VARIABLE STRUCTURE CONTROL SYSTEMS

A thesis submitted to the department of Electrical and Electronic Engineering, University of Cape Town, in partial fulfilment of the requirements for an MSc. degree in Electrical and Electronic Engineering.

by

David W. Ginsberg

December 1989

ACKNOWLEDGMENTS

I would like to thank my supervisor, Professor Martin Braae, for his expert advice and encouragement throughout my MSc.

Thanks to the C.S.I.R. for their financial assistance.

A special word of thanks to my parents, family and friends for all their support and assistance.

ABSTRACT

The primary aims of this thesis, is to provide a body of knowledge on variable structure system theory and to apply the developed design concepts to control practical systems.

It introduces the concept of a structure. The main aim in designing variable structure controllers, is to synthesize a variable structure system from two or more single structure systems, in such a way that the ensuing system out - performs its component structures.

When a sliding mode is defined, the ensuing closed loop behaviour of the system is invariant to plant parameter changes and external disturbances.

A variable structure controller was designed for a servo motor and successfuly applied to the system.

In practice, the phase plane representative point does not slide at infinite frequency with infinitesmal amplitude along the switching surface(s).

Thus, the concept of a quasi - sliding regime was introduced.

For high performance system specifications, the phase plane representative point could cycle about the origin. In some instances, sliding could be lost.

For high speed applications, a novel design modification ensured that the system did not lose sliding. In addition, the controller could track a rapidly changing setpoint. Successful results support the developed theory.

Multivariable systems of variable structure were investigated and practical design methods, which ensured sliding for multi - dimensional systems, were presented. All the design concepts were illustrated by way of example.

Variable structure system theory was applied to flotation systems. A two - tank flotation rig simulator is presented as the main application of the thesis.

Control of the liquid level of a single tank is achieved. This design strategy assumes a constant liquid level in the other tank.

Although the level of the single tank system is successfully controlled, it was seen that the level in the other tank does not remain constant. Thus, multivariable design techniques were employed to control the levels of both tanks simultaneously.

The implementation of a multivariable control scheme to control the two - tank system involved overcoming a number of problems:

1. The control scheme had to facilitate non - zero steady state characteristics.

2. A state observer had to be designed.

3. In order to implement a system setpoint, a novel control scheme was proposed.

Two control schemes were tested on the flotation rig.

The first scheme stabilised the unstable open loop system and maintained the levels of the two tanks at some specified values.

This scheme was successfully used to stabilise the system, reject external disturbances and be insensitive to plant parameter changes.

The second scheme achieved all that the first scheme did, but incorporated a setpoint facility for the levels of the two tanks.

It is concluded that variable structure controllers are an attractive means of control. The controllers ensure that the closed loop system:

1. has a high speed transient response.

2. rejects the effects of external disturbances.

3. exhibits quality performance, despite plant parameter changes.

TABLE OF CONTENTS

PAGE

LIST OF ILLUSTRATIONS

LIST OF FIGURES

This list of figures does not include results from simulations or applications.

PAGE

LIST OF PHOTOGRAPHS

NOMENCLATURE

The following conventions are adhered to throughout the thesis:

1. All vectors are written in bold text.

2. Matrices are written with capital letters in normal text.

3. The symbol,"*", denotes scalar multiplication.

4. The symbol, ".", denotes scalar product of 2 vectors.

A is a matrix with elements a_{ij} such that: $d\mathbf{x}/dt = A\mathbf{x}$ or $d\mathbf{x}/dt = A\mathbf{x} + B\mathbf{u}$.

A_{eq} is a matrix, such that the dynamics of the system can be written in the form: $d\mathbf{x}/dt = A_{eq}\mathbf{x}$.

B is the system gain matrix with elements b_{ij}.

b is the input gain for a single input system.

C is the matrix which defines the surfaces for sliding and has elements c_{ij}.

C_{sys} is a matrix such that: $\mathbf{y} = C_{sys}\mathbf{x}$. This is to avoid ambiguity with the above C.

c defines the slope of the switching line in second order systems.

c_1 corresponds to c for the modified variable structure control system.

c_2 defines the safety switching structure in the modified variable structure control system.

e is an error vector with elements e_i.

F denotes a general vector function.

f denotes 'function of'.

G(s) is the transfer function for a multi input multi output system, with elements $g_{ij}(s)$.

g(s) is the transfer function for single input single output systems.

I is the identity matrix.

k is the gain of a relay signal.

k_i is the ith state feedback gain term.

L is the norm in the two - dimensional phase space.

L_o is a certain minimum radius about the origin which defines the target area.

m is a constant number added to or subtracted from the slope of the switching line. It defines the switching surface.

M is the gain matrix for the state observer.

N normal vector to a surface.

r is the setpoint for single output systems.

r is the system setpoint vector.

t is the time.

t_v is the sampling time.

$dt = T$ is the sampling period.

u is the scalar input to the system.

$\mathbf{u}$ is the system input vector with elements u_i.

$\mathbf{u_{eq}}$ is a mean control vector.

$\mathbf{x}$ is the state vector with elements x_i.

$\mathbf{x_m}$ is a measured state vector.

$\mathbf{x_{est}}$ is the estimated state vector.

y is the system output for single output systems.

$\mathbf{y}$ is the system output vector with elements y_i.

Γ is the gain of the input function for single input systems.

$\Gamma = \alpha$ or
$= \beta$ or
$= \mu$ for the modified variable structure control system.

$\Gamma_{ij} = \alpha_{ij}$ or
$= \beta_{ij}$ such that $u_i = -\sum_{j=1}^{n}(\Gamma_{ij} * x_j)$ and n = system order.

Γ_{ave} is an average feedback gain constant.

σ is the switching line.

σ is the switching surface vector with elements σ_i.

delσ is the error of the measured switching line.

Φ = TB, is a matrix used to calculate the estimated states vector.

θ = I + AT, is a matrix used to calculate the estimated states vector.

δ_i is the ith eigenvalue of the system.

τ is a noise function.

INTRODUCTION

The theory of variable structure systems (VSS) was developed mainly in the U.S.S.R., where much was written on the subject by men such as Emel'yanov and Utkin in the 1960's and 70's.

Their work was chiefly of a theoretical nature, but served as a mathematical basis for this present special class of nonlinear control systems design area.

Despite the steady documentation of the researched theory in the field of VSS, only selected papers have been seen by the English - speaking world to date. In addition, these have often lost their clarity in the translation.

In spite of these hurdles, controllers designed using the theory of VSS have received increased attention over the last decade. This is owing to particular features intrinsic to a VSS design, which are highly desirable and are explored in this thesis.

Physical implementation of controllers, designed using the theory of VSS, have generally been limited to specialised applications to date. Two factors which are responsible for this situation are:

1. The fundamental concepts of VSS were explained by Soviet authors and are thus not readily available to the English - speaking control engineer.

2. The vast majority of papers written on VSS are purely of a theoretical nature.

The primary aim of this thesis, is to provide a body of knowledge on VSS theory and to highlight the practical significance of all design concepts.

In order to fully illustrate the practical significance of VSS, two case studies are presented. Variable structure controllers are applied to a servo motor system and to a flotation plant simulator.

The servo motor is an important system in academic and industrial circles. It is often an integral component of more sophisticated systems, such as radar tracking devices and robots.

Flotation plants are used in the field of mineral extraction. The application of a variable structure controller to such a system is significant for the following reasons:

1. In the present economic climate, improved means of control for such systems could lead to financial benefits.
2. The simulator system is made up of two inputs and two outputs; thus facilitating the study of a multivariable VSS design.

Specifically, the thesis covers a number of areas:

1. It provides a comprehensive literature survey on VSS.

2. It sets out the fundamental theories and concepts of VSS.

3. Non - ideal system characteristics and how they modify the existing theory are studied.

4. The results of variable structure controllers applied to a servo system are presented. The controllers are applied in simulation as well as in practice.

5. Multivariable VSS theory is explored and practical controllers are designed.

6. VSS theory is applied to flotation processes.

Thus, this thesis attempts to present the theory of VSS, apply the given theory to practical systems and provide solutions to problems not dealt with by the existing theory.

It is hoped that this thesis will provide the foundation and motivation for future work on variable structure systems.

CHAPTER 1

FUNDAMENTAL CONCEPTS OF VARIABLE STRUCTURE SYSTEMS

A system that has variable structure (VSS), is a special type of system, in that its structure can vary in time.

The concept of a structure, though simple, is not commonly exploited in control system theory, and thus needs to be explained.

The aim of this chapter is to build up a body of knowledge which will serve as the foundation for the work explored later in this thesis.

1.1 MEANING OF STRUCTURE

A given system, modelled by a set of differential equations, has a particular structure. As examples of this definition of structure, consider the four systems shown in figure 1.1. The two systems (system number 1 and 2) in figure 1.1(a) differ in the sign of the coupling between two variables and are thus said to have different structures.

Another example of systems with different structures is shown in figure 1.1(b), where system number 3 has a feedback connection and system number 4 does not. In fact any system whose phase plane portrait differs in any way to that of another system, is assumed to have a different structure.

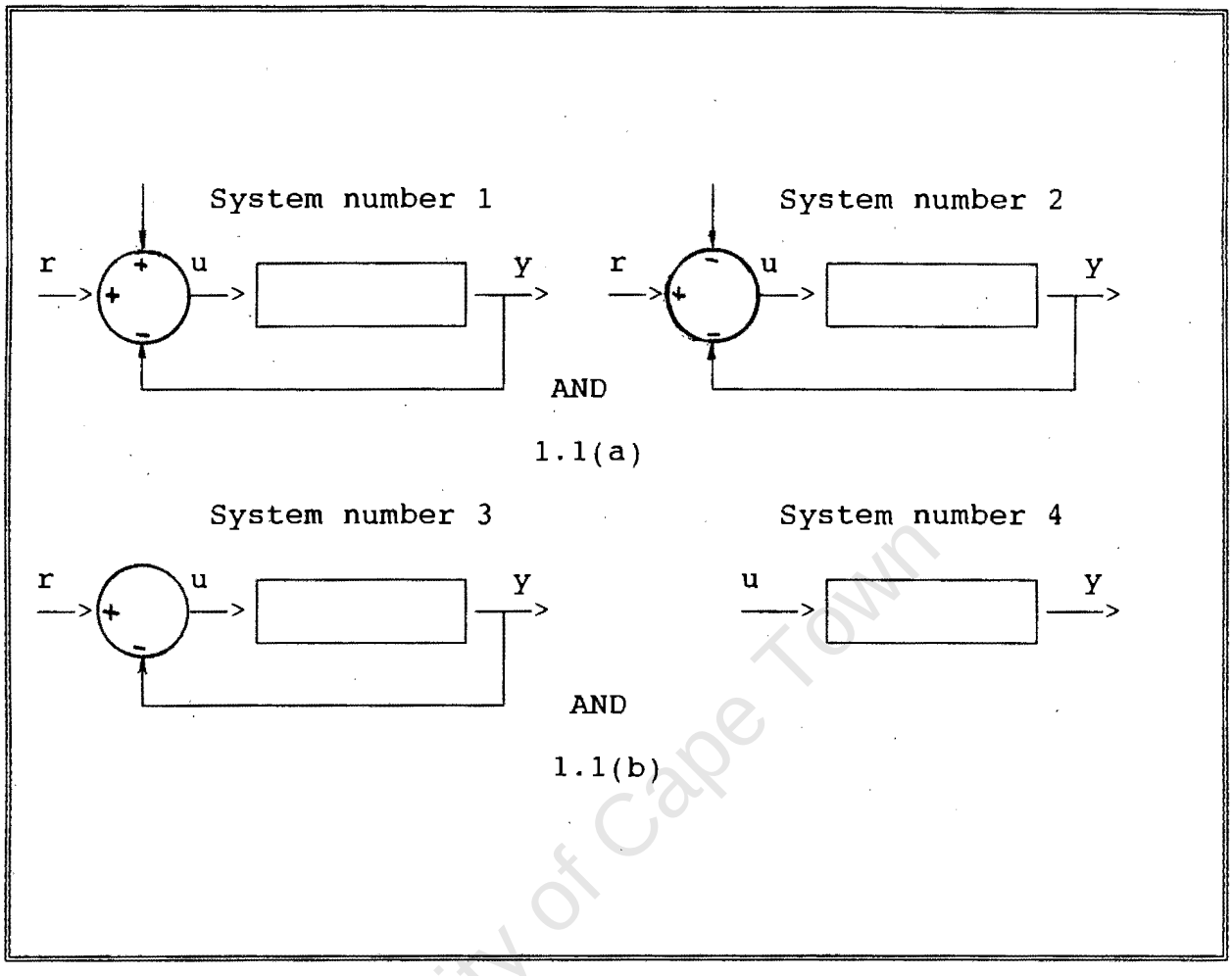


Figure 1.1: Systems with different structures.

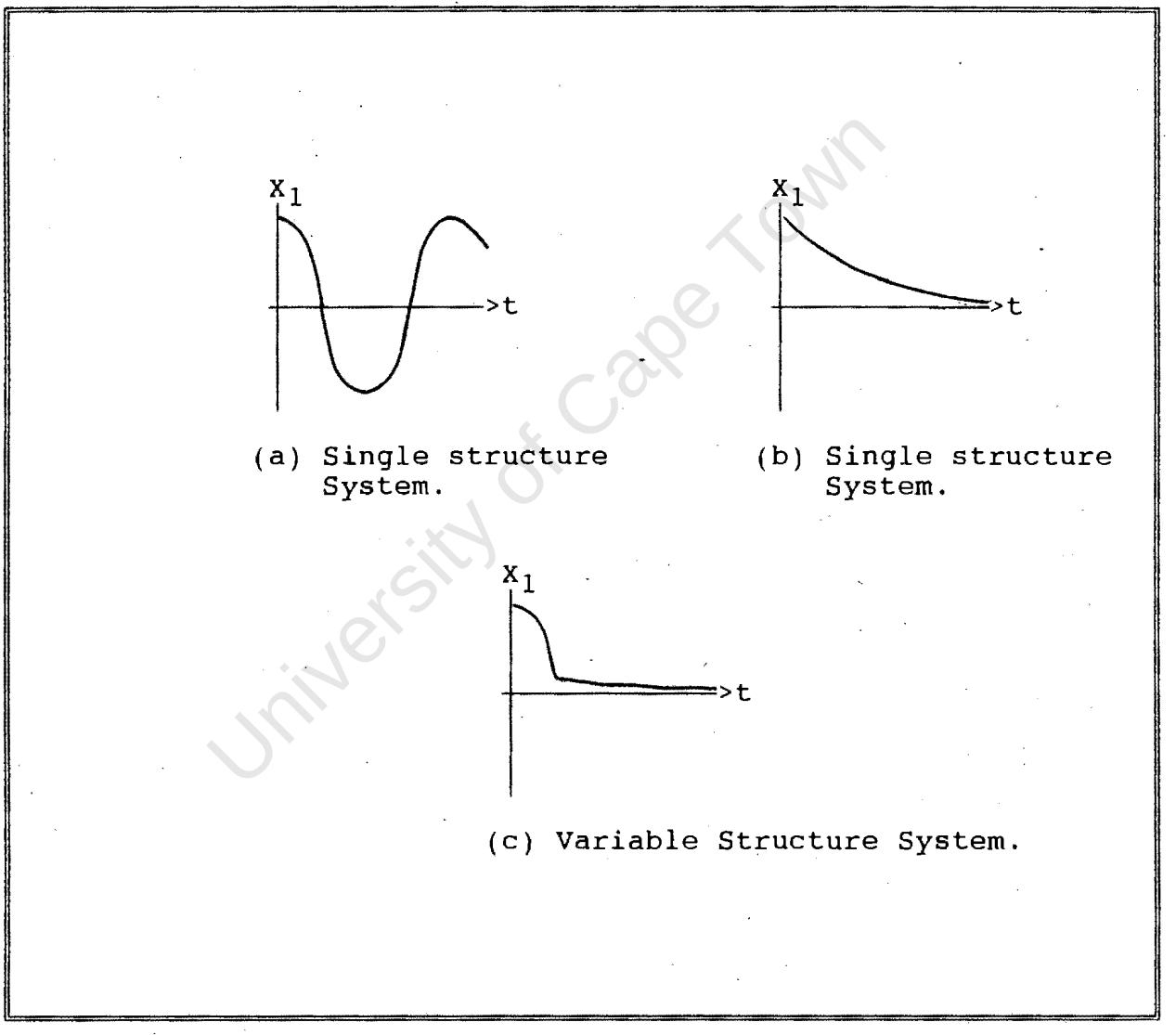
At this early stage, it is important to provide some rationale as to why a control engineer, with all the different methods for designing control systems at his disposal, should want to investigate using VSS theory. The underlying idea behind VSS, is to combine the strengths of particular structures to make a new, superior structure that outperforms its component structures.

This concept is illustrated by the graphs in figure 1.2.

Figure 1.2(a) shows a structure which has a fast response but bad steady state characteristics.

Figure 1.2(b) shows a structure which has good steady state characteristics, but has a slow transient response.

These two structures can be combined to make a new (variable structure) system, which combines the best features of the respective structures. The time response of this new system can be seen in figure 1.2(c).



(a) Single structure System.

(b) Single structure System.

(c) Variable Structure System.

Figure 1.2: Time domain plots of two single structure systems and the ensuing superior variable structure system.

Each structure is represented mathematically by a set of differential equations. These differential equations describe the behaviour of the phase trajectory in the phase plane portrait of the structure.

To gain some practical insight into VSS, simple second order linear systems with no input and constant parameters are considered initially. The open loop state space representation of such systems is:

$$\mathbf{dx}/dt = \mathbf{Ax} \qquad \ldots(1.1)$$

where

$\mathbf{x} = [x_1 \; x_2]^T$.
A = 2x2 system matrix.

This general system can be transformed into an equivalent system in phase canonic form (see appendix 2):

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = - a_1 * x_1(t) - a_2 * x_2(t) \qquad \ldots(1.2)$$

where

$x_1(t)$, $x_2(t)$ are the state variables.

a_1, a_2 are the constant plant parameters.

The eigenvalues, δ_1 and δ_2 of this system, can be obtained by solving the characteristic equation:

$$|A - \delta I| = 0 \qquad \ldots(1.3)$$

where

$$A = \begin{bmatrix} 0 & 1 \\ -a_1 & -a_2 \end{bmatrix}$$

This is equivalent to:

$$\delta^{\wedge}2 + a_2*\delta + a_1 = 0 \qquad \ldots(1.4)$$

The roots, δ_1 and δ_2 of the characteristic equation, are dependent on the values of a_1 and a_2. Each set of values for the roots has an associated characteristic phase portrait.

Specifically, consider the following two second order linear systems with constant coefficients and no input in phase canonic form:

Structure 1:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = -a_1*x_1(t) \qquad \ldots(1.5)$$

where

$a_1 > 0.$
$a_2 = 0.$

The roots of the characteristic equation are purely imaginary and the phase trajectories are ellipses, as shown in figure 1.3(a).

Structure 2:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = +a_1*x_1(t) \qquad \text{...(1.6)}$$

where

$a_1 > 0.$
$a_2 = 0.$

The roots of the characteristic equation are real and of unlike sign:

$$\delta_1 < 0 < \delta_2 \qquad \text{...(1.7)}$$

The phase trajectories are hyperbolas, as shown in figure 1.3(b). The system is unstable apart from the trajectory of the one asymptote:

$$-\delta_1*x_1(t)+x_2(t) = 0 \qquad \text{...(1.8)}$$

This is one of the eigenvectors of the system, with δ_1 being the associated eigenvalue.

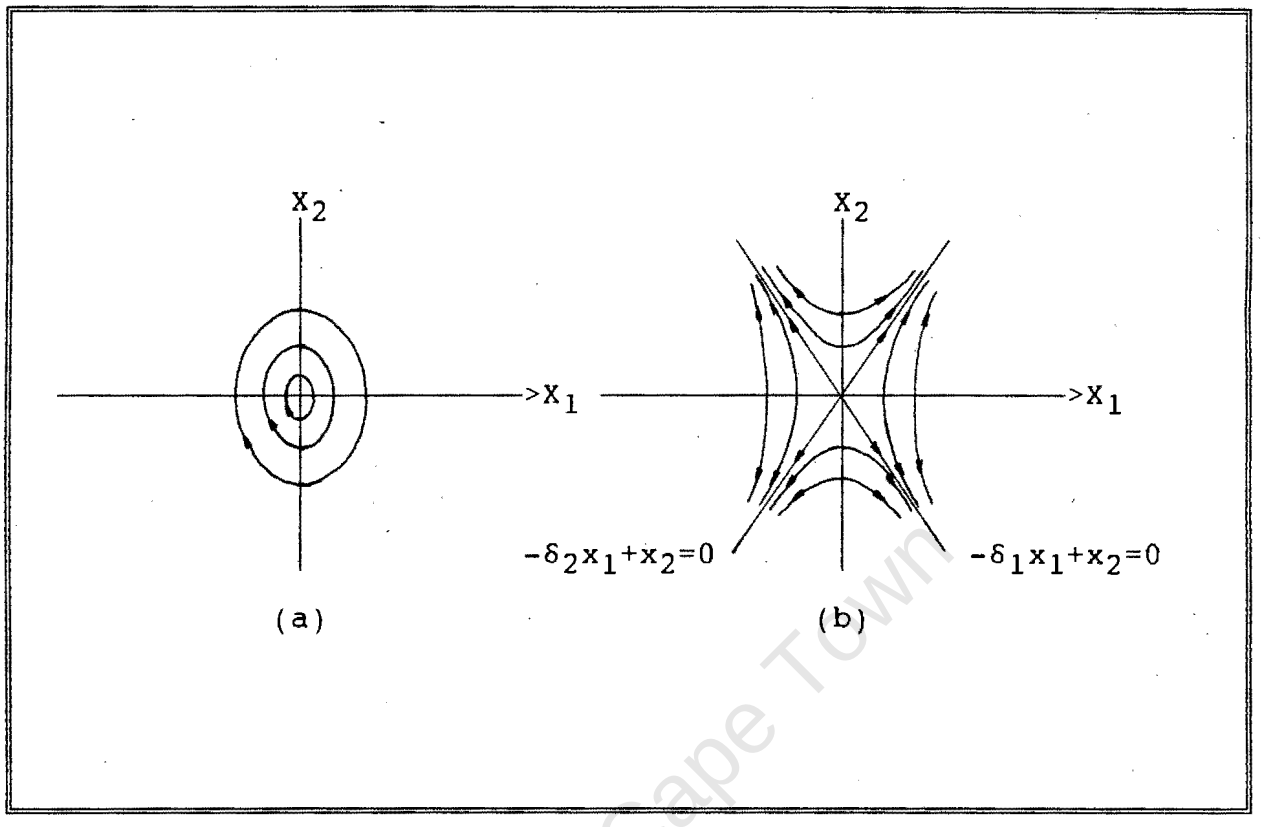


Figure 1.3: Phase portraits for (a) Structure one and (b) Structure two.

If the phase plane is divided into four appropriate regions, labelled 1 to 4 in figure 1.4, a suitably stable VSS is constructed from structures one and two described above. The four regions are demarcated by the lines:

$$x_1(t) = 0 \qquad \ldots(1.9)$$

$$-\delta_1 * x_1(t) + x_2(t) = 0 \qquad \ldots(1.10)$$

Structures one and two are combined into a VSS by switching from one structure to the other in the following way:

If the phase plane representative point (RP) is in either region 1 or 3, the system has structure 1.

If the RP is in either region 2 or 4, the system has structure 2.

Thus, when the RP crosses from region 1 into region 4 for example, the system switches structures from structure 1 to structure 2.

The origin, which is taken to be the stable singular point in the phase plane, is thus reached. This is illustrated in the figure.

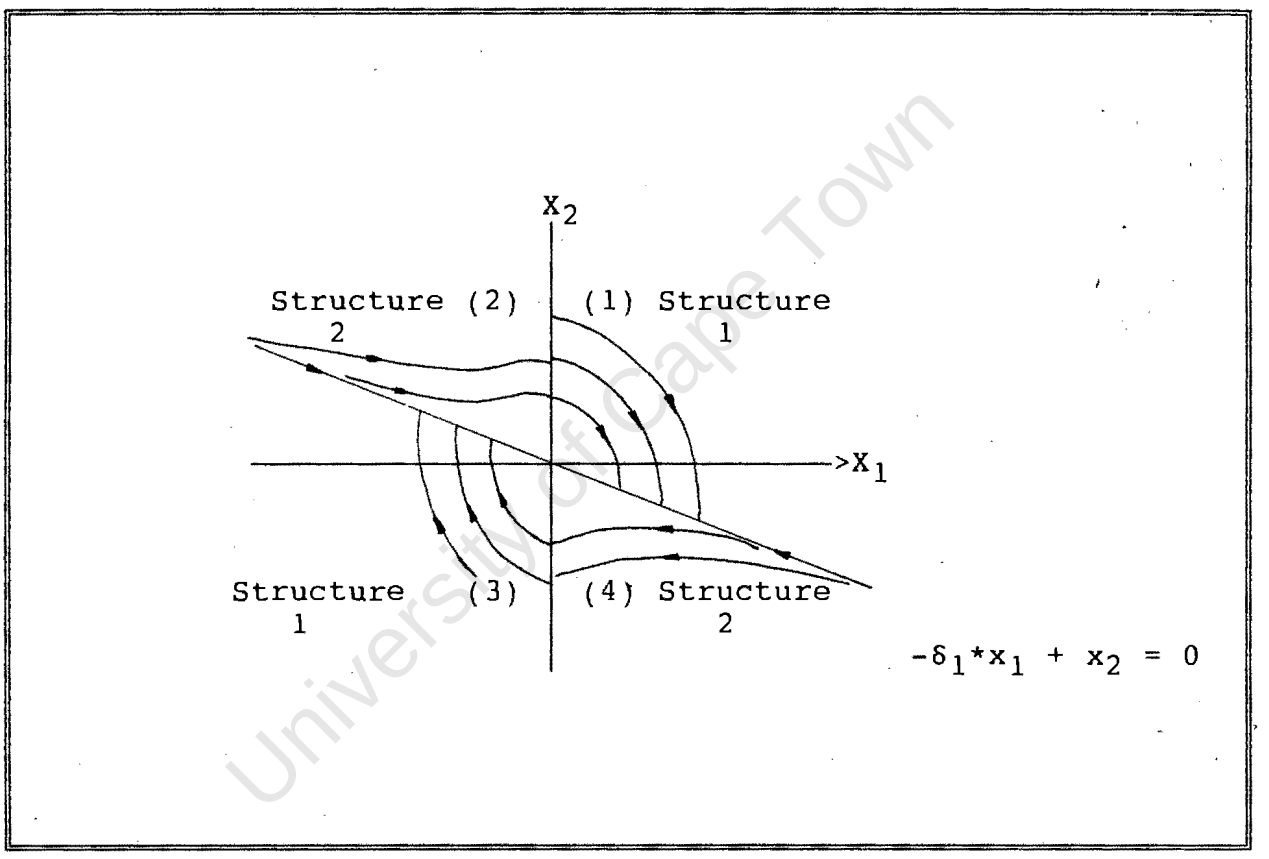


Figure 1.4: A stable VSS.

One interesting feature of VSS, is that stable VSS can be synthesized from inherently unstable component structures. This is clearly seen in the above example. The phase trajectory of Structure 1 never reaches the origin, while that of Structure 2 can only reach the origin along its one stable eigenvector, defined by equation 1.10. Their combined structure however, is asymptotically stable.

This example should not imply that VSS are always made up of unstable component structures. They can be made up from both unstable and stable component structures. The resulting system, if designed properly, should always be able to out - perform any one component structure.

1.2 THE SLIDING MODE

A closer look at the simple VSS made up from structures defined by equations 1.5 and 1.6, reveals a rather idealised situation, as there would be unavoidable fluctuations of the phase trajectory from the one stable asymptote.

VSS is an attractive method of control only when a sliding mode (or sliding regime) is defined.

The best way to explain the sliding mode (SM) is by way of illustration.

In the above example, the phase plane was divided into four region boundaries defined in equations 1.9 and 1.10.

In general, the two - dimensional phase plane can be divided by switching lines:

$$x_1(t) = 0 \qquad \ldots(1.11)$$

$$x_2(t) + c*x_1(t) = 0 \qquad \ldots(1.12)$$

where

c is a constant which defines the slope of the switching line (equation 1.12).

If we choose c such that

$$0 < c < |\delta_1| \qquad \ldots(1.13)$$

In this instance, the phase trajectories of the system defined by Structure 1 and Structure 2 have opposite direction in the region of the switching line. As a result, the trajectory hits the switching line and then, by switching rapidly between the two structures, " slides " along the switching line to the origin. This is illustrated in figure 1.5.

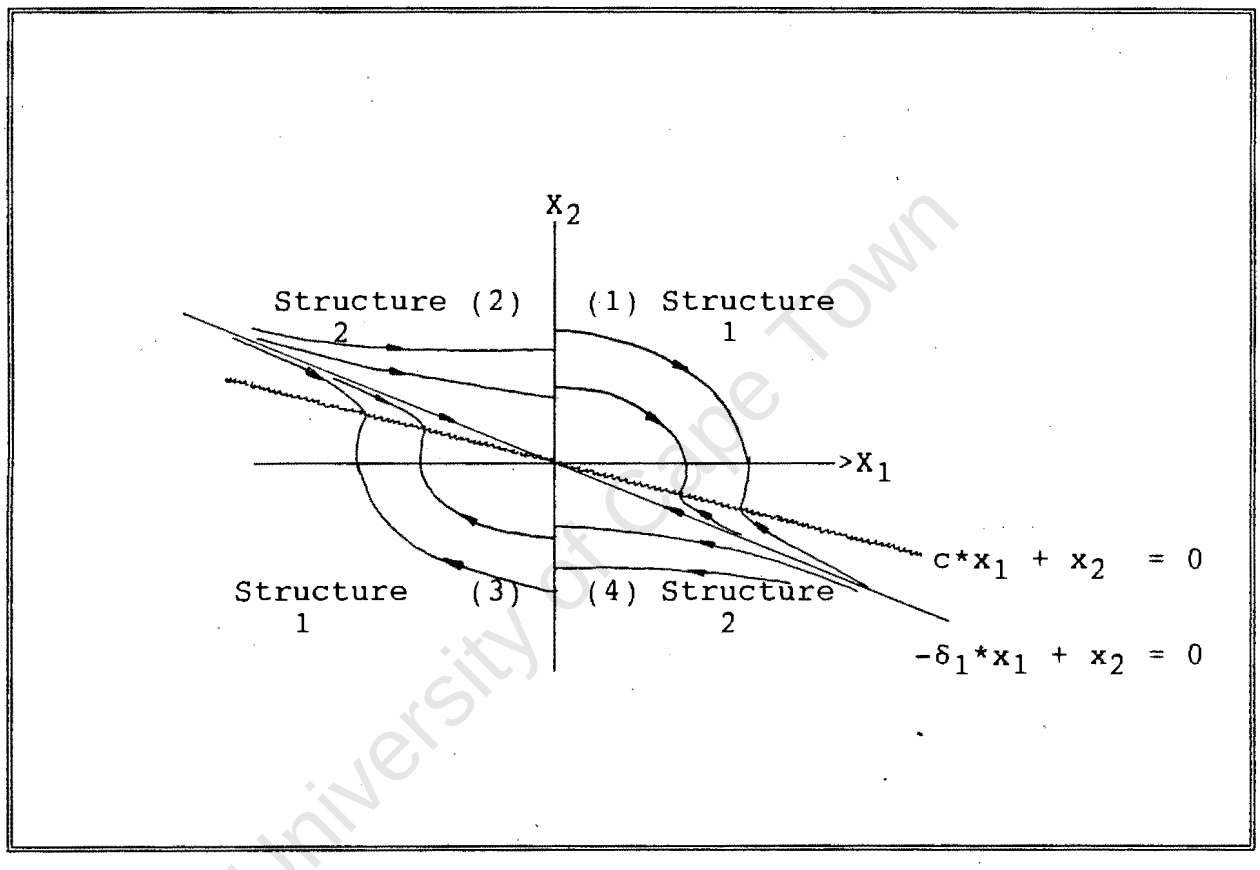


Figure 1.5 : Sliding .

In an ideal situation, once the phase trajectory reaches the switching line, it will oscillate at infinite frequency and infinitesimal amplitude along it. Thus, the system's motion will be described by equation 1.12; the switching line.

Once the system is in the SM, the plant's behavior can be shown to be both independent of parameter changes and external disturbances. This is indeed a powerful asset for a control algorithm and reason enough to consider VSS for practical applications.

To illustrate this, it is apparent that the system's parameters, $\pm a_1$ in the above example, do not appear in equation 1.12. Since this equation describes the motion of the system in the sliding mode, this implies that once the sliding mode has been reached, the plant is insensitive to parameter changes.

Should the system experience an external disturbance, its phase trajectory will be distorted until it hits the switching line. As long as the individual trajectories point in the opposite direction here,however (ie in the vicinity of the switching line), sliding will occur and hence stability is assured.

It is now apparent that when a sliding mode exists, the plant is insensitive to parameter variations and external disturbances. These results are so important,however, that ensuring the existence of the sliding mode is of primary importance when designing a controller using VSS theory. Thus the necessary conditions for the existence of the SM and its reachability are given later in this chapter.

In practice, the switching cannot be done at an infinite frequency, but this does not invalidate the above results. In reality, the motion of the phase trajectory in the SM moves on average along the switching line or surface.

In most cases, having a SM resolves the conflict between static and dynamic accuracy of the system, because it enables one to split the transient into two independent stages:

1. **HITTING STAGE** - a brief motion up to the beginning of the sliding regime. This period of the system's performance is affected by plant parameter changes and external disturbances.

2. **SLIDING REGIME** - rapidly damped oscillations where the motion is invariant to parameter changes and external disturbances.

The obvious approach is to force the system to reach the SM as fast as possible, because once the system is in the SM, stability is assured.

1.3 CLOSED LOOP BEHAVIOUR OF VARIABLE STRUCTURE SYSTEMS

Consider the open loop system that has been looked at so far with an input added:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = - a_2*x_2(t) - a_1*x_1(t) + b*u(t) \qquad ...(1.14)$$

where

$x_1(t)$ could be a position state and $x_2(t)$ a velocity state.

The control signal defining the closed loop system has the form:

$$u(t) = -\Gamma*x_1(t) \qquad ...(1.15)$$

in which the control term Γ is given by:

$$\begin{aligned} \Gamma &= \propto \text{ when } x_1(t)*\sigma > 0 \text{ (Region 1 and 3 in figure 1.4)} \\ &= \beta \text{ when } x_1(t)*\sigma < 0 \text{ (Region 2 and 4 in figure 1.4)} \end{aligned} \qquad ...(1.16)$$

where

$\propto$ and β are constants and

$$\sigma = x_2(t) + c*x_1(t) = 0 \qquad ...(1.17)$$

is the system switching line.

For systems defined by equations 1.14, 1.15, 1.16 and 1.17, the design parameters for the variable structure controller are thus α, β and c.

The form of the control law is easy to understand when α is positive and β is negative. In this case when $\Gamma = \alpha$, a negative feedback structure is created with the gain of the feedback signal being simply α. Similarly, when $\Gamma = \beta$, a positive feedback structure is created, with β, the gain of the feedback signal.

Since the input is a linear function of the x_1 state, the closed loop system can be written in the form of equation 1.1.

$$d\mathbf{x}/dt = A_{eq}\mathbf{x} \qquad \ldots(1.18)$$

where

$$A_{eq} = \begin{bmatrix} 0 & 1 \\ -(a_1+b*\Gamma) & -a_2 \end{bmatrix}$$

and

$\Gamma = \alpha, \beta$.

By choosing specific values for α and β in solving the system's characteristic equation, two structures are created which are defined by their associated eigenvalues.

It has been shown that since the input function is a linear function of the x_1 state, the system can be written in the form of equation 1.18. This means that a sliding mode can easily be defined and thus, the controlled closed loop system will be insensitive to plant parameter changes and external disturbances once the SM has been reached.

1.4 MATHEMATICAL ANALYSIS OF THE SLIDING REGIME .

The necessary and sufficient conditions for the existence of a sliding mode need to be looked at. Before embarking on finding these conditions, a more rigorous look at the definition of the sliding mode needs to be considered. It is, after all, the single most important feature of variable structure systems.

Consider the general system of differential equations:

$$dx_j/dt = f_j(x_1,\ldots,x_n,t) \quad (\text{for } j = 1,2,\ldots,n) \qquad \ldots(1.19)$$

To start the analysis, it is assumed that the respective differential equations are discontinuous on a hypersurface defined by:

$$\sigma(x_1,\ldots x_n) = 0 \qquad \ldots(1.20)$$

This hypersurface is in the phase space:

$$\mathbf{H}(x_1,\ldots x_n) \qquad \ldots(1.21)$$

Equations 1.19 and 1.20 are just generalisations of the second order system, described by equation 1.14, which switches on the switching line described by equation 1.17, in the two - dimensional phase plane.
The discontinuity is such that:

$$\lim_{\sigma\to-0} f_j = f^-_j$$

$$\lim_{\sigma\to+0} f_j = f^+_j \qquad \ldots(1.22)$$

where

$f^+_j <> f^-_j$

The following mathematical analysis sets up conditions to ensure that the phase trajectory always points towards the switching surface in its neighbourhood.

Taking the derivative of σ, defined in equation 1.20, with respect to time, the following expression is obtained:

$$d\sigma/dt = \sum_{j=1}^{n} (d\sigma/dx_j * dx_j/dt) \quad \text{(By the chain rule)}$$

$$= \sum_{j=1}^{n} (d\sigma/dx_j * f_j)$$

$$= \mathbf{F}.\mathbf{grad}\sigma$$

$$= \{ \sum_{j=1}^{n} (d\sigma/dx_j)^2 \}^{0.5} * \{\mathbf{N}.\mathbf{F}\} \qquad \ldots(1.23)$$

where

$\mathbf{F}$ = $(f_1, \ldots f_n)^T$; an n vector.

$\mathbf{N}$ = hypernormal to the hypersurface.

$\mathbf{N}.\mathbf{F}$ = scalar product of the two n vectors.

By equations 1.19, 1.22 and 1.23:

$$\lim_{\sigma \to -0} d\sigma/dt = \mathbf{F}^-.\mathbf{grad}\sigma$$

$$\lim_{\sigma \to +0} d\sigma/dt = \mathbf{F}^+.\mathbf{grad}\sigma \qquad \ldots(1.24)$$

where

$\mathbf{F}^- = (f^-_1, \ldots f^-_n)^T$.

$\mathbf{F}^+ = (f^+_1, \ldots f^+_n)^T$.

At each point of $\sigma = 0$, the signs of the limits in equations 1.24 may stand in one of the following seven relationships:

$$\lim_{\sigma \to +0} d\sigma/dt > 0 < \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25a)$$

$$\lim_{\sigma \to +0} d\sigma/dt < 0 > \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25b)$$

$$\lim_{\sigma \to +0} d\sigma/dt > 0 > \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25c)$$

$$\lim_{\sigma \to +0} d\sigma/dt < 0 < \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25d)$$

$$\lim_{\sigma \to +0} d\sigma/dt = 0 < \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25e)$$

$$\lim_{\sigma \to +0} d\sigma/dt < 0 = \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25f)$$

$$\lim_{\sigma \to +0} d\sigma/dt = 0 = \lim_{\sigma \to -0} d\sigma/dt \qquad \ldots(1.25g)$$

The corresponding phase portraits to equations 1.25 can be seen in figure 1.6.

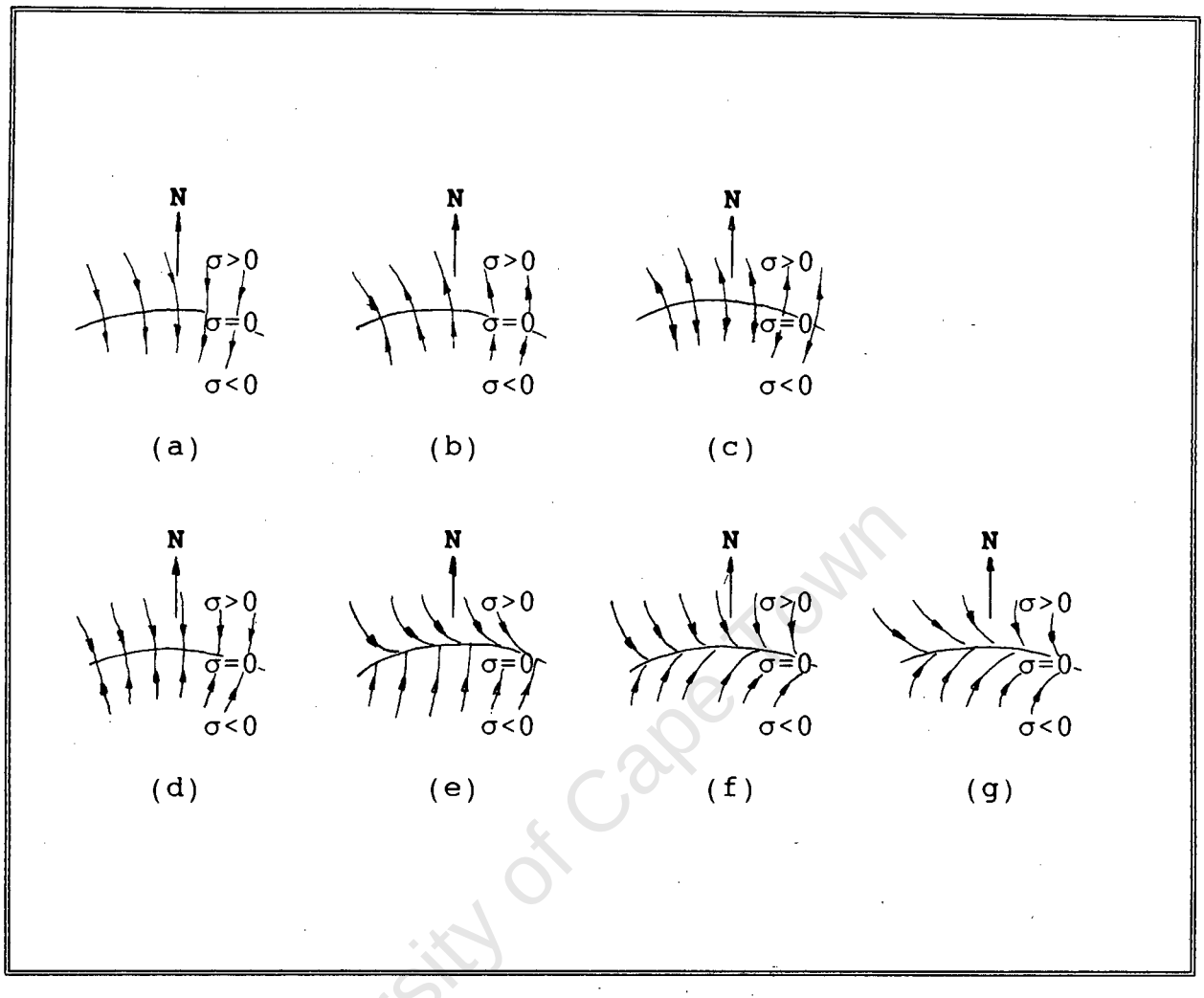


Figure 1.6: Phase portraits corresponding to equations 1.25.

