Steam Temperature Regulation in Fossil Power Plants using Neural-Adaptive Strategies

by

Mehmet Yunt

B.S., Mechanical Engineering
Boğaziçi University, 1998

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science
at the
Massachusetts Institute of Technology
February 2001

Signature of Author .....
Department of Mechanical Engineering
October 6, 2000

Certified by .....
Anuradha M. Annaswamy
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Ain A. Sonin
Chairman, Department Committee on Graduate Students

Steam Temperature Regulation in Fossil Power Plants using Neural-Adaptive Strategies

by

Mehmet Yunt

Submitted to the Department of Mechanical Engineering
on October 6, 2000, in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Abstract

In this report, neural-adaptive controllers are described for steam temperature regulation in fossil fuel power plants. The algorithms were developed in response to the need for improving the steam regulation properties of the existing controllers. These controllers were observed to exhibit temperature spikes in the event of changing operating conditions. Since the PI controllers have no auto-tuning and operate based on linear principles, these controllers cannot adequately accommodate nonlinear and time-dependent system behavior. Steam temperature behavior in power plants is strongly dependent on load output of the boiler, the cleanliness, and the main steam flow and fuel properties. Moreover, the behavior changes as a result of equipment replacement or malfunctions. The heat transfer from the flue gas to steam makes this system inherently nonlinear. In this report, a new class of adaptive controllers was developed for steam temperature regulation. The controllers have the ability to auto-tune their parameters on-line and respond to nonlinear dynamic characteristics in the system.

The specific problem that we address in this report concerns temperature regulation at the outlet of the reheater or superheater. Two different controllers are developed for this purpose. The first is a linear neural adaptive controller that consists of a pole-placement structure and an update law for adjusting linear control parameters. The controller also accommodates plant saturation effects by including them in the adaptation law and effects of set-point disturbances by including integral action. The second controller is a nonlinear controller that not only includes above linear part but also includes a feedback linearizing substructure together with a TANN (Theta (θ) Adaptive Neural Networks) that allows the adjustment of nonlinear parameters. Both controllers are developed using system-identification model of the reheater dynamics.

The controllers are implemented on a full-scale and detailed simulator at the EPRI Instrumentation & Control Center, Kingston, Tennessee using the Foxboro DCS (Distributed Control System). The performance of the controllers is compared to the present PI implementation. An order of magnitude improvement in the settling time after a major temperature upset was achieved as well as an order of magnitude decrease in the maximum temperature deviation. The performance improvement with the neural controllers was observed both in the context of disturbance rejection and command following.

Thesis Supervisor: Anuradha M. Annaswamy
Title: Principal Research Scientist

ACKNOWLEDGMENTS

First, I am grateful to Professor Anuradha Annaswamy for her guidance as well as the opportunity to work on this challenging project. Her persistence and passion in neural network research have been a great inspiration and motivation to overcome the difficulties I encountered on this project. I would like to express my gratitude to Senior Engineer Cyrus Taft at the EPRI Instrumentation & Control Center in Kingston, Tennessee for his help in obtaining necessary information about power plants and solving all simulator related problems.

Many people have contributed to the completion of this thesis. In particular, I would like to thank my colleagues at the Adaptive Control Laboratory for their fruitful discussions, insightful advice and companionship: Aleksander Kojic, Ashish Krupadanam, Nhut Tan Ho, Chengyu Cao and SungBae Park. I am indebted to Namık Kemal Yılmaz for his friendship and patience.

Finally, this thesis would not have come to a conclusion without the support and encouragement of all my family and my fiancée Tuğce Dereboy. This thesis is dedicated to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Power plants are facing increased pressure to optimize efficiency and emissions while operating with reduced staffs and more variability in fuel. The added constraints on plant operation place demands on the plant control system which conventional control techniques cannot meet. If a utility company is to remain competitive, they will have to meet these requirements. To help achieve these goals, advanced control techniques may be necessary that are capable of adapting to system variables, and learning to integrate this information into the knowledge base.

One of the many problems that can benefit from such advanced control system techniques is the regulation of steam temperature at the inlet of the turbines. The steam temperature at the inlet of turbines has to be tightly controlled and excessive deviations from the nominal operating temperatures must be prevented. This is a crucial regulation problem that has direct effect on the plant efficiency. The thermal efficiency of the power plant depends on the pressure at which the boiler operates as water is heated up to steam. A higher boiler pressure results in higher thermal efficiency of the power plant. However, an increase of pressure in the boiler has a serious drawback. Increased steam pressure results in increased moisture content of the steam. This moisture is detrimental to the turbines due to the corrosion that ensues. In addition, it decreases turbine efficiency. Prevention of excess moisture necessitates higher operating temperatures at the inlets of the turbines. The highest temperature is limited by the metallurgical properties of the metal the turbines and the heat exchanging equipment are made of. The equipment used must not be subject to temperatures higher than a certain limit and moreover, excursions of the temperature from the nominal operating temperatures should be kept at a minimum both in magnitude and time to prevent thermal fatigue [8].

A poor control system forces the plant to operate at a lower steam temperature resulting in a drop of efficiency. If steam temperature can be better controlled, it can be pushed closer to the allowable limit. In central power stations, the highest steam temperature is 1050°F. For example, if the steam temperature set point can be elevated from 1000°F (538°C) to 1040°F (560°C), this will increase the available work by 9.4 kcal /kg of steam. [11]. The plant under study, The TVA Kingston Unit 9 boiler is designed for a rated main steam flow 580000 kg/hr (1280000 lb./hr) at 1053°F (567°C). On the simulator, the power plant operates at a nominal reheater and superheater outlet steam temperature of 1000°F (538°C). If the nominal temperature can be elevated to 1040°F (560°C), burning coal with a heating value of 7200 kcal/kg, the yearly saving of coal with the guaranteed design efficiency of 88.46% is 7500 metric tons of coal. Definitely, the search for a better control system is warranted by the potential economic and environmental gains.

1.1 Steam Temperature Regulation

1.1.1 Plant Overview

The current study is carried on a detailed simulator at the EPRI Instrumentation & Control Center in Kingston Tennessee. The simulator at hand is a detailed replica of TVA Kingston Unit 9. In addition to being used as a platform to test experimental procedures, it is also used as a training station for boiler operators since it embodies the current DCS by the Foxboro Company that is implemented on the actual plant. This enables us to directly implement the controllers developed on the Foxboro control system.

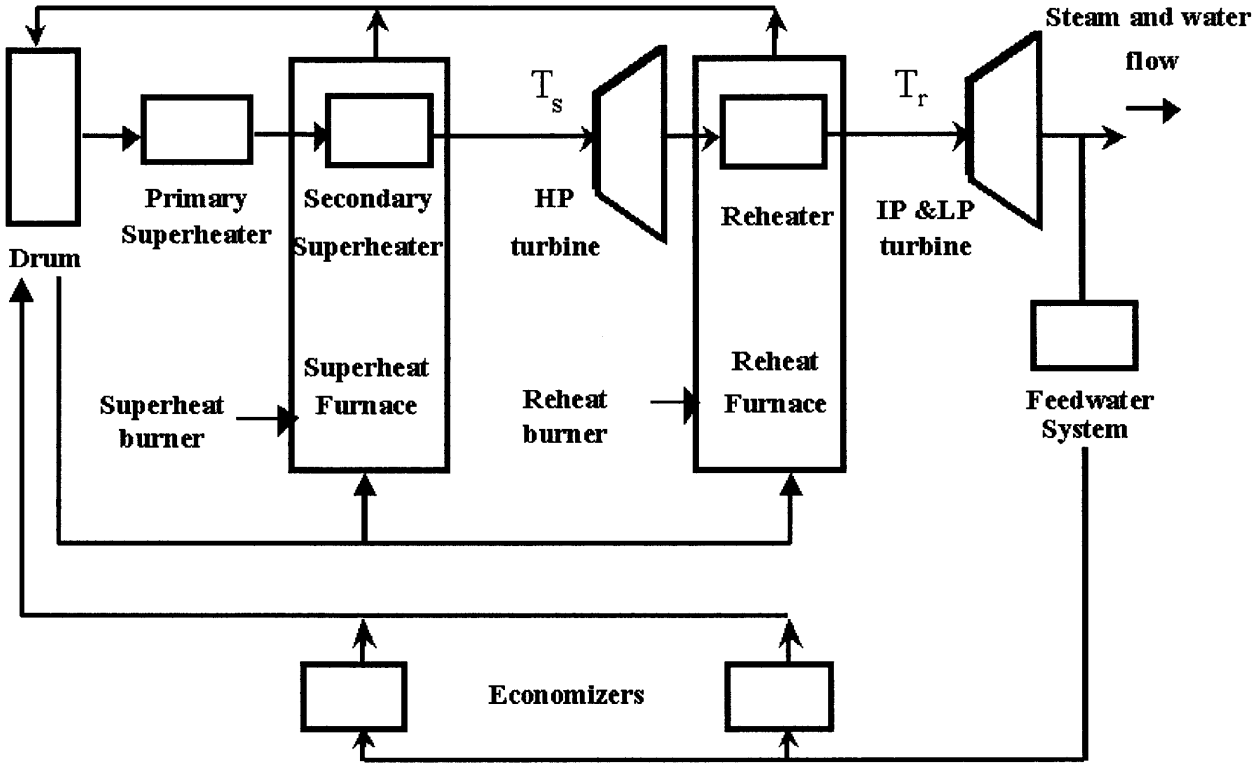

Figure 1-1 Steam and Water Circulation at Unit 9

Before we start discussing controller performances and development, it is useful to supply a plant overview. The Kingston Unit 9 is a twin-furnace forced circulation steam plant with reheat. In Unit 9, there are two furnaces, one for superheating the steam and one for reheating it. The circulation of steam in Unit 9 is depicted in Figure 1-1. Initially, the water is pumped through the economizers by the feedwater system. The economizers, of which there is one for each furnace, are one of the first heat exchangers to facilitate the heat transfer between the water and the hot flue gas, which is the product of the combustion process that takes place in the furnaces. After the economizers, the water enters a reservoir called the drum. The drum is a large storage volume where water and

steam are at thermodynamic equilibrium. It is a volume where hot water and steam from the furnaces and the water from the economizers are collected. In the drum, the water from the economizers mixes with the hot water from the furnace. Due to the density difference, steam and water occupy distinct volumes. Steam collects at the upper parts and water at the lower. It acts as a separation chamber for steam and water. The important point is to prevent water droplets from remaining in the steam to prevent corrosion in the turbines. The steam and water separate, water returns to the furnaces and steam continues towards the primary superheater. The superheat and the reheat furnaces are lined with pipes called the waterwall through which pumps force the water; hence, the name forced circulation. Combustion of coal powder takes place in the furnaces and mainly by radiation, the water in the waterwall undergoes an increase in enthalpy. The next station for steam after the drum, is the primary superheater. The primary superheater is a heat exchanger to facilitate the heat flow from the flue gas to the steam by convection. It is located in the backpass. The backpass is a large conduit where the flue gas produced during combustion flows and finally leaves the unit. The backpass of the reheat and the superheat furnace are the same; consequently, any change in both furnaces affects the steam temperature at the outlet of the primary superheater. After the primary superheater, the secondary superheater is in line. The secondary superheater hangs at the top of the superheat furnace. It is closer to the combustion process than the primary superheater. Radiation plays a dominant role in the heat exchange. The first temperature of interest is the secondary superheater outlet temperature T_s. This temperature has to be tightly controlled.

There are two ways to regulate T_s. One is to spray water of lower enthalpy from the drum at the inlet of the secondary superheater. This technique is called spray attemperation and it is not the desired method. First, it introduces moisture at the inlet of the high-pressure turbine, second it is an energy wasting technique. The second and the preferred method is to use the superheat burners. These are pipes located at the four corners of the furnace through which coal powder is blown into the furnace cavity. The burners point tangential to an imaginary small circle in the center of the furnace. They are designed to create a fireball during plant operation at the center of the furnace. Combustion is most intense at this imaginary circle. By changing the angle of the burners with the horizontal, one can change the vertical location of the fireball. The closer the fireball moves to the secondary superheater, the hotter the steam will become at the outlet of the secondary superheater. The change in the vertical position affects the secondary superheater in two ways. One, the distance involved in radiation heat transfer is less, two, the hot flue gas leaves the furnace in a shorter time. The temperature of the flue gas passing through the secondary superheater is increases since the waterwall surface it traverses is less and the energy lost to the water in the waterwall is less. Therefore, one can control the steam temperature by controlling the tilt angle of the burners. The situation is the same for the reheat furnace. The use of burner tilts is a slower control action than the attemperator sprays since certain amount of time is required to change the temperature of the mass of the superheater or reheater before the effect can be observed on steam temperature. Also the tilts can be rotated from –30 degrees to 30 degrees with respect to the horizontal, so there might be cases where the rotation of the burner tilts is not sufficient to bring the temperature back to desired levels. Then spray attemperation has to be used.

Returning to the circulation of steam, the steam enters the high-pressure turbine through the governor valve. The governor valve is the main element that decides the output of the unit. The governor valve decides the amount of steam flow through the turbines. When higher output is required, the governor valve is opened more. This lets more steam into the turbines, however, the pressure at the governor valve drops as well. The pressure at the drum has to be increased to bring back the pressure at the governor valve to the desired level. To accomplish an increase, the water in the waterwall has to be heated up more. This in turn results in more fuel to be consumed and the firing rate increases at the furnaces.

After the high-pressure turbine, steam passes onto the reheater. Reheating exists only to prevent moisture build up during the passage of steam through the remaining turbine stages [2]. The reheater is identical in location and operation to the secondary superheater. The outlet steam temperature, T_r is the second temperature to be controlled. Its output steam temperature can be similarly controlled as T_s. After the intermediate-pressure and low- pressure turbine, the steam is collected at the condenser, which maintains constant exit conditions for the turbines. The cycle begins anew by the passage of water from the condenser to the feedwater heaters.

The plant is a highly interconnected system. A change or upset in one element in the circulation of the steam has effects on all components. During the operation of the boiler and the turbine, there are many instances when upsets occur in the steam temperatures T_s and T_r. Besides malfunctions and accidents, the main factors affecting the steam temperatures are the following [12]:

1. **Load**: As the load on the boiler increases, that is, as more steam is needed, the amount and temperature of combustion gas increase. In convection type superheaters (reheaters), steam temperature increases with increasing load. In radiant superheaters (reheaters); however, the steam temperature drops. This can be explained as follows: The flame temperature essentially remains constant which means that the amount of energy that can be transferred by radiation per unit time is almost a constant, however, the mass flow rate of steam increases due to load change and hence the power supplied is not enough to sustain the same steam temperature level. In some plants, convection and radiant superheaters are used to obtain a constant temperature over a considerable range of load.

2. **Excess Air**: Excess air that enters the furnace tends to increase the temperature of steam at the superheaters (reheaters). Excess air results in more hot gas passing over the superheaters (reheaters), which in turn increases the steam temperature.

3. **Feedwater Temperature**: As the temperature of the feedwater increases, the temperature rise in the superheater/reheater drops since warmer feedwater results in less fuel being fired, which effects the quantity and temperature of the hot gas passing through the superheaters (reheaters).

4. **Heating Surface Cleanliness**: Removal of ash or slag deposits from heat-absorbing surfaces ahead of the superheater will decrease the superheater temperature because

more heat is absorbed before it reaches the superheater. Ash and slag removal from the superheater's (reheater's) surface will increase the temperature due to the increase in heat absorption.

5. **Use of Saturated Steam**: If saturated steam from the boiler is used for sootblowers, an increased firing rate (amount of fuel being burned) is required to maintain constant main steam temperature. This results in an increase in steam temperature.

6. **Blowdown:** The effect of blowdown using water is similar to the use of saturated steam, but the magnitude is less.

7. **Burner Operation**: The distribution of heat input among burners at different locations or change in the adjustment of a burner angle changes the heat supplied to the superheaters (reheaters).

8. **Fuel**: Changes in fuel quality may result in temperature changes of the steam in the superheaters (reheaters).

1.1.2 Boiler Actuators for Steam Regulation

Sprays and burners are not the only control actuators available for steam regulation. Different power plants have different means to regulate steam temperature; some of the more common methods are listed below [5,12]:

1. **Gas Recirculation**: Some of the hot gas that has passed through the superheater, reheater, economizer and that is on its way to the air heater is diverted into the furnace. If the air is introduced near the burners, the process is called "gas recirculation" and if it is introduced at the furnace outlet near the heaters, then it is called "gas tempering". The amount of recirculated gas is expressed as a percentage of the gas that flows to the air heaters after some of the total gas is diverted. When gas recirculation is employed, the furnace heat absorption drops. Furnace gas outlet temperature may decrease or remain unchanged depending on the gas flow and gas temperature, the temperature of the primary, secondary superheater and reheater temperatures may increase, decrease or remain unchanged

2. **Gas Bypass**: If the convection banks are separated by gas tight baffles into two or more parallel gas passes isolating the portions of the reheater and superheater surfaces, the portion of the gas flow over all or part of the superheater and reheater may be adjusted by regulating dampers. The control is more sluggish than with attemperators.

3. **Temperature Regulation by Burners**: It is possible to regulate steam temperature by selective burner operation. Higher steam temperatures might be obtained at less than full load by operating only the burners that give the highest furnace outlet temperature for gas. A burner may be installed near the furnace exit. Changing the

place where combustion occurs can regulate the steam temperature. Tiltable burners are used for this purpose.

4. **Application of Excess Air**: Decreasing the furnace heat absorption can increase the steam outlet temperature of a convection type superheater. Excess air picks up the heat that would otherwise be absorbed in the furnace and carries it away to the superheaters. Energy lost through the stack increases, but because the temperature of the steam supplied to the turbine increases, the overall efficiency increases.

5. **Differentially Fired Divided Furnace Usage**: In some steam generating units, the superheater receives heat from one section of the furnace only while the other section of the furnace generates only saturated steam. Temperature is regulated by the adjustment of fuel to the different parts of the furnace.

6. **Separately Fired Superheater Usage**: A separate furnace is employed for superheater that is completely separated from the steam-generating unit. This is not economical for power plants.

7. **Attemperation**: Attemperation is the act of lowering the steam temperature. There are two main types of attemperators. One is a heat exchanger where heat is removed from the steam; the second one is a spray that dilutes the steam with water. The second type of attemperator will be discussed further below. The attemperator is usually placed between the primary and the secondary superheater. The spray attemperator provides a quick-acting and sensitive means of control for regulating steam temperature.

1.2 Control Strategies

1.2.1 Background

Steam temperature regulation in power plants is an active field of research. The literature contains the application of many different actuators and control strategies.

Different controllers have been implemented to regulate steam temperature. Except in the context of spray attemperation, most of the controller designs are based on experimental plant models obtained through system identification experiments due to the difficulty of modeling flow and heat transfer phenomena.

Most control strategies developed in the literature concentrate on the use of spray attemperation. As stated earlier, spray attemperation is not the desired way to regulate the steam temperature on Unit 9. The popularity of this approach stems from the fact that a physical model can be easily derived.

A variety of control strategies have been employed to regulate steam temperature using the aforementioned actuators.

The most widely used controller is the venerable proportional and integral controller [5,11,12]. However, the PI controller's performance is not satisfactory as power plants are forced to find more efficient ways to operate due to market competition. Advanced control techniques including optimal control and multivariable control systems have been proposed and implemented [22, 32].

One difficulty encountered is the time-varying, nonlinear nature of reheaters and superheaters. The main reason for variation and nonlinear behavior is the underlying heat transfer process. For example, radiation heat transfer is inversely proportional to the fourth power of the distance between objects [8,17,24]. Empirical convection heat transfer models are strong nonlinear functions of the mass flow rate of the flue gas and steam. [17,24]. The heat transfer process is a strongly dependent on the main steam flow [21,17,34]. This dependence necessitates the on-line tuning of the controller responsible for steam temperature regulation [34]. Steam temperature regulation systems with gain scheduling based on the load of the plant have been in use for quite some time [27].

Moreover, the heat transfer and flow process are distributed parameter systems and their modeling is a trade off between the analytical tractability of the partial differential equations involved and the accuracy of the plant model [13]. Besides, fouling of components, cleaning operations such as sootblowing, equipment replacements and malfunctions change the heat transfer process. In such cases, the necessity of adaptation has also been noted for some time [27].

Several controllers have been developed using spray attemporators including adaptive controllers [34], optimal controllers and generic model controllers [30]. Control strategies have been developed that use gas recirculation as well [27]. Adaptive controllers have been implemented to control temperature by gas circulation [23]. Work on using burners to regulate steam temperature with fixed parameter controllers is also known.

1.2.2 Current Control Systems for Steam Temperature Regulation at Kingston

First, we will look shortly at the control system for the attemporator spray flow technique. The Main controller is a Proportional and Integral (PI) controller. This PI controller supplies the control set point of another PI controller. The main PI controller compares the secondary superheater or reheater outlet steam temperature with the desired temperature. The error is fed to a PI controller to obtain a desired superheater or reheater inlet temperature. This controller is called the trim controller.

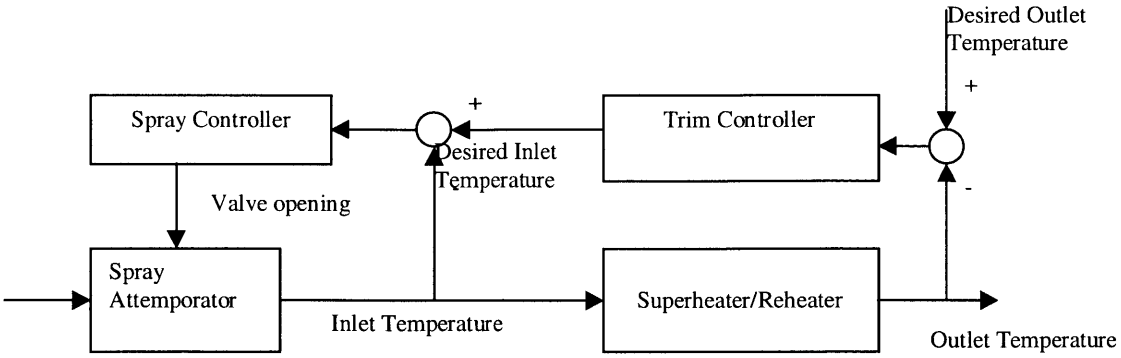

Figure 1-2 Cascaded PI Controller for Attemperation

This temperature is the set point of the second controller. The second controller compares the inlet steam temperature to the desired inlet temperature. This is called the spray controller. The output of the second controller is the valve opening of the spray attemporator valve. This kind of controller architecture is called a cascaded PI controller (Figure 1-2) [11,12]. On Unit 9, there are 2 spray controllers per trim controller of which there is one for the superheater and one for the reheater. Steam flow is divided into two at the attemporators, so two spray controllers are necessary. In this study, we concentrate on improving temperature regulation using the burners. Control algorithms are developed to improve the performance of steam regulation using the burner tilts as actuators. The current controller is a PI controller. (Figure 1-3) A main controller calculates the necessary tilt angle deviation from the nominal tilt angle based on the error between the desired outlet and actual outlet temperature. The burner tilt angle deviation is then used as the set point of four proportional controllers. These controllers are responsible for overseeing that the burners move to the desired position. There are four of these secondary controllers since there are four sets of burners in each furnace. The current control strategy does not meet the demand for tight temperature control. There are many situations during the operation of the unit where the control action is slow and the temperature deviation becomes larger than allowable. The duration for the temperature to come back to desired levels is also unacceptably long. One such situation is during sootblowing (Figure 1-4). As can be seen, the controller is slow in bringing back the temperature to the desired level of 1000^{O} F (538°C).

