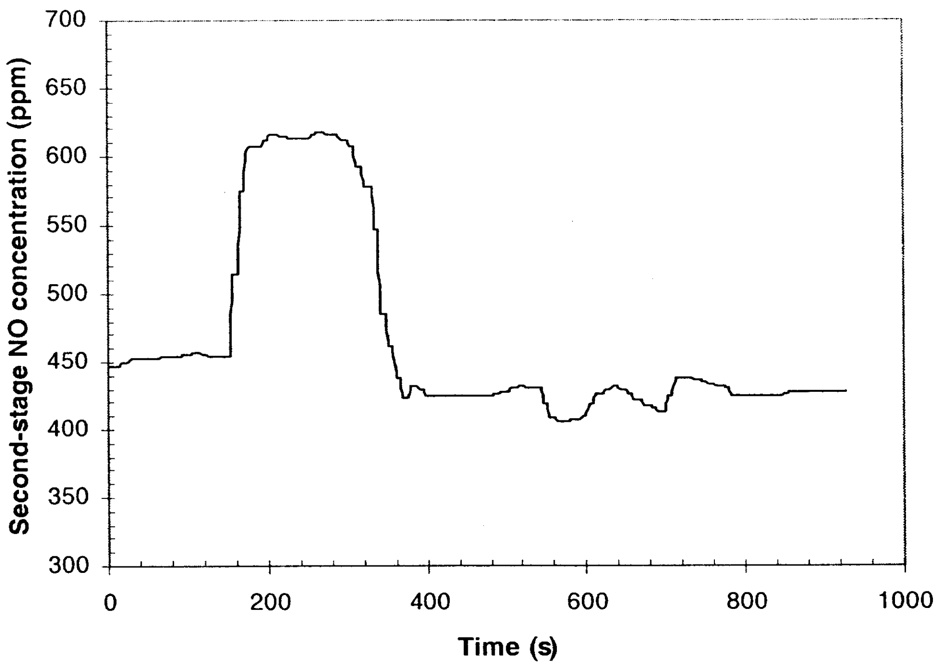


Figure 6.14 Closed-loop second-stage NO response (NMPC/NSTC).

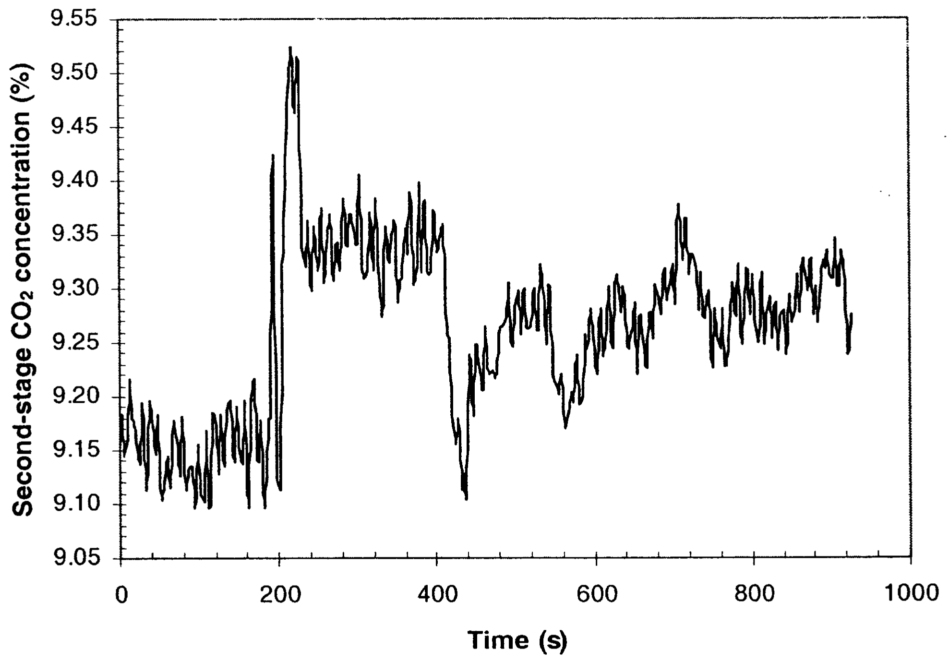


Figure 6.15 Closed-loop second-stage CO_2 response (NMPC/NSTC).

6.5 Conclusions

In this chapter, the usefulness and effectiveness of applying neural networks in a model-based control strategy to control a combustion reactor have been demonstrated. The experiment provided a detailed case study in which neural networks were applied to the nonlinear NO control problem with long process lags due to species sampling/analysis. With the neural network-based model-predictive controller (NMPC), the process was successfully brought back to the setpoint after a step disturbance in the feed stream. To improve the settling time in the presence of significant time delay, the feedforward control technique was applied to the existed feedback control loop through the use of another cluster of neural networks. This cluster of two FMLP neural networks served as the Smith time-delay compensator (NSTC). The combination of NMPC and NSTC represents a feedforward-feedback control configuration. Results show that the Smith time-delay compensator, implemented with the neural networks, is robust and accurate. It is valuable in overcoming the inherent response time lag of sampling and analysis system.

In this study, the active control movement (i.e. transient network response) was enabled by the time stepping function in the Visual Basic program interface. This arbitrary time stepping response, through the programming interface, was possible based on the fact that the combustion process was inherently stable. A stable behavior of the combustion system allowed the time-series characteristics associated with the process (e.g. analysis and sampling lags) to be secluded from the neural network training data. The time-series information was incorporated into the control system by employing a conventional mathematical function and technique (e.g. pre-assigned memory buffers through the Visual Basic program interface).

Without time-series information in the network training data, the neural networks were easily trained with existing steady-state experimental data. This procedure allowed rapid convergence during the training process and avoided any complex weight update algorithms often used in online learning processes. Finally, it should be noted that the time stepping approach, however, is not applicable to processes that are inherently unstable. One of the most common of these unstable systems is a "broomstick balancer" problem, wherein a comprehensive time-series information about the process dynamics has to be incorporated directly into the neural network controller (Hecht-Nielsen, 1989).

CHAPTER 7

PROCESS SIMULATOR

7.1 Introduction

To establish that the ANN-based virtual process control is a viable alternative to existing control strategies, a test using a PID controller was conducted. While PID might not represent state of the art in conventional controllers, it is still a widely accepted standard (Martins and Coelho, 2000; Olsson et al., 2001; Seborg et al., 1989; Stephanopoulos, 1984).

Initial attempts to use a virtual PID in the existing laboratory combustor system were not successful. While trying to tune the PID, a series of continuous cycling process responses occurred as the system was being pushed to its stability limit. The continuous cycling process response can also occur after a large disturbance. With the lab-scale combustor, a potentially hazardous situation resulted from the continuous cycling process.

In order to avoid these safety concerns, a transient process simulator, i.e., a virtual combustor, was created for use in this and future control studies. The simulator is very useful for gathering more information about process behavior, especially if experimenting with the real process is time-consuming, expensive, or a safety concern. Through multiple simulation runs, it is possible to gain insight into how to optimize the real process operation.

Process modeling is a key part in constructing the process simulator. Modeling a chemical process is often a complex task because the exact relationship between input and output variables of most chemical processes are complicated by nonlinear and time-varying behavior.

A theoretical model such as PSR + PFR that accesses the CHEMKIN for the two-staged combustion reactor has been published (Glarborg et al., 1986; Kee et al., 1989). Although, the CHEMKIN simulation is satisfactory in modeling steady-state combustion behaviors from most points of view, its structure and computation algorithm inhibit it to be effectively used as a transient process simulator.

A process model based on artificial neural networks (ANNs) is an alternative simulation approach that has received widespread attention (Booth, 1998; Evenson and Kempe, 1998; Johnson, 1998; Miller and Lemieux, 1998; Radl and Roland, 1995; Reinschmidt and Ling, 1994). The advantages of using the neural networks are twofold. First, the neural network is capable of approximating any continuous nonlinear function to an arbitrary degree of accuracy and should be able to model complex process dynamics successfully. Secondly, a neural network-based model is data-driven and computationally efficient, if its structure is not too complex. As a result, ANN can be easily constructed and integrated into most programming platforms.

In this chapter, a process simulator based on a feedforward multi-layer-perceptron (FMLP) neural network is designed and developed to study the application of process control of the two-stage combustion system. The steady-state combustion behavior is modeled by using a FMLP neural network. The neural network-based combustion process model is statically trained and tested using existing steady-state experimental and simulation data. The simulator is developed on the Windows 98 platform PC using Visual Basic 6.0 as the simulator interface. The transient response of the laboratory combustion system is emulated and enabled by the time-stepping function in the Visual Basic program. The transient process response (open-loop and closed-loop) from the simulator was thoroughly tested and validated against the existing experimental results. Upon validating its performance, the simulator for the combustion system could serve as a supplementary tool, and provide a more flexible and reliable platform in process control studies. The process simulator is especially advantageous in the ranges of operations that are physically limited in the actual experiment. A series of simulation runs under various conditions were made to examine the accuracy of the process simulator. Included in the simulation runs are open-loop with no controller, closed-loop with the PID controller, and closed-loop with the neural network-based controller (NMPC/NSTC).

7.2 Neural Network-Based Steady-State Process Model

7.2.1 Neural Network Structure

A feedforward neural network, shown in Figure 7.1, was developed to model the relationship of various effluent gases from both first and second stages of the combustor to the process inputs. A single-hidden-layer neural network model has two inputs, which are first-stage equivalence ratio (ϕ_1) and NH_3 dopant ratio (NH_3/C_2H_4). There are four nodes in the hidden layer. The network has four outputs, which are the first-stage (PSR) CO and CO_2, and the second-stage (PFR) CO_2 and NO concentrations. The first-stage CO and CO_2 and second-stage CO_2 concentrations as the network outputs demonstrate the effectiveness of staged combustion in the CO burnout process. It is essential to verify that while increasing ϕ_1 in trying to minimize the NO concentration, the effluent CO can be suppressed through the CO to CO_2 burnout process. The second-staged CO concentration is not included in the network output since the overall equivalence ratio (ϕ_o) is maintained at 0.9 at all time in this study. At this fuel-lean overall equivalence ratio ($\phi_o < 1$), it was previously found by Mao and Barat (1996) that the second-stage CO concentration is quite low (less than 0.3%).

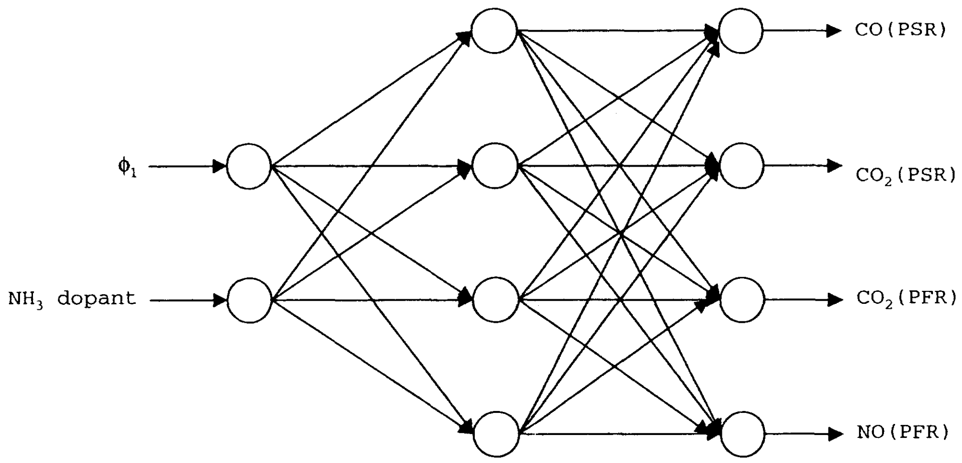


Figure 7.1 Neural network combustion process model.

A total of 30 steady state data points for use in the training process was drawn from the existing experimental and the CHEMKIN simulation database. All chemical reactions and kinetic parameters used by the CHEMKIN simulation were taken from the GRI mechanism (Gas Research Institute, 1994). The training data are shown in Table 7.1. The second and third columns represent the network inputs, which are the first-stage equivalence ratio ϕ_1 and the NH_3 dopant ratio. Columns four to seven represent the network outputs. The fourth and fifth column are first-stage CO and CO_2 concentrations, drawn from the CHEMKIN simulation data. The sixth and seventh columns are second-stage CO_2 and NO concentrations, drawn from the existing experimental data.

Table 7.1 Neural Network Training and Testing Data (Process Model)

Sample No.	ϕ_1	NH_3 dopant ratio	PSR CO (%)	PSR CO_2 (%)	PFR NO (ppm)	PFR CO_2 (%)
1	1.35	0.078	5.58	4.76	470	11.5
2	1.45	0.078	6.45	4.31	914	11.7
3	1.20	0.078	3.33	5.38	580	10.2
4	1.45	0.057	6.38	4.47	680	11.9
5	1.30	0.027	4.73	5.21	300	11.2
6	1.25	0.057	3.81	5.07	490	10.9
7	1.10	0.078	2.20	6.22	770	9.4
8	1.40	0.022	5.91	4.6	300	11.4
9	1.25	0.022	3.95	5.5	320	10.7
10	1.35	0.027	5.47	4.95	270	11.6
11	1.15	0.057	2.73	5.85	620	9.6
12	1.35	0.022	5.45	4.96	230	11.3
13	1.20	0.022	3.16	5.59	318	9.9
14	1.30	0.022	4.71	5.23	260	11.1
15	1.15	0.027	2.65	5.94	440	9.7
16	1.35	0.057	5.53	4.84	400	11.7
17	1.10	0.057	2.15	6.28	630	9.16
18	1.40	0.078	5.91	4.51	670	11.5
19	1.15	0.022	2.05	6.21	370	9.92
20	1.20	0.027	3.17	5.58	370	9.8
21	1.30	0.078	4.87	5.00	440	11.4
22	1.25	0.027	3.97	5.49	345	10.5
23	1.45	0.027	6.56	4.44	500	11.7
24	1.40	0.057	5.96	4.48	500	11.9
25	1.40	0.027	6.02	4.64	350	11.6
26	1.20	0.057	3.27	5.46	530	10.02
27	1.45	0.022	6.37	4.58	400	11.6
28	1.30	0.057	4.81	5.09	420	11.4
29	1.15	0.078	2.69	5.88	660	9.6
30	1.25	0.078	3.81	5.46	520	11.2

In trying to optimize the network performance and prevent overtraining effects, the training technique suggested by Hecht-Nielsen (1989) is implemented in this study to determine the optimal number of training cycles. The technique is especially advantageous when there are enough data to adequately train the neural network, but not enough to hold out for validation and acceptance testing sets. Here, the network was trained five different times. Each time, the training set consists of 30 examples with 6 different examples were randomly held out for using as a testing set (i.e. 24 examples for training and 6 examples for testing). Each time, the network error and the overtraining effects were monitored through the learning curves. After doing this five times, the optimal training cycle was estimated from the average of the five different training cycles where the training and testing error curves level out. Once the training cycle is identified, the network is retrained for one last time using all 30 examples as the training set with no examples held back.

The five sets of the learning curves are shown in Figures 7.2 to 7.6. The optimal learning cycle for each of them are identified by the point where both the training and testing curves level out. The optimal learning cycles are 60, 400, 500, 500, and 1000 with the training errors (MSE) 0.0060, 0.0040, 0.0039, 0.0048, and 0.0040 respectively. The average training cycle is approximately 500 and is used for retraining the network with the complete training data of 30 examples. As shown in Figure 7.7, the final training error is 0.0039. This low value indicates that the neural network model is well-trained and should be sufficiently accurate for our purposes.