Equations 1.25(a) and 1.25(b) correspond to the phase trajectory piercing the switching hypersurface. On either side of the hypersurface, the projection of vector **F** on the hypernormal **N** has the same sign.

Equation 1.25(c) corresponds to the degenerate case. On either side of the hypersurface, vector **F** points away from the hypersurface.

Equations 1.25(d), 1.25(e), 1.25(f) and 1.25(g) correspond to an ideal sliding regime on $\sigma = 0$. In these cases, the phase trajectory remains within an infinitesimal neighbourhood of the hypersurface.

Combining these cases into a single condition:

$$\lim_{\sigma->=+0} d\sigma/dt \leq 0 \leq \lim_{\sigma->-0} d\sigma/dt \qquad \ldots(1.26)$$

Which is equivalent to:

$$\lim_{\sigma->0} \sigma * d\sigma/dt \leq 0 \qquad \ldots(1.27)$$

Equation 1.27 describes the existence condition for a sliding regime on the hypersurface $\sigma = 0$. This general condition always needs to be satisfied for a sliding regime to exist.

Note that the hypersurface is given by:

$$\sigma(x_1, \ldots x_n)^T = 0 \qquad \ldots(1.28)$$

In reality, the components of the switching surface do not equal zero at all times, but oscillate at high frequencies and small amplitudes about zero. This implies that the fast perturbations about the sliding surface are replaced by an average component on the hypersurface itself.

Attention is now turned to existence and reachability conditions of the SM for specific systems.

1.4.1 CONDITIONS FOR THE EXISTENCE OF A SLIDING MODE FOR CLOSED LOOP SINGLE INPUT SYSTEMS IN PRIMARY FEEDBACK CONFIGURATION

Equation 1.27 describes the general condition of existence for a sliding mode.

In this section, this general condition is translated to specific conditions which ensure the existence of a sliding mode for systems incorporated in a primary feedback configuration. These are systems whose inputs are a function of one state only.

1.4.1(a) EXISTENCE CONDITIONS FOR A SLIDING MODE WHEN CONSIDERING SECOND ORDER SYSTEMS WITH CONSTANT PARAMETERS

Consider the open loop process described by equations 1.14 - 1.17:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = -a_2*x_2(t) - a_1*x_1(t) + b*u(t) \qquad \ldots(1.29)$$

The control law is given by:

$$u(t) = -\Gamma*x_1(t) \qquad \ldots(1.30)$$

The switching logic is given by:

$$\Gamma = \alpha \text{ if } x_1(t)*\sigma(t) > 0$$
$$= \beta \text{ if } x_1(t)*\sigma(t) < 0 \qquad \ldots(1.31)$$

and the switching surface is:

$$\sigma(t) = c*x_1(t) + x_2(t) = 0 \qquad \ldots(1.32)$$

Thus:

$$d\sigma/dt = d(x_2 + c*x_1)/dt$$

$$= -a_2*x_2 - a_1*x_1 - b*\Gamma*x_1 + c*x_2$$

$$= -(a_2 - c)*x_2 - (a_1 + b*\Gamma)*x_1 \quad \ldots(1.33)$$

But on the switching line, $\sigma = 0$ and thus:

$$x_2 = -c*x_1 \quad \ldots(1.34)$$

So:

$$d\sigma/dt = (a_2*c - c^{\wedge}2 - a_1 - b*\Gamma)*x_1 \quad \ldots(1.35)$$

Therefore, by equation 1.27:

$$\lim_{\sigma->0} \sigma*d\sigma/dt = (a_2*c - c^{\wedge}2 - a_1 - b*\alpha)*x_1*\sigma \leq 0 \quad \text{if } x_1*\sigma > 0 \quad \ldots(1.36)$$

$$\lim_{\sigma->0} \sigma*d\sigma/dt = (a_2*c - c^{\wedge}2 - a_1 - b*\beta)*x_1*\sigma \geq 0 \quad \text{if } x_1*\sigma < 0 \quad \ldots(1.37)$$

So, existence of a SM is assured if the following conditions hold:

$$\alpha \geq (a_2*c - c^{\wedge}2 - a_1)/b \quad \ldots(1.38)$$

$$\beta \leq (a_2*c - c^{\wedge}2 - a_1)/b \quad \ldots(1.39)$$

It was shown previously that when in the sliding mode, motion of the system moves on average along the switching line. It is now of interest to look at the specific equations governing the motion of the system in the SM.

By equation 1.28, $\sigma = 0$ in the SM. Thus, motion for the closed loop system is described by:

$$dx_1(t)/dt = - c*x_1(t) \qquad \ldots(1.40)$$

which has solution:

$$x_1(t) = x_1(t_0)*\exp\{-c*(t-t_0)\} \qquad \ldots(1.41)$$

where

t_0 = time that the phase trajectory reaches the sliding plane.

From equation 1.41, it is clear that the constant, c, determines the speed of convergence of the system in the sliding mode. Thus, a suitable c value may be chosen to satisfy some transient performance criterion. Gain values for $\propto$ and β are then chosen to ensure the existence of a sliding regime and the specified c value is checked to satisfy equation 1.13.

1.4.1(b) EXISTENCE CONDITIONS FOR A SLIDING MODE FOR SECOND ORDER SYSTEMS WITH VARIABLE PARAMETERS

Consider a second order linear open loop system with variable parameters of the form:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = -a_2(t)*x_2(t) - a_1(t)*x_1(t) + b(t)*u(t) \qquad \ldots(1.42)$$

where

$$a_{1min} \leq a_1(t) \leq a_{1max}$$

$$a_{2min} \leq a_2(t) \leq a_{2max}$$

$$0 < b_{min} \leq b(t) \leq b_{max}$$

The control function is given by:

$$u(t) = - \Gamma * x_1(t) \qquad \ldots(1.43)$$

$$\Gamma = \propto \text{ if } \sigma * x_1(t) > 0$$
$$= \beta \text{ if } \sigma * x_1(t) < 0 \qquad \ldots(1.44)$$

Now, analogous to arriving at conditions for the existence of a SM for second order systems with constant parameters:

$$\lim_{\sigma \to 0} \sigma * d\sigma/dt = (c * a_2(t) - c^2 - a_1(t) - b(t) * \Gamma) * x_1 * \sigma \leq 0 \qquad \ldots(1.45)$$

The following existence conditions are arrived at:

$$\propto \geq \max_{a1,a2,b} \{(c * a_2(t) - c^2 - a_1(t))/b(t)\} \qquad \ldots(1.46)$$

$$\beta \leq \min_{a1,a2,b} \{(c * a_2(t) - c^2 - a_1(t))/b(t)\} \qquad \ldots(1.47)$$

The extrema of equations 1.46 and 1.47 are taken over all possible values for $a_1(t)$, $a_2(t)$ and $b(t)$.

In practice, this means that the absolute value of the gain of the control signal, $|\Gamma|$, has to be big enough to ensure that a sliding regime exists for all possible combinations of structures.

Equations 1.46 and 1.47 are the worst case conditions for Γ.

1.4.1(c) EXISTENCE CONDITIONS FOR A SLIDING REGIME FOR NTH ORDER SYSTEMS WITH CONSTANT PARAMETERS

Consider the following nth order open loop system with constant parameters:

$$dx_j(t)/dt = x_{j+1}(t) \quad (\text{for } j = 1,\ldots,n-1)$$

$$dx_n(t)/dt = -\sum_{j=1}^{n} a_j * x_j(t) + b*u(t) \qquad \ldots(1.48)$$

The control law is specified by:

$$u(t) = -\Gamma * x_1(t) \qquad \ldots(1.49)$$

where

$$\Gamma = \alpha \text{ if } x_1(t)*\sigma > 0$$
$$= \beta \text{ if } x_1(t)*\sigma < 0 \qquad \ldots(1.50)$$

and

$$\sigma = \sum_{j=1}^{n} c_j * x_j(t) = 0 \qquad \ldots(1.51)$$

c_j = constant > 0 (for j = 1,..,n-1).
c_n = 1.

The necessary and sufficient conditions for the existence of a sliding regime are arrived at in the usual way by satisfying equation 1.27.

They are:

$$c_{j-1} - a_n = c_j*(c_{n-1} - a_n) \quad (\text{for } j = 2,\ldots,n) \qquad \ldots(1.52)$$
$$c_n = 1$$

$$\alpha \geq (-a_1 + c_1*a_n - c_1*c_{n-1})/b \qquad \ldots(1.53)$$

$$\beta \leq (-a_1 + c_1*a_n - c_1*c_{n-1})/b \qquad \ldots(1.54)$$

Notice that a new condition on the values of c_j appears. This could be circumvented by making the input a linear function of the derivative states (derivative feedback); the usual approach for such systems.

1.4.2 CONDITIONS FOR HITTING THE SLIDING PLANE

Now that conditions ensuring the existence of a sliding mode have been formulated, it is necessary to ensure that the phase trajectory will in fact reach the sliding mode.

1.4.2(a) HITTING CONDITIONS FOR SECOND ORDER SYSTEMS WITH CONSTANT PARAMETERS

The following theorem was provided by Emel'yanov to ensure hitting occurs for second order systems with constant parameters. It can be found in [8].

THEOREM 1: The phase trajectory of a system described by equation 1.14 will hit the switching line iff the characteristic equation, with $\Gamma = \alpha$, has no non-negative real roots (only complex and negative real roots allowed).

The theorem states that hitting will occur for the negative feedback structure, which has no non - negative real roots. This theorem is satisfied if the phase trajectory merely tends asymptotically to the origin along the switching line.

The characteristic equation, with $\Gamma = \alpha$ is:

$$\delta^2 + a_2*\delta + (a_1 + b*\alpha) = 0 \qquad \ldots(1.55)$$

Thus, if α is greater than zero and the following holds:

$$b*\alpha > [(a_2^2)/4] - a_1 \qquad \ldots(1.56)$$

The characteristic equation has complex conjugate roots with negative real parts and hitting will occur.

Equation 1.56 is thus a sufficient condition for hitting to occur.

It should be noted that by making the controller gain ($\propto$ and β) sufficiently large, both existence and hitting conditions will be satisfied. A large value will also mean a shorter time to reaching the SM. This is in contrast to fixed structure systems, where the controller gain has to be restricted to ensure system stability.

The size of the gains are usually only restricted by the allowable input range of a particular system.

1.4.2(b) HITTING CONDITIONS FOR SECOND ORDER SYSTEMS WITH VARIABLE PARAMETERS

A necessary and sufficient condition for hitting to occur in second order systems with variable parameters is, by theorem 1, that the characteristic equation:

$$\delta^2 + a_{2min}*\delta + a_{1min} + \inf_t\{b(t)*\propto\} = 0 \qquad \ldots(1.57)$$

has no non negative real roots.

1.4.2(c) HITTING CONDITIONS FOR NTH ORDER SYSTEMS WITH CONSTANT PARAMETERS

Theorem 2 was provided by Gerashchenko to ensure hitting occurs for second order systems with constant parameters. It can be found in [8].

THEOREM 2 : The phase trajectory of a system described by equations 1.48 - 1.51 will hit the switching surface iff the characteristic equation, with $\Gamma = (-a_1 + c_1*a_n - c_1*c_{n-1})/b$, has no non-negative real roots except for the root, $\delta_n = c_{n-1} - a_n$.

This theorem is similar to theorem 1. It has been pointed out that for nth order systems, derivative feedback is usually employed.

Thus, for systems whose input is a linear function of k states, the theorem is modified as follows ([12]):

THEOREM 3 : The phase plane trajectory of a system described by equation 1.48, whose input is a linear function of k states ($1 \leq k \leq n$), will hit the switching surface iff all real eigenvalues of the system, with $\Gamma_i = \alpha_i$ ($i = 1,..,k$), be non-negative.

There are various equivalent forms that this theorem may take (see [12]). All that is really pertinent, however, is that for any choice of c_1 to c_n, there exists a minimum controller gain such that if the chosen controller gain is greater than or equal to this minimum gain, the phase trajectory will reach the switching hyperplane from any initial point in the phase plane.

1.5 CONTROLLING SINGLE INPUT SYSTEMS WITH A SETPOINT

Now that the necessary conditions for the existence and reachability of a SM have been established for the general system with a single input, focus is returned to the second order system with constant parameters that was discussed at the beginning of this chapter.

The theory that has been looked at thus far has required that the switching line go through the origin of the phase plane (since the state space equations only model the system dynamics).

Figure 1.7 shows a variable structure system layout to track a specified setpoint.

The way to tackle this problem, is to transform the system in such a way that it switches on the difference between the setpoint and output signal, where the output is made to equal the $x_1(t)$ state. This transformation can be done by using the method outlined in appendix 2.

The system then switches on the error states.

$$de_1(t) = e_2(t)$$
$$de_2(t) = -a_2*e_2(t) - a_1*e_1(t) + b*u(t) \qquad \text{...(1.58)}$$

$$e_1(t) = r - x_1(t) \qquad \text{...(1.59)}$$

$$u(t) = -\Gamma*e_1(t) \qquad \text{...(1.60)}$$

The $e_2(t)$ state is the derivative of the $e_1(t)$ state. In general, this state would have to be obtained by differentiating the $e_1(t)$ state.

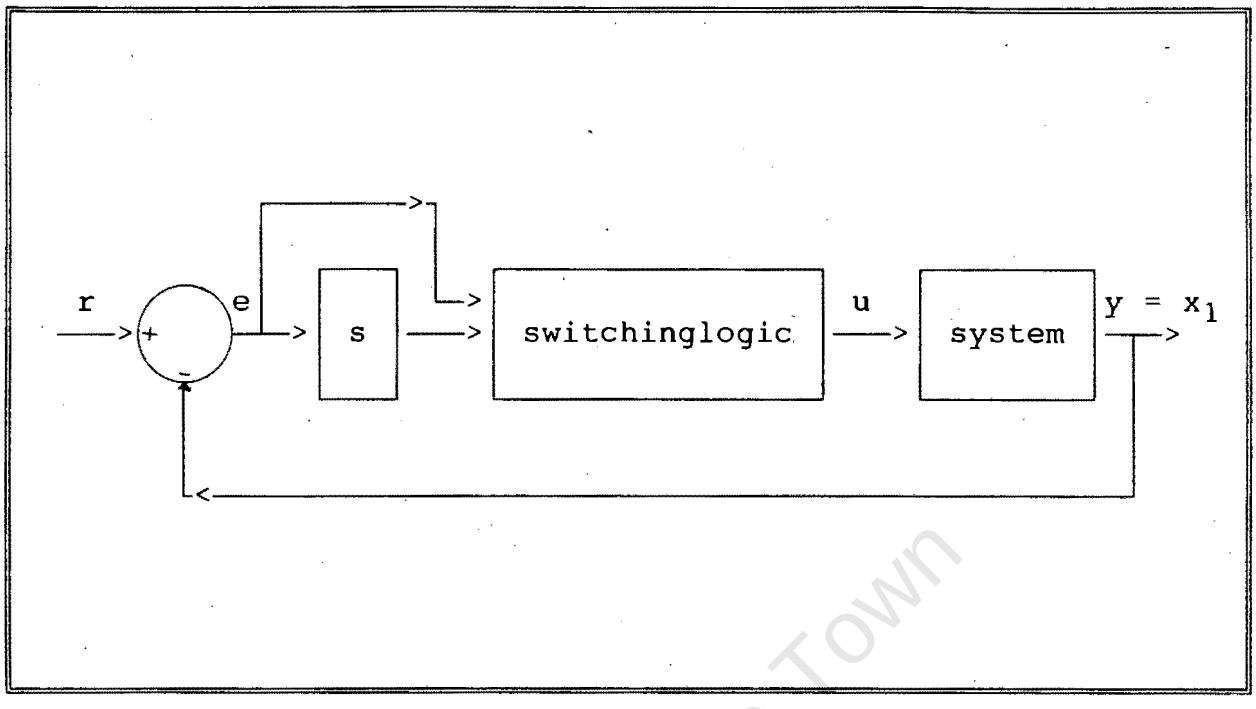


Figure 1.7: Layout of a VSS which is capable of tracking a setpoint.

CONCLUDING REMARKS AND REVIEW ON THE CHAPTER

The chapter started off by introducing the concept of a state. For systems of the form:

$$d\mathbf{x}/dt = A\mathbf{x}$$

the eigenvalues are easily obtainable and they define a particular system structure.

By introducing an input which is a linear function of one (primary feedback) or more (derivative feedback) of the system states, the closed loop system can be written in the form:

$$d\mathbf{x}/dt = A_{eq}\mathbf{x}.$$

The elements of A_{eq} include Γ terms, the feedback gain. Thus, the eigenvalues of the system may be chosen to obtain desired structures.

When a sliding mode is defined, VSS theory has important qualities which are desirable in the design of controllers. Once the system has reached the sliding regime (switching line in second order systems), it is invariant to plant parameter changes or external disturbances.

Conditions for the existence of a sliding mode were analysed from the derived general condition. Hitting or reaching conditions were also looked at.

When the system is in the sliding mode, the system order decreases.

The speed of the transient response in the sliding mode is dependent on the choice of c values. Thus, a relatively large (in size) choice of c in a second order system will ensure a very fast response.

Variable structure systems thus combine speed of response with robustness to noise and model changes to form a highly desirable controlled system.

CHAPTER 2

APPLICATION OF A VARIABLE STRUCTURE CONTROLLER TO A SERVO SYSTEM

This chapter is primarily concerned with the physical implementation of a VSS controller to a single input, linear servo motor. The motor is a second order system and has constant parameters.

There are a number of reasons for looking at such a system:

1. It will serve to illustrate all design concepts introduced in chapter one.

2. The servo motor is an important practical model in control engineering.

3. Practical problems and considerations will be highlighted.

2.1 THE SERVO MOTOR

The D.C. servo motor is a popular and well understood system in industrial and academic circles and has widespread applications. It is often an integral component in sophisticated systems such as robots and radar tracking devices.

A modular servo system is interfaced to an I.B.M. personal computer fitted with a hercules graphics card and a data translation, 12 bit ± 10v analog interface card. This gives an input/output range of -2048 to 2047 counts.

The motor system consists of the following components:

1. Power supply unit.

2. Servo amplifier module. This can be configured for either field control or armature voltage control (used here).

3. The D.C. servo motor.

4. Pre - amplifier module. This is necessary to facilitate bidirectional movement of the motor, for an input signal which can vary between positive and negative values.

5. Operational amplifier unit.

6. Motor tachogenerator. The motor speed is obtained from this module.

7. Output potentiometer. This gives the motor position.

The ensuing block diagram of the closed loop system is shown in figure 2.1.

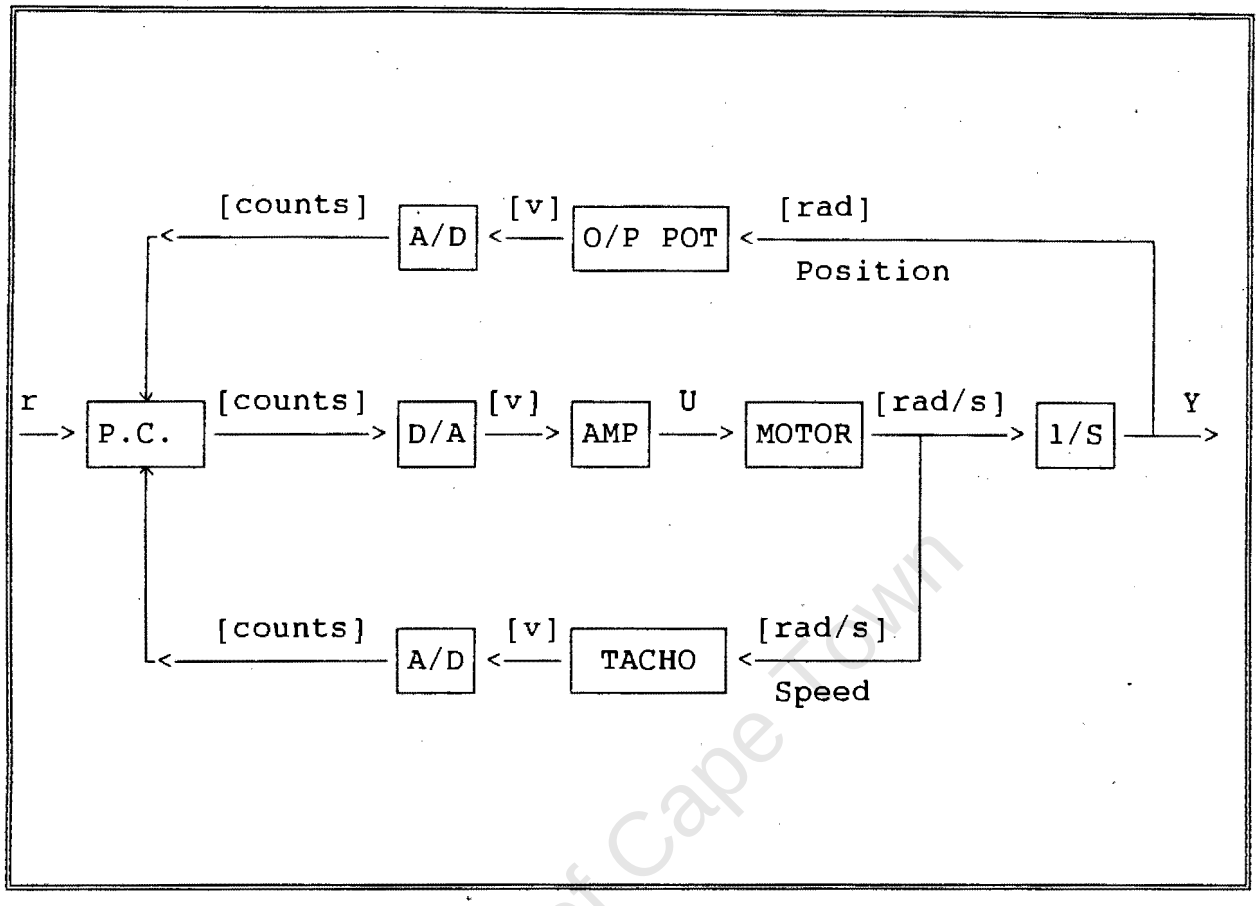


Figure 2.1: Block diagram of the servo motor system.

2.1.1 DYNAMIC BEHAVIOUR OF THE D.C. MOTOR

The following equations describe the dynamic behaviour of the separately excited D.C. servo motor [9]:

$$U_f = R_f * i_f + L_f * di_f/dt \qquad \ldots(2.1)$$

$$U_a = R_a * i_a + L_a * di_a/dt + c * \Phi_f * w \qquad \ldots(2.2)$$

$$M_d = c * \Phi_f * i_a \qquad \ldots(2.3)$$

$$M_d - M_L = J * dw/dt \qquad \ldots(2.4)$$

where

U_f and i_f are the field voltage and current respectively.
U_a and i_a are the armature voltage and current respectively.
R_f and R_a are the respective field and armature resistances.
L_f and L_a are the respective field and armature inductances.
M_d and M_L are the respective driving and load torques.
w is the motor speed.
Φ_f is the induced field flux.
c is a constant.
J is the inertia of the system.

From the above equations, it is clear that there are two ways of controlling the speed of the system:

1. **ARMATURE CONTROL** : By varying the armature voltage (to be used here).
2. **FIELD CONTROL** : By varying the induced field flux.

Figure 2.2 shows the control ranges of the D.C. motor in steady state(from [9]). The experiments described in this chapter are all performed using armature control (done at constant field flux).

Below the base speed, w_o, the field flux is kept at Φ_{fo} while the speed is varied through the armature voltage, U_a. This is the armature control range.

When the rated armature voltage, $\pm U_{ao}$, is reached, a further increase in speed is only possible by decreasing the field flux, Φ_f. This is the field control range.

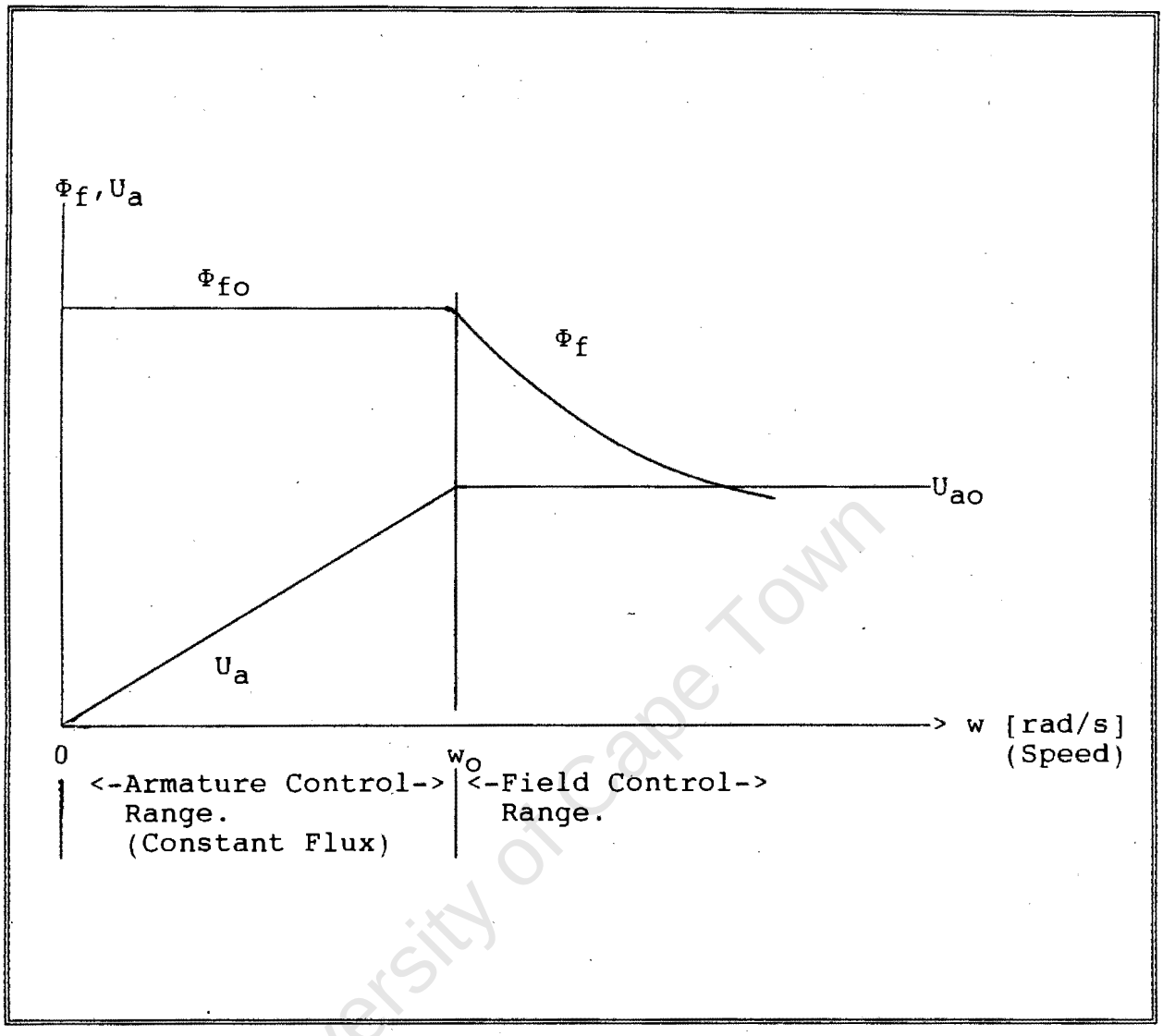


Figure 2.2: Control ranges of the D.C. motor.

Equations 2.1 - 2.4 can be translated into the block diagram of figure 2.3. If the load torque is ignored, the ensuing transfer function (relating the input voltage to the system output speed) describes a second order system. This is seen in equation 2.5.

$$g(s) = c*\Phi_f/[(J*L_a)*s^2 + (J*R_a)*s + (c*\Phi_f)^2] \quad \ldots(2.5)$$

The transfer function has poles at:

$$\delta_{1,2} = -R_a/2L_a \pm \{[(J*R_a^2 - 4*La*(c*\Phi_f)^2)/J]^{1/2}\}/2*L_a \quad \ldots(2.6)$$

Typical values for the particular parameters are [10]:

$J = 20\ [Nms^{\wedge}2]$
$c*\Phi_f = 0.6\ [Wb]$
$R_a = 0.2\ [\Omega]$
$L_a = 0.325\ [H]$

For these values, the poles are:

$\delta_1 = -0.118$
$\delta_2 = -0.5$

Since δ_2 is much faster than δ_1, it can be approximated by its steady state gain.

Thus a servo motor, whilst in actual fact a second order system, can be approximated by a system of first order.

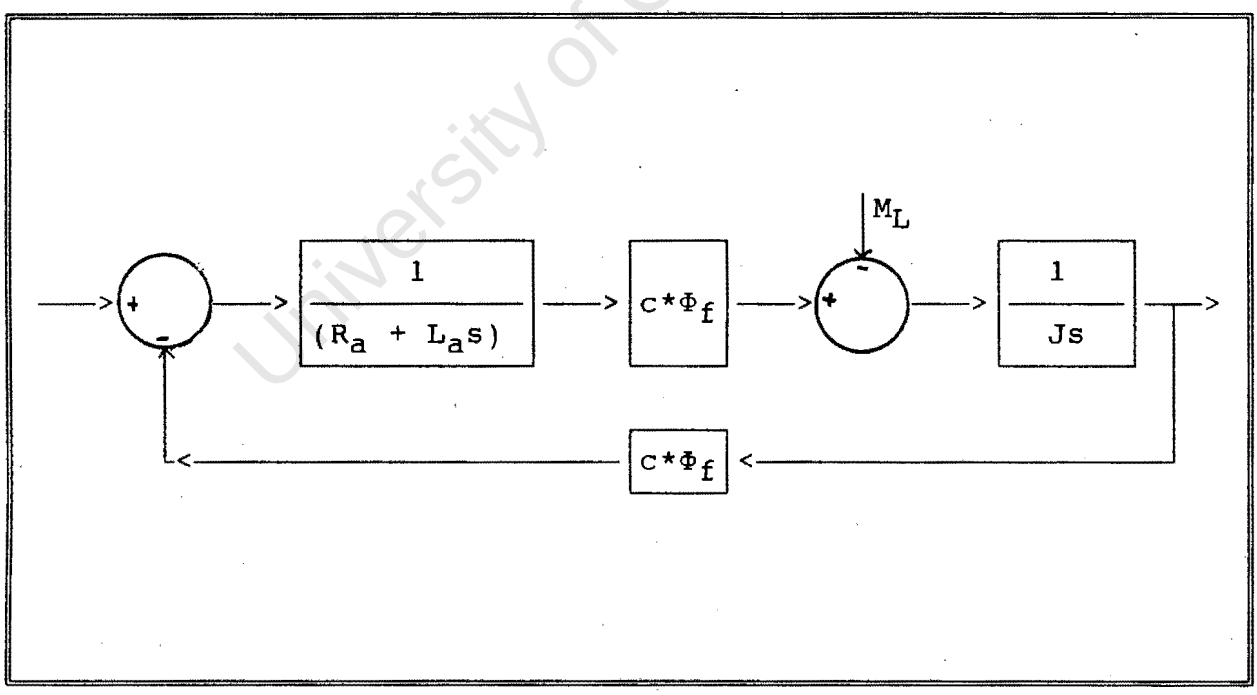


Figure 2.3: Block diagram for armature controlled motor.

To find the servo motor's particular parameter values, simple step tests were performed. A sample plot of the velocity output data for an input step of 50 counts is shown in figure 2.4, with its fitted first order transfer function.

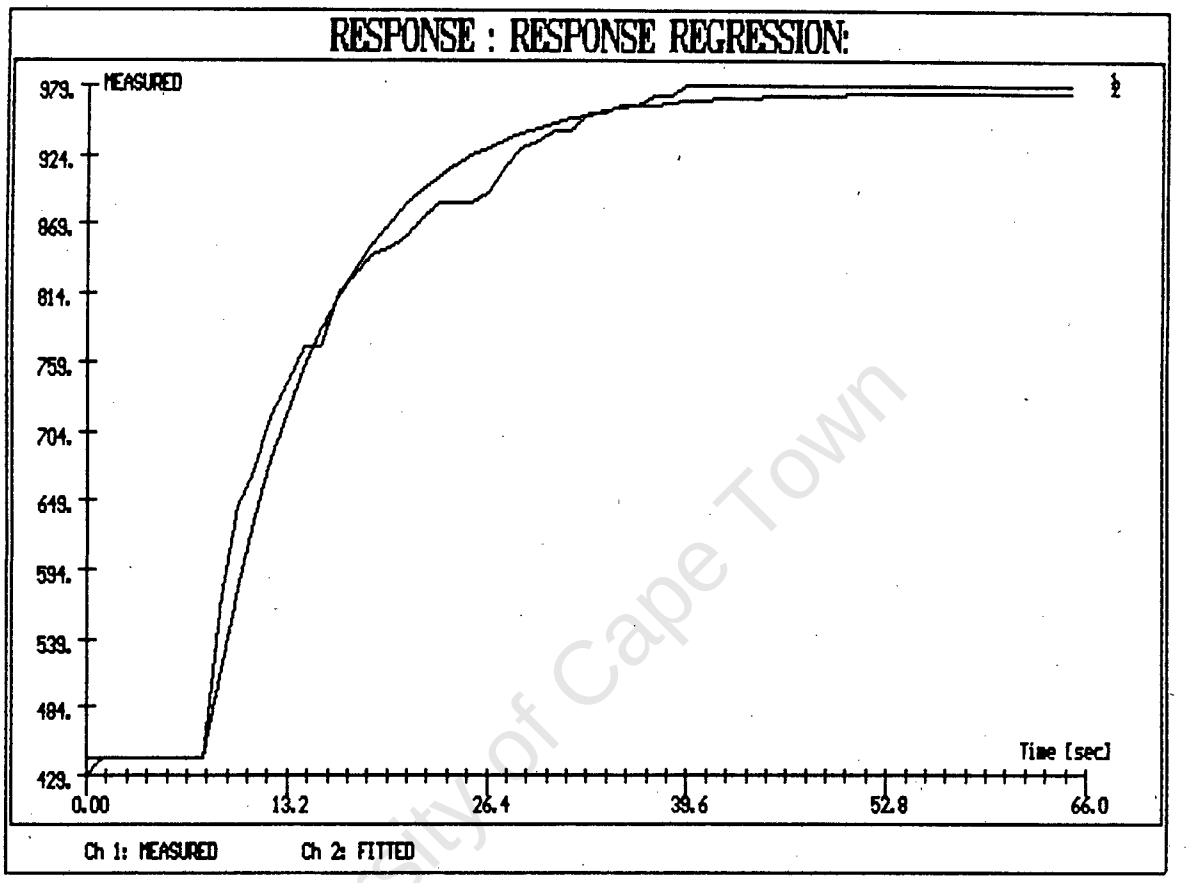


Figure 2.4 : Fitted response from a step test on the servo motor.

As can be seen from figure 2.4, the system can be adequately approximated by a first order response of the form:

$$G(s) = A/(1 + s*T) \quad \text{[counts/count]} \qquad \ldots(2.7)$$

where

Parameter A, which is equivalent to $c*\Phi_f$ divided by the steady state gain of the faster pole, is the system gain. It is measured in counts.

Parameter T, which is equivalent to the inverse of the dominant pole value, is the system time constant. It is measured in seconds.

The results of the step tests are tabulated below.

The input was stepped by 50 counts.

Test number	A [counts]	T [secs]
1.	10.286	8.0
2.	10.000	8.0
3.	11.250	9.5

Parameter	Average Value	Standard Deviation
A [counts]	10.512	0.65
T [secs]	8.5	0.87

Since the servo system is to be configured for position control, the transfer function of equation 2.7 is multiplied by the transfer function of the cascaded integral term (= 1/s). The resulting transfer function, relating the position output to the motor input has the following form:

$$G(s) = A/[s*(1 + s*T)] \quad [\text{counts/count}] \qquad \ldots(2.8)$$

A state space model for this transfer can be found of the form:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = -0.118*x_2(t) + 0.118*u(t) \qquad \ldots(2.9)$$

2.2 VARIABLE STRUCTURE CONTROL OF THE SERVO MOTOR

The variable structure control function is shown in the previous chapter to have the following form:

$$u(t) = -\Gamma * x_1(t) \qquad \ldots(2.10)$$

$$\Gamma = \alpha \text{ when } \sigma * x_1(t) > 0$$
$$= \beta \text{ when } \sigma * x_1(t) < 0 \qquad \ldots(2.11)$$

where

$x_1(t)$ is a position state [counts].
$x_2(t)$ is a velocity state [counts].
$u(t)$ is the scalar input [counts].

From equation 2.9, the plant parameters are:

$$a_1 = 0.000$$
$$a_2 = 0.118$$
$$b = 0.118 \qquad \ldots(2.12)$$

2.3 DESIGN SPECIFICATIONS

The design parameters for the considered VSS are α, β and c. They are chosen to satisfy the existence and reachability conditions of a sliding mode.

The existence of a SM is assured if equations 1.38 and 1.39 are satisfied. Hence:

$$\alpha > (0.118*c - c^{\wedge}2)/0.118 \quad [\text{counts}] \qquad \ldots(2.13)$$

$$\beta < (0.118*c - c^{\wedge}2)/0.118 \quad [\text{counts}] \qquad \ldots(2.14)$$

where

c is the constant which defines the slope of the switching line and hence the speed of the transient response, when the system is in the sliding mode. This is clearly seen from equation 1.41.

The hitting condition is given by equation 1.56. Hence:

$$0.118*\alpha > 0.003 \qquad \ldots(2.15)$$

From equation 1.41, it is evident that the speed of the transient response of the system in the sliding mode is defined by c. The open loop eigenvalues are:

$$\delta_{1,2} = -0.118,\ 0 \qquad \ldots(2.16)$$

This means that for position control, the system is open loop unstable.

Parameter c is chosen to satisfy a desired closed loop time constant of 10 seconds.

$$c = 1/10 = 0.1 \qquad \ldots(2.17)$$

The design procedure is now simply to choose suitable values for α and β such that equations 2.13, 2.14 and 2.15 are satisfied.

In second order systems, the phase portraits of the individual structures have a physical accessibility which does not occur for higher order systems. This meaningful physical understanding can thus be utilised in such systems, by analysing the roots of the characteristic equation. It should be noted that the existence and hitting conditions are, in general, all that need to be satisfied.

The roots of the characteristic equation are given by the following equation:

$$\delta_{1,2} = [-0.118 \pm \{(0.014 - 0.47*\Gamma)\}^{0.5}]/2 \qquad \ldots(2.18)$$

With c being defined by equation 2.17, equations 2.13 and 2.14 become:

$$\alpha > 0.015 \qquad \ldots(2.19)$$

$$\beta < 0.015 \qquad \ldots(2.20)$$

Choose $\alpha = 0.9$ [counts]
$\beta = -0.9$ [counts] $\qquad \ldots(2.21)$

with this choice of gain values, existence and hitting conditions are satisfied.