1.3 Contribution of This Thesis

The literature lacks the use of controllers that directly accommodate plant nonlinearities and that have parameter adaptation capabilities for the nonlinearly occurring parameters. The neural-adaptive controllers proposed for better steam temperature regulation fill this gap. The controllers have the capability to adapt to changing plant conditions. Moreover, in classical adaptive controllers only parameters of the controller that occur linearly can be adapted [2,23] which limits the applicability of the controllers. The neural-adaptive controllers employed, on the other hand, have the capability to adapt any parameter due

to the neural network architecture developed by [1, 36] called the TANN (Theta Adaptive Neural Networks).

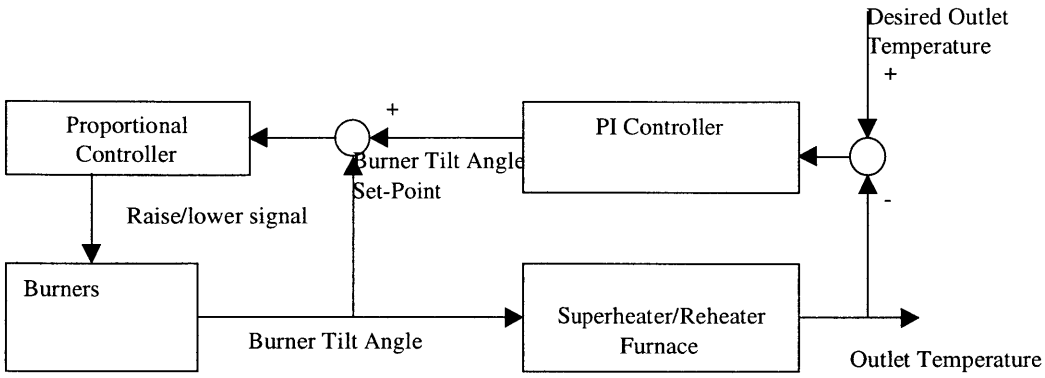

Figure 1-3 Burner Controllers

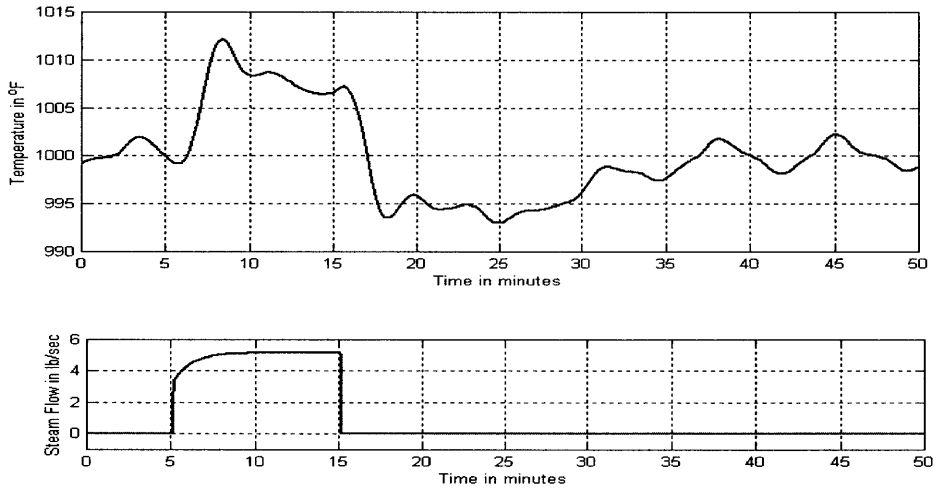

Figure 1-4 Reheat Steam Temperature during Sootblowing

The TANN is a novel way to use neural networks. Instead of using the neural network as the controller of the closed-loop system as is the usual practice [36], a controller is developed using classical control techniques; the TANN is used only to tune the controller parameters, where the parameters occur either linearly or nonlinearly.

The capability of such a control method that can adjust nonlinearly occurring parameters opens many opportunities in control system design. Plant nonlinearities containing such parameters can be directly accommodated by the controller, resulting in a better closed-loop performance.

During this thesis of steam temperature regulation in fossil power plants, the TANN is the control strategy to be used. The actuators are chosen to be the burners whose tilt angles will be modulated by the control strategy.

The development of the controllers will begin with obtaining suitable plant models for design purposes. Two controllers will be developed. One is called the linear neural adaptive controller. Its capabilities include the auto tuning of pole-placement controllers with only linearly occurring controller parameters. The other controller is the nonlinear neural network controller. It is built upon the linear neural adaptive controller expanding its capabilities to nonlinearly parameterized controller structures including feedback linearization.

This thesis is organized as follows. Chapter 2 consists of the methodology applied to find such models. Chapter 3 and Chapter 4 contain the main theoretical development of the controllers. Finally, Chapter 5 contains the results obtained on the EPRI simulator of Unit 9.

2. SYSTEM IDENTIFICATION

The first step in any controller design is the modeling of the open-loop system. Although the EPRI simulator contains a model of TVA Kingston Unit 9, a relatively simpler model still has to be derived. The model on the simulator is very detailed and it does not easily lend itself for control system design. The controllers have to be developed on a simpler model and then tested on the simulator. The simpler model has to be able to accurately describe the temperature dynamics and the effect of the burner tilt angle on steam outlet temperatures. The structure of the model should be able describe the complex behavior of the system accurately. On the other hand, parametric uncertainties due to lack of knowledge of the exact numeric values of the parameters in the model can be accommodated by the adaptation mechanism in the controller.

There are two ways to obtain a simple plant model to be used in controller design. One can assemble a model from constitutive equations such as mass, momentum and energy conservation laws. Many studies are made to model power plants using physical laws [3, 13, 18, 20, 21, 24, 26, and 30]. There are three main drawbacks of this approach for the current steam temperature regulation project. First, the model is very complex. In addition to the reheater and the superheater, mass and energy conservation laws require that the model include the turbine, the drum and the furnaces as well. The burners are located at the bottom of the furnace; the reheater and the superheater are placed at the top. A change in burner tilt angles will directly and indirectly affect the steam outlet temperatures. The direct path involves the modeling of the temperature change of the flue gas; the indirect path involves the change in steam temperature in the waterwall, the drum and the primary superheater. The equations that are required to model the system become many in number and cumbersome to manipulate. Second, any physical model will be specific to the plant at hand. Transferring the control system from one plant to another will require remodeling. Last of all, there are other control loops implemented on Unit 9 such as pressure control, steam flow control and load control loops that directly affect the steam temperature behavior. These control feedback loops have to be implemented on the simpler model as well.

Another approach to modeling is creating a function that estimates the input –output relationship of the model by using experimental results. The difference between the first and the second approach can be best explained by a simple example. Assume the flight time of a thrown rock, as a function of the initial angle of departure from a catapult has to be obtained. Using the previous method, the equations of motion and Newton's Laws can be combined to obtain a formula that will supply the necessary relationship. Using the input-output relationship idea, the stone will be thrown many times at different departure angles from a catapult and a look-up table will be created. Some suitable curve-fitting operation will be carried out to obtain the data pairs for which the experiments will not be done. In this study, the second methodology will be applied and it will be referred to as system identification. We will try to obtain a discrete-time linear model and then identify nonlinearities to improve the performance of the controller. The main advantage of this approach is as follows: The current DCS system by Foxboro facilitates the use of C

programs to carry out system identification experiments. Most other boiler plants also have similar DCS systems installed. Therefore, the system identification procedure can be easily transferred from one plant to another.

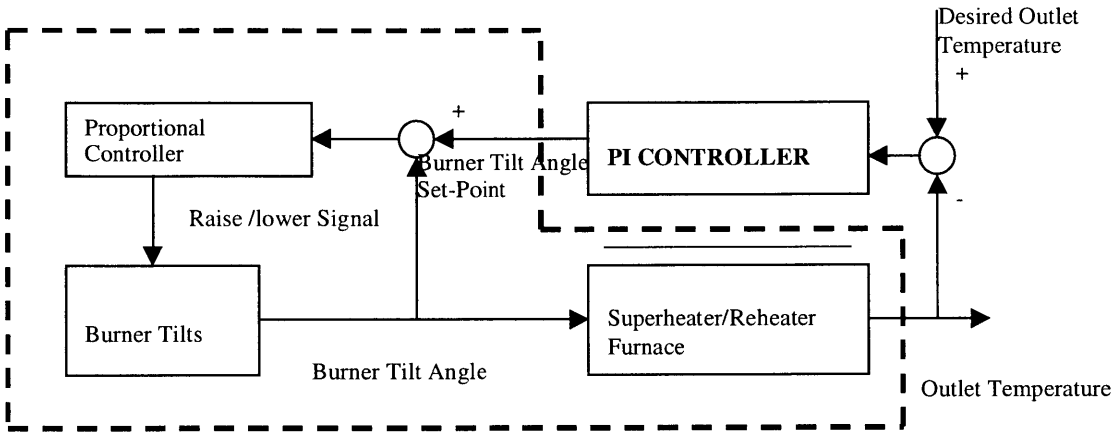

Figure 2-1 Boundary of the System to be Identified (Dashed Lines Mark the Boundary)

The secondary superheater and the reheater are similar heat exchangers. They have similar plant responses to the same input. One technique developed for any one of them can easily be applied to the other one. Therefore, from this point on, only the development of the system identification model for the reheater will be discussed.

2.1 Linear Identification

The first model to be developed is a linear discrete-time model that describes the behavior of reheat steam temperature deviations around a nominal temperature as a function of reheat burner tilt angle deviations around a nominal burner tilt angle. Many plants can be modeled as linear plants in the close proximity of a suitable operating point around which the plant equations are linearized. The resultant models can describe small perturbations around the nominal operating point; however, they lack the ability to describe larger perturbations. Later, the linear model will be augmented by a nonlinear model to alleviate this deficiency.

2.1.1 Definition of the System and Choice of Model Structure

We first describe the specific system that is our focus for system-identification modeling. The system under consideration consists of the reheater and the reheater burner. The input to the system is the reheater burner tilt angle set-point deviation from the nominal to the proportional controller that controls the positioning of the burners (Figure 2-1). The output of the system is the reheater outlet steam temperature deviation from the nominal.

The controllers developed will measure the steam outlet temperature deviation from the nominal and calculate a suitable tilt angle set-point deviation from the nominal.

A part of the following system identification procedure is the choice of model structure. A model of the following form will be used in obtaining the linear model.

$$(2.1)\quad y(t)+a_1 y(t-1)+.....+a_{n_a} y(t-n_a)=b_1 u(t-1)+...+b_{n_b} u(t-n_b)+e(t)$$

The sampling index is *t*, *y(t)* represents the reheater steam outlet temperature deviation from the nominal, *u(t)* is the burner tilt angle set-point deviation from the nominal and *e(t)* is the residual error that cannot be accounted for by the model. It is expected to be a random signal if the model structure is appropriate. The coefficients $\left\{a_1...a_{n_a}, b_1..b_{n_b}\right\}$ are to be estimated. n_a is the order of the model and is never less than n_b due to causality. The $n_a - n_b + 1$ is the delay of the plant.

The linear system identification process aims to calculate the coefficients of the model in equation (2.1). As system identification is a very broad subject, in this thesis, the design of the experiments and the results of those experiments will be discussed. The calculations that led to the numerical values of the coefficients were carried out by using the system identification toolbox of MATLAB. For further computational details, refer to [19].

There are two goals to be met at the end of the system identification procedure. The first goal is to determine the delay and the order of the plant. The controller design requires accurate knowledge of these quantities. The order and the delay tell us the minimum number of parameters needed to describe the system. The delay tells us how long it takes an input to have an effect on the measured output. The delay sets a limit on the speed of the closed-loop performance. The second goal is to determine the plant parameters that will be used in the calculation of the initial controller parameters.

There are some preliminary steps to be taken before the necessary data is collected to come up with the estimated parameters of equation (2.1). Equation (2.1) is a discrete-time model. The plant is a continuous-time process. Therefore, a suitable sampling time has to be determined. Second, note that the proportional controller whose set point is determined by the PI controller is included in the plant. This controller has to be tuned beforehand. Note that if the parameters of the proportional controller change, the adaptation of the controller parameters of the new controllers will compensate for that change. In reality, the proportional controller parameters are no different from plant parameters. Third, a suitable input to the system has to be chosen to excite all modes in order to obtain a model as accurate as possible.

2.1.2 Determination of Sampling Frequency

The sampling time has to be chosen to satisfy the sampling theorem, which states that the frequency with which a signal is sampled has to be at least twice as large as the largest frequency component of the signal. [29]. For an open loop plant to be controlled, a good choice is to select a sampling frequency, which is at least 10 times the bandwidth of the system [19]. Another choice is selecting the sampling frequency as 12-60 times of the natural frequency of the dominant poles of the system.

In this study, an experiment was carried out on the reheater. The plant was brought to steady state at a common operating point. A step change in the burner tilt angles of the reheat furnace was carried out. The open loop step response of the plant was recorded. From the step response, the rise time from 10% to 90% of the steady state value was obtained. The rise time is directly proportional to the open loop dominant natural frequency of the system. We took approximately one thirtieth of this rise time as the sampling interval. This came out to be 5 seconds.

2.1.3 Selection of Experimental Input

The system in consideration has to be excited by an input to obtain output data to be used in the determination of the coefficients of equation (2.1). One important property of the input should be the ability to excite the plant over the frequency range of interest. Returning to the example discussed previously, this is analogous to choosing enough initial departure angles that are spread all over the range of the value the angle may take, so that the data obtained from experimentation is applicable throughout the whole operating range of the catapult.

Common choices for experimental inputs are filtered Gaussian white noise, random binary signals, pseudo-random binary signals and swept sinusoids. For the theoretical development of the properties of the input signals, refer to [19]. In this project, swept sinusoids and random binary signals were used. To obtain random binary signals, first uniform random noise was created. This noise was filtered through a low-pass filter with a cut-off frequency of half the sampling frequency. The sign of the filtered signal was then taken. The binary signal obtain was multiplied with a desired angle set-point deviation. In this experiment, a magnitude of 3 degrees was used. The signal was uploaded to the simulator and used to set the burner tilts. Swept sinusoid signals were obtained from a sinusoid with a frequency that changed continuously over a certain band $\Omega : \omega_1 \le \omega \le \omega_2$ over a certain time $0 \le t \le M$ according to the equation:

$$u(t) = A\cos(\omega_1 t + (\omega_2 - \omega_1)t^2 / (2M)) \tag{2.2}$$

A swept sinusoid signal ranging in frequency from zero to 0.1 Hz was applied as the reheat burner tilt angle set-point deviation and the temperature response was recorded. The system was sampled with a sampling time of 5 seconds, or a sampling frequency of 0.2 Hz, hence the Nyquist frequency was 0.1 Hz. This was taken as the upper bound on

the swept sinusoid frequency range. The amplitude of the swept sinusoid signal was also 3 degrees.

2.1.4 Results of Linear System Identification

The main idea in system identification is to minimize an error between the output of the plant model in equation (2.1) and the experimental data obtained. Experimental data is obtained by measuring the plant output to the specified input discussed in 2.1.3. Data sets similar to that in Figure 2-2 were used to determine the model. Some data sets were used in the calculation of the model parameters whereas some data sets were put aside to test the model obtained. These sets are called test sets. The output of the plant model was obtained by simulating the assumed plant model with the estimated parameters using the same input as used in the experiment. The parameters of the assumed model were optimized to minimize an error that measured the discrepancy of the model output from the real output. If the current model structure did not provide a sufficient good fit, either the model order was increased (the number of parameters of the model was increased.) or a totally new structure was used.

In this study a quadratic error was minimized:

$$J = \frac{1}{N} \sum_{n=1}^{N} \left| y_n^* - \hat{y}_n \right|^2 \tag{2.3}$$

The variable y_n^* represents the experimental output obtained, similarly $\hat{y}_n$ is the model output. N is the number of input output pairs used to calculate the parameters of the model. After a satisfactory set of parameters was obtained, the model was checked using the test sets. The same error criterion as in equation (2.3) was calculated for the parameters using the test sets. If the model performed satisfactorily at this stage as well, then, the model was accepted. If the model did not perform as desired, the whole process was repeated. Sometimes new data was taken, experimental conditions were changed or a different type of input was used. For details of the system identification process, refer to [19].

We checked two aspects of a model before we accepted a model. First, the fit between the experimental and plant data was checked. Second, the error was analyzed to see whether the model had captured the behavior of the plant. Note that in equation (2.1), the behavior of the plant that can not be described by the model is defined as the error, e. If the model captured the characteristics of the plant, the error would be mainly caused by random reasons, it would be a random error. One property of random signals is that their auto-correlation functions are zero except at the origin. This means that the past values of the random signal do not affect the present value of the random signal. It is a way to check that there is no deterministic relationship in the error that should have been captured by the model. The values of auto-correlation function defined in equation (2.4) should lie in a certain band that defines a confidence level.

(2.4) $$R_e(\tau) = \sum_{n=1}^{N} e(t)e(t-\tau)$$

The calculation of the band values is thoroughly explained in [19]. There is a second analysis to be carried out on the error. This requires the calculation of the cross-correlation function of the input and the error as in equation (2.5)

(2.5) $$R_{eu}(\tau) = \sum_{n=1}^{N} e(t)u(t-\tau)$$

The cross-correlation function is to check whether there is a deterministic relationship between the error and the input to the system. Similar to the auto- correlation function, the values of the cross correlation function are expected to lie in a certain band that defines a confidence level. If there are significant values outside the confidence interval, that may mean that there are some dynamics not picked by the model.

In the following paragraphs, we shall present the results of the linear system identification process. The inputs obtained from section 2.1.3 are applied to the system defined in section 2.1.1, which is the reheater burner tilt and reheater combination

As a result of system identification, it was found out that the following second order model structure performed well in modeling the data:

(2.6) $$y(t) + a_1 y(t-1) + a_2 y(t-2) = b_o u(t-1) + b_1 (t-2) + e(t)$$

a_1	-1.86072
a_2	0.8655
b_o	0.0181
b_1	0.0000

Table 2-1 Linear Plant Model Coefficients

The numerical values of the coefficients for equation (2.6) are listed in Table 2.1. When one considers which plant order to choose, one should choose the minimum plant order possible. A plant order higher than second order would outperform the model in equation (2.6) since it has more parameters to adjust. However, the improvement would be miniscule. As one increases the system order, the error calculated in equation (2.3) decreases rapidly at the beginning, but the decrease of the error slows down later. In the initial stages, the error decreases due to improvement in capturing the nature of the plant, in the final stages the little improvement is due to excess parameters estimating the signal *e(t)* . The excess

parameters do not supply any additional data about the behavior of the plant.

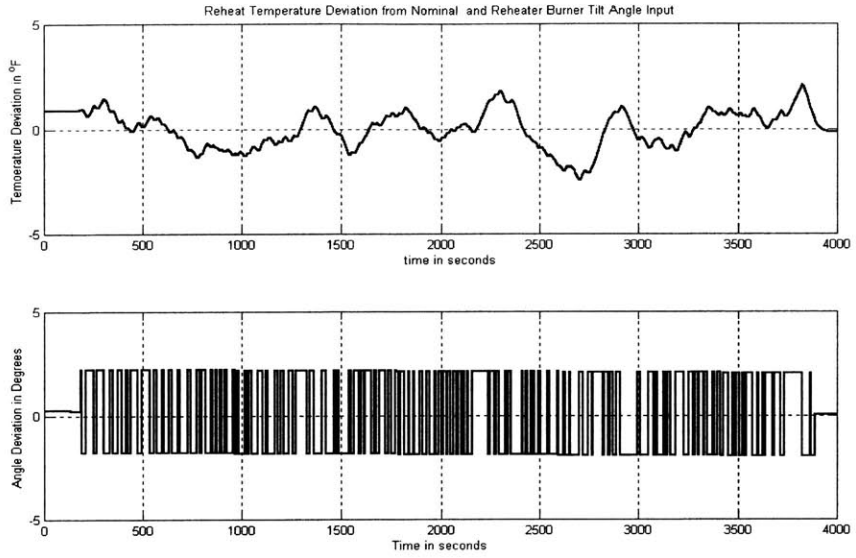

Figure 2-2 Sample Test Set

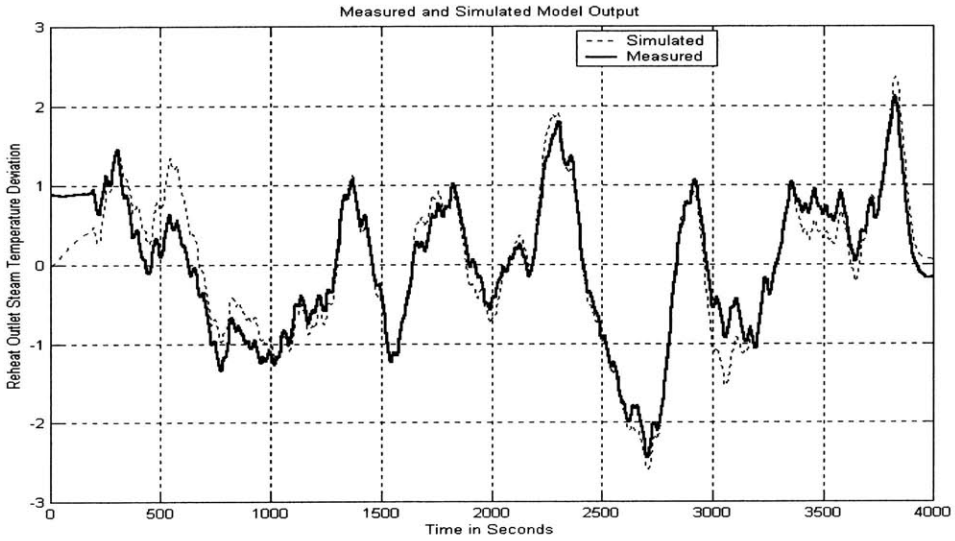

Figure 2-3 Comparison of Model Output and Experimental Data

One cannot expect to achieve zero error since the process is stochastic in nature and random errors will always be present. Finally, a smaller order means a simpler controller. The model obtained is tested on a data set that has not been used in the system identification process. The comparison is depicted in Figure 2-3. It can be seen that the fit is good. In Figure 2-4, it can be seen that the correlation functions mostly within 99% of

the confidence interval (dotted lines). It can be said that the model has picked up the necessary relationship between the input and the output.