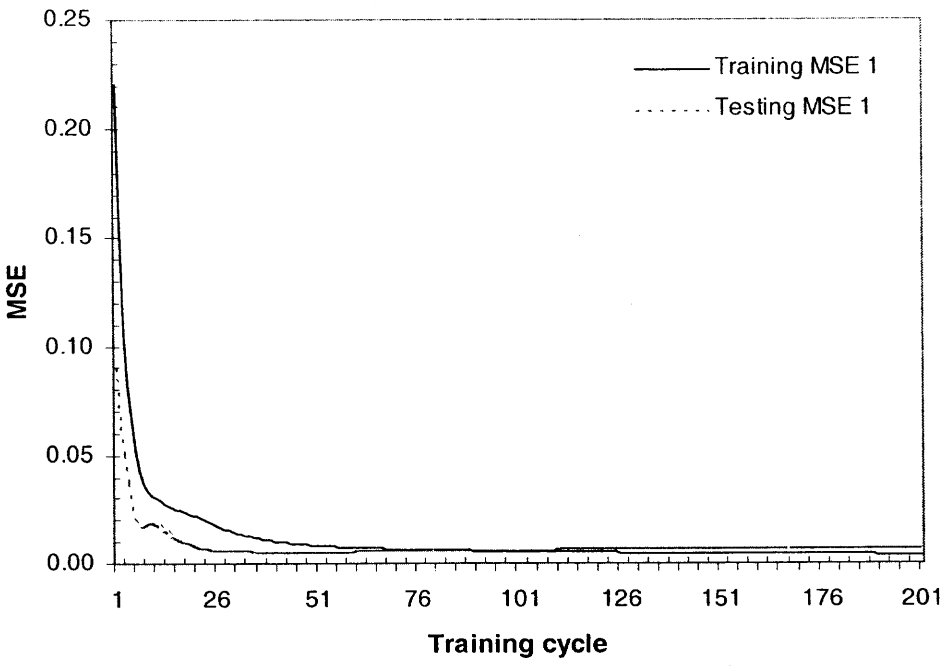


Figure 7.2 Neural network learning curves (training set 1).

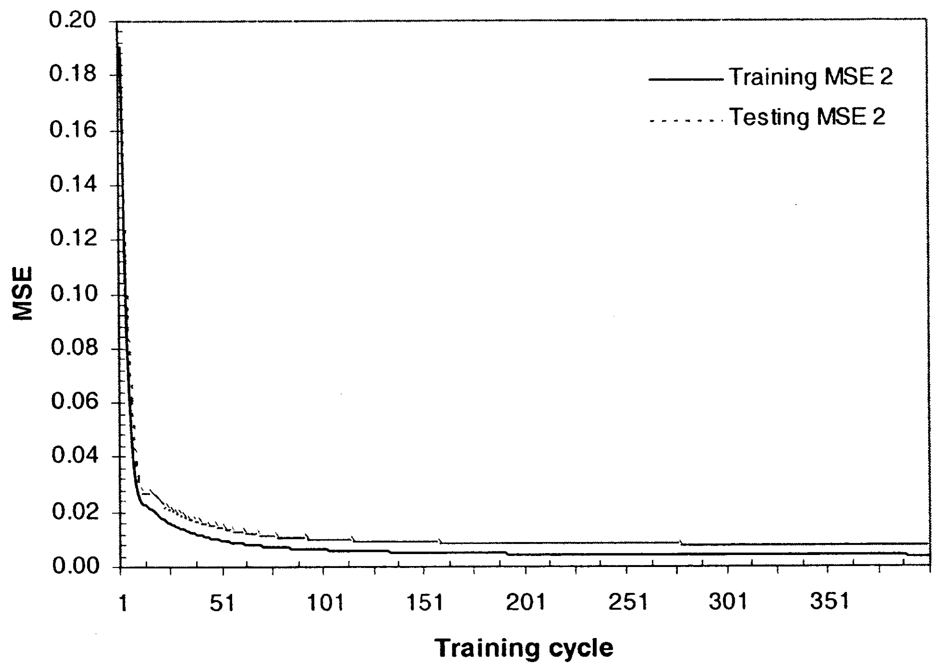


Figure 7.3 Neural network learning curves (training set 2).

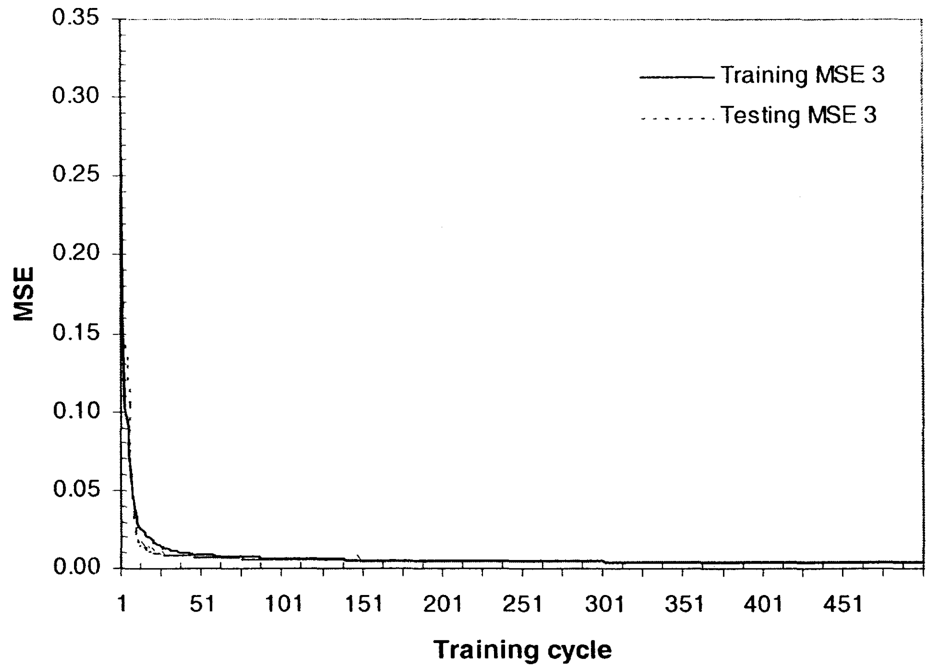


Figure 7.4 Neural network learning curves (training set 3).

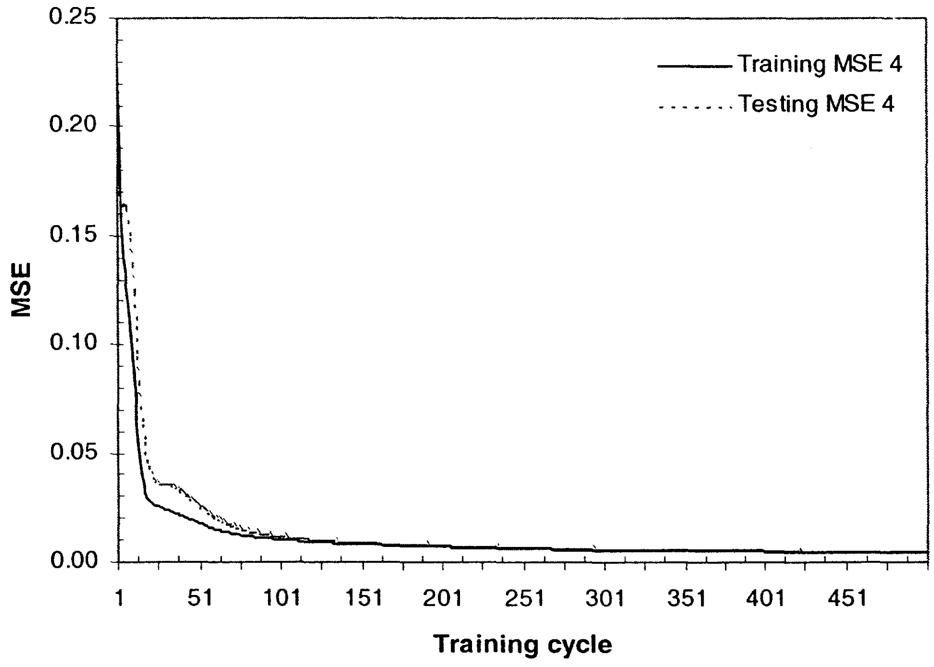


Figure 7.5 Neural network learning curves (training set 4).

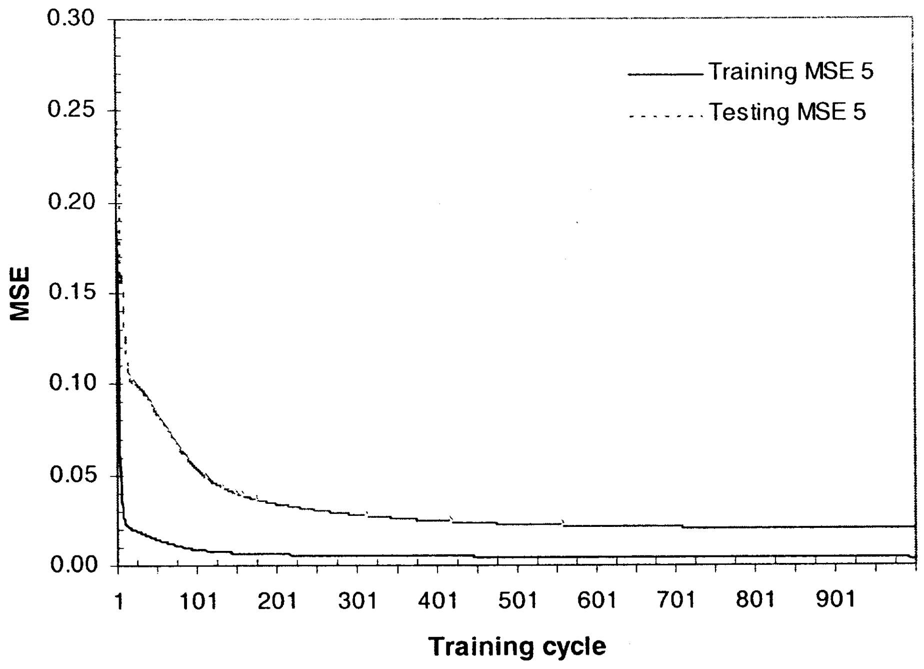


Figure 7.6 Neural network learning curves (training set 5).

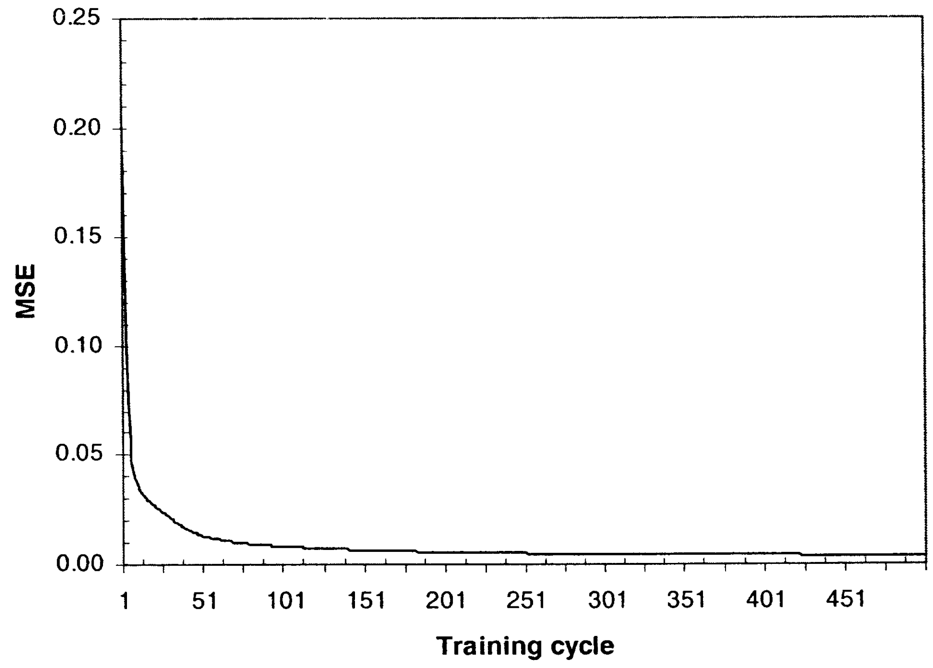


Figure 7.7 Neural network learning curve (complete training set).

7.2.2 Steady-State Model Validations

In trying to maximize the network performance, all existing experimental and CHEMKIN simulation data were utilized in the network training and testing process. To validate the neural network model accuracy, the 30 examples of known output-input pairs, used in the training process, are reused as a testing set in this section. Figures 7.8 to 7.10 show comparisons between the first-stage CO and CO_2 and second-stage CO_2 concentrations generated from the trained neural network model and the testing data under various conditions. Higher first-stage equivalence ratio (ϕ_1) results in an increase in the first-stage CO and the second-stage CO_2 concentrations, and a decrease in the first-stage CO_2 concentration. It is noted that the NH_3 dopant has no significant impact on the CO and CO_2 concentrations. The deviation between the network output and the testing set is minimal. Figures 7.11 also shows a minimal deviation of the network predicted second-stage NO concentrations from the actual experimental results. Modeling results are in good agreement with the experimental data. In addition, based on the observation in this study, but not shown in the validation results, the neural network model showed a good generalization performance for the data it never seen before (i.e. points between the training-data points). It is, therefore, concluded that the neural network provides an accurate steady-state model, in particular for CO, CO_2, and NO concentrations.

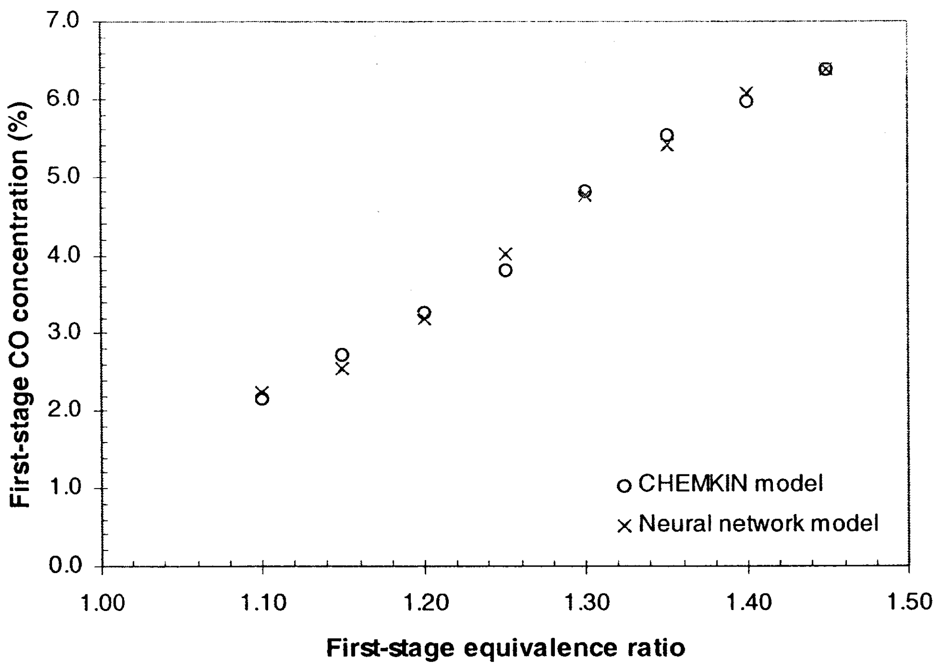


Figure 7.8 First-stage CO concentrations ($NH_3/C_2H_4 = 0.057$).