From equation 2.18, when $\Gamma = \beta = -0.9$, the roots of the characteristic equation are real and of unlike sign. The associated phase plane portrait can be seen in figure 2.5.

$$\delta_1 = -0.390$$
$$\delta_2 = 0.272 \qquad \ldots(2.22)$$

An added check on the c value to ensure sliding is:

$$0 < c < |\delta_1| \qquad \ldots(2.23)$$

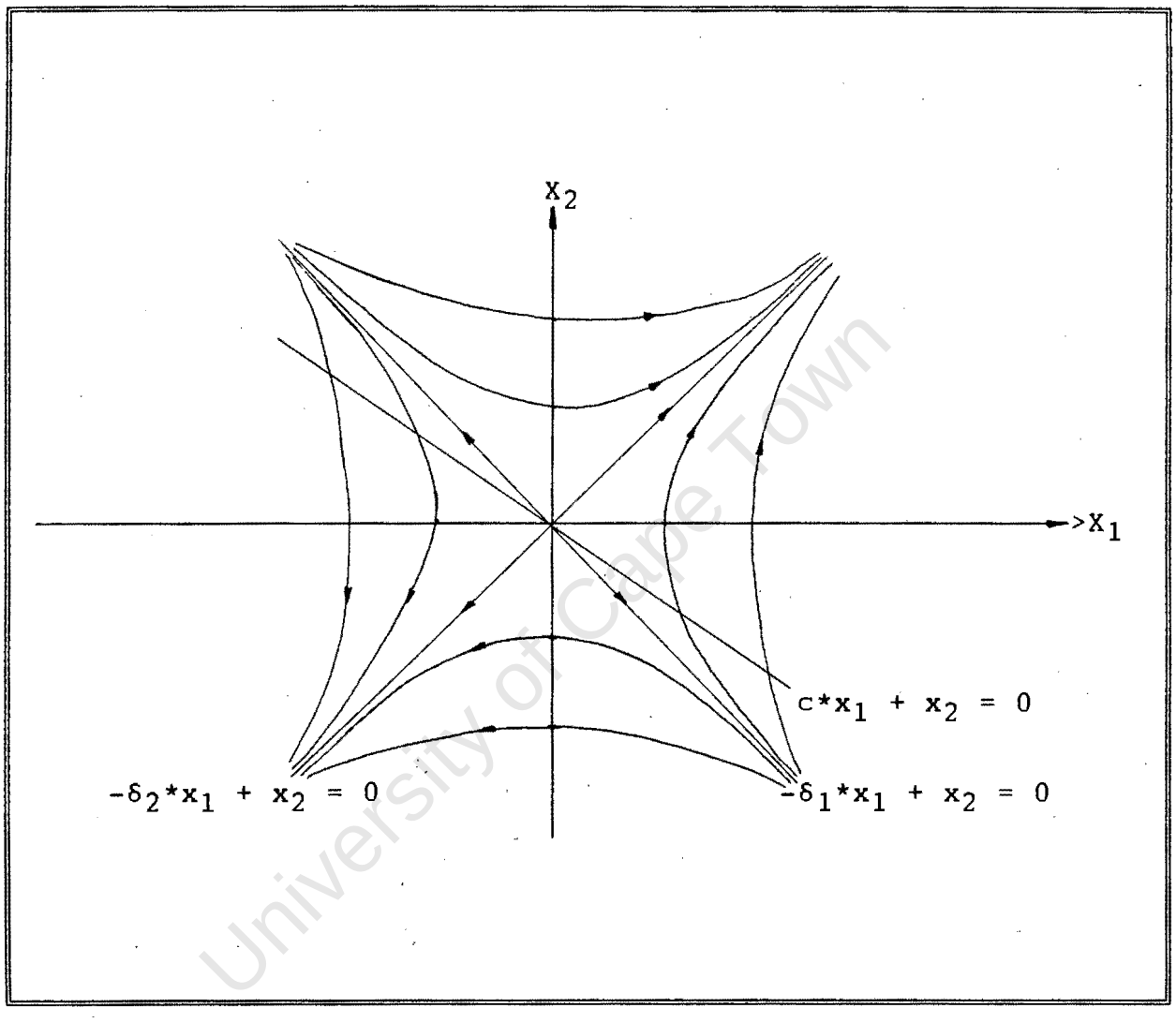


Figure 2.5: Phase portrait for Γ = β.

When $\Gamma = \alpha$, the ensuing structure has complex roots with negative real parts. The associated phase plane portrait is given in figure 2.6.

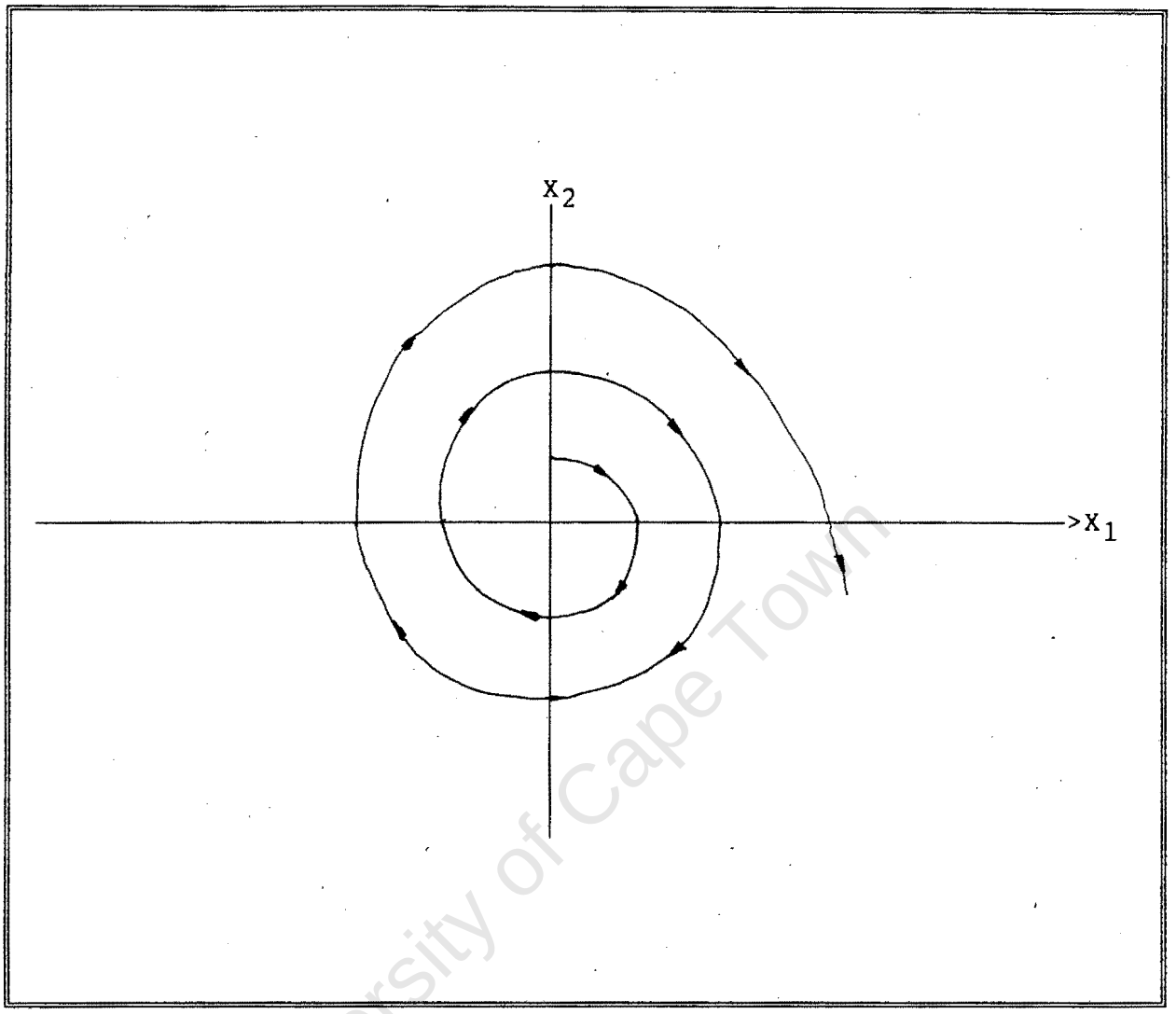


Figure 2.6: Phase portrait for $\Gamma = \alpha$.

2.4 SIMULATION OF THE CLOSED LOOP SERVO MOTOR USING VARIABLE STRUCTURE CONTROL

Before any experimentation was done on the servo motor itself, the system was simulated on an I.B.M. compatible personal computer. The simulator program was written in Turbo Pascal version 3.0 and is based on the Runge - Kutte fourth order method to solve a set of simultaneous differential equations (the program can be found in appendix 3).

EXPERIMENT 1:

The aim of this experiment, is to run a series of simulations with the phase plane representative point (RP) starting at specified areas in the phase plane, and to observe the transient response of the system dynamics.

The specified areas are numbered 1 - 4 in figures 2.7 and 2.8. They correspond to the areas defined by equations 1.9 and 1.10 in chapter 1.

The system has the structure associated with $\Gamma = \alpha$ in regions 1 and 3.

The system has the structure associated with $\Gamma = \beta$ in regions 2 and 4.

The closed loop time constant is 10 seconds. Thus, it was decided that the sampling time for all simulations is:

$$dt = 0.2 \quad [\text{seconds}] \qquad \ldots(2.24)$$

The system states will decay from their initial value (100%) to zero. They will reach within 2% of zero after a time given by:

$$\text{Time} = 4*T = 4/c = 40 \text{ seconds} \qquad \ldots(2.25)$$

Figure 2.7 illustrates the closed loop phase plane portraits of the system when the RP starts in region 1 or 3 of the phase plane.

Figure 2.8 illustrates the closed loop phase plane portraits of the system when the RP starts in region 2 or 4 of the phase plane.

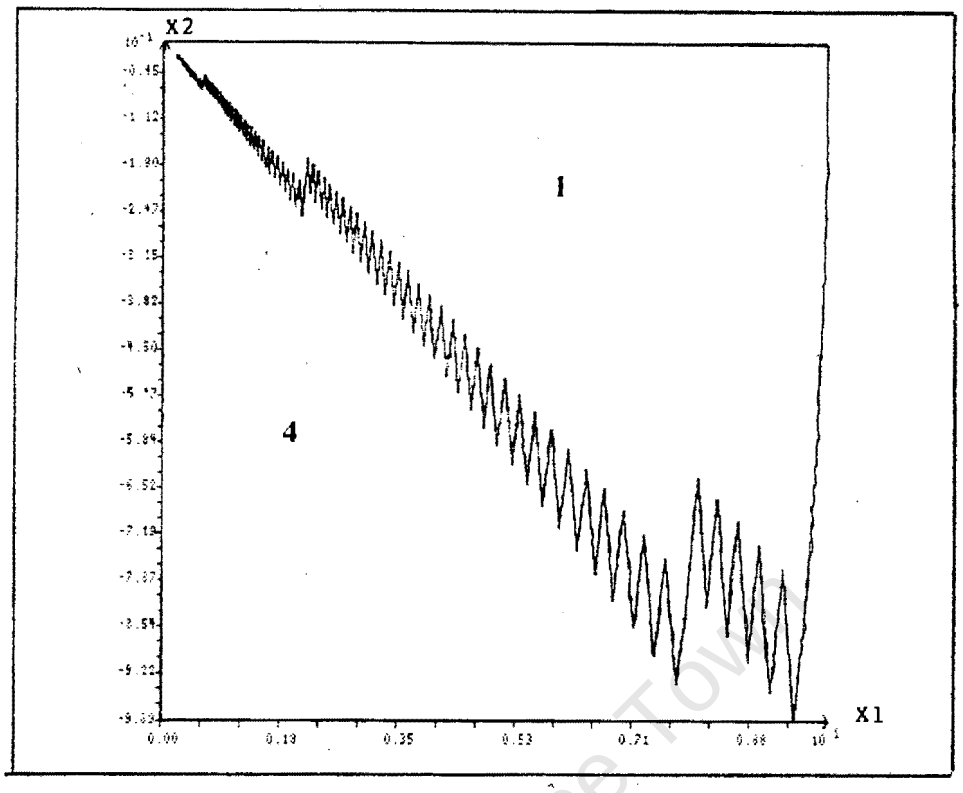
Figure 2.9 illustrates the plots of $x_1(t)$ plotted against time when the RP starts in regions 1 or 3 of the phase plane.

Figure 2.10 illustrates the plots of $x_1(t)$ plotted against time when the RP starts in regions 2 or 4 of the phase plane.

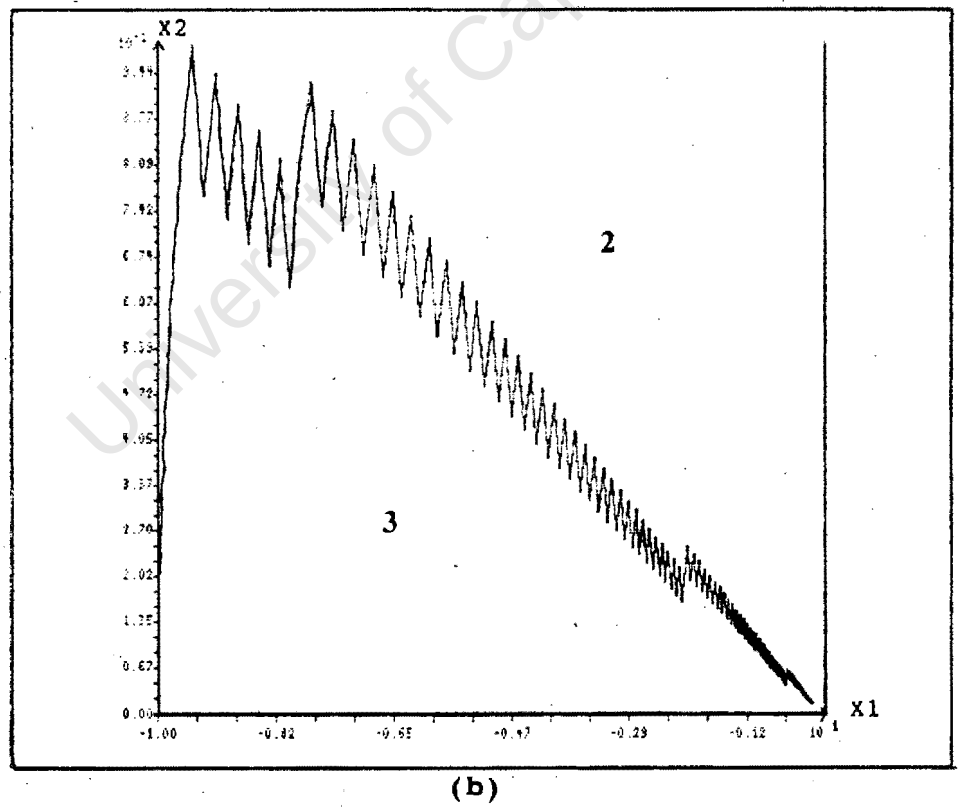
Figure 2.11 illustrates the plots of $x_2(t)$ plotted against time when the RP starts in regions 1 or 3 of the phase plane.

Figure 2.12 illustrates the plots of $x_2(t)$ plotted against time when the RP starts in regions 2 or 4 of the phase plane.

Figure 2.13 illustrates the plots of the input function plotted against time when the RP starts in regions 1 and 2 (respectively) of the phase plane.

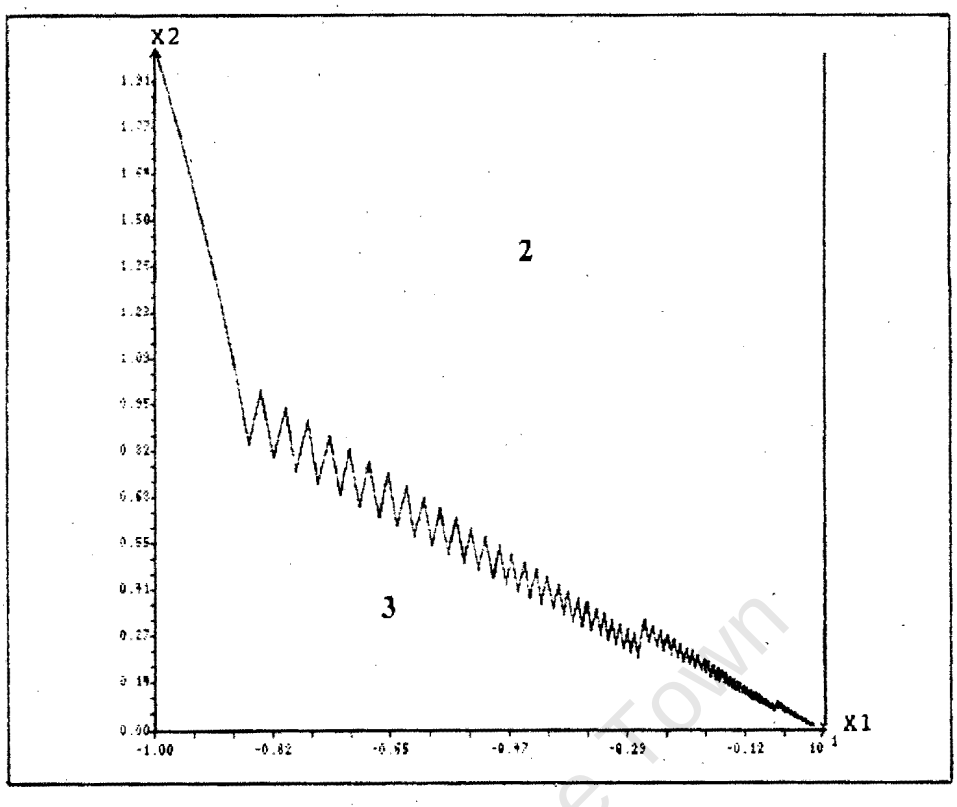


(a)

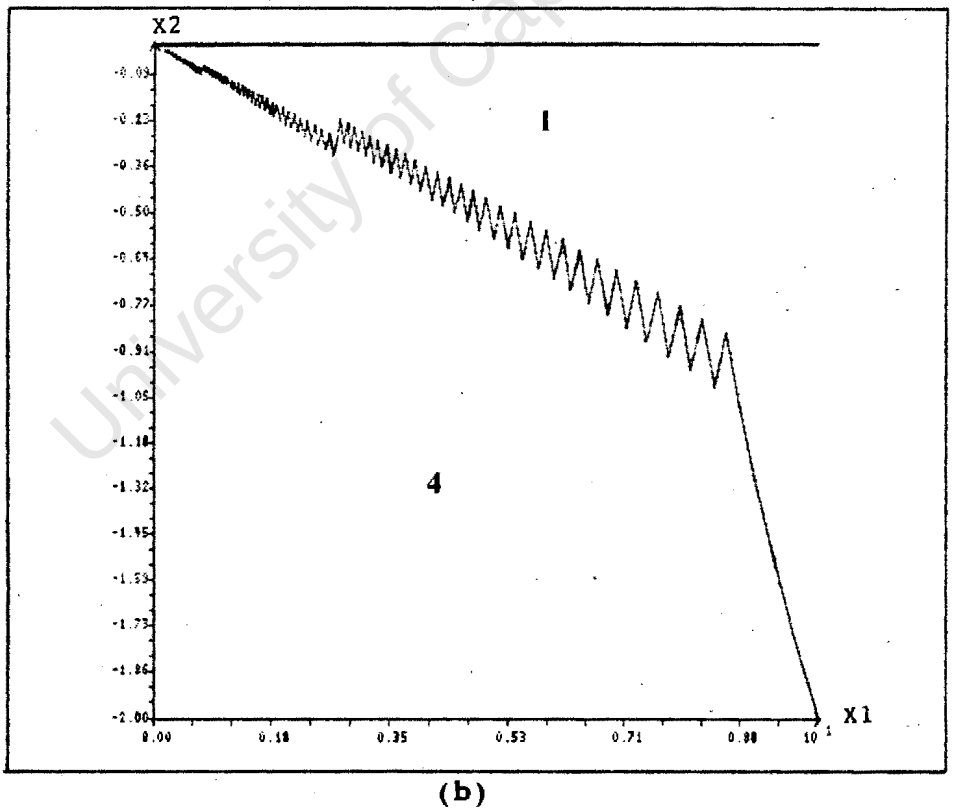


(b)

Figure 2.7: System phase plane portraits when the RP starts in (a) region 1 and (b) region 3.

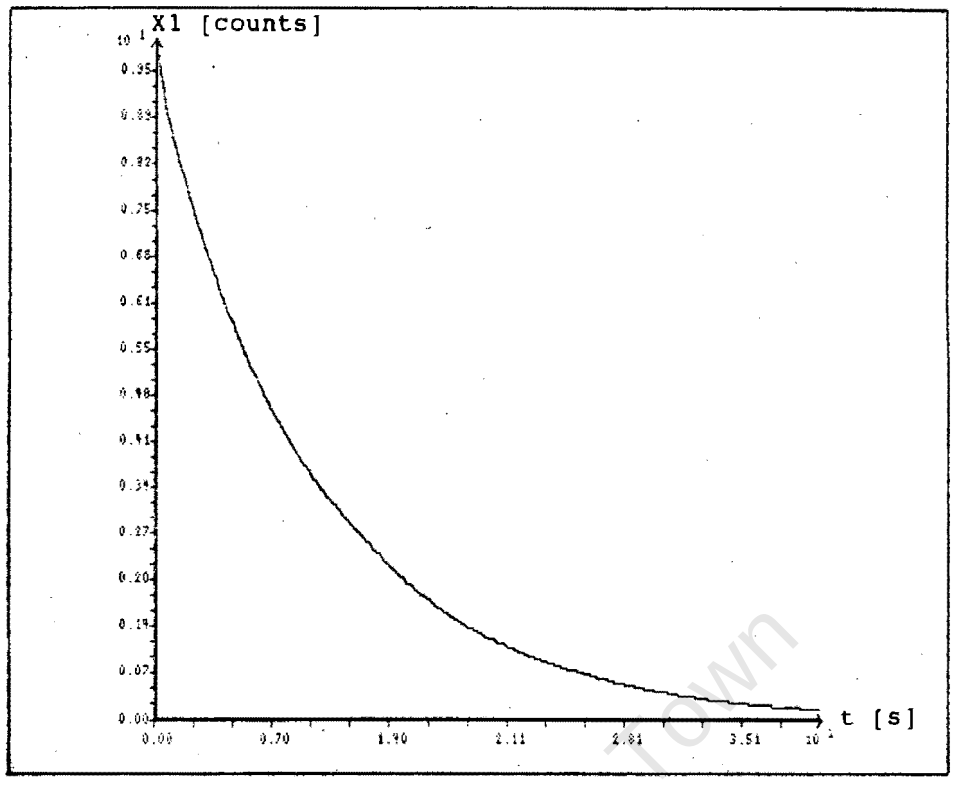


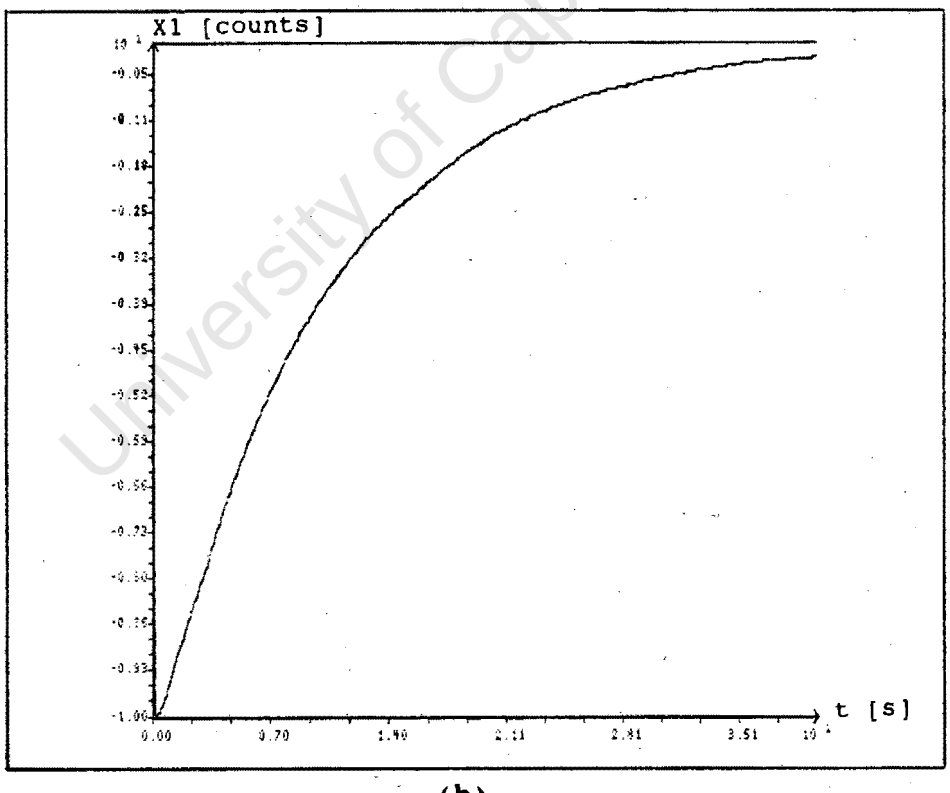
(a)



(b)

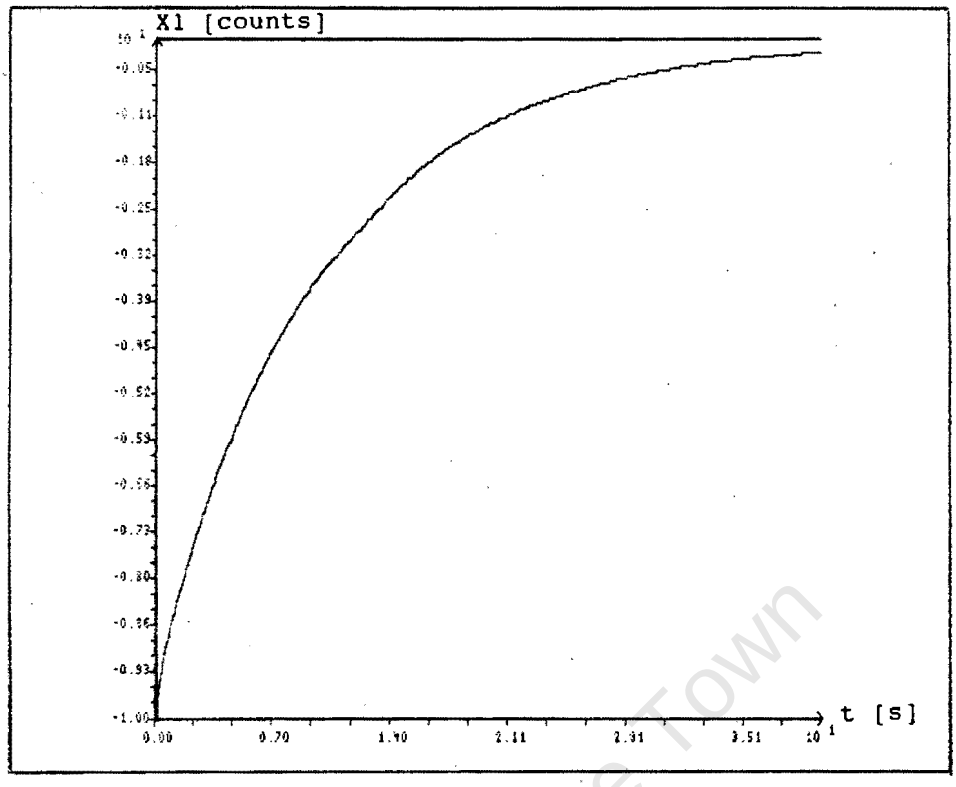
Figure 2.8: System phase plane portraits when the RP starts in (a) region 2 and (b) region 4.



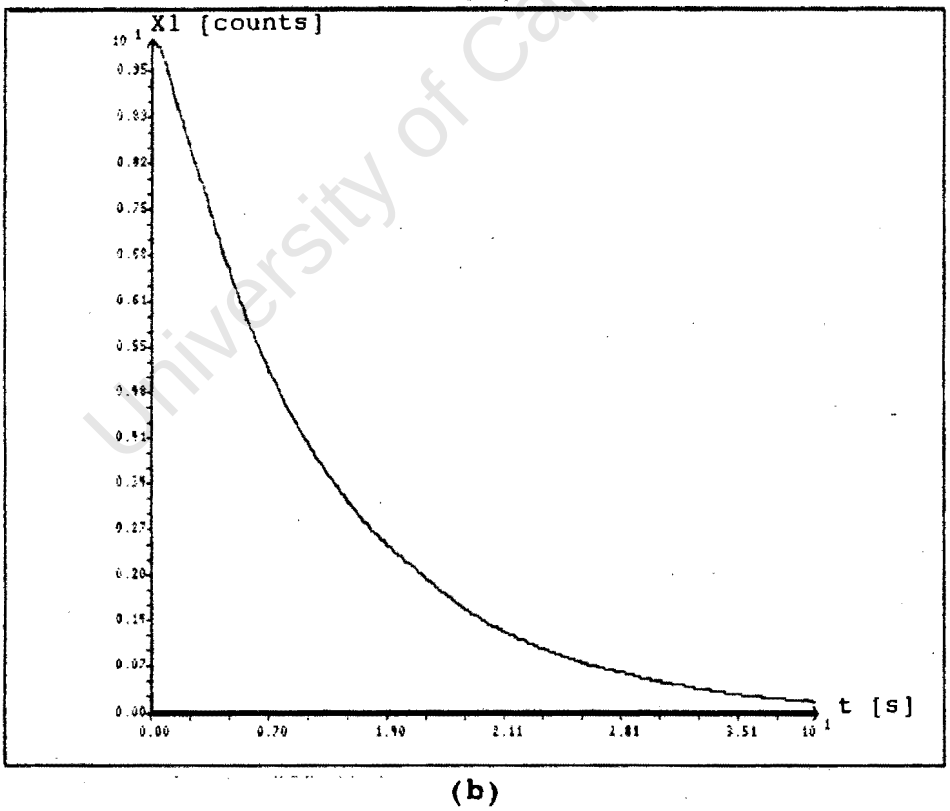


(b)

Figure 2.9: X_1 vs Time when the RP starts in (a) region 1 and (b) region 3.

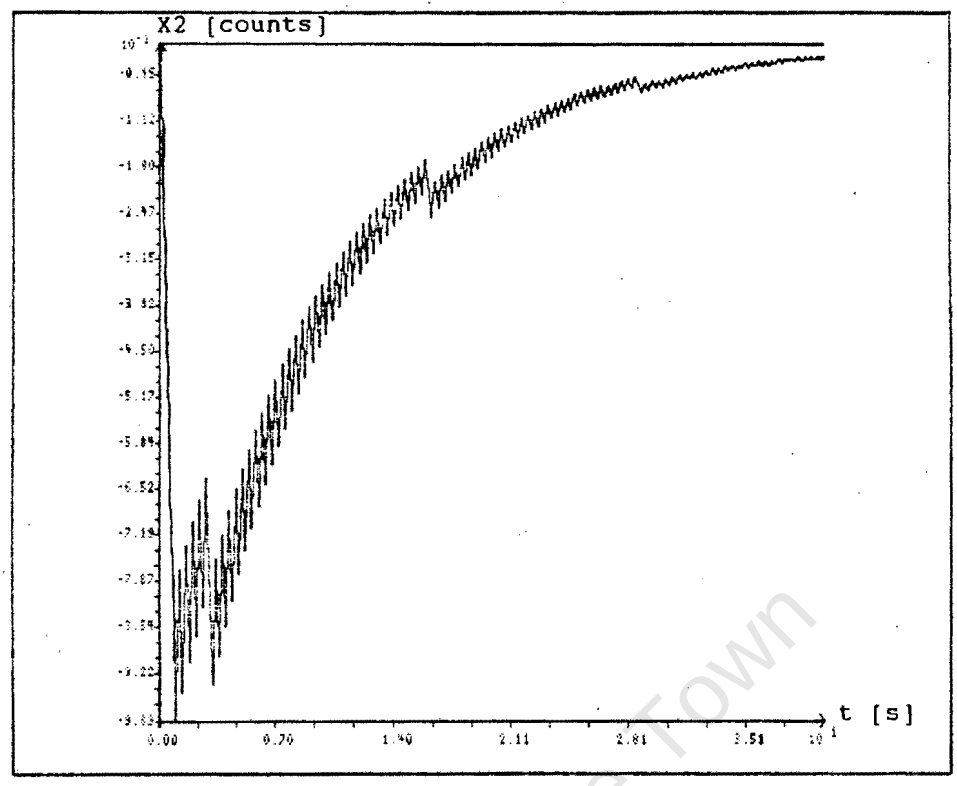


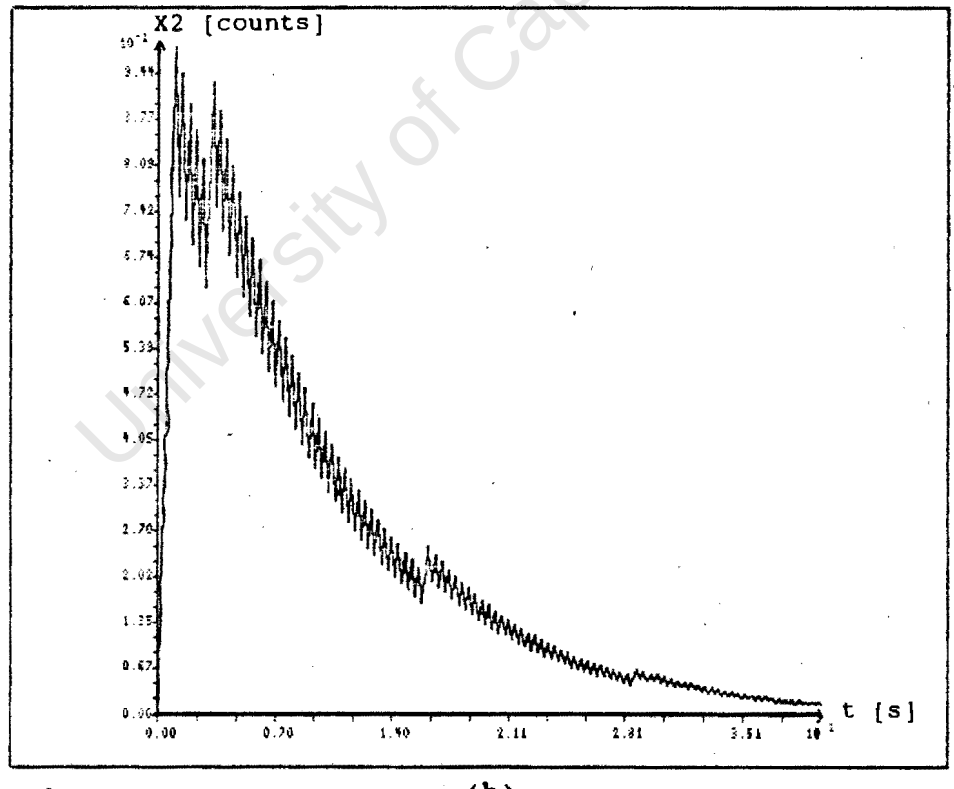
(a)



(b)

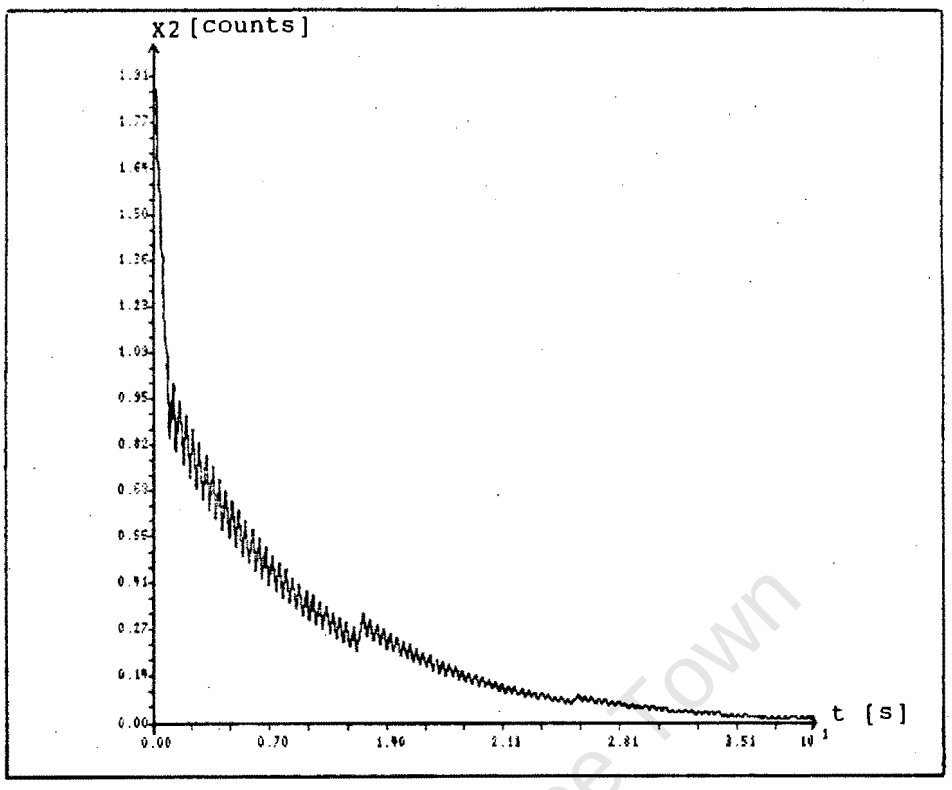
Figure 2.10: X_1 vs Time when the RP starts in (a) region 2 and (b) region 4.



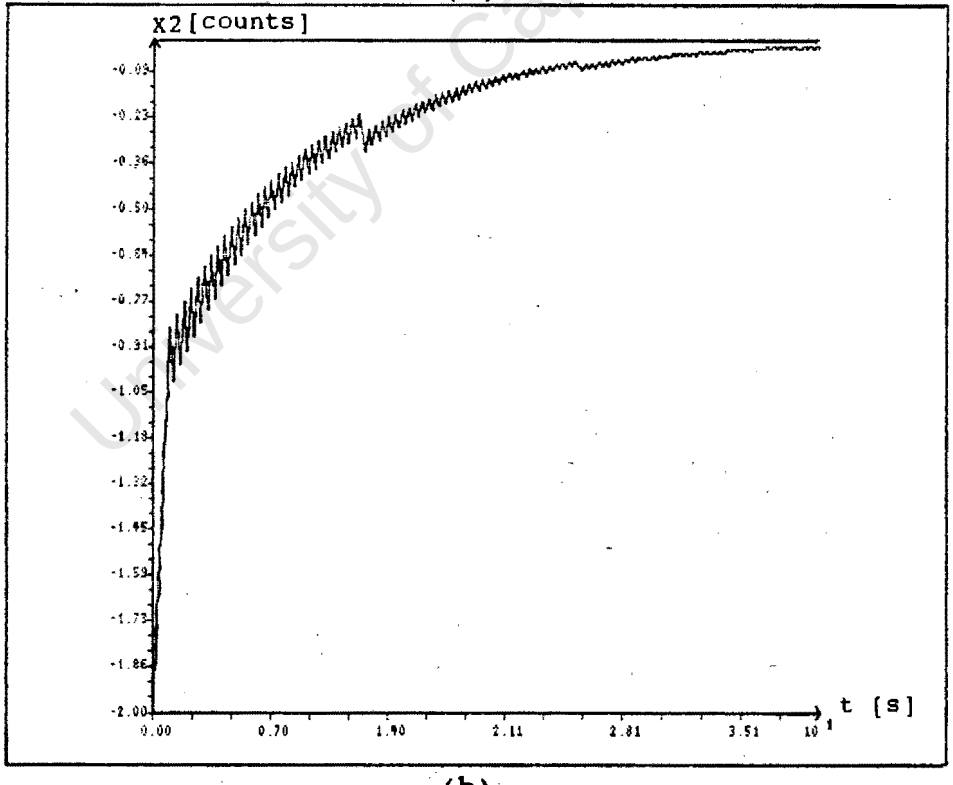


(b)

Figure 2.11: X_2 vs Time when the RP starts in (a) region 1 and (b) region 3.