2.2 Nonlinear Identification

In section 2.1, a discrete-time model for small perturbations around a nominal reheater burner tilt angle and nominal steam temperature was discussed. In this section, our interest is in analyzing the temperature behavior for larger perturbations. To that end, experiments were carried out to assess plant behavior at different nominal operation

Step response tests were carried out to see how the plant behaved for different nominal burner tilt angles. The burner tilt angle was varied from –27 to 27 degrees in increments of three degrees. A step in the tilt angle was applied of 3 degrees magnitude at each nominal burner tilt angle at 180 MW of load output. The experimentation results are depicted in Figure 2-5 .One can clearly observe the different plant responses at different nominal burner tilt angles. The nonlinear behavior is especially clearly seen at –9 degrees of nominal burner tilt angle. Clearly, the system exhibits different responses at different operating conditions, warranting the use of controllers with adaptation capabilities.

As can be seen in Figure 2-5, the reheat steam outlet temperature behavior is a function of the absolute reheater burner tilt angle. The linear model was augmented with a nonlinear part to account for this dependency. A nonlinearity of the following form was found most suitable to model the dependence of the temperature response on the absolute burner tilt angle.

(2.7)
$$g = \frac{\tan(K(y(t-2) - y(t-1))}{\tan\left(K\left|u(t-1) + u_{nominal}\right|\right) + 0.5}$$

In this expression, u(t) is the reheat angle set-point deviation from the nominal, y(t) is the steam temperature deviation from the nominal steam temperature and $u_{nominal}$ is the reheat burner tilt angle at which the nominal plant is defined.

Consequently, after the addition of the nonlinearity the system model became:

(2.8)
$$y(t) + a_1 y(t-1) + a_2 y(t-2) + \frac{\tan(K(y(t-2) - y(t-1))}{\tan\left(K\left|u(t-1) + u_{nominal}\right|\right) + 0.5} = b_0 u(t-1) + b_1 u(t-2) + e(t)$$

The effect of varying K on the step response of the linear model obtained in section 2.2 is depicted in Figure 2-6.

Since a plant model has been derived, the design of controllers and the corresponding adaptation mechanisms can proceed. The following chapters will concentrate on the design of the controllers for the models obtained. Chapter 3 contains the development of

a controller for the plant described in equation 2.6 and Chapter 4 contains the development of a controller for the plant described in equation 2.8

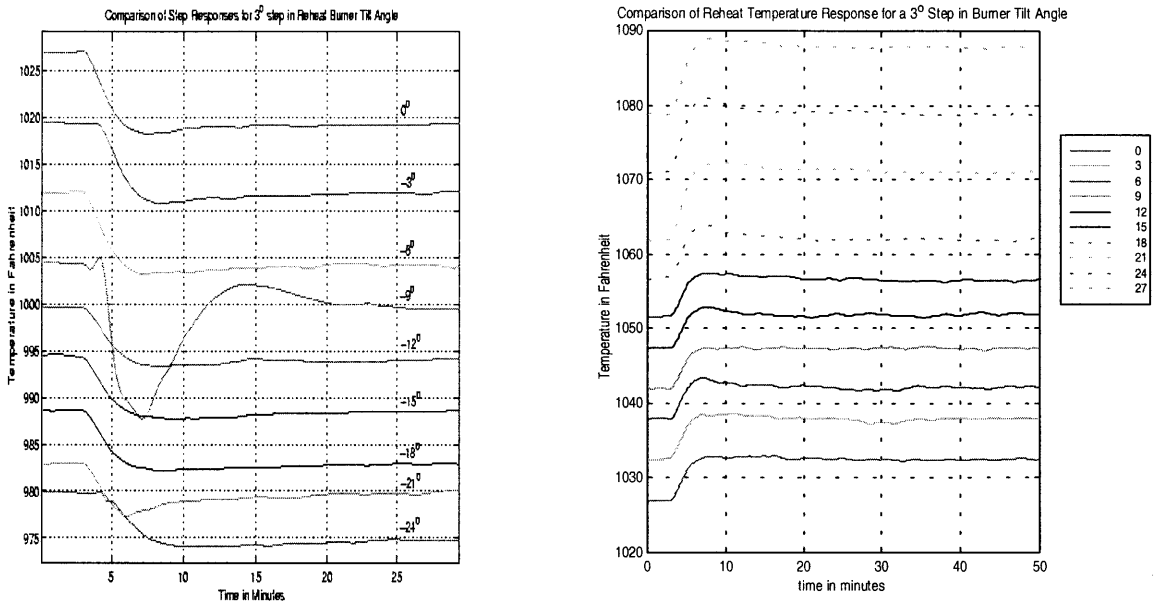

Figure 2-4 Steam Temperature Step Response Behaviors for Different Nominal Burner Tilt Angles

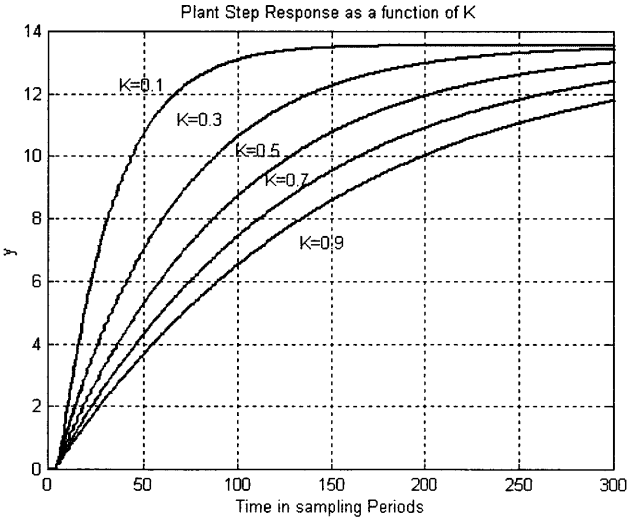

Figure 2-5 Step Responses as a function of K

3. THE LINEAR NEURAL ADAPTIVE CONTROLLER

In this section, we describe a linear neural adaptive controller that is developed for temperature regulation in superheaters and reheaters. The approach used, the resulting controller and adaptation structure are all described in detail.

The starting point for the controller design is the linear plant model obtained in Chapter 2.

$$y(t) + a_1 y(t-1) + a_2 y(t-2) = b_o u(t-1) + b_1 u(t-2) \tag{3.1}$$

The plant model can be rewritten using the forward shift operator q to obtain:

$$(q^2 + a_1 q + a_2) y(t) = (b_o q + b_1) u(t) \tag{3.2}$$

The final form to be used in the development of the controller is:

$$A(q)y(t) = B(q)(u(t) + v(t)) \tag{3.3}$$

The nominal values of the coefficients of the polynomials A(q) and $B(q)$ were determined by the system identification procedure as described in Chapter 2. In equation (3.3), t is the sampling index, u(t) is the input to the system and $y(t)$ is the output. The signal $v(t)$ represents an input disturbance to the system. The reheater burner tilt angle set-point deviation from the nominal is represented by $u(t)$ and the reheater outlet steam temperature deviation from the nominal is represented by $y(t)$.

In order to ensure successful closed-loop performance, the control design must not only include the plant-model, but also the constraints of the actuator. In the context of the reheater, the actuators are the burners. The angle of the burners with the horizontal is changed to control the steam temperature. This angle is called the burner tilt angle. The following properties of the actuator must be considered in the control design:

1. There is actuator saturation.
2. The actuator is rate limited. That is the actuator can not move faster than a certain maximum speed.
3. The actuator moves in discrete increments. That is the actuator can only change its value in discrete amounts, in this case in multiples of 0.25 degrees.

The first and the second properties limit the speed of the controller in regulating the plant. The control action can not be faster than a certain limit. An aggressive controller is also susceptible to noise related problems. This noise consideration also limits the speed of the

controller. The third property of the actuator means that perfect control might not be possible since the actuator cannot move to any arbitrary position. As a result, some error will be present.

3.1 The Controller Structure

Several controller designs are possible for the system in equation (3.3). Some of them are the PID, LQR/LQG and the pole-placement . We use a modified Pole-Placement approach that uses polynomial equations to design a control system. This design method enables us to put the closed-loop poles of the plant at any desired location. It is relatively easy to incorporate different features into the controller such as integral action and the controller equations obtained are easily manipulated to obtain the adaptation rules [2, 28, 29].

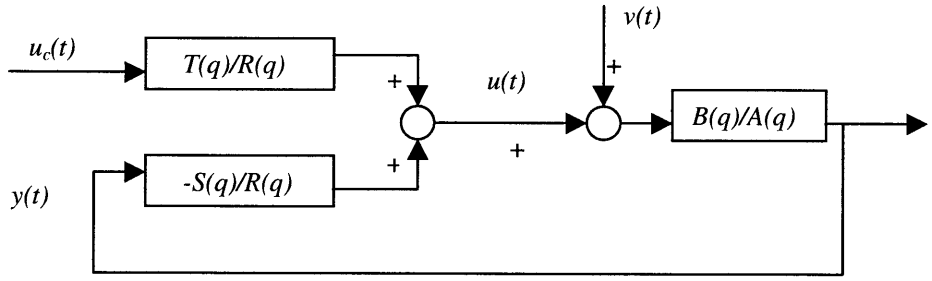

Figure 3-1 Linear Controller Structure

We choose the following controller structure for temperature regulation (see Figure3-1 for a block-diagram):

(3.4) $$R(q)u(t) = T(q)u_c(t) - S(q)y(t)$$

After substitution of the controller equation into the plant equation, the following closed loop transfer function is obtained:

(3.5) $$y(t) = \frac{BT}{AR + BS}u_c(t) + \frac{BR}{AR + BS}v(t)$$

where u_c is the reference input representing the desired reheater outlet temperature deviation from the nominal. $T(q)$ is a polynomial chosen by the designer. $S(q)$ and $R(q)$ are the polynomials of the controller that satisfy the equation

(3.6) $$A(q)R(q) + B(q)S(q) = A_C(q)$$

where $A_c(q)$ is a stable polynomial, that represents the desired poles of the closed-loop system. This equation is called the Diophantine equation and as long as $A(q)$ and $B(q)$ are coprime, a solution $R(q)$ and $S(q)$ exists. There are a few remarks to be made:

1. The polynomials $A(q)$ and $A_c(q)$ are monic.

2. The degree of $R(q)$ is at least one less than the degree of $A(q)$.

3. The degree of $R(q)$ must be equal or larger than the degree of $S(q)$ and $T(q)$ in order to preserve causality.

4. The degree of $A_c(q)$ must be equal to the sum of degrees of $A(q)$ and $R(q)$.

In the following paragraphs, the general structure of the controller in equation (3.4) will be tailored for the specific application of steam temperature regulation. First, the cancellation of the plant zeros and later the inclusion of integral action in to the controller will be described. It is necessary to have integral action in the controller to eliminate constant disturbances.

Obtaining an adaptation mechanism that directly identifies and modifies the controller parameters, which are the coefficients of the polynomials $R(q)$ and $S(q)$, requires the cancellation of the process zeros. The plant numerator B is factored as $B^+ B^-$ where B^+ is the remaining part of the numerator and B^- is the cancelled part. For a design that cancels all the roots of the numerator, B^+ is just a gain. From the closed loop transfer function, it can be deduced that $R(q)$ must have B^- as a factor. $R(q)$ is factored as $R'B^-$. A_c is factored as $B^- A_o A_m$ where A_o is called the observer polynomial and its choice is equivalent to the observer design in the Pole-Placement technique. It has degree one less than the denominator of the open loop plant. A_m is the desired closed loop characteristic equation. Since the polynomial A_o has to be cancelled in the closed loop transfer function, $T(q)$ will have $A_o(q)$ as a factor. Then the Diophantine equation becomes:

$$AR'B^- + B^+ B^- S = B^- A_O A_m \tag{3.7}$$

which simplifies to

$$AR' + B^+ S = A_O A_m \tag{3.8}$$

Note that R is a factor of the numerator in the transfer function between the input disturbance and the plant output. To add integral action, the polynomial R' is factored as $R^+(q-1)$, and as a result the Diophantine equation becomes:

$$AR^+(q-1) + B^+ S = A_O A_m \tag{3.9}$$

Note that R has to be monic since A, A_o and A_m are all monic polynomials. Note also in equation (3.5) that the existence of R in the numerator of the transfer function between the disturbance and the plant output effectively cancels constant disturbances with the aid of the difference operator $(q\text{-}1)$.

The closed loop transfer function becomes

$$y(t) = \frac{B_m A_O}{AR^+(q-1) + B^+ S} u_C(t) = \frac{B_m A_O}{A_O A_m} u_C(t) = \frac{B_m}{A_m} u_C(t) \tag{3.10}$$

where $T(q)$ is factored as $B_m A_o$.

3.2 The Linear Neural Adaptive Controller

The temperature response of the reheater changes at different operating points as discussed in Chapter 2. Different operating conditions, malfunctions and equipment changes necessitate the application of adaptation to keep the same closed-loop performance under varying conditions.

The plant polynomials A and B change as the plant response changes. If A and B were known exactly and/or they never changed, simple calculations would yield the desired controller parameters. However, since A and B are changing, adaptation is necessary. We now proceed to develop the adaptation law for the controller parameters in $R(q)$ and $S(q)$. The starting point of the development is the Diophantine equation.

The final form of the Diophantine equation in equation (3.9) is an identity that can be applied as an operator to a signal. We let $y(t)$ be operated on by this identity:

$$AR^+(q-1)y(t) + B^+ Sy(t) = A_O A_m y(t) \tag{3.11}$$

Substituting $Bu(t)$ for $Ay(t)$, and since $R=B^- R'$, we obtain that

$$B^+ R^+(q-1)u(t) + B^+ Sy(t) = A_O A_m y(t) \tag{3.12}$$

Before proceeding to define the adaptation rule, one more modification may be done. The summations of the coefficients of the polynomials obtained upon simplification of both sides of the equation (3.9) are equal. This leads to the fact that the summation of the coefficients in $B^+S(q)$ is a constant. This can easily be seen if in the above equation one is substituted for q to get the summation of the coefficients on both sides of equation (3.9). This results in the expression $B^+(1)S(1)y(t)=A_o(1)A_m(1)y(t)$. Using this property, the number of parameters to be adapted may be decreased. The polynomial S is redefined to use this property.

(3.13) $$B^{+}S(q) = A_O(1)A_m(1) + (q-1)S'(q)$$

The adaptation will try to keep the equality in the previous equation true. Substituting the right hand side of equation (3.13) into equation (3.12) and manipulating the result, we obtain:

(3.14) $$B^{+}R^{+}(q-1)u(t) + A_0(1)A_m(1)y(t) + (q-1)S'(q)y(t) = A_O A_m y(t)$$

It is desired that the controller change itself as the plant changes, so as to keep the same closed-loop performance. An adaptive controller is required. Many text exist on adaptive controllers [2,14, 28]. An adaptive controller has a control law and adaptation law. The control law calculates the necessary control input whereas the adaptation law tunes the parameters of the control law to obtain optimal performance. Optimality has to be quantified by the definition of an error. This error is a kind of tracking error between the desired closed-loop performance of the system and the actual performance. This error can be defined using the output of the system or it can be defined using the input to the system. We already have the control law by the polynomial design method outlined previously. The following paragraphs of this section will outline the development of the adaptation law.

An error based on the input to the system will be developed. The reason why an input error instead of an output error is chosen is that the input error formulation takes care of actuator saturation automatically. For further discussion on input and output errors, refer to [14].

Let $q^m + R^*$ be equal to $R^+(q\text{-}1)$, where m is the degree of $R'(q\text{-}1)$. Defining S^* to be S'/B^+ and G to be $1/B^+$, and solving for $u(t)$ we obtain :

(3.15)
$$u(t) = -R^{*}u(t-m) - (q-1)S^{*}(q)y(t-m) + G(A_O A_m y(t-m) - A_O(1)A_m(1)y(t-m))$$

Equation (3.15) can be considered as a different way of writing equation (3.3). Suppose, as a general case the plant had a delay of d samples. Then equation (3.3) would become:

(3.16) $$A(q)y(t) = B(q)(u(t-d) + v(t-d))$$

Then similar to equation (3.3), the plant equation in (3.16) could be rewritten as:

(3.17)
$$u(t-d) = -R^{*}u(t-m-d) - (q-1)S^{*}(q)y(t-m) + G(A_O A_m y(t-m) - A_O(1)A_m(1)y(t-m))$$

If all plant parameters were known exactly, *u(t-d)* obtained from equation (3.17) would be the same as the input to the system *u(t-d)* obtained by the control law. However, this is not the case and between these two quantities, there is a discrepancy. To distinguish the output of equation (3.17) from the control input to the plant *u(t-d)*, we shall call it $\hat{u}(t-d)$. As a result, equation (3.17) becomes:

(3.18)

$$\hat{u}(t-d) = -R^*u(t-m-d) - (q-1)S^*(q)y(t-m) + G(A_O A_m y(t-m) - A_O(1)A_m(1)y(t-m))$$

The error to be minimized becomes:

(3.19)
$$e = u(t) - \hat{u}(t)$$

The control law is obtained from equation (3.2). The desired closed-loop behavior is chosen as:

(3.20)
$$A_m(q) = A_m(1)u_c(t-d)$$

Note that *T(q)* in equation (3.4) becomes $T(q) = A_o(q)A_m(1)$ and B_m is $A_m(1)$. This choice of *T(q)* will force the output of the system to be equal to the reference signal at steady state.

The control input to the plant is obtained from the following equation after manipulating the general controller structure in equation (3.4):

(3.21)

$$u(t) = -R^*u(t-m) - (q-1)S^*(q)y(t-m+d) + G(A_O A_m(1)u_c(t-m) - A_O(1)A_m(1)y(t-m+d))$$

One last thing to consider about the control law is the saturation of the actuators. Let the plant input be constrained between u_{max} and u_{min}. If *u(t)* is larger than u_{max}, it will be set to u_{max}, else if it is less than u_{min}, it will be set to u_{min}.

We developed the control law and the error to be minimized. We shall proceed to define the adaptation law. The adaptation law carries out an on-line minimization of the error at every sampling instant and changes the plant parameters.

Let the estimated controller parameter vector at time *t* be equal to
$\hat{\theta}^T(t) = [r_{m-1}, r_{m-2}....r_1, s_n, s_{n-1}..s_1, G]$, where r_i are the coefficients of the R^* polynomial, s_i are the coefficients of the S^* polynomial and *G* is a gain. Define the vector of measured signals at time *t* as

$$\Phi^T(t) = [u(t-1), u(t-2)..u(t-m), (q-1)y(t-m+n), (q-1)y(t-m+n-1)...(q-1)y(t-m)]$$

.

Then the adaptation law is becomes:

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \frac{\gamma \Phi(t)}{\alpha + \Phi(t)^T \Phi(t)} e \tag{3.22}$$

The adaptation gain γ has a value between 0 and 2 and α is a positive number. The stability proof can be found in [2, 14].

3.3 Application to Steam Regulation in Reheaters

The linear neural adaptive controller will be applied to the second order model obtained from the system identification. As a result, the following equations are obtained:

The plant equation from Chapter 2 is:

$$\begin{aligned} & y(t+2) + a_1 y(t+1) + a_2 y(t) = b_0 u(t+1) + b_1 u(t) \\ & (q^2 + a_1 q + a_2) y(t) = (b_o q + b_1) u(t) \end{aligned} \tag{3.23}$$

Desired closed loop equation:

$$\begin{aligned} & y(t+2) + a_{m1} y(t+1) + a_{m2} y(t) = (1 + a_{m1} + a_{m2}) u_c(t+1) \\ & (q^2 + a_{m1} q + a_{m2}) y(t) = (1 + a_{m1} + a_{m2}) q u_c(t) \end{aligned} \tag{3.24}$$

The observer polynomial:

$$A_O(q) = q + a_o \tag{3.25}$$

The Diophantine equation in (3.9) becomes:

(3.26)

$$(q^2 + a_1 q + a_2)(q-1)(q + \frac{b_1}{b_0}) + b_0 (q + \frac{b_1}{b_0})(s_0 q^2 + s_1 q + s_2) = (q + a_0)(q + \frac{b_1}{b_0})(q^2 + a_{m1} q + a_{m2})$$

The redefiniton of the of the *S (q)* polynomial is:

$$\begin{aligned} & b_0 (s_0 + s_1 + s_2) = (1 + a_0)(1 + a_{1m} + a_{2m}) \\ & b_0 S(q) = (1 + a_0)(1 + a_{1m} + a_{2m}) + b_0 (s_0 q + (s_1 + s_0)) \end{aligned} \tag{3.27}$$

The control input is calculated as follows:

(3.28)

$$u(t) = G(A_O(q)A_m(1)u_C(t-1) - A_o(1)A_m(1)y(t-2)) - u(t-1) + r(u(t-2) - u(t-1)) + s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

The plant input is constrained between u_{max} and u_{min}. If $u(t)$ is larger than u_{max}, it will be set to u_{max}, else if it is less than u_{min}, it will be set to u_{min}.

Finally, $\hat{u}(t)$ is calculated as follows:

(3.29)

$$\hat{u}(t) = G(A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2)) - u(t-1) + r(u(t-2) - u(t-1)) + s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

The θ and Φ vectors come out to be:

(3.30)

$$\hat{\theta}(t) = [r, s_0, s_1, G]$$

$$\Phi(t) = [u(t-2) - u(t-1), y(t-2) - y(t), y(t-2) - y(t-1), (A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2))]$$

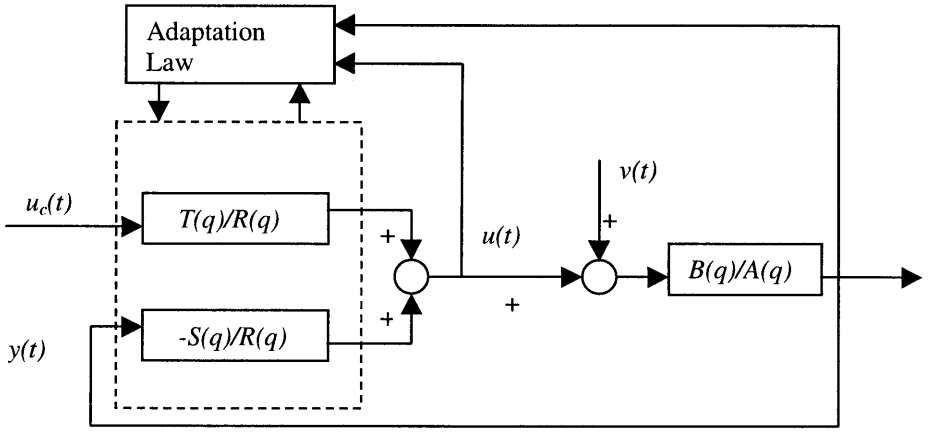

Figure 3-2 Linear Neural Adaptive Controller

Equation (3.28), (3.29), (3.19) and (3.22) constitute the linear neural adaptive controller (Figure 3-2) applied to the EPRI simulator. The performance of the linear neural adaptive controller will be discussed in Chapter 5.