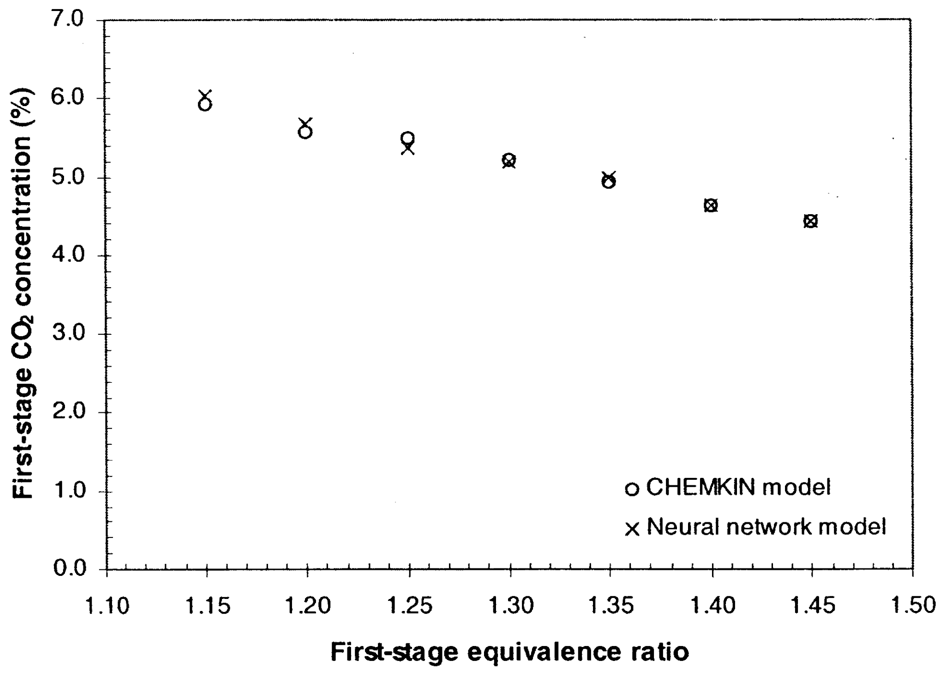


Figure 7.9 First-stage CO_2 concentrations ($NH_3/C_2H_4 = 0.027$).

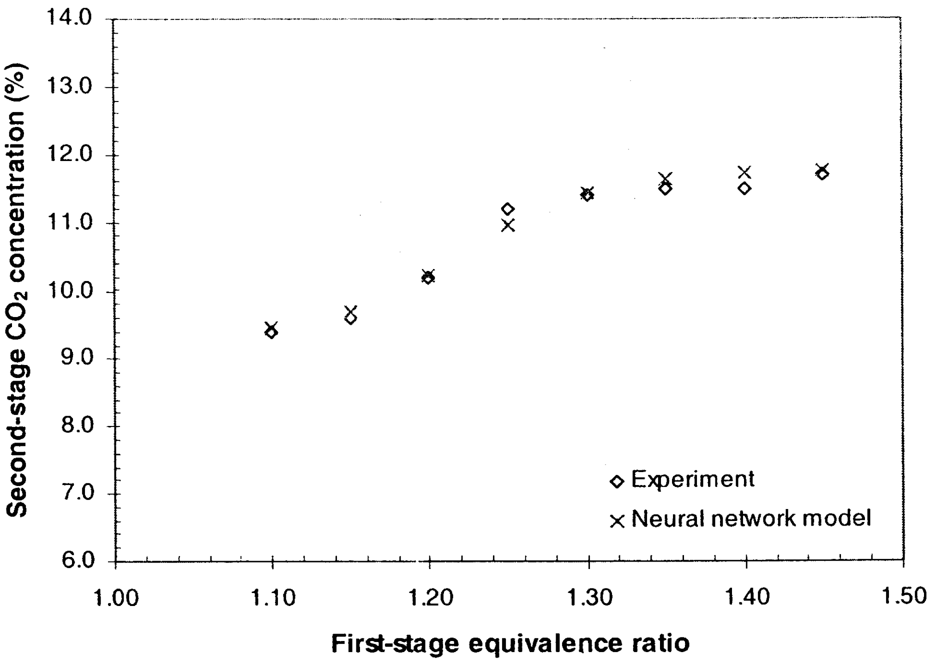


Figure 7.10 Second-stage CO_2 concentrations ($NH_3/C_2H_4 = 0.078$).

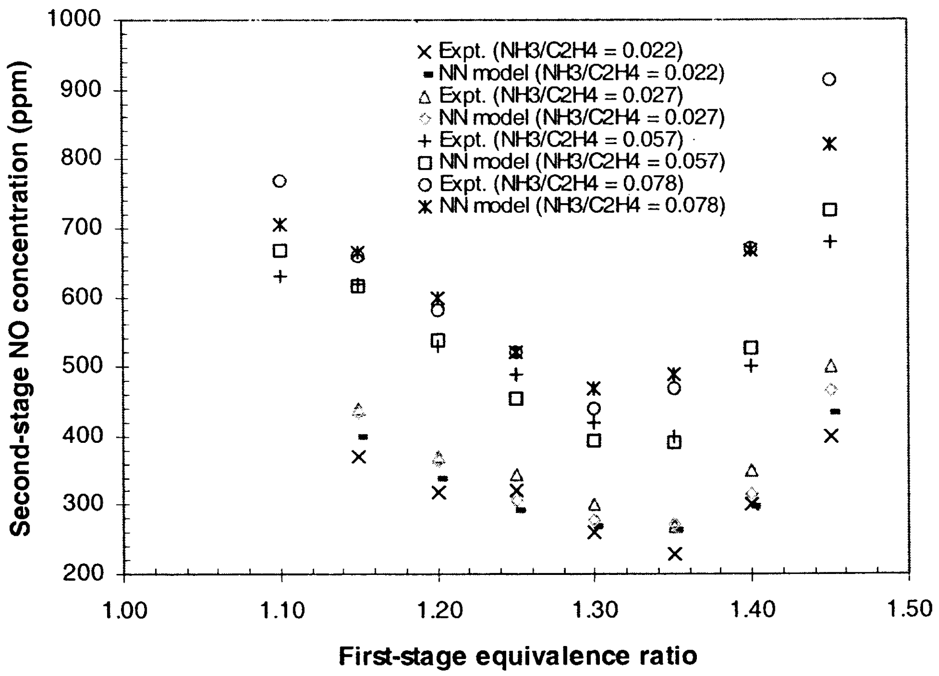


Figure 7.11 Second-stage NO concentrations.

7.3 The Process Simulator

The neural network process model is embedded into the Visual Basic process simulator program in the form of a dynamic link library file (DLL). The DLL file is an executable file of the trained neural network process model. It performs a network mapping function between the process inputs and outputs.

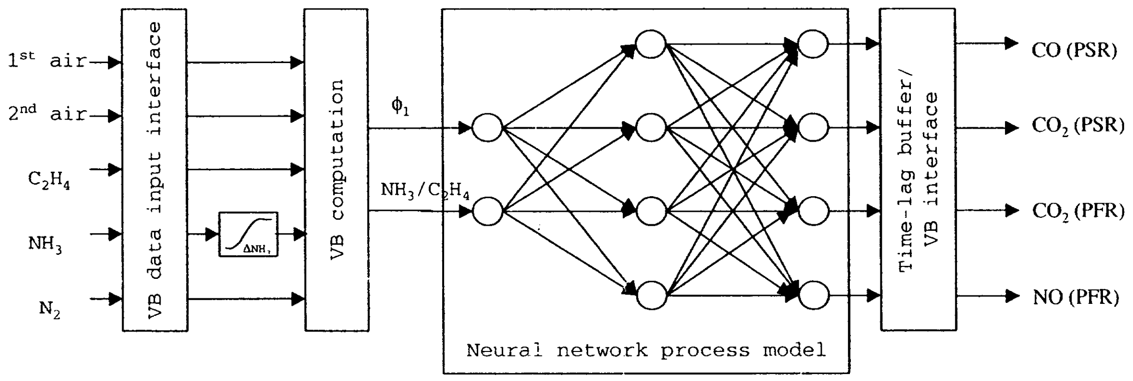


Figure 7.12 Process simulation flow chart.

The simplified flow chart of the simulation program is shown in Figure 7.12. The simulation interface was programmed in Visual Basic 6.0 environment on a Windows 98 platform PC. The simulation interface was integrated into the main operating interface. The program source code is listed in the Appendix. The simulation interface is shown in Figure 3.6. Here, instead of sampling process outputs from the Fluke data logger, the process outputs are generated internally by the neural network process model embedded in the simulation interface. The timer, a built-in tool in Visual Basic, is an essential tool for enabling the simulation program to run almost continuously. The simulator was designed to include all adjustable inputs to the combustor as they appeared in the actual laboratory facility. These adjustable inputs are the feed flow rates of the primary air (A_1), secondary air (A_2), primary fuel (C_2H_4), secondary fuel (NH_3), and dilution N_2. As shown in Figure 3.6, the number in each textbox represents the actual laboratory rotameter reading for each feed flow rate. The user can simply adjust these settings through the spin button next to each textbox on the simulation interface. The simulator was set to execute at every preset timer interval (5 seconds). For each timer interval, the process simulator inputs the combustion process inputs from the Visual Basic interface and computes essential information to be directed into the neural network model.

Additional programming tools are used to address the time-dependent behavior associated with the process and equipment dynamics, instrumentation response times, and time lags between the process inputs and gas monitors. In particular, a Visual Basic built-in hyperbolic tangent function is used to provide gradual transition effects of selected signals and process disturbances, which in turn make simulated process responses look mechanically similar to real process behaviors. Additionally, the timer component in Visual Basic is utilized to emulate the dead time in the gas sampling system. For simplicity in this study, lag-times in each gas analyzer are the same, here set to the value of 200 seconds. The impacts of adjusting the manipulated variables (primary and secondary air flow rates) on the process outputs are instantaneous.

The neural network model predicts the process outputs, which are the first-stage CO and CO_2 and second-stage CO_2 and NO concentrations. All of these are displayed in the textboxes on the process simulator interface.

7.4 Validation Runs and Results

7.4.1 Open-Loop Validation

For the purpose of validation, the same experimental program described in the previous chapter is exercised here. The initial ϕ_I was 1.14, while ϕ_O was set at 0.9. The initial NH_3/C_2H_4 ratio in the feed was 0.027. A step disturbance was then applied to the NH_3 flowrate to raise the dopant ratio to 0.057. A comparison of open-loop second-stage NO profiles between actual experimental and simulation results is shown in Figure 7.13. The simulated open-loop second-stage NO profile is in good agreement with the result from the actual experiment.

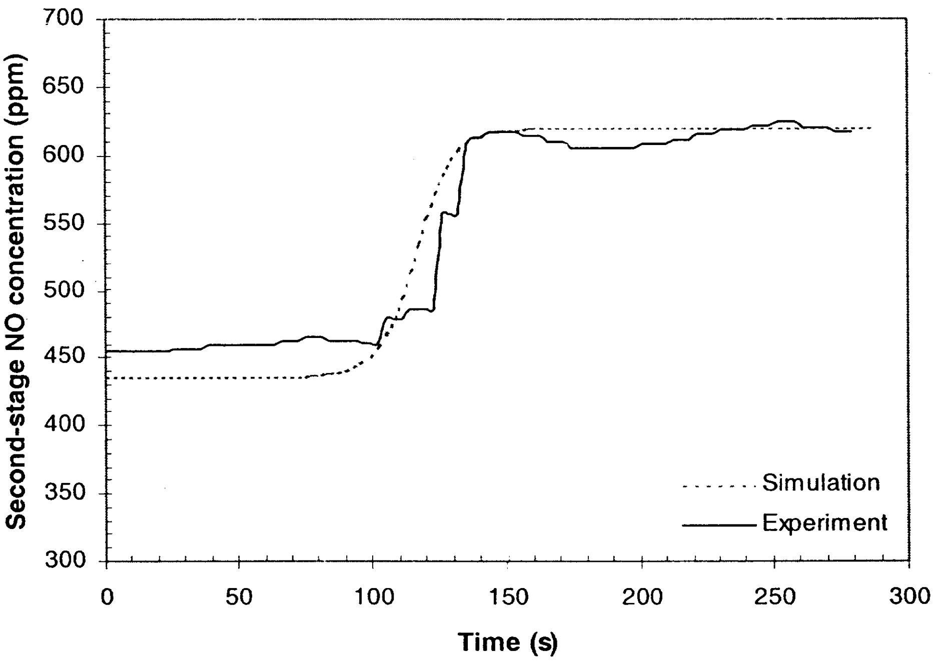


Figure 7.13 Open-loop second-stage NO responses (validation).

7.4.2 Closed-Loop Validation

Finally, a process control simulation is exercised. The neural network-based controller, used in the previous chapter, is put into the test. Again, the simulation was carried out using the same condition and disturbance as previously demonstrated in Chapter 6 and used in the open-loop validation above.

Figure 7.14 shows comparison between the simulation and experimental results of the second-stage NO profiles under neural network-based model-predictive controller (NMPC) equipped with the neural network-based Smith time-delay compensator (NSTC). The simulation result of NO control process is in good agreement with the experimental result obtained earlier in Chapter 6. Figure 7.15 shows a simulated second-stage CO_2 profile under the same control condition and disturbance setting. Comparing with the experimental second-stage CO_2 profile obtained earlier (Figure 6.10), the simulated second-stage CO_2 profile looks somewhat different from the actual experimental results. However, the two profiles share one thing in common – an upward trend of the CO_2 concentrations as time advances (i.e. as ϕ_1 increases in trying to minimize NO concentration). The difference between the CO_2 simulation and the CO_2 experimental results can be attributed to several sources, most prominent of which are the sensitivity of the CO_2 gas analyzer, the control signal lags, and the movements of the two electronic valves (one controlling primary air flow rate and the other controlling secondary air flow rate). The impact of the imperfect valve movements is imminent especially for a process with a short residence time like the lab-scale combustion system in this study as it is shown by the fluctuation in the CO_2 response in Figure 6.10.

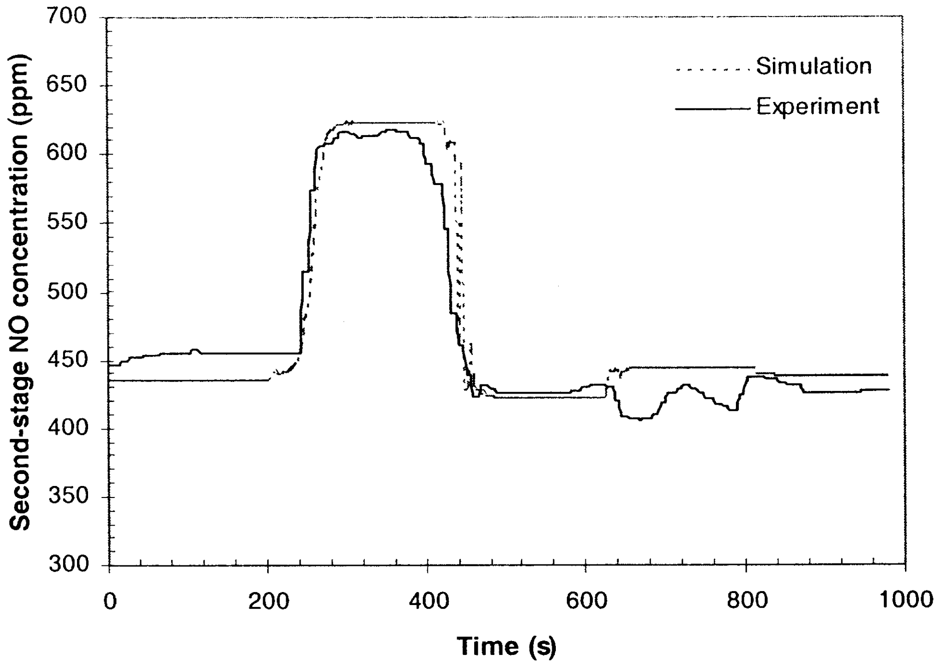


Figure 7.14 Closed-loop second-stage NO responses (validation).