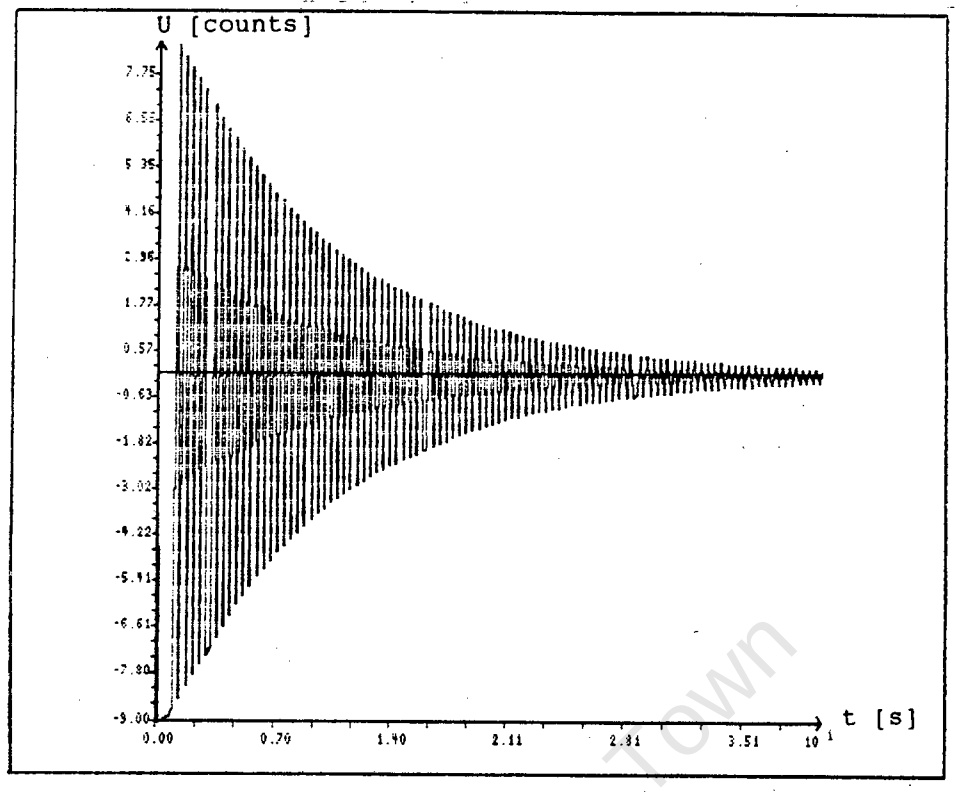


(a)

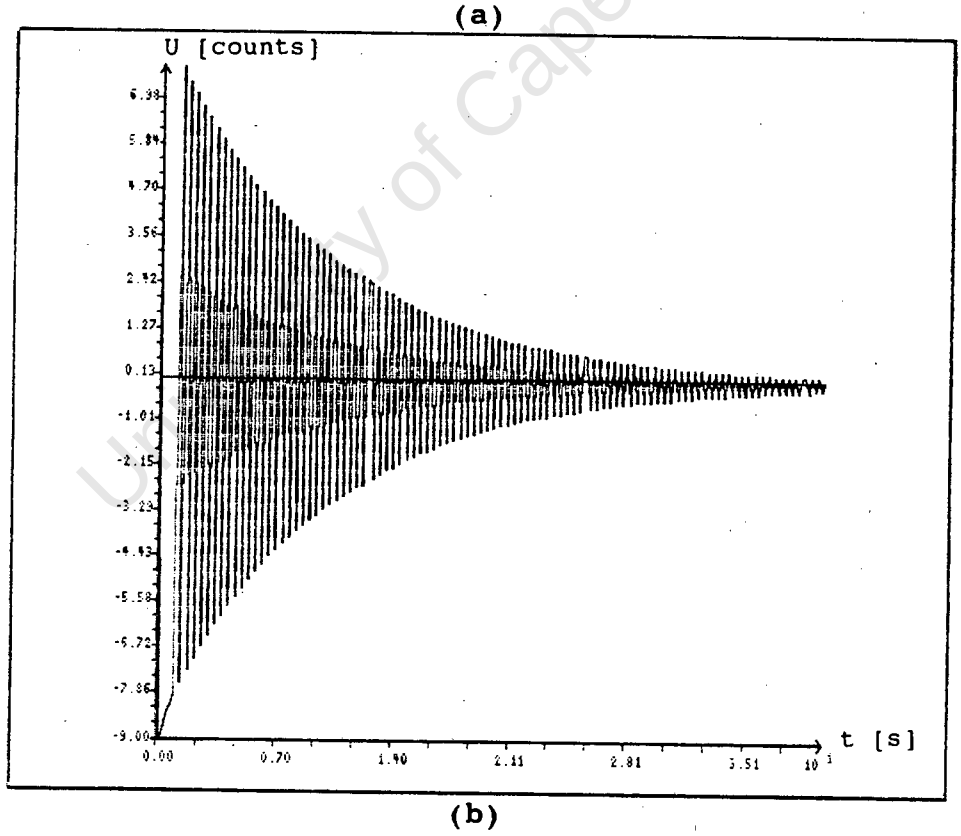


(b)

Figure 2.12 : X_2 vs Time when the RP starts in (a) region 2 and (b) region 4.



(a)



(b)

Figure 2.13 : Input vs Time when the RP starts in (a) region 1 and (b) region 2.

From the above figures, it is evident that the VSS is globally stable.

The concept of sliding has been illustrated in the phase plane.

The transients of the system states decay from their initial value (0%) to their final value (100%). They reach within 2% of their final value in about 40 seconds; as predicted.

EXPERIMENT 2:

The aim of this experiment is to repeat the above simulations with an incorporated setpoint for the position output.

In chapter 1 it was noted, that to implement a setpoint, the output is generally made to equal to state x_1. An error state, $e_1(t)$, is defined by subtracting the output from the setpoint. The second error state, $e_2(t)$, is obtained by differentiating $e_1(t)$.

In practice, differentiation is to be avoided wherever possible.

A closer look at the considered system equations however, reveals that differentiation is not necessary.

Specifically:

$$e_1 = r - x_1 \qquad \ldots(2.26)$$

$$u = -\Gamma * e_1 \qquad \ldots(2.27)$$

$$x_2 = -de_1/dt \qquad \ldots(2.28)$$

$$de_2/dt = -0.118*e_2 + 0.118*u(t) \qquad \ldots(2.29)$$

Equation 2.29 has the identical form of the original system. Instead of differentiating e_1 to get the second state, all that is required is to reverse the sign of the incoming x_2 state.

This reversal in sign, however, means a change in sign for α and β (from the general condition for the existence of a SM in equation 1.27).

Thus:

$$\alpha = -0.9 \text{ [counts]} \qquad \ldots(2.30)$$

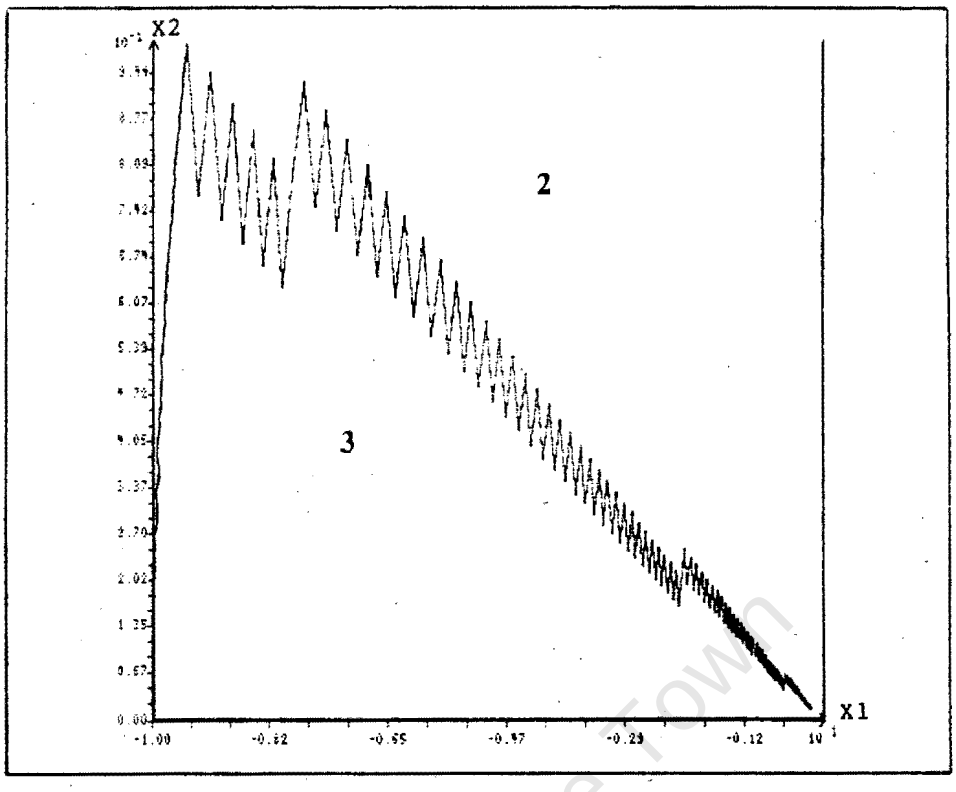
$$\beta = 0.9 \text{ [counts]} \qquad \ldots(2.31)$$

Figure 2.14 illustrates the closed loop phase plane portraits of the system when the RP starts in region 3 or 2 of the phase plane.

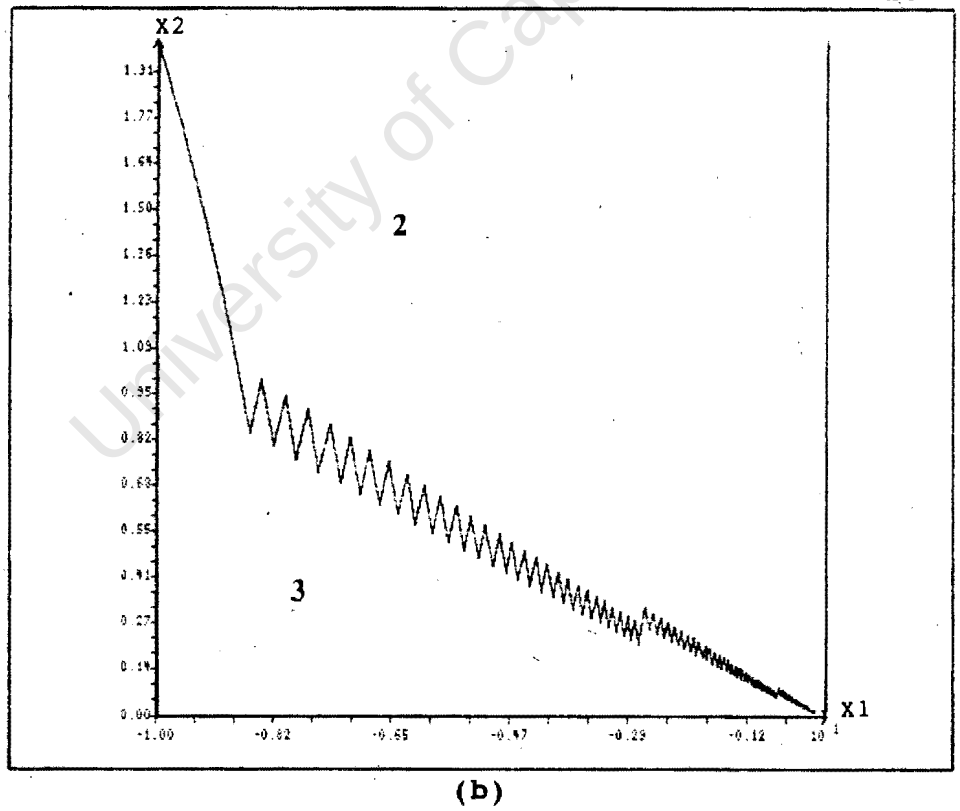
Figure 2.15 illustrates the plots of X_1 vs Time for the cases when the RP starts in region 3 and 2 of the phase plane.

Figure 2.16 illustrates X_2 plotted against time for the cases when the RP starts in region 3 and 2 of the phase plane.

Figure 2.17 illustrates the input function plotted against time for the cases when the RP starts in region 3 and 2 of the phase plane.

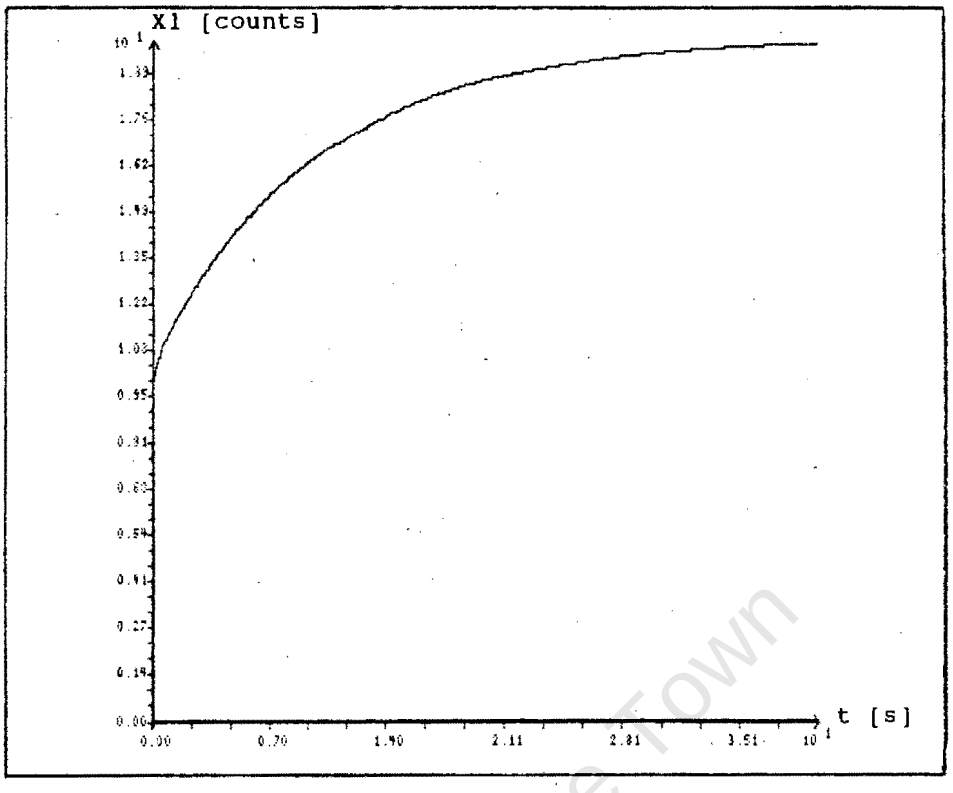


(a)

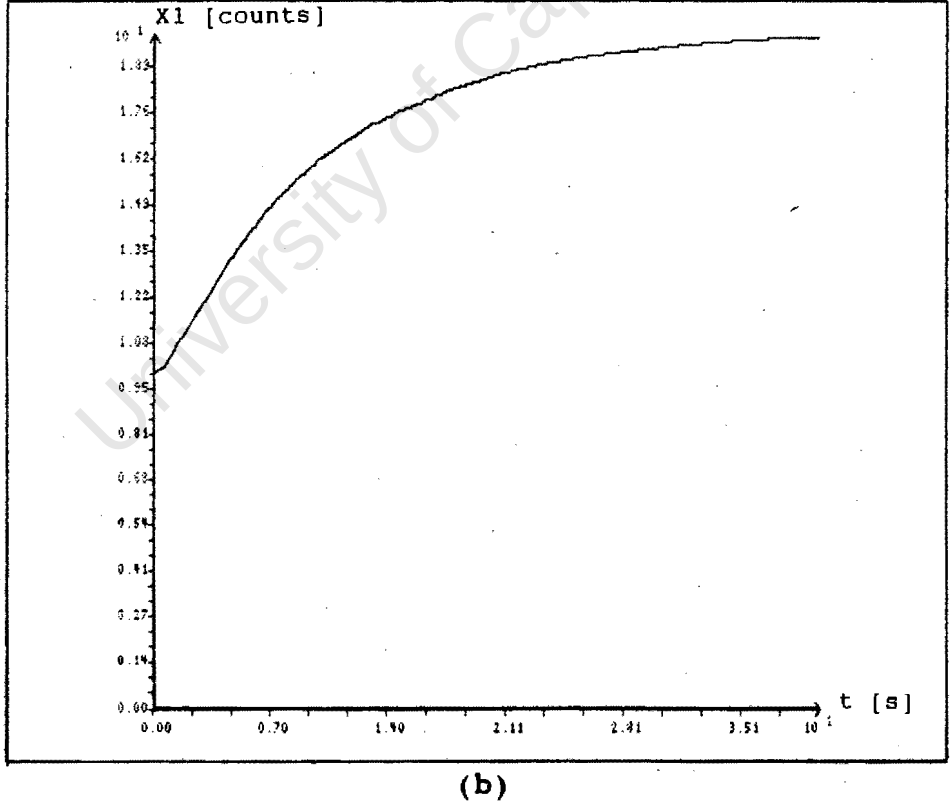


(b)

Figure 2.14: System phase plane portraits when the RP starts in (a) region 3 and (b) region 2.

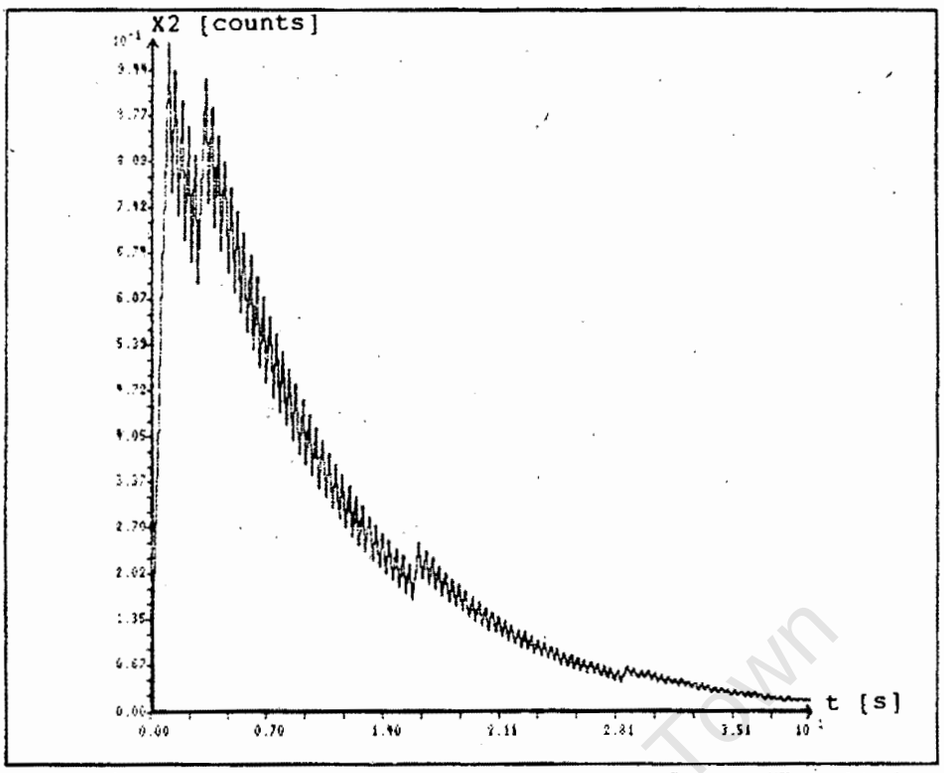


(a)

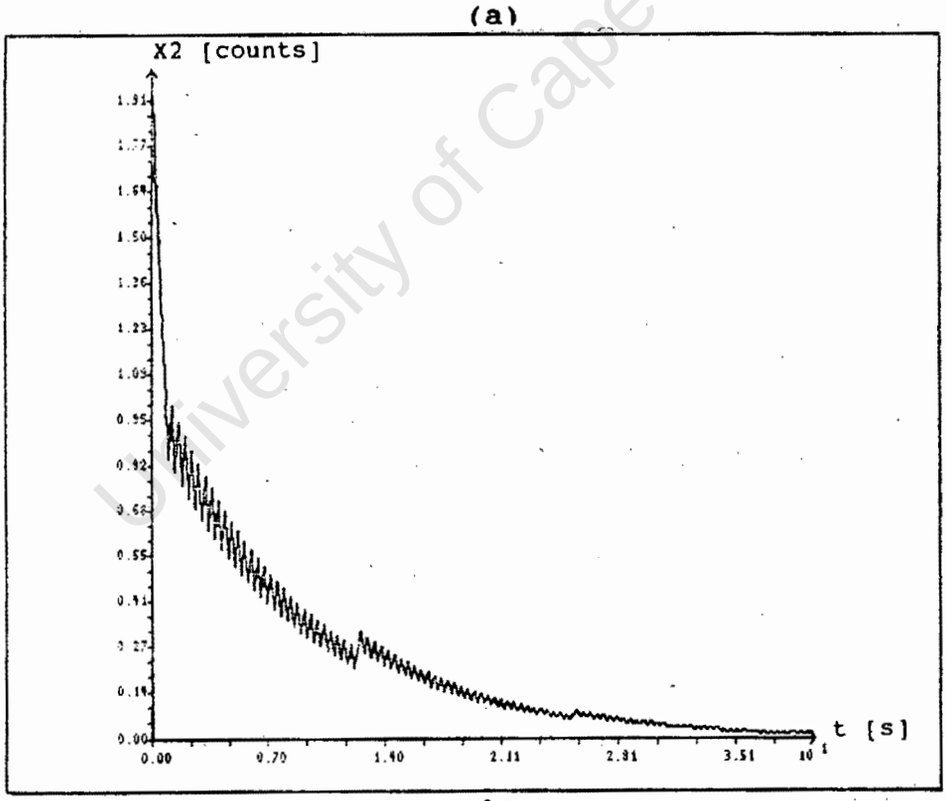


(b)

Figure 2.15: X_1 vs Time when the RP starts in (a) region 3 and (b) region 2.



(a)



(b)

Figure 2.16: X_2 vs Time when the RP starts in (a) region 3 and (b) region 2.

From the above figures, it is evident that a setpoint has been succesfully implemented.

Once again, the transients of the states decay from their initial value (0%) to their final value (100%). They reach within 2% of their final value in about 40 seconds; as predicted.

A practical note of concern is the high frequency input. This problem will be looked at again in more detail.

2.5 PRACTICAL IMPLEMENTATION OF A VARIABLE STRUCTURE CONTROLLER FOR THE SERVO MOTOR SYSTEM

Attention is now turned to the physical implementation of a variable structure controller for the servo motor.

The control program was written in Turbo Pascal version 3.0 (see appendix 4).

In order to reduce the fast switching of the input signal, a first order low pass filter was implemented in the control software. This is justifiable if its pole is fast enough (relative to the system's poles as seen in the s plane) so that the servo remains an effective second order system (This includes the integral term of the position output potentiometer).

An acceptance test routine was run which was defined by the following stages:

1. From 0 to 50 samples, the system was in open loop with no input being applied to the servo.

2. From 50 to 200 samples, the system was in closed loop control and the setpoint = 500 counts.

3. From 200 to 400 samples, the system was in closed loop control and the setpoint = 1000 counts.

4. From 400 to 600 samples, the system was in closed loop control and the setpoint ramped down by 3 counts every sample.

The parameters of the closed loop system were chosen to satisfy equations 2.13 - 2.15 (existence and hitting conditions).

Specifically:

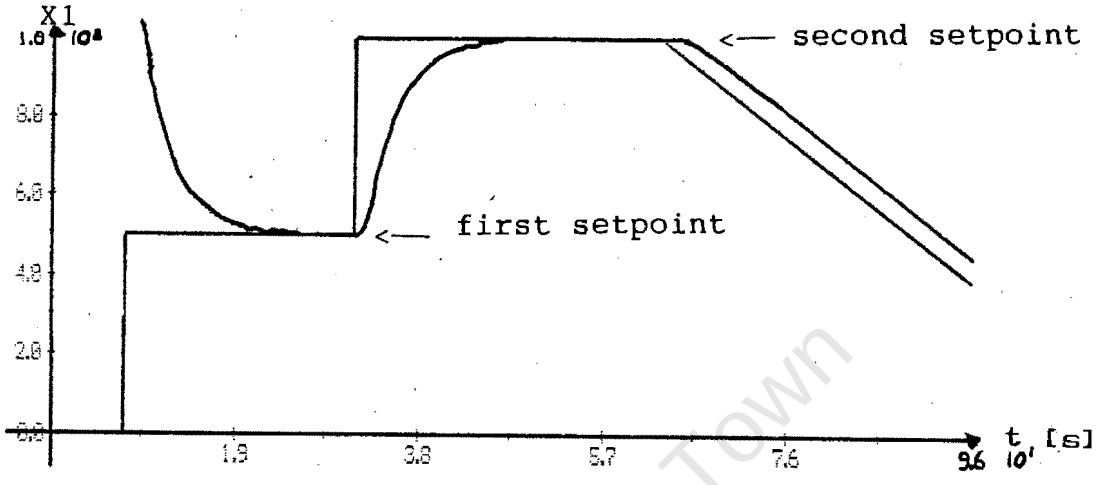
$$c = 0.3$$
$$\alpha = -0.9 \text{ [counts]}$$
$$\beta = 0.9 \text{ [counts]} \qquad \ldots(2.32)$$

These parameters ensure a VSS which is made up of the same component structures as the previous design.

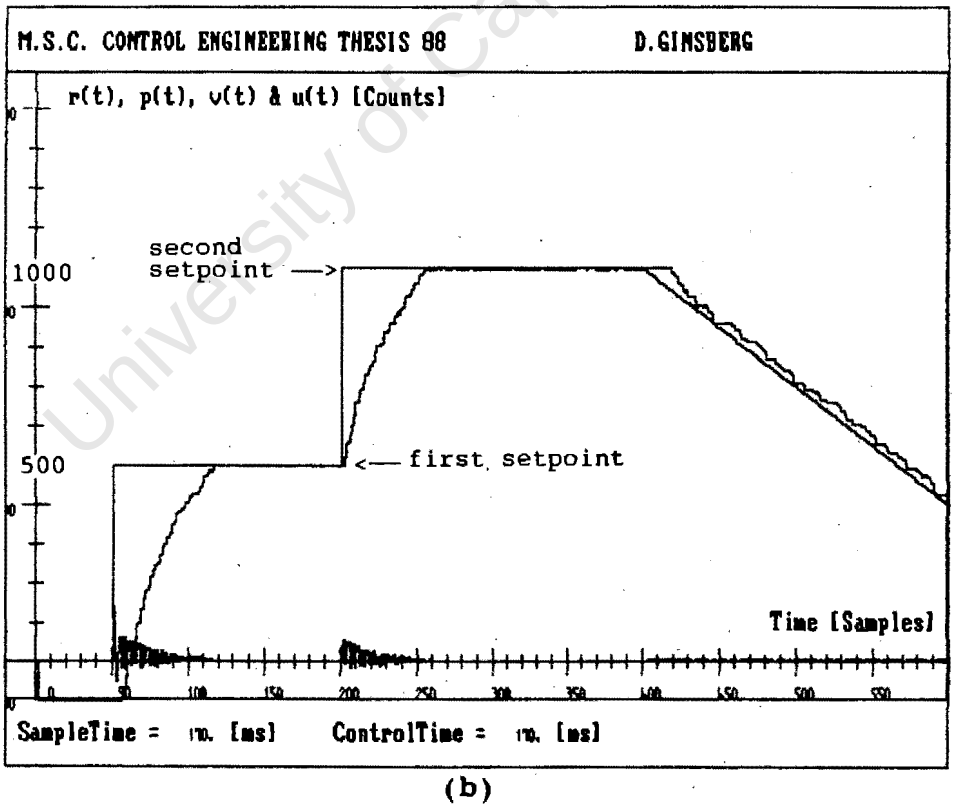
The choice of c corresponds to a system time constant of 3.33 seconds.

Figure 2.18 shows the position control of the servo motor in simulation and in practice.

Figure 2.19 shows the ensuing phase plane portrait.



(a)



(b)

Figure 2.18:Servo output vs time in (a) simulation and (b) in practice.

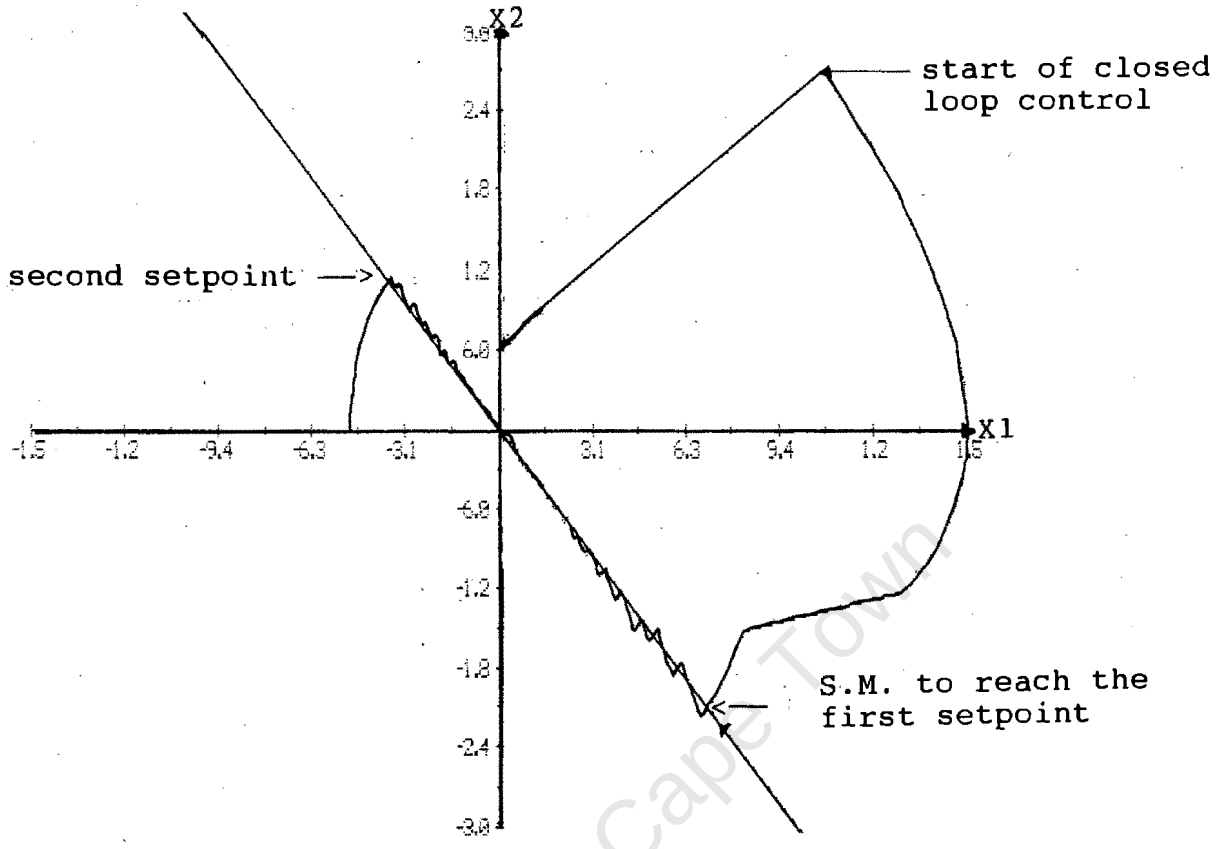


Figure 2.19: Phase plane portrait.

From the above figures, it is evident that the results obtained from the practical implementation of a variable structure controller closely resemble those obtained in simulation.

There is a small error when the controller tries to track a rapidly changing setpoint.

The high frequency input is seen to die away very quickly for the cases when a constant setpoint is defined. The input consists of high frequency, small oscillations, when the controller attempts to track athe rapidly changing setpoint.

CONCLUDING REMARKS ON THE CHAPTER

The aim of this chapter was to synthesize the relevant Variable Structure System theory with the theory governing the behaviour of a servo motor and highlight practical areas of note.

The first practical problem focused on was that of modelling the motor. It was shown how the theoretically obtained order of the model could be legitimately reduced. Practical step tests performed on the motor supported this reduction in system order.

Once the model had been put into the state space format, a variable structure design was made. This design was then tested in simulation. The results obtained were as predicted.

Analysis of the system equations showed that the problem of having to differentiate the error state to implement a setpoint, could be circumvented.

A practical variable structure controller was successfully implemented on the servo motor. This highlighted a number of practical issues.

A low pass filter, which was cascaded to the motor input to reduce the high frequency switching, did not noticeably effect the system performance; thus highlighting variable structure system's insensitivity to plant parameter changes when in the sliding mode.

The above figures show that the theoretical plots correspond well to those obtained in practice.

Variable structure controllers have been shown to be fast, robust and easy to design for a single input servo motor system.

CHAPTER 3

NON - IDEAL SYSTEMS OF VARIABLE STRUCTURE

This chapter is primarily concerned with highlighting the effects of various non - ideal characteristics in practical variable structure systems (VSS). Emphasis will be placed on systems with digitally implemented controllers.

Owing to the physical limitations of viewing the phase plane in more than two dimensions, this chapter deals primarily with second order systems. The practical insight attained in studying such systems provides a fundamental understanding when working with systems of higher order.

A more realistic view of the behaviour of the phase trajectory in the phase plane will be given.

When attempting to describe noise of a general nature for a general system, the description becomes extremely cumbersome and all physical and practical insight into the problem is lost. Thus, in keeping with the policy of the chapter to provide practical insight into areas of concern, only selected types of disturbances and systems will be looked at. These highlight where non - ideal VSS differ from theoretical VSS.

It will be shown that even though practical systems depart from ideal VSS, their transients are not significantly different from the ideal. There is, however, a direct dependence between the level of non - ideal behaviour and the degree of approximation.

Although VSS can be designed for specific non - ideal characteristics, such as a pure delay in the plant or sampling the data from the plant (see [8]), the specific existence and hitting (of a SM) conditions which overcome these characteristics will not be proved here. The mathematics is cumbersome to deal with and provides no practical insight into the non - ideal characteristic.

3.1 PROBLEM AREAS IN THE TWO - DIMENSIONAL PHASE PLANE WHEN SWITCHING SECOND ORDER STRUCTURES

In this section, problem areas in the two - dimensional phase plane will be isolated and the effect on the performance of switching will be analysed.

It is of primary importance to obtain conditions whereby correct switching of structures is guaranteed and to isolate the areas in the phase plane where this cannot be done.

Regardless of their nature, all system characteristics departing from the ideal will be lumped together and called noise. They will be represented by the following function:

$$|\tau_j(t)| \leq \delta_j + v_j * L(t) \quad (j = 1,2) \qquad \ldots(3.1)$$

where

δ_j, v_j $(j = 1,2)$ are all nonnegative functions.

$L(t)$ is the norm in the two - dimensional phase space and is defined by:

$$L(t)^{\wedge}2 = x_1(t)^{\wedge}2 + x_2(t)^{\wedge}2 \qquad \ldots(3.2)$$

$\tau_1(t)$, $\tau_2(t)$ are noise functions added to the x_1 and x_2 states respectively. In effect, this means that the measured states are given by:

$$x_{mi} = x_i + \tau_i \qquad \ldots(3.3)$$

Itkis shows in [8], that such a representation of noise suitably describes the effects of a number of non - ideal features in practical VSS; including pure delays in the plant and controller.

3.1.1 SYSTEM SWITCHING LINE

In reality, the structure of a system is switched in a finite time interval at a high but not infinite frequency. This is due to, amongst other effects, the sampling time of the system. This, together with the other delays of the system, means that a second order system switches when:

$$\sigma(t) + del\sigma(t) = 0 \qquad \ldots(3.4)$$

where

$$del\sigma(t) = c*\tau_1 + \tau_2 \qquad \ldots(3.5)$$

Thus it is shown in figure 3.1, that instead of switching on the switching line, $x_2(t) + c*x_1(t) = 0$, the system switches on a sector of the phase plane about the switching line. The sector is defined by the lines:

$$x_2(t) + (c-m)*x_1(t) = 0 \qquad \ldots(3.6)$$

$$x_2(t) + (c+m)*x_1(t) = 0 \qquad \ldots(3.7)$$

where

m is a positive constant added to or subtracted from c and is a function of the noise signals.

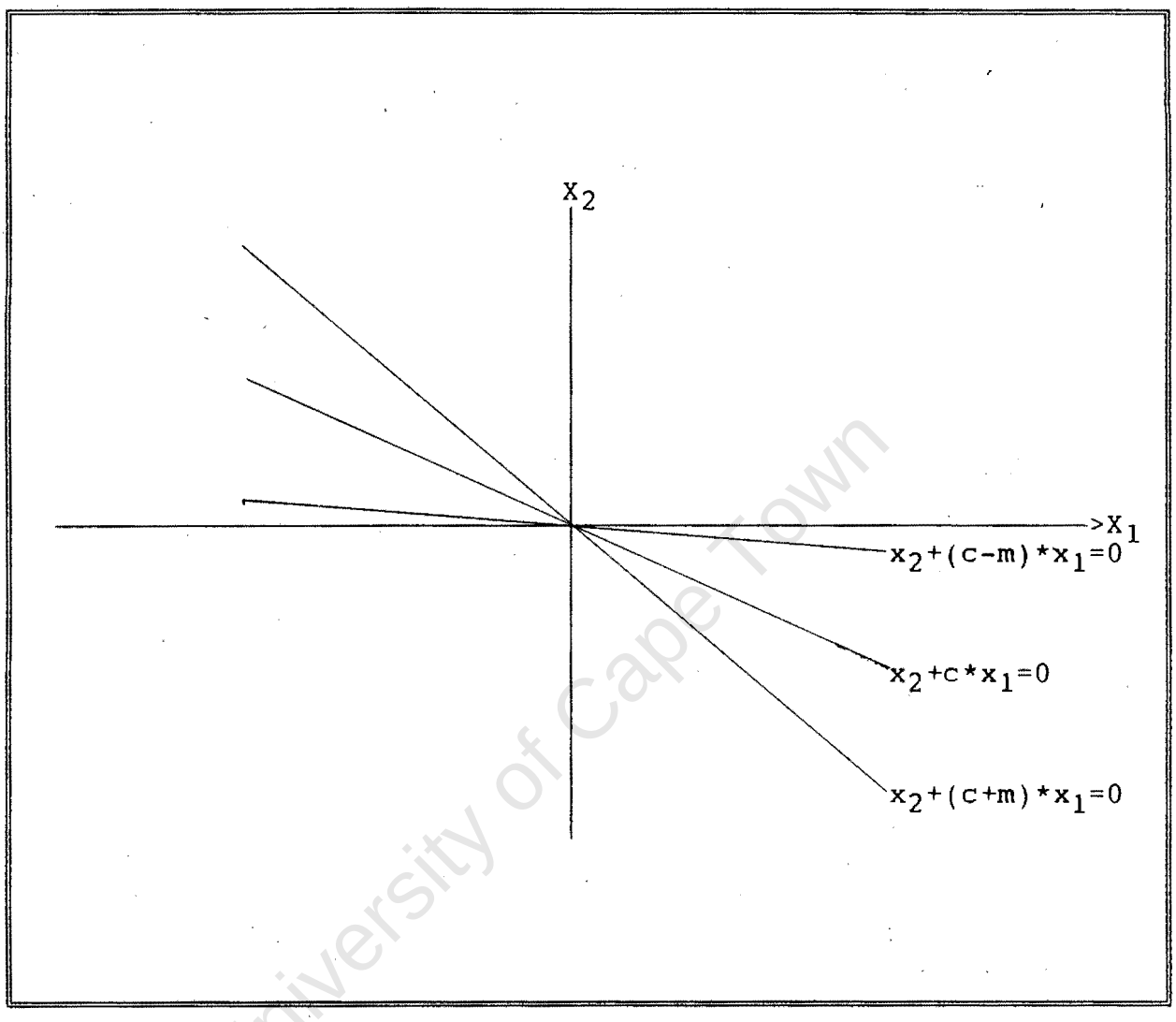


Figure 3.1: The Sliding Area for second order systems.

An area in the phase plane has been isolated where practical switching differs from theoretical switching. Instead of switching about a line, the system switches about an area. This "quasi" - sliding regime replaces the theoretical concept of switching at an infinite frequency with infinitesimal amplitude. The consequences of this non - ideal behaviour will become apparent later in this chapter.

This area will be referred to as the **Sliding Area**.

3.1.2 AREA ABOUT THE ORIGIN

For a system to be stable, its states must reach the origin of the phase plane. Owing to its strategic importance, some minimum area about the origin is thus the second area in the phase plane to be isolated.

This minimum area about the origin is dependent on the noise functions and is defined as:

$$L_o = (c*\delta_1 + \delta_2)/\{m/(1 + c + m) - c*v_1 - v_2 \} \qquad ...(3.8)$$

When the phase plane representative point (RP) is inside this area:

$$L_o\hat{}2 > x_1(t)\hat{}2 + x_2(t)\hat{}2 \qquad ...(3.9)$$

By equation 3.2, equation 3.9 can be written:

$$L_o\hat{}2 > L(t)\hat{}2 \qquad ...(3.10)$$

Figure 3.2 shows this area in the second order phase plane.

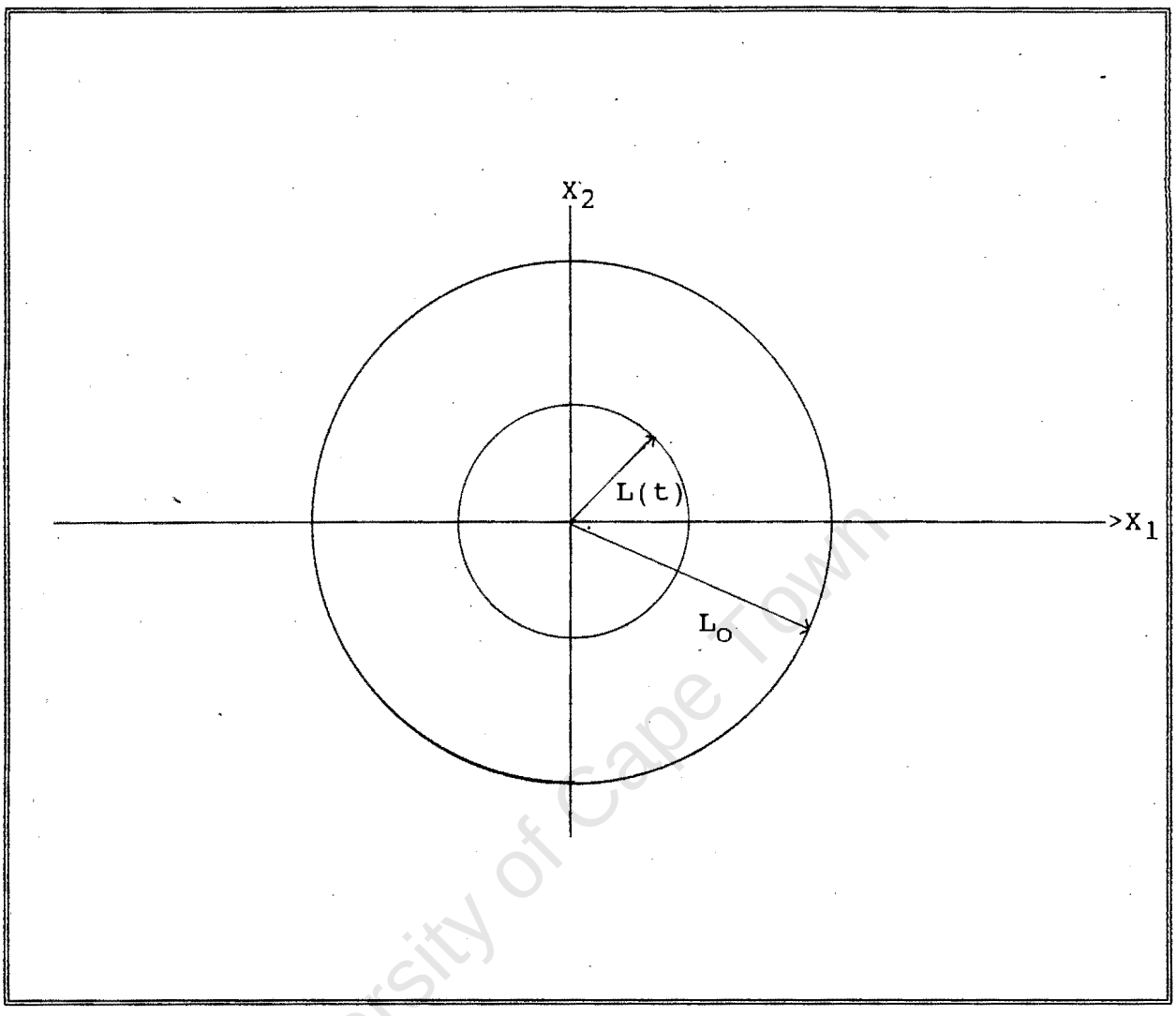


Figure 3.2: Region L_o defined in the phase plane.

