SINGLE INPUT SINGLE OUTPUT ADAPTIVE CONTROL FOR LABORATORY-BASED SYNCHRONOUS MACHINE

BY

IGNATIUS PRIJOWIBOWO

This report is submitted as fulfilment of the requirements
for the degree of master of technology

in

Department of Electrical and Electronic Engineering
University of Tasmania
Australia

1994

Supervisor
Mr. G. The'

DECLARATION

To the best of my knowledge, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

IGNATIUS PRIJOWIBOWO

ACKNOWLEDGMENTS

Thank you to my supervisor Mr Gregory The' for his time, assistance and encouragement in the development of this thesis. I would like to thank to Mr. B. Wherrett for his assistance in power laboratory. Thank you also to Mrs. Margaret Eldridge for proof reading.

ABSTRACT

The idea of designing a control system that can adjust its own structure and parameters to cope with a specified purpose, was very appealing. This thesis investigates the application of adaptive control using pole-zero placement method to control the Automatic Voltage Controller of a Synchronous generator. In this investigation the recursive least squares with exponential forgetting factor was used to identify the plant parameters.

CONTENTS

CHAPTER 1

INTRODUCTION

Conventional analog automatic voltage regulators (AVR) were designed using classical control theory which employed linear transfer function models. This designed theory was based on linearised models which were only valid at a certain set of operating conditions. A synchronous generator is operate on non linear and time-variant systems, with a wide range of variation of system parameters and unmodelled disturbances. Classical control unable to respond satisfactorily over the whole range of system and operating conditions. In synchronous generator systems, the variation of set points will not change the system parameters, so that the same type of controller can be used. However, the variation of external parameters such as transmission line impedances and equivalent-load impedances and the other machines behaviour impose stringent requirement on self-tuning regulators.

This report is structured as follows:
Chapter 2 presents an overview of adaptive control system design methods and some basic techniques of adaptive controller with their problems and advantages. The self-tuning controller contain a parameter estimator which characterise the process based on its input and output. The parameter estimations using recursive least squares with exponential forgetting factors are investigated in Chapter 3. This chapter investigates the parameter estimation using some exponential forgetting factors and different P matrix.

To validate whether the synchronous generator is controllable or not, a simulation of a nonlinear synchronous machine is solved by using fourth order Runge-Kuta. This simulation is discussed in Chapter 4. Based on the simulation result, the pole-zero adaptive controller and predictive controllers are also presented in this chapter.

The controllers that are discussed in Chapter 4 are implemented in controlling the exciter of a 7.5 KVA synchronous generator in laboratory by a personal computer. The real time calculations are monitored from screen by using Quinn Curtis software and data acquisitions is obtained from Boston technology. These aspects are discussed in Chapter 5.

Chapter 6 provides the conclusion of the preceeding chapters. The simulation and programmming are shown in appendices.

CHAPTER 2

ADAPTIVE CONTROL

2.1 General

The intended achievements of control theory in controlling a dynimical system are maintaining outputs of a system around prescribed constant values and ensuring that the overall system optimises a specified performance criterion. To achieve these goals, computation of suitable control input based on the observed outputs of the system must be conducted. The basic processes incorporated in controlling a system include the mathematical modelling of the system, identification of the system based on experimental data, developing convenient mathematical forms from the outputs of the system to synthesise the control inputs and apply them to the system to obtain the intended behaviour.

The development of the control theory can be classified into three main categories: deterministic control theory, stochastic control theory and adaptive control theory.

In deterministic control theory, it was assumed that the system was linear time invariant with complete knowledge of the controlled systems. This theory was used succsessfully for feedback control systems.

Stochastic control theory was concerned with uncertainties that were inherent in the control systems. To cope with the stochastic conditions, linear deterministic control theory was extended. However, these theories need sufficient a priori knowledge of the systems and their environment. The meaning of the a priori knowledge is the information of all physical systems can be included in order to reduce the number of the model parameters to be estimated. In real operation, especially while dealing with complex dynamic systems operating in a complex environment, there are some uncertain situations where a complete a priori of the

systems cannot be provided. The difficulties were observed when substantial amounts of uncertainty were present in the systems. So, the demands of faster and more accurate controllers became clearly evident when the existing theory were inadequate to successfully handle the problems. An adaptive control theory with a capability to adjust its performance and environment changes was desired.
The term "adaptive system" was introduced into control theory to represent control systems that monitor their own performance and adjust their control mechanism in the direction of improved performance. The most important feature of adaptive control is its ability to adjust itself to predetermined ranges of set points in various dynamic processes.

2.2 Adaptive Control Designs

In controlling a process which is nonlinear, time-varying and has unknown dynamics with unknown disturbances acting upon it, needs self adaptive control algorithms that have some learning capabilities. So far, there is no general analytical solution which has been found to solve such complex problems. A possible approach to the solution of these problems is to accumulate dynamically all information about the system response and to simultanously generate an acceptable control signal in an adaptive feedback manner.

The operating quality of the adaptive control can be deduced from how efficient and quick the ability of the adaptive system is to generate control signals to optimise the performance of the dynamic process.

There are two popular approaches to designing adaptive control systems[1] They are the theory of Model Reference Adaptive Systems (MRAS) and the theory of Self Tuning Regulator (STR).

2.2.1 Model Reference Adaptive Systems (MRAS)

Model Reference Adaptive Systems (figure 2.1), originally were developed by Whitaker, Yamron and Kezer (1958) to solve the servo problem. In MRAS, the adaptive controller forces the plant to perform like a reference model, where the model represents the performance of a desired system. MRAS have fairly high speed adaptation where the identification for dynamic plant performance is not required.

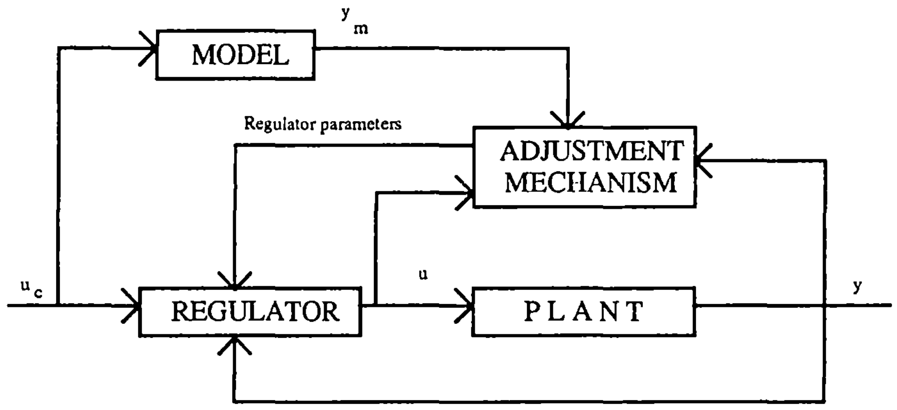


Figure 2.1 Block diagram of Model Reference Adaptive Systems

The task of the adjustment mechanism in the block diagram, is to minimise the error between the plant output y and model output y_m. The minimum error is then used by the adjustment mechanism to modify the regulator parameters. The problem is to determine the adjustment mechanism so that it not only brings the error to zero but also produces a stable result. This is a difficult problem to solve, because simple linear feedback from the error to the controller parameters is unable to guarantee a stable result [2].

There are two principle approaches of MRAS to consider the estimation of the unknown plant[1] i.e. direct and indirect control.

1. Indirect Control

The parameter estimation of the unknown plant, is derived from its input and output. The estimated parameters are used to generate a feedback control function to adjust the parameters of the controller.

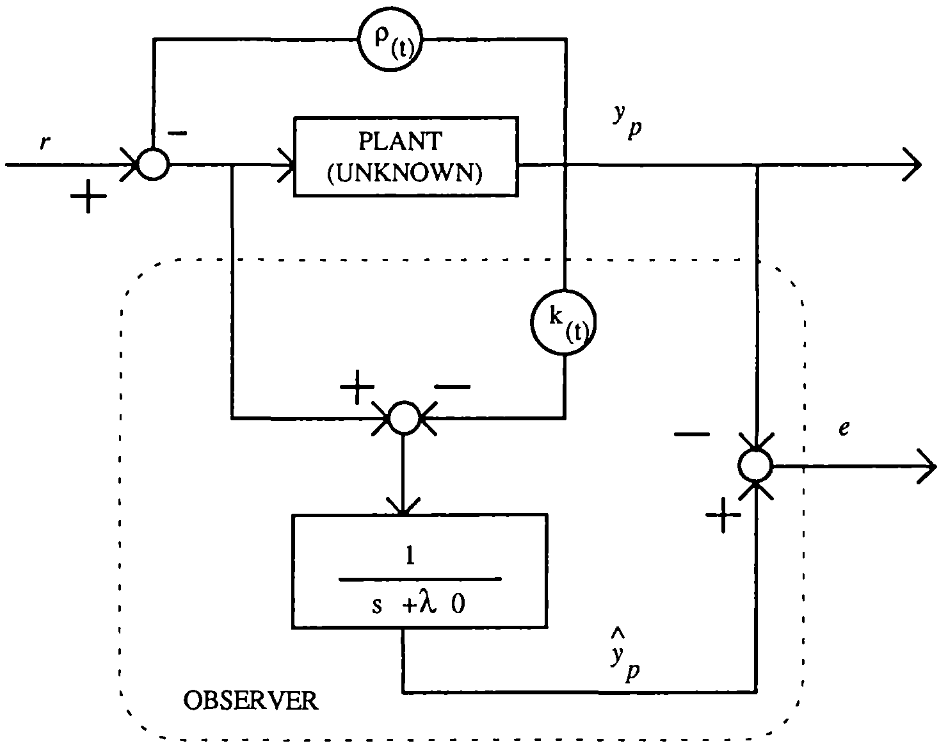


Figure 2.2 Indirect control of MRAS

2. Direct Control

This approach does not have an explicit plant identification. The controller parameters are updated from the control error.

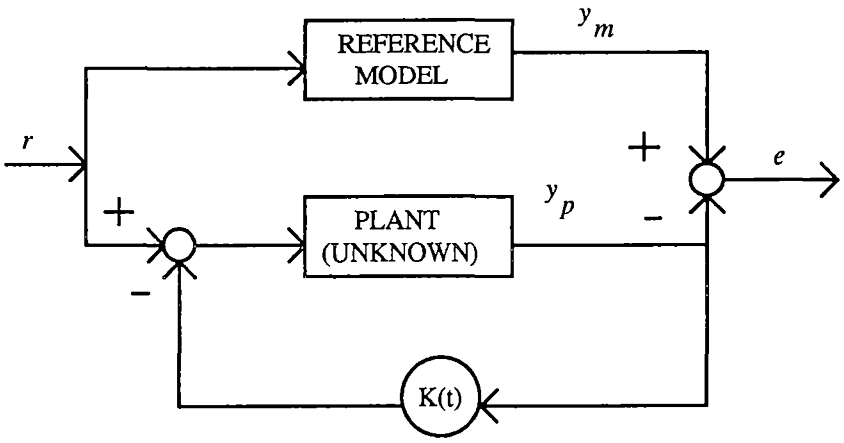


Figure 2.3 Direct control of MRAS

The main differences between direct and indirect MRAS are as follows:

Model of the desired control is explicitly used in direct control, whereas a model of the plant identified on-line is used in indirect control.

Control error in direct control and identification error in indirect control are used to update the controllers.

2.2.2 Self-tuning Regulators (STR)

Self-tuning regulators play a significant role in adaptive controllers. They are relatively easy to implement in microprocessors, and are applicable to complex processes with dynamic characteristics and stochastic disturbances.

The self-tuning regulator was originally introduced by Kalman in 1958, however, due to unavailability of sophisticated computers and the lack of theory to support it, it was not well developed until Astrom and Wittenmark developed the STR for the stochastic minimum variance control in 1973. The STR consists of three major parts, a parameter estimator, a controller calculation and a controller with adjustable parameters. The parameter estimator identifies the parameters of the plant from its input and output. The controller design computes the parameter of the controller base on plant parameters. The controller gives input signals to the plant from set point *w* and controller design.

From the identification of the controller parameters algorithm, the STR can be classified in two ways: Indirect and Direct Self-tuner algorithm.

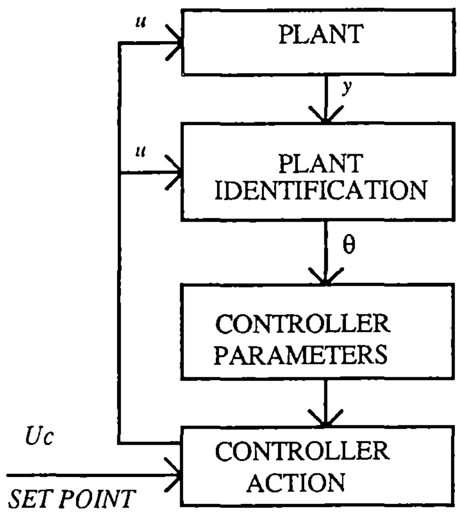


Figure 2.2 Block diagram of Indirect STR

1. Indirect Self-Tuner Algorithm

With Indirect Self-Tuner algorithm, the controller parameters are updated after identification of the plant parameters and controller design. The Indirect Self-Tuner is achieved by an iteration at each sample interval through the following cycles:

Step 1. Identification of the plant parameters at each sampling by Recursive Least Square.

Step 2. Calculating the design controller parameters.

Step 3. Updating the control signal parameters by control law.

Steps 1 to 3 are repeated at each sampling period. After several iterations, controller parameters converge when the estimated plant parameters reach a steady value.

To obtain good estimates and control, it may be noted that the input of the plant must be constantly excited or rich in frequencies. If there are no changes for a long time, the gain of the parameter estimator may become very large, and a change in the command signal may produce large changes in the parameter estimates θ and in the process output y[3]. This is usually the case with fixed parameter controllers in industrial processes.

2. Direct Self-Tuner Algorithm

The Direct Self-Tuner algorithm (figure 2.3) updates the controller parameters directly from the parameters of the model. As a result the design controller that calculates the solution of the Diophantine equations can be eliminated.

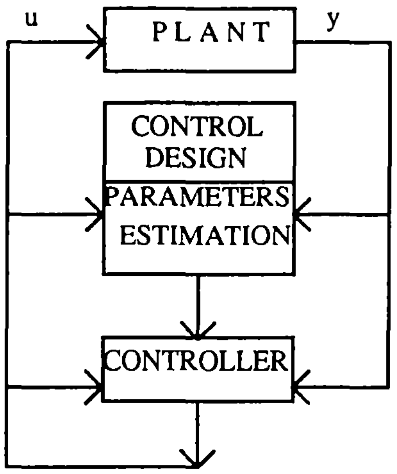


Figure 2.3 Block diagram of Direct Self-Tuner

The direct STR consists of two steps of operations:

Step 1. Estimation of the plant parameters.

Step 2. Calculation of the control signals.

Step 1 and 2 are repeated each sampling period

The operational steps in direct STR show the elimination of the second step of indirect STR. This elimination can be achieved by selecting proper model structure. Direct STR algorithm is valid only for minimum-phase plants [1].

2.2.3 Gain Scheduling

Another design for parameter adaptive control is Gain scheduling (figure 2.4.)

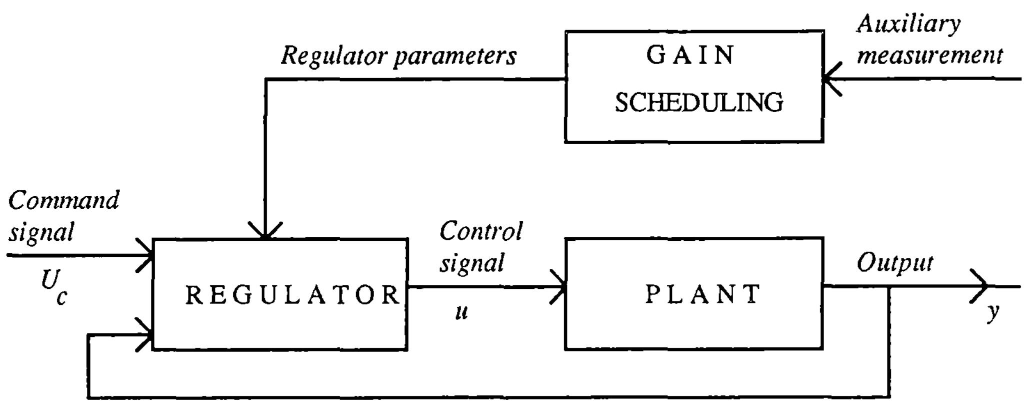


Figure 2.4 Block diagram of Gain scheduling

The objective of Gain scheduling is to reduce the influence of parameter variations by changing the regulator parameters as a function of auxiliary variables. The regulator parameters are determined by using some suitable design methods at number of operating conditions after scheduling variables are found. The main problem of Gain scheduling is to obtain proper scheduling variables, which are usually based on the physical knowledge of a system. The advantages of Gain scheduling are that the regulator parameters can be changed quickly in response to process changes and the effects of parameter variations are minimised. These advantages can be attained by selection of auxiliary variables that correlate well with the changes in process dynamic.

The disadvantages of this approach are :

1. There is no feedback to compensate the error of Gain schedule.
2. Evaluation of the performance and stability of the system must be checked by simulations.
3. In some cases several operating conditions must be simulated to determine regulator parameters.

CHAPTER 3

SYSTEM IDENTIFICATION

3.1 General

Identification of system parameters is one basic step in adaptive control algorithms. There are many parameter identification methods which can be used to identify system parameters. Least squares is one particular method which can be used for parameter identification. Section 3.2 discusses the least squares method based on the linear process model.

In order to obtain on-line parameter identification, the recursive parameter identification method is the best suited method[4]. Section 3.3 discusses recursive least squares.

The algorithm of controller procedure recommends process models in different equations as these equations are easy to implement in digital computers.

3.2 Recursive Least Squares

A discrete time process model can be written in a general *Z* transfer function where the polynomial numerator's degree is one degree less than the denominator. Equation 3.1 shows the general transfer function as

$$\frac{Y(z)}{U(z)} = G(z) = \frac{b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}} \tag{3.1}$$

where the plant input and output are denoted by *Y(z)* and *U(z)* respectively, and the model parameters are b_n and a_n.

To identify parameters in a dynamic process system, least squares needs a large amount of plant input and output to be provided prior to parameters identification.

In high order systems, provision of plant input/output is time consuming and inefficient.

To cope with this problem, the recursive least squares method is applied. This method needs relatively less plant input/output than least square methods.

For k times measurement, the inverse transform of equation 3.1 can be written as

$$y_k = b_1 u_{k-1} + b_2 u_{k-2} + \ldots + b_n u_{k-n} - a_1 y_{k-1} - a_2 y_{k-2} - \ldots - a_n y_{k-n} \qquad (3.2)$$

Equation 3.2 is a recursive equation that can be used to identify model parameters and obtain the next model's output based on the previous input and output.