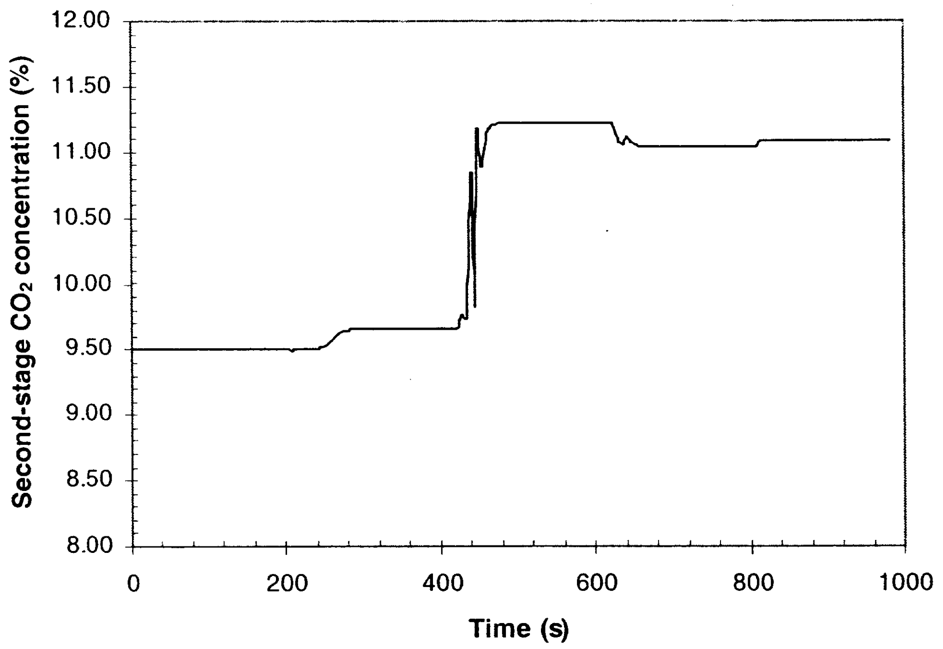


Figure 7.15 Closed-loop second-stage CO_2 response (validation).

To test the performance of the neural network-based controller, we compared it with a classic PID controller. The PID control loop was put into service by attaching it to the process simulator. The PID had to be conservatively tuned by a trial and error method and set to execute every 90 seconds in order to avoid a built-up integral term that could cause a run away process. It was observed during the PID tuning process that the PID controller becomes unreliable when the process approaches the bottom of an approximately parabolic well surface (Figure 7.11). Figure 7.16 shows a simulated second-stage NO profile under the tuned PID controller after a step disturbance of the NH_3 in the feed. All settings are the same as previously exercised with the neural network-based controller.

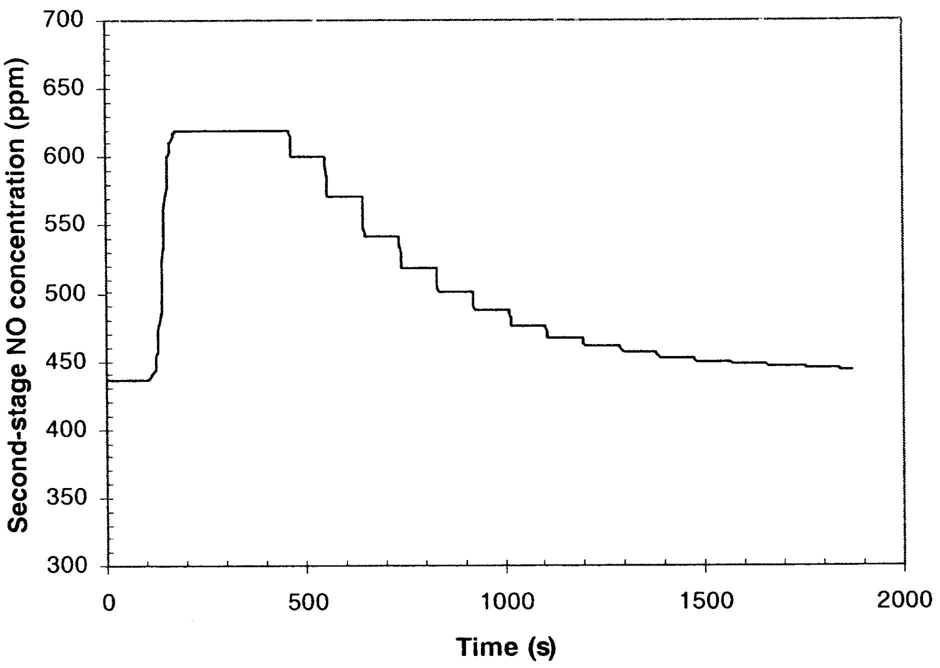


Figure 7.16 Simulated second-stage NO response (PID, setpoint 440 ppm).

Comparing Figure 7.16 to Figure 7.14, the performance of the neural network-based controller is better than that of the PID. The presence of process lags does not alter the neural network-based controller performance in a significant way. Although the PID controller is capable of eliminating the process offset, the process response under the PID controller tends to be sluggish with a very long process settling time (1800 seconds). The long process settling time is partly the result of the conservative combination of PID settings used to avoid the undesirable process response caused by the build-up integral term. It should also be noted that the Smith time-delay compensation structure cannot be implemented in a traditional PID control configuration because the PID does not have the capability of identifying the unknown feed stream composition (i.e. the feed stream composition is an essential input to the neural network-base Smith time-delay compensator). Clearly, the process settling time under the PID is significantly longer than the neural network-based controller.

7.5 Simulation Runs and Results

Using the process simulator, more general studies and insight into the NO control process can be gained. One of the most important issues that should be addressed is the usefulness and capability of the staged-combustor to suppress the effluent CO. In particular, while NO concentration is minimized by reduced oxygen availability at higher ϕ_1 values, CO burnout is reduced with a resulting increase in the CO concentration in the first stage. However, a very low effluent CO can be achieved by increasing the second-stage air flow rate. This is especially well illustrated by Mao and Barat (1996).

In addition to the control base-case as demonstrated so far, a new control simulation program is presented here for a fuller investigation and to ensure that the neural network-based controller is robust within the range of training data.

Here, the initial ϕ_l was 1.15, while ϕ_o was set at 0.9. The initial NH_3/C_2H_4 ratio in the feed was 0.035. The measured NO concentration from the second stage was 500 ppm. This value was then used as the process setpoint. Noted that this NO value is not at the minimum of the second-stage NO vs. ϕ_l curve (Figure 6.1). A step disturbance was then applied to the NH_3 flow rate to raise the dopant ratio to 0.078. Figure 7.17 shows the simulated open-loop second-stage NO concentration. The NO concentration rose to 660 ppm, which is in good agreement with the actual experimental result obtained earlier (Figures 6.1 and 6.11).

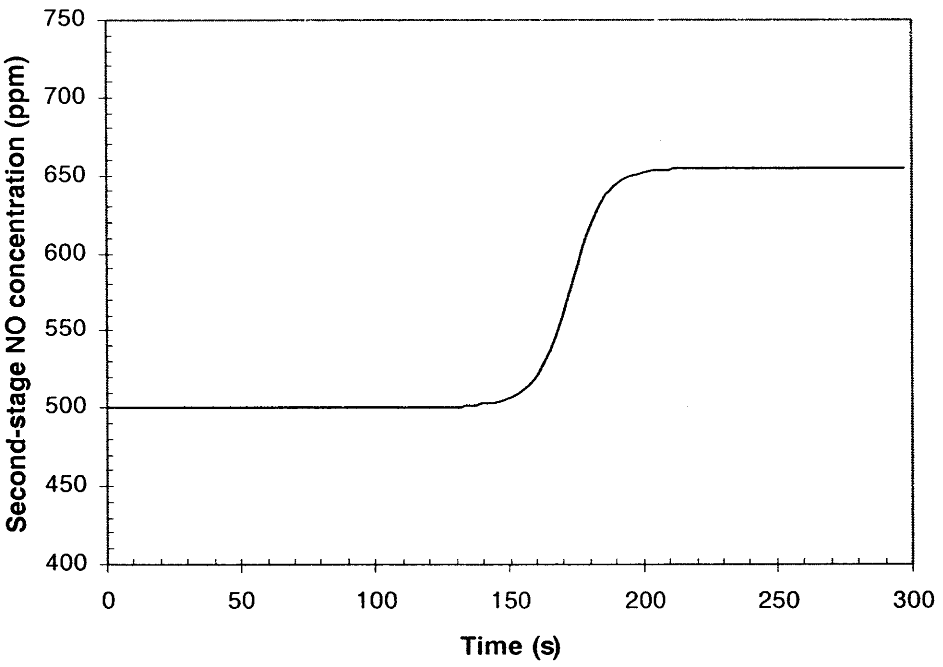


Figure 7.17 Simulated open-loop second-stage NO response.

In closed-loop, the controller increased ϕ_1 to about 1.25 by reducing the primary air flow rate in order to bring the NO back to the setpoint (500 ppm). The secondary air flow rate was simultaneously increased in order to maintain ϕ_o. Figures 7.18 to 7.19 show simulated closed-loop profiles of various species under the traditional PID controller. The PID successfully brought the NO concentration to the desired setpoint within 1800 seconds. The same observation about the second-stage CO_2 concentration can be made as in the previous section. From Figure 7.19, the first-stage CO concentration dramatically increases (from 2.1 to 4.2%) as ϕ_1 increases in trying to maintain the effluent NO at the desired setpoint. However, regardless of how high the first-stage CO concentration is, the effluent CO from the second-stage outlet (not shown) is much lower, thus reflecting the additional conversion from CO to CO_2 attained in the second stage. This statement is confirmed by the increase in the second-stage CO_2 concentration illustrated in Figure 7.19.

Figures 7.20 and 7.21 show the simulated closed-loop profiles of various species under the neural network-based model-predictive controller (NMPC) equipped with the neural network-based Smith time-delay compensator (NSTC). As shown in Figure 7.20, the second-stage NO concentration is brought back to the setpoint within a much shorter time frame than the PID controller does. The first-stage CO and CO_2 and second-stage CO_2 profiles are illustrated in Figure 7.21 All profiles are very consistent with the detailed reaction pathway described earlier and elsewhere (Mao, 1995; Mao and Barat, 1996).

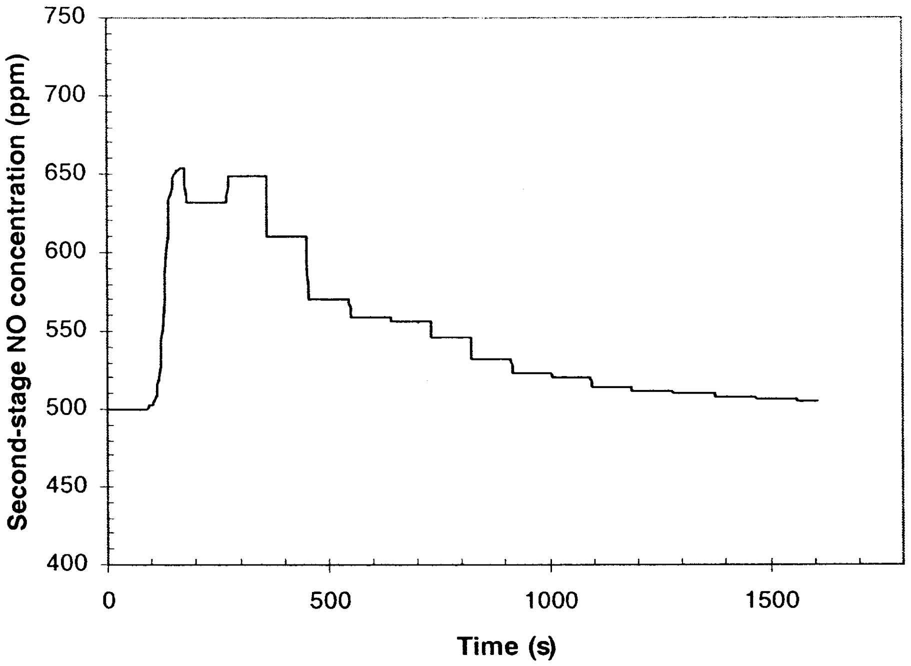


Figure 7.18 Simulated second-stage NO response (PID, setpoint 500 ppm).

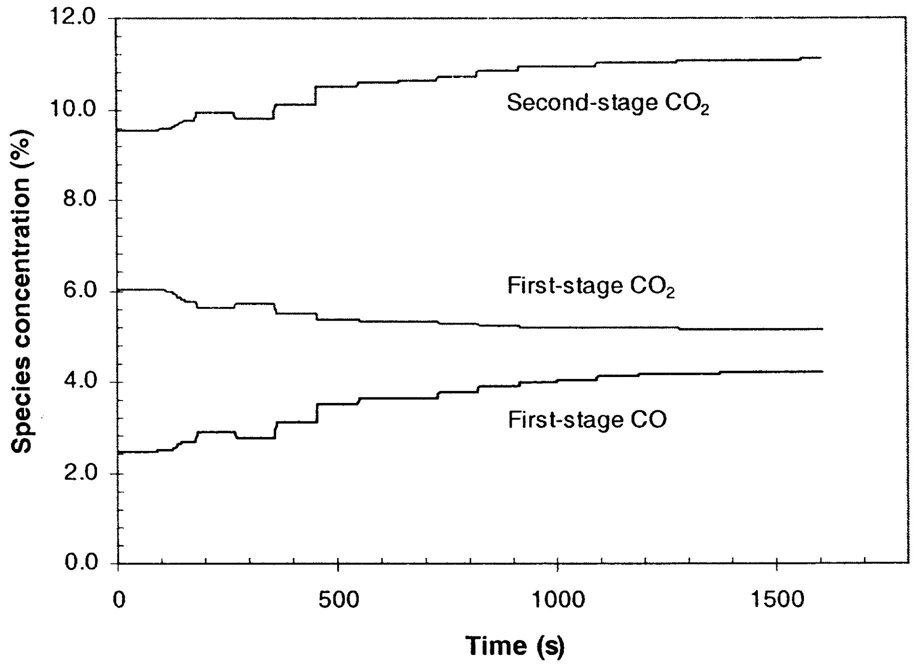


Figure 7.19 Simulated CO and CO_2 responses (PID).

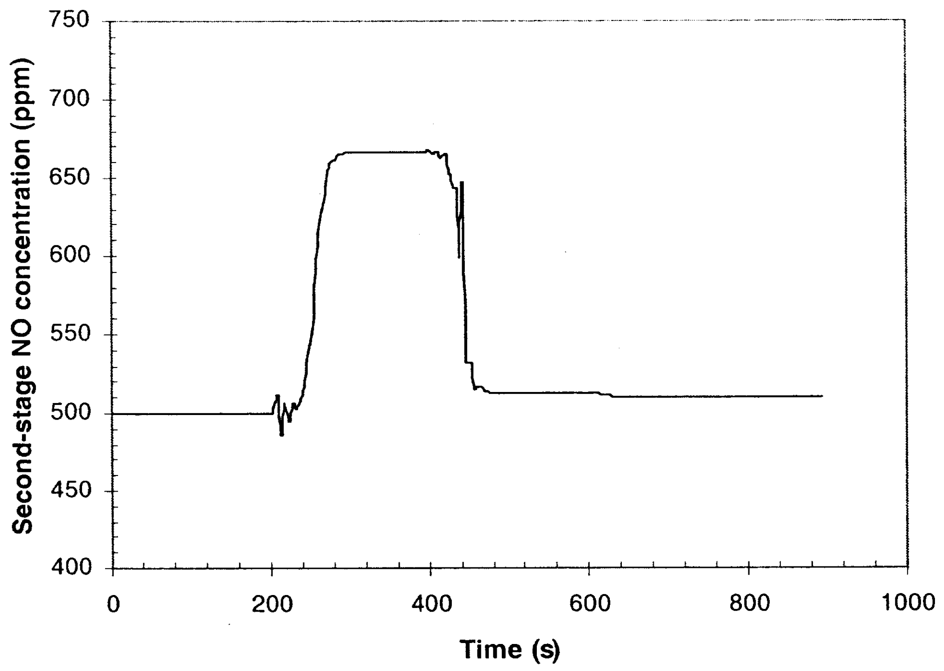


Figure 7.20 Simulated second-stage NO response (NMPC/NSTC).