This area will be referred to as the **Target Area.**

3.1.3 CONDITIONS THAT GUARANTEE CORRECT SWITCHING

Two areas in the phase plane have been isolated. Thus, at any instant in time, the RP must be in one, or some combination (either the intersection or union), of the following areas:

1. The Sliding Area.
2. The Target Area.
3. The rest of the phase plane (phase plane minus the areas of 1 and 2)

Conditions that guarantee correct switching of structures, depending where the RP is in the phase plane, will be found in this section.

In order to arrive at suitable conditions, Itkis [8] proves the following lemma.

LEMMA 1 :

If

$$c*v_1 + v_2 < m/(1 + c + m) \qquad \ldots(3.11)$$

and

$$L(t) > Lo \qquad \ldots(3.12)$$

Then

$$\text{sign}[\sigma(t)+\text{del}\sigma(t)] = \text{sign}[x_2(t)+(c+m)*x_1(t)] \qquad \ldots(3.13)$$

This means that if the RP is outside the union of the sliding and target area (area number 3), and equation 3.11 holds, then $L(t) > L_o$ and conditions satisfying Lemma 1 are met.

Consequently, when the RP is in area 3, the controller correctly determines the instantaneous structure of the system, even in the presence of noise. If the RP is in one of areas 1 or 2, however, correct determination of the structure cannot be guaranteed.

Since both areas 1 and 2 are dependent on the noise as well as the value of c, this result is in effect a practical limitation on the choice of c.

Regions in the phase plane have been isolated as areas of concern. This has practical significance when designing variable structure controllers.

3.1.4 INDEPENDENCE OF THE CONTROL SIGNAL TO THE SIGNS OF THE MEASURED STATES

In the previous section, it was shown that when the RP is near the origin, or about the ideal switching line, correct determination of structures cannot always be guaranteed. This section shows that the control function is in fact only dependent on the sign of the actual switching surface.

The measured states are the sum of the values of the actual states and a noise function term; as defined by equation 3.3. The instantaneous structure is dependent on both the sign of the measured states and the sign of the switching line.

$$\begin{aligned} \Gamma &= \alpha \quad \text{if } (x_1 + \tau_1)*\{\sigma(t) + \text{del}\sigma(t)\} > 0 \\ &= \beta \quad \text{if } (x_1 + \tau_1)*\{\sigma(t) + \text{del}\sigma(t)\} < 0 \end{aligned} \qquad \ldots(3.14)$$

Thus, it would seem natural that the control signal would also be dependent on both the signs of the measured states and the actual switching surface.

This, however, is not the case.

The control signal has the form :

$$u(t) = -\Gamma^*\{x_1(t) + \tau_1(t)\} \qquad \ldots(3.15)$$

An average gain term is defined as follows:

$$\Gamma_{ave} \equiv (\alpha + \beta)/2 \qquad \ldots(3.16)$$

If, in addition:

$$k \equiv \alpha - \Gamma_{ave} \qquad \ldots(3.17)$$

Then, by equations 3.14, 3.16 and 3.17:

$$\begin{aligned} \Gamma &= \Gamma_{ave} + k \quad \text{if } (x_1 + \tau_1)^*(\sigma + \text{del}\sigma) > 0 \\ &= \Gamma_{ave} - k \quad \text{if } (x_1 + \tau_1)^*(\sigma + \text{del}\sigma) < 0 \end{aligned} \qquad \ldots(3.18)$$

Now, it is generally true that:

$$|z| = z^*\text{sign}(z) \qquad \ldots(3.19)$$

So, combining this equation with equations 3.15 and 3.18:

$$u = -\Gamma_{ave}^*(x_1+\tau_1) - k^*|x_1+\tau_1|^*\text{sign}(\sigma+\text{del}\sigma) \qquad \ldots(3.20)$$

Equation 3.20 splits the control signal into two components:

1. A linear state feedback control term = $-\Gamma_{ave}*(x_1+\tau_1)$.

2. A variable amplitude relay signal term, which is equal to $-k*|x_1+\tau_1|*\text{sign}(\sigma+\text{del}\sigma)$, whose sign is dependent only on $(\sigma+\text{del}\sigma)$.

When symmetric control is employed, $\alpha = \beta$, and the Γ_{ave} term disappears. It is clear that the control function is independent of the signs of the measured states and depends only on the sign of the actual switching function, $(\sigma+\text{del}\sigma)$.

For the non - symmetric case when $\alpha <> \beta$, the first term in the control law may be viewed as just an internal feedback in the plant.

The system may now be written as:

$$dx_1/dt = x_2$$

$$dx_2/dt = -a_{n1}*x_1 - a_2*x_2 + k*|x_1+\tau_1|*\text{sign}(\sigma+\text{del}\sigma) \quad ...(3.21)$$

where

$$a_{n1} = a_1 + \Gamma_{ave} \quad ...(3.22)$$

This is the sum of the actual plant parameter, a_1, and the corrective action term, Γ_{ave}.

Thus, even for non - symmetric control, the controller is independent of the sign of the measured coordinates and depends only on the sign of the actual switching function.

For relatively small values of c, modest transient performance results. Despite delays in the system caused by, amongst other things, sampling, the controller switches structures correctly in the quasi-sliding regime . This is depicted in figure 3.3.

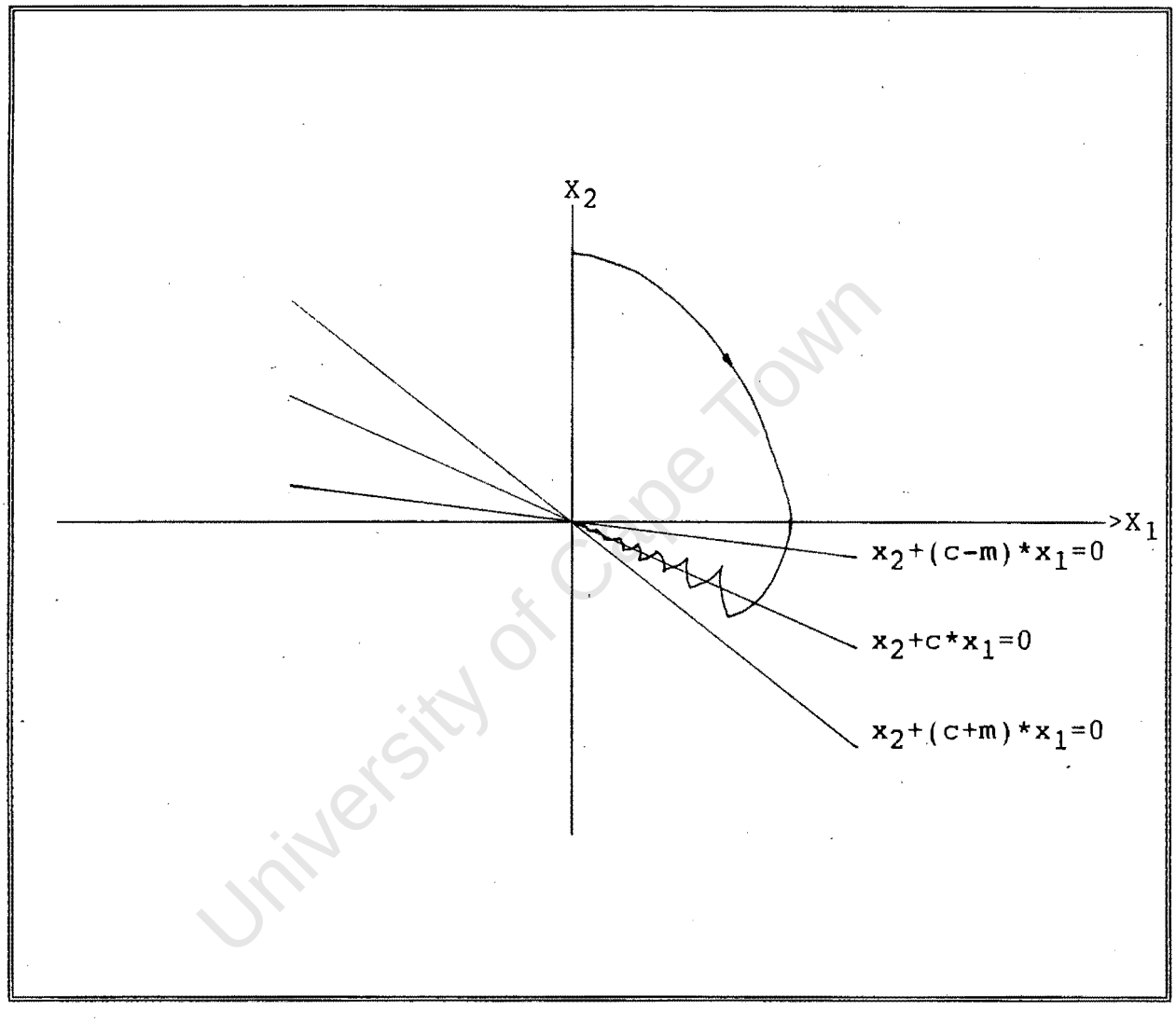


Figure 3.3: Quasi sliding.

For high speed applications, α, β and c are chosen to be large values. This means that the phase trajectory might, at some stage of sliding in the union of the Sliding and Target area, pierce the eigenvector $x_2(t) - \delta_1 * x_1(t) = 0$, defined in equation 1.8. Sliding is lost and the trajectory could get into a cycle about the origin. This is illustrated in figure 3.4.

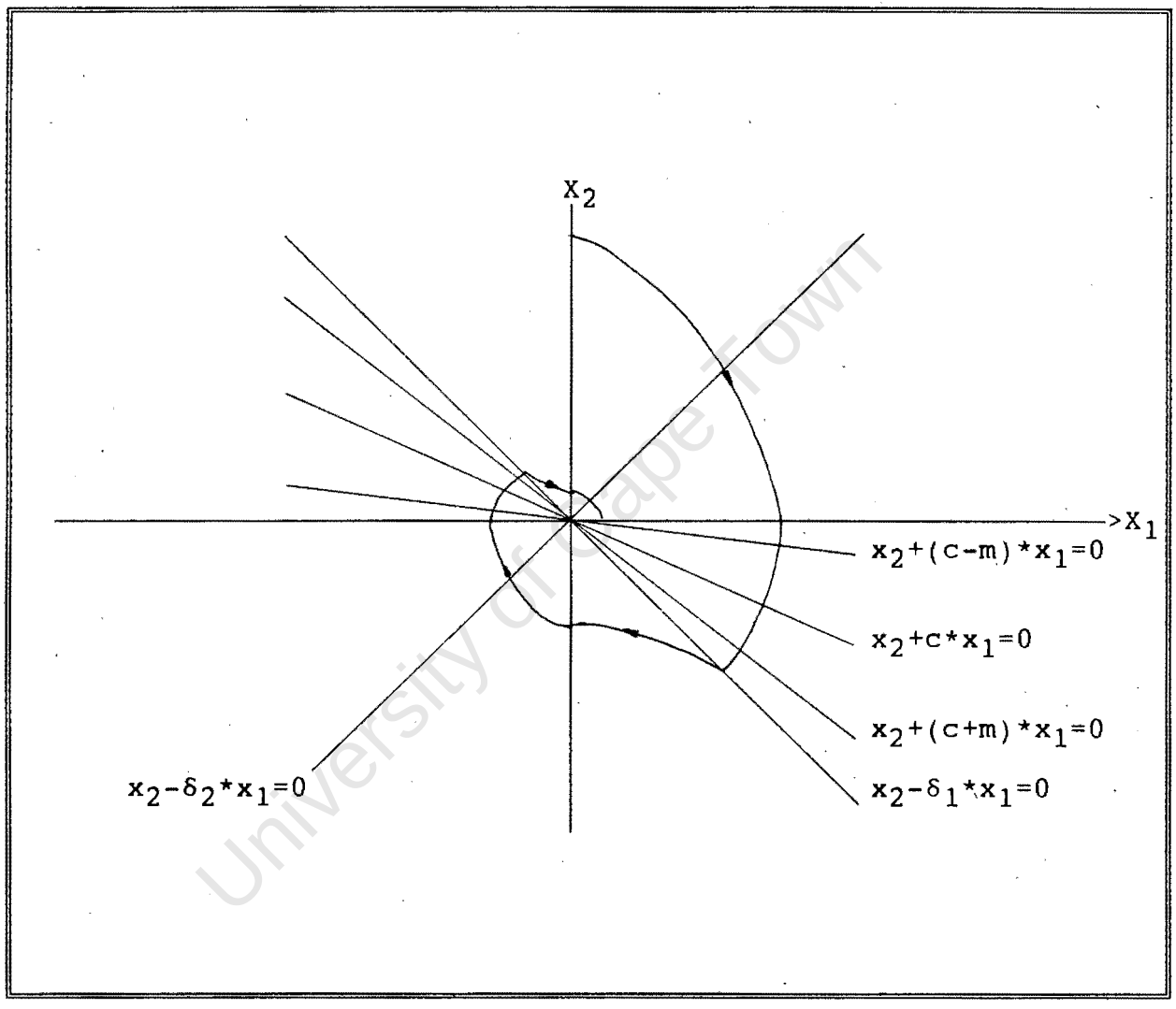


Figure 3.4:Loss of sliding.

Another practical problem which has not been looked at thus far, is tracking a rapidly changing setpoint. A sliding mode is difficult to define in this instance. This situation might arise when the setpoint varies about a pedestal value. The controller could get to the pedestal value of the setpoint, but could not guarantee following the rapidly changing perturbations.

One way to overcome these difficulties, is to add to the control design in the following way:

1. Add another structure to the system of the same form as when $\Gamma = \beta$ but let $\Gamma = \mu$ such that:

$$|\mu| > |\beta| \qquad \ldots(3.23)$$

A suitable value of μ will ensure that when the RP pierces the asymptote $x_2(t) - \delta_1 * x_1(t) = 0$, $\Gamma = \mu$ and the system will be switched without the loss of sliding. This also means that the trajectory will reach the switching line in a shorter time, but when it comes close to it, the more conservative (and hence usually the more stable) $\Gamma = \beta$ value is used.

This added structure will ensure sliding when in the quasi sliding regime, but as the trajectory approaches the Target region, it becomes increasingly difficult to prevent the system from getting into a cycle.

2. A way to overcome the cycling problem is to utilise a dual mode control scheme. When the RP is in the Target region, an ordinary state feedback control structure is implemented. This removes (at least limits) the unwanted cycling. Consequently, the output transient overshoot is limited.

The concept of a dual mode control system fits in to the philosophy of VSS. The state feedback controller is, in effect, just another structure.

It also improves the system's tracking capabilities.

The state feedback scheme is shown in figure 3.5.

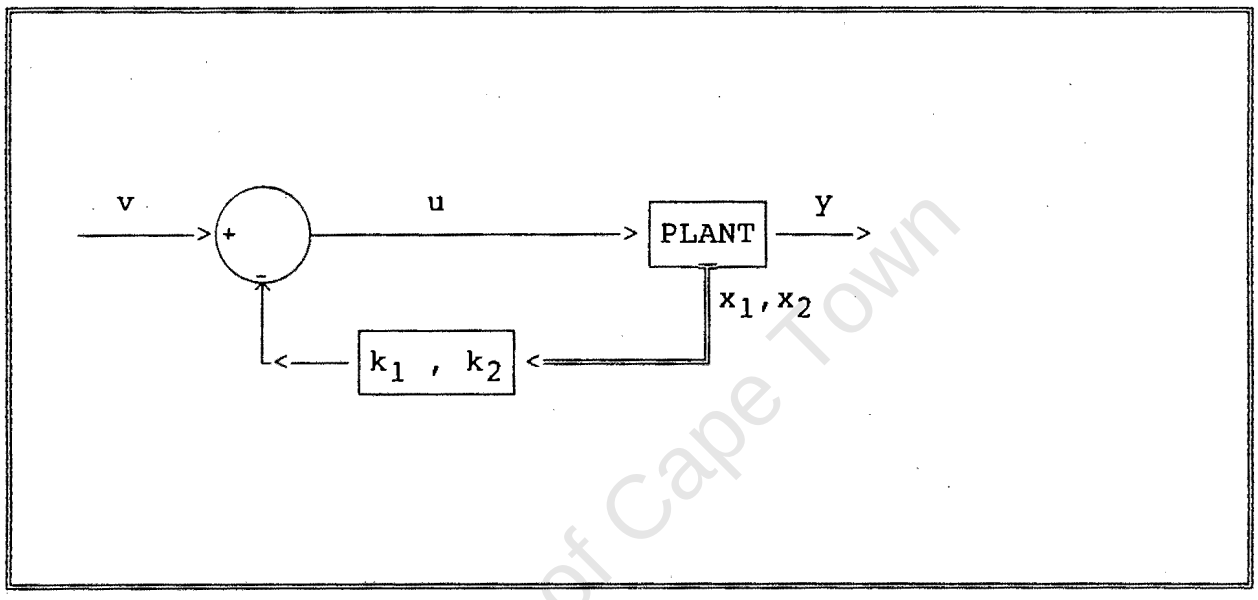


Figure 3.5: State feedback control.

The state feedback control scheme is summarised shortly:

The poles of the closed loop system depend on the constants k_1, k_2 and are given by:

$$|sI - A + bk^T| = 0 \qquad \ldots(3.24)$$

Suitable values for k_1 and k_2 can easily be chosen to give the system stable poles. They can be conservatively chosen to ensure stability (at the expense of speed) as the system only uses this control scheme when near the system setpoint.

This dual mode control scheme utilises the VSS - designed controller to reach an area close to the origin in the phase plane. During this part of the transient, changes in plant parameters have little effect on the control scheme. In addition, a high speed transient is obtained.

Once the Target area has been reached, the more conservative state feedback control scheme is employed. This allows for rapidly varying perturbations about a pedestal setpoint and removes the unwanted cycling problem.

The specified areas of control are shown in figure 3.6.

The constant c_1 in the figure, is just the chosen c value for structures defined when $\Gamma = \alpha, \beta$.

The constant c_2 in the figure, is chosen in the following way:

$$c_1 \leq c_2 \leq |\delta_1| \qquad \ldots(3.25)$$

where

The constant δ_1 in the figure, corresponds to the stable eigenvector of the structure defined by $\Gamma = \mu$.

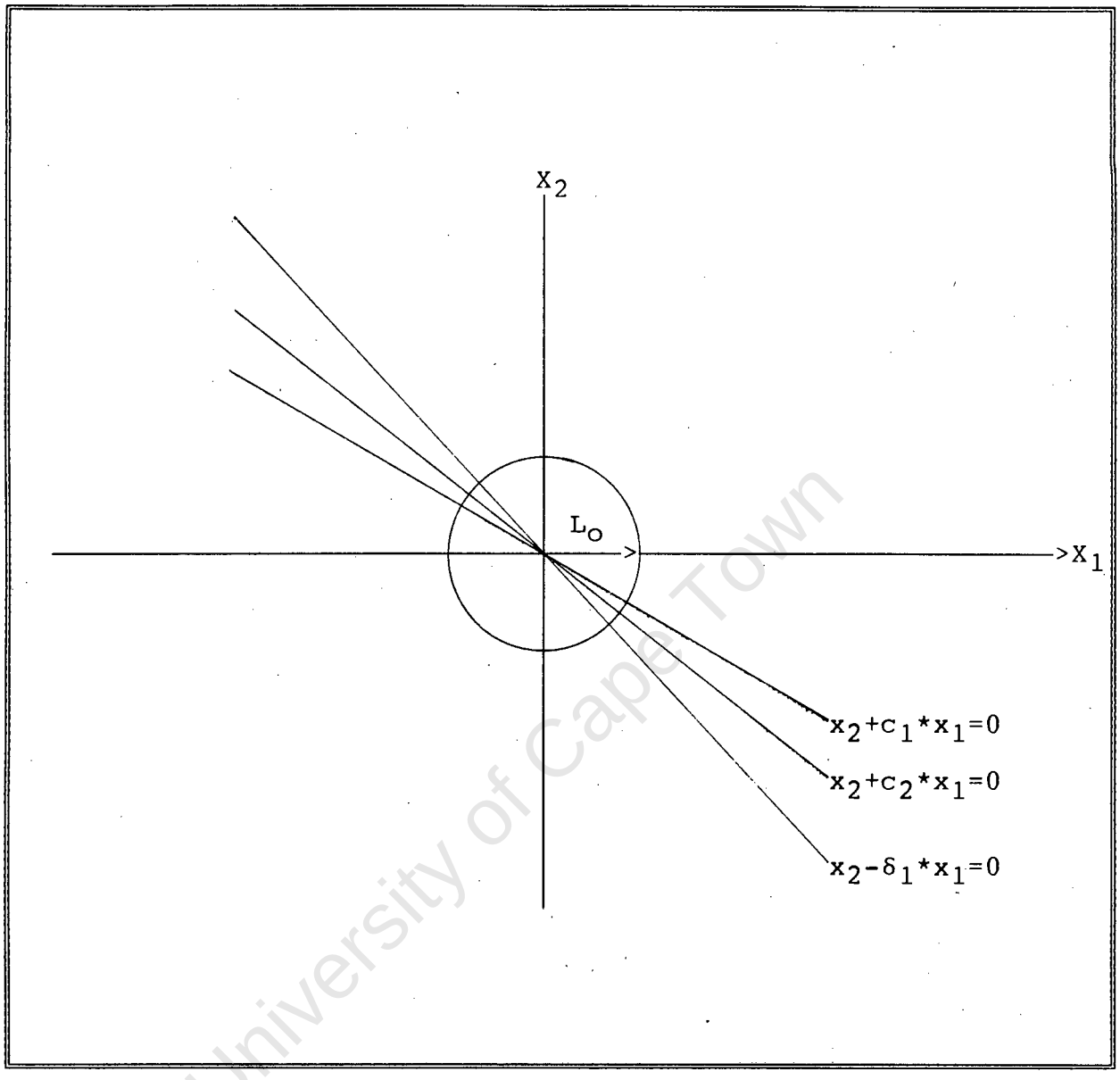


Figure 3.6: Modified control scheme.

3.3 IMPLEMENTATION OF MODIFIED CONTROLLER TO THE SERVO SYSTEM.

The servo system looked at in chapter 2 and defined by equations 2.9 - 2.11 is once again considered.

The acceptance test routine, defined and carried out in chapter 2, was repeated:

1. From 0 to 50 samples, the system is in open loop with no input being applied to the servo.

2. From 50 to 200 samples, the system is in closed loop control and the setpoint = 500 counts.

3. From 200 to 400 samples, the system is in closed loop control and the setpoint = 1000 counts.

4. From 400 to 600 samples, the system is in closed loop control and the setpoint ramps down by 3 counts every sample.

To satisfy a system time constant of 0.5 seconds, c_1 is chosen to be:

Let $c_1 = 1/0.5 = 2.000$ ...(3.26)

The equations governing the existence and hitting conditions for a sliding mode, equations 1.38,1.39 and 1.56, are satisfied for the following choice of gain values:

$$\Gamma = \alpha = 40 \text{ [counts]}$$
$$= \beta = -40 \text{ [counts]} \quad \text{...(3.27)}$$

When $\Gamma = \alpha$, the roots are complex and have negative real parts. The corresponding phase portrait is the same as that of figure 2.6.

When $\Gamma = \beta$, the roots are real and are of unlike sign. The corresponding phase portrait is the same as that of figure 2.5.

The roots of the characteristic equation are, by equation 2.18:

$$\delta_1 = -2.23$$
$$\delta_2 = 2.11 \qquad \ldots(3.28)$$

Let $c_2 = 2.23$ $\qquad \ldots(3.29)$

The equations governing the existence and hitting conditions for a sliding mode, equations 1.38,1.39 and 1.56, must now be satisfied for c_2 and $\Gamma = \alpha$ and μ; with μ replacing β.

Thus:

$$\mu = -45 \text{ [counts]} \qquad \ldots(3.30)$$

When $\Gamma = \mu$, the roots are real and of unlike sign. The corresponding phase portrait is the same as that of figure 2.5.

L_o is chosen in practice by experimental trial and error to obtain a suitable Target region.

Choose $L_o = 100$ [counts] $\qquad \ldots(3.31)$

k_1 , k_2 were suitably chosen to place closed loop poles in the desired location when state feedback control is used.

Figure 3.7 shows the position control of the unmodified system (ie without the structure defined by $\Gamma = \mu$).

Figure 3.8 illustrates the RP cycling about the origin.

The setpoints are reached in about 2 seconds, as expected.

The position output overshoots the setpoint. This is as a result of the RP cycling about the origin in the phase plane.

Figure 3.9 shows position control of the modified system.

Figure 3.10 illustrates the associated phase plane diagram.

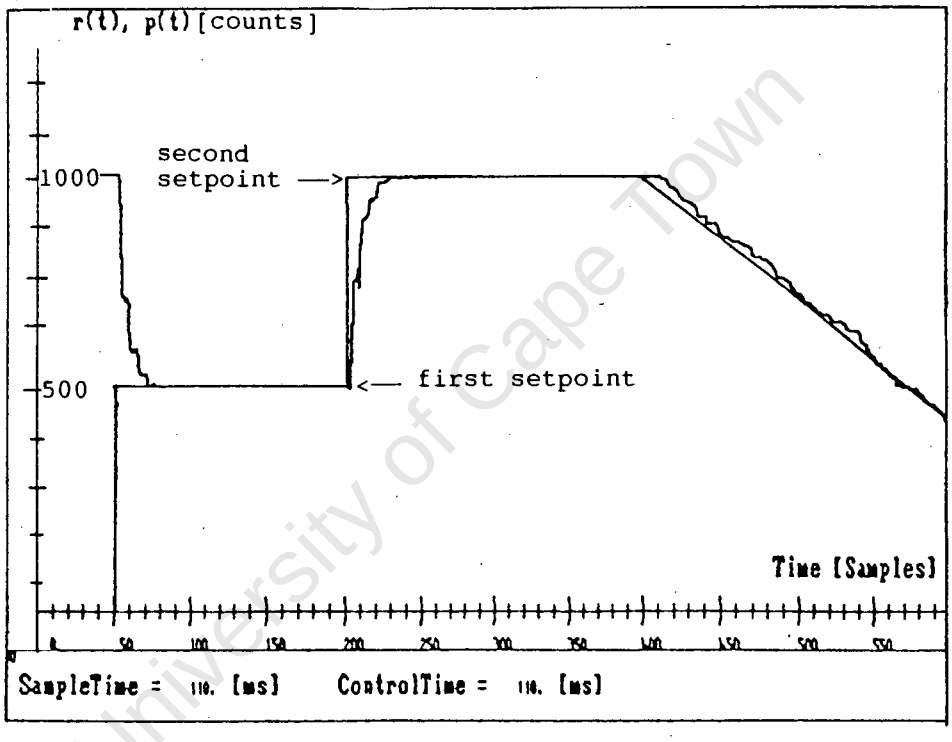


Figure 3.9 : Modified system

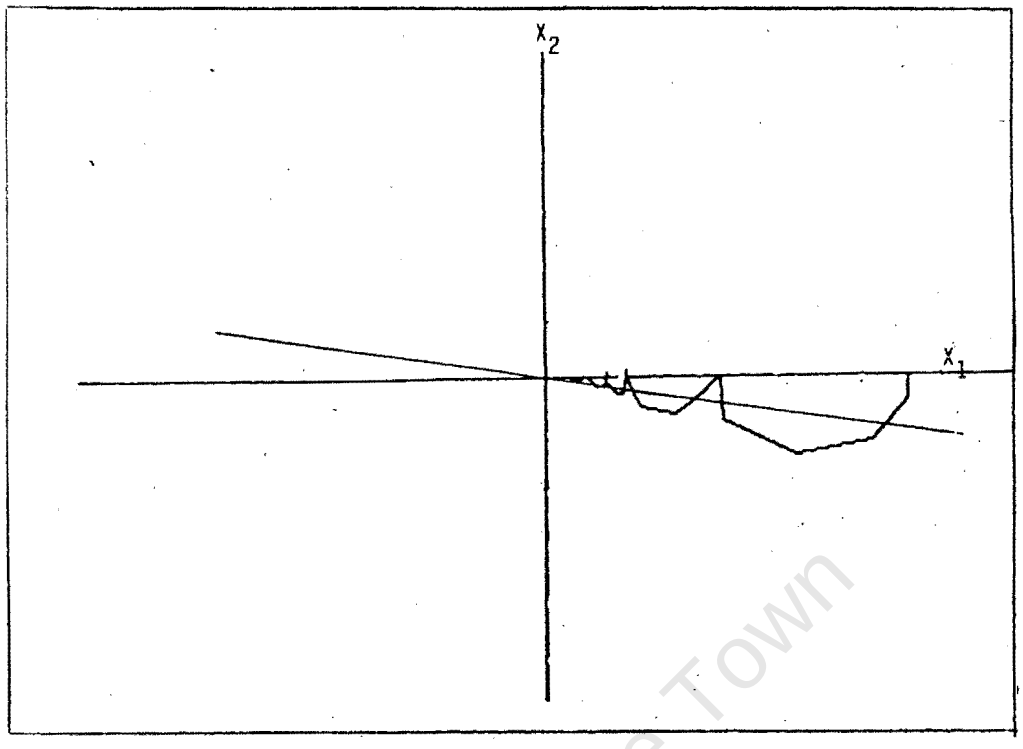


Figure 3.10 : Modified system phase portrait

The setpoint is reached after approximately 3.3 seconds (cf the 2 seconds of the unmodified strategy). There is no overshoot of the setpoint, however, and there is good tracking of the ramp setpoint.

Figure 3.10 shows that with this modified strategy, there is no loss of sliding.

CONCLUDING REMARKS ON THE CHAPTER

This chapter shows that VSS controllers can be designed for systems with non - ideal characteristics.

Non - ideal characteristics were lumped together and described by a noise function.

Areas in the phase plane were identified as places where non - ideal systems could lose sliding. These two areas are about the origin (the Target area) and the switching line (the Sliding area).

Instead of switching about a switching line, a quasi-sliding regime is defined for real systems to switch on.

Areas in the phase plane were identified where , despite the presence of noise , the system would switch structures correctly.

When the RP is in the union of the Target and Sliding areas, correct switching of structures cannot be guaranteed.

Surprisingly, it was established that the control function is independent of the sign of the measured states and depends entirely on the sign of the switching function.

The non - ideal behaviour of second order systems gives a good physical understanding of sliding in practical control systems.

The chapter is rounded off with a novel design proposal for a discrete controller to control a second order system using VSS theory. This digital controller design ensures that there is no loss of the sliding mode, when small transient times are required. The controller is also capable of tracking rapidly changing small perturbations about a pedestal setpoint.

The design directly addresses the two areas of concern in the phase plane.

Practical results are given to support the developed theory.

CHAPTER 4

MULTIVARIABLE SYSTEMS OF VARIABLE STRUCTURE

Up until this stage, only systems with a single input have been considered. These systems have an associated two or more states which have been fed back and used to shape the input function.

There are many systems however, which have more than one input and thus need to be analysed using multivariable techniques. They constitute an ever increasing important class of systems, as the complexity of technology grows.

4.1 CENTRALIZED CONTROLLER FOR A NUMBER OF SINGLE - INPUT SYSTEMS

In some applications, it is of financial benefit to control m single input systems using just one centralised control scheme. This is the most simple variation of the single - input systems which have been considered up to now, and thus it serves as a transitory step to understanding multi - input systems of variable structure.

If all the m systems are linear,the open loop macro system can be written in the form:

$$\mathbf{dx}(t)/dt = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \qquad \ldots(4.1)$$

where

$A(t)$ is an nxn matrix with elements $a_{ij}(t)$.
$B(t)$ is an nxm matrix with elements $b_{ij}(t)$.
$\mathbf{x}(t) = [x_1, x_2, \ldots, x_n]^T$ is an n vector of the macro system states.
$\mathbf{u}(t) = [u_1, u_2, \ldots, u_m]^T$ is an m vector containing the inputs of the individual systems.

The input vector is such that only one of its elements is non - zero at any particular time.

The central control device is connected to the system which is in the most "perilous" situation. This is determined by some comparison index, f_j $(j = 1, \ldots, m)$.

The control law then has the form:

$$u_j(t_v) = -1 \quad \text{if } f_j(t_v) > \max_{j \neq i = 1, \ldots, m} \{f_i(t_v)\}$$

$$= 0 \quad \text{if } f_j(t_v) < \max_{i = 1, \ldots, m} \{f_i(t_v)\} \qquad \ldots(4.2)$$

$$\sum_{j=1}^{m} u_j(t_v) = -1 \qquad \ldots(4.3)$$

where

t_v $(v = 0, 1, \ldots)$ is the sampling time.

One particular comparison index, is the distance of a chosen state of a system from the origin. If the distance from the first state of the jth system, x_{j1}, to the origin is greater than the distance from the corresponding state of the other systems to the origin, then the jth element of the controller is connected to the jth system.

This can be written mathematically in the following way:

$$u_j(t) = -1 \quad \text{if } |x_{j1}(t_v)| > \max_{j<>k=1,\ldots,m} |x_{k1}(t_v)|$$

$$= 0 \quad \text{if } |x_{j1}(t_v)| < \max_{k=1,\ldots,m} |x_{k1}(t_v)| \qquad \ldots(4.4)$$

where

$(j = 1,\ldots m)$ and $t \in [t_v, t_{v+1}]$.

This controller ensures that the control signal is only applied to the unstable systems. If the sampling time is small enough, a quasi-sliding regime exists for each system and hence stability is assured.

Another comparison index is the absolute distance of the phase plane representative point (RP) from the switching hyperplane.

This index is written in the following form:

$$f_j = |\sigma_j| \; ; \; (j = 1,\ldots m) \qquad \ldots(4.5)$$

where

f_j is the jth comparison index.

σ_j is the jth switching function, corresponding to the jth single input system.

It can be shown that using such comparison indices as the basis of the control law of equations 4.2 and 4.3, it is relatively easy to set up a controller that ensures the existence of a quasi sliding regime for each linear independent sub - system making up the macro system.

4.2 MULTI INPUT MULTI OUTPUT SYSTEMS

Attention is now focused on systems which have more than one input and output. The theory which was developed for attaining a sliding regime for scalar controlled systems is generalised for the more complex multi input, multi output (MIMO) systems.

Since VSS theory was developed from a mathematical environment, there exist a number of papers describing theoretical concepts relating to VSS. In particular, Drazenovic provides generalised invariance conditions to noise and plant parameter changes for multivariable VSS in [4].

Whilst this has undoubted theoretical value, it provides no practical design method or design criteria.

This chapter only looks at specific design techniques which make VSS theory an attractive practical option when choosing a control method for multivariable systems.

4.2.1 CREATING SLIDING REGIMES FOR MIMO SYSTEMS

Consider the following generalised MIMO system without disturbances:

$$\mathbf{dx/dt = Ax + Bu} \qquad \ldots(4.6)$$

where

A is an nxn matrix.
B is an nxm matrix.
$\mathbf{x} = [x_1, x_2, \ldots x_n]^T$ is a vector containing the system states.
$\mathbf{u} = [u_1, u_2, \ldots u_m]^T$ is a vector containing the system inputs.

Just as the sliding mode of the single input system considered in chapter 1 was defined by a switching line, the sliding mode of a multivariable system is defined by a set of switching hyperplanes (multi - dimensional surfaces in the phase plane which are some linear combination of the system states).

A system with m inputs has m switching hyperplanes. The input control vector is chosen in such a way that sliding regimes exist simultaneously (ideally) on all m hyperplanes.

When the RP reaches a particular hypersurface, it cannot leave it, but moves on average along the intersection of that hyperplane and any other hyperplane that it has reached.

As an example, the control function u_1 might ensure sliding on the hyperplane σ_1. The control function, u_2, will then ensure sliding on $\sigma_1 \cap \sigma_2$, whilst u_i will ensure sliding on $\sigma_1 \cap \sigma_2 \ldots \cap \sigma_i$. Finally, u_m will ensure sliding on $\sigma_1 \cap \sigma_2 \ldots \cap \sigma_m$.

An illustrative example of sliding for a system with two inputs, and hence two hyperplanes, can be seen in figure 4.1.

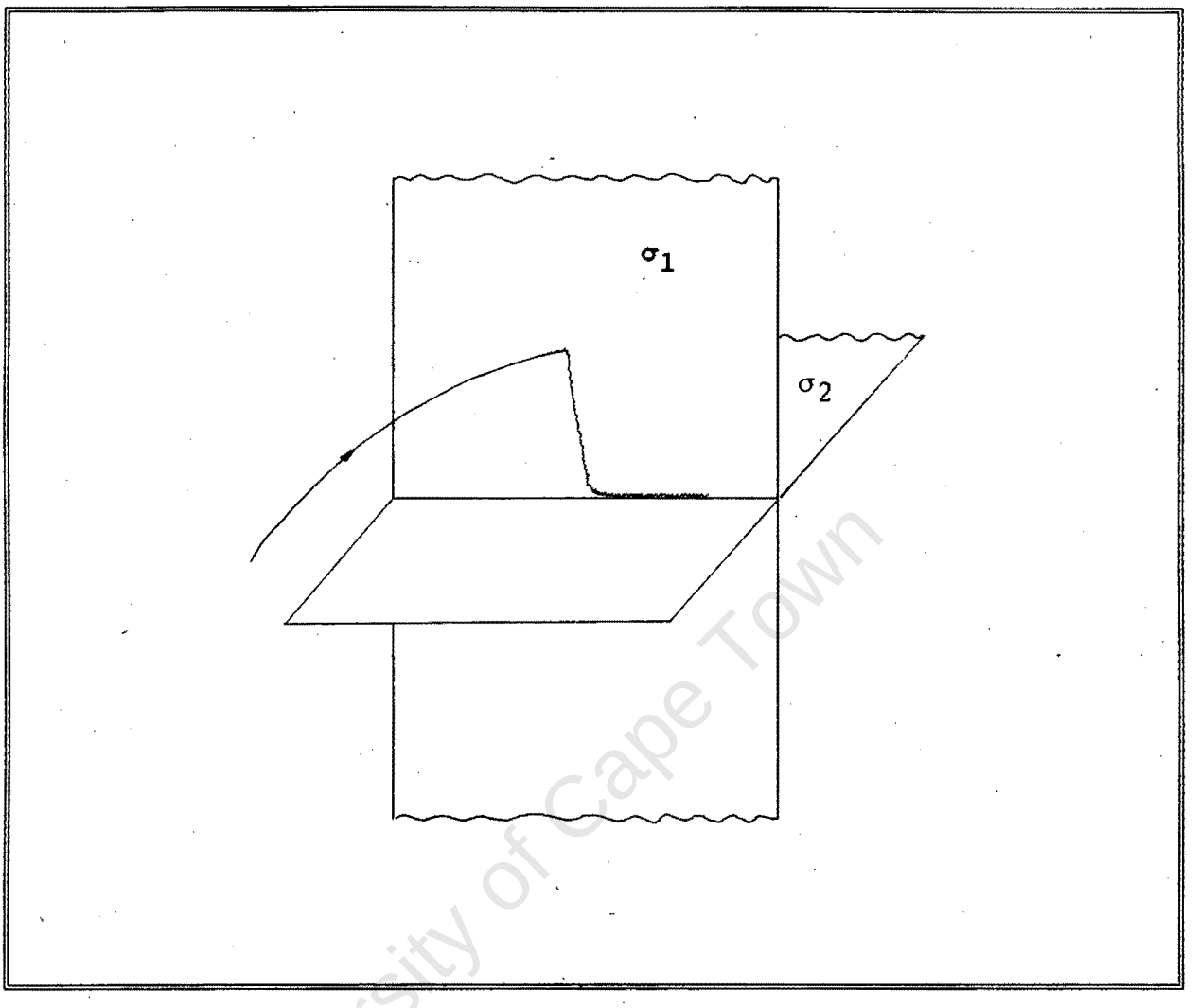


Figure 4.1: Example of sliding motion on two hyperplanes.