The turbo generator can be modelled as a second order difference equation

$$y_k = b_1 u_{k-1} + b_2 u_{k-2} - a_1 y_{k-1} - a_2 y_{k-2} \qquad (3.3)$$

and depicted in figure 3.1

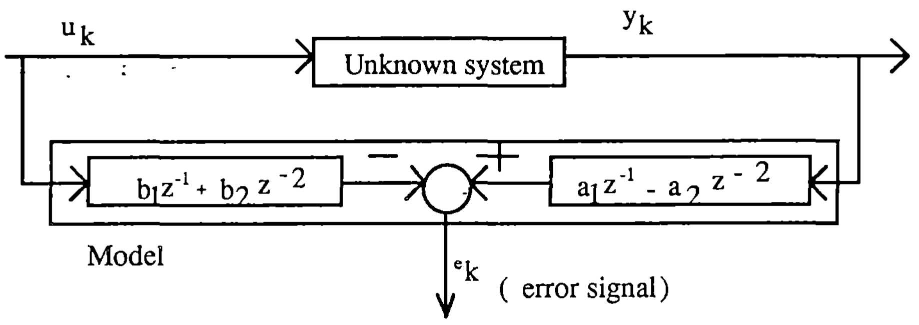


Figure 3.1 Unknown parameter identifications

The error signal e_k is given by

$$e_k = y_k - b_1 u_{k-1} - b_2 u_{k-2} + a_1 y_{k-1} + a_2 y_{k-2}$$

or

$$y_k = b_1 u_{k-1} + b_2 u_{k-2} - a_1 y_{k-1} - a_2 y_{k-2} + e_k \qquad (3.4)$$

In a matrix form, equation 3.4 can be represented as

$$Y_k = \boldsymbol{H}_k \theta_k + e_k \qquad (3.5)$$

where known function $\boldsymbol{H}_k$ can be written as

$$\boldsymbol{H}_k = [\, u_{k-1} \quad u_{k-2} \quad y_{k-1} \quad y_k \,]$$

and the unknown vector θ_k

$$\theta_k^T = [\, b_1 \quad b_2 \quad -a_1 \quad -a_2 \,]$$

and the error vector e_k

$$e_k^T = [\, e_1 \quad e_2 \quad e_3 \quad e_4]$$

The error vector e_k becomes zero when the loss function J

$$J = \frac{1}{2}\sum_{i=1}^{N} e_k^2 \tag{3.6}$$

is minimal. This minimisation can be achieved because matrix

$$H_k^T H_k$$

is non singular and the input signals are persistently excited or sufficiently rich. Now, the least square estimate δ is given by

$$\hat{\theta}_k = [\, H_k^T H_k]^{-1} H_k^T y_k \text{ [6]} \tag{3.7}$$

By introducing a quantity $P_k = [\, H_k^T Hk \,]^{-1}$ then

$$\hat{\theta}_k = P_k H_k^T y_k \tag{3.8}$$

For additional measurement k+1 then

$$\hat{\theta}_{k+1} = \hat{\theta}_k + k_{k+1}(y_{k+1} H_{k+1}\hat{\theta}_k) \tag{3.9}$$

and

$$P_{k+1} = (P_k - k_{k+1} H_{k+1} P_k) \tag{3.10}$$

where

$$k_{k+1} = P_k H_{k+1}^T (1 + H_{k+1} P_k H_{k+1}^T)^{-1} \tag{3.11}$$

3.3 RLS identification with an Exponential Forgetting Factor

The Recursive least squares method as developed by Astrom and Wittenmark was designed to identify the parameters of a process model which operated in time invariant. In a system such as the turbogenerator, the system is time varying and has non linear process control. Hence, the plant parameters may never converge when the recursive least squares method is applied. This problem can be solved by introducing a weighting factor namely exponential forgetting factor (δ). This factor discards old data to speed up the identified plant parameters convergence into a steady state value.

According to Astrom and Wittenmark[6] the equation 3.9 to 3.11 become

$$\hat{\theta}_{k+1} = \hat{\theta}_k + k_{k+1}(y_{k+1} - H_{k+1}\hat{\theta}_k) \qquad (3.12)$$

$$P_{k+1} = (P_k - k_{k+1}H_{k+1}P_k)/\delta \qquad (3.13)$$

$$k_{k+1} = P_k H^T_{k+1}(d + H_{k+1}P_k H^T_{k+1})^{-1} \qquad (3.13)$$

The value of the exponential forgetting factor (δ) is defined as in the range between 0 to 1. If δ is small, the recent data will have more weight than the old data.

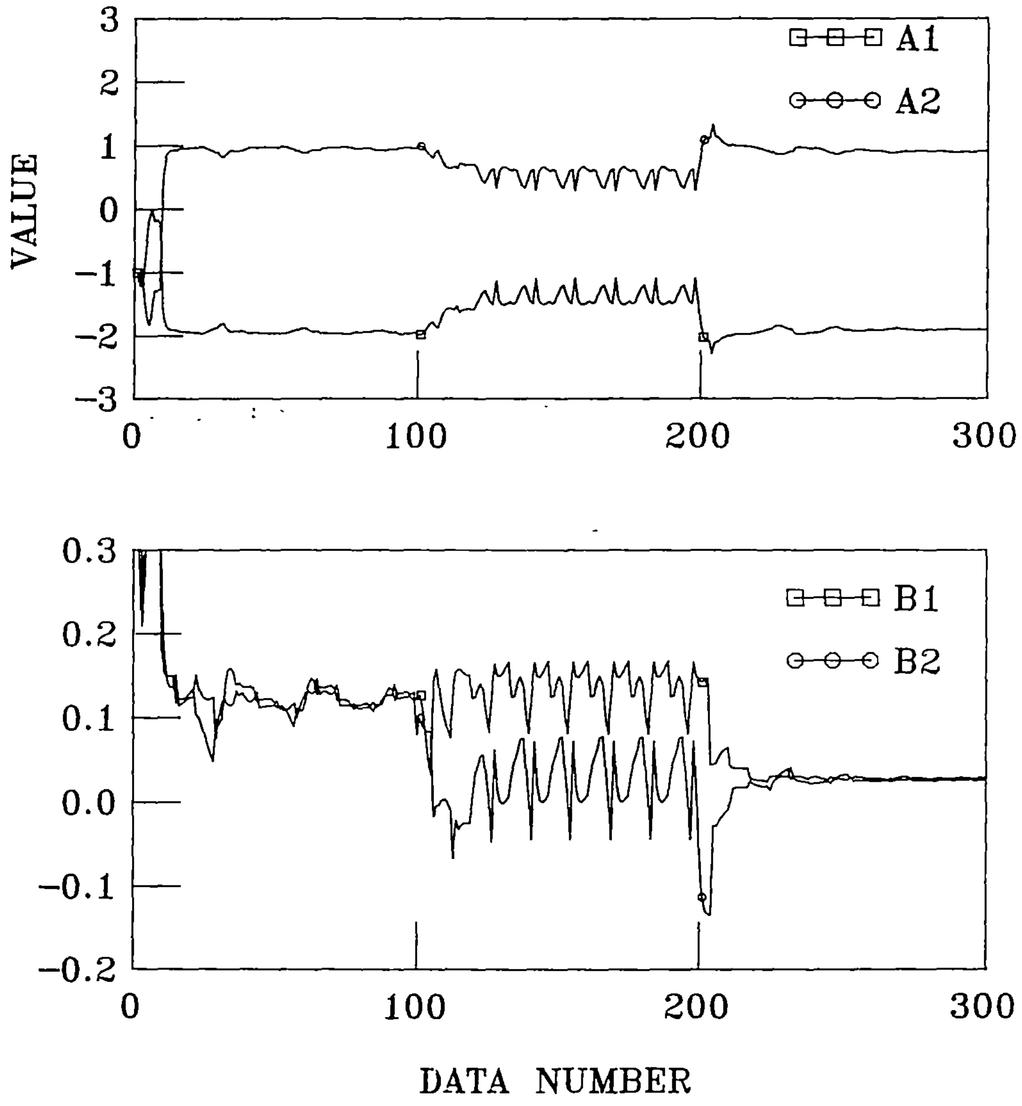


Figure 3.2 Parameter identifications with δ =0.8 and Pmatrix = 10

To investigate the effect of different δ and P matrix, the following conditions are applied:

To investigate the effect of different δ and P matrix, the following conditions are applied:

- The initial value of $\theta = [\,1\ 1\ 1\ 1]$.
- Number of data = 300.
- Plant parameters change at data number 100 and 200 as shown in table 3.1

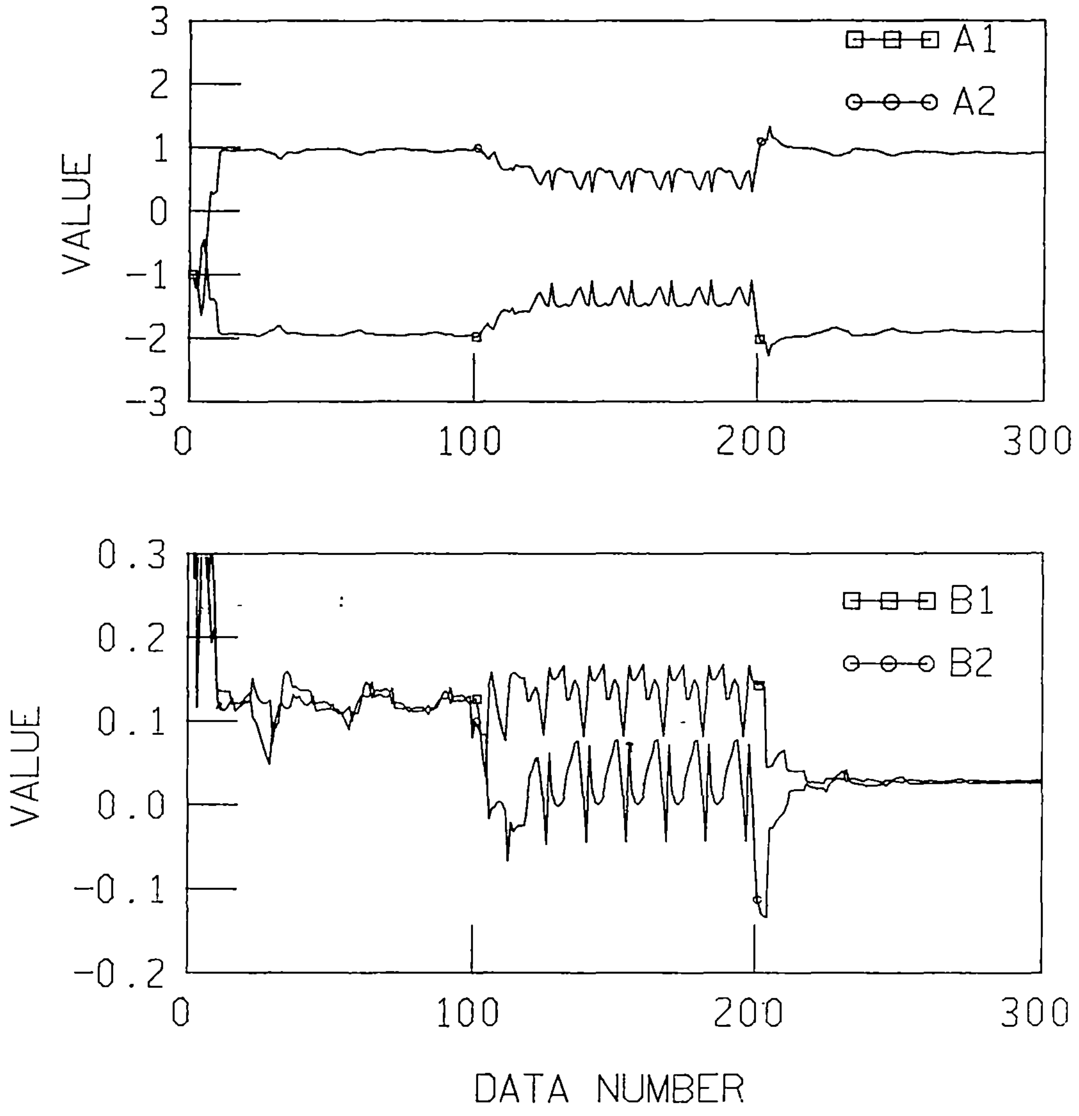


Figure 3.3 Parameter identification using RLS with $\delta = 0.8$ and P matrix = 100

Parameters	Iteration 1 to 100	Iteration 101 to 200	Iteration 201 to 300
a_1	- 1.9213	-1.2348	- 1.88762
a_2	0.9607	0.4493	0.91289
b_1	0.09931	0.1215	0.02515
b_2	0.09798	0.0929	0.02254

Table 3.1. Table of plant parameter changes at iteration 100 and 200.

Figure 3.2 shows second order parameter identifications with δ value 0.8, and elements of diagonal P matrix = 10. It can be seen the identified parameters slow to converge to the a steady state value and have noisy parameters but fast enough to track the parameter variation when the plant parameters change.

With δ value = 0.8 and P matrix = 100, figure 3.3 shows the identified parameters converge faster than the experiment result with P matrix =10, but the identified parameters are noisy.

In figure 3.4, the δ value = 0.95 and P matrix = 100 are applied. The result of these values shows the identified parameters are less noisy, but the response is slower than δ value = 0.8. Empirically, δ equal to 0.98 is usually used and shown in Figure 3.5. The result shows the parameter's noise is reduced and the response is faster than δ equal to 0.99 (see Figure 3.6) to track the parameter change. This result uses similar starting value θ^T and elements of diagonal P matrix.

If the parameters of the process model stay constant for a long time, the exponent forgetting factor δ does not work well. As δ is less than 1, the estimator will then discount old data even though there is no new parameter change in the recent data. This condition can be explained as follows. When there is no new parameter change then $k_{k+1}H_{k+1}P_k$ of equation 3.13 becomes zero [7] so

$$P_{k+1} = P_k/\delta \tag{3.14}$$

As a result, the value of P_{k+1} will grow exponentially and become too large to be handled by computer. Since P_k is so large, any change in the plant parameters may then lead to large changes in the parameter identification, thus causing an inaccurate identification of the system or estimator wind-up.

To control this problem, Hagglund[7] proposed algorithms which only discount data where there new information exists.

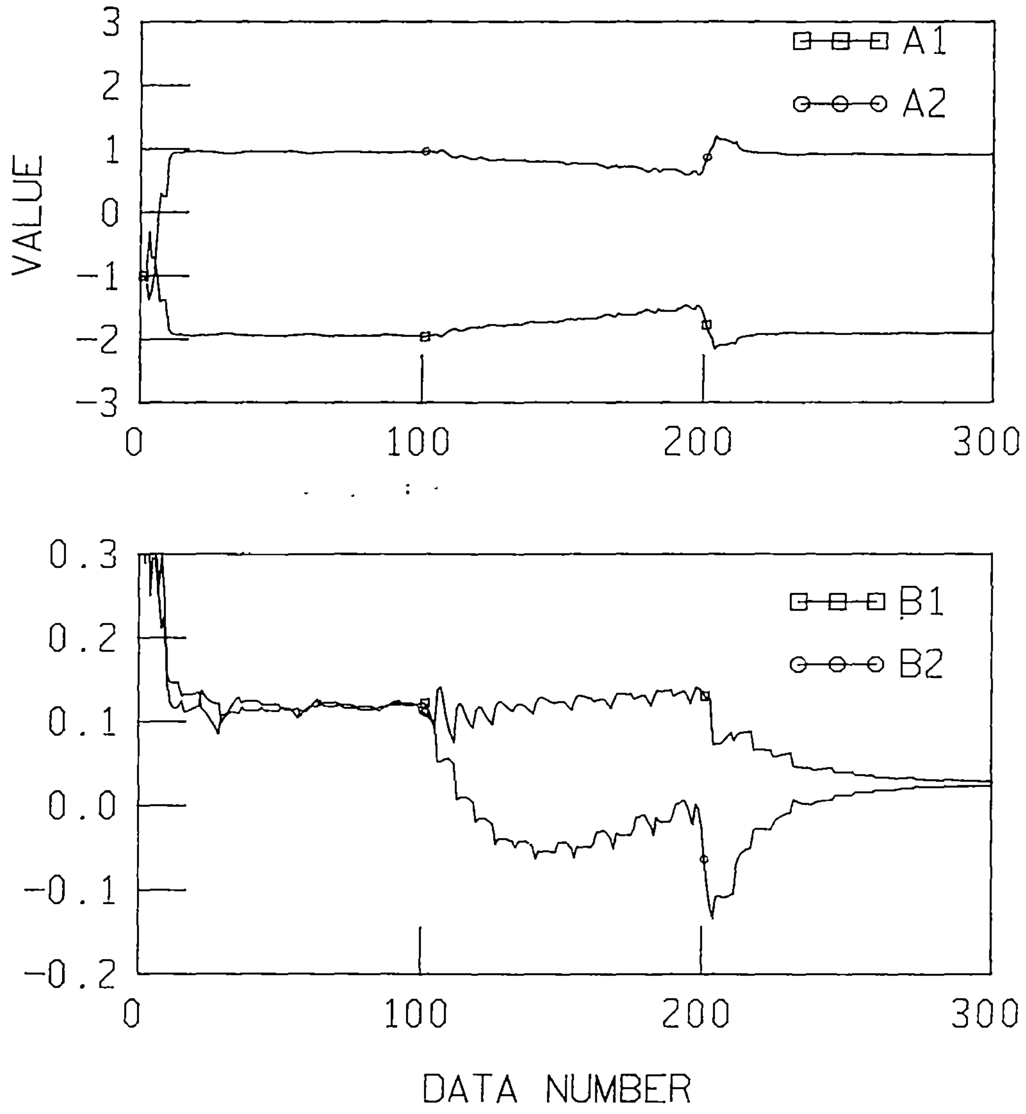


Figure 3.4 Parameter identification using RLS with δ = 0.95 and P matrix = 100

3.4 U-D Covariance Factorisation

Biermann and Thorton developed a procedure to identify plant parameters in U-D algorithms. This method factorised matrix P as

$$P=UDU^T \tag{3.14}$$

where U is an upper triangular matrix and D is a diagonal of the P matrix.

The U-D method is relatively efficient in calculation. This method updates the square root matrix P without square root calculation.