4. THE NONLINEAR NEURAL NETWORK CONTROLLER

A neural-adaptive controller will be developed for the nonlinear system equation obtained in Chapter 2. The nonlinear plant is of the form:

(4.1)

$$y(t)+a_1 y(t-1)+a_2 y(t-2)+\frac{\tan(K(y(t-2)-y(t-1))}{\tan\left(K\left|u(t-1)+u_{nominal}\right|\right)+0.5}=b_o u(t-1)+b_1 u(t-2)$$

Unlike the previous controller, the nonlinear neural network controller has the power to control plants that have nonlinearities and it has a novel method of adapting parameters that occur nonlinearly in the system model. The nonlinear plant model discussed in Chapter 2 requires a controller that accommodates the nonlinearity in the plant. Furthermore, the parameters in the nonlinearity require the application of a different adaptation law than that obtained in Chapter 3.

Before proceeding with the development of the controller, the terms nonlinearly occurring parameters and linearly occurring parameters will be defined. Consider the following system:

(4.2) $$y(t)= A\sin(\omega_1 t)+B\cos(\omega_2 t)$$

The parameters A and B are said to occur linearly since the plant output $y(t)$ can be written as a vector product of $\theta=[A,B]^T$ and $\Phi=[\cos(\omega_1 t),\cos(\omega_2 t)]^T$ whereas the plant output can not be written as a vector dot product of the parameters ω_1,ω_2. Therefore, ω_1,ω_2 are parameters that occur nonlinearly.

The nonlinearity discussed in Chapter 2 is of the form:

(4.3) $$g(u,y)=G\frac{\tan(Ky)}{\tan\left(K|u|\right)+0.5}$$

The parameter K in equation (4.3) occurs nonlinearly whereas the parameter G occurs linearly. The behavior of $g(u,y)$ for different parameter values of K and G is graphed in Figure 4-1 for fixed y and varying u. Note that, the function values increase or decrease proportionally to the change in the value of G at every u value whereas such a relationship with respect to K does not exist.

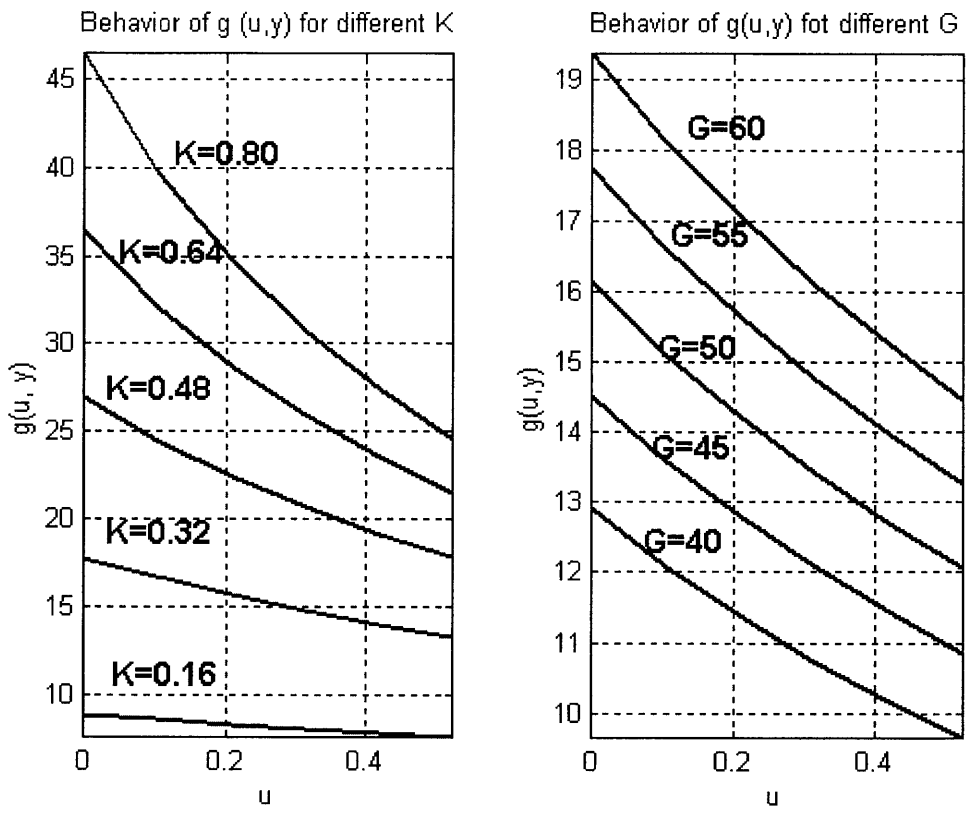

Figure 4-1 The Behavior of *g(u,y)* with respect to *K* and *G*

Adaptation of the linearly occurring parameters can be accomplished using equation (3.20) in Chapter 3 or in general with any gradient-descent-based algorithm [2,14,28]. Adaptation of nonlinearly occurring parameters is a current topic of research and one answer to this problem is the TANN (Theta (θ) Adaptive Neural Networks). The controller described in this chapter includes the TANN as an adaptation mechanism to tune nonlinearly occurring parameters in systems.

This chapter begins with a short overview of neural networks and why they are useful in control systems. We continue by explaining the TANN and a subclass of TANN for plants that have both linearly occurring and nonlinearly occurring parameters. This subclass enables a simplification in the implementation of the TANN by separating the adaptation of linearly occurring parameters from the nonlinearly occurring ones. The rest of the chapter is devoted to the application of this subclass to the steam regulation problem.

4.1 Overview of Artificial Neural Networks

Artificial Neural Networks (ANN) refer to computational units composed of simple and interconnected linear or nonlinear functions.

In this study, the ability of ANN to approximate any given nonlinear function to desired degree of accuracy using finitely parameterized units is utilized to synthesize a controller structure for the temperature control problem. The ANN used, possess the "universal approximator " property. The following definition describes such a class of neural networks N. Let Γ be the set of all continuous mappings of $\Re^n$ to $\Re^m$, and let N subset of Γ.

Definition 1: Given $\varepsilon > 0$ and a compact set $K \subset \Re^n$, for every $f \in \Gamma$, there exist an $N \in$ N such that $|N(x) - f(x)| < \varepsilon \quad \forall x \in K$.

This definition implies that for any continuous function, an ANN, N∈N, can be found to approximate the function to the desired accuracy in a compact set.

The ANN architectures used in this study possessing the universal approximator property are the RBF (Radial Basis Function) and Multilayer Feedforward Neural Networks with Sigmoidal Activation Functions. The approximation error ε is a function of the number of computational units in these networks [36].

In essence, the approximation of a function from given data points is an ill-posed inverse problem. A set of data points may accept many functions that approximate the given data well, but the important point is to achieve good approximation for data points not used in the formation of the approximating function. The ability to perform satisfactorily on data points not encountered before is called *generalization.* The desired performance of the neural network is to have good generalization. This, in turn limits the number of computational units used since a large number of computational units will result in *memorization* of the data presented (poor generalization) rather than what is termed as "learning" analogous to the behavior of biological entities.

The process with which an ANN is made to approximate the underlying function of a given data set is an optimization process in which the parameters of the computational units are calculated as to minimize a certain function of the error between the ANN output and the data set. This is called training of the ANN analogous to biological processes. Further details corresponding to this process will be explained in subsequent sections.

4.2 Theta (θ) Adaptive Neural Networks (TANN)

The TANN has been developed [1,36] to offer nonlinear parameter estimation capabilities both for the parameter estimation and the control problems.

A large number of physical processes can be modeled using mass, momentum and energy conservation. These system models are usually nonlinearly parameterized. In the case of parameter uncertainty, parameter estimation has to be done. Most current algorithms developed for parameter estimation assume a linear formulation. The algorithms developed for linear parameter estimation do not perform satisfactorily in such cases where the system is nonlinearly parameterized.

The TANN is developed to fill this gap in nonlinear parameter estimation cases. It uses a neural network in a novel method to estimate nonlinearly occurring parameters. The neural network supplies a stable adaptation rule to be used on-line. The neural network calculates the necessary changes to be made to the parameters in a stable manner by using the measurable outputs and inputs of the system. Prior to implementation, the neural network has to be trained off-line. The neural network must be taught the functional relationship between the measurable signals and the parameters to be estimated. During the training phase, different parameter values are put into the system to measure the change of observable signals. This experimental procedure requires a controlled environment similar to the actual plant. An accurate model, an extensive simulation model such as the Kingston Simulator at the EPRI I&C Center in Kingston, Tennessee or an experimental setup easily satisfy this requirement.

We are now ready to discuss the usage of TANN for control systems.

4.3 The TANN Algorithm for Control of Systems

The TANN allows great flexibility for the design of the controller. The designer may choose from a wide selection of control strategies and let the TANN do the necessary parameter adaptation (see Figure 4-2). Some notable benefits of this flexibility and off-line training are the improved speed of the controller and the opportunity to assess the performance of the TANN before implementation. Other ANN strategies use an ANN as the controller. The parameters of this neural network are adapted online. This strategy requires the adaptation of many parameters without a guarantee of stability. Moreover, adaptation of many parameters takes more time in contrast to the TANN, which adapts the parameters of a known controller

structure with fewer parameters. Second, the resultant neural network of the off-line training can be tested for satisfactory performance without implementing on the

The implementation of a d-step-ahead controller with TANN is outlined in the following paragraphs.

The discrete-time system under consideration is of the form

$$y(t+d) = f_r(\omega(t), u(t), \theta) \tag{4.4}$$

where $u(t)$ is the input to the system, $y(t+d)$ is the output of the system at time t and $t+d$ respectively. θ is the unknown parameter vector that occurs nonlinearly in the system equation and ω_t is defined as

(4.5) $\omega^T(t) = [y(t),...y(t-n+1),u(t-1),..,u(t-m-d-1)]$, $n \geq 1, m \geq 0, m+d = n$.

As stated before, the choice of controller lies with the designer. The choice of controller will determine the definition of error similar to the error defined in Chapter 3. Here we shall develop the theory for a d-step-ahead controller for regulation. For any other controller, the development is similar.

If the goal is to regulate the system around zero, using a d-step-ahead technique [14], the control input is calculated as:

(4.6) $$u(t) = K(\omega(t),0,\hat{\theta})$$

where $\hat{\theta}$ is the parameter estimate at time t. If θ is known, then the system can be brought to zero at the desired future time. However, the true parameter is not known. Therefore, the plant can not be brought to the desired temperature at the desired time. There must be a tuning mechanism that will adjust the parameters to ensure the desired performance. The TANN is utilized to provide the necessary parameter estimate $\hat{\theta}$ and tune the controller.

The parameter $\hat{\theta}$ is updated using an ANN N as

(4.7) $$\Delta\hat{\theta}(t) = \hat{\theta}(t) - \hat{\theta}(t-1) = N(y(t),\omega(t-d),u(t-d),\hat{\theta}(t-1))$$

A properly trained network enables the output of the plant to converge to a close neighborhood of the desired value [1 , 36].

The TANN is trained by first creating a training set as follows. Let $y_t \in Y_1$, $u_t \in U_1$ and $\hat{\theta}$, $\theta \in \Theta$. First $\omega \in Y_1 \times U_1$ and $\hat{\theta}$, $\hat{\theta} \in \Theta$ are sampled. Their values are $\omega_1, \hat{\theta}_1, \theta_1$. If d is larger than one then $\hat{\theta}$ is sampled as $\hat{\theta}_1^d$ again such that $\left|\theta_1 - \hat{\theta}_1^d\right| \leq \left|\theta_1 - \hat{\theta}_1\right|$. Once the data is sampled the following can be calculated; $u_1 = K(\omega_1,0,\hat{\theta}_1)$ and $y_1 = f_r(\omega_1,u_1,\theta_1)$.

If $y_1 \in Y_1$, this pattern is ignored.

The following quantities are calculated:

$$C(\phi) = \frac{\partial K}{\partial \theta}(\omega_1, y_1, \theta_0)$$

$$\hat{u}_1 = K(\omega_1, y_1, \hat{\theta}_1^d)$$

(4.8)
$$\Delta V_{d_1} = -a_1 \frac{2 + |C(\phi)|^2}{(1 + |C(\phi)|^2)^2}(u_1 - \hat{u}_1)^2$$

$$L_{d_1} = a_2 \frac{|C(\phi)|^2}{(1 + |C(\phi)|^2)^2}(u_1 - \hat{u}_1)^2$$

$$0 < a_1 < a_2 \leq 1$$

A data set is formed as $(y_1, u_1, \omega_1, \theta_1, \theta_1^d, \Delta V_{d1}, L_1)$.

The TANN will provide $\Delta\hat{\theta}$ for each data set. The following quantities are then calculated:0

(4.9)
$$\Delta V_1 = 2\Delta\hat{\theta}_1^T(\hat{\theta}_1^d - \theta_1) + \Delta\hat{\theta}_1^T\Delta\hat{\theta}_1$$

$$L_1 = \Delta\hat{\theta}_1^T\Delta\hat{\theta}_1$$

The ANN is trained so as to have $L_s < L_{d_s}$ and $\Delta V_s < \Delta V_{d_s}$ for all s.

The logic behind having $L_s < L_{d_s}$ and $\Delta V_s < \Delta V_{d_s}$ is as follows: ΔV_s governs how fast the parameter will converge whereas L_s prevents the parameter update to get larger as the system approaches the origin and the control input u becomes smaller and smaller. ANN trained as such will lead to closed loop stability [1,36].

The training of the TANN can be cast as an optimization problem with the following cost function.

(4.10)
$$J = \frac{1}{2}\sum_{s=1}^{M}\{(\max(0, \Delta V_s - \Delta V_{d_s}))^2 + \frac{1}{b^2}(\max(0, L_s - L_{d_s}))^2\}$$

M is the number of training sets, b is a coefficient to adjust the relative importance of the second term.

4.4 The TANN Algorithm for Control of Plants with Linear Substructure

Equation (4.4) is a very general description of a system. It shows the power and wide applicability of the TANN controller. However, the implementation of the TANN requires the training of a neural network. This might turn out to be time consuming. If the control law obtained for a particular plant includes a linearly parameterized substructure, the TANN can be used to estimate only the nonlinearly occurring parameters and a gradient-descent based algorithm can be applied to the rest. This would reduce the size of the neural network and the time it takes to train it.

Consider the control law of the following form:

$$u(t) = \theta_L{}^T \Phi_L + g(\theta_N, \Phi_N) \tag{4.11}$$

It is desired to estimate the θ_L with a gradient-based algorithm, and θ_N by a neural network. Φ_L and Φ_N are signals that are measurable include the present and past values of the system output and the plant input. In this project, these are the past and present values of the deviations from the nominal reheater tilt angle set point and deviations of the reheater outlet temperature from the nominal temperature.

We choose the following adaptation law for the estimated parameters:

$$\hat{\theta}_L(t+1) = \hat{\theta}_L(t) + \upsilon k \Phi_L(t) \tag{4.12}$$

$$\hat{\theta}_N(t+1) = \hat{\theta}_N(t) + \upsilon k N(\Phi_L(t), \hat{\theta}_N(t)) \tag{4.13}$$

The neural network N will take as its argument Φ_L and Φ_N. k and υ are defined as follows:

$$k = \frac{\alpha e}{1 + C_N^T(t) C_N(t) + \Phi_L(t)^T \Phi_L(t)} \quad \alpha \in (0.1) \tag{4.14}$$

$$\upsilon = \begin{cases} 1 \text{ if } \Delta V_d(t) \geq \varepsilon \\ 0 \text{ otherwise} \end{cases} \tag{4.15}$$

$$C_N = \left. \frac{\partial g}{\partial \theta_N} \right|_{\theta_{No}} \tag{4.16}$$

$$e = u(t) - \hat{u}(t) \tag{4.17}$$

The definition of e changes from controller design to controller design. As mentioned in the previous section. It is an error signal that measures the discrepancy between the actual closed loop performance and the desired closed loop performance. The e used in steam temperature regulation will be obtained later on in the chapter.

The training of the neural network requires the creation of a training set. The training set consists of data sets of different combinations of $\hat{\theta}_N, \theta_N, \Phi_N$ over the respective ranges of the parameters and signals. A sample training set S_i is $\{\hat{\theta}_{N_i}, \theta_{N_i}, \Phi_{N_i}\}$. A fixed set of parameter values θ_{N_o} is determined. Two quantities for each training set is calculated:

$$L_i = \left| C_{N_i} \right|^2 \tag{4.18}$$

$$D_i = g(\Phi_{Ni}, \hat{\theta}_{N_i}) - g(\Phi_{Ni}, \theta_{N_i}) \tag{4.19}$$

The neural network is trained so as to obtain the following conditions for each data set in the training set:

$$\left| N_i(W) \right| \le L_i \tag{4.20}$$

$$N_i(W)^T \left(\hat{\theta}_{N_i} - \theta_{N_i} \right) \le D_i \tag{4.21}$$

W denotes the set of parameters of the neural network to be optimized. Note that, unlike the general TANN, the control parameters to be estimated by the neural network involves only the nonlinearly occurring parameters. Note also that the error e is not required in the training.

A cost function of the following form is used in training the neural network over the total M number of data sets:

(4.22)

$$J = \frac{1}{2} \sum_{i=1}^{M} \left\{ (\max\{0, N_i(W)^T \left(\hat{\theta}_{N_i} - \theta_{N_i} \right) - D_i\})^2 + \frac{1}{b^2} (\{\max\{0, \left| N_i(W) \right|^2 - L_i)^2\}) \right\}$$

The parameter b determines the relative importance of each target. W is the parameters or weights of the neural network.

A few words about the training procedure are in order. The training of a neural network is an optimization problem and it involves the minimization of a nonlinear function. There are many optimization techniques available. Gradient-descent methods are slow, Gauss-Newton Methods (especially a modified form called the Levenberg-Marquardt method) are faster but require a lot of. One could use the Down Hill Simplex method, which uses function evaluations only. In this study, it is observed that Quasi-Newton methods are both fast enough and are not as memory intensive as Gauss-Newton methods memory [7,36,10,15].

The Quasi-Newton method requires knowledge of the first derivative of the function J with respect to any neural network parameter w_j. One could calculate the derivative analytically or numerically. For large number of neural network parameters, numerical computation of the derivative might become very time consuming. The analytical derivative of J is obtained over the subset P_1 of the training set where inequality (4.18) does not hold and over the subset P_2 where inequality (4.19) does not hold. The analytical derivative is:

$$\frac{\partial J}{\partial w_j} = \frac{1}{b^2} \sum_{i \in P_2} \left\{ 2N^T \frac{\partial N_i}{\partial w_j} \right\} + \sum_{i \in P_1} \left\{ \left(\hat{\theta}_{N_i} - \theta_{N_i} \right)^T \frac{\partial N_i}{\partial w_j} \right\} \tag{4.23}$$

An iteration of the Quasi-Newton Method is outlined below, for a more detailed treatment, see [10]. The algorithm is initialized with an initial positive definite guess of the Hessian of J called B, and an initial set of weights W_c. A search direction d is calculated using the gradient of J at point W_c.

$$d = -B^{-1} \frac{\partial J}{\partial w}\bigg|_{W_C} \tag{4.24}$$

A line search is used to find a new point W_n.

$$\begin{aligned} & W_n = W_C + \lambda d, \lambda > 0 \text{ such that} \\ & J(W_n) \le J(W_c) + \alpha d^T \frac{\partial J}{\partial w}\bigg|_{W_c} \quad \alpha \in (0,0.5) \end{aligned} \tag{4.25}$$

The gradient at the new point is checked for optimality. If the optimality condition is met, the algorithm is terminated, else B is updated and another iteration is carried out at the new point obtained.

$$\text{if } \left\|\frac{\partial J}{\partial w}\right\| < \varepsilon, \text{ stop}$$

else

(4.26)
$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

$$s = W_n - W_c$$

$$y = \frac{\partial J}{\partial w}\bigg|_{W_n} - \frac{\partial J}{\partial w}\bigg|_{W_c}$$

The training procedure described above is valid regardless of the architecture of the neural network used. The only part that differs is the gradient of N_i with respect to the network parameters and the evaluation of the neural network to find N_i.

Having developed the necessary TANN background, we proceed to discuss the nonlinear neural network controller.

4.5 Application to Steam Regulation in Reheaters

We are going to discuss a plant with the following model:

(4.27) $$y(t+2) + a_1 y(t+1) + a_2 y(t) + f(\theta_N, \Phi_N) = b_0 u(t+1) + b_1 u(t)$$

We will set the desired closed-loop behavior to that of a second order system:

(4.28)
$$y(t+2) + a_{m1} y(t+1) + a_{m2} y(t) = (1 + a_{m1} + a_{m2}) u_c(t+1)$$
$$(q^2 + a_{m1} q + a_{m2}) y(t) = (1 + a_{m1} + a_{m2}) q u_c(t)$$

Again, as in Chapter 3, an observer polynomial is chosen:

(4.29) $$A_O(q) = q + a_o$$

The controller is made up of two parts. The first part is to feedback-linearize the plant and the second part is to place the closed-loop poles at the desired locations. The feedback linearization part aims to get rid of the nonlinearity of the plant. The second part of the controller will set the resultant system's behavior to that of the desired closed loop system.

The controller is of the form:

(4.30) $$u(t) = u_L(t) + u_N(t)$$

The parts of the controller are defined as $u_N(t)$ the feedback linearization part, and $u_L(t)$, the pole placement part (see Figure 4-3 for a block diagram of the controller).

(4.31) $$u_N(t) = Kf(\theta_N, \Phi_N)$$

(4.32)
$$u_L(t) = G(A_O(q)A_m(1)u_C(t-1) - A_o(1)A_m(1)y(t-2)) - u_L(t-1) + r(u_L(t-2) - u_L(t-1)) + s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

The plant input is constrained between u_{max} and u_{min}. If $u(t)$ is larger than u_{max}, it will be set to u_{max}, else if it is less than u_{min}, it will be set to u_{min}.