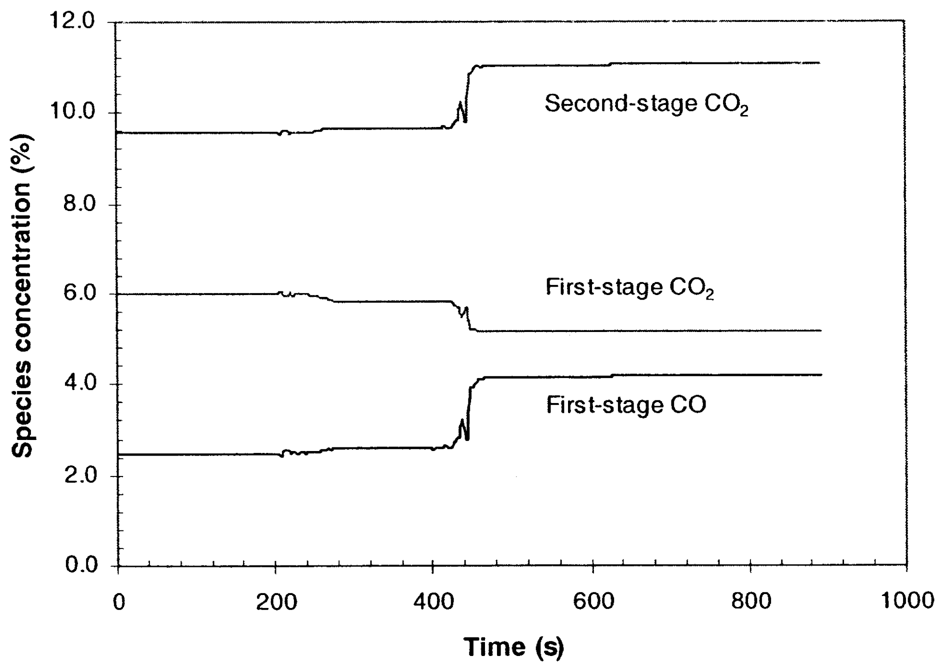


Figure 7.21 Simulated CO and CO_2 response (NMPC/NSTC).

7.6 Conclusions

A combustion process simulator was developed in this chapter. A statically trained FMLP neural network is proven to be an effective tool in modeling the steady-state two-stage combustion process. The training data were drawn from the existing experimental results and CHEMKIN simulation data. The training technique suggested by Hecht-Nielsen (1989) was applied to determine the optimal training cycle.

A process simulator was built on the Visual Basic programming platform. The well-trained neural network process model was embedded into the simulator interface as a DLL file. The simulator provided an accurate transient model for the actual combustion system and proved to be an effective approach for a process control study. With its help, it is possible to make a fuller investigation and establish that the proposed neural network-based virtual process control is a viable alternative to the conventional PID controller. A closed-loop simulation using the PID controller was carried out. The neural network-based controller shows a superior performance over the PID controller in terms of both process settling time and compatibility with the Smith time-delay algorithm. The process simulator also provided insight into the suppressing of the effluent CO through the staged combustion. In this study, the CO burnout rate was indicated by the measurements of the first-stage CO and CO_2 and the second-stage CO_2 concentrations.

Finally, it is important to point out that with a more advanced programming skill - specifically that one would be building on the work done here, as well as a fuller experimental database, a more sophisticated process simulator, which incorporates more detailed mechanical-combustion behaviors, more combustion-related species, and essential temperature information, can be obtained.

CHAPTER 8

CONCLUSIONS AND COMMENTS

8.1 Summary

This research effort has demonstrated the usefulness and effectiveness of applying artificial neural networks in a process model-based strategy to control a combustion process. The experiments and simulations provide detailed case studies in which neural networks were applied to both linear and nonlinear control problems.

In detail, an overview of the limitations associated with conventional control and modeling methods was provided in Chapter 1, followed by the objectives of the dissertation. Chapter 2 reviewed the relevant research literature. Included in this discussion are the topics of process dynamics and control, the model-based control concept, and the roles of artificial neural networks in various control applications.

Detailed descriptions of the combustion facilities, gas sampling and analysis system, and control system were presented in Chapter 3. Chapter 4 contained an overview of the neural network topologies and computation algorithms. A detailed discussion of the neural network supervised learning process was included.

A feedback process control study for a monotonic control problem was presented in Chapter 5. The statically trained neural network was used in conjunction with the proportional controller in the proportional neural network control (PNNC) structure for controlling second-stage O_2 level. Without the complication of tuning the PID controller and drastically changing the control configuration, the experimental results showed significant performance improvement over the proportional controller when the neural network was applied as an intelligent tool for updating the bias value. The proposed PNNC structure offers a great potential to apply toward other feedback control problems.

The usefulness and effectiveness of applying neural networks in a model-based control strategy to control a staged combustion process was demonstrated in Chapter 6. The experiment provided a detailed case study in which two clusters of neural networks were applied to the nonlinear NO/CO control problem with significant time delay due to the species sampling/analysis process. The proposed control system falls under the feedforward-feedback control category. The key novelty in this study lies in the structure and the functionalities of the two clusters of steady-state trained FMLP neural networks. The first cluster (NMPC) consists of two neural networks. The first network identifies the amount of ammonia in the feed. Based on that value and the NO setpoint, the second network adjusts the first-stage equivalence ratio ϕ_1. The second cluster (NSTC) also consists of two neural networks. It is the process emulator and serves as the Smith time-delay compensator. The NMPC/NSTC successfully brought NO emissions under control after a step disturbance in the feed composition stream. Unlike those neural network-based control applications in the control field, which typically involve a time-consuming and ineffective network learning process, training NMPC and NSTC was simple with existing steady-state experimental data. With a limited number of neurons and steady-state training patterns, the network training converged readily. The active control movement was incorporated into the controller indirectly thought the use of the mathematical function and programming technique. This approach proved to be reliable and effective in coping with the time-dependent characteristics associated with the inherently stable process.

The process simulator for the combustion system was built and validated in Chapter 7. A neural network was used to model the staged combustion process inputs and outputs. The well-trained neural network model was embedded into the Visual Basic simulator interface. Although the neural network was trained with a steady state historical database, the transient network responses were enabled by additional programming tools and techniques. The process simulator was especially useful for some operating ranges that are physically restricted in the real experimental process. Through the simulation runs, the superiority of the neural network-based process controller over the PID controller was demonstrated in terms of both process settling time and compatibility with the Smith time-delay algorithm. With the process simulator, more insight and process control knowledge were effectively obtained.

Finally, it is important to mention that the control applications originally designed and developed in this study are not limited to the combustion process. With sufficient steady-state training data, the control systems are applicable to processes in others engineering and science fields.

8.2 Future Work

Although the research objectives outlined in Chapter 1 have been achieved, it is important to point out the potential work to be done in the future. The ability of the neural network-based controller system to handle several inputs, as well as the general improvement possible if sampling/analysis delay were shortened, suggest that an online, multi-species, fast analyzer working together with the new controllers would be helpful.

The main focus now is shifted toward a real-time, online, optical-based species monitoring system, namely a compact Fourier transform infrared (FTIR) spectrometer. Early in this research, a short collaboration existed with Optomechanical Enterprise, Inc. This firm developed a wavefront-dividing FTIR interferometer that was compact and online. The unit had the potential for near-simultaneous detection of multiple species of interest to staged combustion; namely, NO, CO, CO_2, C_2H_2, C_6H_6, etc. As demonstrated in Chapter 7, a multiple input neural network-based controller is very robust. Therefore, a marriage of the FTIR unit with the neural network controller seemed logical.

The FTIR system was interfaced to the staged combustor. Infrared transmitting windows were mounted on the exhaust chamber. A specialty heat exchanger to cool the gases just upstream of the detection area was installed. Operation of the combustor with this modified exhaust system proved to be difficult, though not impossible.

Considerable effort was spent on the electronic-interfacing of the FTIR to the neural network PC. The FTIR itself interfaced to a laptop PC, which was connected to the main PC. Error-free transfer of information between the interferometer laptop and the main PC was successfully demonstrated.

It was found that the process of collecting the optical spectrum was long and thus inhibited the real-time measurement method. The optical spectrum of the combustion effluent was very complex, and suggested several unknown compounds, some of which were beyond the capability of the optical detector tó characterize. In addition, some unburned hydrocarbons had their detectable wavelengths in a very close proximity and thus made it even more difficult to characterize.

Operation of the FTIR spectrometer with the combustor proved to be problematic, and was suspended. Research on the neural network-based control proceeded. A fuller process control investigation could be carried out by experiment if and when the FTIR spectrometer becomes available and reliable. Many benefits could be potentially gained by using the FTIR spectrometer for monitoring sample gases from the combustor while controlling with the neural networks.

First, the long lag time in the hydrocarbons sampling and analysis process could be minimized through the use of the FTIR spectrometer. Typically, the unburned hydrocarbons are measured through a batch process by a gas chromatography method. The long gas chromatogram batch process inhibits the process monitoring and control of the hazardous unburned hydrocarbons.

Second, a process control of some hydrocarbons such as CH_4, C_2H_2, C_2H_4, and C_6H_6 could be achieved. As demonstrated by Mao and Barat (1996), in addition to the dramatic increase of the CO in the first stage as the first-stage equivalence ratio ϕ_1 increases, the higher ϕ_1 also results in increased concentrations of unburned hydrocarbons such as CH_4, C_2H_2, and C_2H_4. These species, however, can be consumed in the second stage. Along with the NO, CO, and CO_2 measurements, it would be helpful to have a fast and reliable mean to measure these unburned hydrocarbon compounds.

Such an optical measuring method would integrate well into the neural network-based controller for a multi-species control exercise. Data collected through the FTIR spectrometer could be used for developing and training the neural network-based controller as well as for expanding the current neural network