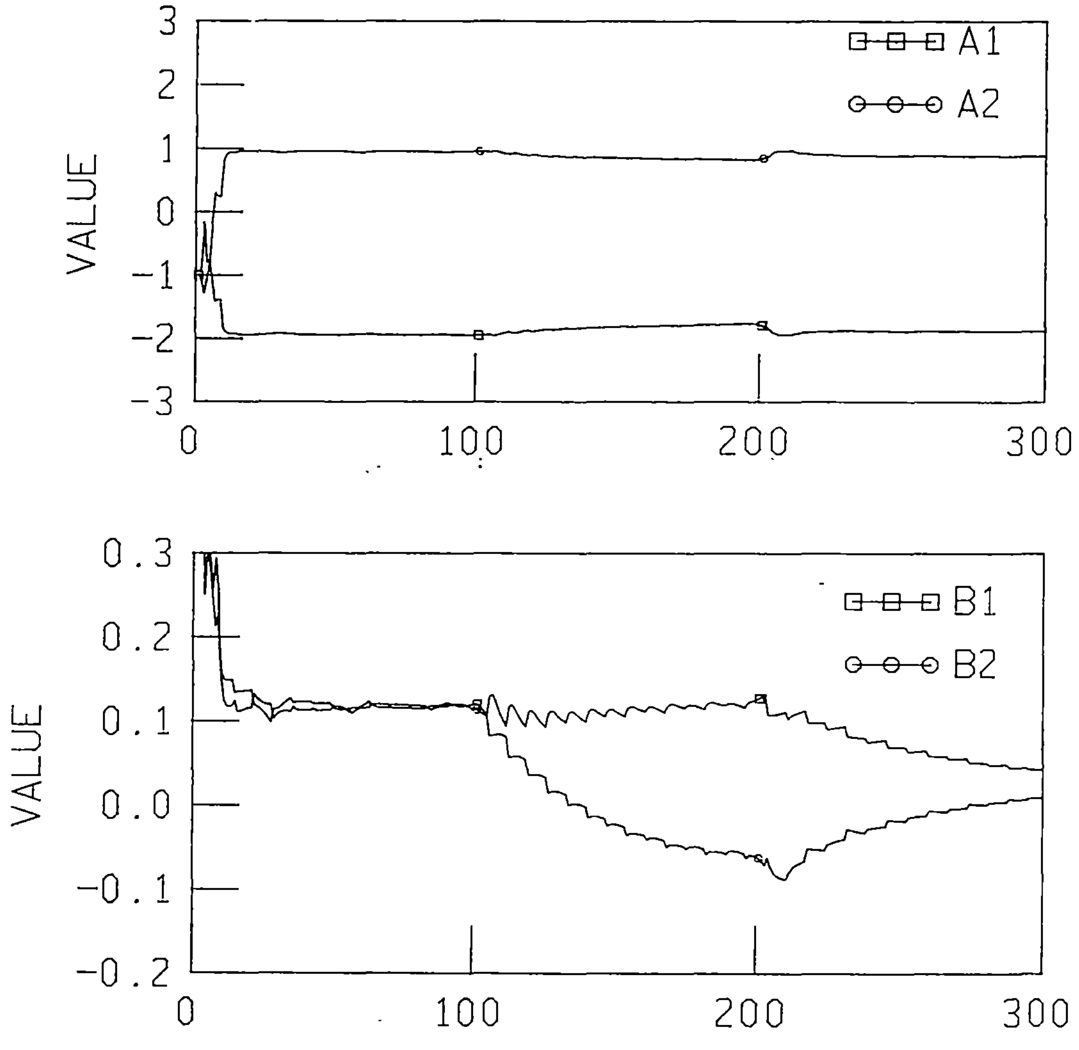
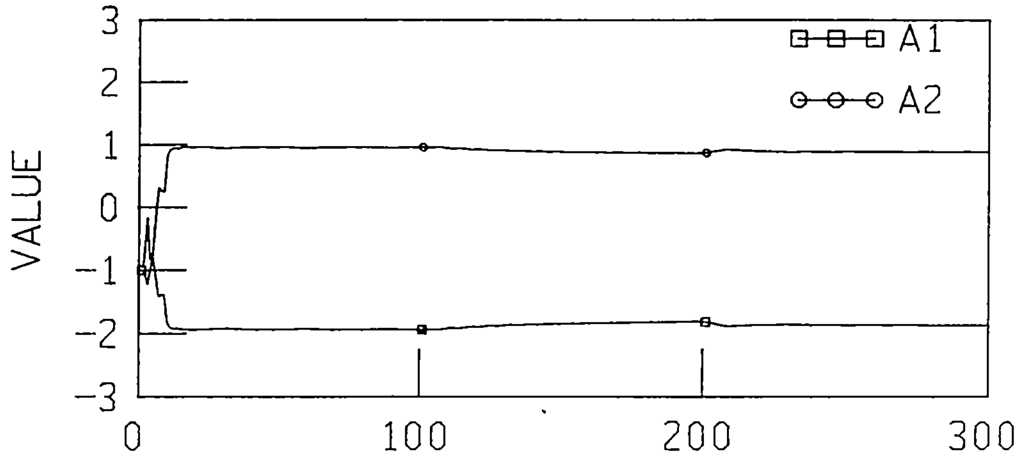


Figure 3.5 Parameter identification using RLS with $\delta = 0.98$ and P matrix $= 100$



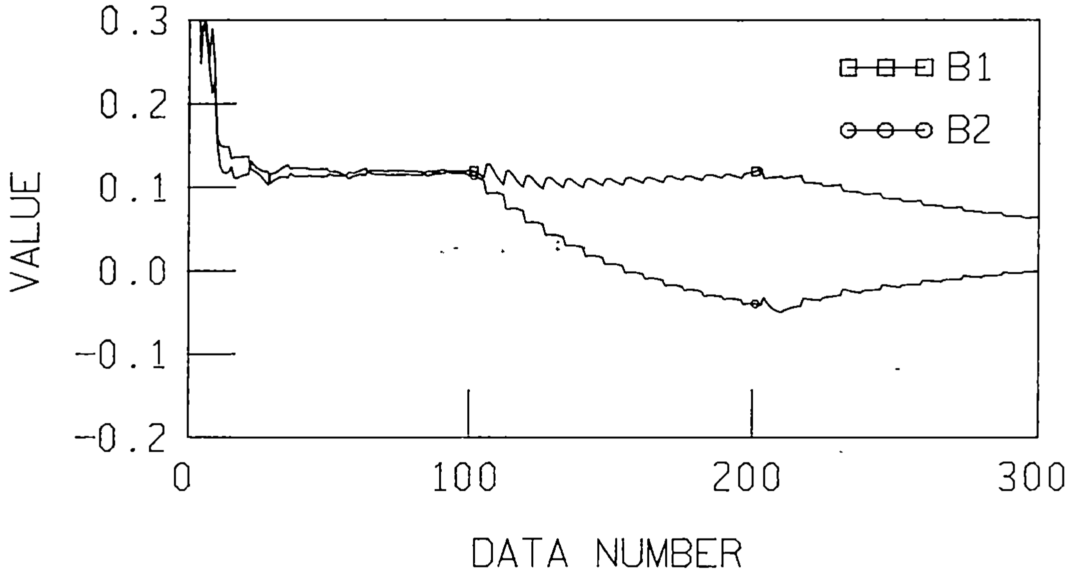


Figure 3.6 Parameter identification using RLS with $\delta = 0.99$ and P matrix $= 100$

CHAPTER 4

CONTROLLER DESIGN

4.1 General

This chapter presents an on-line solution to a controller design problem for a system with Single Input Single Output (SISO) and also with known parameters. Section 4.2 presents the controller structure of pole-zero placement design. The procedure to determine the polynomials *R, S* and *T* for second order systems is also reviewed in this section. The controller with integrating property is covered in section 4.3.

4.2 A.C. Turbo Generator Model

Before a controller can be implemented into a system, it requires a model for validation. The model must describe the dominant dynamic properties of the system to be controlled. A model of the turbogenerator based on Park's equations as described in reference 8 was used in this experiment.

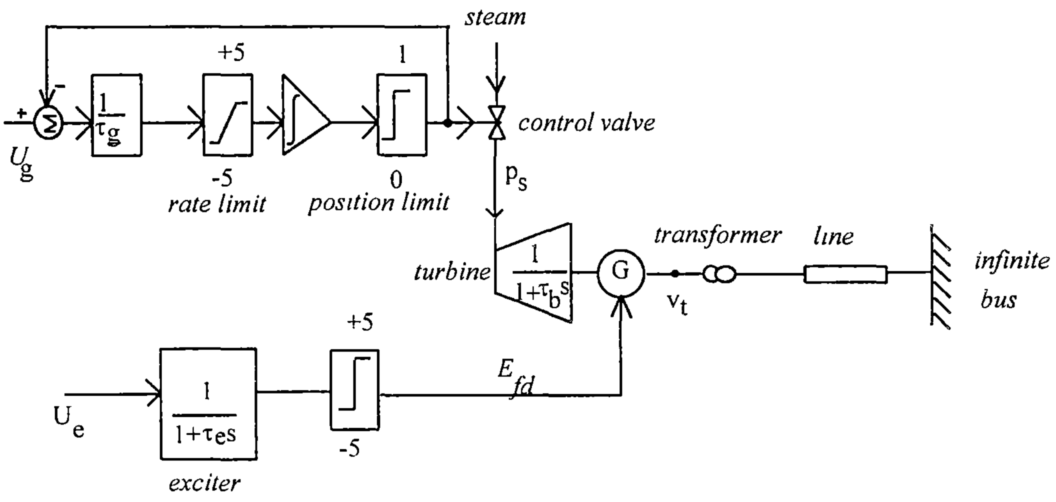


Figure 4.1 Schematic diagram of the open loop a.c. turbogenerator

Figure 4.1 shows the open loop system consisting of synchronous machine connected to an infinite bus through a transformer and a transmission line. Steam turbine as a prime mover and excitation system are also shown in this figure. The exciter input U_e applied to the exciter is assumed to be available for optimal variation and is therefore used as a control variable. The governor input U_g is taken as a second control variable.

The following state vectors represent a synchronous machine of the above system, and the state equations are described in Appendix 1.

$$X = [\delta, \dot{\delta}\, \psi_{fd}, E_{fd}, P_s, T_m]^T \tag{4.1}$$

$$U = \left[U_e, U_g\right]^T \tag{4.2}$$

$$Y = [P_t, v_t, \dot{\delta}, E_{fd}]^T \tag{4.3}$$

The four components of the output vector Y, real power output P_t , terminal voltage v_t, rotor angle velocity $\dot{\delta}$, and field voltage E_{fd} can be measured in a full-scale plant. The output vector components have been chosen by omitting inaccessible measurements to avoid the effect of deterioration in performance and loss of guarantee that the closed loop control system will no longer be stable.

4.3 Open Loop Test

In control system design, it is desirable to be able to predict whether or not a system is controllable. A state X_1 of a system is controllable if it is possible for the input vector to transfer any state X_0 at any previous time t_0 to the state X_1 in a finite amount of time. A mathematical model is a simplification of the real physical system. Thus, while most physical systems are controllable, their models might not be, and it is important to know when this occurs.

Consider a first order model $\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu}$. This model is controllable if $\boldsymbol{B} \neq 0$.

The linearised form of the state and output equations in Appendix 1 can be described as

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu} \tag{4.4}$$

$$\boldsymbol{y} = \boldsymbol{Cx} \tag{4.5}$$

where $\boldsymbol{B}$ and $\boldsymbol{C} \neq 0$

Based on these equations, the mathematical model is controllable, and the second order controller may be applied in this experiment.

The open loop test was conducted to test the controllability of the synchronous machine by checking the step response of the initial steady state vectors. This test solves the six first order non linear equations as mentioned in Appendix 1 by using Runge-Kutta procedure in Appendix 2 instead of an analitical solution.

The system equations of the synchronous machine based on Appendix 1 are as follows:

$$\dot{X}_1 = X_2 \tag{4.6}$$

$$X_2 = [(X_6 - 1.256 * X_3 * \sin(X_1) + 0.922$$

$$* \sin(X_1) * \cos(X_1) - 0.08 * X_2] * 29.637 \tag{4.7}$$

$$\dot{X}_3 = 0.180726 * X_4 - 0.561 * X_3 + 0.422 * \cos(X_1) \tag{4.8}$$

$$\dot{X}_4 = (-X_4 + U_1) * 10 \tag{4.9}$$

$$\dot{X}_5 = (-X_5 + K_v) * 10 \tag{4.10}$$

$$\dot{X}_6 = (-X_6 + X_5) * 2 \tag{4.11}$$

The terminal power and terminal voltage may be expressed in terms of the output state variable by

$$Y_1 = 1.256 X_1 \sin X_1 - 0.922 \sin X_1 \cos X_1 \tag{4.12}$$

$$Y_2 = \sqrt{(v_d^2 + v_q^2)} \tag{4.13}$$

where

$$v_d = 0.798 \sin X_1$$

$$v_q = 0.5905X_3 + 0.365\cos X_1$$

The step input to the open loop simulation was obtained by changing the real power from 0.8 p.u to 0.9 p.u as the step input.

The fourth order Runge-Kutta method as listed in the computer progamming of the open loop simulation in Appendix 4 was applied to solve the non linear equations.

The time interval h = 0.05 second and the following initial steady state vectors were used in the fourth order Runge-Kutta method for the open loop test.

$$X_{ss} = [1, 0, 1.152, 2.314, 0.8, 0.8]^T$$
$$Y_{ss} = [0.9, 1.109, 0, 2.495]^T$$
$$U_{ss} = [2.314, 0.563]^T$$

the increasing step in the input vector was

$$U_{ss} = [2.495, 0.634]^T$$

The new steady state conditions of the open loop simulation were as follows:

$$X_{ss} = [1.078, 0, 1.160, 2.495, 0.9, 0.9]^T$$
$$Y_{ss} = [0.9, 1.109, 0, 2.495]^T$$

Figure 4.2 exhibits the open loop test results against time in responding to the step input. This figure is also proof that the control law can be implemented to design a stable close loop design.

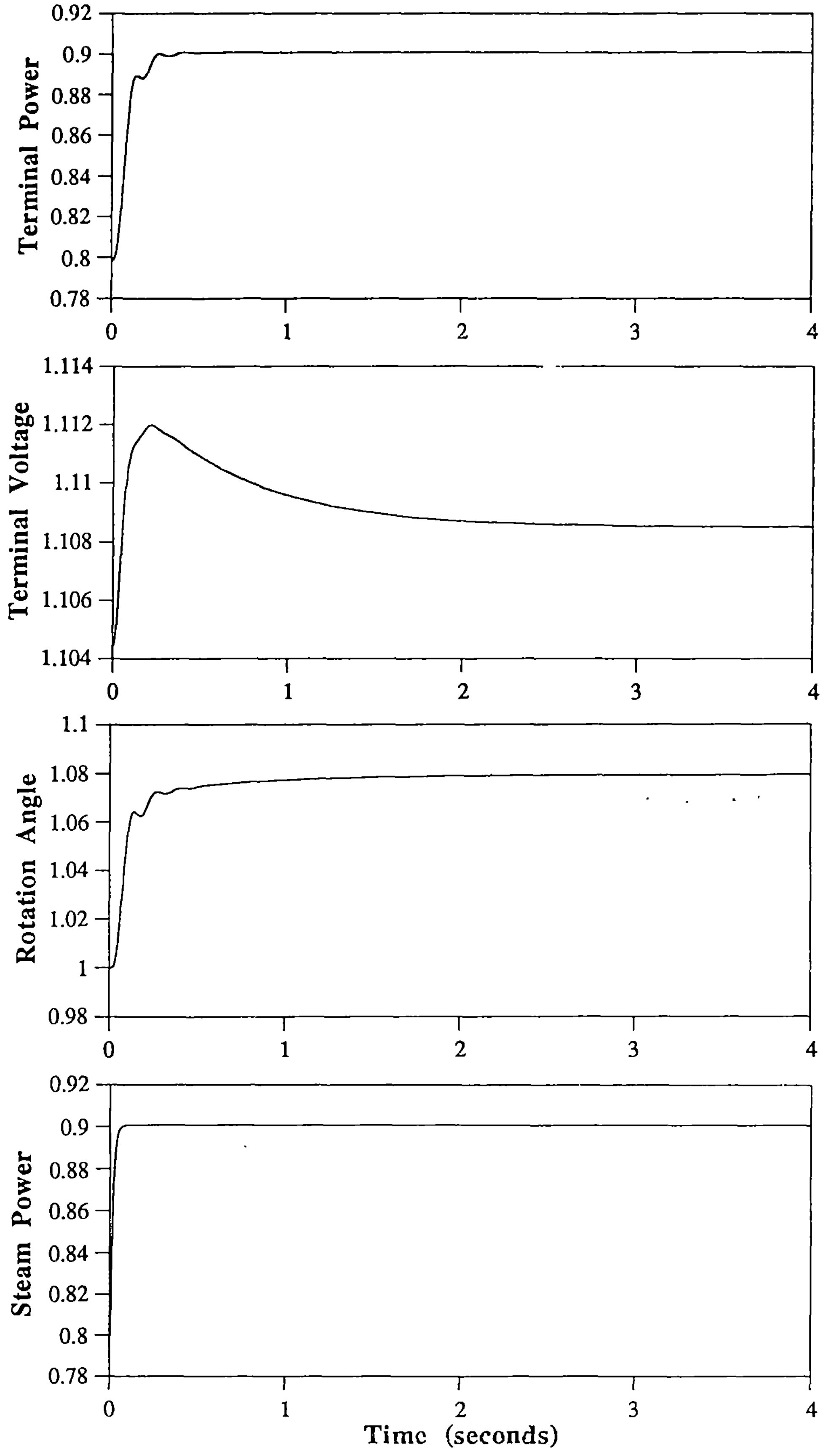


Figure 4.2 Open loop test of the mathematical model of the synchronous generator

4.4 Controller Structure

The open loop test of the mathematical model of the synchronous generator has proved that the control law can be implemented. Thus, it is desired to find a control law such that the appropriate response to command inputs is obtained. The implementation of the controller in this project is the Pole-Zero placement method that was introduced by Astrom and Wittenmark[6]. Figure 4.3 shows a block diagram of the controller structure of a system using Pole-Zero placement.

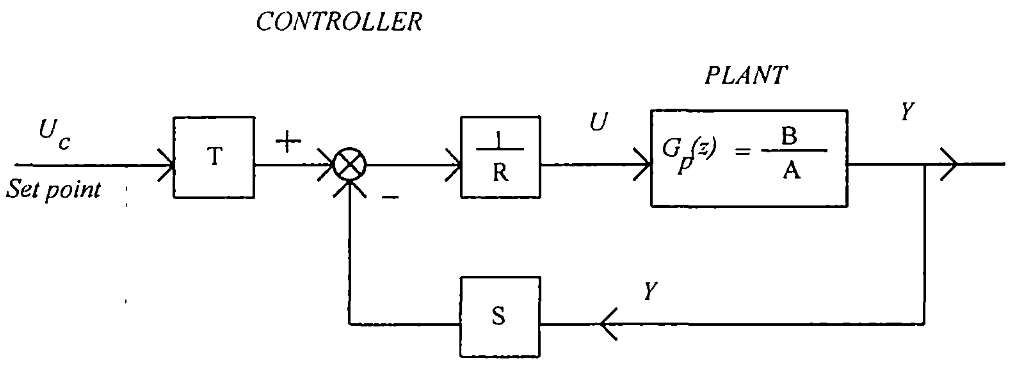


Figure 4.3 Block diagram of controller structure

Plant in figure 4.3, has a single input *u* and a single output *y* and it can be expressed by a transfer function.

$$A_{(z)}Y_{(z)} = B_{(z)}U_{(z)} \tag{4.14}$$

where *A* and *B* are polynomials in z, they do not have any common factors.

$$A(z) = 1 + a_1 z + a_2 z + \ldots\ldots. + a_n z$$

$$B(z) = b_1 + b_2 z + \ldots\ldots\ldots\ldots + b_n z$$

R, S and *T* are polynomials in z.

The coefficient of the highest power in *A* and *R* is assumed to be unity or monic.The control law of the controller can be written in z as

$$U_{(z)} = \frac{1}{R_{(z)}} T_{(z)} U_{c(z)} - S_{(z)} Y_{(z)}$$

or

$$U_{(z)} R_{(z)} = T_{(z)} U_{c(z)} - S_{(z)} Y_{(z)} \tag{4.15}$$

This equation consists of a feedforward with pulse transfer function

$$H_{ff(z)} = \frac{T_{(z)}}{R_{(z)}} \qquad (4.16)$$

and a feedback with pulse transfer function

$$H_{fb(z)} = \frac{S_{(z)}}{R_{(z)}} \qquad (4.17)$$

To ensure the causality of the feedforward and the feedback transfer functions,

$$deg\, R \geq deg\, T \qquad (4.18)$$

$$deg\, R \geq deg\, S \qquad (4.19)$$

The controller specification can be expressed as a model that gives the intended response to command signals, and it can be written as a closed loop transfer function

$$G_m(z) = \frac{B_{m(z)}}{A_{m(z)}} \qquad (4.20)$$

where

$$A_m = z^2 + n_1 z + n_2$$

$$B_m = 1 + m_1 z + m_2$$

A_m and B_m are monic and coprime, and also the zeros of A_m are assumed to be inside unit circle z.

Generally equation (4.20) requires an observer dynamic, because with output feedback, there will be additional dynamics that are not excited by the command signal. The observer dynamic is performed by specifying the characteristic polynomial Ao as the observer.

The model transfer function $G_{m(z)}$ influences the sensitivity of the closed loop system to modelling error and to high frequently measurement noise.

The consequences of the inequalities of equation 4.18 and 4.19 are

$$deg\, Am - deg\, Bm \geq deg\, A - deg\, B \qquad (4.21)$$

$$deg\ Ao \geq 2\ deg\ A - deg\ Am - deg\ B^{+} - 1 \tag{4.22}$$

Equation 4.21 implies that the delay in the model $G_{m(z)}$ must be at least as large as the delay in the plant transfer function $G_{p(z)}$. Equation 4.22 can be used to obtain the observer polynomial Ao, and it implies that the degree of Ao must be sufficiently high in order to obtain a causal control law.