Finally, $\hat{u}(t)$, an estimate of the plant input is calculated as follows:

(4.33)

$$\hat{u}_L(t) = G(A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2) - u_L(t-1) + r(u_L(t-2) - u_L(t-1)) + s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$
$$\hat{u}(t) = \hat{u}_L(t) + u_N(t)$$

Using $u(t)$ and $\hat{u}(t)$, the error in equation (4.17) is calculated.

The parameter estimates r, s_0, s_1 and G are updated using equation (10) which is a gradient-descent algorithm , and the estimated parameters K, θ_N are updated using equation (11), which requires the TANN. The θ_L and Φ_L vectors come out to be :

(4.34)

$$\hat{\theta}_L(t) = [r, s_0, s_1, G]$$
$$\Phi_L(t) = [u_L(t-2) - u_L(t-1), y(t-2) - y(t), y(t-2) - y(t-1), (A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2))]$$

Equation (4.27), (4.28), (4.30) and (4.31) are implemented on the simulator. The results are discussed in Chapter 5.

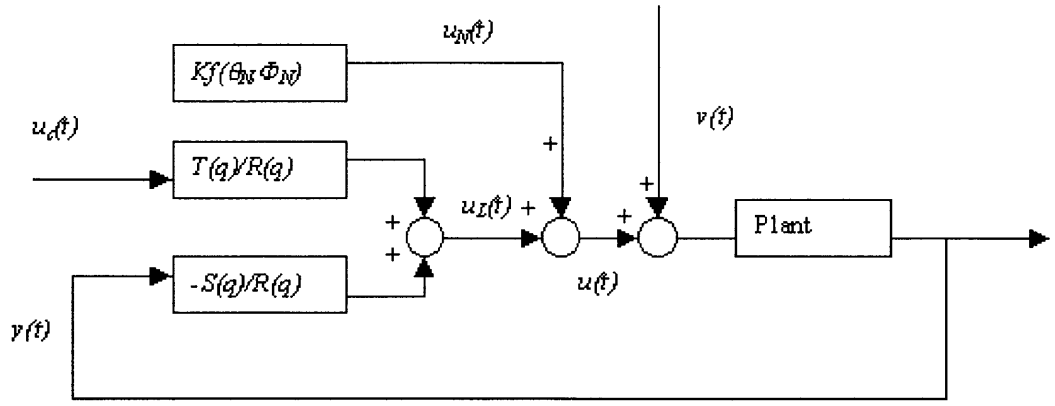

Figure 4-2 The Nonlinear Neural Network Controller

5. RESULTS

The controllers developed in the past chapters were applied to the reheater steam temperature regulation problem. The neural-adaptive controllers developed in Chapters 3 and 4 were implemented on the full-scale simulator model at the EPRI Instrumentation & Control Center. The linear and nonlinear neural network controllers were implemented as C code on the FOXBORO control system attached to the simulator of the plant. For a detailed diagram, refer to Figure 5-1.

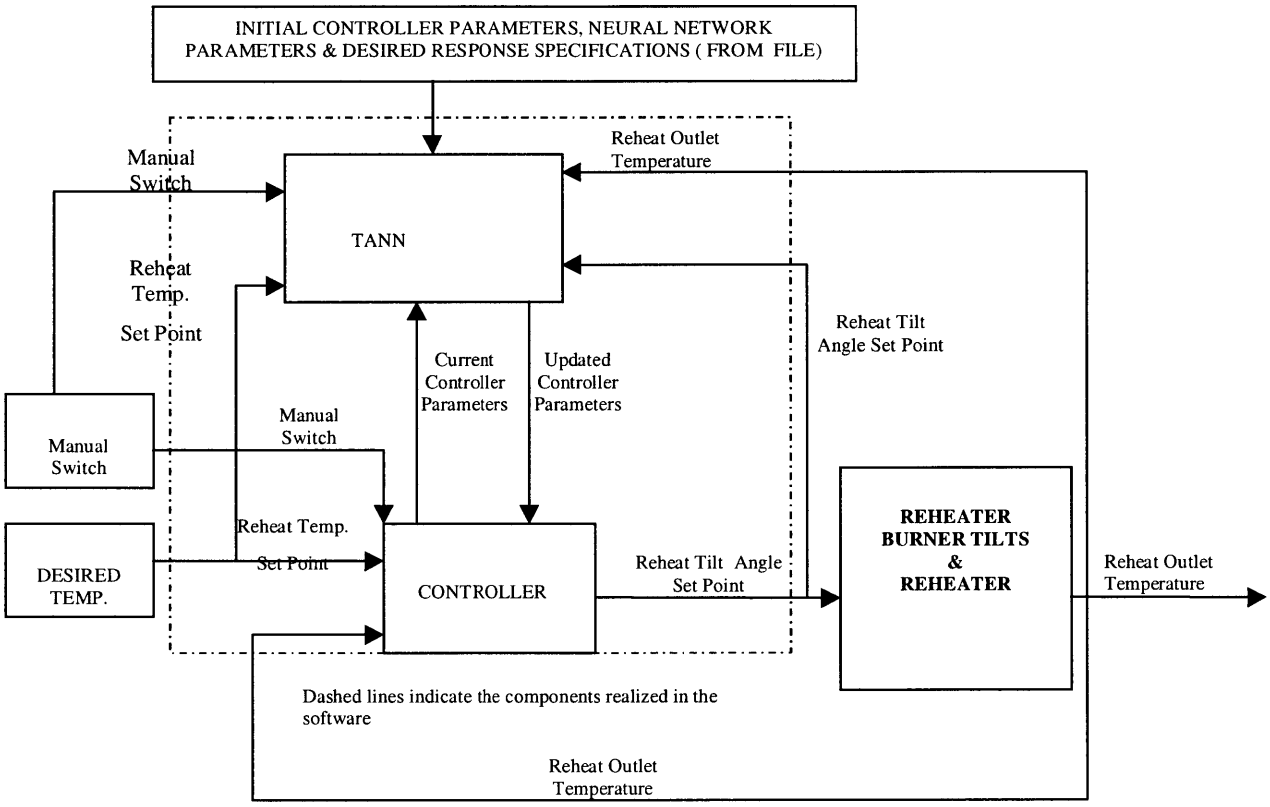

Figure 5-1 Controller Implementation

In this chapter, we present the simulation results obtained on the EPRI simulator. The first section is about the implementation of the linear neural and nonlinear neural network controllers as well as the existing PI controller. The second section is a comparison of performance on sootblowing which is a routine plant operation causing excessive temperature upsets. The sootblowing operation is carried out to prevent fouling. Fouling is the degradation of heat transfer from the flue gas to steam and air. The reason for this

degradation is the deposition of ash on the metal surfaces. As a consequence, these surfaces have to be cleaned repeatedly. This cleaning is achieved by sootblowing. On the simulator at EPRI, fouling is simulated as a percentage decrease in the heat transfer coefficients of the furnace, superheater, air heater and the reheater surfaces. The neural-adaptive controllers and the PI are compared at 0% fouling and 20% fouling during sootblowing. The last section displays the adaptive capabilities of the linear and nonlinear neural network controllers. It contains step response tests to show parameter adaptation.

5.1 Implementation of the Controllers

5.1.1 PI Controller

The PI controller is the current regulator on the plant. For the sootblowing simulations, the current PI controller settings were adjusted in order to obtain the best performance during sootblowing. Hence, the simulations of the PI controller were carried out for different sets of Proportional (K_p) and Integral (K_I) gains. The PI controller was tuned for different fouling levels of the plant.

One set of gains is called the current because it is the set of gains currently implemented. The other two sets of gains are the optimal PI gains for a plant with 0% fouling and 20% fouling. These optimal gains are optimal in the sense that the maximum unwanted deviation is as small as possible and the settling time of the system is as small as possible. It is a well known phenomenon that a system will start showing oscillatory behavior once the P and I gains are too high. On the other hand, the temperature deviation decreases with increasing Proportional and Integral gain. The tuning of the PI controller is a compromise between these two behaviors. Many techniques exist for adjustment and the reader should refer to [4].

The Foxboro control system uses the gains PBAND and INT instead of Proportional and Integral gains. The INT variable is the Integral Time expressed in minutes. PBAND denotes the percentage of the change in steam temperature with respect to the temperature range 0-1200 °F that will result in full deflection of the burners (60 degrees). PBAND and INT are defined as

$$PBAND = 100K_s / K_p$$

$$K_s = 60\,\mathrm{deg}/1200^{o}F \tag{5.1}$$

$$INT = K_p / (60K_I)$$

	Proportional Gain (PBAND)	Integral Gain (INT)
Current	0.333 degrees/ ^{0}F (15.0)	0.001 degrees/ (^{0}F sec) (5 min.)
0% fouling	1.000 degrees/ ^{0}F (5.0)	0.017 degrees/ (^{0}F sec) (1 min.)
20% fouling	1.111 degrees/ ^{0}F (4.5)	0. 015 degrees/ (^{0}F sec) (1.2 min.)

Table 5-1 PI Gains

The PI gains used in the simulations are tabulated in Table 5.1

5.1.2 Linear Neural Adaptive Controller

The second controller to be implemented was the linear neural adaptive controller. We applied the design procedure developed in Chapter 3 to the linear plant model obtained in Chapter 2:

(5.2) $$y(t+2)+a_1 y(t+1)+a_2 y(t)=b_0 u(t+1)+b_1 u(t)$$

A desired closed-loop system was chosen:

(5.3) $$y(t+2)+a_{m1} y(t+1)+a_{m2} y(t)=(1+a_{m1}+a_{m2})u_c(t+1)$$

The desired closed loop equation was calculated from a continuous-time second order plant model. For a second order closed-loop system in continuous-time, the performance is defined in terms of the closed loop natural frequency (ω_n) and the damping ratio (ζ). For this study, the natural frequency was chosen to be 0.05 rad/sec and the damping was chosen as 0.8. These specifications are for continuous-time plants. The desired closed-loop system is discrete-time. Therefore, the poles obtained in continuous-time were transformed into discrete-time poles for a sampling time of 5 seconds by zero-pole matching. The details can be found in any discrete-time control design text such as [29].

In addition, an observer polynomial had to be decided upon as discussed in Chapter 3:

(5.4) $$A_O(q)=q+a_o$$

Carrying out the design developed in Chapter 3, we obtained the following control law (5.5) and the corresponding adaptation law (5.7):

(5.5)

$$u(t) = G(A_O(q)A_m(1)u_c(t-1) - A_o(1)A_m(1)y(t-2)) - u(t-1) + r(u(t-2) - u(t-1))$$
$$+ s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

The θ and Φ vectors come out to be:

$$\hat{\theta}(t) = [s_0, s_1, G]$$
$$\Phi(t) = [y(t-2) - y(t), y(t-2) - y(t-1), (A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2))] \quad (5.6)$$

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \frac{\gamma\Phi(t)}{\alpha + \Phi(t)^T\Phi(t)}e \quad (5.7)$$
$$e = u(t) - \hat{u}(t)$$

(5.8)

$$\hat{u}(t) = G(A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2)) - u(t-1) + r(u(t-2) - u(t-1))$$
$$+ s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

The numerical values for the coefficients of the model in equation (5.2) are listed in Table 5.2; the coefficients of the desired closed loop characteristic equation and the observer polynomial are in Table 5.3. The initial controller parameters for adaptation are in Table 5.4 and the parameters of equation (5.7) can be found in Table 5.5.

a_1	-1.86072
a_2	0.8655
b_o	0.0181
b_1	0.0000

Table 5-2 Linear plant Parameters

a_{m1}	-1.6522
a_{m2}	0.7047
a_o	0.0525

Table 5-3 Closed-Loop and Observer Polynomial Coefficients

$G(0)$	55.5555
$s_1(0)$	19.9163
$s_0(0)$	-34.2855
$r(0)$	0.0000

Table 5-4 Initial Linear Neural Adaptive Controller Parameters

α	10.0
γ	0.05

Table 5-5 Adaptation Parameters

We would like to add some comments as to the choice of the numerical values.

The plant parameters were obtained as a result of the system identification process described in Chapter 2, considering a linear parameterization of the system.

The observer polynomial was chosen so as to have the observer pole as far from the plant poles without causing excessive actuator behavior. Note that the observer pole is a closed loop pole of the system. The observer pole was chosen between 0 and 1, so that the pole was well damped. In discrete-time systems, real negative poles have oscillatory behavior [29]. The parameters γ and α were experimentally tuned to obtain the best parameter adaptation.

In this implementation of the controller, the parameter r was not estimated. The range of values that r can take is between -1 and 1. As can be noted from the derivation of the

controller, r is the zero of the plant that is cancelled. If the zero lies outside the range of –1 and 1, then the closed-loop system becomes unstable. Also, there is an order of magnitude difference between the other parameters and r. The adaptation gain has to be kept small so that r does not leave the [-1,1] interval, but this will slow down the adaptation of the rest of the parameters significantly. Therefore, the parameter r was not estimated. One could if desired still adapt for r by constraining the values of r between –1 and 1.

5.1.3 The Nonlinear Neural Controller

5.1.3.1 Online Implementation

As discussed in Chapter 2, the performance of the reheater is nonlinear. The linear plant model in equation (5.2) was modified include a nonlinear term $f(\theta_N, \Phi_N)$. The resulting plant model was:

$$y(t+2) + a_1 y(t+1) + a_2 y(t) + f(\theta_N, \Phi_N) = b_0 u(t+1) + b_1 u(t) \tag{5.9}$$

While several choices of f are possible, we found that a nonlinearly parameterized f with the following form was found to be most suitable:

$$f(\Phi_N, \theta_N) = \frac{\tan(K(y(t-2) - y(t-1))}{\tan\left(K\left|u(t-1) + u_{nominal}\right|\right) + 0.5} \tag{5.10}$$

It should be noted that several such nonlinearities can be added if needed to enable representation of more complex nonlinear behavior in the power plant. The general plant model is given by:

$$y(t+2) + a_1 y(t+1) + a_2 y(t) + \sum_{k=1}^{K} f_k(\theta_{N_k}, \Phi_{N_k}) = b_0 u(t+1) + b_1 u(t) \tag{5.11}$$

Applying the controller design in Chapter 4, the control law was obtained as:

$$u(t) = u_L(t) + u_N(t) \tag{5.12}$$

$$u_N(t) = G\frac{\tan(K(y(t-2) - y(t-1))}{\tan\left(K\left|u(t-1) + u_{nominal}\right|\right) + 0.5} \tag{5.13}$$

(5.14)

$$u_L(t) = G(A_O(q)A_m(1)u_c(t-1) - A_o(1)A_m(1)y(t-2)) - u_L(t-1) + r(u_L(t-2) - u_L(t-1)) + s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

The parameter estimates, s_0, s_1 *were* updated using equation (5.16), and the estimated parameter K was updated using equation (5.17).

(5.15)

$$\hat{\theta}_N(t) = K$$
$$\Phi_N(t) = [y(t-2) - y(t-1),\ u(t-1) + u_{nominal}]$$
$$\hat{\theta}_L(t) = [s_0, s_1]$$
$$\Phi_L(t) = [y(t-2) - y(t),\ y(t-2) - y(t-1)]$$

(5.16)

$$\hat{\theta}_L(t+1) = \hat{\theta}_L(t) + \upsilon k \Phi_L(t)$$

(5.17)

$$\hat{\theta}_N(t+1) = \hat{\theta}_N(t) + \upsilon k N(\Phi_L(t), \hat{\theta}_N(t))$$

(5.18)

$$k = \frac{\alpha e}{1 + C_N^T(t)C_N(t) + \Phi_L(t)^T \Phi_L(t)} \quad \alpha \in (0,1)$$

(5.19)

$$e = u(t) - \hat{u}(t)$$

(5.20)

$$\hat{u}(t) = \hat{u}_L(t) + u_N(t)$$
$$\hat{u}_L(t) = G(A_O(q)A_m(q)y(t+1) - A_o(1)A_m(1)y(t-2) - u_L(t-1) + r(u_L(t-2) - u_L(t-1)) + s_0(y(t-1) - y(t)) + (s_0 + s_1)(y(t-2) - y(t-1))$$

(5.21)

$$C_N = \left.\frac{\partial f}{\partial K}\right|_{K_0}$$

υ is defined as follows:

(5.22)

$$\upsilon = \begin{cases} 1 \text{ if } \Delta V_d(t) \geq \varepsilon \\ 0 \text{ otherwise} \end{cases}$$

For the numerical values used in the controller, refer to Table 5.6.

Radial Basis Function network architecture was employed to calculate N in equation (5.17). The input to the neural network during on-line operation at every sampling instant was the vector x:

$$x = \{K, \Delta y, |u|\}, \tag{5.23}$$

$$\begin{aligned} \Delta y &= y(t-1) - y(t-2) \\ K &= K \\ |u| &= |u(t-1) - u_{nominal}| \end{aligned} \tag{5.24}$$

The neural network output for each x was calculated in the following manner

$$\begin{aligned} z_l &= e^{-\frac{|(x-c_l|^2}{2\sigma_l^2}}, \\ N &= \sum_{l=1}^{M} (w_l z_l), \end{aligned} \tag{5.25}$$

The c_l are the centers and the σ_l are the spread of each Gaussian Function used. The w_l are the weights (parameters) of the neural network. The determination of c_l, σ_l and w_l and their significance will be discussed in the off-line training section.

G	55.5555	$K(0)$	0.0500
$s_1(0)$	19.9163	ε	0.5
$s_0(0)$	-34.2855	K_o	0.8
$r(0)$	0.0000	α	0.0500

Table 5-6 Initial Nonlinear Neural Network Parameters

5.1.3.2 Off-line Training

The neural network had to be trained prior to implementation for parameter adaptation. In the following paragraphs, the training procedure is outline.

First, a training set of 1000 data sets was created. Each data set s_i was of the form $s_i = \{K_i, (K_{true})_i, \Delta y_i, |u|_i\}$. Each element in the data set was sampled randomly in its corresponding range (Table 5.7).

	Minimum	Maximum
K	-0.1	0.8
K_{True}	-0.1	0.8
Δy	-4	4
$\lvert u \rvert$	0	$\pi/6$

Table 5-7 The Range of Values used in Neural Network Training

As the specific structure of the neural network, the Radial Basis Function (RBF) architecture was chosen. The neural network consisted of a set of Gaussian Functions of the form: $f(x) = \exp(-|x-c|^2/2\sigma^2)$. The idea is to approximate a function as the superposition of Gaussians similar to Fourier Series that uses sinusoids. The Gaussian Function calculates the distance between input vector x and the center c. The center is the place where the maximum of the Gaussian is located in the input space. The center is unique for each Gaussian used. The σ value of each Gaussian determines how far from the center in the input space the Gaussian is effective. A larger spread means that the Gaussian is effective in a larger space since a large σ makes the value in the exponential smaller, meaning that the value the exponential returns is far from zero for relatively large distances between x and c.

There are many techniques to find the optimal c_l, σ_l and w_l. One can incorporate all of them in an optimization scheme, or one could optimize them separately. The centers can be chosen randomly and then a clustering algorithm can be used to optimally scatter them with respect to the training set . For further details on clustering, refer to [25,16,9]. The spread values can be determined by trial and error. There are some guidelines to decide how to choose the spreads. The spreads should not be very large. Large spreads mean too much overlapping of the Gaussians used. This in turn means that the Gaussians cannot approximate relatively fast variations of the function to be approximated. On the other hand, very small spreads mean that there are some regions in the input space that are not covered by the Gaussians. One heuristic rule is to choose the spreads as a multiple of the distance between centers that are closest to each other.

In this study, the centers were determined by clustering and the spreads were chosen by trial and error.

The output of the neural network for each data set was calculated as :

$$x_i = \{K_i, \Delta y_i, |u|_i\}$$

$$z_l = e^{-\frac{|(x-c_l|^2}{2\sigma_l^2}}, \tag{5.26}$$

$$N_i = \sum_{l=1}^{H} (w_l z_l),$$

H is the total number of centers. As described in Chapter 4, the output of the network N_i is checked for the validity of two conditions:

$$L_i = \left|\frac{\partial f}{\partial K}\right|_{\{K_o, \Delta y_i, |u|_i\}}^2 \tag{5.27}$$

$$D_i = G\left(\frac{\tan(K\Delta y)}{\tan(K|u|)+0.5} - \frac{\tan(K_{True}\Delta y)}{\tan(K_{True}|u|)+0.5}\right) \tag{5.28}$$

$$|N_i| \le L_i \tag{5.29}$$

$$N_i\left(K_i - (K_{True})_i\right) \le D_i \tag{5.30}$$

The cost function J was then minimized by calculating the optimal w by the Quasi-Newton method.

$$J = \frac{1}{2}\sum_{i=1}^{M}\left\{(\max\{0, N_i\left(K_i - \left(K_{True}\right)_i\right) - D_i\})^2 + \frac{1}{b^2}(\{\max\{0, |N_i|^2 - L_i)^2\})\right\} \tag{5.31}$$

In this study, the training set consisted of 1000 data sets, the number of centers and weights was 624. The spread, σ for all centers was determined to be 0.3.

5.2 Sootblowing Experiments

In our simulation studies, the whole plant was degraded. The sootblowing was carried out on the reheater furnace for all experiments. The reheater sootblowing steam flow is given so as to pinpoint the time during which sootblowing was carried out. Sootblowing begins five minutes after the initiation of the steam temperature recording. Sootblowing

ends automatically ten minutes later. Sootblowing is the act of cleaning the reheater and the superheater external surfaces by steam, which is taken from the drum or main steam flow. The sootblowing operation is carried out to prevent fouling. Fouling is the degradation of heat transfer from the flue gas to steam and air. The reason for degradation is deposition of ash on metal surfaces. These surfaces have to be cleaned repeatedly. This cleaning is achieved by sootblowing. Fouling is implemented on the simulator as a percentage decrease in heat transfer coefficients of the metal surfaces.