The hyperplanes have the following general form:

$$\sigma = Cx \qquad \ldots(4.7)$$

where each individual hyperplane is given by:

$$\sigma_i = \sum_{j=1}^{n} c_{ij} * x_j \qquad \ldots(4.8)$$

C is an mxn matrix with elements c_{ij}. Without loss of generality, one of the c_{ij} for each specific hyperplane, σ_i, may be set to unity. This is often c_{in}.

$$c_{in} = 1.0 \qquad \ldots(4.9)$$

$\boldsymbol{\sigma} = [\sigma_1,\ldots,\sigma_m]^T$ is an m vector.

For ideal variable structure systems, equation 4.7 is equated to zero by definition:

$$\boldsymbol{\sigma} = C\mathbf{x} = \mathbf{0} \qquad \ldots(4.10)$$

In reality,however, each individual hyperplane does not equate to zero simultaneously with all other hyperplanes. They equal to small quantities which change sign at high frequencies.

In addition, when the system slides, it is required that:

$$d\boldsymbol{\sigma}/dt = \mathbf{0} \qquad \ldots(4.11)$$

Since all hyperplanes do not vanish simultaneously,however, equation 4.11 needs to be modified.

The aim of the following analysis is to illustrate how matrix C effects the motion of the system in the sliding mode, and hence gives design criteria for specifying the hyperplanes.

Differentiating equation 4.7, the following expression is obtained:

$$d\boldsymbol{\sigma}/dt = C*d\mathbf{x}/dt \qquad \ldots(4.12)$$

Inserting system equation 4.6 into equation 4.12:

$$d\sigma/dt = CA\mathbf{x} + CB\mathbf{u} \qquad \ldots(4.13)$$

In order to study the sliding mode, Utkin formulated what is known as the equivalent control technique. This equivalent or mean control vector can be defined by equating equation 4.13 to zero according to equation 4.11. Rearranging the resulting equation, an expression for this mean control vector is obtained:

$$\mathbf{u}_{eq} = -(CB)^{-1}CA\mathbf{x} \qquad \ldots(4.14)$$

This equivalent vector input is effectively the average value of the actual input vector which maintains the RP on the hypersurface defined by equation 4.10.

The actual control signals are made up of a low frequency component and a high frequency component ([15]). The equivalent control corresponds to the low frequency or average component of the actual input vector, without the high frequency perturbations.

As long as matrix (CB) is not singular (it must have rank m), an expression for the average motion in the sliding regime can be obtained by inserting equation 4.14 into equation 4.6:

$$d\mathbf{x}/dt = [I - B(CB)^{-1}C]A\mathbf{x} \qquad \ldots(4.15)$$

where

I is an nxn identity matrix.

B is an nxm matrix.

$(CB)^{-1}$ is an mxm matrix.

C is an mxn matrix.

A is an nxn matrix.

Motion is now described by a set of (n-m) first order differential equations, bringing the system order down by m. The reason for the system reduction in order is that now the m vector ,σ , is equated to the zero vector; thus eliminating m state variables. The state vector moves on an (n-m) dimensional manifold of the intersection of the m hyperplanes.

One of the important ramifications of the reduction in system order, is that now only (n-m) closed loop eigenvalues need be specified. The remaining m eigenvalues should be zero.

Ghezawi , Zinober and Billings (in [5]) use projector theory to prove this reduction of system order.

The important result that has been obtained is that equation 4.15, which describes the motion of the system in the sliding mode, has the following form:

$$dx/dt = A_{eq}x \qquad \ldots(4.16)$$

where

$A_{eq} = [I - B(CB)^{-1}C]A$ and is an nxn matrix with rank (n-m).

In this form, the system eigenvalues, which define the RP in the phase plane, can be arbitarily chosen by specifying suitable values for matrix C. These eigenvalues are in effect the eigenvalues of the low frequency sub - system.

Thus, by adopting the method of equivalent control, suitable hyperplanes to meet the required system performance are easily designed.

This is done by solving equation 4.15 and choosing suitable values for matrix C to satisfy the speed requirements of the system transients.

The eigenvalues can now be chosen to give a slow but robust design, or to ensure high speed transients at the cost of some system insensitivity.

Specific methods to synthesize matrix C for a specified choice of closed loop eigenvalues are given in [5]. These methods are rather cumbersome to use in practice, and it was found that adequate results were obtained by solving equation 4.15 on a trial and error basis using some form of C.A.D. or matrix manipulation package. A package called Matlab was used to design all C matrices in this and the following chapter.

4.2.2 CONTROL FUNCTIONS FOR VARIABLE STRUCTURE MIMO SYSTEMS

By choosing matrix C, the hyperplanes on which sliding will occur are specified. It remains to design the specific control functions which will ensure that the sliding mode will be reached and that conditions ensuring its existence will be satisfied.

This section aims to provide these design criteria for the input vector to achieve an overall robust design.

It was shown in chapter 1, that the general condition for any one sliding hyperplane to exist is given by:

$$\lim_{\sigma \to 0} \sigma * d\sigma/dt < 0 \qquad \ldots(4.17)$$

Since a system with m inputs has m hyperplanes, the necessary and sufficient conditions for individual surface reachability by each corresponding control input is:

$$\lim_{\sigma i \to 0} \sigma_i * d\sigma_i/dt < 0 \quad (i = 1,\ldots,m) \qquad \ldots(4.18)$$

The equivalent control vector defined in equation 4.14 is not used in practice, since its components are dependent on system parameters and external disturbances; making it inherently sensitive to noise.

The chosen input vector should satisfy the following objectives:

1. It should drive the system towards the sliding surfaces.

2. It should ensure that the system is invariant to parameter changes and external disturbances once in the sliding mode. In practice, the RP will not slide at infinite frequency and with infinitesmal amplitude. This means that the respective components of the input vector should overshoot the sliding surface and then correct its action. This approximate, or quasi-sliding, is what will be deemed satisfactory for practical systems.

Extending the theory from single input systems, the natural form of the input functions is:

$$u_i = -\sum_{j=1}^{n} \Gamma_{ij} * x_j \quad (i = 1,\ldots m) \qquad \ldots(4.19)$$

where

$$\Gamma_{ij} = \alpha_{ij} \text{ if } \sigma_i * x_j > 0$$
$$= \beta_{ij} \text{ if } \sigma_i * x_j < 0 \qquad \ldots(4.20)$$

Each input is then designed to make the RP reach the corresponding sliding surface. The individual gains α_{ij}, β_{ij}, are chosen to satisfy equation 4.17. If the individual gains of the controller inputs are big enough, equation 4.17 ensures that each individual input will drive the system towards its corresponding hyperplane. This is only true,however, when the other system inputs are ignored. Thus, designing the controller components individually, may result in a conflict arising between inputs.

It is thus clear, that specifying sliding on all m hyperplanes simultaneously, is not a good practical solution.

One way to overcome the reachability problem, is to assign arbitarily an input hierarchy. This method, which was formulated by Utkin (1974), directly takes into account parameter uncertainties. The algorithm can be found in Appendix 1.

The control algorithm is more easily understood by considering the following design example given by Ramirez and Dwyer [11].

Consider the following linear system:

$$d\mathbf{x}/dt = A\mathbf{x} + B\mathbf{u} \qquad \ldots(4.21)$$

where

$$A = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ -1.09486E\text{-}6 & 0.0 & 0.0 & -0.01499 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.01499 & -1.09486E\text{-}6 & 0.0 \end{bmatrix} \quad \ldots(4.22)$$

$$B = \begin{bmatrix} 0.0 & 0.0 \\ 7.4E\text{-}4 & 0.0 \\ 0.0 & 0.0 \\ -1.3E\text{-}4 & 7.5E\text{-}4 \end{bmatrix} \quad \ldots(4.23)$$

Here , n = 4 and m = 2.

In general, the two hyperplanes are defined to be:

$$\sigma_1 = c_{11}x_1 + c_{12}x_2 + c_{13}x_3 + c_{14}x_4 \quad \ldots(4.24)$$

$$\sigma_2 = c_{21}x_1 + c_{22}x_2 + c_{23}x_3 + c_{24}x_4 \quad \ldots(4.25)$$

Differentiating the above two equations and inserting the relevant values from equations 4.22 and 4.23, the following expressions are obtained:

$$d\sigma_1/dt = (-1.09486E\text{-}6*c_{12})*x_1 + (c_{11}+0.01499*c_{14})*x_2 + (-1.09486E\text{-}6*c_{14})*x_3 + (-0.01499*c_{12}+c_{13})*x_4 + (0.00074*c_{12}-0.00013c_{14})*u_1 + (0.00075*c_{14})*u_2 \quad \ldots(4.26)$$

$$d\sigma_2/dt = (-1.09486E\text{-}6*c_{22})*x_1 + (c_{21}+0.01499*c_{24})*x_2 + (-1.09486E\text{-}6*c_{24})*x_3 + (-0.01499*c_{22}+c_{13})*x_4 + (0.00074*c_{22}-0.00013c_{24})*u_1 + (0.00075*c_{24})*u_2 \quad \ldots(4.27)$$

The elements of the C matrix, c_{ij}, are chosen to satisfy the following requirements:

1. One of equations 4.26, 4.27 should be dependent on only one input directly.

2. Equations 4.26 and 4.27 should be written in terms of the chosen sliding hyperplanes. This is a direct result of the findings from chapter 3 that the control function is dependent on the sign of the switching function. With their inclusion in the above equations, the sign of the switching functions is also taken into account.

3. The choice should satisfy the desired transient performance.

Choosing $c_{14} = 0.0$ ensures that equation 4.26 becomes dependent on one input only.

To ensure that expressions for the particular switching functions are "massaged" into equations 4.26 and 4.27 with as little dependence on the system parameters as possible, terms involving states multiplied by elements of matrix C (and not by elements of either the A or B matrices) are isolated.

In equation 4.26, these are $c_{11}*x_2$ and $c_{13}*x_4$.

In equation 4.27, they are $c_{21}*x_2$ and $c_{13}*x_4$.

Since c_{14} has been chosen to equal zero to remove the term involving u_2 from equation 4.26, hyperplane 1 does not have a term involving x_4 and thus hyperplane 2 will be chosen to be massaged into the equations via x_4.

Specifically, if for conveniance $c_{24} = 1.0$ and $c_{22} = 0.0$:

$$\sigma_2 = x_4 - (c_{21}*x_1 + c_{23}*x_3) \qquad \ldots(4.28)$$

Since σ_1 will be massaged into equations 4.26 and 4.27 in the same manner, $c_{12} = 1.0$ for conveniance.

$$\sigma_1 = x_2 - (c_{11}*x_1 + c_{13}*x_3) \qquad \ldots(4.29)$$

The remaining values in the C matrix are chosen by solving equation 4.15 to obtain the desired two (n-m) closed loop eigenvalues.

The chosen C matrix is:

$$C = \begin{bmatrix} 0.12 & 1.00 & 0.03 & 0.00 \\ 0.03 & 0.00 & 0.12 & 1.00 \end{bmatrix} \qquad \ldots(4.30)$$

The following expressions are obtained:

$$\sigma_1*d\sigma_1/dt = (c_{11})*\sigma_1^{\wedge 2} + (c_{13}-0.01499)\sigma_1*\sigma_2 + (c_{21}*0.01499 - 1.09486E\text{-}6 - c_{11}^{\wedge 2} - c_{13}*c_{21})*x_1*\sigma_1 + (0.01499*c_{23} - c_{11}*c_{13} - c_{23}*c_{13})*x_3*\sigma_1 + (0.00074)*u_1*\sigma_1 \qquad \ldots(4.31)$$

$$\sigma_2*d\sigma_2/dt = (c_{23})\sigma_2^{\wedge 2} + (c_{21}+0.01499)\sigma_1*\sigma_2 + (-c_{11}*0.01499 - c_{21}*c_{11} - c_{23}*c_{21})*x_1*\sigma_2 + (-c_{13}*0.01499 - 1.09486E\text{-}6 - c_{21}*c_{13} - c_{23}^{\wedge 2})*x_3*\sigma_2 + (-0.00013)u_1*\sigma_2 + 0.00075*u_2*\sigma_2 \qquad \ldots(4.32)$$

In this form, with $\sigma_1*d\sigma_1/dt$ dependent on only one input, u_1, the supposed precedence is $u_2 \rightarrow u_1$.

This means that equation 4.17 is solved using equation 4.31 and assuming ideal sliding conditions for σ_2 (assume that both σ_2 and $d\sigma_2/dt$ are equal to zero).

After solving equation 4.17 using equation 4.31 and with $\sigma_2 = 0$, u_1 has the form:

$$u_1 = -\Gamma_{11}*x_1 - \Gamma_{13}*x_3 \qquad \ldots(4.33)$$

where

$$\Gamma_{11} = \alpha_{11} > [0.01499*c_{21} - 1.09486E\text{-}6 - c_{11}^{\wedge 2} - c_{13}*c_{21}] \quad \text{if } \sigma_1*x_1 > 0$$

$$= \beta_{11} < [0.01499*c_{21} - 1.09486E\text{-}6 - c_{11}^{\wedge 2} - c_{13}*c_{21}] \quad \text{if } \sigma_1*x_1 < 0$$

$$\Gamma_{13} = \alpha_{13} > [0.01499*c_{23} - c_{11}*c_{13} - c_{23}*c_{13}] \quad \text{if } \sigma_1*x_3 > 0$$

$$= \beta_{13} < [0.01499*c_{23} - c_{11}*c_{13} - c_{23}*c_{13}] \quad \text{if } \sigma_1*x_3 < 0 \qquad \ldots(4.34)$$

The robustness of the controller is due to u_2 being specified in terms of σ_1 and u_1. Solving equation 4.17 using equation 4.37 to obtain the respective gains, it is clear that u_2 has the following form:

$$u_2 = -\Gamma_{21}*x_1 - \Gamma_{23}*x_3 - \Gamma_2*\sigma_1 \qquad \ldots(4.35)$$

where

$$\Gamma_{21} = \alpha_{21} > 1327.28[-0.01499*c_{11} - c_{21}*c_{11} - c_{23}*c_{21} - \alpha_{11}*0.00013] \quad \text{if } \sigma_2*x_1 > 0$$
$$= \beta_{21} < 1327.28[-0.01499*c_{11} - c_{21}*c_{11} - c_{23}*c_{21} - \beta_{11}*0.00013] \quad \text{if } \sigma_2*x_1 < 0$$

$$\Gamma_{23} = \alpha_{23} > 1327.28[-0.01499*c_{13} - 1.09486E\text{-}6 - c_{21}*c_{13} - c_{23}^{\wedge 2} - \alpha_{13}*0.00013] \quad \text{if } \sigma_2*x_3 > 0$$
$$= \beta_{23} < 1327.28[-0.01499*c_{13} - 1.09486E\text{-}6 - c_{21}*c_{13} - c_{23}^{\wedge 2} - \beta_{13}*0.00013] \quad \text{if } \sigma_2*x_3 < 0$$

$$\Gamma_2 = \alpha_2 > 1327.28[c_{21} + 0.01499] \quad \text{if } \sigma_2*\sigma_1 > 0$$
$$= \beta_2 < 1327.28[c_{21} + 0.01499] \quad \text{if } \sigma_2*\sigma_1 < 0$$

...(4.36)

The above system was then simulated in a number of experiments.

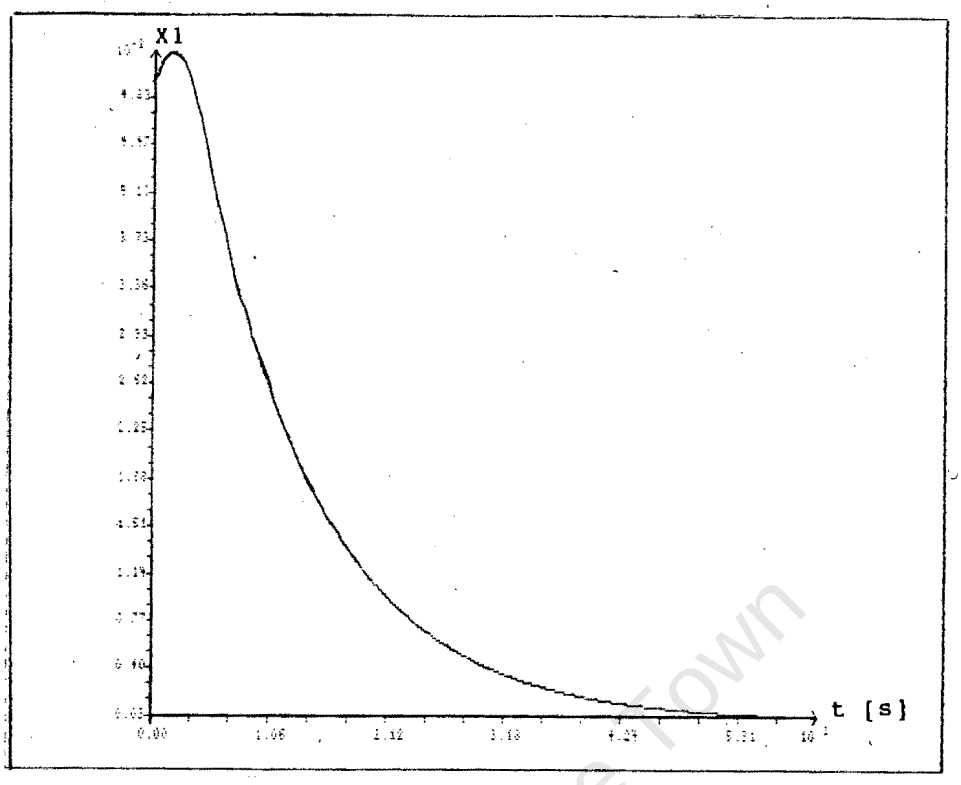
Experiment :	1	2	3
α_{11} [counts] :	27.0	27.0	-1.0
β_{11} [counts] :	-27.0	-27.0	-25.0
α_{13} [counts] :	14.0	14.0	-0.5
β_{13} [counts] :	-14.0	-14.0	-10.0
α_{21} [counts] :	21.0	21.0	-0.75
β_{21} [counts] :	-21.0	-21.0	-17.0
α_{23} [counts] :	28.0	28.0	-1.0
β_{23} [counts] :	-28.0	-28.0	-25.0
α_2 [counts] :	66.0	66.0	65.0
β_2 [counts] :	-66.0	-66.0	55.0
time step :	0.125[s]	1.0[s]	1.0[s]

This experiment was carried out with a very fast sampling frequency relative to the closed loop eigenvalues. A 0.125 second sampling period corresponds to an eigenvalue sitting at:

$$1/0.125 = -8.0 \text{ [rad/sec]} \qquad \ldots(4.39)$$

in the s plane.

Figure 4.2 and 4.3 illustrates the simulation of the closed loop dynamics of the four system states. Figure 4.4 illustrates the dynamics of the two control functions.



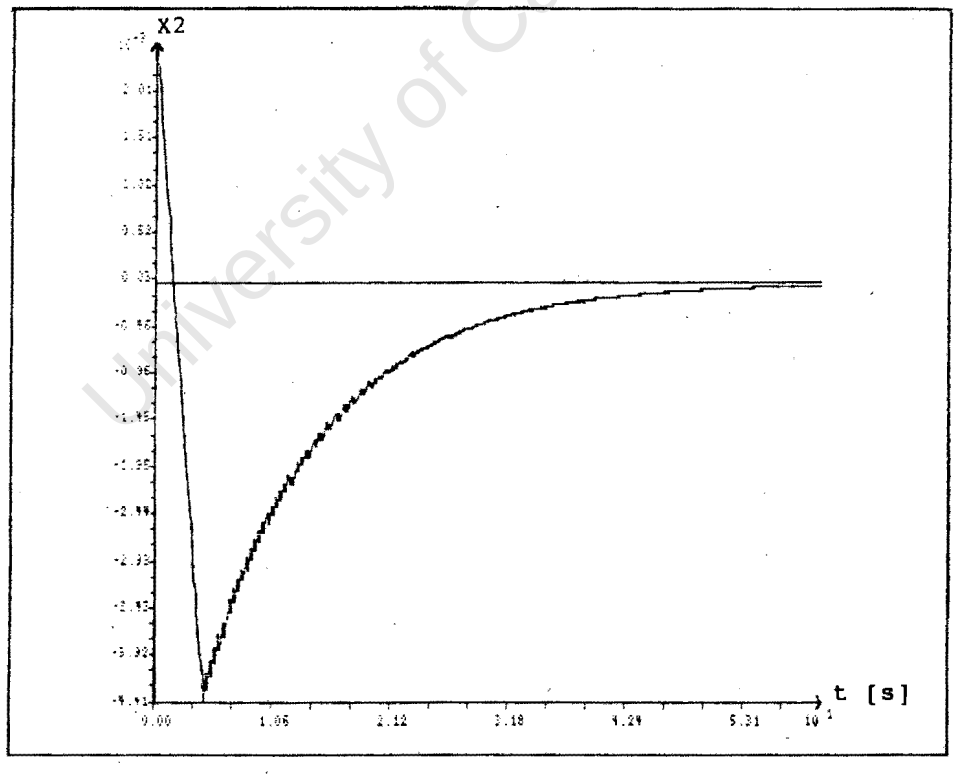


Figure 4.2 : The closed loop dynamics of states one and two .

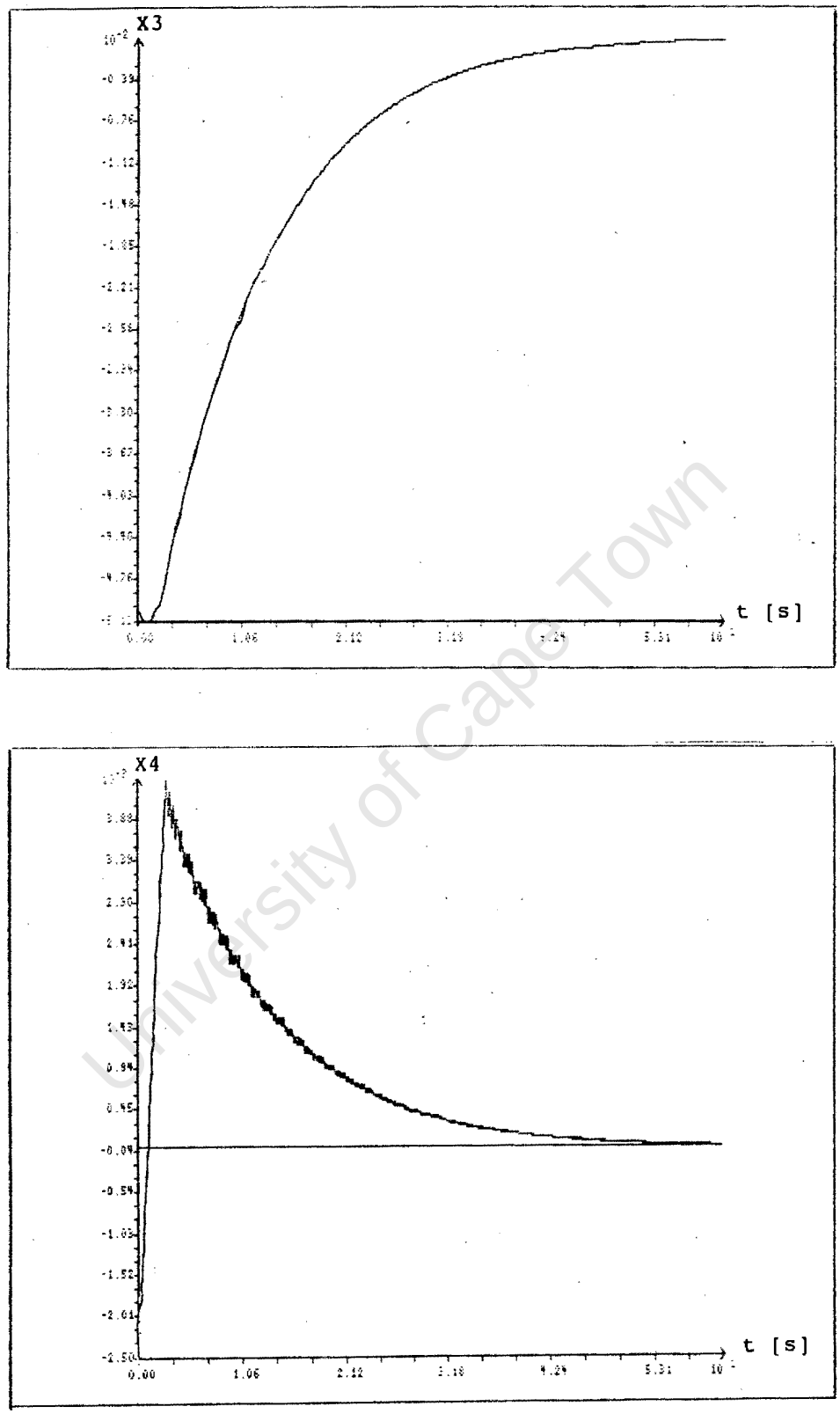


Figure 4.3 : The closed loop dynamics of states three and four .

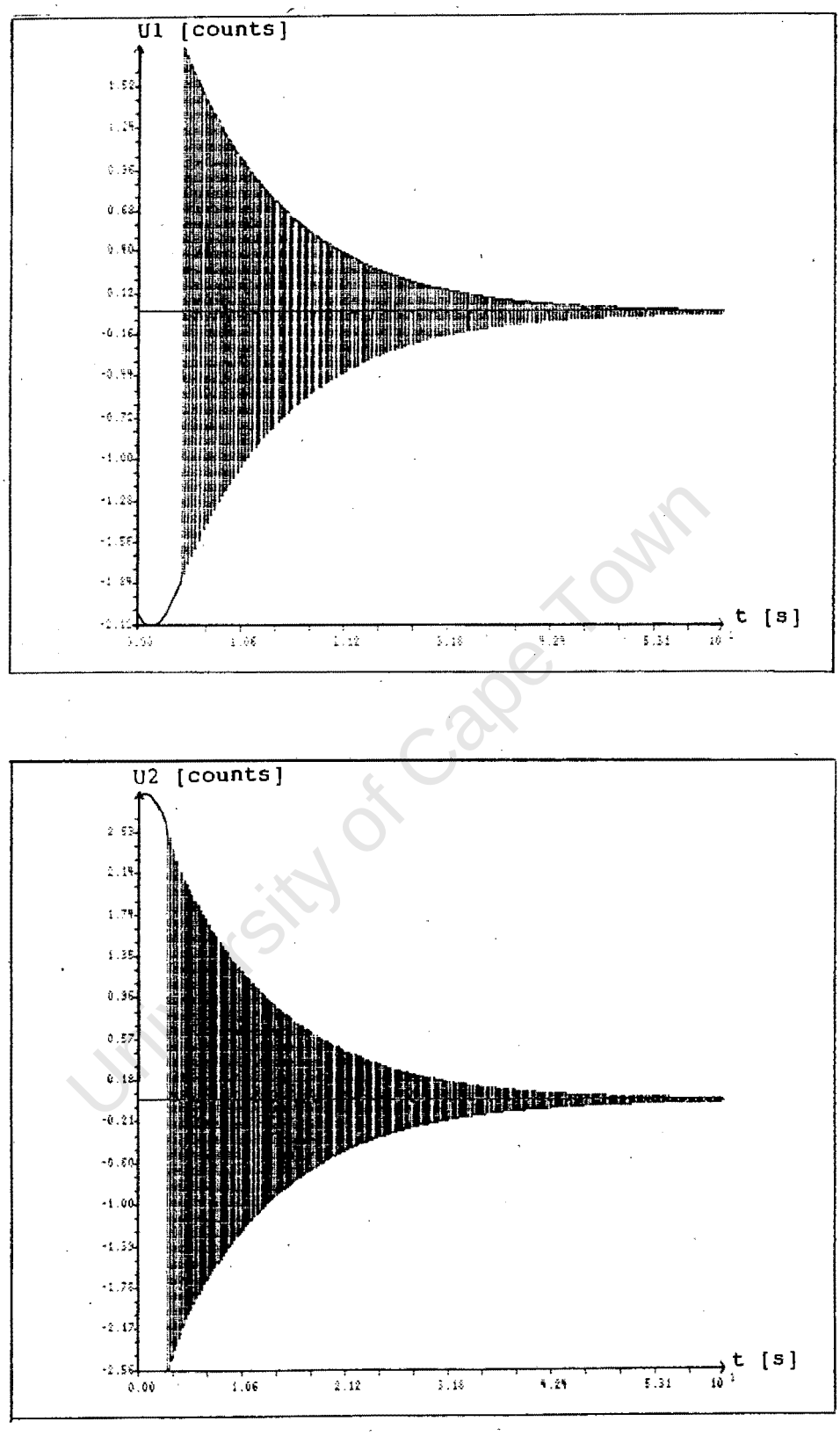


Figure 4.4 : The closed loop dynamics of the input functions .

It is estimated that the system reaches the sliding mode in approximately 5.3 seconds. The system has reached within 2% of its steady state in approximately 45 seconds. Thus, the closed loop behaviour is as predicted.

EXPERIMENT 2:

In experiment two, the effect of a slower sampling frequency was investigated.

Figures 4.5 and 4.6 illustrate the dynamics of the four system states, whilst figure 4.7 illustrates the system's control functions.

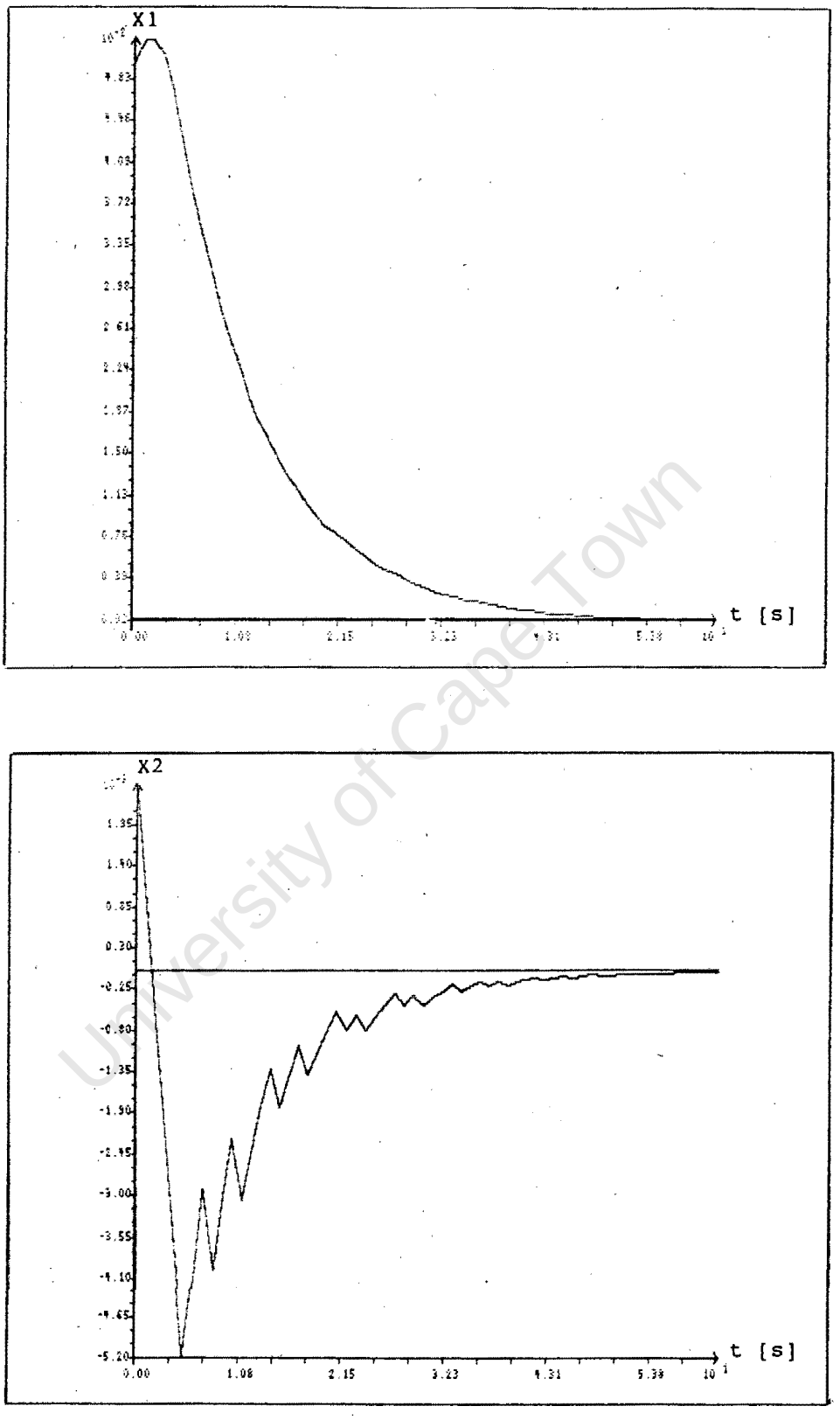


Figure 4.5 : The closed loop dynamics of states one and two .

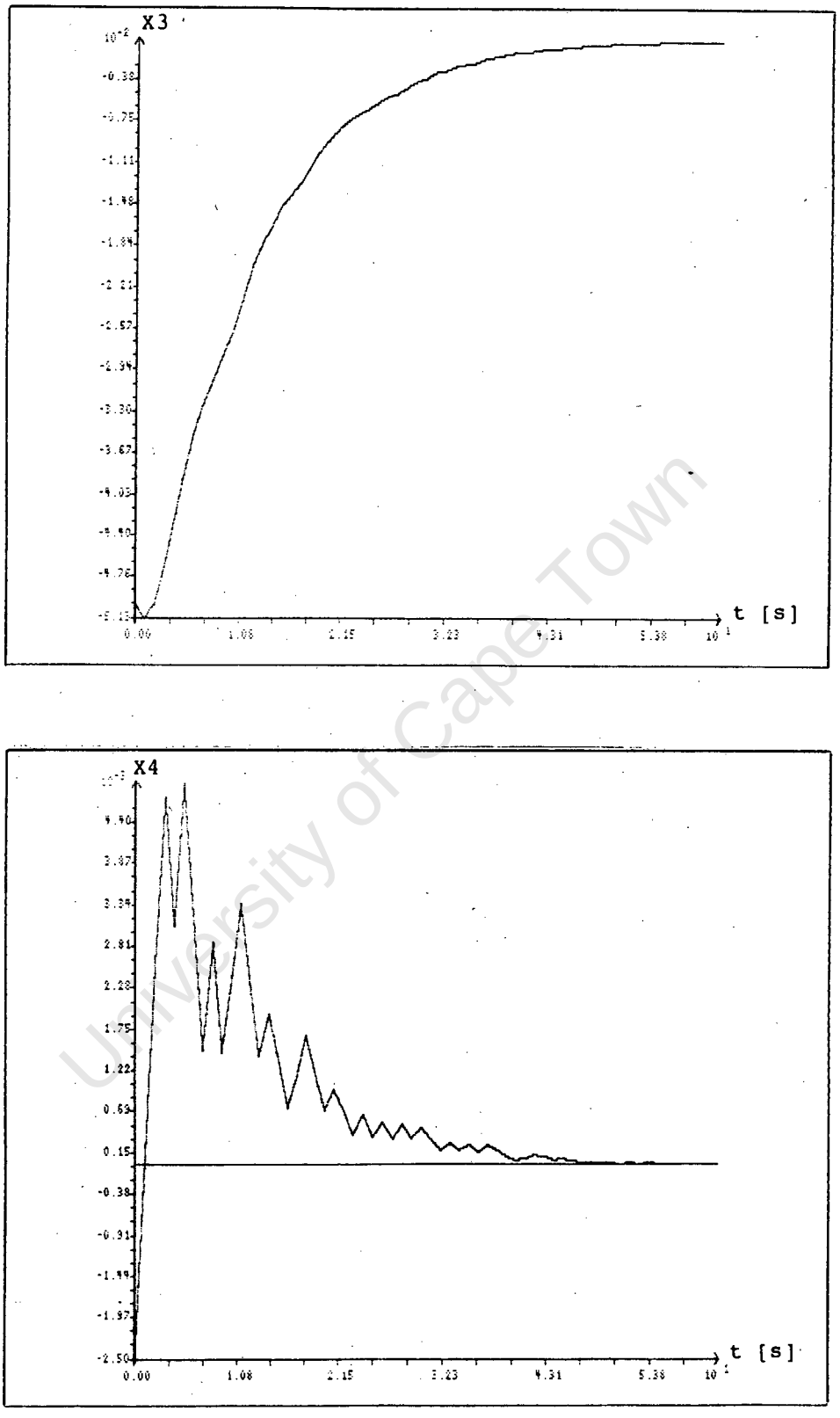
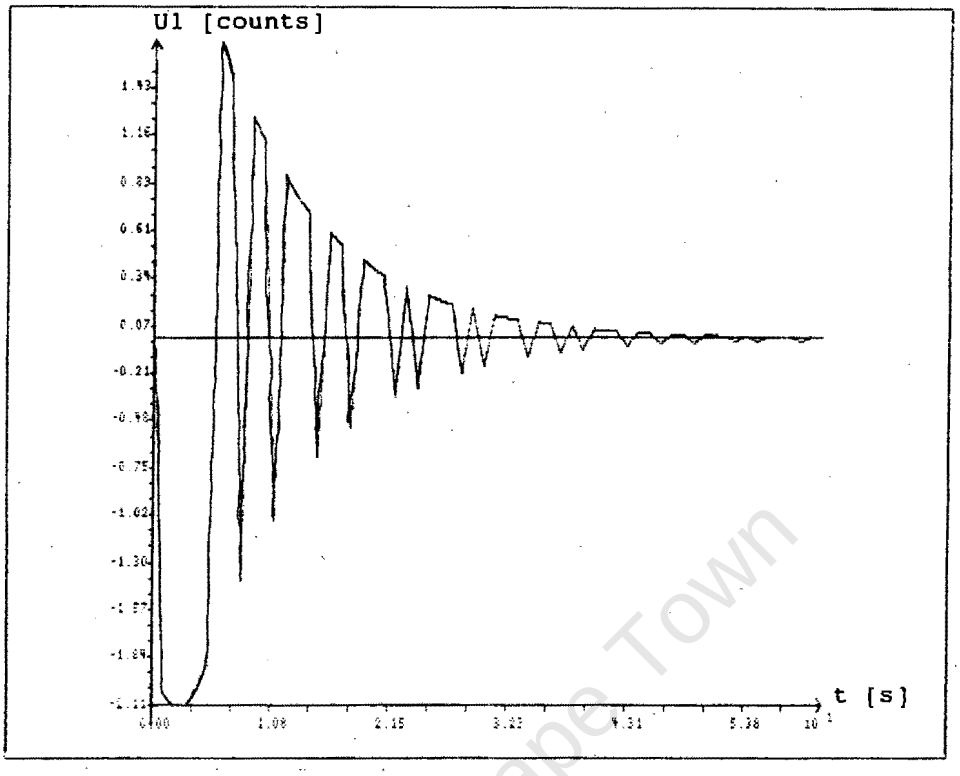


Figure 4.6 : The closed loop dynamics of states three and four .



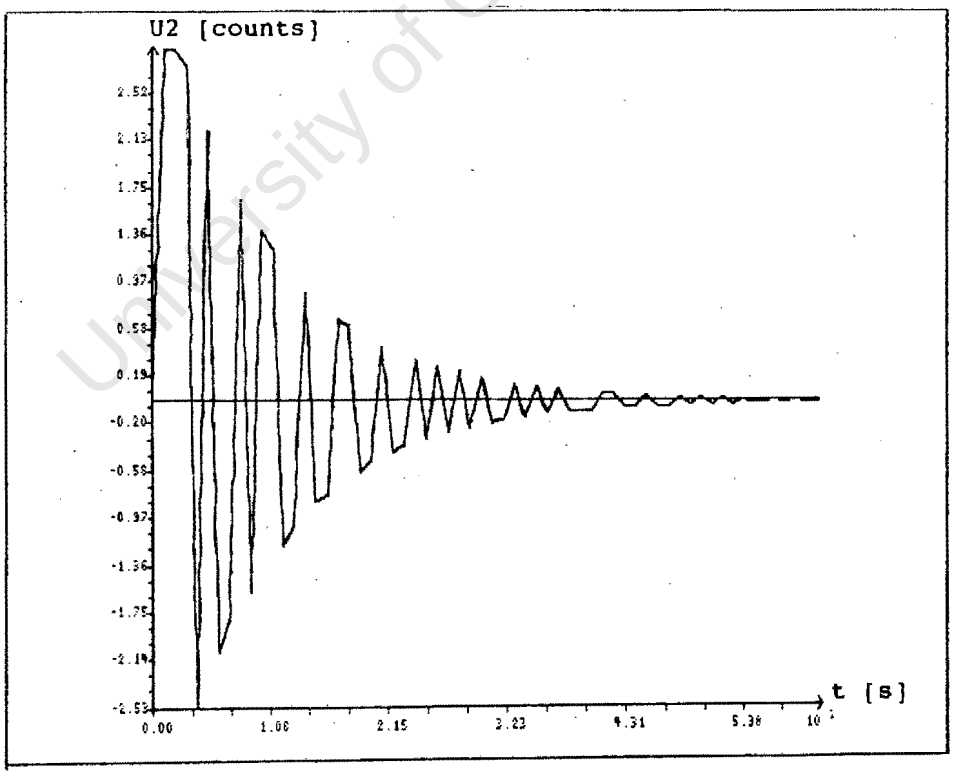


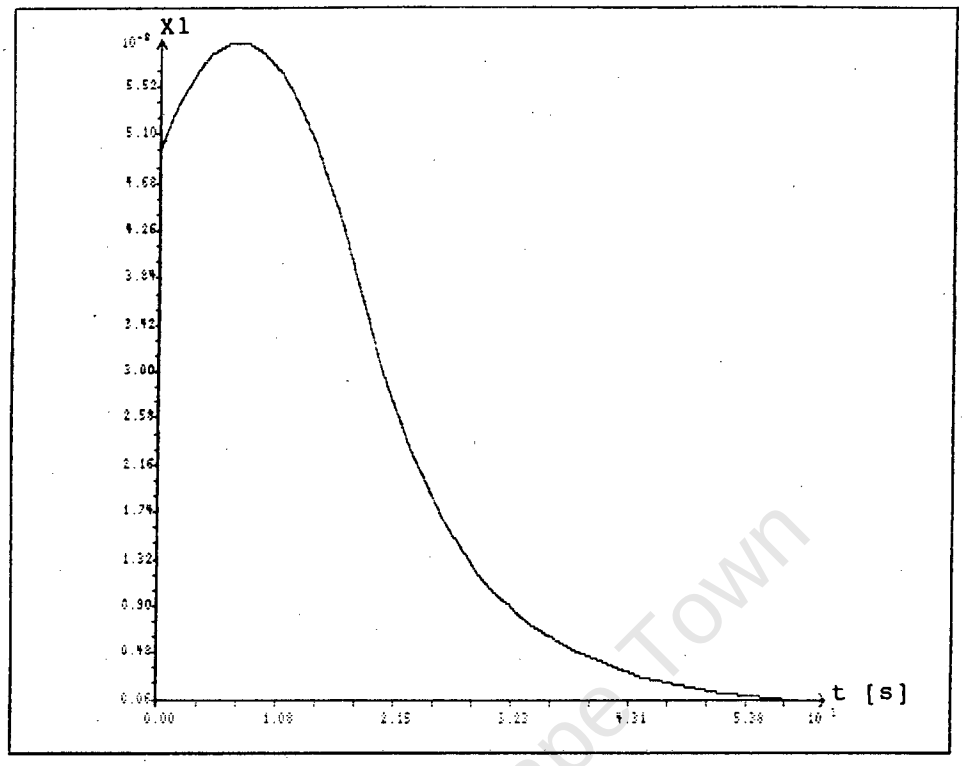
Figure 4.7 : The closed loop dynamics of the input functions .