If the computation time of the computer is a small fraction of the sampling time, then it is common to use

$$deg\ R = deg\ S = deg\ T \tag{4.23}$$

and if the computation time is close to sampling time, then

$$deg\ R = 1 + deg\ T = 1 + deg\ R \tag{4.24}$$

this means that the control law has a time delay of one sampling period.

The input and output relationship after $U_{(z)}$ to be eliminated can be written as

$$\frac{Y_{(z)}}{U_{c(z)}} = \frac{BT}{AR + BS} \tag{4.25}$$

By suitable selection of the polynomial R, S and T, that satisfy Equation 4.25, it is desired to obtaine a model transfer function $G_{m(z)}$ equal to Equation 4.25. Hence, the desired closed loop transfer function can be written as

$$\frac{BT}{AR + BS} = \frac{B_m}{A_m} \tag{4.26}$$

The zeros of the closed loop system are the zeros of the polynomials B and T, and the poles of the closed loop system are the solution of

$$AR + BS = 0$$

To satisfy the condition in Equation 4.26, there must then be cancellation of poles and zeros. Consider the zeros of the polynomial B that represents the zeros of the open loop. If any root of the denominator BA_m of Equation 4.26 is outside the unit circle or if B has unstable roots or poorly damped roots near the unit circle, this condition is undesirable. Because such zeros of B are unable to be cancelled, the polynomial B is factorised as

$$B = B^- B^+ \qquad (4.27)$$

where B^+ has well damped roots located inside the specified region and B^- has the remaining unstable or poorly damped roots outside the specified region. Equation 4.26 shows that B^- must be a factor of B_m so

$$Bm = B'mB^- \qquad (4.28)$$

To enable B^+ to be cancelled, it must be a factor of R. By introducing R' that is a monic, hence

$$R = B^+R' \qquad (4.29)$$

In order to obtain Equation 4.16 and 4.17 causal without delay time, so

$$Deg\ R' = deg\ Am + deg\ Ao - deg\ A \qquad (4.30)$$

$$Deg\ S = deg\ A - 1 \qquad (4.31)$$

Now, equation 4.25 can be rewritten as :

$$\frac{T}{(AR'+B^-S)} = \frac{B'_m}{A_m} \qquad (4.31)$$

The degree of A_m is normally less than the dominator of Equation 4.26 *(AR + BS),* so that there are factors which cancel. This cancelling factor is the observer polynomial *Ao*. The roots of *Ao* are assumed in the unit circle.

By comparing the nominator and the denominator of Equation 4.31, the following conditions are obtained:

$$T = B'm\ Ao \qquad (4.32)$$

and

$$AR' + B^-S = A\ oAm \qquad (4.33)$$

(4.14)

Consider the second order plant transfer function

$$G_{p(z)} = \frac{B_{(z)}}{A_{(z)}} = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2}$$

$$= \frac{b_1[z + \frac{b_1}{b_2}]}{z^2 + a_1 z + a_2} \qquad (4.34)$$

Depending on the zero of the open loop system of Equation 4.34, the value of $\frac{b_2}{b_1}$ can lie outside or inside the unit circle. Therefore, Equation 4.34 has two possible solutions.

Case(i) $\frac{b_2}{b_1} < 1$

In this case the zero is inside the unit circle and it should be cancelled in the closed loop system. Due to the zero being inside the unit circle, then

$$B^- = b_1$$

and

$$B^+ = z + \frac{b_2}{b_1}$$

By selecting observer $Ao = 1$ and from Equation 4.21, the degree of Am is obtained as

$$0 = 4 - \deg Am - 1 - 1$$

$$\text{degree } Am = 2$$

Referring to Equation 4.28 $B_m^{'}$ can be written as

$$B_m^{'} = \frac{m_1}{b_1} z + \frac{m_2}{b_1}$$

Deriving from Equation 4.30 the degree of R' is zero, hence $R' = r_0' = 1$.

From Equation 4.31, the degree S is equal to one, so, the polynomial S is

$$S = SoZ + S_1$$

The polynomials S, R and T can be obtained from Equation 4.33,4.29 and 4.32 respectively as follows:

$$s_0 = \frac{n_1 - a_1}{b_1} \qquad (4.35)$$

$$s_1 = \frac{n_2 - a_2}{b_1} \tag{4.36}$$

From Equation 4.29 the polynomial R can be obtained as follows

$$R = B^+ R'$$
$$= [z + \frac{b2}{b1}] r_0'$$
$$= [z + \frac{b_2}{b_1}]$$
$$r_1 = \frac{b_2}{b_1} \tag{4.37}$$

Hence the polynomial R is

$$R = z + \frac{b_2}{b_1}$$

The coeficient of the polynomial T can be obtained as follows

$$T = \frac{m_1}{b_1} z + \frac{m_2}{b_1}$$
$$t_0 = \frac{m_1}{b_1} \tag{4.38}$$
$$t_1 = \frac{m_2}{b_1} \tag{4.39}$$

The control law can be written as

$$u_k = t_o U_c + t_1 U_c - s_0 y_k - s_1 y_{k-1} - r_1 U_{k-1}$$

Case (ii) $\frac{b_2}{b_1} > 1$

The zero is outside the unit circle, this zero is unstable and it cannot be cancelled. So, it must be included in the closed loop system.

$$B^+ = 1 \tag{4.40}$$

and

$$B^- = b_1 z + b_2 \tag{4.41}$$

By considering Equation 4.31

$$deg\, S = Deg\, A - 1$$

$$= 2 - 1 = 1$$

and Equation 4.22

$$deg\, S = deg\, R = deg\, T$$

hence

$$R = R_0 z + R_1$$

Consider Equation 4.29

$$R = B^+ R'$$

Because $B^+ = 1$ then

$$R = R'$$

$$R' = R_0 z + R_1$$

Recalling Equation 4.33

$$AR' + B^- S = A\, oAm$$

and substituting Equation 4.40 and 4.41 into Equation 4.33 gives

$$(z^2 + a_1 z + a_2)(R_0 z + R_1) + (b_1 z + b_2)(S_0 z + S_1) = AoAm \qquad (4.42)$$

Let $Ao = 1$ and expanding Equation 4.42 gives

$$R_0 z^3 + (R_1 + a_1 R_0 + b_1 S_0) z^2 + (a_1 R_1 + a_2 R_0 + b_1 s_1 + b_2 s_0) z + a_2 R_1 + b_2 S_1$$

$$= z^2 + n_1 z + n_2 \qquad (4.43)$$

Comparing the coefficients on both sides of Equation 4.43 gives the solution to be

$$R_0 = 0$$

$$R_1 + a_1 R_0 + b_1 S_0 = 1 \qquad (4.44)$$

$$a_1 R_1 + a_2 R_0 + b_1 S_1 + b_2 S_0 = n_1$$

$$a_1 R_1 + b_1 S_1 + b_2 S_0 = n_1 \qquad (4.45)$$

$$a_2 R_1 + b_2 S_1 = n_2 \qquad (4.46)$$

Equation 4.44 and 4.46can be written as

$$R_1 = 1 - b_1 S_0 \qquad (4.47)$$

and

$$R_1 = \frac{n_2 - b_2 s_1}{a_2} \qquad (4.48)$$

Substituting Equation 4.47 into Equation 4.48 gives

$$\frac{n_2 - b_2 s_1}{a_2} = 1 - b_1 S_0 \tag{4.49}$$

$$S_0 = -\frac{n_2 + b_2 s_1}{a_2 b_1} + 1 \tag{4.50}$$

Substituting Equation 4.48 and 4.50 into Equation 4.45 gives

$$S_1 = \frac{n_2 a_1 + b_2 a_2 - a_2 n_1}{a_2 b_1 + a_1 b_2 + \frac{b_2^2}{b_1}} \tag{4.51}$$

The polynomial T can be derived from the nominator of Equation 4.26,

$$BT = Bm$$

$$(b_1 z + b_2)(T_0 z + T_1) = m_1 z + m_2 \tag{4.52}$$

Expanding Equation 4.52

$$b_1 T_0 z^2 + (b_1 T_1 + b_2 T_0) z + b_2 T_1 = n_1 z + n_2 \tag{4.53}$$

Comparing the coefficients on both sides of Equation 4.53 gives the solution to be

$$b_1 T_0 = 0$$

$$b_1 T_1 + b_2 T_0 = m_1$$

since $b_1 \neq 0$, then

$$T_0 = 0$$

$$T_1 = \frac{m_1}{b_1} \tag{4.54}$$

The control law is

$$RU = TUc - SY$$

Substituting Equation 4.48 , 4.50 and 4.54 into Equation 4.45 gives

$$R_1 U = T_1 U_c - (S_0 z + S_1)(Y)$$

The controller design can be described as

$$U_k = \frac{1}{R_1}(T_1 U_c - S_0 Y_k - S_1 Y_{k-1}) \tag{4.55}$$

4.5 Controller with Integrity Properties

The controller must have the ability to eliminate steady state errors, which can be generated from calibration errors or the other disturbances.

To cope with this problem, Astrom and Wittenmark proposed controllers with a forced integral action so that the plant will always have a pole at z = 1 This can be achieved by specifying $(z-1)^l$ to be a factor of *R*.

By considering Equation 4.29, the polynomial *R* can be written as

$$R = B+(z-1)l\, R'' \qquad (4.56)$$

and by accommodating this equation into the Equation 4.33, so this equation is then replaced by

$$A(z-1)l\, R'' + B^{-}S = AoAm \qquad (4.57)$$

The polynomial T can be derived from Equation (4.32)

$$T = B'mAo$$

The advantage of this scheme is that the controllers will always have integral actions to impose any error in the plant to go to zero.

Referring to equation 4.57 to 4.32, the controller polynomials for second order systems can be given as follows[8]:

By selecting $Ao = 1$ and $l = 1$

$$\deg Ao = 2 \deg A - \deg Am - \deg B+ - l - 1 \qquad (4.59)$$

and

$$\deg R'1 = \deg Ao + \deg Am - \deg A - l \qquad (4.60)$$

and

$$\deg S = \deg A - l - 1 \qquad (4.61)$$

Hence, the polynomials *R*, *S* and *T* can be rewritten as follows:

$$R = z^2 + (\frac{b_2}{b_1} - 1)z - \frac{b_2}{b_1} \quad (4.62)$$

$$S = s_0 z^2 + s_1 z + s_2 \quad (4.63)$$

where

$$s_0 = \frac{n_1 + 1 - a_1}{b_1}$$

$$s_1 = \frac{n_2 + a_1 - a_2}{b_1}$$

$$s_2 = \frac{n_3 + a_2}{b_1}$$

and

$$T = \frac{m_1}{b_1} z^2 + \frac{m_2}{b_1} z + \frac{m_3}{b_1} \quad (4.64)$$

The controller can be written as

$$\begin{aligned} U_k = {} & T_0 U_k + T_1 U_{k-1} + T_2 U_{k-2} - S_0 Y_k \\ & - S_1 Y_{k-1} - R_0 U_k - R_1 U_{k-2} \end{aligned} \quad (4.65)$$

4.6 Controller Simulations

The implementation of the parameter identification and the design controller are conducted into several type of simulations.

The general conditions of the simulation are as follows:

- Controller specifications
    - Rising time 1 second
    - Samplling time 0.1 second
- Data number 600
- Parameter change at data number 125 and 375

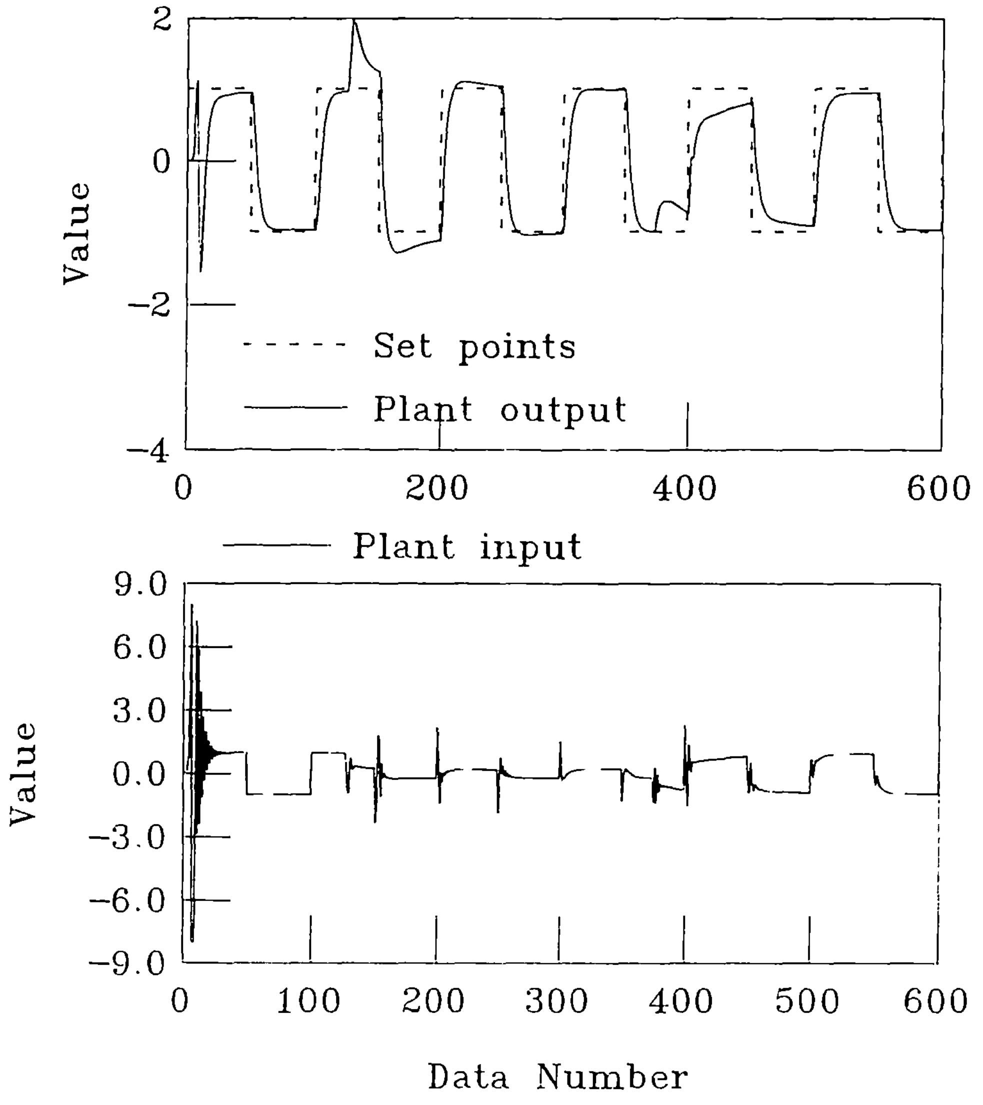


Figure 4.4 Three different second order plants with initial P matrix =100 and $\delta = 0.98$

Figure 4.4 shows the behaviour of adaptive control in controlling three second order plants with the following equations

Plant 1: $y_k = 0.0621596u_{k-1} + 0.0476u_{k-2} - 1.33596y_{k-1} + 0.4493y_{k-2}$

Plant 2: $y_k = 0.099313u_{k-1} + 0.0986u_{k-2} - 1.9213y_{k-1} + 0.9607y_{k-2}$

Plant 3: $y_k = 0.1215216u_{k-1} + 0.0929u_{k-2} - 1.23484y_{k-1} + 0.4493y_{k-2}$

It can be seen that the adaptive control has successfully tracked the set point. The change of the plants is indicated with spikes at data number 125 and 375

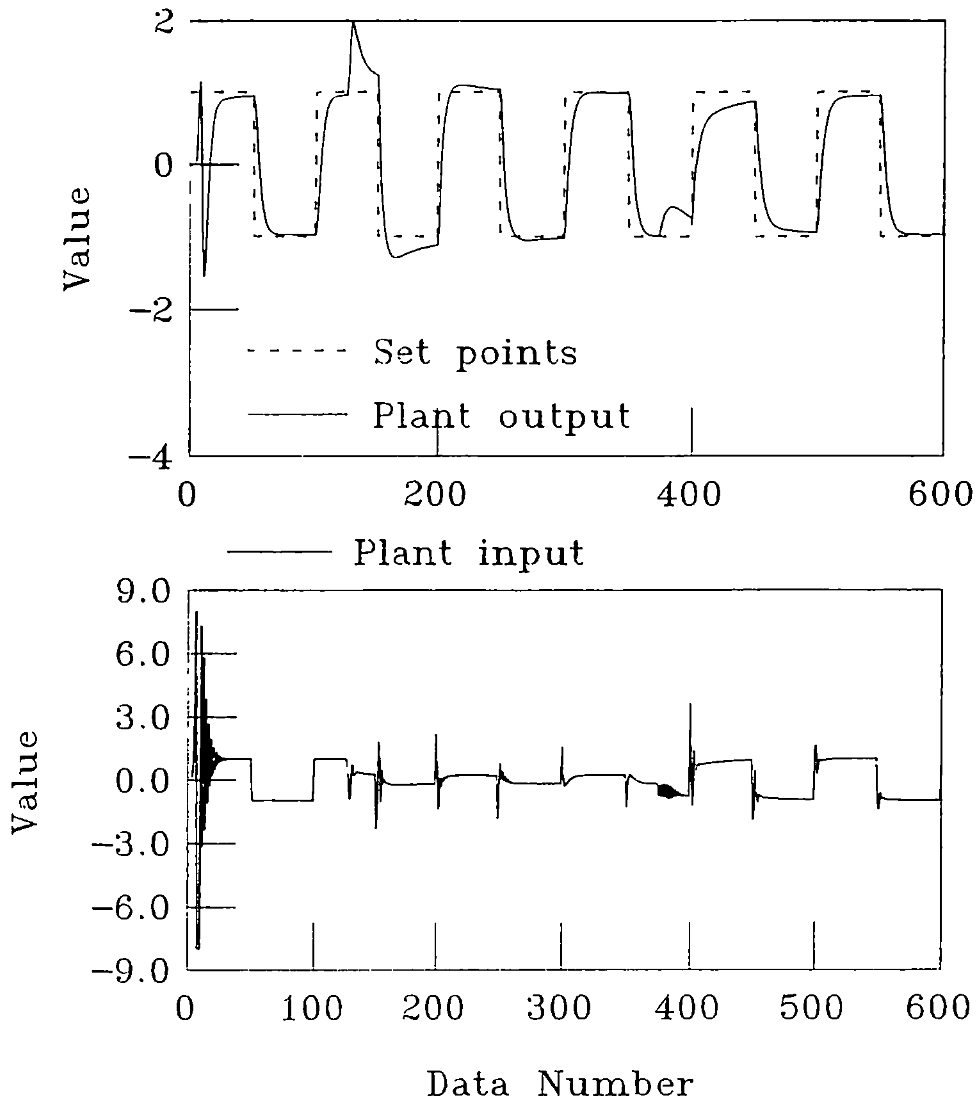


Figure 4.5 Plant 1 is reused at data number 375 instead of plant 3, with initial P matrix =100 and δ = 0.98

Compared to the transient period of plant 1 at the begining of the data number, the transient period of plant 1 after the change of the plant at data number 375 needs a longer time to converge to the set point.