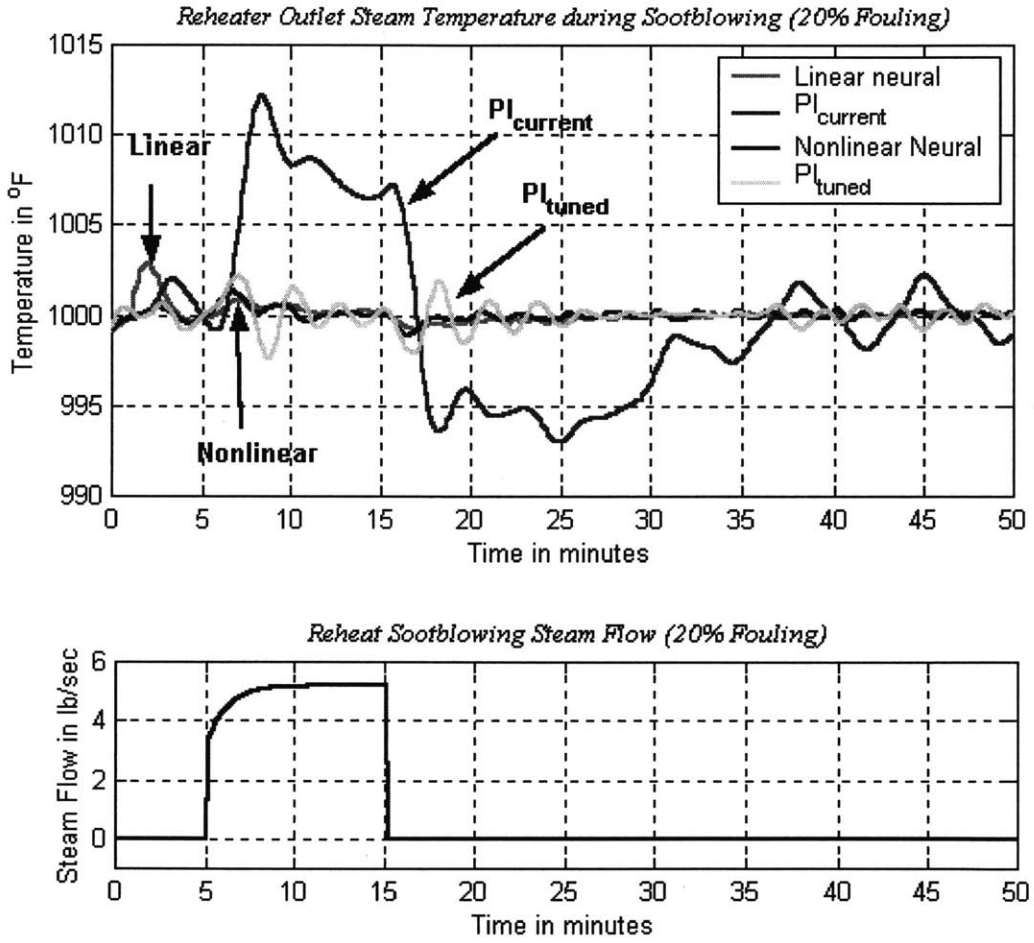

Figure 5-2 Reheat Steam Temperature during Sootblowing (20% Fouling)

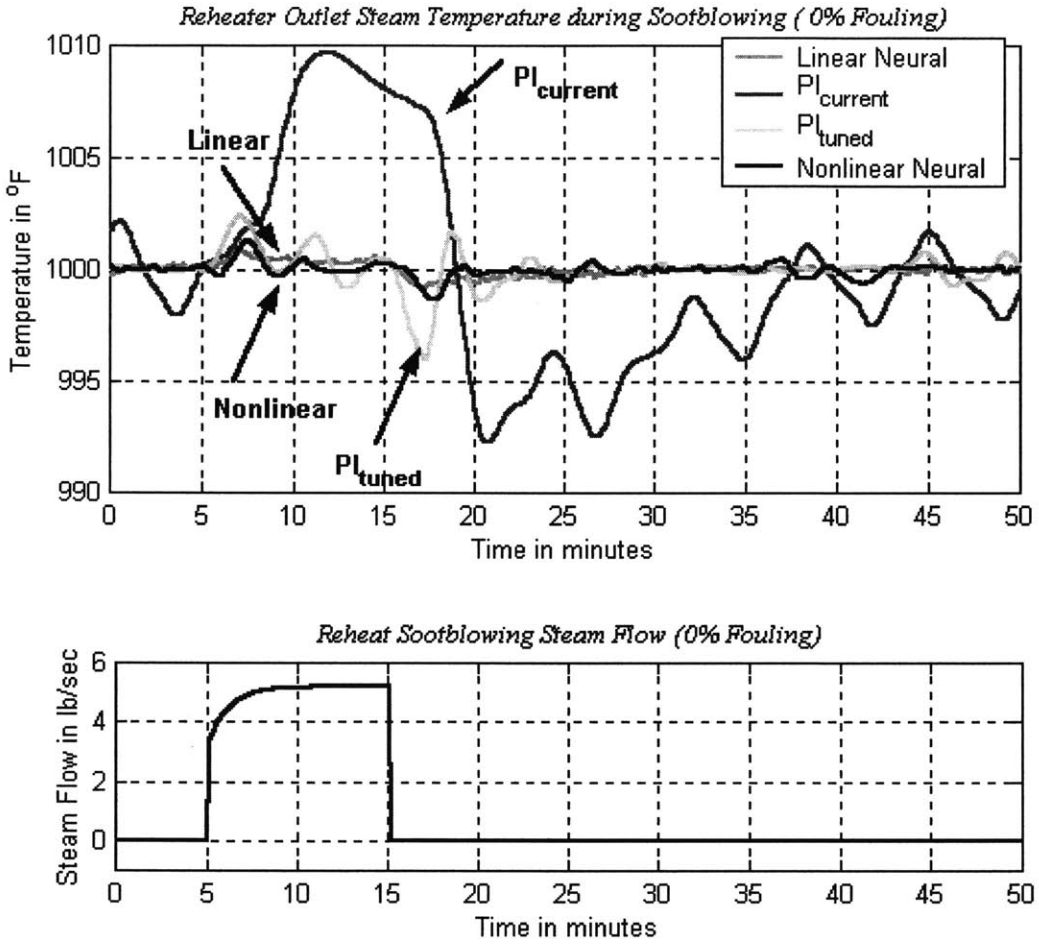

Figure 5-3 Reheat Steam Temperature during Sootblowing (0% Fouling)

That is, a 10% fouling of a metal surface corresponds to 10% decrease in the heat transfer coefficient of the metal surface from the nominal value. For this experiment, the controllers are compared at 0% fouling and 20% fouling during sootblowing. One advantage of a simulator is that one can save and reuse a certain setting or initial condition. The same initial condition and settings are used for the comparisons.

It is evident that the neural-adaptive controllers outperform the $PI_{current}$ controller (Figure 5.2 and Figure 5.3). The undesired temperature deviation is reduced from around 10^0 Fahrenheit to around 1.5^0 Fahrenheit, an improvement of 85%. The time it takes for the temperature to come back to the nominal temperature has been drastically reduced. The PI_{tuned} controller is an improvement over the $PI_{current}$, however, the performance is not as good as the neural controllers. The maximum deviation of temperature is around 3 degrees. The temperature shows an oscillatory behavior, which indicates that the controller brought the system close to instability. Moreover, the PI needs to be retuned for different operating conditions such as different plant load levels and fouling. The new

controllers do not need adjustment since they have self-tuning capabilities. The nonlinear neural network controller improves the performance of the linear neural adaptive controller as can be deduced from the improvement on the settling time of the nonlinear controller. The control inputs are compared in (Figure 5-4). The control input for the new controllers are calculated at five-second intervals, so they are discrete functions of time. The PI controllers are continuous–time controllers. This make the PI inputs smoother in nature than the neural controllers. Naturally, for a tighter control of temperature, the new controllers employ a larger bandwidth and this is visible from the graphs. Another improvement of the nonlinear neural network controller is that it has a smoother output than the linear neural adaptive controller does. The controller actuator bandwidth is not unreasonable since there was no problem in its implementation on the simulator, which includes some of the actuator constraints. If, in implementation on the actual plant, bandwidth constraints does occur, the sampling time of the system can be increased or the desired-closed loop poles can be chosen to be somewhat slower.

5.3 Step Response Tests

The adaptation capabilities of the neural controllers have to be checked. For this purpose, the controller parameters are detuned from the nominal and a series of step response tests are carried out to check whether the output of the system converges to the desired output. The desired temperature level in the reheater is changed in a stepwise manner and the controllers are let to adapt.

The parameters of the controllers are detuned and a series of step response tests are carried out to see how the parameters are updated and how the performance of the system is affected.

From the step response graphs (Figure 5-5), it is clear that the TANN is capable of adapting the parameters of the controllers successfully. The plant outputs become almost the same as the desired. This shows that in case the controllers are detuned due a change in the plant, the adaptation mechanisms will be able to compensate for that change.

The parameters do not converge exactly to the desired parameter values because the step change in the desired temperature does not have the necessary power to excite the adaptation mechanism fully (Figure 5.6 and Figure 5.7). This is a known concept in adaptive control where it is said that the signal is not persistently exciting. For further discussion on this topic, refer to [2,28]. The plant could have been excited with signals that have the persistent excitation property, but such signals would be uncommon in usual plant operations and therefore would not give an idea of what actually might happen.

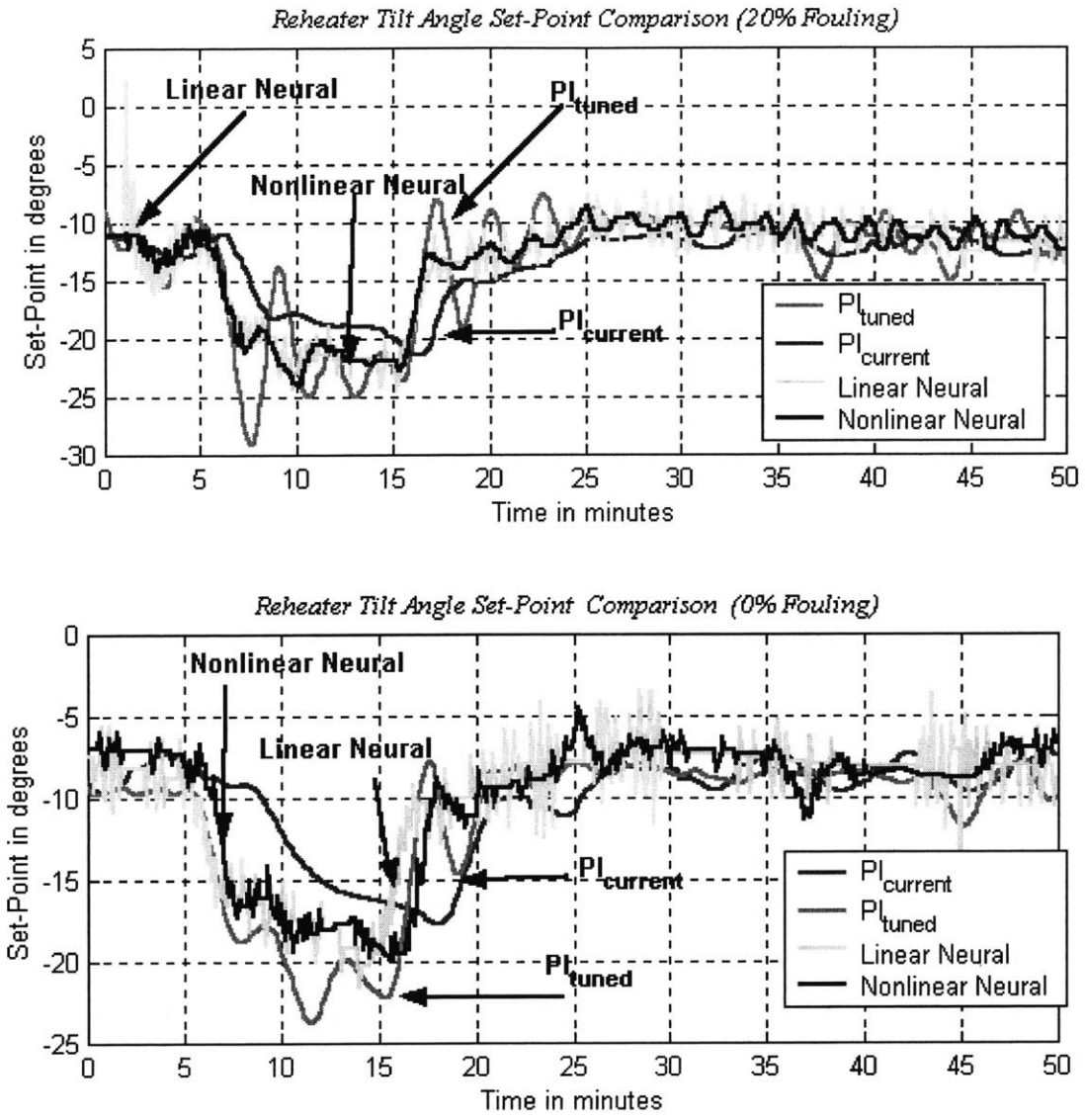

Figure 5-4 Actuator Set-Points

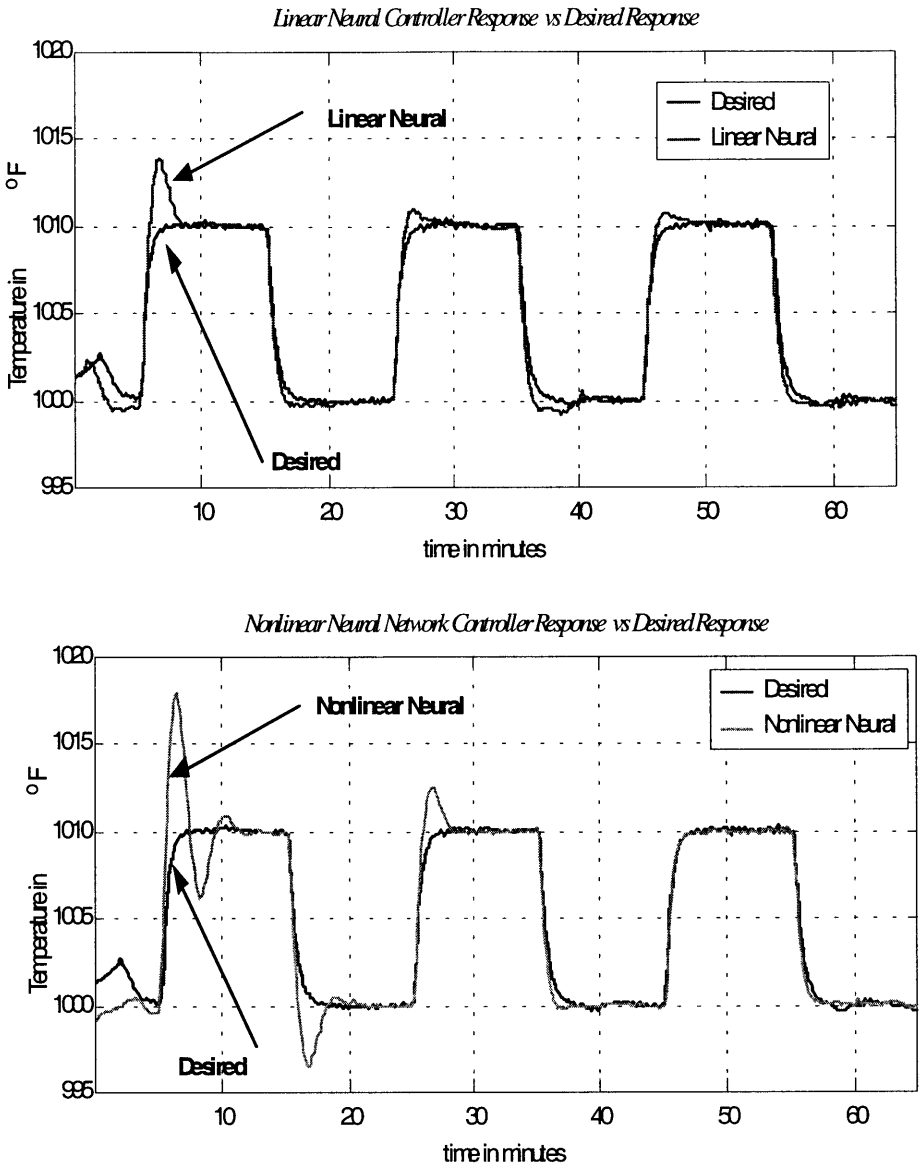

Figure 5-5 Step Response of the Neural-Adaptive Controllers

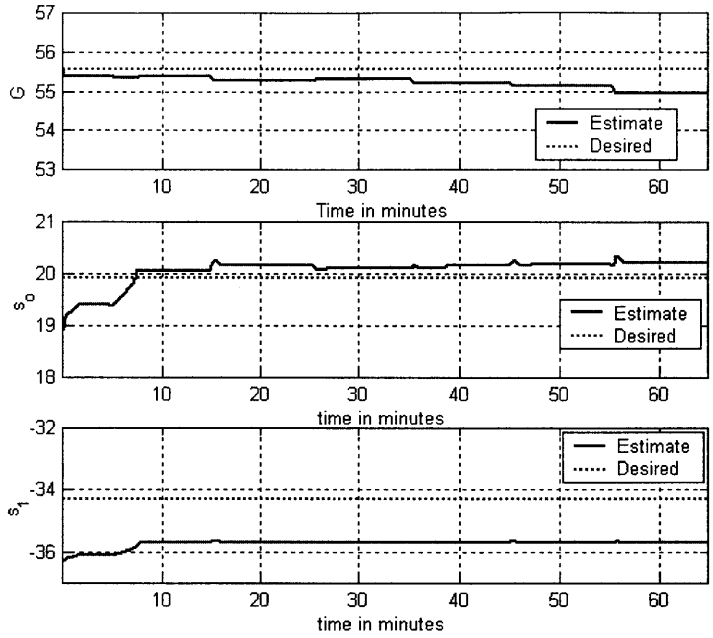

Figure 5-6 Estimated Parameters for Linear Neural Adaptive Controller

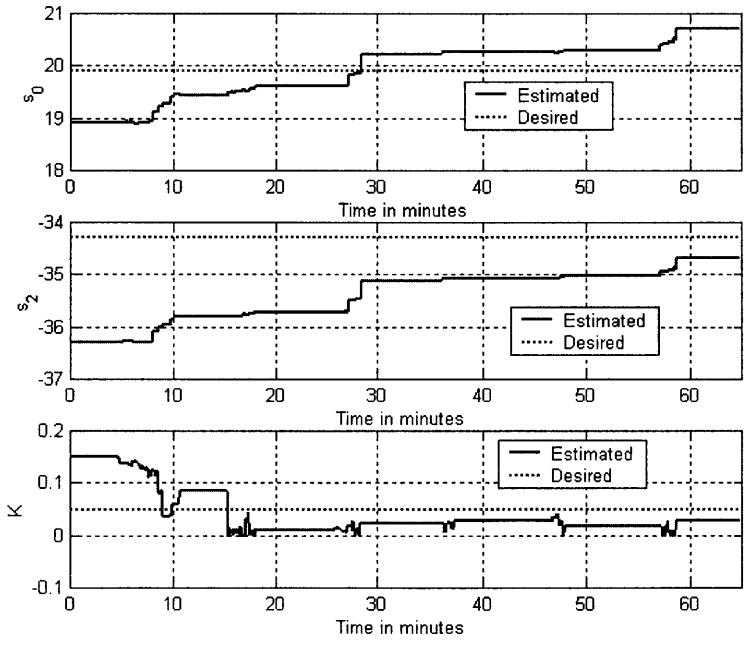

Figure 5-7 Estimated Parameters for Nonlinear Neural Network Controller

6. CONCLUSION

In this thesis, neural-adaptive are described for steam temperature regulation in fossil fuel power plants. The algorithms were developed in response to the need for improving the steam regulation properties of the existing controllers. Currently, PI controllers are used. These controllers were observed to exhibit temperature spikes in the event of changing operating conditions. Since the PI controllers have no auto-tuning and operate based on linear principles, nonlinear and these controllers cannot adequately accommodate time-dependent system behavior. Steam temperature behavior in power plants is strongly dependent on load output of the boiler, the cleanliness, and the main steam flow and fuel properties. Moreover, the behavior changes as a result of equipment replacement or malfunctions. The heat transfer from the flue gas to steam makes this system inherently nonlinear. A new class of adaptive controllers was developed so as to address the nonlinear characteristics of steam temperature dynamics in a fossil power plant. The controllers have the ability to auto-tune their parameters on-line and respond to nonlinear dynamic characteristics.

The specific problem that we addressed concerns temperature regulation at the outlet of the reheater or superheater. The first controller referred to as the linear neural adaptive controller is capable of controlling linearly parameterized systems. A linear model was obtained by system identification as a starting point for development. The control law is a Pole-Placement control algorithm, which is linearly parameterized. The adaptation law supplies the auto-tuning capability using a gradient-descent-based algorithm. The controller also accommodates plant saturation effects by including them in the adaptation law and effects of set-point disturbances by including an integral action. The second algorithm developed is called the nonlinear neural network controller. It is an improvement over the first controller in the sense that it extends the capabilities of the first controller to control nonlinear plants. The controller has a pole-placement structure augmented by a feedback-linearizing substructure. The parameter adaptation is jointly managed by a gradient-descent-based algorithm for linearly occurring parameters and by the TANN [1, 36] for nonlinearly occurring parameters.

The algorithms are implemented on a full-scale and detailed simulator at the EPRI Instrumentation & Control Center, Kingston, Tennessee using the Foxboro DCS system. The performance of the controllers is compared to the present PI implementation. An order of magnitude improvement in the settling time after a major temperature upset was achieved as well as an order of magnitude decrease in the maximum temperature deviation. Performance improvement was observed both in the context of command following and disturbance rejection.

The work in this thesis can be extended in several directions.

1. A systematic approach to the development and identification of nonlinearities in systems can be developed, as is the case in linear identification.

2. The controllers can be extended into the MIMO case where the superheater and the reheater temperature can be controlled in a combined manner. Although SISO policies work well, MIMO controllers can still improve performance by utilizing the coupling between reheat and superheat temperatures.
3. The techniques employed can be applied to other control problems in boilers. The TANN can be used in other control problems such as pressure control and load control.
4. The steam regulation problem is one aspect of boiler controls. Three different controllers are implemented to regulate three inter-related properties of steam: the main pressure controller at the inlet of the high-pressure turbine, the temperature controller at the reheater and superheater and the load controller controlling the governor valve and the firing rate. A unified controller that controls pressure and load as well as steam temperature can be designed to optimize the load output of the plant.

7. REFERENCES

1. Annaswamy, A.M. and Yu S. ," Adaptive Control of Nonlinear Dynamic Systems Using θ-Adaptive Neural Networks", *Automatica*, Vol. 33, No.11, pp.1975-1995, 1997

2. Åström, Karl J. and Wittenmark, Björn, *Adaptive Control 2. Edition*, Reading Massachusetts: Addison-Wesley Publishing Company, Inc., 1995.

3. Åström, Karl J., Bell, R. D., "A Nonlinear Model for Steam Generation Process." *IFAC 12th Triennial World Congress,* Sydney, Australia, 1993.

4. Åström, Karl J. and Hägglund, T., *Automatic Tuning of PID Controllers,* Research Triangle Park, NC: ISA, 1988.