APPENDIX

VISUAL BASIC 6.0 SOURCE CODES

The operating and simulation interfaces are written in Visual Basic 6.0. The original source codes developed in this study, along with essential subroutines from NeuroDimension, Inc. (NeuroSolutions) and Keithley Instruments, Inc. (DriverLinx), are listed in this section.

```
Option Explicit
Const LogicalDevice As Integer = 0
Const LogicalChannel As Integer = 0
Const ChannelGain As Single = -1#
Private NumberOfChannels As Integer
Private LogicalChannels(0 To 1) As Integer
Private ChannelGains(0 To 1) As Single
Const SamplingFrequency As Single = 2#
Const SamplingPeriod As Single = 1 / SamplingFrequency
Const NumberOfBuffers As Integer = 1
Const NumberOfSamples As Integer = 2
Const FullSweep As Single = NumberOfSamples *
SamplingPeriod
Dim DataArray() As Single
Private strStatus As String
Const msgRunning As String = "Running"
Const msgStopped As String = "Stopped"
Const BackGroundForeGround As Integer = 1
Const Digital As Boolean = False
Const YAxisOffset = 0

Private Sub chkSimplePID_Click()
txtIntegral.Text = 0
txtDerivative.Text = 0
End Sub

Private Sub cmdNN_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual basics\pid_o2\process.dll"
nn.loadWeights "d:\my visual basics\pid_o2\process.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\pid_o2\processnew.dll"
nn.loadWeights "c:\my documents\neural
networks\pid_o2\processnew.nsw"
End If
Dim inputdata(0 To 0, 0 To 0) As Variant
inputdata(0, 0) = Val(txtPHI1.Text)
nn.inputdata = inputdata
Dim OutputData As Variant
OutputData = nn.getResponse
txtO2.Text = OutputData(0, 0)
Me.txtO2.Refresh
End Sub

Private Sub cmdNN2_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual basics\pid_o2\phi_N2.dll"
nn.loadWeights "d:\my visual basics\pid_o2\phi_N2.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\simulationck\n2andphi.dll"
nn.loadWeights "c:\my documents\neural
networks\simulationck\n2andphi.nsw"
End If
Dim inputdata(0 To 0, 0 To 0) As Variant
inputdata(0, 0) = Val(txtPHI1.Text)
nn.inputdata = inputdata
Dim OutputData As Variant
OutputData = nn.getResponse
txtN2.Text = OutputData(0, 0)
Me.txtN2.Refresh
End Sub

Private Sub cmdNN3_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual basics\pid_o2\O2_PHI.dll"
nn.loadWeights "d:\my visual basics\pid_o2\O2_PHI.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\pid_o2\O2PHI.dll"
nn.loadWeights "c:\my documents\neural
networks\pid_o2\O2PHI.nsw"
End If
Dim inputdata(0 To 0, 0 To 0) As Variant
inputdata(0, 0) = Val(txtpfrO2.Text)
nn.inputdata = inputdata
Dim OutputData As Variant
OutputData = nn.getResponse
txtPredictPHI1.Text = OutputData(0, 0)
Me.txtPredictPHI1.Refresh
End Sub

Private Sub cmdNN4_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual
basics\pid_o2\NH3_phi_NO_ratio.dll"
nn.loadWeights "d:\my visual
basics\pid_o2\NH3_phi_NO_ratio.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\PID_O2\NH3_PHI_NO_ratio.dll"
nn.loadWeights "c:\my documents\neural
networks\PID_O2\NH3_PHI_NO_ratio.nsw"
End If
Dim inputdata(0 To 1, 0 To 0) As Variant
inputdata(0, 0) = Val(txtPHI1.Text)
inputdata(1, 0) = Val(txtSmithNO.Text) +
Val(txtSmithDeltaNO.Text)
nn.inputdata = inputdata
Dim OutputData As Variant
OutputData = nn.getResponse
Dim bufferRatio
bufferRatio = CDec(OutputData(0, 0))
txtPredictF2F1.Text = Left$(bufferRatio, 6)
Me.txtPredictF2F1.Refresh
txtSmithF2F1.Text = Val(txtPredictF2F1.Text)
End Sub

Private Sub cmdNN5_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual
basics\pid_o2\NH3_NO_Ratio_phi.dll"
nn.loadWeights "d:\my visual
basics\pid_o2\NH3_NO_Ratio_phi.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\PID_O2\setpoint_nh3_phi1.dll"
nn.loadWeights "c:\my documents\neural
networks\PID_O2\setpoint_nh3_phi1.nsw"
End If
Dim inputdata(0 To 1, 0 To 0) As Variant
inputdata(0, 0) = Val(txtSetpoint.Text)
inputdata(1, 0) = Val(txtPredictF2F1.Text)
nn.inputdata = inputdata
Dim OutputData As Variant
OutputData = nn.getResponse
txtPredictPHI1.Text = OutputData(0, 0)
Me.txtPredictPHI1.Refresh
txtSmithPHI1.Text = Val(txtPredictPHI1.Text)
End Sub

Private Sub cmdNN6_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
```

```
nn.dllPathName = "d:\my visual
basics\pid_o2\NH3_phi_NO_C2H2.dll"
nn.loadWeights "d:\my visual
basics\pid_o2\NH3_phi_NO_C2H2.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\PID_O2\processA.dll"
nn.loadWeights "c:\my documents\neural
networks\PID_O2\processA.nsw"
End If
Dim inputdata(0 To 1, 0 To 0)  As Variant
inputdata(0, 0) = Val(txtPHI1.Text)
inputdata(1, 0) = Val(txtF2F1.Text)
nn.inputdata = inputdata
Dim OutputData  As Variant
OutputData = nn.getResponse
txtNO.Text = OutputData(0, 2)
Me.txtNO.Refresh
txtCO2.Text = OutputData(0, 3)
Me.txtCO2.Refresh
txtCO.Text = OutputData(0, 0)
Me.txtCO.Refresh
txtCO2f.Text = OutputData(0, 1)
Me.txtCO2f.Refresh
End Sub

Private Sub cmdNN7_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual basics\pid_o2\phi_CO.dll"
nn.loadWeights "d:\my visual basics\pid_o2\phi_CO.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\PID_O2\PHI_CO.dll"
nn.loadWeights "c:\my documents\neural
networks\PID_O2\PHI_CO.nsw"
End If
Dim inputdata(0 To 0, 0 To 0)  As Variant
inputdata(0, 0) = Val(txtPHI1.Text)
nn.inputdata = inputdata
Dim OutputData  As Variant
OutputData = nn.getResponse
txtCO.Text = OutputData(0, 0)
Me.txtCO.Refresh
End Sub

Private Sub cmdReset_Click()
txtA1.Text = 28
txtActualA1.Text = 28
txtF1.Text = 43
txtActualF1.Text = 43
txtF2.Text = 5.7
txtActualF2.Text = 3.6
txtN2.Text = 14.06
txtActualN2.Text = 14.06
txtA2.Text = 0
txtActualA2.Text = 0
txtTimer.Text = 0
FlagDist = 0
Tcounter = 0
txtLag.Text = 60
txtSetpoint.Text = 450
OptNO.Value = True
txtBias.Text = 50
txtPHIoSetpoint.Text = 0.91
txtGain.Text = 0
End Sub

Private Sub cmdResetPID_Click()
PIDData.ErrAccum = 0
PIDData.Err0 = 0
PIDData.Err1 = 0
PIDData.Err2 = 0
PIDData.Err3 = 0
End Sub

Private Sub cmdSet_Click()
Tcounter = 0
TdiffF1 = Val(txtF1.Text) - Val(txtActualF1.Text)
TdiffF2 = Val(txtF2.Text) - Val(txtActualF2.Text)
BufferF1 = Val(txtActualF1.Text)
BufferF2 = Val(txtActualF2.Text)
FlagDist = 1
End Sub

Private Sub cmdSetA1_Click()
TcounterA1 = 0
TdiffA1 = Val(txtA1.Text) - Val(txtActualA1.Text)
BufferA1 = Val(txtActualA1.Text)
FlagSetA1 = 1
End Sub

Private Sub cmdSmithNN8_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual
basics\pid_o2\NH3_phi_NO_C2H2.dll"
nn.loadWeights "d:\my visual
basics\pid_o2\NH3_phi_NO_C2H2.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\PID_O2\NH3_PHI_NO_C2H2.dll"
nn.loadWeights "c:\my documents\neural
networks\PID_O2\NH3_PHI_NO_C2H2.nsw"
End If
Dim inputdata(0 To 1, 0 To 0)  As Variant
inputdata(0, 0) = Val(txtSmithPHI1Delay.Text)
inputdata(1, 0) = Val(txtSmithF2F1.Text)
nn.inputdata = inputdata
Dim OutputData  As Variant
OutputData = nn.getResponse
txtSmithDelayNO.Text = OutputData(0, 0)
Me.txtSmithDelayNO.Refresh
End Sub

Private Sub cmdSmithNN9_Click()
Dim nn As New NSRecallNetwork
If Bypass = 1 Then
nn.dllPathName = "d:\my visual
basics\pid_o2\NH3_phi_NO_C2H2.dll"
nn.loadWeights "d:\my visual
basics\pid_o2\NH3_phi_NO_C2H2.nsw"
Else
nn.dllPathName = "c:\my documents\neural
networks\PID_O2\NH3_PHI_NO_C2H2.dll"
nn.loadWeights "c:\my documents\neural
networks\PID_O2\NH3_PHI_NO_C2H2.nsw"
End If
Dim inputdata(0 To 1, 0 To 0)  As Variant
inputdata(0, 0) = Val(txtSmithPHI1.Text)
inputdata(1, 0) = Val(txtSmithF2F1.Text)
nn.inputdata = inputdata
Dim OutputData  As Variant
OutputData = nn.getResponse
txtSmithNO.Text = OutputData(0, 0)
Me.txtSmithNO.Refresh
End Sub

Private Sub cmdSmithStart_Click()
Timer3.Enabled = True
```

```
End Sub

Private Sub cmdSmithStop_Click()
Timer3.Enabled = False
End Sub

Private Sub cmdStart_Click()
Timer1.Enabled = True
End Sub

Private Sub cmdStartP_Click()
InputLast = Val(txtpfrO2.Text)
InputDF = Val(txtpfrO2.Text)
Feedback = 0
Timer2.Enabled = True
End Sub

Private Sub cmdStop_Click()
Timer1.Enabled = False
End Sub

Private Sub Command1_Click()
End
End Sub

Private Sub cmdStopP_Click()
Timer2.Enabled = False
End Sub

Private Sub DriverLINXSR1_DataLost(task As Integer, device
As Integer, subsystem As Integer, mode As Integer, bufIndex As
Long, bufElement As Long)
StopDriverLINXIO DriverLINXSR1
lblStatus.Caption = "Data Lost"
MsgBox "Data Lost!", vbOKOnly, Me.Name
cmdExit_Click
End Sub

Private Sub ExitItem_Click()
Command1_Click
End Sub

Private Sub Form_Load()
Bypass = 2
If Bypass = 1 Then
Dim DLDriverName As String
Dim DLResultCode As Integer
Dim DLMessage As String
Dim channelNumber As Integer
Dim i As Integer
Dim Model As String
Dim Msg As String
CenterForm Me
DLDriverName = OpenDriverLINXDriver(DriverLINXSR1, "",
True)
If DLDriverName = "" Then
MsgBox "DriverLINX driver not opened.", vbOKOnly,
Me.Name
End
End If
DLResultCode = InitializeDriverLINXDevice(DriverLINXSR1,
LogicalDevice)
If DLResultCode <> DL_NoErr Then
ShowDriverLINXStatus DriverLINXSR1
End
End If
Model = GetModelName(DriverLINXSR1, DriverLINXLDD1)
If Not HasDriverLINXSubsystem(DriverLINXSR1,
DriverLINXLDD1, DL_AO) Then
Msg = "This " & Model & " does not support output input"
MsgBox Msg, vbOKOnly, Me.Name
End
End If
If Not ((DoesDeviceSupportDMA(DriverLINXSR1,
DriverLINXLDD1, DL_AO)) Or
(DoesDeviceSupportIRQ(DriverLINXSR1, DriverLINXLDD1,
DL_AO))) Then
Msg = "This " & Model & " does not support waveform quality
analog output."
MsgBox Msg, vbOKOnly, Me.Name
End
End If
GetDriverLINXStatus DriverLINXSR1, DLMessage
lblStatus.Caption = DLMessage
InitCRT pictureCRT, FullSweep, 20, Digital, YAxisOffset
channelNumber =
HowManyDriverLINXLogicalChannels(DriverLINXSR1,
DriverLINXLDD1, DL_AO)
For i = 0 To 1
InitChannelSelector editChannelSelector(i),
vscrollChannelSelector(i), LogicalChannel, channelNumber
Next i
If Not editChannelSelector(0).Enabled Then
Msg = "This " & Model & " has no analog output channels."
MsgBox Msg, vbOKOnly, Me.Name
End
End If
LogicalChannels(0) = editChannelSelector(0).Text
LogicalChannels(1) = LogicalChannels(0)
vscrollChannelSelector(1).Value = 1
NumberOfChannels = ((LogicalChannels(1) -
LogicalChannels(0)) + 1)
ReDim DataArray(0 To NumberOfChannels - 1, 0 To
NumberOfSamples - 1)
ChannelGains(0) = ChannelGain
ChannelGains(1) = ChannelGain
SetupDriverLINXMultiChannelStartStopList DriverLINXSR1,
DriverLINXLDD1, LogicalDevice, DL_AO, LogicalChannels(),
ChannelGains(), SamplingFrequency, NumberOfSamples,
NumberOfBuffers, BackGroundForeGround
MSComm1.PortOpen = True
MSComm2.PortOpen = True
End If
Set application = GetObject(, "excel.application")
Derivative = Val(txtDerivative.Text)
ResetRate = Val(txtIntegral.Text)
DFilter = Val(txtFilter.Text)
End Sub

Private Sub cmdWrite_Click()
Dim DLMessage As String
Dim DLResultCode As Integer
Dim Result As Long
Dim Output1
Dim Output2
Output1 = Val(Text1.Text)
Output2 = Val(Text2.Text)
CreateSineWaves Output1, Output2, NumberOfSamples,
DataArray(), NumberOfChannels    '1 stands for 1 volt peak
voltage
Result = PutDriverLINXAIBuffer(DriverLINXSR1, 0,
NumberOfSamples * NumberOfChannels, DataArray())
DriverLINXSR1.Refresh
DLResultCode = GetDriverLINXStatus(DriverLINXSR1,
DLMessage)
If DLResultCode <> DL_NoErr Then
Debug.Print "Result Code = " & DLResultCode
strStatus = DLMessage
cmdWrite.Enabled = True
End If
```

```
lblStatus.Caption = strStatus
End Sub

Private Sub DriverLINXSR1_ServiceStart(task As Integer,
device As Integer, subsystem As Integer, mode As Integer)
strStatus = msgRunning
lblStatus.Caption = strStatus
cmdWrite.Enabled = False
End Sub

Private Sub DriverLINXSR1_ServiceDone(task As Integer,
device As Integer, subsystem As Integer, mode As Integer)
strStatus = msgStopped
lblStatus.Caption = strStatus
cmdWrite.Enabled = True
End Sub

Private Sub cmdExit_Click()
Dim DLStatusCode As Integer
Text1.Text = 0
Text2.Text = 0
cmdWrite_Click
DLStatusCode = StopDriverLINXIO(DriverLINXSR1)
If DLStatusCode = DL_NoErr Then
Do
DoEvents
Loop Until cmdWrite.Enabled
End If
CloseDriverLINXDriver DriverLINXSR1
Unload Me
End Sub

Private Sub optNOCO_Click()
txtPHIoSetpoint.Text = 1.14
End Sub

Private Sub StartControllerItem_Click()
cmdStartP_Click
End Sub

Private Sub StartSamplingItem_Click()
cmdStart_Click
End Sub

Private Sub StopControllerItem_Click()
cmdStopP_Click
End Sub

Private Sub StopSamplingItem_Click()
cmdStop_Click
End Sub

Private Sub Timer1_Timer()
If chkChemkin.Value = 1 Then
txtActualA1Add.Text = Val(txtA1add.Text)
If FlagSetA1 = 1 Then
TcounterA1 = TcounterA1 + 0.2
TnA1 = TcounterA1
TkA1 = 1 + (Exp(TnA1 - 3) - Exp(-TnA1 + 3)) / (Exp(TnA1 - 3)
+ Exp(-TnA1 + 3))
TtanhA1 = TkA1 / 2
txtActualA1.Text = BufferA1 + (TdiffA1 * TkA1 / 2)
End If
If FlagDist = 1 Then
Tcounter = Tcounter + 0.2
Tn = Tcounter
Tk = 1 + (Exp(Tn - 3) - Exp(-Tn + 3)) / (Exp(Tn - 3) + Exp(-Tn
+ 3))
Ttanh = Tk / 2
txtActualF1.Text = BufferF1 + (TdiffF1 * Tk / 2)
txtActualF2.Text = BufferF2 + (TdiffF2 * Tk / 2)
End If
txtActualN2.Text = Val(txtN2.Text)
txtActualA2.Text = txtA2.Text
MolarA1 = (((txtActualA1 * 0.26732) - 0.07857) +
(txtActualA1Add.Text * 4 / 11)) * 0.019546617 * 0.76
MassA1 = MolarA1 * 28.84
MolarF1 = ((txtActualF1.Text * 0.0187 * 0.76) - 0.0528) *
0.019546617
MassF1 = MolarF1 * 28#
MolarA2 = (txtActualA2.Text / 179.509572)
MassA2 = MolarA2 * 28.84
VolF2 = 0.00005988 + 0.00497929 * Val(txtActualF2.Text) -
0.00163899 * Val(txtActualF2.Text) ^ 2 + 0.00053131 *
Val(txtActualF2.Text) ^ 3 - 0.00004522 * Val(txtActualF2.Text)
^ 4 + 0.00000124 * Val(txtActualF2.Text) ^ 5
MolarF2 = VolF2 * 0.019546617
Dim dummyf2f1
dummyf2f1 = CDec(MolarF2 / MolarF1)
txtF2F1.Text = Left$(dummyf2f1, 6)
txtActualF2F1.Text = Val(txtF2F1.Text)
txtPHI1 = (3 * MolarF1 + 0.75 * MolarF2) / (0.21 * MolarA1)
txtPHIo = (3 * MolarF1 + 0.75 * MolarF2) / (0.21 * (MolarA1 +
MolarA2))
cmdNN2_Click
MolarN2 = (txtActualN2.Text * 4 / 11) * 0.01954
MassN2 = MolarN2 * 28#
txtPsrFlow.Text = MassA1 + MassF1 + MassN2
txtInjFlow.Text = MassA2
txtXO2 = MolarA1 * 0.21 / (MolarA1 + MolarF1 + MolarN2 +
MolarF2)
txtXF1 = CDec(MolarF1 / (MolarA1 + MolarF1 + MolarN2 +
MolarF2))
txtXN2 = (0.79 * MolarA1 + MolarN2) / (MolarA1 + MolarF1 +
MolarN2 + MolarF2)
cmdNN6_Click
Static i
If i > Val(txtLag.Text) Then
i = 0
End If
lagO2(i) = Val(txtO2.Text)
lagCO2(i) = Val(txtCO2.Text)
lagNO(i) = Val(txtNO.Text)
lagCO(i) = Val(txtCO.Text)
lagCO2F(i) = Val(txtCO2f.Text)
If i < Val(txtLag.Text) Then '9 or 99
txtpfrO2.Text = lagO2(i + 1)
txtpfrCO2.Text = lagCO2(i + 1)
txtpfrNO.Text = lagNO(i + 1)
txtpsrCO.Text = lagCO(i + 1)
txtPSRCO2.Text = lagCO2F(i + 1)
Else
txtpfrO2.Text = lagO2(0)
txtpfrCO2.Text = lagCO2(0)
txtpfrNO.Text = lagNO(0)
txtpsrCO.Text = lagCO(0)
txtPSRCO2.Text = lagCO2F(0)
End If
i = i + 1
Else
MolarA1 = (((txtA1 * 0.26732) - 0.07857) + (txtA1add.Text * 4
/ 11)) * 0.019546617 * 0.76
MassA1 = MolarA1 * 28.84
MolarF1 = ((txtF1.Text * 0.0187 * 0.76) - 0.0528) *
0.019546617
MassF1 = MolarF1 * 28#
MolarN2 = (txtN2.Text * 4 / 11) * 0.019546617
MassN2 = MolarN2 * 28#
MolarA2 = (txtA2.Text / 179.509572)
MassA2 = MolarA2 * 28.84
```

```
VolF2 = 0.00005988 + 0.00497929 * Val(txtF2.Text) -
0.00163899 * Val(txtF2.Text) ^ 2 + 0.00053131 *
Val(txtF2.Text) ^ 3 - 0.00004522 * Val(txtF2.Text) ^ 4 +
0.00000124 * Val(txtF2.Text) ^ 5
MolarF2 = VolF2 * 0.019546617
txtF2F1.Text = MolarF2 / MolarF1
Dim dummyf2f1Actual
dummyf2f1Actual = CDec(MolarF2 / MolarF1)
txtF2F1.Text = Left$(dummyf2f1Actual, 6)
txtPHI1 = (3 * MolarF1 + 0.75 * MolarF2) / (0.21 * MolarA1)
txtPHIo = (3 * MolarF1 + 0.75 * MolarF2) / (0.21 * (MolarA1 +
MolarA2))
Dim Fluke, Fluke1, Optical, Firstpos, Wordlen, Final
Dim FTemp, FO2, FCO2, FNO, FCO
MSComm1.Output = "*trg"
MSComm1.Output = Chr$(13)
MSComm1.Output = "last?"
MSComm1.Output = Chr$(13)
Fluke1 = MSComm1.Input
Label57.Caption = Fluke1
Label41.Caption = Mid$(Fluke1, 6, 9)
Label42.Caption = Mid$(Fluke1, 17, 9)
Label43.Caption = Mid$(Fluke1, 28, 9)
Label44.Caption = Mid$(Fluke1, 39, 9)
Label45.Caption = Mid$(Fluke1, 50, 9)
FTemp = Val(Label41.Caption)
FO2 = CDec(Val(Label42.Caption) * 19.5 / 0.785)
FCO2 = CDec(Val(Label43.Caption) * 10.4 / 4.38)
FNO = CDec(Val(Label44.Caption) * 100)
FCO = CDec(Val(Label45.Caption))
txtpsrT.Text = Left$(FTemp, 6)
txtpfrO2.Text = Left$(FO2, 6)
txtpfrCO2.Text = Left$(FCO2, 6)
txtpfrNO.Text = Left$(FNO, 6)
txtpsrCO.Text = Left$(FCO, 6)
Optical = MSComm2.Input
If Len(Optical) > 0 Then
txtC2H2.Text = Left$(Optical, 5)
Else
txtC2H2.Text = txtC2H2.Text
End If
End If
Static j
If j > 29 Then
application.run "module1.poopfirstcolumn", Timer, O2, EA1,
EA1Add, EF1, P, Integral, D, Offset, Bias, EOutput, Setpoint,
Emaxrate, CO2, NO, Tpsr, C2H2, CO, EF2, EA2
application.run "module1.macro1"