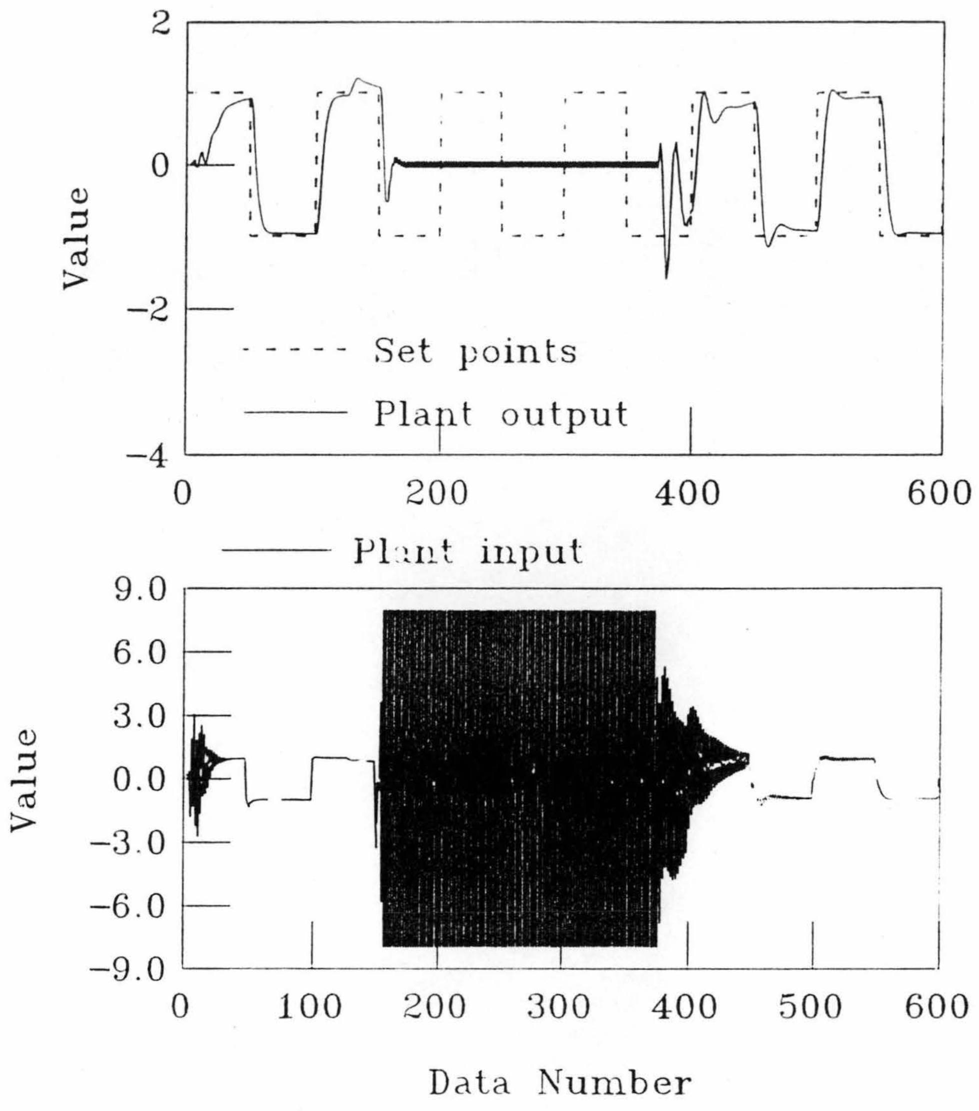


Figure 4.6 Third order plant is applied at data number 125 instead of plant 2

By using third order plant

$$y_k = 0.020228u_{k-1} + 0.0596u_{k-2} + 0.111u_{k-3} + 1.90513y_{k-1} - 1.27704y_{k-2} + 0.3012y_{k-3}$$

It can be seen that the controller unable to track the set point

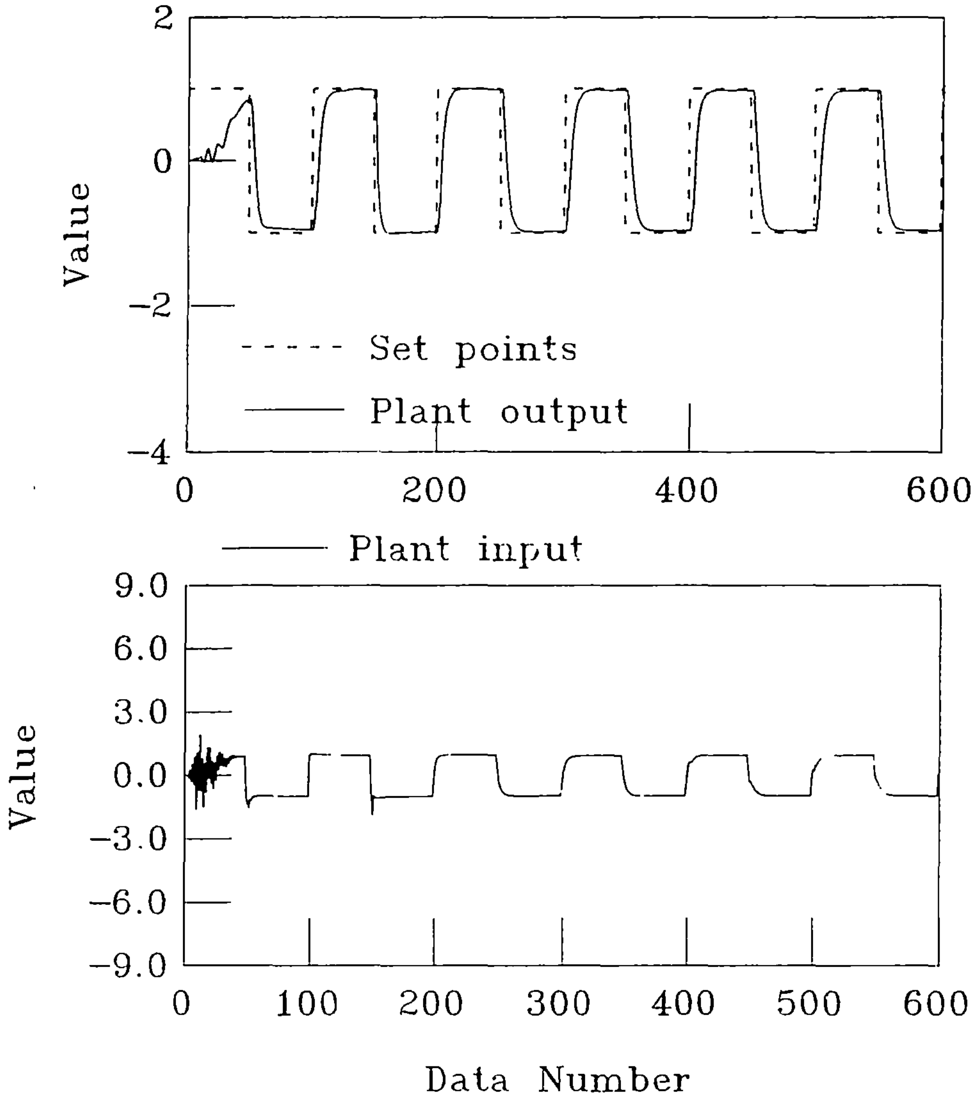


Figure 4.7 First order plant is used instead of plant 2 at data number 125

CHAPTER 5

APPLICATION OF CONTROLLERS

5.1 System Configuration

The controllers discussed in Chapter 4 are implemented in a laboratory model of a power generating system. The system consists of a 7.5 KVA synchronous generator which is connected to an infinite bus by transmission lines. A DC motor drives the synchronous generator simulating a turbine which is used in the experiment. The block diagram of the adaptive controller controls the exciter of the synchronous generator and is shown in figure 5.1

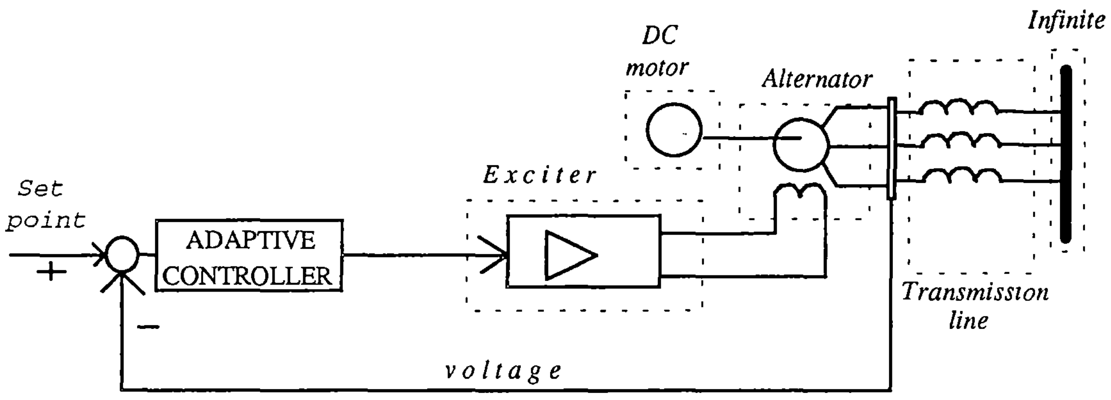


Figure 5.1 Diagram of turbogenerator system.

The experiment was based on the synchronous machine operated at a fixed governor with terminal voltage varied at different set points. The exciter voltage was adjusted adaptively by the adaptive controller to obtain the desired terminal voltage.

An explicit self-tuning regulator was selected as the basis of adaptive controller design in this experiment.

5.2 Laboratory Configuration

The control configuration schematic is shown in figure 5.2

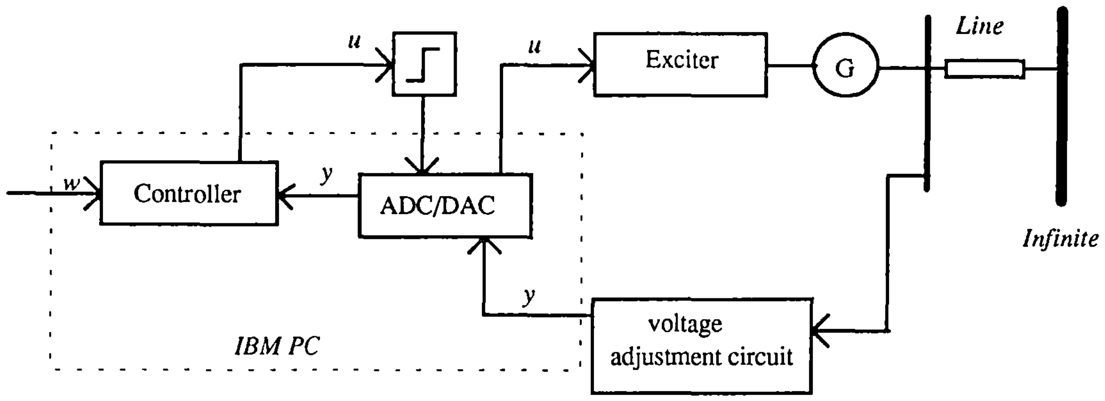


Figure 5.2 Hardware configuration schematic

The instrumentation and hardware used in the experiment are as follows:

1. An IBM personal computer with real time graphics (QUINN CURTIS) and PC-30 for AD/DA converter programming software. Both varieties software can be run by program language PASCAL, C and FORTRAN.
2. An AD/DA converter (Boston card technology) installed in personal computer.
3. A voltage adjustment circuit for terminal voltage measurement.
4. Field exciter controller (Robicon) is activated by the analog signal from D/A converter. This controller consists of a four quadrant thyristor rectifier.
5. A fixed voltage DC motor to drive the synchronous generator.
6. A inductance simulating transmission line.
7. A 7.5 KVA synchronous three phase generator connected to an infinite bus via inductance.

The digital output *u* of the controller is connected to the DAC inputs of the Boston technology card to convert its digital values into analog values. The controller output values are set at ± 10 V to avoid field excitation over voltage. The controller feedback is obtained from the terminal voltage *y*. This voltage is connected to a voltage adjustment circuit to reduce and rectify the AC voltage into DC voltage.

Computer programming in TURBO PASCAL was written to run the PC-30 module and QUINN CURTIS during investigation.

PC-30 is a unit program which converts analog signals into discrete signals before being passed on to the PC and vice versa. The ADC/DAC used on the PC-30 board have 12 bit resolution. The analog input and output of the controller derived from the terminal voltage and passed on to the exciter were converted into discrete signals by PC-30. Unit PC30io1 was written to set up the PC-30. This unit was assigned for initiating the signals conversion, limiting the analog output into $\pm$ 10 Volt, setting the sampling time, checking the AD/DA conversions and PC-30 installation error.

Unit RTPLOT1 was written for plotting the real time calculation result on the screen by using the QUINN CURTIS module. This unit split the screen into two windows. The plant output and set point were displayed on one window, and the exciter input was displayed on the other window.

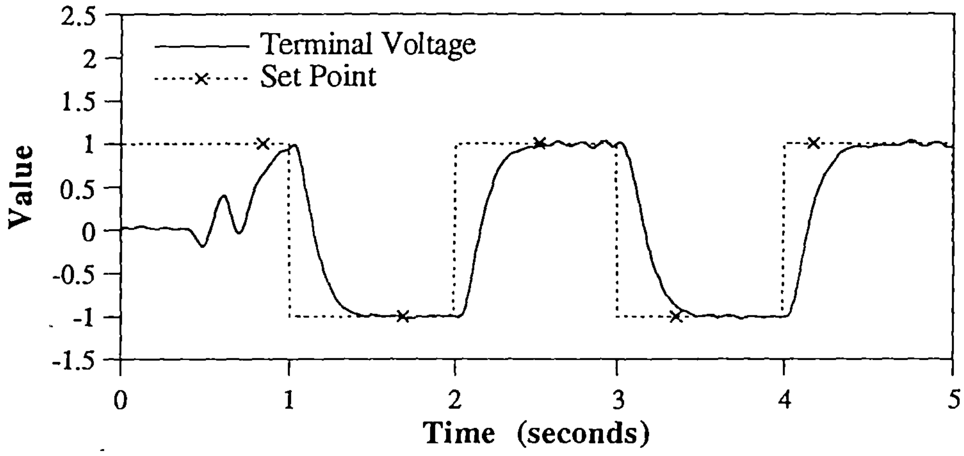
Program PLT.pas plotted the calculation result of the controllers using HGRAPH. HGRAPH is a series of procedures and functions which enable the production of two and three dimensional plots in color. The plot can be displayed on the screen or plotted by a pen plotter. The graph may also be incorporated into a document written in WORD for WINDOWS.

5.3 Controllers Implementations

The controller was implemented at the governor in a fixed set point, and the variation of the terminal voltage was set by the program. Two types of controller were developed in this investigation. The first was the adaptive controller while the other was the fixed controller or non adaptive controller.

The investigations were conducted with the following conditions:

- The experiments were initiated with $\theta^T = [\ 1\ 1\ 1\ 1\ ]$.
- The exponential forgetting factor $\delta = 0.98$.
- P matrix = 100 .
- The sampling time = 10 ms .
- The period of each investigation = 5 seconds .



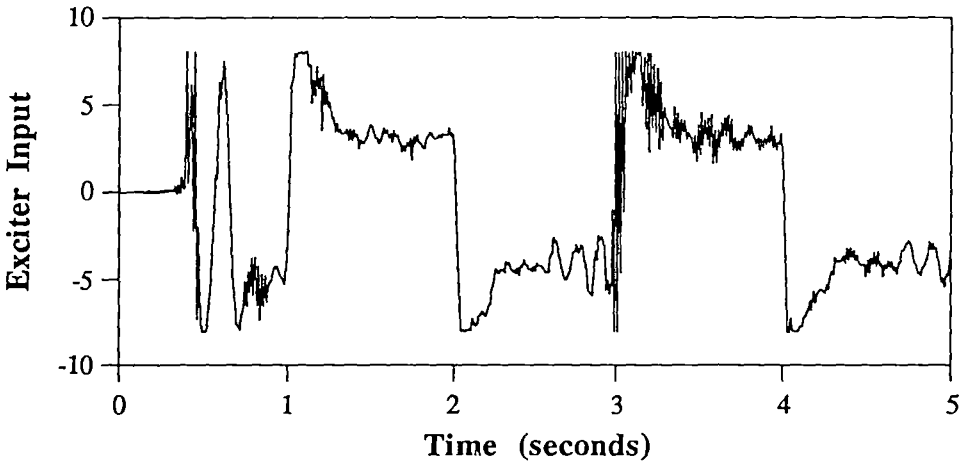


Figure 5.3 Adaptive controller using integrator

5.3.1 Adaptive Controller Investigations

To eliminate the steady state error, the principle of integrity properties in adaptive controllers was used in this experiment. The terminal voltage and the exciter input were minitored through real time graphic and recorded in a data file for plotting. The ability of the controller to track the variation of terminal voltage was investigated by changing the set point every second as shown in figure 5.3 and at every 0.5 second in figure 5.4. These figures show the ability of the controllers to track the given set point and converge on the set point within 0.3 second

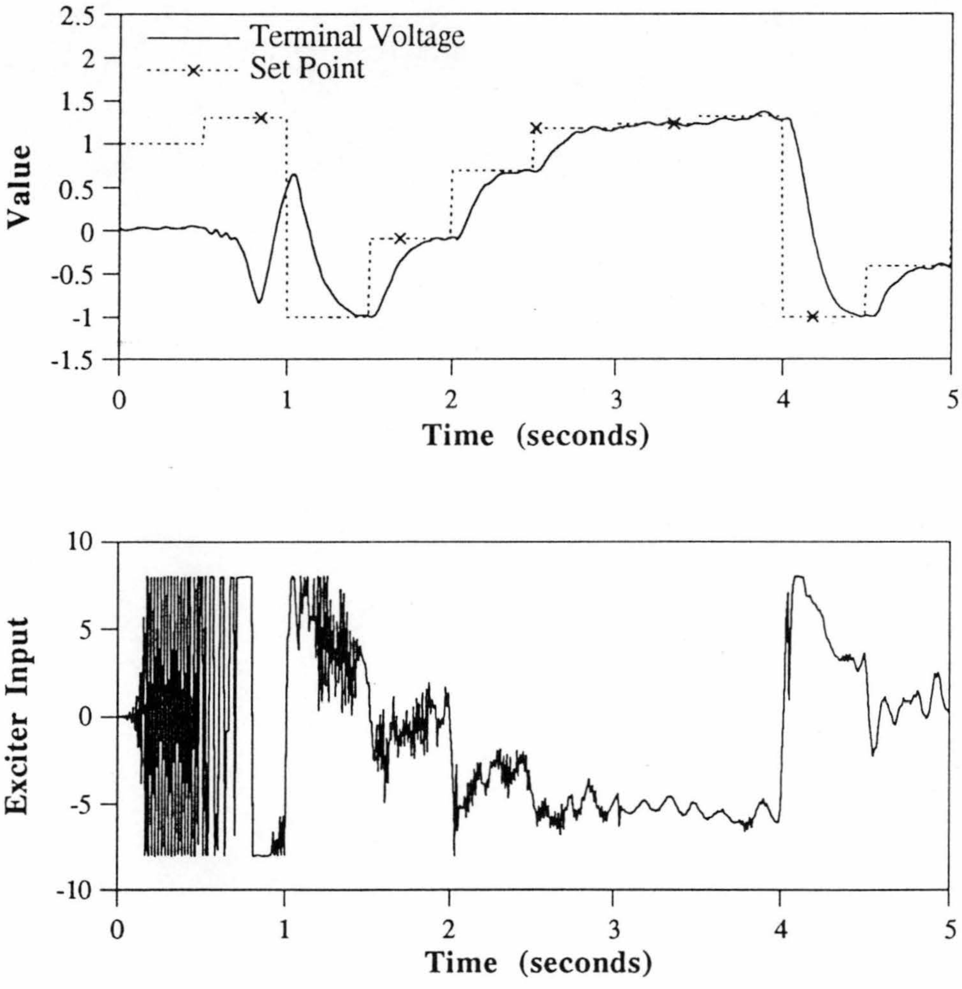


Figure 5.4 Adaptive controller using integrator with terminal voltage are randomly set every 0.5 second

To examine the adaptivity of the controllers, the reactance of the transmission line was disconnected from the machine at iteration 250th or at 2.5 second after starting. Figure 5.5 shows the ability of the controller to track the change of plant parameters after the transient occured. The terminal voltage returned to the set point within 0.3 second or 30 iterations.