Since the sampling frequency is still greater than the frequency of the closed loop poles, the system behaves closely to that of experiment 1.

The quasi sliding phenomenon is more pronounced in this experiment. Despite this imperfect way of sliding, the system's closed loop behaviour is not radically different from that exhibited in experiment 1.

EXPERIMENT 3:

The results of experiment 3 illustrate that the gain values, α_{ij} and β_{ij}, need not be chosen to be symmetric. These results can be seen in figures 4.8, 4.9 and 4.10.



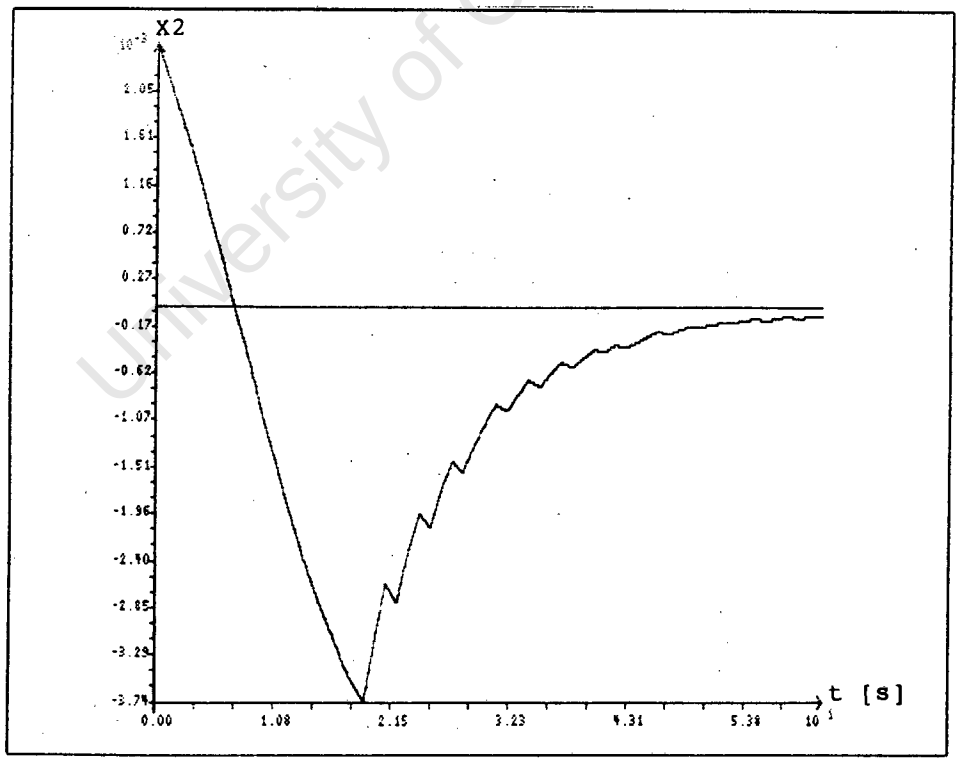


Figure 4.8 : The closed loop dynamics of states one and two .

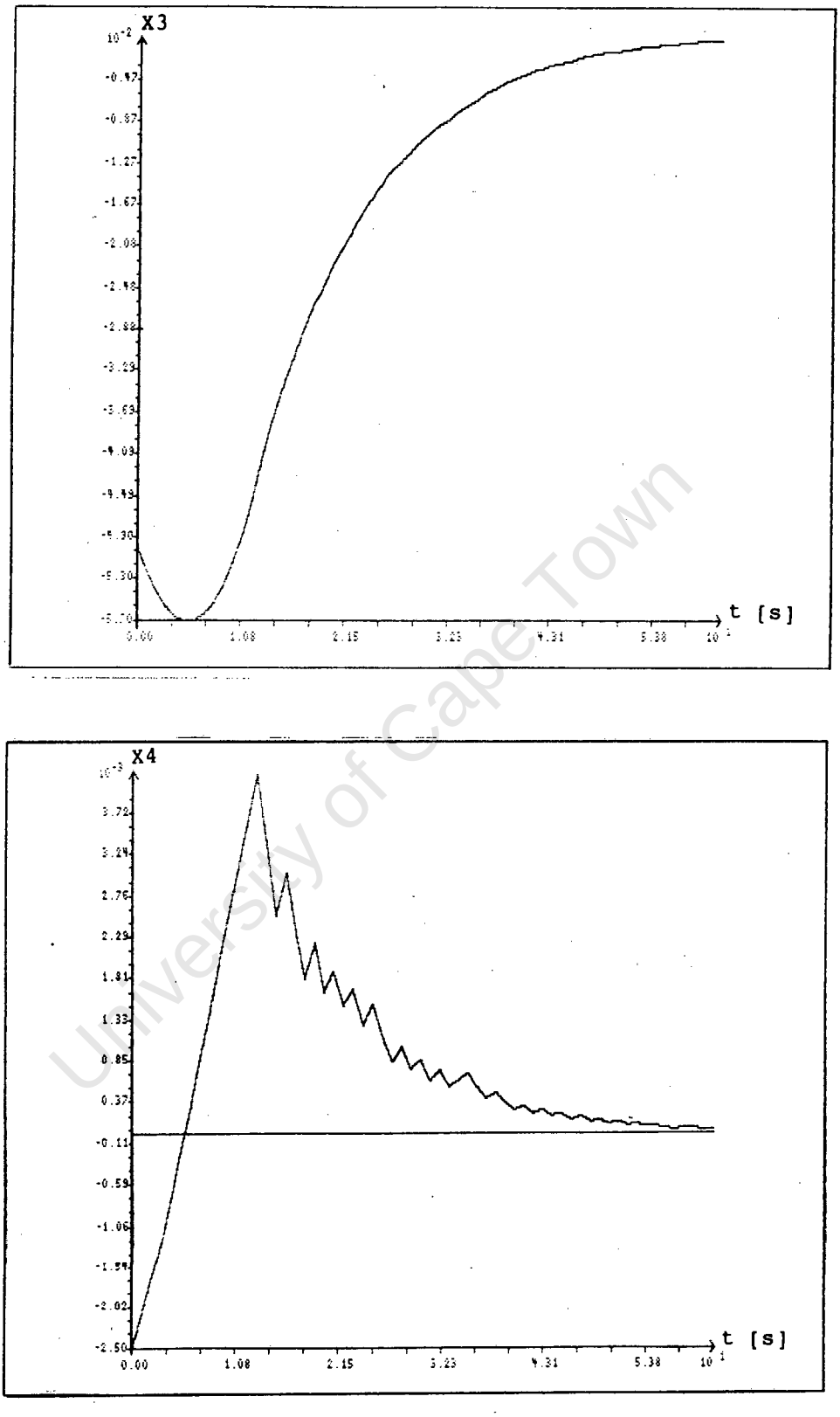


Figure 4.9 : The closed loop dynamics of states three and four .

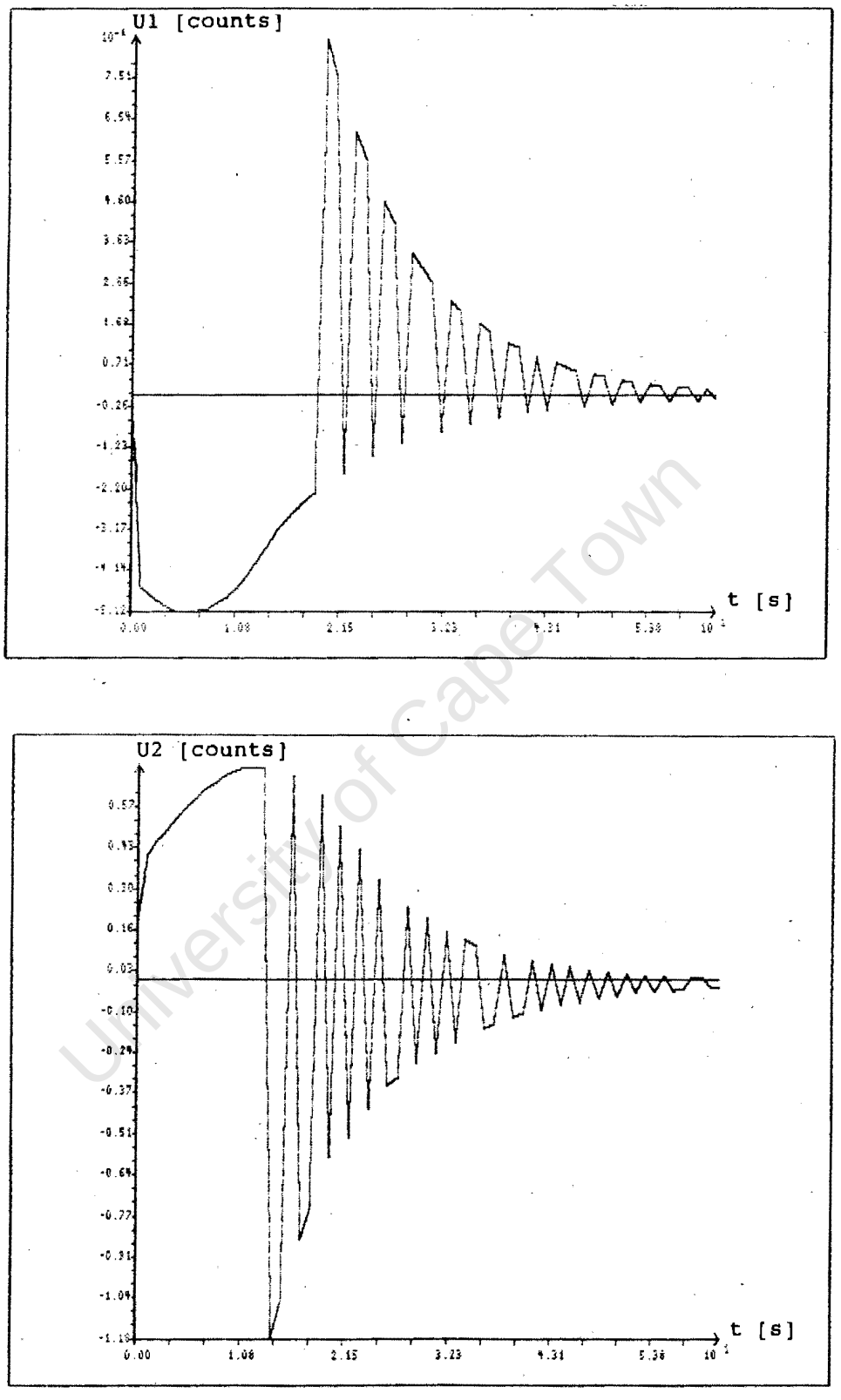


Figure 4.10 : The closed loop dynamics of the input functions .

REVIEW OF THE CHAPTER

In order to facilitate an easy transition from VSS theory of single input systems to VSS theory of multivariable processes, centralised control schemes were discussed. In such schemes, a number of single input systems are controlled by one centralised controller.

General variable structure controllers for multi input, multi output systems were analysed.

The design was split up firstly by defining an equivalent control scheme to define the sliding mode, and then by designing the specific control functions that ensure the existence of the sliding mode.

In the equivalent control method, an equivalent system A matrix (equation 4.16) is found, which is dependent on the C matrix. By choosing C, the closed loop eigenvalues are specified. Thus, a desired closed loop transient performance can be met by the design.

To ensure that a conflict does not arise between input functions attempting to drive the RP to their associated hyperplane, a control hierarchy is specified.

This design principle and the method of equivalent control are illustrated with an example.

A number of simulated experiments were carried out on the ensuing design.

It was found that despite practical problems, such as having a finite sampling rate, the system behaved as predicted.

CHAPTER 5

APPLICATION OF VARIABLE STRUCTURE SYSTEM THEORY TO A FLOTATION CELL SYSTEM

5.1 THE FLOTATION PROCESS

Flotation is a well known process for mineral extraction and refinement. Grains of mineral ore, or compound in a pulp (slurry), rise to the slurry surface in a cell (tank) by the action of bubbles of air. The grains are caught in a froth formed on the surface of the cell and are removed with the froth. Not all the grains are caught in the froth and thus a network of cells is usually used in an attempt to extract the remaining grains.

From an economic perspective, it is important to control the levels of the individual cells so as to avoid losing any grains.

Figure 5.1 shows the cell configuration of a flotation plant simulator. The cell has a number of outputs:

(1). **The Concentrate Output** - The inverted U - piece prevents any product from flowing out of the cell until the level of slurry exceeds the height of the cross - piece. The breather pipe on the cross - piece prevents any product from being siphoned out of the cell once the product level drops below the level of the cross - piece. The concentrate stream is not controlled directly.

(2). **Tailings Output** - The tailings stream is controlled by a pneumatic control valve. This allows the resultant level of the cell to be controlled.

(3).**Additional Outputs** - In addition to the above mentioned outputs, there are safety Cell and System overflow streams pipes to prevent flooding.

Each cell is controlled by monitoring its level and applying a suitable control signal to the output valve.

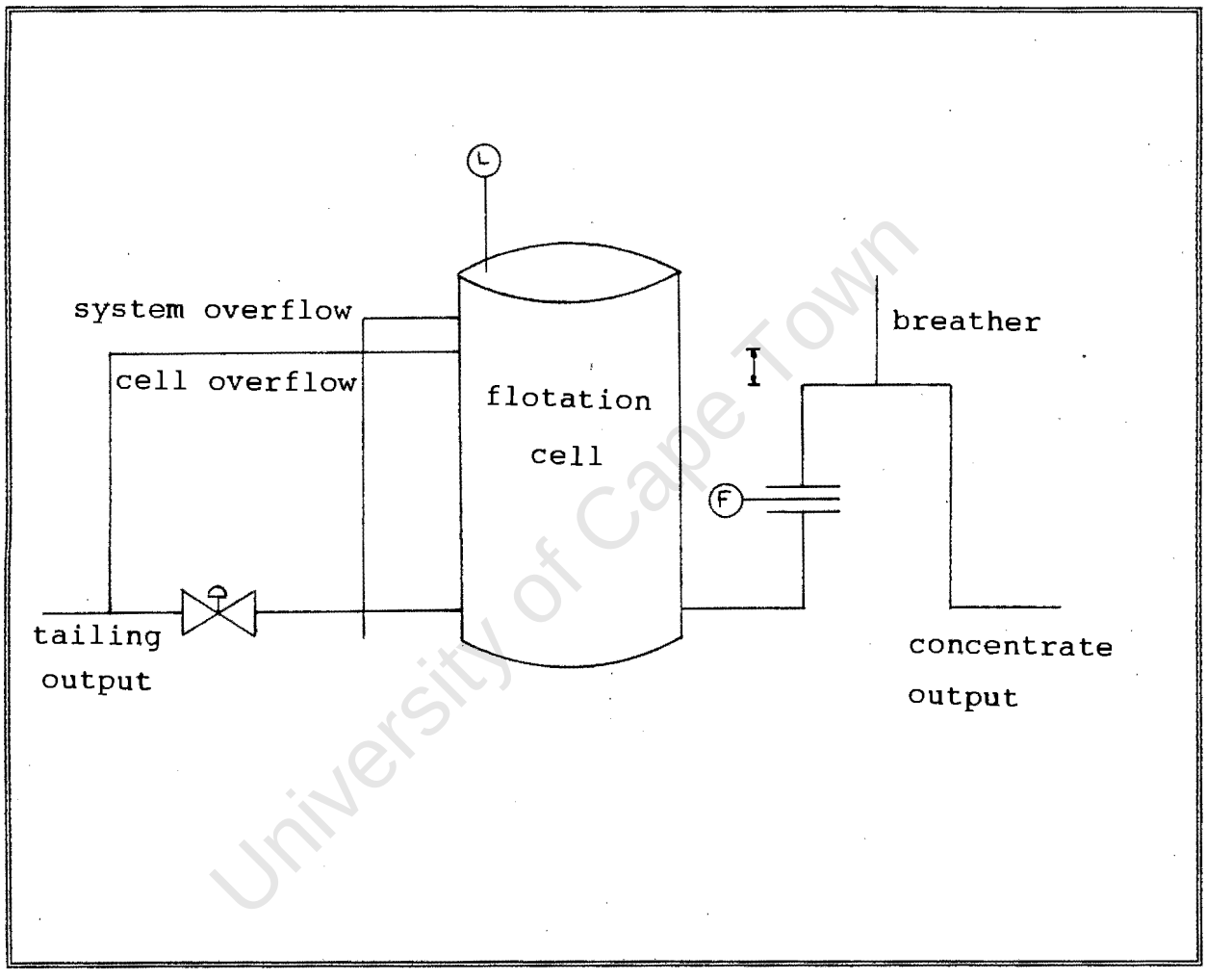


Figure 5.1: Cell configuration of a flotation plant simulator.

5.2 THE PLANT DESCRIPTION

Figure 5.2 shows the two tank system studied. A photograph of the rig can be seen in photograph 1. It consists of the following components:

1. **ROUGHER TANK** : This tank differs from the above cell description in only one respect; its concentrate output pipe has been sealed off. Thus, this output has been removed from the system.

2. **SCAVENGER TANK** : This tank is the same as the cell described in the previous section.

3. **SUMP** : This cell is smaller than the rougher and scavenger cells. It has only one input, the concentrate output of the scavenger tank, and one output, which is fed back as an input to the rougher tank.

4. **SPILLAGE CATCH TRAY** : This is both the source and the sink for water pumped into and out of the tanks.

5. **PUMPS** : The two pumps are dry running and of a fixed speed.

6. **MANUALLY ADJUSTED VALVES** : By changing the initial setting of these valves, the dynamic behaviour of the system changes.

7.COMPUTER AND INTERFACING : The levels of the two tanks are monitored by level probe transducers. The ensuing voltage is interfaced to the I.B.M. compatible personal computer via a 12 bit, 0 to 4095 counts, A/D card.

The driving signal from the computer is converted to an analogue signal via a 12 bit, D/A card. The signal is then converted to a pressure to open and close the valves.

It is assumed that the manually adjusted valves are at a constant setting for a particular set of experiments.

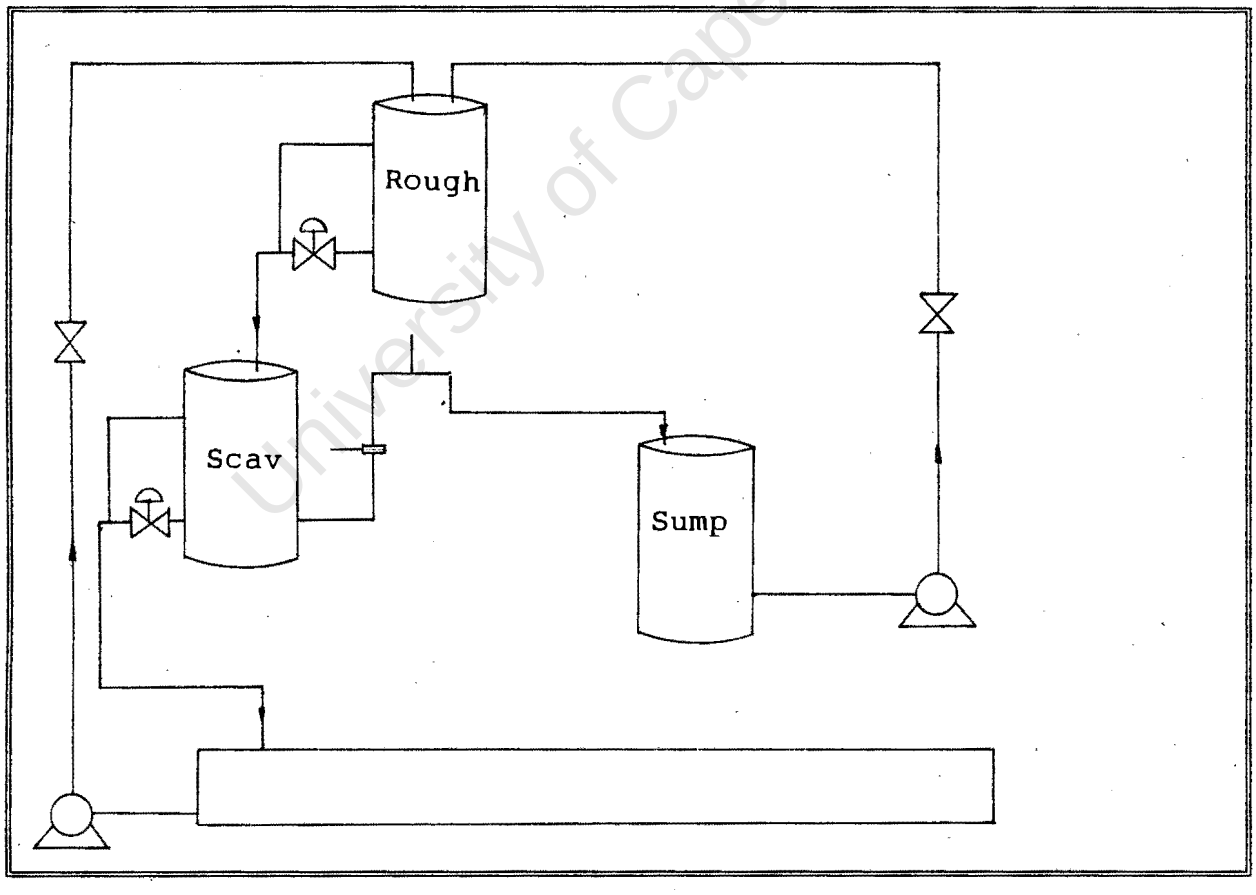


Figure 5.2 : The Plant.

Photograph 1: The Flotation Rig.

The plant has two inputs and two measured outputs. The first cell (the rougher tank) is fed by a constant, uncontrolled flow of water from the spillage catch tray. In addition, its tailing output serves as the input to the second cell (the scavenger tank).

The concentrate output from the scavenger tank goes to the sump, where it is fed back up to the rougher tank.

The tailing output of the scavenger tank goes out to the spillage catch tray. It is important to note that this is the only output where water can be removed from the two tank system.

The system's behaviour differs from that of a linear, time invariant plant in a number of areas. These are immediately apparent after studying its open loop behaviour.

1. The cell model parameters change as the water levels in the tanks vary. There is a radical change when the levels cross or operate in the region of an outlet point.

   As an example of the radical changes in parameters, the rougher tank was modelled with a first order transfer function for a particular manual valve setting ([6]):

   $$G(s) = A/(1 + s*T) \quad [\%/\%] \qquad \ldots(5.1)$$

   where

   A is the gain of the transfer function and T is the time constant.

   These two parameters were found to vary according to:

   $$0.5 \leq A \leq 2.8 \quad [\%] \qquad \ldots(5.2)$$

   $$60 \leq T \leq 300 \quad [\text{secs}] \qquad \ldots(5.3)$$

2. Owing to the suction of the feedback pump linking the sump to the rougher tank, a sinusoidal wave is superimposed onto the output of the rougher tank. The size of the amplitude of the sinusoid, is dependent on the level of the scavenger tank.

   This superimposed noise is illustrated in figure 5.3, where a regression exercise was performed on the open loop step test data of the rougher tank to obtain its parameters.

   When the level of the scavenger tank is below the height of the inverted U - piece, no water reaches the sump. This means that whilst this has in effect got rid of the noise problem, it has cut off the feedback signal to the rougher tank; radically changing the system model.

   Once the water level of the scavenger tank is above the minimum height ensuring a feedback signal, the amplitude of the sinusoidal noise can vary, as more or less water goes into the sump.

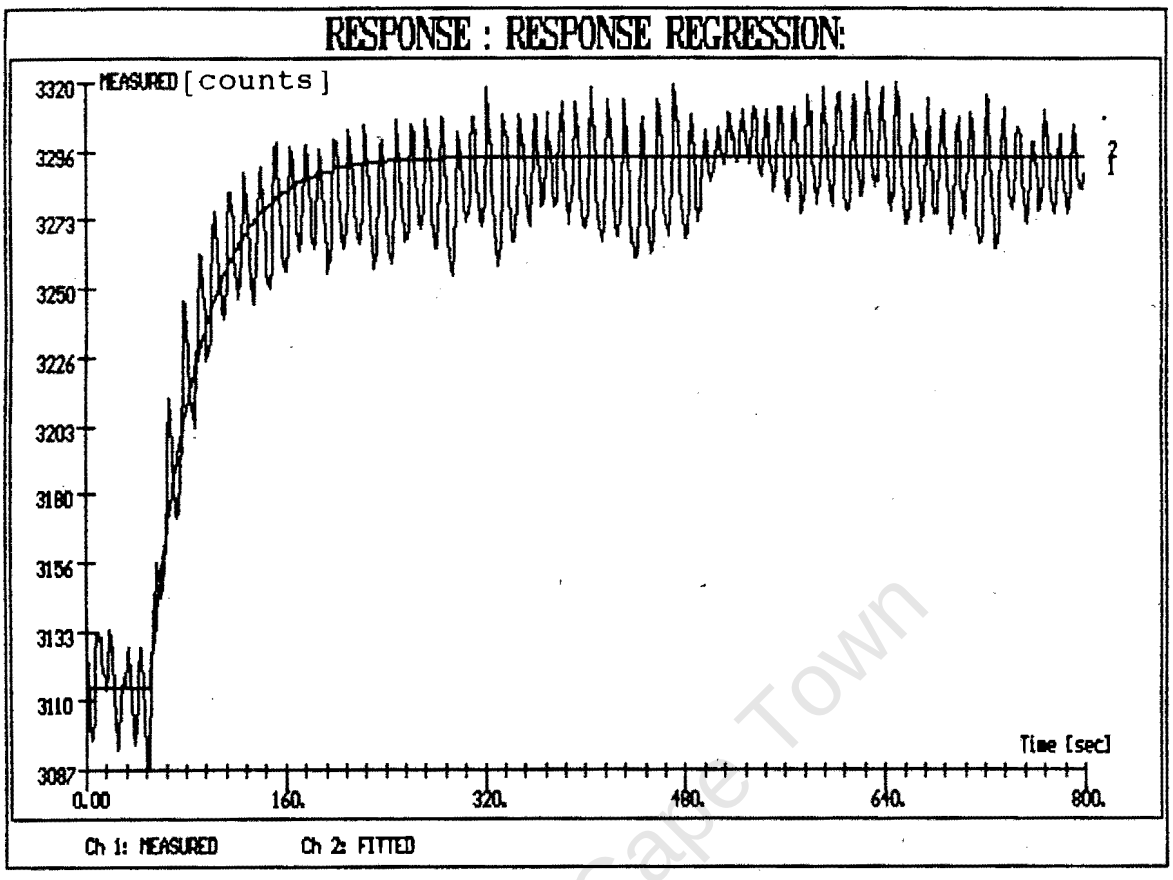


Figure 5.3: Illustration of the noise superimposed on the rougher output signal by the action of the feedback pump.

3. The valves controlling the output flow from the tanks are highly non - linear. This further reduces the operating region.

4. The amount of water in the spillage catch tray diminishes with evaporation. Care should be taken to ensure that a minimum satisfactory amount of water is available in the system.

Clearly, the system is seen to depart from an ideal linear time invariant plant in many ways.

5.3 CLOSED LOOP CONTROL OF THE ROUGHER TANK

This section describes the design and implementation of variable structure controllers to control the water level of the rougher tank. It is assumed that the water level of the scavenger tank remains reasonably constant throughout the experimentation.

5.3.1 SYSTEM MODEL

When defining a system model, a linear region has to be sought. From figure 5.1, one sensible linear operating region for the rougher and scavenger tanks is somewhere between their respective input and outlet pipes.

The chosen operating region can be seen in figure 5.4.

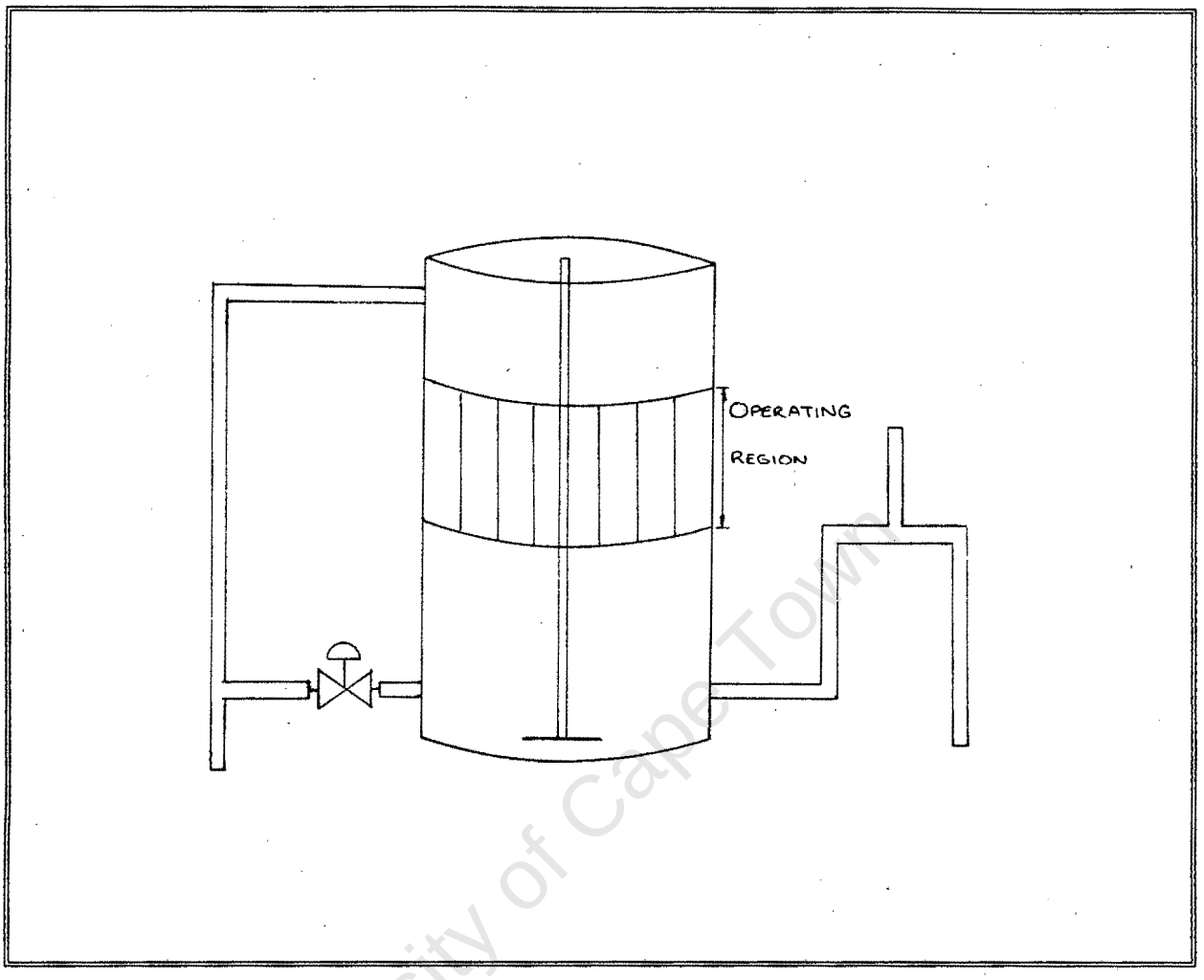


Figure 5.4: Operating region for Rougher and Scavenger tanks.

The procedure for the step tests was first to get the levels of the scavenger and rougher tanks to settle in the specified operating region. The rougher valve was then stepped by a fixed amount and the rougher level was recorded.

Within this region, plant parameters varied to a much lesser degree. A sample step test result is shown in figure 5.5.

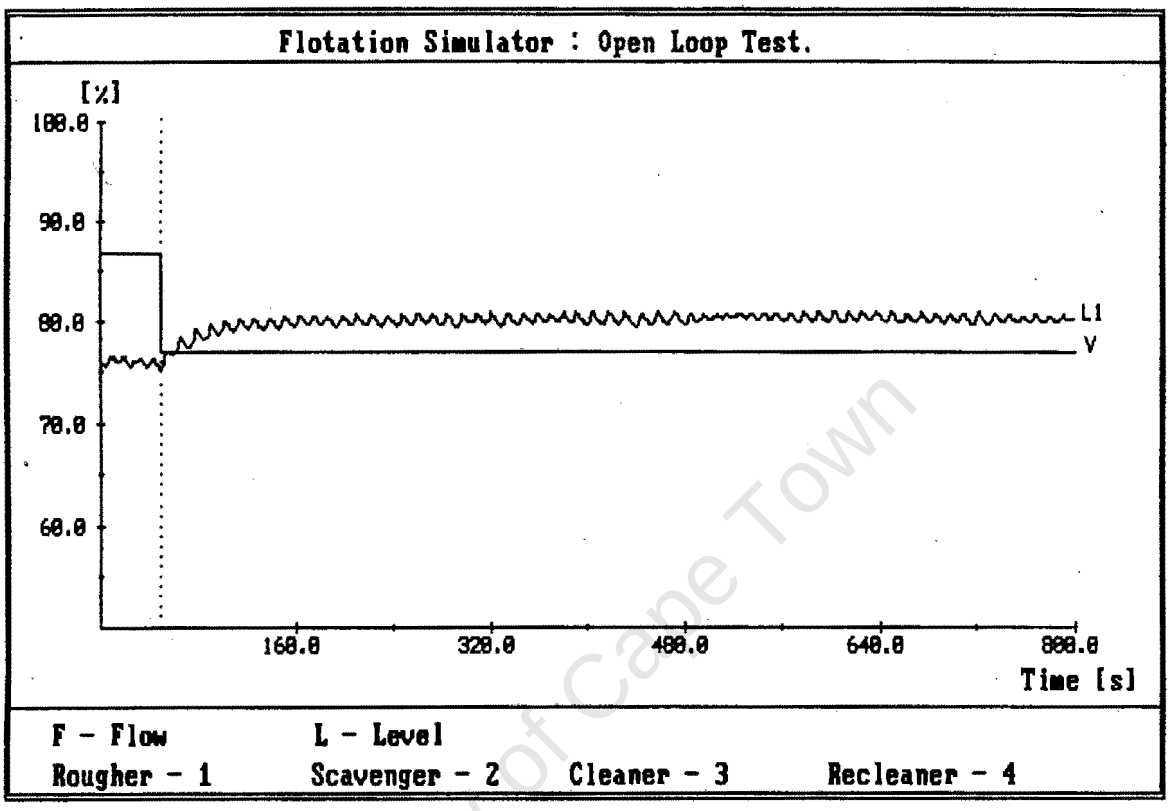


Figure 5.5: Open loop step test on Rougher tank.

A nonlinear regression program, NELM ([3]), was run on the step test data.

The result of the regression exercise is seen in figure 5.3.

The model for this particular manual valve setting is assumed to be of first order .

Even within this operating region, it was found that the gain varied according to:

$$0.40 \leq \text{Gain} \leq 0.44 \quad [\%/\%] \qquad \ldots(5.4)$$

The time constant was seen to vary according to:

$$33.9 \leq \text{Time constant} \leq 49.8 \quad [\text{secs}] \qquad \ldots(5.5)$$

For simulation purposes, a model with fixed parameters was chosen. Specifically, it is assumed that the level of slurry (water) in the tank can be related to the valve setting by the transfer function:

$$G(s) = 0.44/(1 + 33.9*s) \quad [\%/\%] \qquad \ldots(5.6)$$

There is an immediate problem which arises from having a model of first order: variable structure systems are assumed to be at least of second order to obtain the minimum two states.

The problem can be alleviated by cascading the plant with an integrating filter with a transfer function:

$$G_f(s) = 1/s \quad [\%/\%] \qquad \ldots(5.7)$$

The effective system transfer function is now of second order. This has some interesting consequences:

1. The high frequency control signal of variable structure controllers is an undesirable feature, as it could damage the control valves. By placing the integrating term between the VSS input and the actual plant input, the high frequency perturbations are smoothed out.

2. The counts range of the input to the valves is 0 - 4095. This means that no negative input signal can be facilitated.

   The addition of the integrating filter helps alleviate this problem by summing the input values from an initial pedestal value.

3. The addition of a pole at the origin ensures a type 1 system; thus facilitating setpoint tracking.

The transfer function of equation 5.6 is thus modified by multiplying it with a "1/s" term. The following open loop state space model in phase canonical form can be obtained:

$$dx_1(t)/dt = x_2(t)$$

$$dx_2(t)/dt = -0.0293*x_2(t) + 0.013*u(t)$$

$$y(t) = [1\ 0]\mathbf{x} \qquad \ldots(5.8)$$

where

$y(t)$, $x_1(t)$ and $x_2(t)$ are measured in counts.

5.3.2 DESIGN OF VARIABLE STRUCTURE CONTROLLER

Suitable gain values for α and β (for a particular c value) need to be chosen to ensure a sliding mode exists and that it will be reached.

In the chosen tank operating region, the state space model of equation 5.8 is more accurately defined by the following values for $a_1(t)$,$a_2(t)$ and $b(t)$. The variations in model parameters were obtained by converting the values from equations 5.4 and 5.5.

$$a_1(t) = 0.0$$

$$0.0201 \leq a_2(t) \leq 0.0295$$

$$0.0081 \leq b(t) \leq 0.0130 \qquad \ldots(5.9)$$

The gain values for α and β for second order systems with variable parameters have to satisfy the existence (of a sliding mode) conditions given by equations 1.46 and 1.47. Since the system is to switch on the error states, the inequality signs are reversed.

Hence:

$$\alpha \leq \min_{a_1,a_2,b} \{(a_2*c - c^{\wedge}2 - a_1)/b\} \qquad \ldots(5.10)$$

$$\beta \geq \max_{a_1,a_2,b} \{(a_2*c - c^{\wedge}2 - a_1)/b\} \qquad \ldots(5.11)$$

Specifically, for a chosen c value = 0.01 (corresponding to a time constant of 100 seconds), equations 5.10 and 5.11 become:

$$\alpha \leq 0.0232 \text{ [counts]}$$

$$\beta \geq 0.0488 \text{ [counts]} \qquad \ldots(5.12)$$

When the system is out of the operating region, the plant parameters change more drastically. Despite the greater variations on the plant parameters, gain values can be chosen for α and β to satisfy equations 5.10 and 5.11.

5.3.3 PRACTICAL CONSIDERATIONS

There are a number of practical issues to consider when actually implementing a variable structure controller to the plant.

1. **CONTROL SIGNAL** : (a) This high frequency signal can place unreasonable strain on the control valve.
   (b) The input range to the control valve is limited to 0 - 4095 counts. Thus, a negative input cannot be catered for.

2. **SETPOINT TRACKING** : This problem has been solved by facilitating switching based on the error states.

3. **STATE ESTIMATOR** : Since only state one is available, some method is required to estimate state two.

It was noted earlier in the chapter that the addition of an integrating filter helps alleviate the problems associated with the variable structure control signal. The high frequency component of the signal is smoothed and for a suitably high initial input value, the summing action of the filter keeps the input positive.

It is important to note, however, that smoothing the input function of variable structure controllers is not recommended in general. The high frequency perturbations about the mean value of the signal is what gives VSS their inherent insensitivity to noise and plant parameter changes [5]. In this example, however, the filter has been taken into account as part of the process model. Thus as far as the cascaded model is concerned, the high frequency perturbations exist, but this signal is smoothed before reaching the control valve.