j = 0
End If
Timer(j) = Val(txtTimer.Text)
O2(j) = Val(txtpfrO2.Text)
EA1(j) = Val(txtA1.Text)
EA1Add(j) = Val(txtA1add.Text)
EF1(j) = Val(txtActualF1.Text)
P(j) = Val(txtGain.Text)
Integral(j) = Val(txtIntegral.Text)
D(j) = Val(txtDerivative.Text)
Offset(j) = Val(txtOffset.Text)
Bias(j) = Val(txtBias.Text)
EOutput(j) = Val(txtOutput.Text)
Setpoint(j) = Val(txtSetpoint.Text)
Emaxrate(j) = Val(txtMaxRate.Text)
CO2(j) = Val(txtpfrCO2.Text)
NO(j) = Val(txtpfrNO.Text)
Tpsr(j) = Val(txtpsrT.Text)
C2H2(j) = Val(txtPSRCO2.Text)
CO(j) = Val(txtpsrCO.Text)
EF2(j) = Val(txtF2.Text)
EA2(j) = Val(txtA2.Text)
j = j + 1
Static k
If k > Val(txtLag.Text) Then
k = 0
End If
lagSmithPHI1(k) = Val(txtPHI1.Text)
If k < Val(txtLag.Text) Then
txtSmithPHI1Delay.Text = lagSmithPHI1(k + 1)
Else
txtSmithPHI1Delay.Text = lagSmithPHI1(0)
End If
k = k + 1
txtTimer.Text = Val(txtTimer.Text) + Val(Timer1.Interval) *
0.001
End Sub

Private Sub Timer2_Timer()
Dim ControllerInterval
ControllerInterval = Val(txtTimer2.Text)
If ControllerInterval > (Val(txtPeriods.Text) - 2) Then
txtTimer2.Text = 0
If chkCim.Value = 1 Then
MolarA1 = (((txtA1 * 0.26732) - 0.07857) + (txtA1add.Text * 4
/ 11)) * 0.019546617 * 0.76
MassA1 = MolarA1 * 28.84
MolarF1 = (((txtF1.Text) * 0.0187 * 0.76) - 0.0528) *
0.019546617
MassF1 = MolarF1 * 28#
VolF2 = 0.00005988 + 0.00497929 * Val(txtActualF2.Text) -
0.00163899 * Val(txtActualF2.Text) ^ 2 + 0.00053131 *
Val(txtActualF2.Text) ^ 3 - 0.00004522 * Val(txtActualF2.Text)
^ 4 + 0.00000124 * Val(txtActualF2.Text) ^ 5
MolarF2 = VolF2 * 0.019546617
MassF2 = MolarF2 * 17
Dim dummyf2f1
dummyf2f1 = CDec(MolarF2 / MolarF1)
txtF2F1.Text = Left$(dummyf2f1, 6)
MolarA2 = (txtA2.Text) / 179.509572
MassA2 = MolarA2 * 28.84
End If
txtActualPHIo.Text = Val(txtPHIo.Text)
If OptO2.Value = True Then
txtActual.Text = txtpfrO2.Text
ElseIf OptNO.Value = True Then
txtActual.Text = txtpfrNO.Text
ElseIf optNOCO.Value = True Then
txtActual.Text = txtpfrNO.Text
ElseIf OptC2H2.Value = True Then
txtActual.Text = Val(txtC2H2.Text) / 1000
End If
Dim Output, Offset
Offset = 100 * Abs((Val(txtSetpoint.Text) - Val(txtActual.Text))
/ Val(txtSetpoint.Text))
Static b
If b > 7 Then
If Offset > Val(txtOffset.Text) Then
If chkNN3.Value = 1 Then
cmdNN3_Click
Dim Fuel, Air, AirAdd, SCFM, Reading, BiasDummy
Fuel = 0.21 * (Val(txtPredictPHI1.Text)) * (MolarA1 +
MolarA1Add) / 3
Air = 3 * Fuel / (0.21 * Val(txtPHI1Setpoint.Text)) 'Setpoint
phi used to be = 0.64
AirAdd = Air - (((Val(txtA1.Text) * 0.26732) - 0.07857) *
0.01954 * 0.76)
SCFM = AirAdd * 82.05 * 294.26 / 471.9744432
Reading = 11 * SCFM / (4 * 0.76)
BiasDummy = Reading * 100 / 10
If BiasDummy > 100 Then
txtBias.Text = 100
```

```
ElseIf BiasDummy < 0 Then
txtBias.Text = 0
Else
txtBias.Text = BiasDummy
End If
b = 0
End If
End If
End If
b = b + 1
If chkNNNO.Value = 1 Then
cmdNN4_Click
cmdNN5_Click
Dim NOMolarA1, NOA1Add, NOSCFM, NOReading,
NOBiasDummy
NOMolarA1 = ((3 * MolarF1 / 0.21) + (0.75 * MolarF2 / 0.21)) /
Val(txtPredictPHI1.Text)
NOA1Add = NOMolarA1 - (((txtA1 * 0.26732) - 0.07857)) *
0.019546617 * 0.76
NOSCFM = NOA1Add * 82.05 * 294.26 / 471.9744432
NOReading = 11 * NOSCFM / (4 * 0.76)
NOBiasDummy = NOReading * 100 / 10
If NOBiasDummy > 100 Then
txtBias.Text = 100
ElseIf NOBiasDummy < 0 Then
txtBias.Text = 0
Else
txtBias.Text = NOBiasDummy
End If
End If
If chkCim.Value = 1 Then
ProcessPID
Dim k
k = PIDData.Current - PIDData.Setpoint
If AOMultChForm.chkReset.Value = 1 Then
If Abs(k) < Val(txtWindup.Text) Then
End If
End If
Dim Foutput
If OptO2.Value = True Then
Foutput = Val(txtOutput.Text) + Val(txtBias.Text)
Else
Foutput = ((-1) * Val(txtOutput.Text)) + Val(txtBias.Text)
End If
If Foutput > 100 Then
txtFinalOutput.Text = 100
ElseIf Foutput < 0 Then
txtFinalOutput.Text = 0
Else
txtFinalOutput.Text = Foutput
End If
End If
txtSignal.Text = (0.01 * (Val(txtMaxValve.Text) -
Val(txtMinValve.Text)) * Val(txtFinalOutput.Text)) +
Val(txtMinValve.Text)
txtflow.Text = 10 * (Val(txtSignal.Text) -
Val(txtMinValve.Text)) / (Val(txtMaxValve.Text) -
Val(txtMinValve.Text))
DummyA2 = ((((3 * MolarF1 / 0.21) + (0.75 * MolarF2 / 0.21))
/ Val(txtPHIoSetpoint.Text)) - (MolarA1 + 0)) / 0.019546617
txtFlowA2.Text = 8 * DummyA2 / (3 * 0.76)
txtSignal2.Text = (0.1 * (Val(txtMaxValve2.Text) -
Val(txtMinValve2.Text)) * Val(txtFlowA2.Text)) +
Val(txtMinValve2.Text)
If optNOCO.Value = True Then
txtA2.Text = Val(txtSignal2.Text)
End If
txtA1add.Text = Val(txtflow.Text)
If optNOCO.Value = True Then
txtA2.Text = Val(txtFlowA2.Text)
End If
Text1.Text = Val(txtSignal.Text)
Text2.Text = Val(txtSignal2.Text)
If Bypass = 1 Then
cmdWrite_Click
End If
Else
txtTimer2.Text = Val(txtTimer2.Text) + 1
End If
End Sub

Private Sub Timer3_Timer()
cmdSmithNN8_Click
txtSmithDeltaNO.Text = Val(txtpfrNO.Text) -
Val(txtSmithDelayNO.Text)
cmdSmithNN9_Click
End Sub

Private Sub txtA1add_Change()
End Sub

Private Sub UpDown1_DownClick()
txtA1.Text = Val(txtA1.Text) - 1
FlagA1 = 0
BufferA1 = Val(txtActualA1.Text)
End Sub

Private Sub UpDown1_UpClick()
txtA1.Text = Val(txtA1.Text) + 1
FlagA1 = 1
BufferA1 = Val(txtActualA1.Text)
End Sub

Private Sub UpDown10_DownClick()
txtMaxValve.Text = Val(txtMaxValve.Text) - 0.1
End Sub

Private Sub UpDown10_UpClick()
txtMaxValve.Text = Val(txtMaxValve.Text) + 0.1
End Sub

Private Sub UpDown11_DownClick()
txtOffset.Text = Val(txtOffset.Text) - 5
End Sub

Private Sub UpDown11_UpClick()
txtOffset.Text = Val(txtOffset.Text) + 5
End Sub

Private Sub UpDown12_DownClick()
txtIntegral.Text = Val(txtIntegral.Text) - 0.1
ResetRate = Val(txtIntegral.Text)
End Sub

Private Sub UpDown12_UpClick()
txtIntegral.Text = Val(txtIntegral.Text) + 0.1
ResetRate = Val(txtIntegral.Text)
End Sub

Private Sub UpDown13_DownClick()
txtDerivative.Text = Val(txtDerivative.Text) - 1
Derivative = Val(txtDerivative.Text)
End Sub

Private Sub UpDown13_UpClick()
txtDerivative.Text = Val(txtDerivative.Text) + 1
Derivative = Val(txtDerivative.Text)
End Sub

Private Sub UpDown14_DownClick()
```

```
txtFilter.Text = Val(txtFilter.Text) - 10
DFilter = Val(txtFilter.Text)
End Sub

Private Sub UpDown14_UpClick()
txtFilter.Text = Val(txtFilter.Text) + 10
DFilter = Val(txtFilter.Text)
End Sub

Private Sub UpDown15_DownClick()
txtPeriods.Text = Val(txtPeriods.Text) - 1
End Sub

Private Sub UpDown15_UpClick()
txtPeriods.Text = Val(txtPeriods.Text) + 1
End Sub

Private Sub UpDown18_DownClick()
txtF2.Text = Val(txtF2.Text) - 0.2
End Sub

Private Sub UpDown18_UpClick()
txtF2.Text = Val(txtF2.Text) + 0.2
End Sub

Private Sub UpDown19_DownClick()
txtMaxValve2.Text = Val(txtMaxValve2.Text) - 0.1
End Sub

Private Sub UpDown19_UpClick()
txtMaxValve2.Text = Val(txtMaxValve2.Text) + 0.1
End Sub

Private Sub UpDown2_DownClick()
txtF1.Text = Val(txtF1.Text) - 0.5
FlagF1 = 0
BufferF1 = Val(txtActualF1.Text)
End Sub

Private Sub UpDown2_UpClick()
txtF1.Text = Val(txtF1.Text) + 0.5
FlagF1 = 1
BufferF1 = Val(txtActualF1.Text)
End Sub

Private Sub UpDown20_DownClick()
txtMinValve2.Text = Val(txtMinValve2.Text) - 0.1
End Sub

Private Sub UpDown20_UpClick()
txtMinValve2.Text = Val(txtMinValve2.Text) + 0.1
End Sub

Private Sub UpDown21_DownClick()
txtBias2.Text = Val(txtBias2.Text) - 5
End Sub

Private Sub UpDown21_UpClick()
txtBias2.Text = Val(txtBias2.Text) + 5
End Sub

Private Sub UpDown22_DownClick()
txtPHIoSetpoint.Text = Val(txtPHIoSetpoint.Text) - 0.01
End Sub

Private Sub UpDown22_UpClick()
txtPHIoSetpoint.Text = Val(txtPHIoSetpoint.Text) + 0.01
End Sub

Private Sub UpDown23_DownClick()
txtF2F1.Text = Val(txtF2F1.Text) - 0.05
End Sub

Private Sub UpDown23_UpClick()
txtF2F1.Text = Val(txtF2F1.Text) + 0.05
End Sub

Private Sub UpDown24_DownClick()
txtPHI1Setpoint.Text = Val(txtPHI1Setpoint.Text) - 0.05
End Sub

Private Sub UpDown24_UpClick()
txtPHI1Setpoint.Text = Val(txtPHI1Setpoint.Text) + 0.05
End Sub

Private Sub UpDown25_DownClick()
txtSmithInterval.Text = Val(txtSmithInterval.Text) - 1
End Sub

Private Sub UpDown25_UpClick()
txtSmithInterval.Text = Val(txtSmithInterval.Text) + 1
End Sub

Private Sub UpDown26_DownClick()
txtLag.Text = Val(txtLag.Text) - 5
End Sub

Private Sub UpDown26_UpClick()
txtLag.Text = Val(txtLag.Text) + 5
End Sub

Private Sub UpDown3_DownClick()
txtN2.Text = Val(txtN2.Text) - 0.3
FlagN2 = 0
BufferN2 = Val(txtActualN2.Text)
End Sub

Private Sub UpDown3_UpClick()
txtN2.Text = Val(txtN2.Text) + 0.3
FlagN2 = 1
BufferN2 = Val(txtActualN2.Text)
End Sub

Private Sub UpDown4_DownClick()
txtA2.Text = Val(txtA2.Text) - 0.2
FlagA2 = 0
BufferA2 = Val(txtActualA2.Text)
End Sub

Private Sub UpDown4_UpClick()
txtA2.Text = Val(txtA2.Text) + 0.2
FlagA2 = 1
BufferA2 = Val(txtActualA2.Text)
End Sub

Private Sub UpDown5_DownClick()
txtA1add.Text = Val(txtA1add.Text) - 0.5
FlagA1Add = 0
BufferA1Add = Val(txtActualA1Add.Text)
End Sub

Private Sub UpDown5_UpClick()
txtA1add.Text = Val(txtA1add.Text) + 0.5
FlagA1Add = 1
BufferA1Add = Val(txtActualA1Add.Text)
End Sub

Private Sub UpDown6_DownClick()
txtBias.Text = Val(txtBias.Text) - 5
End Sub
```

```
Private Sub UpDown6_UpClick()
txtBias.Text = Val(txtBias.Text) + 5
End Sub

Private Sub UpDown7_DownClick()
txtGain.Text = Val(txtGain.Text) - 1
End Sub

Private Sub UpDown7_UpClick()
txtGain.Text = Val(txtGain.Text) + 1
End Sub

Private Sub UpDown8_DownClick()
txtSetpoint.Text = Val(txtSetpoint.Text) - 0.5
End Sub

Private Sub UpDown8_UpClick()
txtSetpoint.Text = Val(txtSetpoint.Text) + 0.5
End Sub

Private Sub UpDown9_DownClick()
txtMinValve.Text = Val(txtMinValve.Text) - 0.1
End Sub

Private Sub UpDown9_UpClick()
txtMinValve.Text = Val(txtMinValve.Text) + 0.1
End Sub

Private Sub vscrollChannelSelector_Change(Index As Integer)
Dim otherChannel As Integer
LogicalChannels(0) = vscrollChannelSelector(0).Value
LogicalChannels(1) = vscrollChannelSelector(1).Value
editChannelSelector(Index).Text = LogicalChannels(Index)
If LogicalChannels(0) > LogicalChannels(1) Then
Select Case Index
Case 0
otherChannel = 1
Case 1
otherChannel = 0
End Select
LogicalChannels(otherChannel) = LogicalChannels(Index)
vscrollChannelSelector(otherChannel).Value =
LogicalChannels(Index)
editChannelSelector(otherChannel).Text =
LogicalChannels(Index)
End If
NumberOfChannels = ((LogicalChannels(1) -
LogicalChannels(0)) + 1)
ReDim DataArray(0 To NumberOfChannels - 1, 0 To
NumberOfSamples - 1) As Single
With DriverLINXSR1
.Sel_chan_start = LogicalChannels(0)
.Sel_chan_stop = LogicalChannels(1)
.Sel_buf_samples = NumberOfSamples * NumberOfChannels
End With
End Sub

Private Sub vscrollChannelSelector_Scroll(Index As Integer)
Dim channels(0 To 1) As Integer
Dim otherChannel As Integer
channels(0) = vscrollChannelSelector(0).Value
channels(1) = vscrollChannelSelector(1).Value
editChannelSelector(Index).Text = channels(Index)
editChannelSelector(Index).Refresh
If channels(0) > channels(1) Then
Select Case Index
Case 0
otherChannel = 1
Case 1
otherChannel = 0
End Select
channels(otherChannel) = channels(Index)
vscrollChannelSelector(otherChannel).Value = channels(Index)
editChannelSelector(otherChannel).Text = channels(Index)
editChannelSelector(otherChannel).Refresh
End If
End Sub
```

```
Global MolarA1
Global MassA1
Global FlagA1
Global BufferA1
Global FlagSetA1
Global TdiffA1
Global TcounterA1
Global TnA1
Global TtanhA1
Global TkA1
Global F1
Global MolarF1
Global MassF1
Global FlagF1
Global BufferF1
Global TdiffF1
Global Tcounter
Global Tn
Global Ttanh
Global Tk
Global FlagDist
Global F2
Global MolarF2
Global MassF2
Global BufferF2
Global VolF2
Global TdiffF2
Global N2
Global MolarN2
Global MassN2
Global FlagN2
Global BufferN2
Global A2
Global MolarA2
Global MassA2
Global FlagA2
Global BufferA2
Global DummyA2
Global A1Add
Global MolarA1Add
Global MassA1Add
Global FlagA1Add
Global BufferA1Add
Global TdiffA1add
Global TcounterA1add
Global TnA1add
Global TtanhA1add
Global TkA1add
Global lag(100) As Variant
Global lagO2(100) As Variant
Global lagCO2(100) As Variant
Global lagNO(100) As Variant
Global lagCO(100) As Variant
Global lagCO2F(100) As Variant
Global lagSmithNO(100) As Variant
Global lagSmithPHI1(120) As Variant
Global application As Object
Global activeworkbook As Object
Global activesheet As Object
Global activerange As Object
Global O2(30) As Variant
Global EA1(30) As Variant
Global EA1Add(30) As Variant
Global Timer(30) As Variant
Global EF1(30) As Variant
Global P(30) As Variant
Global Integral(30) As Variant
Global D(30) As Variant
Global Offset(30) As Variant
Global Bias(30) As Variant
Global EOutput(30) As Variant
Global Setpoint(30) As Variant
Global Emaxrate(30) As Variant
Global CO2(30) As Variant
Global NO(30) As Variant
Global Tpsr(30) As Variant
Global C2H2(30) As Variant
Global CO(30) As Variant
Global EF2(30) As Variant
Global EA2(30) As Variant
Global Bypass
Global Inputt
Global Inputd
Global InputLast
Global InputDF
Global Feedback
Global Derivative
Global gain
Global ResetRate
Global DFilter
Global OutputTemp
Global Output
Global BufferO2
Global FlagO2
Global Increment

Type PIDLoop
Setpoint As Single
Current As Single
Period As Single
Pgain As Single
Igain As Single
Dgain As Single
Maxrate As Single
Control As Single
Err0 As Single
Err1 As Single
Err2 As Single
Err3 As Single
ErrAccum As Single
Filter As Single
Previous As Single
End Type
Global PIDData As PIDLoop

Public Sub ProcessPID()
PIDData.Pgain = Val(AOMultChForm.txtGain.Text)
PIDData.Igain = Val(AOMultChForm.txtIntegral.Text)
PIDData.Dgain = Val(AOMultChForm.txtDerivative.Text)
If AOMultChForm.OptO2.Value = True Then
PIDData.Setpoint = Val(AOMultChForm.txtSetpoint.Text) * 4
PIDData.Current = Val(AOMultChForm.txtActual.Text) * 4
Else
PIDData.Setpoint = (-1) *
Val(AOMultChForm.txtSetpoint.Text) / 10
PIDData.Current = (-1) * Val(AOMultChForm.txtActual.Text) /
10
End If
PIDData.Period = Val(AOMultChForm.txtPeriods.Text)
PIDData.Maxrate = Val(AOMultChForm.txtMaxRate.Text)
Dim NewErrAccum!, Result!, currentVal!, RTest!, Delta!
currentVal! = (1 - PIDData.Filter) * PIDData.Current +
PIDData.Filter * PIDData.Control
PIDData.Err3 = PIDData.Err2
PIDData.Err2 = PIDData.Err1
PIDData.Err1 = PIDData.Err0
PIDData.Err0 = PIDData.Setpoint - currentVal!
NewErrAccum! = PIDData.ErrAccum + PIDData.Err1
Delta! = (PIDData.Err0 + 3# * (PIDData.Err1 - PIDData.Err2) -
PIDData.Err3) / 6#
Result! = PIDData.Pgain * PIDData.Err0 + (PIDData.Pgain *
PIDData.Igain * PIDData.Period) * NewErrAccum! +
(PIDData.Pgain * PIDData.Dgain / PIDData.Period) * Delta!
RTest! = (Result! - PIDData.Control) / PIDData.Period
If Abs(RTest!) > PIDData.Maxrate Then Result! =
PIDData.Control + PIDData.Maxrate * Sgn(RTest!) *
PIDData.Period
If Result! > 100# Then
Result! = 100#
Else
If Result! < 0 Then
Result! = 0
Else
PIDData.ErrAccum = NewErrAccum!
End If
End If
PIDData.Previous = PIDData.Current
PIDData.Control = Result!
AOMultChForm.txtOutput.Text = PIDData.Control
AOMultChForm.txt.Text = PIDData.ErrAccum
End Sub

Sub Macro1()
 Sheets("Sheet2").Select
   Range("A2:T31").Select
   Selection.Copy
   Sheets("Sheet1").Select
   Range("A1").Select
   Selection.End(xlDown).Select
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveSheet.Paste
   Range("A1").Select
   Sheets("Sheet2").Select
   Application.CutCopyMode = False
End Sub

Public Sub poopfirstcolumn(timer As Variant, O2 As Variant,
A1 As Variant, A1add As Variant, F1 As Variant, P As Variant,
I As Variant, D As Variant, Offset As Variant, Bias As Variant,
Output As Variant, Setpoint As Variant, Maxrate As Variant,
CO2 As Variant, NO As Variant, Tpsr As Variant, C2H2 As
Variant, CO As Variant, F2 As Variant, A2 As Variant)
Dim lLoop As Long
Dim lCount As Long
Dim sh As Worksheet
lCount = 30
Set sh = ThisWorkbook.Sheets("sheet2") 'sheet2
For lLoop = 0 To lCount 'lLoop = 0 - 30
sh.Cells(lLoop + 2, 1).Value = timer(lLoop)
sh.Cells(lLoop + 2, 2).Value = O2(lLoop)
sh.Cells(lLoop + 2, 3).Value = A1(lLoop)
sh.Cells(lLoop + 2, 4).Value = A1add(lLoop)
sh.Cells(lLoop + 2, 5).Value = F1(lLoop)
sh.Cells(lLoop + 2, 6).Value = P(lLoop)
sh.Cells(lLoop + 2, 7).Value = I(lLoop)
sh.Cells(lLoop + 2, 8).Value = D(lLoop)
sh.Cells(lLoop + 2, 9).Value = Offset(lLoop)
sh.Cells(lLoop + 2, 10).Value = Bias(lLoop)
sh.Cells(lLoop + 2, 11).Value = Output(lLoop)
sh.Cells(lLoop + 2, 12).Value = Setpoint(lLoop)
sh.Cells(lLoop + 2, 13).Value = Maxrate(lLoop)
sh.Cells(lLoop + 2, 14).Value = CO2(lLoop)
sh.Cells(lLoop + 2, 15).Value = NO(lLoop)
sh.Cells(lLoop + 2, 16).Value = Tpsr(lLoop)
sh.Cells(lLoop + 2, 17).Value = C2H2(lLoop)
sh.Cells(lLoop + 2, 18).Value = CO(lLoop)
sh.Cells(lLoop + 2, 19).Value = F2(lLoop)
sh.Cells(lLoop + 2, 20).Value = A2(lLoop)
Next lLoop
End Sub
```