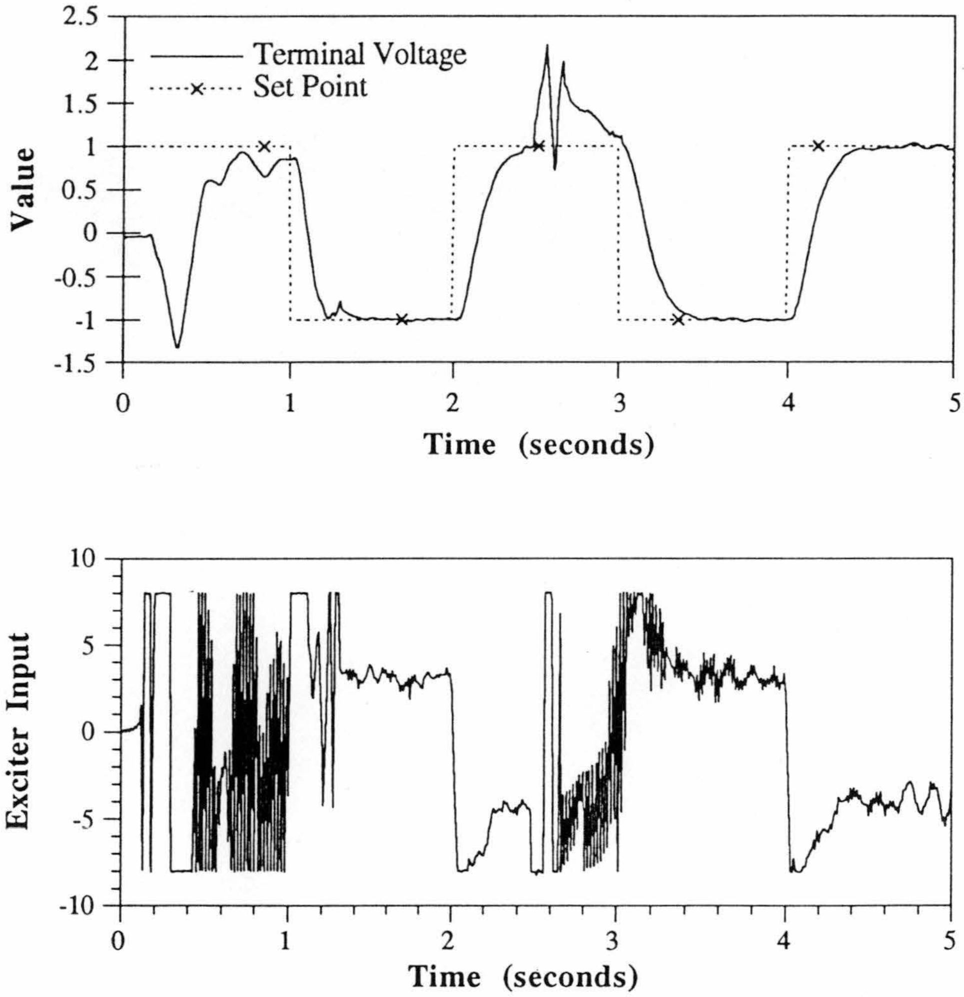


Figure 5.5 Adaptive controller using integrator with plant disconnected from the line at 2.5 seconds

The effect of no inherent integrator in the controller is exhibited in figure 5.6. The controllers are unable to converge on the terminal set point due to steady state errors. However, it is shown that the terminal voltage is stable.

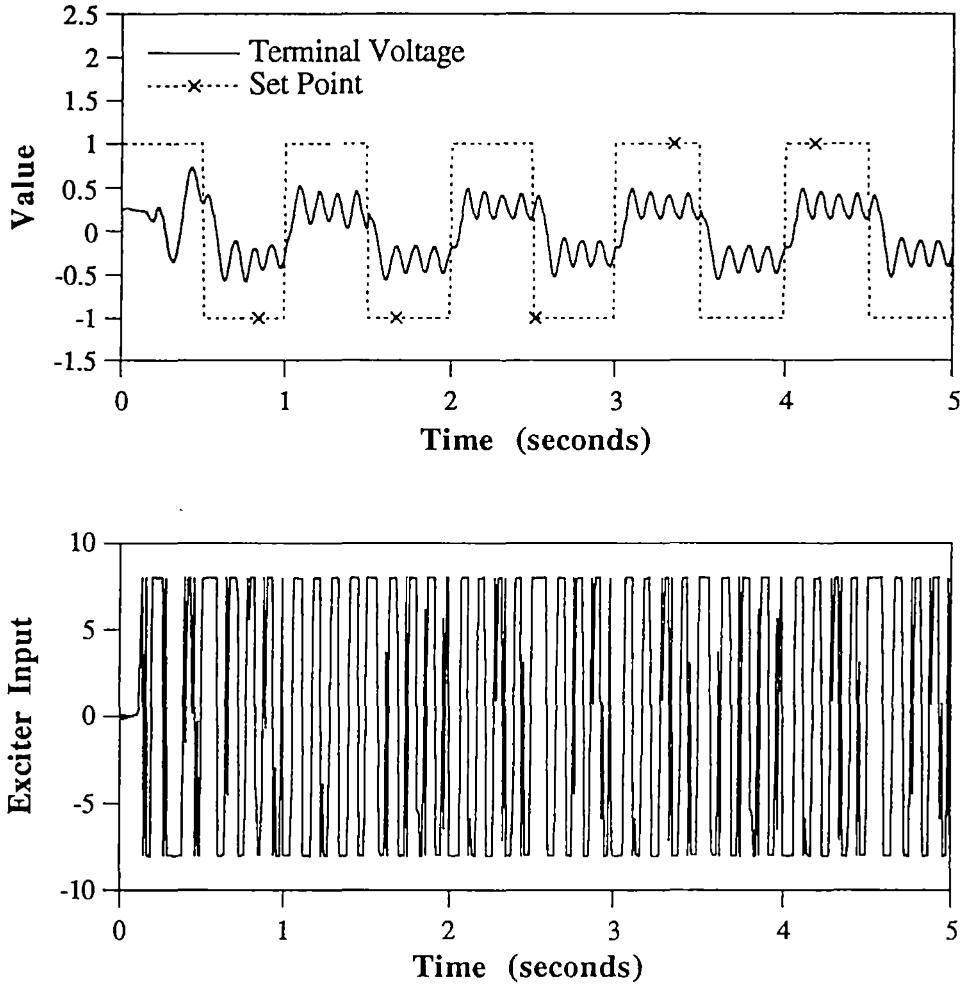


Figure 5.6 Adaptive controller without integrator

5.3.2 Fixed Controller

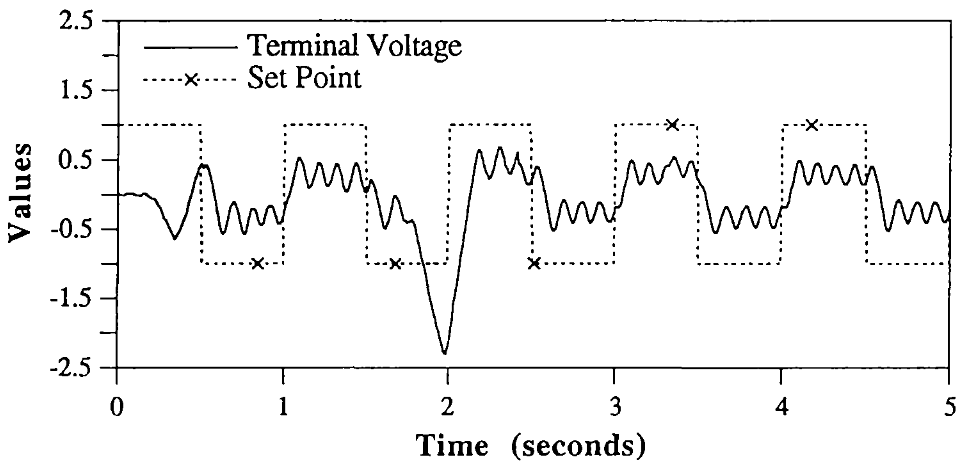
In order to obtain the character of a fixed controller, the experiment without adaptive controller was conducted. This experiment used fixed parameters which were randomly obtained from the plant identification. The plant parameters are

a_1 = - 1.71906

a_2 = 0.74066

b_1 = 0.01294

b_2 = 0.01193



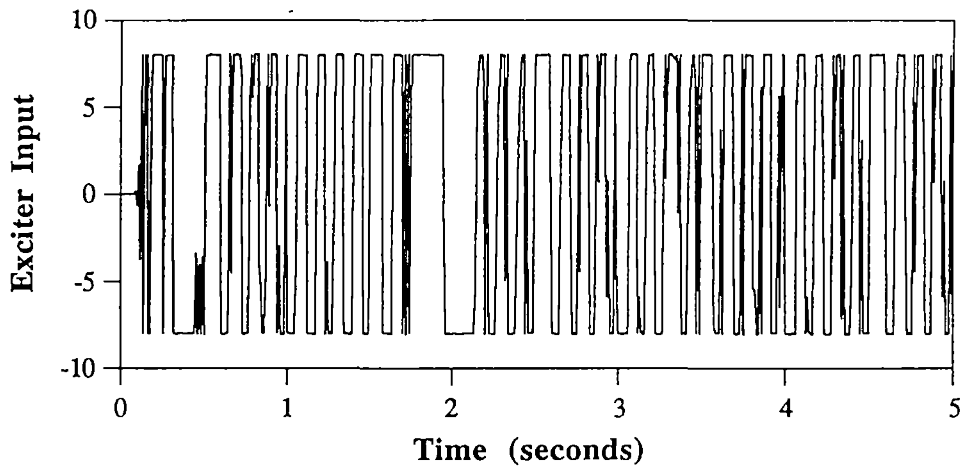


Figure 5.7 Non adaptive controller without integrator with plant disconnected from the line at 1.7 seconds

Figure 5.7 shows constant terminal voltage during the steady state condition. At iteration 170 or 1.7 second after starting point, the generator was disconnected from the reactance. As a result of the disconnection, the plant parameters were changed and the controller tried to adjust the excitation to recover the transient. The transient period took 0.5 second to recover into the steady state condition. However, the transient time was relatively longer than the adaptive control.

CHAPTER 6

SUMMARY

From the simulations and laboratory experiments, it can be seen that to obtain a good performance of an adaptive controller each step of the algorithm has to be considered carefully.

Parameter identification using recursive least squares with exponential forgetting factors as discussed in Chapter 3 demonstrated the ability to provide online plant parameters identification. This ability was prooved by validating the identified parameters againts the real plant. The exponential forgetting factor $\delta = 0.8$ and P matrix = 100 with initial $\theta^T = [1\ 1\ 1\ 1]^T$ were choose for computer simulation in Chapter 4 and laboratory investigation in Chapter 5. These values gave smooth identified parameters and fast response without giving high spike parameters during transient periods.

The open loop test proved that the control laws could be applied to design a stable closed loop control system for a synchronous generator. This test was conducted by using fourth order Runge Kutta procedure to solve first order non linear equations.

Simulations in various conditions indicated that the second order adaptive control had a good damping property and the application of an integrator in the control loop could eliminate the steady state error.

The voltage variations in the point where the parameter changed could not be accepted in real operations. This was due to the limitatition of the exciter voltage.

In the real machine that connected to infinit bus the voltage variations do not so often vary with relatively high magnitude

It was demonstrated that adaptive control with pole-zero algorithm is able to handle a process in time varying and also having a stable performance in the presence of disturbances. It does not mean that the fixed controller cannot be used. For a simple control, the PID type with automatic tuning as proposed by Astrom and Hagglund[9] can be implemented

It was shown that the adaptive controller is sufficiently robust to handle a stochastic and non linear process.

APPENDIX 1

Mathematical Model of Synchronous Generator

Symbols of machine variables to be used in simulating synchronous machine are listed as follow

v_d, v_q	= Stator volages in *d*- and *q*- axis circuits
v_t	= Terminal voltage
Ψ_{fd}	= Field flux linkage
x_d, x_q	= Synchronous reactances in *d*- and *q*-axis circuits
x_{ad}	= Stator rotor mutual reactance
x_{fd}	= Self reactance of field winding
U_e	= Input to exciter
e_d, e_q	= Components of busbar voltage in *d*- and *q*-axis
e	= Busbar voltage
δ	= Rotor angle, radian
T_m	= Mechanical torque input to rotor
P_s	= Steam power
H	= Inertia constant
T_e	= Electrical torque
P_t, P_b	= Real power output at terminal and busbar
$Q_{,t}, Q_b$	= Reactive power at terminal and busbar
τ_e	= Exciter time constant
τ_g	= Governor valve time constant
τ_b	= Turbine time constant
U_g	= Input to governor
ω	= Angular frequency of the rotor
ω_0	= Angular frequency of the infinite bus
K_d	= Mechanical damping torque coefficient

Synchronous generator

The following equations and assumptions are based on Park's equations for a synchronous generator. Beside that, there are some additional assumptions for simulations.

(i) The effect of change of speed, and the rate of change flux linkage in the stator voltage expressions, is negligible.

(ii) Line and stator resistances, and the effect of transients in the transmission lines are negligible.

(iii) No magnetic saturations.

(iv) The effect of damper windings can be accounted for by adjustment of the damping coefficient T$_d$ in the mechanical equation of motion.

$$v_d = -\Psi_q \quad \text{(a1.1)}$$

$$v_q = \Psi_d \quad \text{(a1.2)}$$

$$v_{fd} = R_{fd}\Psi_q \quad \text{(a1.3)}$$

$$\Psi_d = x_{ad}\, i_{fd} \quad \text{a1.4)}$$

$$\Psi_q = -x_q\, i_q \quad \text{(a1.5)}$$

$$\Psi_{fd} = x_{fd}\, i_{fd} - x_{ad}\, i_d \quad \text{(a1.6)}$$

$$T_e = \Psi_d\, i_q - i_d\Psi_q \quad \text{(a1.6)}$$

$$\delta = \frac{\omega_0}{2H}(T_m - T_e - K_d\,\delta - T_d\,\dot{\delta}\ ) \quad \text{(a1.8)}$$

$$v_t^2 = v_d^2 + v_q^2 \quad \text{(a1.9)}$$

Transmission system

$$v_d = e.\sin\delta - x_e i_q \quad \text{(a1.10)}$$

$$v_q = e.\cos\delta + x_e i_q \quad \text{(a1.11)}$$

where

$$x_e = x_t + x_l \qquad \text{(a1.12)}$$

Prime mover

$$\dot{G}_v = \frac{1}{\tau_g}u_g - \frac{1}{\tau_g}G_v \qquad -5 \leq \dot{G}_V \leq 5 \qquad \text{(a1.13)}$$

$$P_S = K_V\,G_V \qquad 0 \leq \dot{G}_V \leq 1 \qquad \text{(a1.14)}$$

Excitation system

$$\dot{E}_{fd} = \frac{1}{\tau_e}U_e - \frac{1}{\tau_e}E_{fd} \qquad -5 \leq E_{fd} \leq 5 \qquad \text{(a1.15)}$$

Control variables

X	=	(6 x 1) state vector
U	=	(2 x 1) input-control vector
Y	=	(4 x 1) output-measurement vector

System parameters represent a 37.5 MVA generator

MVA	=	37.5
MW	=	30
p.f	=	0.8
KV	=	11.8
r/min	=	3000
x_d	=	0.2 p.u
x_q	=	1.86 p.u
x_{ad}	=	1.86 p.u
x_{fd}	=	2.0 p.u
R_{fd}	=	0.00107 p.u
H	=	5.3 MWs/MVA

T_d	=	0.05
x_t	=	0.345 p.u
x_l	=	0.125 p.u
e	=	1 p.u
τ_e	=	0.1s
τ_q	=	0.1s
τ_b	=	0.5s
K_v	=	1.42

Constants in a.c turbogenerator

Defining

$$x'_d = x_d - \frac{x_{ad}^2}{x_{fd}}$$

$$x'_{dl} = x'_d + x_e$$

$$x_{dl} = x_d + x_e$$

$$x_{ql} = x_q + x_e$$

the constants are

$$K_1 = \frac{e x_{ad}}{x_{fd} x'_{dl}}$$

$$K_2 = e^2 \frac{(x'_d - x_q)}{x'_{dl} \cdot x_{ql}}$$

$$K_3 = \frac{-r_f x_{dl} \omega_0}{x_{fd} x'_{dl}}$$

$$K_4 = \frac{x_{ad} \omega_0 r_f e}{x_{fd} x'_{dl}}$$

$$K_5 = \frac{x_q . e}{x_{ql}}$$

$$K_6 = \frac{x_e \cdot x_{ad}}{x'_{dl} \cdot x_{fd}}$$

$$K_7 = \frac{x'_d \cdot e}{x'_{dl}}$$

APPENDIX 2

RUNGE-KUTTA ALGORITHM

First order non linear equation can be solved easier by numerical method. One numerical method that widely be used is fourth order Runge-Kutta. For a non linear function $x = f(y, t)$, after a very small interval time h the value of x can be represented as follows

$$y_{k+1} = y_k + \tfrac{1}{6}g_1 + \tfrac{1}{3}g_2 + \tfrac{1}{3}g_3 + \tfrac{1}{6}g_4 \qquad \text{(a2.1)}$$

where

$$g_1 = h.f(t_k, y_k)$$

$$g_2 = h.f(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}g_1)$$

$$g_3 = h.f(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}g_2)$$

$$g_4 = h.f(t_k + h, y_k + g_3)$$

Equation a2.1 is used to solved the six non linear equation that simulate a generating system as shown in chapter 4.

APPENDIX 3

Program listing of an open loop test of a synchronous machine using Runge-Kutta procedure

```
program openloop;

Uses
 crt, hgrglb, hgrlow, hgrlin, hgraxi, hgrstr, hgrlgn;

Const

 h  = 0.05;{step of integration}

type
 vector = array[1..6]of real;

Var
 s0,s1,s2,s3,sq,x,ex,pp  : vector;
 Datanum                 : integer;
 u1,u2,vd,vq             : real;
 y1,y2                   : array[1..610] of real;
 OutFile                 : text;

Procedure init;
  begin
  ex[1]  := 1;{delta - rotor angle in radians}
  ex[2]  := 0;{rotor angular velocity}
  ex[3]  := 1.152;{field flux linkage}
  ex[4]  := 2.314;{field voltage}
  ex[5]  := 0.8;{power steam}
  ex[6]  := 0.8;{mechanical torque}
  u1    := 2.495;{input to exciter}
  u2    := 0.634;{input to governor}
end;

procedure set_eqn;

begin
  sq[1] := x[2];
  sq[2] := 29.7169*(x[6]-1.2564*x[3]*sin(x[1])
        +0.9218*sin(x[1])*cos(x[1])- 0.08*x[2]);
  sq[3] := 0.1812*x[4]-0.5623*x[3]+0.4237*cos(x[1]);
  sq[4] := 10.0*(-x[4]+u1);
  sq[5] := 10.0*(-x[5]+1.42*u2);
  sq[6] := 2.0*(-x[6]+x[5]);
end;
```

APPENDIX 2

RUNGE-KUTTA ALGORITHM

First order non linear equation can be solved easier by numerical method. One numerical method that widely be used is fourth order Runge-Kutta. For a non linear function $x = f(y, t)$, after a very small interval time h the value of x can be represented as follows

$$y_{k+1} = y_k + \tfrac{1}{6}g_1 + \tfrac{1}{3}g_2 + \tfrac{1}{3}g_3 + \tfrac{1}{6}g_4 \qquad \text{(a2.1)}$$

where

$$g_1 = h.f(t_k, y_k)$$

$$g_2 = h.f(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}g_1)$$

$$g_3 = h.f(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}g_2)$$

$$g_4 = h.f(t_k + h, y_k + g_3)$$

Equation a2.1 is used to solved the six non linear equation that simulate a generating system as shown in chapter 4.