5. Babcock, G.H., Wilcox, S., *Steam, Its Generation and Use*, New York: Babcock & Wilcox, 1978

6. Berkowitz, David A. (editor), *Proceedings of the Seminar on Boiler Modeling*, Bedford, Massachusetts: The MITRE Corporation, 1974.

7. Bertsekas, Dimitri P., *Nonlinear Programming*, Belmont, Mass. : Athena Scientific, 1995

8. Çengel Yunus A., Boles Michael A., *Thermodynamics, An Engineering Approach, Second Edition,* New York: McGraw-Hill, Inc., 1994

9. Chinrungrueng, Chedsada and Séquin, Carlo, H., "Optimal Adaptive K-Means Algorithm with Dynamics Adjustment of Learning Rate", *IEEE TRANSACTIONS ON NEURAL NETWORKS,* VOL. 6., No. 1, January 1995.

10. Dennis, John, E., and Schnabel, Robert B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Eglewood Cliffs, N.J: Prentice-Hall, 1983.

11. Dukelow, Sam G., Lipták, B. G., "Boiler Control and Optimization", *Instrument Engineers Handbook, 3. Edition, Process Control*, Pennsylvania: Chilton (Belá G. Lipták editor-in-chief) Book Company, 1995.

12. Dukelow, Sam G., *The Control of Boilers*, Research Triangle Park, NC: ISA, 1986.

13. Eklund, K., "Linear drum-boiler-turbine models", Ph.D Thesis, TFRT-1001, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1971

14. Goodwin, Graham, C., Sin, Kai, Sang, *Adaptive Filtering and Prediction,* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1984.

15. Hagan, Martin T., Menhaj, Mohammad B. "Training FeedForward Networks with the Marquardt Algorithm", *IEEE TRANSACTIONS ON NEURAL NETWORKS,* VOL. 5., No. 6, November 1994.

16. Hassoun, Mohamad, H., *Fundamentals of Artificial Neural Networks,* Cambridge, Massachusetts: The MIT Press, 1995.

17. Incropera, Frank P., DeWitt, David P., *Introduction to Heat Transfer, Third Edition* New York: John Wiley & Sons, 1996

18. Kwan, H.W., Anderson, J.H., 'A Mathematical Model of a 200 MW Boiler", *Int. J. Control, Vol. 12,* No. 6, pp. 977-998, 1970.

19. Ljung, Lennart, *System Identification: Theory for the User 2. Edition*, New Jersey: Prentice-Hall, Inc., 1999.

20. Maffezoni, C., "Boiler-Turbine Dynamics in Power-Plant Control", *Control Eng. Practice,* Vol.5, No.3, pp.301-312, 1997.

21. Maffezoni, C., "Issues in Modeling and Simulation of Power Plants", *IFAC Symposium on Control of Power Plants and Power Systems*, Munich (Germany), 1992.

22. Mann, J., and Lausterer, G. K., "Temperature Control Using State Feedback in Fossil Fired Power Plant", *IFAC Symposium on Control of Power Plants and Power Systems*, Munich (Germany), 1992.

23. Matsumura, S., Ogata, K., Fuji, S, Shioya, H. and Nakamura, H., " Adaptive Control for the Steam Temperature of Thermal Power Plants", *Control Eng. Practice,* Vol.2, No.4, pp.567-575, 1994.

24. McDonald, J.P., Kwatny, H. G., Spare, J. H., "Nonlinear model of a reheat boiler-turbine generator system", Philadelphia Electric Company, Research Division Report No. E-196 (1971).

25. Mehrotra, Kishan, Mohan, Chilukuri K., Ranka, Sanjay, *Elements of Artificial Neural Networks,* Cambridge, Massachusetts: The MIT Press, 1996.

26. Mello, F.P., "Dynamic Models for Fossil Fueled Steam Units in Power System Studies", *IEEE Transactions on Power Systems,* Vol.6, No.2, pp. 753-761, May 1991.

27. Nakamura, H., Toyoda, Yukihiro and Oda, Keiji, "An Adaptive Control System for Steam Temperature of Fossil-Fuel-Fired Thermal Power Plant", *IFAC Power Plants and Power Systems*, Cancun, Mexico, 1995.

28. Narendra, K.S. and Annaswamy, A.M., *Stable Adaptive Systems*, Englewood Cliffs, N.J: Prentice-Hall, Inc., 1989.

29. Ogata, Katsuhiko, *Discrete-Time Control Systems*, New Jersey: Prentice-Hall, Inc., 1995.

30. Paranjape, R.D.," Modeling and Control of a Supercritical Coal Fired Boiler", Ph.D Dissertation, Texas Tech University, Lubbock, TX, 79409, 1996.

31. Surgenor, B.W., and Pieper, J.K., "Practical Optimal Multivariable Control Revisited with Application to Steam Temperature Control in a Boiler", ASME Transactions, 89-WA/DSC-23, San Francisco, 1989.

32. Uchida, M., Nakamura, H., Toyota, Y. and Kushishashi M., 'Implementation of Optimal Control at a Supercritical Variable-Pressure Thermal Power Plant", *IFAC Power Systems and Power Plant Control*, Beijing, 1986.

33. Unbehauen, H. and Kocaarslan, I., "Experimental Modeling and Adaptive Power Control of a 750 MW Once-Through Boiler", *IFAC 11th Triennial World Congress,* Talinn, Estonia, 1991.

34. Unbehauen, H., "Load dependent Multivariable Steam Temperature Control System in a Boiler", *Automatica,* Vol.5, pp.421-432, 1969.

35. Yang, Z.Y., Shen Z.J. and Ren, M, "Model Reference Adaptive Prediction Control of Steam Temperature Process in Power Plant", *IFAC 11th Triennial World Congress,* Talinn, Estonia, 1991.

36. Yu, Ssu-Hsin, "Model-based identification and Control of Nonlinear Dynamic Systems using Neural Networks", Ph.D Thesis, M.I.T, Cambridge, MA 02139, 1996.

APPENDIX A: C-CODE FOR LINEAR NEURAL ADAPTIVE CONTROLLER FOR FOXBORO DCS

```
/*PROGRAM WRITEN FOR ADAPTIVE CONTROL*/
/*THE INPUT & OUTPUT AND REFEREFENCE SIGNALS ARE FILTERED*/
/*WRITTEN BY MEHMET YUNT FINAL FORM  MAY 20, 2000*/
/*THE FOLLOWING ARE THE FOXBORO NAMES OF THE VARIABLES TO BE READ*/

/*NAME                          VARIABLE          TYPE        */

/*REHEAT CONTROL TOGGLE         9RH:B27.MA        BOOLEAN     */
/*REHEAT TEMPERATURE            9RH:B4.OUT        FLOAT       */
/*REHEAT TEMPERATURE SETP       9RH:B27.SPT       FLOAT       */
/*REHEAT TILT  HILIMIT          9RH:B27.HOLIM     FLOAT       */
/*REHEAT TILT  LOLIMIT          9RH:B27.LOLIM     FLOAT       */
/*REHEAT ACTUAL ANGLE1          9RH:B28.MEAS      FLOAT       */
/*REHEAT ACTUAL ANGLE2          9RH:B31.MEAS      FLOAT       */
/*REHEAT ACTUAL ANGLE3          9RH:B34.MEAS      FLOAT       */
/*REHEAT ACTUAL ANGLE4          9RH:B37.MEAS      FLOAT       */

/*THE FOXBORO VARIABLE WHERE THE CONTROL OUTPUT WILL BE WRITTEN      */

/*REHEAT TILT ANGLE          9RH:B27.OUT    FLOAT          */

/*Use uread in main function to write and scopen readval combination in function to read*/
/*Controller will function as long as PI on Reheater is disabled, 9RH:B27.MA=0          */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define WRITPARAMS  1
#define INPUTS  1
#define OUTPUTS  1
#define READPARAMS  10
#define N_Parameter 9
#define N_temp  3
#define N_inp 2
typedef union
      {
         long lval;
         float fval;
         } IAXVAL;


 void  openReadSet(int *readDset, int readIndex[]);
 void  readInputs(int readDset, int readIndex[]);
 void  controller(void);
 void  adaptation(void);
 void  readData(void);
 void  printData(void);


float
Par[N_Parameter+1],u[6],uc[6],y[6],uf[6],yfa[6],yfb[6],Set_RHtilt,TCURRENT,TILTEQ,TEQ,MAXTHE
TA;
float Am[3],Ac[4],T[2],AmAo1,Gamma,Alpha,G,r,s1,s2,TS,Am1,NORM,Ao[3],ROBUST,RADAPT;
float LOLIMIT, ACTUAL1, ACTUAL2, ACTUAL3, ACTUAL4, RHSET, SETPOINT, HILIMIT;
char  TOGGLE,CHECKTOGGLE;
```

```
main()
{

   int i,j,k,reterr,z,error[1], readIndex[READPARAMS], readDset;
   FILE *fptr;
   static int gw[1]=        {1};
   static int valtyp[1]     =       {3};
   static int nument        =       1;
   char name[1][32];
   IAXVAL value[WRITPARAMS], rvalue[WRITPARAMS];

   z=1;

   printf("Program started succesfully \n");
   fflush(stdout);


   printf("Input Equilibrium Temperature \n");
   scanf("%f",&TEQ);

   printf("Input Equilibrium TILT ANGLE \n");
   scanf("%f",&TILTEQ);

   printf("Input Sampling Time \n");
   scanf("%f",&TS);

   printf("Adapt for R -> y=1.0, n=0.0 \n");
   scanf("%f",&RADAPT);

   printf("Apply Robustification y=1.0, n=0.0 \n");
   scanf("%f",&ROBUST);

    printf("Maximum Norm  = \n");
   scanf("%f",&MAXTHETA);

    fptr =fopen("/opt/epri/mit/Param.dat","w");

     /*Open the parameter file*/
   readData();
   printf("The Initial Parameter Estimates are read from the Parameter file \n");
   fflush(stdout);

         uc[1]=SETPOINT-TEQ;
uc[2]=SETPOINT-TEQ;
         uc[3]=SETPOINT-TEQ;


   /*Open read set*/
   openReadSet(&readDset, readIndex);
   printf("Set to be read is open \n");


   /*Read first data*/
```

```
readInputs(readDset, readIndex);
printf("First values are read \n");
printData();


/*The  PastTemp and PastInp vectors must be initialized  */
/*2 minutes of time for switching toggle on*/
/*Actually 2 cycles are enough for initialization*/

   for  (i =1; i<=5; i++){

      readInputs(readDset, readIndex);
        u[6-i]=RHSET-TILTEQ;
        y[6-i]= TCURRENT-TEQ;
         uc[6-i]=SETPOINT-TEQ;
         uf [6-i]=0.0;
        yfa[6-i]=0.0;
        yfb[6-i]=0.0;

         sleep(TS);
   }
      printf("Initialization Complete  \n");
      printData();
      fflush(stdout);

CHECKTOGGLE=TOGGLE;

while(CHECKTOGGLE==TOGGLE){

                readInputs(readDset,readIndex);

       /*Writing the current parameters to file*/

fprintf(fptr,"%f %f %f %f \n",G,r,s1,s2);

printf("%f %f %f %f\n",G,r,s1,s2);

                fflush(stdout);
       /*Adaptation of parameters*/

                adaptation();

       /*Control input is calculated*/

                controller();

printf("Control input  calculated \n");
                fflush(stdout);

                /*Control input is written*/
                strncpy(name[0],"9RH:B27.OUT", 32);
                value[0].fval=Set_RHtilt;
                uwrite(gw, &nument, name, valtyp, value, error, &reterr);
                printf("RHSET = %f  \n",Set_RHtilt);
                fflush(stdout);
                sleep(TS);
```

```
    }/*Parentheses of while*/

      fclose(fptr);
      printf("Exiting program PI controller is on \n");
      fflush(stdout);


      /*Closing the read set*/
      clsset(readDset ,&reterr);
      printf("The sets are closed \n");
      fflush(stdout);


         /*End program*/
         printf("Program is shutting down  \n");
}


/*The adaptation has robustness properties   */
/*Everytime adaptation is called the y value is shifted*/
/*The signals are filtered                                  */

void adaptation()

{

 float YF,Kbar,ubar,Error,dkdr,dkds1,dkds2,dkdG,Co,dTheta[5],Flag;
 int i;

FILE* fptr;
Flag =1;
fptr=fopen("FSignals.dat","a+");

/* SHIFTING THE Y VECTOR */
     y[5]=y[4];
     y[4]=y[3];
     y[3]=y[2];
     y[2]=y[1];
     y[1]=TCURRENT-TEQ;

     if (fabs(y[1]-y[2])<0.01000) {
        y[1]=y[2] ;
}

      for(i=1; i<=4; i++) {

        yfa[6-i]=yfa[6-i-1];
        yfb[6-i]=yfb[6-i-1];
          uf[6-i]=uf[6-i-1];
       }


 /*Filtering the signals*/
/*YF   = Ac[1]*y[1]+Ac[2]*y[2]+Ac[3]*y[3]+Ac[4]*y[4]-AmAo1*y[2];*/
/*ubar  = G*YF+u[2]+r*(u[3]-u[2])+s1*(y[4]-y[2])+s2*(y[4]-y[3]);*/
```

```
/*Error = u[1]-ubar;*/

  yfa[1] =-Ac[2]*yfa[2]-Ac[3]*yfa[3]-Ac[4]*yfa[4]+AmAo1*y[2];
  yfb[1] =-Ac[2]*yfb[2]-Ac[3]*yfb[3]-Ac[4]*yfb[4]+y[2];
  uf [1] =-Ac[2]*uf[2]-Ac[3]*uf[3]-Ac[4]*uf[4]+u[1];

   YF     = y[1]-yfa[1];
   ubar   = G*YF+uf[2]+r*(uf[3]-uf[2])+s1*(yfb[4]-yfb[2])+s2*(yfb[4]-yfb[3]);
   Error  = uf[1]-ubar;


   /*Adaptation of the parameters*/

   /*dkdr*/

   dkdr=(uf[3]-uf[2])*RADAPT ;

  /*dkds1*/

   dkds1=(yfb[4]-yfb[2]);

  /*dkds2*/

  dkds2=(yfb[4]-yfb[3]);

  /*dkdG*/

  dkdG =YF;

fprintf(fptr,"%f %f %f %f %f %f %f %f %f %f
\n",yfa[1],yfb[2],uf[1],YF,ubar,Error,dkdr,dkds1,dkds2,dkdG);


if ((y[1]-uc[1])*(y[1]-uc[1])<0.5){Flag=0;}


fclose(fptr);
 Co  = RADAPT *dkdr*dkdr+dkds1*dkds1+dkds2*dkds2+dkdG*dkdG;

  dTheta[1] = Gamma*dkdr/(Alpha+Co)*Error;
  dTheta[2] = Gamma*dkds1/(Alpha+Co)*Error;
  dTheta[3] = Gamma*dkds2/(Alpha+Co)*Error;
  dTheta[4] = Gamma*dkdG/(Alpha+Co)*Error;

/*The Robustness Criteria will be implemented here after the whereabouts of the true parameters are
known*/

/*Modifying Control Parameters*/

  r  =r+dTheta[1]*RADAPT*Flag;
  s1 =s1+dTheta[2]*Flag;
  s2 =s2+dTheta[3]*Flag;
     G =G+dTheta[4]*Flag;

/*Robustness*/
```

```
        if(ROBUST>0.0){

        NORM = G*G+s1*s1+s2*s2+RADAPT*r*r ;
        NORM = sqrt(NORM);

                if (NORM>MAXTHETA){
                G =G/NORM*MAXTHETA;
                r =r/NORM*MAXTHETA;
        s1=s1/NORM*MAXTHETA;
                s2=s2/NORM*MAXTHETA;
          }
        }

}/*End Adaptation*/

/*This function calculates the Reheater Tilt Setpoint*/
/*The parameter vector is global                */


void  controller()

{

        /*shift the control input vector*/

     float K;
     u[5]=u[4];
     u[4]=u[3];
     u[3]=u[2];
     u[2]=u[1];
     u[1]=Set_RHtilt-TILTEQ;

        /*shift the reference input*/
        uc[2]=uc[1];
      uc[1]=SETPOINT-TEQ;

        printf("the u->1 to 6 vector: %f,%f,%f,%f,%f \n",u[1],u[2],u[3],u[4],u[5]);
        printf("the y->1 to 6 vector: %f,%f,%f,%f,%f \n",y[1],y[2],y[3],y[4],y[5]);


        K = Am1*(Ao[1]*uc[1]+Ao[2]*uc[2])-AmAo1*y[1];

Set_RHtilt=u[1]+r*(u[2]-u[1])+s1*(y[3]-y[1])+s2*(y[3]-y[2])+G*K;

       if (fabs(Set_RHtilt-u[1])<0.300){Set_RHtilt=u[1];}

        Set_RHtilt=Set_RHtilt+TILTEQ;


        printf("The Calculated Control Input = %f \n",Set_RHtilt);

        /*Check for saturation of input*/

        if (Set_RHtilt>=HILIMIT) {

                Set_RHtilt = HILIMIT;
```

```
                        }

        if (Set_RHtilt<=LOLIMIT)
        {
                Set_RHtilt = LOLIMIT;
                }
    printf("The Saturated Control Input = %f \n",Set_RHtilt);


}/*End of controller*/

        void openReadSet(int *readDset, int index[])

{
        int i;
        int gw[10]          =       {1,1,1,1,1,1,1,1,1,1 };
        int valtyp[10]      =       {5,3,3,3,3,3,3,3,3,3};
        float rdelta[10]    =       {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.1,0.1,0.1};
        float wdelta[10]    =       {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.1, 0.1,0.1};
        int nument          =       10;
        int rsr             =       1;
        int wsr             =       1;
        int clexit          =       1;
        int acctyp          =       1;
        char name[READPARAMS][32];
        int reterr, error[READPARAMS];
        strncpy(name[0],"9RH:B27.MA", 32);
        strncpy(name[1],"9RH:B4.OUT" , 32);
        strncpy(name[2],"9RH:B27.SPT" , 32);
        strncpy(name[3],"9RH:B27.HOLIM", 32);
        strncpy(name[4],"9RH:B27.LOLIM", 32);
        strncpy(name[5],"9RH:B28.MEAS", 32);
        strncpy(name[6],"9RH:B31.MEAS", 32);
        strncpy(name[7],"9RH:B34.MEAS", 32);
        strncpy(name[8],"9RH:B37.MEAS", 32);
        strncpy(name[9],"9RH:B27.OUT", 32);

        printf("The name variable is created");
    fflush(stdout);

scopen(gw,nument,name,valtyp,acctyp,rdelta,clexit,rsr,wsr,wdelta,readDset,index,error,&reterr);

    printf("Scopen Succesful");

        if ((*readDset)==0) {

                printf("Cannot open set correctly \n");
                printf("reterr = %d\n",reterr);
                for(i=0;i<READPARAMS ;i++)
                        {
                 printf ("%-32s %-04x\n", name[i],error[i]    );
                        }
            exit(0);
        }
}/*end of function*/
```

```
void readInputs(int set, int index[])
{
        typedef union

        {
                long lval;
                float fval;
                char bval;
        } MIAXVAL;

        int i;
        MIAXVAL  value[READPARAMS];
        int status,type;

        if (gsnent(set)!=READPARAMS)

        {
         printf("Read from the Wrong Set  \n");
         exit(1);
        }

        for(i=1; i<READPARAMS;i++)/*1 Read the float values*/
        {
         value[i].lval=readval(index[i]);
         status =readsta(index[i]);
         type=status &  7;

         if(type != 3)
                {
                        printf("the type of %d th read parameter is not float, but type %d!\n",i,type);
                }  /*1*/
        }

     /*Read the Boolean Toggle Variable*/

    value[0].bval=readval(index[0]);
        status =readsta(index[0]);
        type=status &  7;

        if(type != 5)
         {
          printf("the type of %d th read parameter is not Boolean, but type %d!\n",i,type);
     }


TOGGLE          =       value[0].bval;
TCURRENT        =       value[1].fval;
SETPOINT        =       value[2].fval;
HILIMIT                 =       value[3].fval;
LOLIMIT         =       value[4].fval;
ACTUAL1         =       value[5].fval;
ACTUAL2         =       value[6].fval;
ACTUAL3         =       value[7].fval;
ACTUAL4         =       value[8].fval;
RHSET           =       value[9].fval;
```

```
}/*End function*/


/*The Mparameter.dat File contains the following                    */
/*Initial estimates for the four parameters->4 parameters           */
/*G =gain or 1/bo                                                   */
/*r =feedback denominator coefficient                               */
/*s1=feedback numerator coefficient                                 */
/*s2=feedback numerator coefficient                                 */
/*The coefficients for the desired following transfer functions     */

/*Am=third order monic polynomial ->2 coefficients                  */
/*Ao=first order monic polynomial ->1 coefficient                   */
/*T =first order      polynomial ->0 coefficient                    */
/*T contains the desired gain as well(Am'(1))                       */
/*Gamma =adaption gain              ->1 parameter                   */
/*alpha =adaptation constant        ->1 parameter                   */

void  readData(void )
{
        int i;

        FILE *fptr ;

        fptr = fopen("/opt/epri/mit/Mparameter.dat", "r");
        printf(" File Pointer assigned succesfully \n");
        fflush(stdout);

    /*PAR is of length 10 but the zeroth element is not used*/
        for(i=1; i<=N_Parameter; i++)  fscanf(fptr,"%f ", &Par[i]);

        G =Par[1];
        r =Par[2];
        s1=Par[3];
        s2=Par[4];
        Am[1] = 1.0;
        Am[2] = Par[5];
Am[3] = Par[6];
        Ao[1] = 1.0;
        Ao[2] = Par[7];
        Gamma = Par[8];
        Alpha = Par[9];

    printf("Values read sucesfully \n");
    fclose (fptr);

        Ac[1] =1.0;
        Ac[2] =Ao[2]+Am[2];
        Ac[3] =Ao[2]*Am[2]+Am[3];
        Ac[4] =Am[3]*Ao[2];
        Am1  = Am[1]+Am[2]+Am[3];
        AmAo1=(Ao[1]+Ao[2])*(Am[1]+Am[2]+Am[3]);

        printf("Initial G,r,s1,s2 are: %f,  %f, %f, %f \n",G,r,s1,s2);
        printf("Am vector: %f, %f, %f \n",Am[1],Am[2],Am[3]);
```

```
        printf("Ac vector: %f, %f, %f, %f \n",Ac[1],Ac[2],Ac[3],Ac[4]);
        printf("Gamma and alpha: %f,%f \n",Gamma,Alpha);

}

void printData(void)
{

        printf("TOGGLE          =       %d  \n", TOGGLE);
        printf("TCURRENT        =       %f  \n", TCURRENT);
        printf("HILIMIT         =       %f  \n",  HILIMIT);
        printf("LOLIMIT         =       %f  \n",  LOLIMIT);
        printf("ACTUAL1         =       %f  \n",  ACTUAL1);
        printf("ACTUAL2         =       %f  \n",  ACTUAL2);
        printf("ACTUAL3         =       %f  \n",  ACTUAL3);
        printf("ACTUAL4         =       %f  \n",  ACTUAL4);
        printf("RHSET           =       %f  \n",  RHSET);


}/*End Function*/
```