REFERENCES

Allen, M. G., C. T. Butler, S. A. Johnson, E. Y. Lo, and F. Russo (1993). "An Imaging Neural Network Combustion Control System for Utility Boiler Applications,"*Combustion and Flame*, vol. 94, 205-214.

Bhat, N. and T. J. McAvoy (1990). "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems," *Computers and Chemical Engineering*, vol. 14 (4/5), 573-583.

Booth, R. C. (1998). "Neural Network-Based Combustion Optimization Reduced NO_x Emissions While Improving Performance," *Proceedings of the American Power Conference*, vol. 60 (2), 667-672.

Te Braake, H. A. B., Eric J. L. Van Can, Jacquelien M. A. Scherpen, and Henk B. Verbruggen (1998). "Control of Nonlinear Chemical Processes Using Neural Models and Feedback Linearization," *Computers and Chemical Engineering*, vol. 22 (7-8), 1113-1127.

Cheng, Y., T. W. Karjala, and D.M. Himmelblau (1995). "Identification of Nonlinear Dynamic Processes with Unknown and Variable Dead Time Using an Internal Recurrent Neural Network," *Industrial and Engineering Chemistry Research*, vol. 34, 1735.

Chovan, T., Thierry Catfolis, and Kürt Meert (1996). "Neural Network Architecture for Process Control Based on the RTRL Algorithm," *AIChE Journal*, vol. 42 (2), 493-502.

Evenson, E. J. and Michael J. Kempe (1998). "Neural Network Modeling and Heuristic Control of Electric Arc Furnace Process Optimization," in *Proceedings of 1998 Electr. Furn. Conference*, 675-687.

Gas Research Institute – GRI (1994). http://www/gri.org.

Glarborg, P., R. J. Kee, J. F. Grcar, and J. A. Miller (1986). "PSR: A Fortran Program for Modeling Well-Stirred Reactors," *Sandia Report*, Sand 86-8209, UC-4.

Gomm, J. B., D. Williams, J. T. Evans, and S. K. Doherty (1997). "Neural Network Applications in Process Modeling and Predictive Control," *Trans. Inst. MC.*, vol. 19 (4), 175-184.

Hecht-Nielsen, Robert (1990). "Neurocomputing," *Addison-Wesley Publishing Company.*

Hernández, E. and Y. Arkun (1992). "Study of the Control-Relevant Properties of Backpropagation Neural Network Models of Nonlinear Dynamical Systems," *Computers and Chemical Engineering*, vol. 16 (4), 227-240.

Hoskins, J. C. and D. M. Himmelblau (1992). "Process Control via Artificial Neural Networks and Reinforcement Learning," *Computers and Chemical Engineering*, vol. 16 (4), 241-251.

Johnson, Mark L. (1998). "Neural Network Technology as a Pollution Prevention Tool in the Electric Utility Industry," in *Proceedings of the 1998 International Join Power Generation Conference*, vol. 7. 559-563.

Kavchak, M. and H. Budman (1999). "Adaptive Neural Network Structures for Non-Linear Process Estimation and Control," *Computers and Chemical Engineering*, vol. 23, 1209-1228.

Kee, R. J., F. M. Rupley, and J. A. Miller (1989). "Chemkin-II: A Fortran Chemical Kinetics Package for the Analysis of Gas Phase Chemical Kinetics," *Sandia Report*, Sand 89-8009B, UC-706.

Mao, Fuhe (1995). "Combustion of Methyl Chloride, Monomethyl Amine, and Their Mixtures in a Two Stage Turbulent Flow Reactor," *Doctoral Dissertation*, New Jersey Institute of Technology.

Mao, F., D. Kretkowski, and Robert B. Barat (1994). "A Two Zone Turbulent Flow Reactor Facility for Studies of Staged Combustion of simulated Hazardous Wastes," *Combustion Science and Technology*, vol. 102, 145-164.

Mao, Fuhe and Robert B. Barat (1996). "Minimization of NO During Staged Combustion of CH_3NH_2," *Combustion and Flame*, vol. 105, 557-568.

Mao, Fuhe and Robert B. Barat (1996). "Experimental and Modeling Studies of Staged Combustion Using a Reactor Engineering Approach," *Chem. Eng. Comm.*, vol. 145, 1-21.

Mao, Zhuoxiong, Jack Demirgian, Alex Mathew, and Richard Hyre (1995). "Use of Fourier Transform Infrared Spectrometry as a Continuous Emission Monitor," *Waste Management*, vol. 15 (8), 567-577.

Martins, Fernando G. and Manuel A.N. Coelho (2000). "Application of Feedforward Artificial Neural Networks to Improve Process Control of PID-based Control Algorithms," *Computers and Chemical Engineering*, vol. 24, 853-858.

Milanič, Srečko, Davorka Šel, Nadja Hvala, Stanko Strmčnik, and Rihard Karba (1997). "Applying Artificial Neural Network Models to Control a Time Variant Chemical Plant," *Computers and Chemical Engineering*, vol. 21, S637-S642.

Miller, C. Andrew and Paul M. Lemieux (1998). "Application of Intelligent Controls in EPA's Combustion Technology Research and Development," *U.S. EPA*, Nation Risk Management Research Laboratory, Research Triangle Park, NC.

Nascimento, C. A. O., Reinaldo Giudici, and Roberto Guardani (2000). "Neural Network Based Approach for Optimization of Industrial Chemical Processes," *Computers and Chemical Engineering*, vol. 24, 2303-2314.

Nikravesh, M., A. E. Farell, and T. G. Stanford (2000). "Control of Nonisothermal CSTR with Time Varying Parameters via Dynamic Neural Network Control (DNNC)," *Chemical Engineering Journal*, vol. 76, 1-16.

Olsson, Jan-Ola, Per Tunestål, and Bengt Johansson (2001). "Closed-Loop Control of an HCCI Engine," *SAE*, vol. 1 (1031), 111-120.

Palancar, María C., José M. Aragón, and José S. Torrecilla (1998). "pH-Control System Based on Artificial Neural Networks," *Industrial and Engineering Chemistry Research*, vol. 37, 2729-2740.

Radl, Brad J. and Willie Roland, Jr. (1995). "A Neural Network Based Optimization System Provides On-line Coal Fired Furnace Air Flow Balancing for Heat Rate Improvement and NO_x Reduction," *Advance Instrumental Control*, vol. 50, 405-414.

Reinschmidt, Kenneth F. and Bo Ling (1994). "Neural Networks for NO_x Control," in *Proceedings of 1994 Annual Meeting of the American Power Conference*, 354-359.

Salem, Tara B. (1996). "Fuel-Rich Combustion of Ethylene and Air in a Two Stage Turbulent Flow Reactor," *Master Thesis*, New Jersey Institute of Technology.

Seborg, Dale E., Thomas F. Edgar, and Duncan A. Mellichamp (1989). "Process Dynamics and Control," *John Wiley & Sons, New York.*

Shu, Huailin and Youguo Pi (2000). "PID Neural Networks for Time-delay Systems," *Computers and Chemical Engineering*, vol. 24, 859-862.

Stephanopoulos, George (1984). "Chemical Process Control An Introduction to Theory and Practice," *PTR Prentice Hall, Englewood Cliffs, New Jersey.*

Syu, Mei-J and Bow-C. Chen (1998). "Back-Propagation Neural Network Adaptive Control of a Continuous Wastewater Treatment Process," *Industrial and Engineering Chemistry Research*, vol. 37, 3625-3630.

Tao, Wenjing and Hans Burkhardt (1994). "Application of Fuzzy Logic and Neural Network to the Control of a Flame Process," *Intelligent Systems Engineering*, Conference Publication, IEE, no. 395, 235-240.

Tendulkar, Shilpa B., Sanjeev S. Tambe, Ishwar Chandra, P. V. Rao, R. V. Naik, and B. D. Kulkarni (1998). "Hydroxylation of Phenol to Dihydroxybenzenes: Development of Artificial Neural-Network-Based Process Identification and Model Predictive Control Strategies for a Pilot Plant Scale Reactor," *Industrial and Engineering Chemistry Research*, vol. 37, 2081-2085.

Zurada, Jaced M. (1992). "Introduction to Artificial Neural Systems," *West Publishing Company*.