APPENDIX 3

Program listing of an open loop test of a synchronous machine using Runge-Kutta procedure

```
program openloop;

Uses
 crt, hgrglb, hgrlow, hgrlin, hgraxi, hgrstr, hgrlgn;

Const

 h  = 0.05;{step of integration}

type
 vector = array[1..6]of real;

Var
 s0,s1,s2,s3,sq,x,ex,pp  : vector;
 Datanum                 : integer;
 u1,u2,vd,vq             : real;
 y1,y2                   : array[1..610] of real;
 OutFile                 : text;

Procedure init;
 begin
 ex[1]  := 1;{delta - rotor angle in radians}
 ex[2]  := 0;{rotor angular velocity}
 ex[3]  := 1.152;{field flux linkage}
 ex[4]  := 2.314;{field voltage}
 ex[5]  := 0.8;{power steam}
 ex[6]  := 0.8;{mechanical torque}
 u1     := 2.495;{input to exciter}
 u2     := 0.634;{input to governor}
end;

procedure set_eqn;

begin
 sq[1] := x[2];
 sq[2] := 29.7169*(x[6]-1.2564*x[3]*sin(x[1])
       +0.9218*sin(x[1])*cos(x[1])- 0.08*x[2]);
 sq[3] := 0.1812*x[4]-0.5623*x[3]+0.4237*cos(x[1]);
 sq[4] := 10.0*(-x[4]+u1);
 sq[5] := 10.0*(-x[5]+1.42*u2);
 sq[6] := 2.0*(-x[6]+x[5]);
end;
```

```
procedure Rungkut;
var
 i :integer;
Begin
 y1[k] := 1.2564*ex[3]*sin(ex[1])
        -0.9218*sin(ex[1])*cos(ex[1]); {terminal power}

   vd  := 0.798*sin(ex[1]); {Direct axis voltage}
   vq  := 0.5905*ex[3]+0.3650*cos(ex[1]); {quadrature axis voltage}

 y2[k] := sqrt(vd*vd +vq*vq); {terminal voltage}

 for i := 1 to 6 do
 x[i]  := ex[i];
  set_eqn;
  for i:=1 to 6 do
  s0[i]:=h*sq[i];
 for i:= 1 to 6 do
 x[i]:=ex[i]+0.5*s0[i];
 set_eqn;
 for i := 1 to 6 do
 s1[i]:= h*sq[i];
 for i :=1 to  6 do
 x[i]:=ex[i]+0.5*s1[i];
 set_eqn;
 for i :=1 to 6 do
 s2[i]:=h*sq[i];
 for i :=1 to 6 do
 x[i]:=ex[i]+s2[i];
 set_eqn;
 for i := 1 to 6 do
 s3[i]:= h*sq[i];

 writeln(OutFile,Datanum:4,' ',y1[k]:2:5,' ',y2[k]:2:5,' ',ex[1]:2:5,' ',ex[5]:2:5,' ');
 writeln(Datanum:4,' ',y1[k]:2:5,' ',y2[k]:2:5,' ',ex[1]:2:5,' ',ex[5]:2:5,' ');
for i:=1 to 6 do
 begin
  ex[i]:=ex[i]+(1/6)*(s0[i]+2*s1[i]+2*s2[i]+s3[i]);
 end;
end;

{main program}
begin
init;
Datanum:=1;
assign(OutFile,'openloop.dat');
rewrite(OutFile);
repeat
rungkut;
```

```
Datanum:=Datanum+1;
until k>600;
close(OutFile);
readln;
END.
```

Appendix 4

Turbo pascal program for adaptive controller using pole-zero method without integrator.

```
program PZ;
{$N+,E+}

  uses
    crt, pc30io1, rtplot1;



  const
    plots = true;

    NPoints = 500;
    npar = 4;
    noff = 6;
    n = 4;

  type
    vec1 = array[1..npar] of real;
    vec2 = array[1..noff] of real;
    Datarray = array[1..NPoints] of real;

  var
    delta, a1, a2, b1, b2, m2: real;
    s0, s1, r1, t0, t1: real;
    wk, wk1, wk2, uk, uk1, uk2, yp, yp1, yp2: real;
    ym, ym1, ym2: real;
    i, count: integer;
    p_out, DataNum, P_command, p_inpt, P_model: Datarray;
    theta, fi, diagn: vec1;
    offdiag: vec2;
    answer: char;
    answer1: char;
    outfile: text;
    outname: string;


  procedure InitIdent (uk1, uk2, yp1, yp2: real; var theta, diagn, fi: vec1; offdiag:
vec2);

    var
      P0, theta0: real;
      i: integer;
  begin
    for i := 1 to npar do
```

```
            begin
                theta[i] := 1;
                diagn[i] := 100;
            end;{i}
{form fi}
        fi[1] := uk2;
        fi[2] := uk1;
        fi[3] := yp2;
        fi[4] := yp1;

        for i := 1 to noff do
            offdiag[i] := 0.0;
    end;{of InitIdent}

    procedure ident (yp, delta: real; var theta, fi, diagn: vec1; var offdiag: vec2);
        var
            kf, ku, i, j: integer;
            perr, fj, vj, alphaj, ajlast, pj, w: real;
            k: vec1;

    begin
        perr := yp;
        for i := 1 to n do
            perr := perr - theta[i] * fi[i];
     (*Calculate gain and covariance using U-D method*)
        fj := fi[1];
        vj := diagn[1] * fj;
        k[1] := vj;
        alphaj := 1.0 + vj * fj;
        diagn[1] := diagn[1] / alphaj / delta;
        if n > 1 then
            begin
                kf := 0;
                ku := 0;
                for j := 2 to n do
                    begin
                        fj := fi[j];
                        for i := 1 to j - 1 do
                            begin (*f=fi*u*)
                                kf := kf + 1;
                                fj := fj + fi[i] * offdiag[kf];
                            end;(*i*)
                        vj := fj * diagn[j]; (*v = D*f*)
                        k[j] := vj;
                        ajlast := alphaj;
                        alphaj := ajlast + vj * fj;
                        diagn[j] := diagn[j] * ajlast / alphaj / delta;
                        pj := -fj / ajlast;
                        for i := 1 to j - 1 do
                    begin
            (*kj+1 ;=kj+vj*uj*)
           (*uj :=uj+pj*kj*)
                     ku := ku + 1;
                     w := offdiag[ku] + k[i] * pj;
                     k[i] := k[i] + offdiag[ku] * vj;
                     offdiag[ku] := w;
                  end;(*i*)
              end;(*j*)
          end;(*if n>1 then*)
(*update parameter estimates*)
      for i := 1 to n do
          begin
              theta[i] := theta[i] + perr * k[i] / alphaj;
          end;{i}
   end; (*LS*)


   procedure design (a1, a2, b1, b2: real; var s0, s1, r1, t0, t1: real);

   if abs (b2/b1) < 1 then
   begin
      t0 := 0.0621596 / b1;
      t1 := 0.0476 / b1;
      s0 := (-1.33596 - a1) / b1;
      s1 := (0.4493289 - a2) / b1;
      r1 := b2 / b1;
   end;

   else
   begin
      t1:=0.0621596/b1;
      s1:=((0.44932*a1)+(b2*a2)-(-1.33596*a2))/((a2*b1)+(a1*b2)+(b2*b2/b1));
      s0:=-(((0.4492+(b2*s1))/(a2*b1))+1;
      r1=((0.44932-(b2*s1))/a2;
   procedure model (wk1, wk2, ym1, ym2: real; var ym: real);
   begin
      ym := (0.0621596 * wk1) + (0.0476 * wk2) + (1.33596 * ym1) - (0.4493289
              * ym2);
   end;


   procedure action (t0, t1, s0, s1, r1, wk, wk1, yp, yp1, uk1: real; var uk: real);
   begin
      if abs (b2/b1) < 1 then
          uk := (t0 * wk) + (t1 * wk1) - (s0 * yp) - (s1 * yp1) - (r1 * uk1);
      else
          uk:=((t1*wk)-(s0*yk)-(s1*yk1))/r1;
   end;
```

```
{main program}
begin
   randomize;
{initial value}
   count := 50;
   wk := 1;
   wk1 := 0.0;
   wk2 := 0.0;
   uk1 := 0.0;
   uk2 := 0.0;
   yp1 := 0.0;
   yp2 := 0.0;
   ym1 := 0.0;
   ym2 := 0.0;
   delta := 0.98;


   InitIdent(uk1, uk2, yp1, yp2, theta, diagn, fi, offdiag);
   if plots then
      pltinit;

   for i := 1 to NPoints do
      begin
  {create wk}
         count := count - 1;
         if count = 0 then
            begin
               wk := -wk;
               count := 50;
            end;{of wk}

{calculate plant model (YM)}
         model(wk1, wk2, ym1, ym2, ym);


{calculate plant output (YP)}
         yp := rdadc(13);

{Identification of plant parameters}
         ident(yp, delta, theta, fi, diagn, offdiag);

         b1 := theta[1];
         b2 := theta[2];
         a1 := -theta[3];
         a2 := -theta[4];
```

```
{calculate controller parameters}
          design(a1, a2, b1, b2, s0, s1, r1, t0, t1);

{calculate plant input}
          action(t0, t1, s0, s1, r1, wk, wk1, yp, yp1, uk1, uk);
          if (uk > 8) then
             uk := 8 + (random - 0.5) * 0.1;
          if (uk < -8) then
             uk := -8 + (random - 0.5) * 0.1;
{write new fi}
          fi[2] := fi[1];
          fi[4] := fi[3];
          fi[1] := uk;
          fi[3] := yp;
{read data and create array for plotting }
          p_command[i] := wk;
          p_out[i] := yp;
          P_model[i] := ym;
          p_inpt[i] := uk;
          DataNum[i] := i;
{read new data}
          uk2 := uk1;
          uk1 := uk;
          yp2 := yp1;
          yp1 := yp;
          wk2 := wk1;
          wk1 := wk;
          ym2 := ym1;
          ym1 := ym;
          endsmpl;
          dacout(0, uk);
          if plots then
             pltupdate(wk, ym, yp, uk);
      end;{iteration}

   readln;
   dacout(0, 0);
   if plots then
      pltclose;
   writeln('save ? y(es) or n(o)= ');
   readln(answer);
   if (answer = 'y') or (answer = 'Y') then
      begin
          write('name for outfile = ');
          readln(outname);
          assign(outfile, outname);
          rewrite(outfile);
          for i := 1 to Npoints do
```

```
            writeln(outfile, i : 4, ' ', p_out[i] : 8 : 5, ' ', p_model[i] : 8 : 5, ' ',
p_command[i] : 8 : 5, ' ', p_inpt[i] : 8 : 5);
        writeln(outfile, i : 4, ' ', p_out[i] : 8 : 5, ' ', p_model[i] : 8 : 5, ' ',
p_command[i] : 8 : 5, ' ', p_inpt[i] : 8 : 5);
        close(outfile);
    end;
end.
```

APPENDIX 5

Turbo pascal program for PC 30.

```
unit pc30io1;
{$N+,E+}
interface

   uses
      pc30;

   procedure dacout (chan: integer; val: double);
   procedure dout (chan, val: integer);
   procedure endsmpl;
   procedure initpc30 (sampletime: integer);
   function rdadc (chan: integer): double;
   function rdcntr: integer;

implementation

   procedure pc30error;
   begin
      halt;
   end;

   procedure dacout;
      var
         err: integer;
   begin
      if (val < -10.0) then
         val := -10.0;
      if (val > 10.0) then
         val := 10.0;
      val := (10.0 - val) / 20 * 4095;
      err := da_out(chan, trunc(val));
      if (err <> ok_30) then
         begin
            writeln('Pc30-error  Da_out');
            Pc30error;
         end;
   end;


   procedure dout;
   begin
      d_out(chan, val);
   end;
```

```
procedure endsmpl;
   var
      old_val, new_val: word;
begin
   dout(0, 0);
   old_val := cntr_read;
   new_val := cntr_read;
   while (new_val <= old_val) do
      begin
         old_val := new_val;
         new_val := cntr_read;
      end;
   dout(0, 1);
end;


procedure initpc30;
   const
      two_MHz_to_10KHz = 200;
      ten_KHz_to_1KHz = 10;
      pc30_base = $700;
   var
      err: integer;
      ws: word;
begin
   set_base(pc30_base);
   err := diag;
   if (err <> ok_30) then
      begin
         writeln('Pc30 board not found or bad.');
         Pc30error;
      end;
   if (err = ok_30) then
      begin
         ad_prescaler(two_MHz_to_10KHz);
         ad_clock(ten_KHz_to_1KHz);
         cntr_cfg(2);
         ws := word(sampletime);
         cntr_write(ws);
         d_mode(0, 0, 1);  { set third digital i/o's to input }
         dacout(0, 0);
      end;
end;


function rdadc;
   var
      err, in_val: integer;
begin
```

```
        err := ad_in(chan, in_val);
        if (err <> ok_30) then
            begin
                writeln('Pc30-error rdadc.');
                Pc30error;
            end;
        if (err = ok_30) then
            rdadc := (in_val - 2047.5) * 10 / 4095;
    end;


    function rdcntr;
        var
            val: word;
    begin
        val := cntr_read;
        rdcntr := integer(val);
    end;

begin
    initpc30(10);
end.
```

APPENDIX 6

Turbo pascal program for plotting.

```
program plt;

    uses
       hgrglb, hgrlow, hgrlin, hgraxi, hgrstr, screen, crt, hgrlgn, hgrr3d;


    const
       Npoints = 500;

    type
       Datarray = array[1..Npoints] of real;
    var
       Input_file: text;
       DataNum, p_out, p_model, p_command, p_inpt: datarray;
       i: integer;
       idevice: integer;
begin
    assign(Input_file, 'a:\fixt1.dat');
    reset(Input_file);
    I := 1;
    while not eof(Input_file) do
       begin
          readln(Input_file, DataNum[I], p_out[I], p_model[I], p_command[i],
p_inpt[i]);
          I := I + 1;
       end;
    close(Input_file);
    writeln('enter 0=screen, 2=plotter, 3=printer');
    readln(idevice);
{Plotting WK, YP & YM }
    INIPLT(idevice, normal, 1.0);
{graph 1}
    graphboundary(2000, 9000, 4000, 6000);
    setlegend(2200, 6800, 550);
    scale(0.0, Npoints, -4.0, 3.0);
    setfont(bold, false);
    axis(100.0, '10.0', '', 2, 2, '10.0', 'Value', 2);
    polyline(DataNum, P_command, Npoints, 2, 0, 0, 0, 2);
    writelegend('Set point', 2, 2, 2, 0, 2);
    polyline(DataNum, p_out, Npoints, 4, 0, 0, 0, 0);
    writelegend('Terminal voltage', 4, 2, 2, 0, 0);
{graph 2}
    graphboundary(2000, 9000, 1000, 3000);
    setlegend(2200, 3200, 550);
    scale(0.0, Npoints, -20.0, 20.0);
    setfont(bold, false);
    axis(100.0, '10.0', 'Time', 2, 10.0, '10.1', 'Value', 2);
    polyline(DataNum, P_inpt, Npoints, 2, 0, 0, 0, 0);
    writelegend('Exciter input', 2, 0, 2, 0, 0);
    endplt;
end.
```

APPENDIX 7

Turbo pascal program for adaptive controller using pole-zero method with integrator.

```
program PZ_INTEG;
{$N+,E+}

  uses
    crt, pc30io1, rtplot1;
  const
    plots = true;
    Npoints = 500;
    npar = 4;
    noff = 6;
    n = 4;




  type
    vec1 = array[1..npar] of real;
    vec2 = array[1..noff] of real;
    Datarray = array[1..NPoints] of real;

  var
    delta, a1, a2, b1, b2: real;
    s0, s1, s2, r0, r1, t0, t1, t2: real;
    wk, wk1, wk2, wk3, uk, uk1, uk2, yp1, yp2, yp: real;
    ym, ym1, ym2, ym3: real;
    i, count: integer;
    p_out, DataNum, P_command, p_inpt, P_model: Datarray;
    theta, fi, diagn: vec1;
    offdiag: vec2;
    answer: char;
    answer1: char;
    outfile: text;
    outname: string;


{Initialise parameter identifications}
  procedure InitIdent (uk1, uk2, yp1, yp2: real; var theta, diagn, fi: vec1; offdiag:
vec2);
    var
      P0, theta0: real;
      i: integer;

  begin
    for i := 1 to npar do
      begin
```

```
                theta[i] := 1;
                diagn[i] := 100;
            end;{i}

{form fi}
        fi[1] := uk1;
        fi[2] := uk2;
        fi[3] := yp1;
        fi[4] := yp2;

        for i := 1 to noff do
            offdiag[i] := 0.0;
    end;{of InitIdent}

{Identify parameters}
    procedure ident (yp, delta: real; var theta, fi, diagn: vec1; var offdiag: vec2);
        var
            kf, ku, i, j: integer;
            perr, fj, vj, alphaj, ajlast, pj, w: real;
            k: vec1;

    begin
        perr := yp;
        for i := 1 to n do
            perr := perr - theta[i] * fi[i];
     (*Calculate gain and covariance using U-D method*)
        fj := fi[1];
        vj := diagn[1] * fj;
        k[1] := vj;
        alphaj := 1.0 + vj * fj;
        diagn[1] := diagn[1] / alphaj / delta;
        if n > 1 then
            begin
                kf := 0;
                ku := 0;
                for j := 2 to n do
                    begin
                        fj := fi[j];
                        for i := 1 to j - 1 do
                            begin (*f=fi*u*)
                                kf := kf + 1;
                                fj := fj + fi[i] * offdiag[kf];
                            end;(*i*)
                        vj := fj * diagn[j]; (*v = D*f*)
                        k[j] := vj;
                        ajlast := alphaj;
                        alphaj := ajlast + vj * fj;
                        diagn[j] := diagn[j] * ajlast / alphaj / delta;
                        pj := -fj / ajlast;
                    for i := 1 to j - 1 do
                        begin
               (*kj+1 ;=kj+vj*uj*)
               (*uj :=uj+pj*kj*)
                          ku := ku + 1;
                          w := offdiag[ku] + k[i] * pj;
                          k[i] := k[i] + offdiag[ku] * vj;
                          offdiag[ku] := w;
                        end;(*i*)
                 end;(*j*)
             end;(*if n>1 then*)
         (*update parameter estimates*)
         for i := 1 to n do
             begin
                 theta[i] := theta[i] + perr * k[i] / alphaj;
             end;{i}

     end; (*LS*)

  {Determine controller parameters}
     procedure design (a1, a2, b1, b2: real; var s0, s1, s2, r0, r1, t0, t1, t2: real);
     begin

        t0 := 0.00800497 / b1;
        t1 := 0.0237882 / b1;
        t2 := 0.0043925736 / b1;
        s0 := (-2.009887787 + 1 - a1) / b1;
        s1 := (1.347268086 + a1 - a2) / b1;
        s2 := (-0.30119422 + a2) / b1;
        r0 := (b2 / b1 - 1);
        r1 := -b2 / b1;

     end;{of design}


     procedure model (wk1, wk2, wk3, ym1, ym2, ym3: real; var ym: real);
     begin

        ym := (0.00800497 * wk1) + (0.0237882 * wk2) + (0.0043925736 * wk3) +
  (2.009887787 * ym1) - (1.347268086 * ym2) + (0.30119422 * ym3);
     end;


     procedure action (t0, t1, t2, s0, s1, s2, r0, r1, wk, wk1, wk2, yp, yp1, yp2, uk1,
  uk2: real; var uk: real);
     begin
        uk := (t0 * wk) + (t1 * wk1) + (t2 * wk2) - (s0 * yp) - (s1 * yp1) - (s2 * yp2)
  - (r0 * uk1) - (r1 * uk2);
     end;
```

```
{main program}
begin

{initial value}
   randomize;
   count := 50;
   wk := 1;
   wk1 := 0;
   wk2 := 0;
   wk3 := 0;
   uk1 := 0;
   uk2 := 0;
   yp1 := 0;
   yp2 := 0;
   ym1 := 0;
   ym2 := 0;
   ym3 := 0;
   delta := 0.98;


   InitIdent(uk1, uk2, yp1, yp2, theta, diagn, fi, offdiag);
   if plots then
      pltinit;

   for i := 1 to NPoints do
      begin
  {create wk}
         count := count - 1;
         if count = 0 then
            begin
               wk := wk + random;
      {wk:=-wk;}
               count := 50;
            end;{of wk}
         if (wk > 2) then
            wk := -1;

{calculate plant model (YM)}
         model(wk1, wk2, wk3, ym1, ym2, ym3, ym);

{ plant output (YP)}
         yp := rdadc(13)

{Identification of plant parameters}
         ident(yp, delta, theta, fi, diagn, offdiag);

         b1 := theta[1];
```

```
        b2 := theta[2];
        a1 := -theta[3];
        a2 := -theta[4];

{calculate controller parameters}
        design(a1, a2, b1, b2, s0, s1, s2, r0, r1, t0, t1, t2);

{calculate plant input}

        action(t0, t1, t2, s0, s1, s2, r0, r1, wk, wk1, wk2, yp, yp1, yp2, uk1, uk2,
uk);
        if (uk > 8) then
           uk := 8 + (random - 0.5) * 0.1;
        if (uk < -8) then
           uk := -8 + (random - 0.5) * 0.1;


 {write new fi}
        fi[2] := fi[1];
        fi[4] := fi[3];
        fi[1] := uk;
        fi[3] := yp;


 {read data and create array for plotting }
        p_command[i] := wk;
        p_out[i] := yp;
        P_model[i] := ym;
        p_inpt[i] := uk;
        DataNum[i] := i;

 {read new data}
        uk2 := uk1;
        uk1 := uk;
        yp2 := yp1;
        yp1 := yp;
        wk3 := wk2;
        wk2 := wk1;
        wk1 := wk;
        ym3 := ym2;
        ym2 := ym1;
        ym1 := ym;

        endsmpl;
        dacout(0, uk);
        if plots then
           pltupdate(wk, ym, yp, uk);
     end;{iteration}
   readln;
```

```
    dacout(0, 0);
    if plots then
       pltclose;
    writeln('save ? y(es) or n(o)= ');
    readln(answer);
    if (answer = 'y') or (answer = 'Y') then
       begin
           write('name for outfile = ');
           readln(outname);
           assign(outfile, outname);
           rewrite(outfile);
           for i := 1 to Npoints do
               writeln(outfile, i : 4, '  ', p_out[i] : 8 : 5, '  ', p_model[i] : 8 : 5, '  ',
p_command[i] : 8 : 5, '  ', p_inpt[i] : 8 : 5);
           close(outfile);
       end;{of i}
end.
```