APPENDIX B: C-CODE FOR NONLINEAR NEURAL NETWORK CONTROLLER FOR FOXBORO DCS

```
WRITTEN BY MEHMET YUNT FINAL FORM  July 25, 2000                                  */
/*THE FOLLOWING ARE THE FOXBORO NAMES OF THE VARIABLES TO BE READ     */
/*NAME                                VARIABLE              TYPE
        */
/*REHEAT CONTROL TOGGLE             9RH:B27.MA            BOOLEAN      */
/*REHEAT TEMPERATURE                9RH:B4.OUT            FLOAT        */
/*REHEAT TEMPERATURE SETP           9RH:B27.SPT           FLOAT        */
/*REHEAT TILT  HILIMIT               9RH:B27.HOLIM        FLOAT         */
/*REHEAT TILT  LOLIMIT              9RH:B27.LOLIM         FLOAT        */
/*REHEAT ACTUAL ANGLE1              9RH:B28.MEAS          FLOAT         */
/*REHEAT ACTUAL ANGLE2              9RH:B31.MEAS          FLOAT        */
/*REHEAT ACTUAL ANGLE3              9RH:B34.MEAS          FLOAT        */
/*REHEAT ACTUAL ANGLE4              9RH:B37.MEAS          FLOAT        */
/*THE FOXBORO VARIABLE WHERE THE CONTROL OUTPUT WILL BE WRITTEN     */
/*REHEAT TILT ANGLE                 9RH:B27.OUT           FLOAT         */

/*Use uread in main function to write and scopen readval combination in function to read*/
/*Controller will function as long as PI on Reheater is disabled, 9RH:B27.MA=0
        */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define WRITPARAMS  1
#define INPUTS  1
#define OUTPUTS  1
#define READPARAMS  10
#define N_Parameter 9
#define N_temp  3
#define N_inp 2
#define pi  3.1416
typedef union

     {
        long lval;
        float fval;
        } IAXVAL;

void  openReadSet(int *readDset, int readIndex[]);
void  readInputs(int readDset, int readIndex[]);
void  controller(void);
void  adaptation(void);
void  readData(void);
void  printData(void);
void  INITRBF(void);
void  RBF(void);
void  NONLIN(void);
void  NONLINDER(void);

float Par[N_Parameter+1],u[6],uc[6],ul[6],y[6],Set_RHtilt,TCURRENT,TILTEQ,TEQ,MAXTHETA;
float
Am[3],Ac[4],T[2],AmAo1,Gamma,Alpha,G,r,s1,s2,TS,Am1,NORM,Ao[3],ROBUST,RADAPT,Weights[1
001][5],Center[1001][5],SIGMA;
float LOLIMIT, ACTUAL1, ACTUAL2, ACTUAL3, ACTUAL4, RHSET, SETPOINT, HILIMIT,
Z[1001],INPUT[5], OUTPUT[5],NSWITCH, KParam ;
```

```
float NONLINDEROUT, NONLINOUT;
char  TOGGLE,CHECKTOGGLE;
int   CENTERNUM, INPUTDIM, OUTPUTDIM, NDECISION;

main()
{
   int i,reterr,error[1], readIndex[READPARAMS], readDset;
   FILE *fptr;
   static int gw[1]          ={1};
   static int valtyp[1]      ={3};
   static int nument         =1;
   char name[1][32];

   IAXVAL value[WRITPARAMS];


   printf("Program started succesfully \n");
   fflush(stdout);

   printf("Input Equilibrium Temperature \n");
   scanf("%f",&TEQ);

   printf("Input Equilibrium TILT ANGLE \n");
   scanf("%f",&TILTEQ);

   printf("Input Sampling Time \n");
   scanf("%f",&TS);

   printf("Adapt for R -> y=1.0, n=0.0 \n");
   scanf("%f",&RADAPT);

   printf("Apply Robustification y=1.0, n=0.0 \n");
   scanf("%f",&ROBUST);

   printf("Maximum Norm  = \n");
   scanf("%f",&MAXTHETA);

   printf("Turn on the Neural Network? y =1.0, n=0.0 \n");
   scanf("%f",&NSWITCH);

   fptr =fopen("/opt/epri/mit/Param.dat","w");

   /*Open the parameter file*/
   readData();
   printf("The Initial Parameter Estimates are read from the Parameter file \n");
   fflush(stdout);

  /*Initialize the Neural Network  if NSWITCH=1.0*/

  if (NSWITCH == 1.0){

                INITRBF();

                        /*If something is wrong with the Neural network */

printf(" If something is wrong with the neural network, enter 0 and quit program, else enter 1 \n");
```

```
                scanf("%d", &NDECISION);

                if (NDECISION==0){exit ;}

}

        uc[1]=SETPOINT-TEQ;
        uc[2]=SETPOINT-TEQ;
        uc[3]=SETPOINT-TEQ;

/*Open read set*/
openReadSet(&readDset, readIndex);
printf("Set to be read is open \n");


/*Read first data*/
readInputs(readDset, readIndex);
printf("First values are read \n");
printData();

/*The  PastTemp and PastInp vectors must be initialized  */
/*2 minutes of time for switching toggle on*/
/*Actually 2 cycles are enough for initialization*/

   for  (i =1; i<=5; i++){

                readInputs(readDset, readIndex);

   u[6-i] =RHSET-TILTEQ;
   ul[6-i]=u[6-i];
   y[6-i] =TCURRENT-TEQ;
   uc[6-i]=SETPOINT-TEQ;

        sleep(TS);
   }
      printf("Initialization Complete  \n");
      printData();
      fflush(stdout);

CHECKTOGGLE=TOGGLE;

while(CHECKTOGGLE==TOGGLE){
        readInputs(readDset,readIndex);

        /*Writing the current parameters to file*/
        fprintf(fptr,"%f %f %f %f %f \n",G,r,s1,s2,KParam);
        printf("%f %f %f %f %f \n",G,r,s1,s2,KParam);
        fflush(stdout);

        /*Adaptation of parameters*/

        adaptation();

        /*Control input is calculated*/
```

```
            controller();
            printf("Control input  calculated \n");
             fflush(stdout);

            /*Control input is written*/
            strncpy(name[0],"9RH:B27.OUT", 32);
            value[0].fval=Set_RHtilt;
            uwrite(gw, &nument, name, valtyp, value, error, &reterr);
            printf("RHSET = %f  \n",Set_RHtilt);
            fflush(stdout);
            sleep(TS);

     }/*Parentheses of while*/

     fclose(fptr);
     printf("Exiting program PI controller is on \n");
     fflush(stdout);


    /*Closing the read set*/
     clsset(readDset ,&reterr);
     printf("The sets are closed \n");
     fflush(stdout);


    /*End program*/
 printf("Program is shutting down  \n");

return 0 ;
}


/*The adaptation has robustness properties*/
/*Everytime adaptation is called the y value is shifted*/
/*The signals are filtered,if not filtered sinusoidal noise disrupts adaptation*/
void adaptation()

{

  float YF,ubar,Error,dkdr,dkds1,dkds2,dkdG,Co,dTheta[5],Flag, dKParam;


            FILE* fptr;
            Flag =1;
            fptr=fopen("FSignals.dat","a+");
     /* SHIFTING THE Y VECTOR */
       y[5]=y[4];
       y[4]=y[3];
       y[3]=y[2];
       y[2]=y[1];
       y[1]=TCURRENT-TEQ;

       if (fabs(y[1]-y[2])<0.01000) {
           y[1]=y[2] ;
}
```

```
/*Define Point and  call Nonlinder to calculatre NONLINDEROUT*/

if (NSWITCH==1.0) {

        /*Define the INPUT variable*/

        INPUT[1]= y[3]-y[2];
        INPUT[2]= fabs((u[2]+TILTEQ)*pi/180.0);
        INPUT[3]= KParam;

                /*Call The NONLIN FUNCTION TO CALCULATE THE NONLINOUT */

        NONLIN();
        NONLINDER();

                /*Call the Neural network */

        RBF();
        }/*End of Neural part*/


YF   = Ac[1]*y[1]+Ac[2]*y[2]+Ac[3]*y[3]+Ac[4]*y[4]-AmAo1*y[4];
ubar  = ul[2]+r*(ul[3]-u[2])+s1*(y[4]-y[2])+s2*(y[4]-y[3])-G*NONLINOUT+G*YF;

Error = u[1]-ubar;

Flag= 1.0;

  /*Adaptation of the parameters*/

  /*dkdr*/

  dkdr=(ul[3]-ul[2])*RADAPT ;

 /*dkds1*/

 dkds1=(y[4]-y[2]);

/*dkds2*/

 dkds2=(y[4]-y[3]);

/*dkdG*/

dkdG =YF;

if ((y[1]-uc[1])*(y[1]-uc[1])<0.5){Flag=0;}

 fprintf(fptr,"%f %f %f %f %f %f %f %f %f
\n",YF,ubar,Error,dkdr,dkds1,dkds2,dkdG,NONLIN,NONLINDER);

  fclose(fptr);
```

```
Co=RADAPT *dkdr*dkdr+dkds1*dkds1+dkds2*dkds2+(1-NSWITCH)*dkdG*dkdG+
NSWITCH*NONLINDEROUT*G*G*NONLINDEROUT;
 dTheta[1] = Gamma*dkdr/(Alpha+ Co)*Error;
 dTheta[2] = Gamma*dkds1/(Alpha + Co)*Error;
 dTheta[3] = Gamma*dkds2/(Alpha + Co)*Error;
 dTheta[4] = Gamma*dkdG/(Alpha + Co)*Error;

 dKParam   = 0.001*NSWITCH*OUTPUT[1]/(Alpha + Co)*Error;

 /*The Robustness Criteria will be implemented here after the whereabouts of the true parameters are
known*/

/*Modifying Control Parameters*/

     r =r+dTheta[1]*RADAPT*Flag;
     s1=s1+dTheta[2]*Flag;
     s2=s2+dTheta[3]*Flag;
     G =G+(1-NSWITCH)*dTheta[4]*Flag;
     KParam = KParam+NSWITCH*dKParam;

     if (KParam>0.8){
                                   KParam = 0.8;
     }

     if (KParam<0.0) {
                                   KParam = 0.0;
     }


/*Robustness*/


          if(ROBUST>0.0){
          NORM = G*G+s1*s1+s2*s2+RADAPT*r*r ;
          NORM = sqrt(NORM);

          if (NORM>MAXTHETA){
                    G =G/NORM*MAXTHETA;
                    r =r/NORM*MAXTHETA;
                    s1=s1/NORM*MAXTHETA;
                    s2=s2/NORM*MAXTHETA;
            }
          }

}/*End Adaptation*/

/*This function calculates the Reheater Tilt Setpoint*/
/*The parameter vector is global                */



void  controller()


{
```

```
/*shift the control input vector*/

  float K;


  /*shift the reference input*/
       uc[2]=uc[1];
       uc[1]=SETPOINT-TEQ;

      u[5]=u[4];
      u[4]=u[3];
      u[3]=u[2];
      u[2]=u[1];
      u[1]=Set_RHtilt-TILTEQ;


      ul[5]=ul[4];
      ul[4]=ul[3];
      ul[3]=ul[2];
      ul[2]=ul[1];

 K = Am1*(Ao[1]*uc[1]+Ao[2]*uc[2])-AmAo1*y[3];
 ul[1]=ul[2]+r*(ul[3]-ul[2])+s1*(y[3]-y[1])+s2*(y[3]-y[2]);


printf("the u ->1 to 5 vector: %f,%f,%f,%f,%f \n",u[1],u[2],u[3],u[4],u[5]);
printf("the ul->1 to 5 vector: %f,%f,%f,%f,%f \n",ul[1],ul[2],ul[3],ul[4],ul[5]);
printf("the y ->1 to 5 vector: %f,%f,%f,%f,%f \n",y[1],y[2],y[3],y[4],y[5]);


 if (NSWITCH==1){

                              /*Define the INPUT variable*/
                              INPUT[1]= y[2]-y[1];
                              INPUT[2]= fabs((u[1]+TILTEQ)*pi/180);
                              INPUT[3]= KParam;


  /*Call The NONLIN FUNCTION TO CALCULATE THE NONLINOUT */

 NONLIN();

  }

         Set_RHtilt = ul[1]-NSWITCH*G*NONLINOUT+G*K;

         /*Set_RHtilt=u[1]+r*(u[2]-u[1])+s1*(y[3]-y[1])+s2*(y[3]-y[2])+G*K */
  if (fabs(Set_RHtilt-u[1])<0.300){Set_RHtilt=u[1];}


         Set_RHtilt=Set_RHtilt+TILTEQ;

         printf("The Calculated Control Input = %f \n",Set_RHtilt);

         /*Check for saturation of input*/
```

```
        if (Set_RHtilt>=HILIMIT) {

                Set_RHtilt = HILIMIT;
                        }

        if (Set_RHtilt<=LOLIMIT)
        {
                Set_RHtilt = LOLIMIT;
                }
    printf("The Saturated Control Input = %f \n",Set_RHtilt);


}/*End of controller*/

        void openReadSet(int *readDset, int index[])

{
        int i;
        int gw[10]          =       {1,1,1,1,1,1,1,1,1,1 };
        int valtyp[10]      =       {5,3,3,3,3,3,3,3,3,3};
        float rdelta[10]    =       {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.1,0.1,0.1};
        float wdelta[10]    =       {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.1, 0.1,0.1};
        int nument          =       10;
        int rsr             =       1;
        int wsr             =       1;
        int clexit          =        1;
        int acctyp          =       1;
        char name[READPARAMS][32];
        int reterr, error[READPARAMS];
        strncpy(name[0],"9RH:B27.MA", 32);
        strncpy(name[1],"9RH:B4.OUT" , 32);
        strncpy(name[2],"9RH:B27.SPT" , 32);
        strncpy(name[3],"9RH:B27.HOLIM", 32);
        strncpy(name[4],"9RH:B27.LOLIM", 32);
        strncpy(name[5],"9RH:B28.MEAS", 32);
        strncpy(name[6],"9RH:B31.MEAS", 32);
        strncpy(name[7],"9RH:B34.MEAS", 32);
        strncpy(name[8],"9RH:B37.MEAS", 32);
        strncpy(name[9],"9RH:B27.OUT", 32);

        printf("The name variable is created \n");
    fflush(stdout);

        scopen(gw,nument,name,valtyp,acctyp,rdelta,clexit,rsr,wsr,wdelta,readDset,index,error,&reterr);

    printf("Scopen Succesful \n");

        if ((*readDset)==0) {

                printf("Cannot open set correctly \n");
                printf("reterr = %d\n",reterr);
                for(i=0;i<READPARAMS ;i++)
                        {
                 printf ("%-32s %-04x\n", name[i],error[i]    );
                        }
            exit(0);
```

```
        }
}/*end of function*/


void readInputs(int set, int index[])
{
        typedef union

         {
                long lval;
                float fval;
                char bval;
        } MIAXVAL;

        int i;
        MIAXVAL  value[READPARAMS];
        int status,type;

        if (gsnent(set)!=READPARAMS)

        {
         printf("Read from the Wrong Set  \n");
         exit(1);
        }

        for(i=1; i<READPARAMS;i++)/*1 Read the float values*/
        {
         value[i].lval=readval(index[i]);
         status =readsta(index[i]);
         type=status &  7;

         if(type != 3)
                {
                        printf("the type of %d th read parameter is not float, but type %d!\n",i,type);
                }  /*1*/
        }

     /*Read the Boolean Toggle Variable*/

  value[0].bval=readval(index[0]);
        status =readsta(index[0]);
        type=status & 7;

        if(type != 5)

         {
          printf("the type of %d th read parameter is not Boolean, but type %d!\n",i,type);
     }


TOGGLE          =       value[0].bval;
TCURRENT        =       value[1].fval;
SETPOINT        =       value[2].fval;
HILIMIT         =       value[3].fval;
LOLIMIT         =       value[4].fval;
ACTUAL1         =       value[5].fval;
```

```
ACTUAL2     =     value[6].fval;
ACTUAL3     =     value[7].fval;
ACTUAL4     =     value[8].fval;
RHSET       =     value[9].fval;

}/*End function*/

/*The Mparameter.dat File contains the following                */
/*Initial estimates for the four parameters->4 parameters       */
/*G =gain or 1/bo                                               */
/*r =feedback denominator coefficient                           */
/*s1=feedback numerator coefficient                             */
/*s2=feedback numerator coefficient                             */
/*The coefficients for the desired following transfer functions */
/*Am=third order monic polynomial ->2 coefficients              */
/*Ao=first order monic polynomial ->1 coefficient               */
/*T =first order     polynomial ->0 coefficient                 */
/*T contains the desired gain as well(Am'(1))                   */
/*Gamma =adaption gain            ->1 parameter                 */
/*alpha =adaptation constant      ->1 parameter                  */

void  readData(void )
{
        int i;

        FILE *fptr ;

        fptr = fopen("/opt/epri/mit/Mparameter.dat", "r");
        printf(" File Pointer assigned succesfully \n");
        fflush(stdout);

    /*PAR is of length 10 but the zeroth element is not used*/
        for(i=1; i<=N_Parameter; i++)  fscanf(fptr,"%f ", &Par[i]);

G =Par[1];
        r =Par[2];
        s1=Par[3];
        s2=Par[4];
         Am[1] = 1.0;
        Am[2] = Par[5];
        Am[3] = Par[6];
        Ao[1] = 1.0;
        Ao[2] = Par[7];
        Gamma = Par[8];
         Alpha = Par[9];

    printf("Values read sucesfully \n");
    fclose (fptr);

        Ac[1] =1.0;
        Ac[2] =Ao[2]+Am[2];
        Ac[3] =Ao[2]*Am[2]+Am[3];
        Ac[4] =Am[3]*Ao[2];
        Am1  = Am[1]+Am[2]+Am[3];
        AmAo1=(Ao[1]+Ao[2])*(Am[1]+Am[2]+Am[3]);
```

```
        printf("Initial G,r,s1,s2 are: %f,  %f, %f, %f \n",G,r,s1,s2);
        printf("Am vector: %f, %f, %f \n",Am[1],Am[2],Am[3]);
        printf("Ac vector: %f, %f, %f, %f \n",Ac[1],Ac[2],Ac[3],Ac[4]);
        printf("Gamma and alpha: %f,%f \n",Gamma,Alpha);

}

void printData(void)
{

        printf("TOGGLE          =       %d  \n", TOGGLE);
        printf("TCURRENT        =       %f  \n", TCURRENT);
        printf("HILIMIT         =       %f  \n", HILIMIT);
        printf("LOLIMIT         =       %f  \n", LOLIMIT);
        printf("ACTUAL1         =       %f  \n", ACTUAL1);
        printf("ACTUAL2         =       %f  \n", ACTUAL2);
        printf("ACTUAL3         =       %f  \n", ACTUAL3);
        printf("ACTUAL4         =       %f  \n", ACTUAL4);
        printf("RHSET           =       %f  \n", RHSET);


}/*End Function*/


void INITRBF(void)

{
        int i,j,NPAR[4];

        FILE *fptr ;

        fptr = fopen("/opt/epri/mit/Neural.dat", "r");
        printf("File Pointer assigned succesfully \n");
        fflush(stdout);

  /*PAR is of length 10 but the zeroth element is not used*/
        for(i=1; i<=3; i++) fscanf(fptr, "%d\n",&NPAR[i]);


  INPUTDIM =NPAR[1];
  OUTPUTDIM=NPAR[2];
  CENTERNUM=NPAR[3];

printf("Values read sucesfully \n");
 printf("Inputdim, outputdim,centernum: %d, %d, %d \n", INPUTDIM, OUTPUTDIM, CENTERNUM);

fclose (fptr);

        fptr = fopen("/opt/epri/mit/KParameter.dat","r");
                fscanf(fptr,"%f\n", &KParam);
                fscanf(fptr,"%f\n", &SIGMA) ;
        fclose(fptr);

 printf("KParam, SIGMA : %f, %f \n ", KParam, SIGMA);

fptr =fopen("/opt/epri/mit/Weights.dat","r");
```

```
        for(i=1; i<=CENTERNUM; i++){
        for (j=1;j<=OUTPUTDIM; j++) fscanf(fptr,"%f \n",&Weights[i][j]);}

printf("Weights are read \n");
printf("Weights[1][1], Weights[2][1],Weights[3][1]: %f, %f, %f  \n", Weights[1][1],
Weights[2][1],Weights[3][1]);


        fclose(fptr);

        fptr = fopen("/opt/epri/mit/Centers.dat","r");

        for(i=1; i<=CENTERNUM; i++){
          for (j=1;j<=INPUTDIM; j++) fscanf(fptr,"%f ",&Center[i][j]);}
    printf("Centers are read \n");
    printf("Center[1][1], Center[2][1], Center[3][1], Center[4][1], Center[5][1]: %f, %f, %f, %f, %f  \n",
Center[1][1], Center[2][1],Center[3][1], Center[4][1], Center[5][1]);
  fclose(fptr);

 }


void RBF()

{  int i,j;

   for(i = 1; i<=CENTERNUM; i++){

                Z[i] = 0.00;

                for(j =1; j <=INPUTDIM; j++){
                        Z[i]= Z[i]+(INPUT[j]-Center[i][j])*(INPUT[j]-Center[i][j]);
                        }

        Z[i] =-Z[i]/SIGMA;
        Z[i] =exp(Z[i]);
   }

for(j = 1; j <=OUTPUTDIM; j++){
                        OUTPUT[j] = 0.0;

                for(i = 1; i<=CENTERNUM; i++){

      OUTPUT[j] = OUTPUT[j]+Z[i]*Weights[i][j];

                        }
   }

}



void NONLIN()

{
```

```
NONLINOUT =tan(KParam*INPUT[1])/(tan(KParam*(INPUT[2]))+0.5);
}



void NONLINDER()

{

float a,b,c,d,KParam0;
KParam0=0.8;
a =(tan(KParam0*INPUT[1]))*(tan(KParam0*INPUT[1]));
b =(tan(KParam0*(INPUT[2]))+0.5)*(tan(KParam0*(INPUT[2]))+0.5);
c =(tan(KParam0*INPUT[2]))*(tan(KParam0*INPUT[2]));

NONLINDEROUT = (1+a)*(INPUT[1])/(tan(KParam0*(INPUT[2]))+0.5)-
tan(KParam0*INPUT[1])/b*(1+c)*INPUT[2];

}
```