APPENDIX 8

Turbo pascal program for Quinn Curtis.

```
unit rtplot1;
interface
 {$m 32000,0,655360}
 {$N+,E+}

 {***** IMPORTANT *****}
 {The default directory for BGI files & fonts is c:\dosapp\tp\bgi}
 {ie DEFAULTBGIDIR in the file rtstdhdr.pas ='c:\dosapp\tp\bgi'}

   uses
      Graph, rtstdhdr, rtgsubs, rtgraph;

   procedure pltinit;
   procedure pltupdate (r, d, y, u: RealType);
   procedure pltclose;

implementation


   var
      lc, lf: rtintarraytype;   {line colors & styles}
      tags: tagarraytype;    {tag names}
      title: titletype;    {scroll graph title}
      units: tagtype;        {scroll graph units}
      ratchf: BOOLEAN;      {staircase method}
   {These variables are described in detail in the user manual}
      timeint, sampleint, miny, maxy: RealType;
      rt, lalarm, halarm, stpnt: RealType;
      nt, grid, xdecs, ydecs, updatenumber: INTEGER;

   procedure pltinit;
      var
         i: integer;
   begin
  { INITIALIZE THE GRAPHICS ADAPTER,}

{   SET UP 2 REAL TIME WINDOWS}
      rtinitgraphics(defaultbgidir, 2, 1);

  { Size windows }
      rtsetpercentwindow(rtstat[0], 0.01, 0.01, 0.99, 0.49);
      rtsetpercentwindow(rtstat[1], 0.01, 0.50, 0.99, 0.99);
```

```
        lc[0] := 14;
        lf[0] := 0;
        lc[1] := 13;
        lf[1] := 0;
        lc[2] := 12;
        lf[1] := 0;

    { set up individual parameters for each plot seperately }

    {graph 1}
        timeint := 500;
        sampleint := 1;
        miny := -3;
        maxy := 3;
        nt := 3;
        rt := 0;
        grid := 0;
        ratchf := false;
        stpnt := 0;
        lalarm := -0.0;
        halarm := 0.0;
        title := 'Input, Desired & Actual outputs';
        units := 'voltage';
        tags[0] := 'Input';
        tags[1] := 'Desired Response';
        tags[2] := 'Actual Response';
        rtinitwindowcolors(rtstat[0], 7, 0, 1, 4, 15, 15, 15);
        rtsetupscrollgraph(rtstat[0], timeint, sampleint, miny, maxy, rt, nt, grid,
lalarm, halarm, stpnt, 2, 0, title, units, tags, lc, lf, ratchf);
        rtborderwindow(rtstat[0], 15);

     {graph 2}
        timeint := 500;
        sampleint := 1;
        miny := -10;
        maxy := 10;
        nt := 1;
        rt := 0;
        grid := 0;
        ratchf := true;
        stpnt := 0;
        lalarm := 0;
        halarm := 0;
        title := 'Control Action';
        units := 'units';
        tags[0] := '';
        rtinitwindowcolors(rtstat[1], 7, 0, 1, 4, 15, 15, 15);
        rtsetupscrollgraph(rtstat[1], timeint, sampleint, miny, maxy, rt, nt, grid,
lalarm, halarm, stpnt, 2, 0, title, units, tags, lc, lf, ratchf);
```

```
    rtborderwindow(rtstat[1], 15);
  end;

  procedure pltupdate;
    var
       yvalues: rtvaluearraytype; {hold current R-T value}
       upd: boolean;
  begin
    upd := (updatenumber = 100);
    yvalues[0] := r;
    yvalues[1] := d;
    yvalues[2] := y;
    rtupdatescrollgraph(rtstat[0], yvalues);
    if upd then
       rtdrawalarmlines(rtstat[0]);

    yvalues[0] := u;
    rtupdatescrollgraph(rtstat[1], yvalues);
    if upd then
       rtdrawalarmlines(rtstat[1]);

    if upd then
       updatenumber := 0
    else
       updatenumber := updatenumber + 1;

  end;

  procedure pltclose;
  begin
    rtclosegraphics(1);
  end;

end.
```

APPENDIX 9

Turbo Pascal Program for simulations

```
program ADP_NO_INTEGRATION;



Uses
  hgrglb, hgrlow, hgrlin, hgraxi, hgrstr,
    screen, crt, hgrlgn;



Const

 NPoints=800;
 npar=4;
 noff=6;
 n=4;

Type
 vec1 = array[1..npar] of real;
 vec2 = array[1..noff] of real;
 Datarray = array[1..NPoints] of real;

var
 delta,a1,a2,b1,b2,m2:real;
 s0,s1,r1,t0,t1 :real;
 wk,wk1,wk2,uk,uk1,uk2,yp,yp1,yp2 : real;
 ym,ym1,ym2 : real;
 change_point1,change_point2,i,j,count,idevice: integer;
 p_out,DataNum,P_command,p_inpt,P_model :Datarray;
 theta,fi,diagn :vec1;
 offdiag :vec2;
 answer:char;
 answer1:char;
 outfile:text;
 outname:string;


procedure Plant( uk1,uk2,yp1,yp2 :real; var yp : real);



const
  change_point1 = 125;
  change_point2 = 375;
  begin
    if i < change_point1 then
  yp :=(0.0621596*uk1)+(0.0476*uk2)+(1.33596*yp1)-(0.4493289*yp2)+1e-10
      else if (change_point1 < i) and (i <change_point2) then
       yp := (0.0993*uk1) + (0.098*uk2)+(1.9213*yp1)-(0.9607*yp2)+1e-10
     else
       yp := (0.1215*uk1) +(0.0929*uk2)+(1.2348*yp1)-(0.4493*yp2)+1e-10;

{change_point1 := 125;
 change_point2 := 275;
 begin
   if j < change_point1 then

yp := (0.286165*uk1) +(0.146527*uk2)+(0.702897*yp1)
```

```
              -(0.13533527*yp2)+1e-1
        else if (change_point1 < j) and (j <change_point2) then
    yp := (0.125192*uk1) + (0.0838624*uk2)+(1.09213972*yp1)
          -(0.301194206*yp2)+1e-1
       else
            yp := (0.0410075*uk1) +(0.0225053*uk2)+(1.4742344*yp1)
            -(0.548811*yp2)+1e-10; }


     end;
       {  yp :=(0.0621596*uk1) + (0.0476*uk2) + (1.33596*yp1)
        - (0.4493289*yp2);
     end;}


   procedure InitIdent(uk1,uk2,yp1,yp2 :real;
                var theta,diagn,fi :vec1;offdiag: vec2);



   var
    P0,theta0 :real;
    i:integer;
   begin
     for i:=1 to npar do
      begin
       theta[i] :=1;
       diagn[i]:=100;
      end;{i}

    {form fi}
    fi[1]:=uk2;
```

```
 fi[2]:=uk1;
 fi[3]:=yp2;
 fi[4]:=yp1;

 for i:=1 to noff do offdiag[i] :=0.0;
end;{of InitIdent}

procedure ident(yp,delta:real; var theta,fi,diagn:vec1; var offdiag:vec2);
var
 kf,ku,i,j : integer;
 perr,fj,vj,alphaj,ajlast,pj,w :real;
 k:vec1;

begin
    perr:=yp;
     for i:=1 to n do perr:=perr-theta[i]*fi[i];
    (*Calculate gain and covariance using U-D method*)
    fj :=fi[1];
    vj :=diagn[1]*fj;
    k[1] :=vj;
    alphaj:=1.0+vj*fj;
    diagn[1] :=diagn[1]/alphaj/delta;
    if n>1 then
    begin
      kf:=0;
      ku:=0;
      for j:=2 to n do
        begin
        fj :=fi[j];
```

```
          for i :=1 to j-1 do
          begin (*f=fi*u*)
            kf := kf+1;
            fj:=fj +fi[i]*offdiag[kf];
          end;(*i*)
          vj :=fj*diagn[j]; (*v = D*f*)
          k[j] :=vj;
          ajlast := alphaj;
          alphaj :=ajlast+vj*fj;
          diagn[j] := diagn[j]*ajlast/alphaj/delta;
          pj:=-fj/ajlast;
          for i :=1 to j-1 do
          begin
            (*kj+1 ;=kj+vj*uj*)
            (*uj :=uj+pj*kj*)
            ku :=ku+1;
            w:=offdiag[ku]+k[i]*pj;
            k[i] :=k[i]+offdiag[ku]*vj;
            offdiag[ku] :=w;
          end;(*i*)
        end;(*j*)
      end;(*if n>1 then*)
       (*update parameter estimates*)
       for i :=1 to n do
         begin
          theta[i] := theta[i]+perr*k[i]/alphaj;
         end;{i}
  end; (*LS*)
```

```
Procedure design(a1,a2,b1,b2 :real; var s0,s1,r1,t0,t1 :real);

 begin

  { t0:=0.0954946/b1;
   t1:=0.073072/b1;
   s0:=(-1.2807623-a1)/b1;
   s1:=(0.4493289-a2)/b2;
   r1:=b2/b1;
   t0:=0.0410075/b1;
   t1:=0.0225053/b1;
   s0:=(-1.4742344-a1)/b1;
   s1:=(0.5488116-a2)/b2;
   r1:=b2/b1;}
   t0:=0.0621596/b1;
   t1:=0.0476/b1;
   s0:=(-1.33596-a1)/b1;
   s1:=(0.4493289-a2)/b1;
   r1:=b2/b1;
 end;
Procedure model(wk1,wk2,ym1,ym2: real; var ym:real);
 begin
  { ym := (0.0954946*wk1)+(0.073072*wk2)+(1.2807623)-(0.5488116*ym2);
   ym := (0.445934*wk1)+(0.1626039*wk2)+(0.44125*ym1)-(0.049787*ym2);}
 ym := (0.0621596*wk1)+(0.0476*wk2)
  +(1.33596*ym1)-(0.4493289*ym2);
 end;
Procedure action(t0,t1,s0,s1,r1,wk,wk1,yp,yp1,uk1:real; var uk :real);
```

```
  begin
   uk:= (t0*wk) + (t1*wk1) - (s0*yp) - (s1*yp1) - (r1*uk1);
  end;



{main program}
begin


{initial value}
 count:=50;
 wk:=1;
 wk1:=0.0;
 wk2:=0.0;
 uk1:=0.0;
 uk2:=0.0;
 yp1:=0.0;
 yp2:=0.0;
 ym1:=0.0;
 ym2:=0.0;
 delta:=0.98;
 InitIdent(uk1,uk2,yp1,yp2,theta,diagn,fi,offdiag);
{  if plots then pltinit;
 }
 for i:=1 to NPoints do
  begin


  {create wk}
 count:=count-1;
 if count=0 then
   begin
     wk:=-wk;
     count:=50;
    end;{of wk}

 {calculate plant model (YM)}
 model(wk1,wk2,ym1,ym2,ym);



 {calculate plant output (YP)}
 {if simulate then} plant(uk1,uk2,yp1,yp2,yp);
{ if (yp>18) then yp:=18+(random-0.5)*0.1;
 if (yp<-18) then yp:=-18+(random-0.5)*0.1; }
{ else yp:=rdadc(13);}
 {Identification of plant parameters}
 ident(yp,delta,theta,fi,diagn,offdiag);

 b1:=theta[1];
 b2:=theta[2];
 a1:=-theta[3];
 a2:=-theta[4];

 {calculate controller parameters}
 design(a1,a2,b1,b2,s0,s1,r1,t0,t1);

 {calculate plant input}
 action(t0,t1,s0,s1,r1,wk,wk1,yp,yp1,uk1,uk);
```

```
if (uk>8) then uk:=8{+(random-0.5)*0.1};
if (uk<-8) then uk:=-8{+(random-0.5)*0.1};

{write new fi}
fi[2]:=fi[1];
fi[4]:=fi[3];
fi[1]:= uk;
fi[3]:=yp;



{read data and create array for plotting }
p_command[i] :=wk;
p_out[i] :=yp;
P_model[i] :=ym;
p_inpt[i] :=uk;
DataNum[i]:=i;

{read new data}
uk2:=uk1;
uk1:=uk;
yp2:=yp1;
yp1:=yp;
wk2:=wk1;
wk1:=wk;
ym2:=ym1;
ym1:=ym;
{ if not(simulate) then endsmpl;
if not(simulate) then dacout(0,uk);
```

```
  if plots then pltupdate(wk,ym,yp,uk);
  if not(plots) then
   writeln('a1: ',a1:6:4,' a2: ',a2:6:4,' b1: ',b1:6:4,' b2: ',b2:6:4);
 }
 end;{iteration}
{  readln;
 if not(simulate) then dacout(0,0);
 if plots then pltclose;
 writeln('save ? y(es) or n(o)= ');
 readln(answer);
 if (answer='y') or (answer='Y') then
 begin
 write('name for outfile = ');
 readln(outname);
 assign(outfile,outname);
 rewrite(outfile);
 for i:=1 to Npoints do
 writeln(outfile,i:4,'  ',p_out[i]:8:5,'  ',p_model[i]:8:5,'  '
 ,p_command[i]:8:5,'  ',p_inpt[i]:8:5);
 writeln(outfile,i:4,'  ',p_out[i]:8:5,'  ',p_model[i]:8:5,'  '
 ,p_command[i]:8:5,'  ',p_inpt[i]:8:5);
 close(outfile);
 end;}


 {Plotting WK, YP & YM }
 writeln('enter 0=screen, 2=plotter, 3=printer');
  readln(idevice);
  INIPLT(idevice, normal, 1.0);
 { iniplt(0, normal, 1);}
graphboundary(2000, 6000, 4000, 6500);
 setlegend(2200, 6950, 550);
 scale(0.0, Npoints, -3.0,3.0);
 setfont(bold,false);
 axis(200.0,'10.0','',2,2.0, '10.0', 'Value',2);
 polyline(DataNum, P_command, Npoints, 2, 0, 0, 0, 2);
 writelegend('WK' , 2, 0, 2, 0, 2);
 polyline(DataNum, p_out, Npoints, 0, 0, 0, 0, 0);
 writelegend('YP' , 0,0 ,2, 0, 0);
{  polyline(DataNum, p_model, Npoints, 5, 0, 0, 0, 0);
 writelegend('YM' , 5 ,0 ,2, 0, 0);}


 graphboundary(2000, 6000, 1000, 3000);
 setlegend(2200,3200, 550);
 scale(0.0, Npoints, -9.0,9.0);
 setfont(bold,false);
 axis(100.0,'10.0','Data Number',2,3.0, '10.1', 'Value',2);
 polyline(DataNum,P_inpt, Npoints, 2, 0, 0, 0, 0);
 writelegend('uk' , 2, 0, 2, 0, 0);
 endplt;
END.
```

REFERENCES

1. Chalam,V.V., *Adaptive control systems techniques and applications*, in Electrical Engineering and Electronics/39, Marcel Dekker Inc,(1987).

2. Egardt, B., *Stability of adaptive controllers,* Springer-Verlag, Berlin (1979).

3. Astrom, K, J., *Design principles for self-tuning regulators*, in Methods and Applications of Adaptive Control, (Ubenhauen, ed), Springer-Verlag, Berlin pp.1-20,(1980).

4. Iserman, R., *Parameter adaptive control algorithms-a tutorial,* Automatica, 18: pp 513-528,(1982).

5. Astrom, K. J., Wittenmark, B., *Self tuning controllers based on pole-zero placement,* Proc. IEE, 127, Part D: pp 120-130,(1980).

6 Astrom, K. J., Wittenmark, B., *Computer-controlled systems theory and design*, Prentice-Hall, (1990).

7 Astrom, K. J., Wittenmark, B., *Practical issue in the implementation of self-tuning control,* Automatica, 20: pp 595-605,(1984).

8 The',G., *Advance controll lecture notes*, University of Tasmania, (1993).

9 Astrom, K.,Hagglund, T., *Automatic tuning of simple regulator with specifications on pphase and amplitude margins,* Automatica, 20: pp 645-651,(1984).

BIBLIOGRAPHY

1 Clarke, D., W., *Introduction to self-tuning controllers*, in Self-tuning and adaptive control(Harris, C., Billings, S.,A), Peter peregrinus,(1981).

2 Clarke, D., W., *Implementation of self tuning controllers,* in Self-tuning and adaptive control(Harris, C., Billings, S.,A), Peter peregrinus,(1981).

3. Astrom, K. J., Wittenmark, B.,*On self tuning regulators,* Automatica, 9: pp 185-199,(1973).

4 Jacobs, O., L., R., *Introduction to adaptive control,*in Self-tuning and adaptive control(Harris, C., Billings, S.,A), Peter peregrinus,(1981).

5 Clarke, D., W., Gawthrop, P., J., *Self-tuning control,* Proc. IEE, 126 : 929-934(1975).

6. Astrom, K. J., Wittenmark, B., *Simple self tuning controllers,*in Methods and Applications of Adaptive Control, (Ubenhauen, ed), Springer-Verlag, Berlin pp.1-20,(1980).

7 Peterka, V., *Predictor-based self-tuning control,* Automatica, 20: pp 39-50,(1984).