For particular control parameters, it is not always possible to have a big enough initial plant input to prevent the variable structure controller from needing to go negative in value. This problem is compounded by the highly nonlinear nature of the control valve and small linear operating region.

The addition of the integrating filter places a pole at the origin of the cascaded system; making it a type 1 system. This facilitates setpoint tracking by making the output signal directly equal to state one.

Since state one can be directly measured, only state two needs to be estimated. State two is the derivative of state one. A simple difference equation is used to estimate the derivative.

5.3.4 IMPLEMENTATION

Figure 5.6 shows the final system configuration .

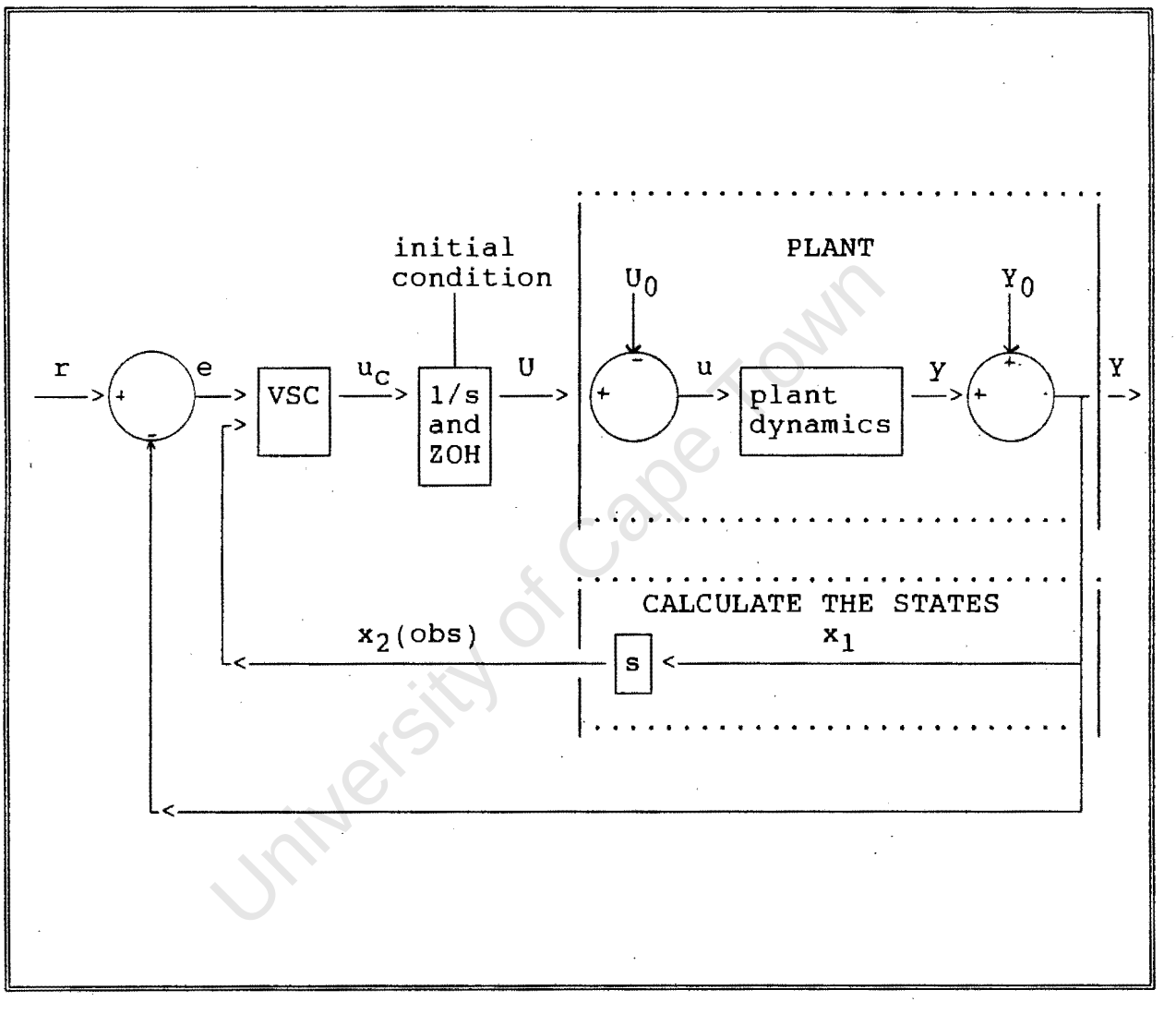


Figure 5.6: System configuration.

The equations which govern the controller are :

$$x_1(t)\_observed = x_1(t) = Y(t) \qquad \ldots(5.13)$$

$$x_2(t)\_observed = x_2(t) = (x_1(t) - x_1(t)\_prev)/dt \qquad \ldots(5.14)$$

where

$x_1(t)\_observed$, $x_2(t)\_observed$ are the estimated plant states.

The error states of the system are:

$$e_1(t) = r - x_1(t) \qquad \ldots(5.15)$$

$$e_2(t) = -x_2(t) \qquad \ldots(5.16)$$

The control signal then has the form:

$$u_c(t) = -\Gamma * e_1(t) \qquad \ldots(5.17)$$

where

$$\begin{aligned} \Gamma &= \alpha \quad \text{if } \sigma(t)*e_1(t) > 0 \\ &= \beta \quad \text{if } \sigma(t)*e_1(t) < 0 \end{aligned} \qquad \ldots(5.18)$$

and

$$\sigma(t) = c*e_1(t) + e_2(t) \qquad \ldots(5.19)$$

The integrator and zero order hold block is described by:

$$U(t) = U_{prev} + u_c*dt \qquad \ldots(5.20)$$

In addition:

$$\begin{aligned} \text{If } U(t) > 4095 &\text{ , then } U(t) = 4095 \text{ [counts].} \\ < 0 &\qquad\quad = 0 \quad \text{[counts]} \end{aligned} \qquad \ldots(5.21)$$

A number of experiments were performed in simulation and on the flotation rig.

Since the open loop time constant of the dynamic model is in the order of 35 to 50 seconds and the closed loop time constant will be chosen to range between 10 and 20 seconds, the following sampling time can be assumed for all experiments:

$$dt = 1 \text{ [second]} \qquad \text{...(5.22)}$$

EXPERIMENT 1:

The model of equation 5.8 was used in a computer simulation of the process. A noise signal was added to the model output to simulate the superimposed noise of the feedback signal to the rougher tank. From figure 5.3, the added noise is estimated to be of the form:

$$noise = 18*\sin(0.62832*t) \qquad \text{...(5.23)}$$

Equation 1.41 in chapter 1, describing the motion of the system in the sliding mode, gives a criterion to choosing a suitable c value. Since a closed loop time constant of 20 seconds was decided on, the switching line is defined by :

$$c = 1/20 = 0.05 \qquad \text{...(5.24)}$$

For this specific value of c, gain values for α and β were chosen to satisfy the existence conditions for a sliding mode given by equations 5.10 and 5.11.

Apart from satisfying the salient equations, the chosen gain values have to be big enough to overcome such problems as a finite switching time and variations in plant parameters.

On the other hand, their absolute size is limited by practical considerations such as the input signal being confined to 0 - 4095 counts.

Thus, practical choice of suitable gain values, α and β, is to a certain extent made by trial and error.

For simplicity, symmetric gain values were chosen. Specifically:

$$\alpha = -0.4$$

$$\beta = 0.4 \qquad \ldots(5.25)$$

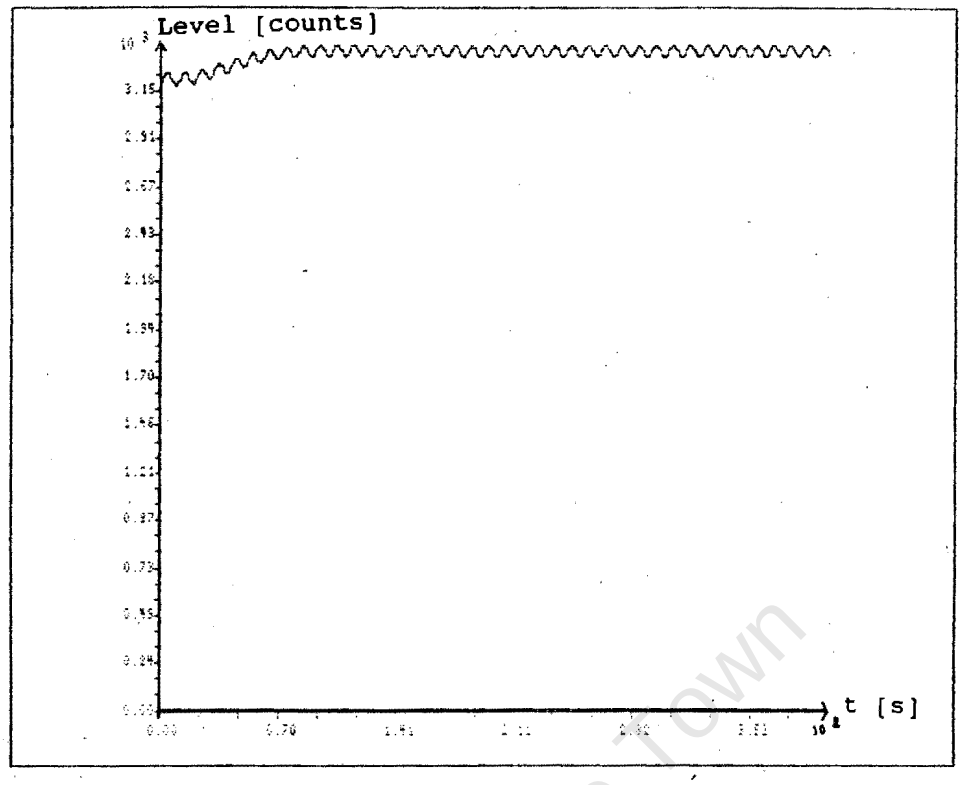
Initial conditions were set at:

x_1 = the output level state = 3200 [counts]. This is in the operating region.
x_2 = dx_1/dt = 0 [counts]
U = initial input to the system = 500 [counts] ...(5.26)

Equations governing the controller were as in equations 5.13 - 5.21.

The total period of simulation was for 400 seconds.

Figure 5.7 shows the closed loop level response to a step of 164 counts (4%).



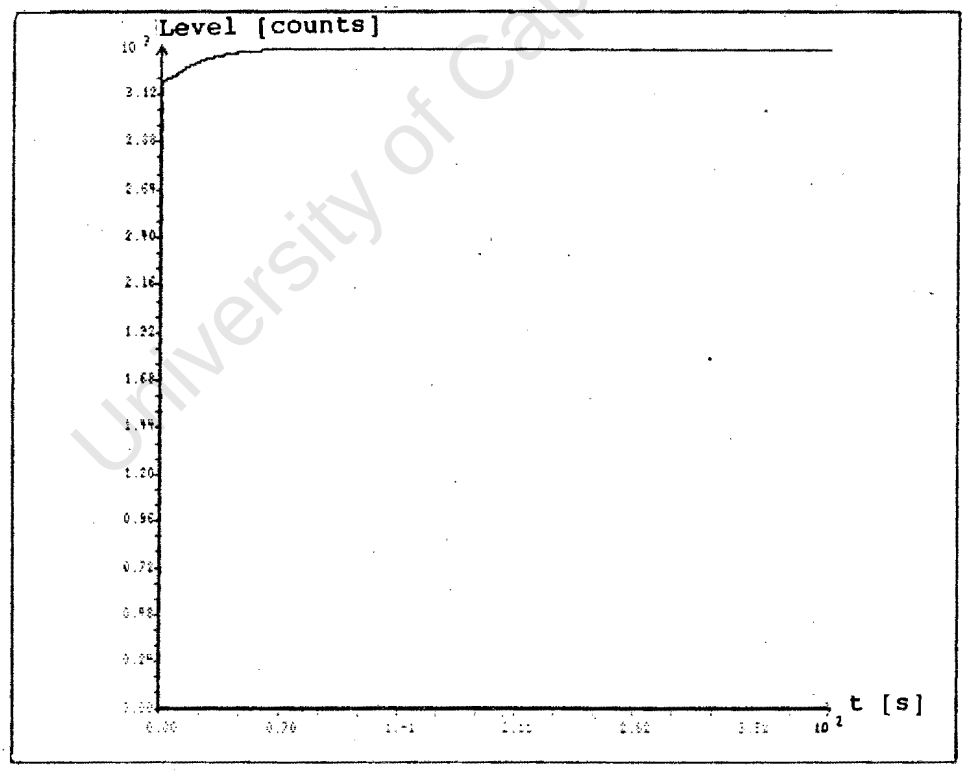


Figure 5.7 :Level response of rougher tank (a).with noise and (b) without noise.

There are a few points to note about this figure:

1. The system responds immediately and reaches the new setpoint.

2. When the system model is simulated with the added superimposed noise, as in figure 5.7(a), the error signal does not go to zero. This is as opposed to the case where the system is simulated without the noise signal; illustrated in figure 5.7(b).

3. The output changes from its initial value (0%) to its final value. The system is expected to reach within 2% of the setpoint in 4*20 = 80 seconds.

   From figure 5.7, the estimated time to reach the given setpoint is approximately 70 seconds, which corresponds with the theoretical prediction.

Figure 5.8 shows the perturbations about the switching line. The system starts to slide almost immediately and remains in the sliding mode throughout the simulation.

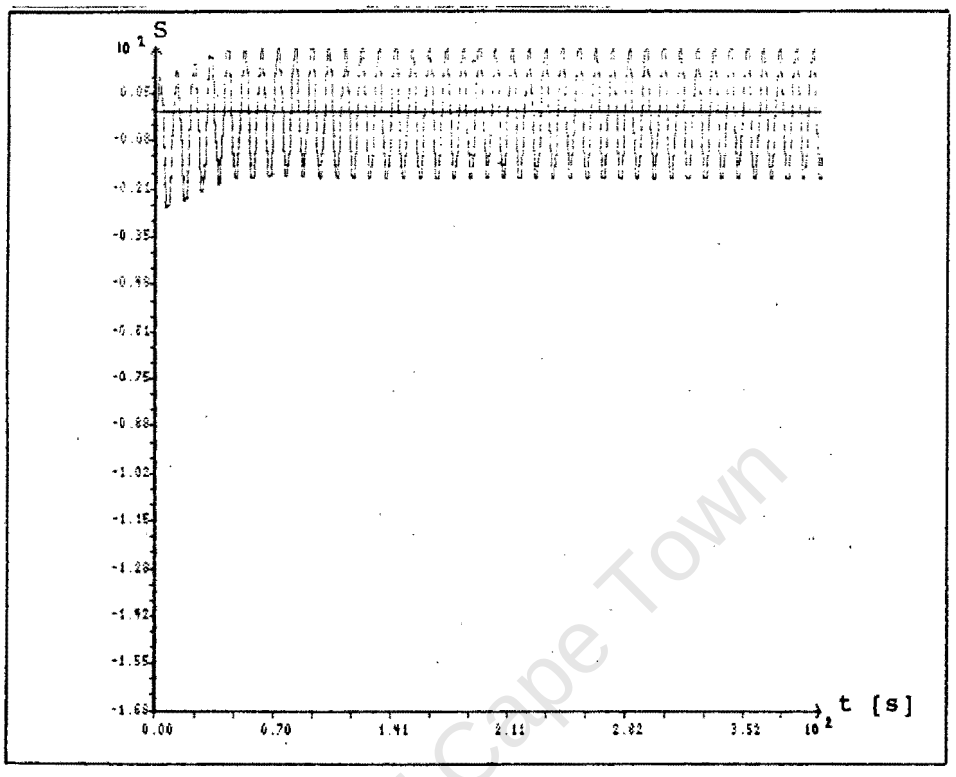
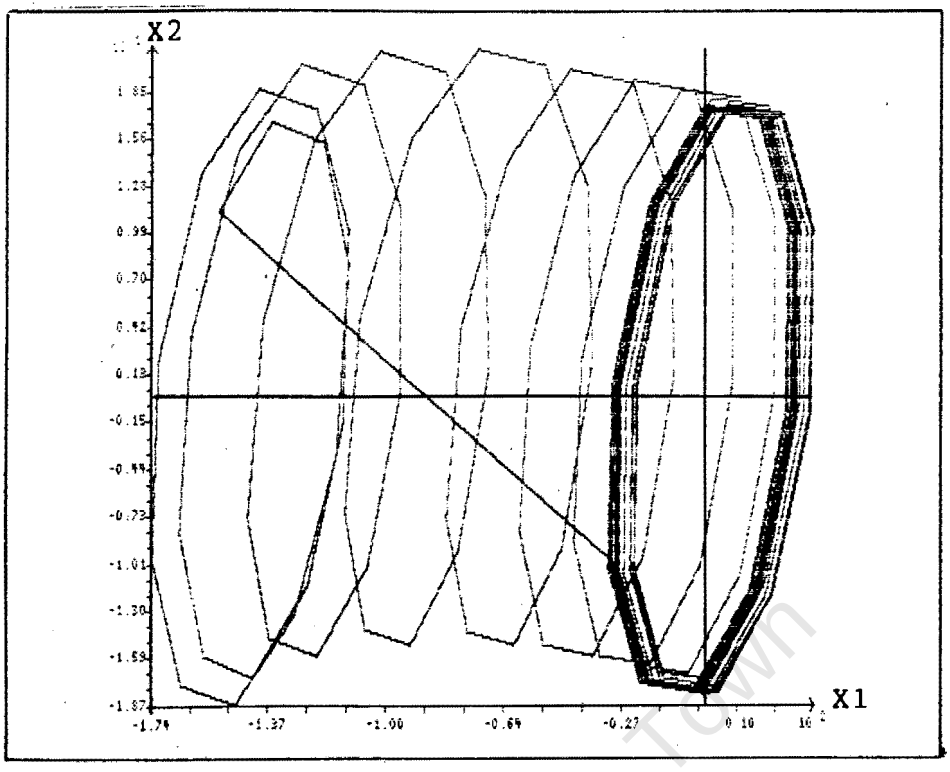


Figure 5.8:System switching line.

Figure 5.9(a) illustrates the cycling phenomenon discussed in chapter 3. The RP spirals in towards the origin and finally remains indefinitely caught in a cycle about the origin.

Without the noise being added to the system, no such cycling exists, as illustrated in figure 5.9(b).



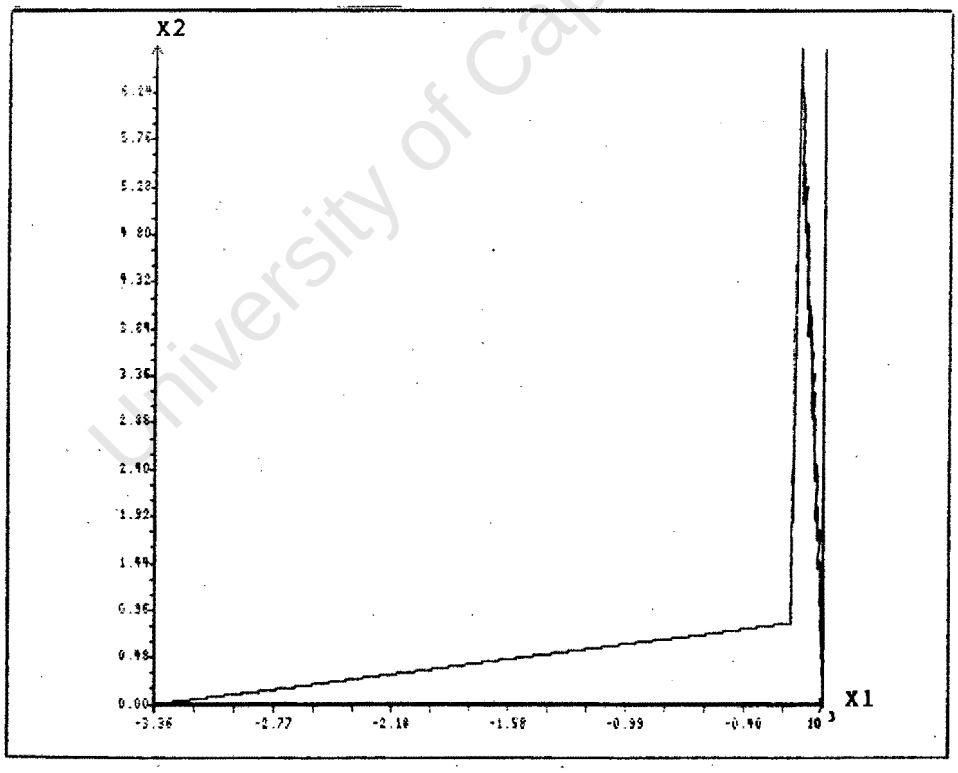


Figure 5.9 :Phase portrait of system (a) with noise and (b) without noise.

Figure 5.10 shows the variable structure control signal plotted against time. As the system slides in towards the origin, the amplitude of the signal diminishes. It does not disappear altogether however, owing to the system cycling about the origin.

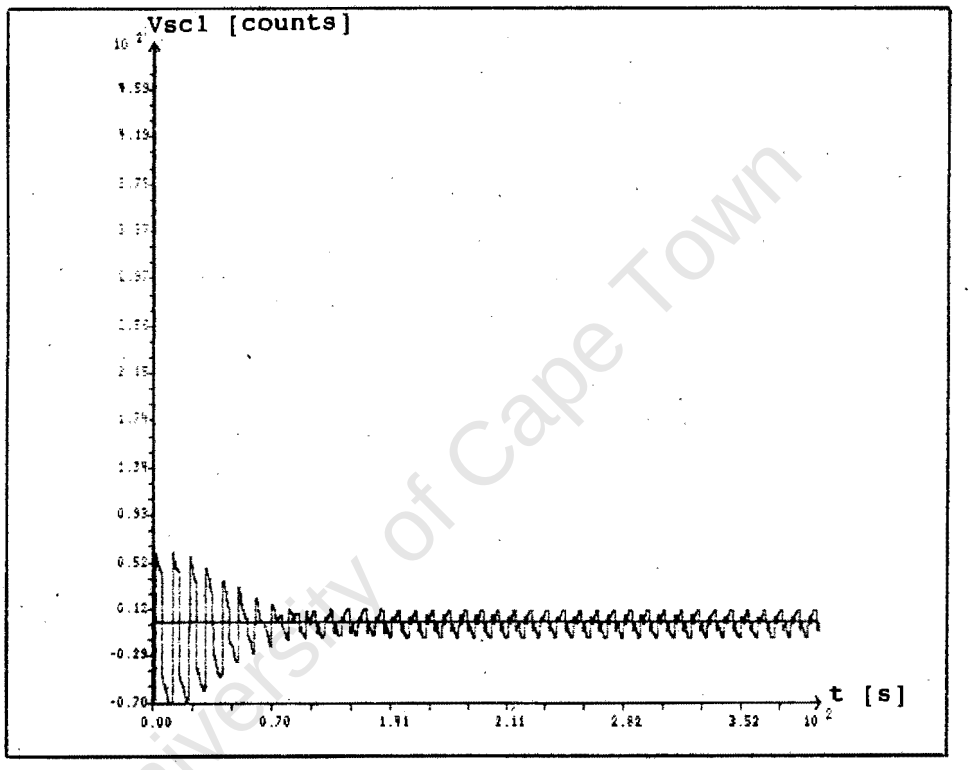


Figure 5.10: Variable structure control signal.

Figure 5.11 illustrates the effect of integrating the variable structure control signal. The 0 - 4095 counts constraint on the input function is seen to be met.

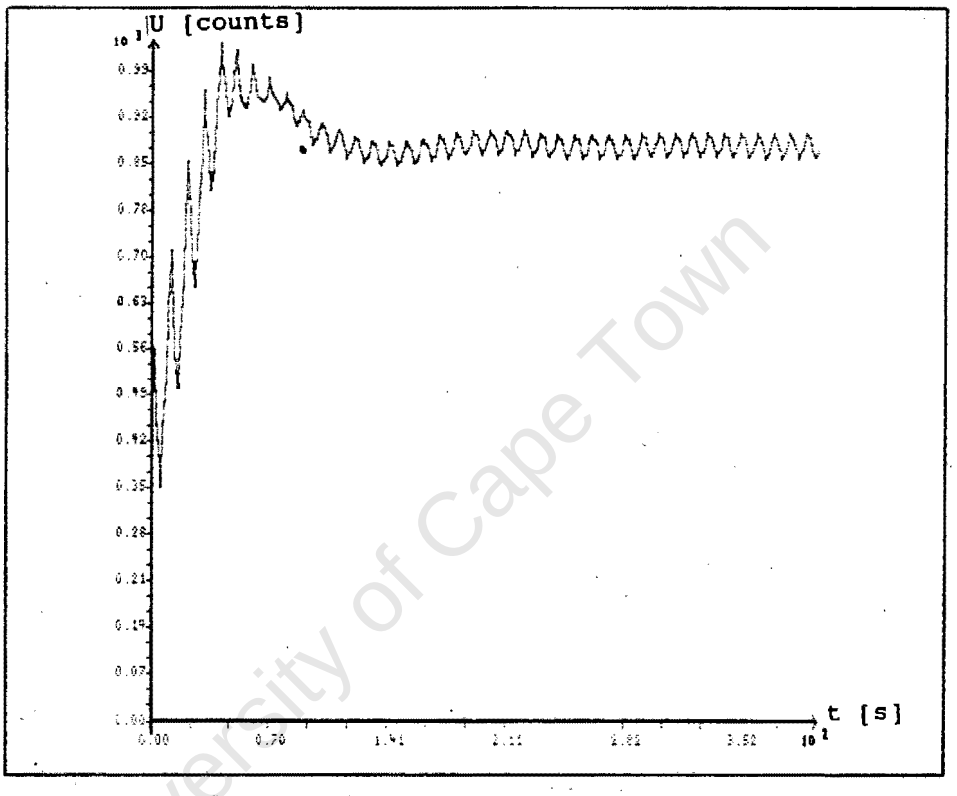


Figure 5.11 : System input.

The same controller was then implemented in the real system. It was assumed that the scavenger level remained constant in the defined linear region for the duration of the test.

This experiment runs for 550 seconds.

Figure 5.12 shows the level of the rougher tank, the level of the scavenger tank and the input function. The level of the rougher tank is seen to follow the setpoint. The time to reach steady state is estimated from this plot to be about 62 seconds. This corresponds well to the estimated 70 seconds of the corresponding simulated plot and theoretical prediction of 80 seconds.

It is important to note, that the output level of the scavenger tank does not remain constant.

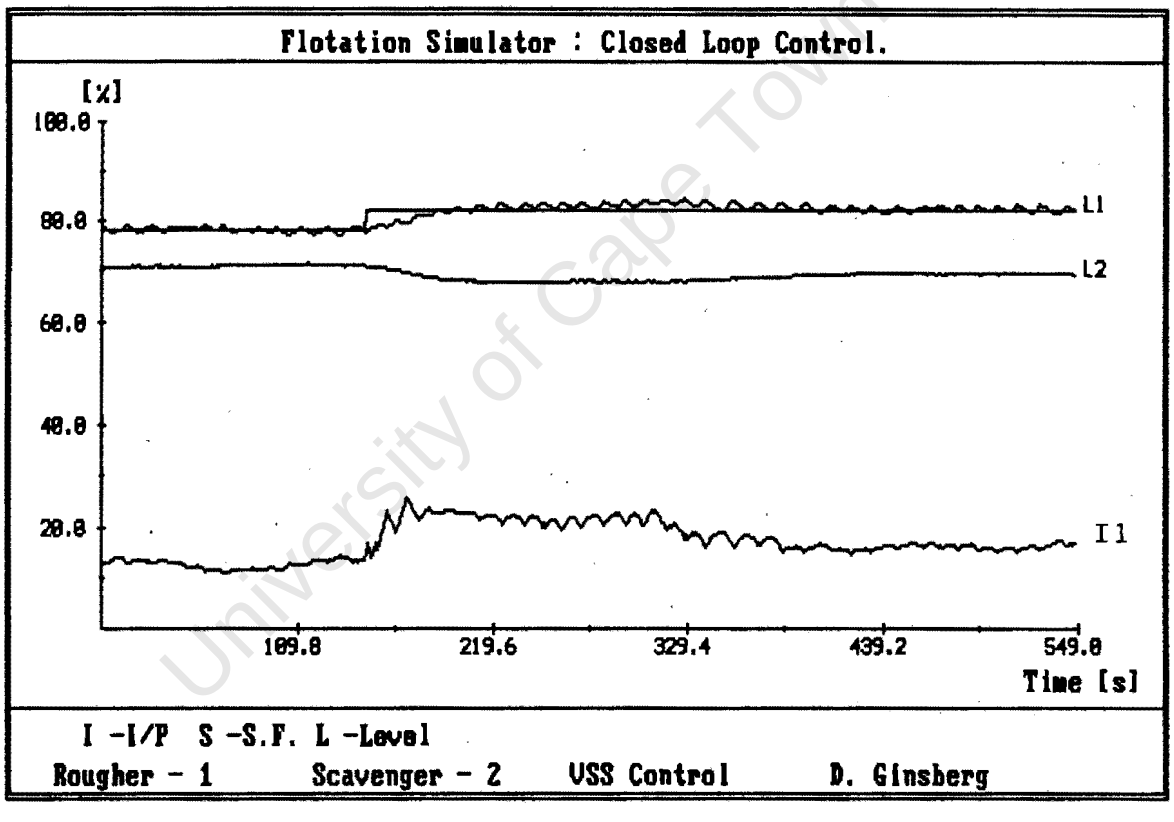


Figure 5.12 : Closed loop step test - control level of rougher tank.

The practical results shown in the above figures correspond reasonably well with the theoretical plots attained in simulation.

EXPERIMENT 2:

The aim of this experiment is to see the effects of increasing the magnitudes of the design parameters.

In order to double the predicted speed of response, the chosen c value is double the value of the previous experiment.

$$c = 0.1 \qquad \ldots(5.27)$$

This gives a time constant of 10 seconds.

Again, to satisfy the salient existence and hitting conditions as well as the other criteria discussed in the previous experiment:

$$\alpha = -5 \text{ [counts]}$$

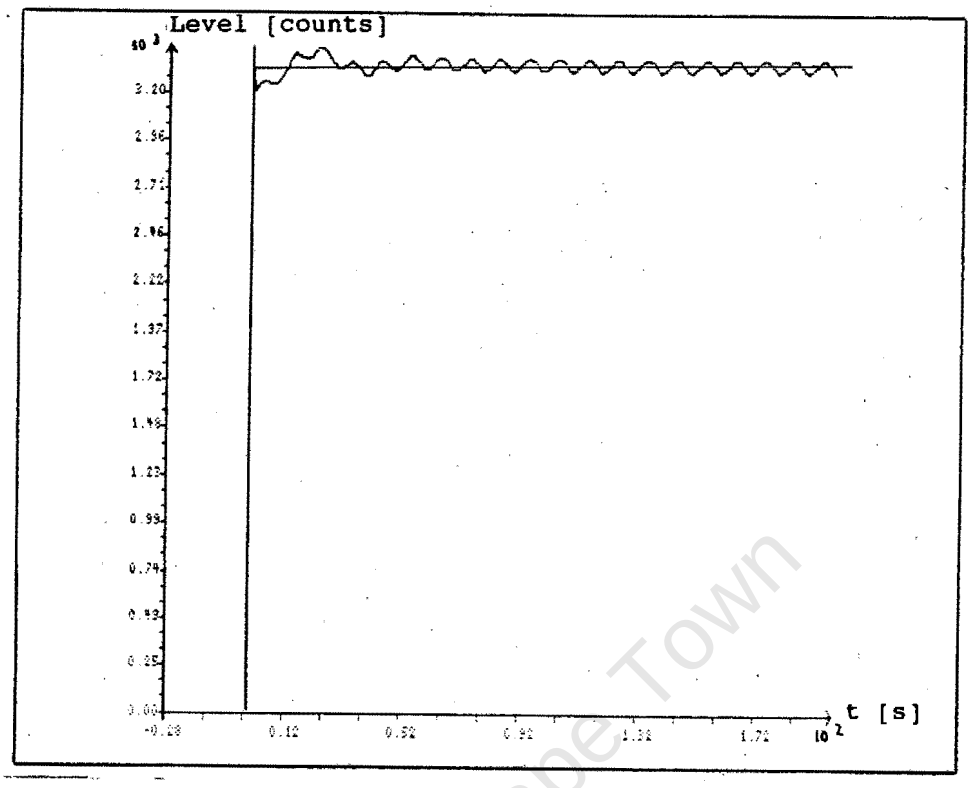
$$\beta = 2 \text{ [counts]} \qquad \ldots(5.28)$$

The gains were not chosen to be symmetric in order to help facilitate the limited input range of 0 - 4095 counts.

The experiment was run under the same initial conditions as experiment 1. The simulated system was run for 200 seconds while the real plant was run for 300 seconds; ensuring a 100 second interval to exhibit all initial conditions.

Figures 5.13(a) and 5.13(b) show the level response to the change in setpoint in simulation and on the flotation plant respectively.

Despite the relatively bad steady state initial conditions exhibited in figure 5.13(b), the system responds much faster than the system of experiment 1. It is estimated that the system reaches its setpoint in about 13 seconds (cf about 70 seconds in experiment 1). Again, this is within the predicted 40 seconds to reach within 2% of its final value.



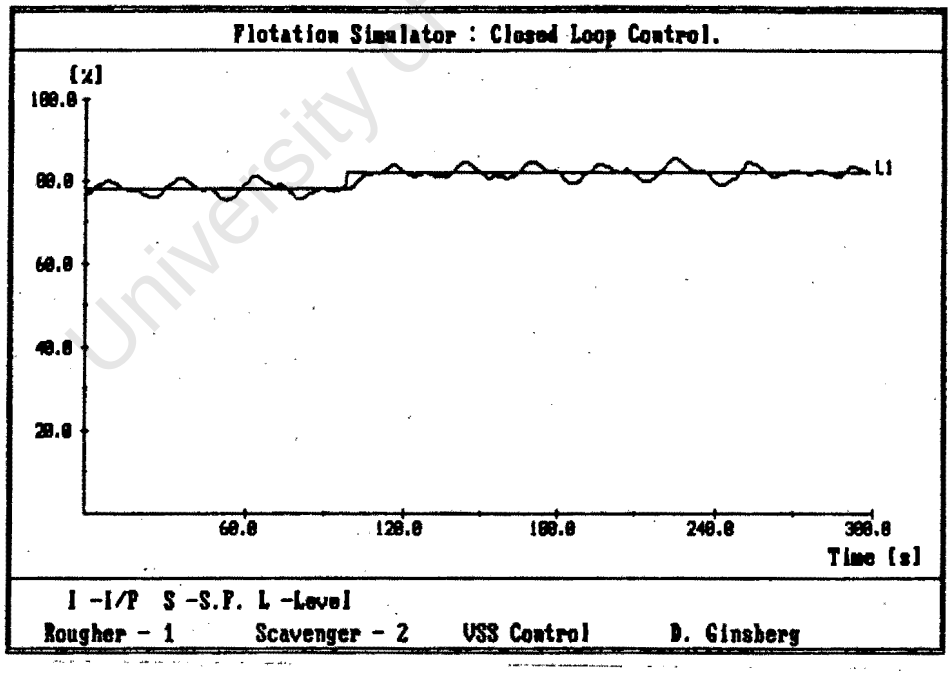


Figure 5.13:System level in (a) simulation and (b) in practice.

The cost of this improved response in terms of speed, is exhibited in figure 5.14 illustrating the system input.

There are a number of important points to note here:

1. The dynamic range of the input signal is about 58 % as compared with the 12% in experiment 1. This is chiefly owing to two factors:

   (a)The control action is cut off by the allowable dynamic range of the input function. This problem can be alleviated by an increased initial valve setting.

   (b)The increased magnitude of the gain constants.

2. Despite poor initial steady state conditions, the input signal starts to diminish towards the end of the simulation.

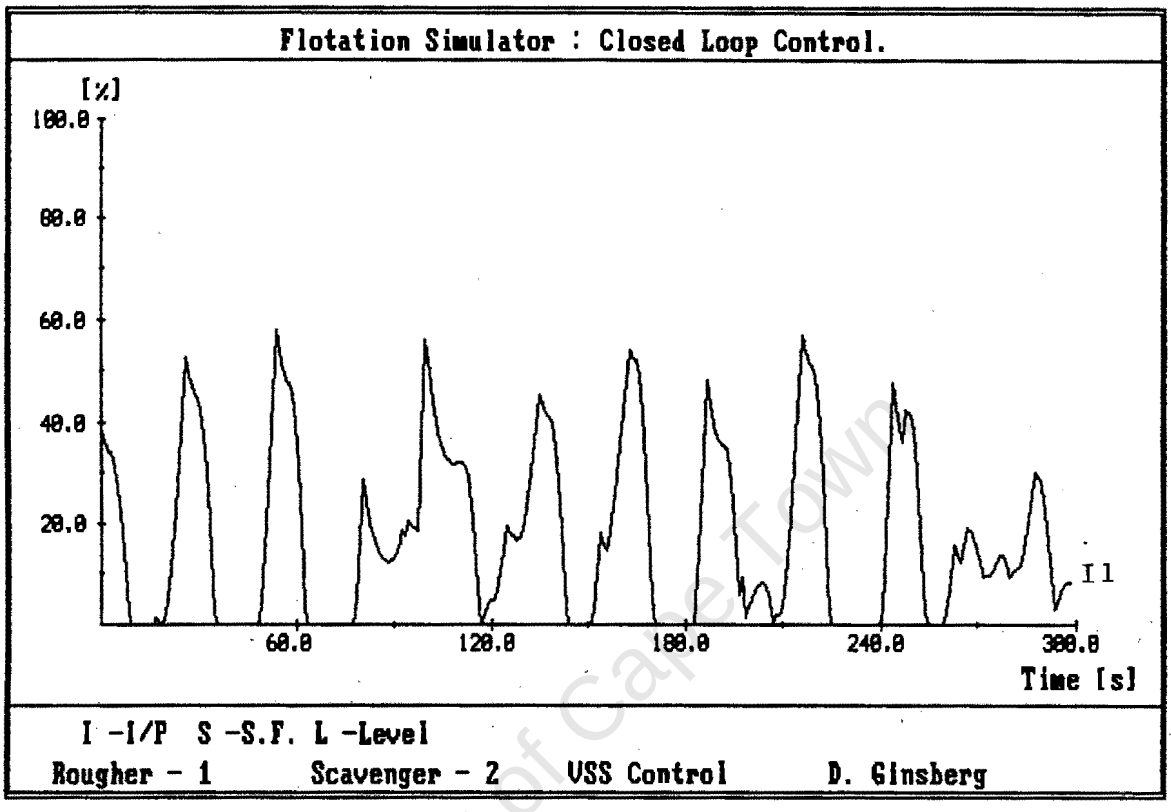


Figure 5.14: System input.

Figure 5.15 illustrates that the switching line does not equal to zero, but has small values which change sign at high frequency. Since the values do not ever appreciably differ from zero, however, it can be deduced that despite all the non - ideal factors present in this experiment, the system does not leave the sliding mode.

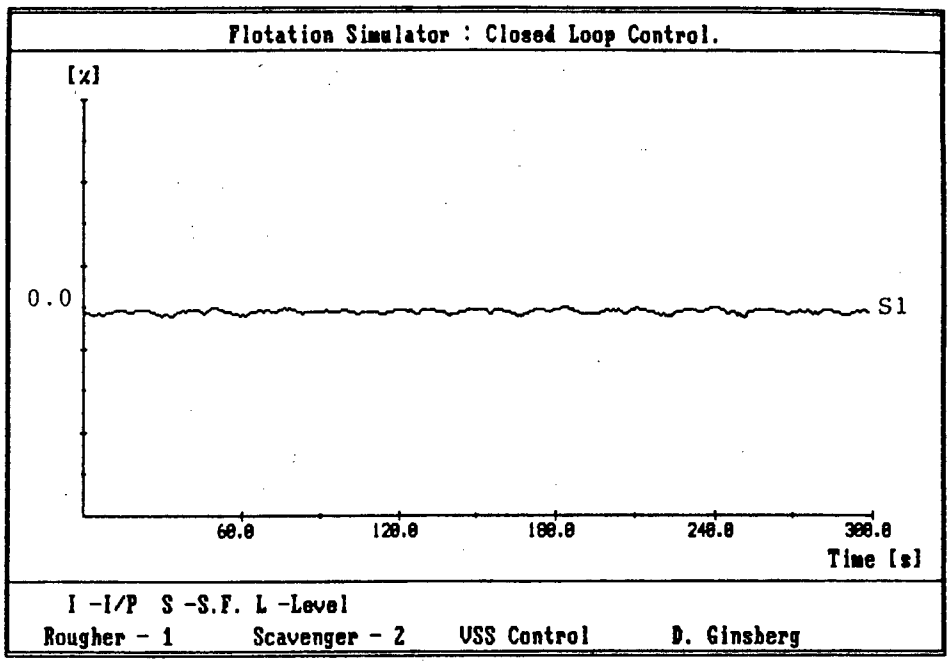


Figure 5.15: System switching line.

The outcome of this experiment shows that in comparison with experiment 1, the system exhibits improved closed loop transient performance in terms of speed, but at the cost of a much larger input function driving the valve and deteriorated steady state characteristics. It must be noted that whilst the initial action of the input function would cover a large dynamic range in comparison with the input in experiment 1, remaining within the dynamic range of the system would facilitate that this large signal would quickly diminish in size. Owing to noise on the states, however, the variations of the input signal will not disappear altogether; as would be the case for a system without noise.

5.4 IMPLEMENTATION OF A MULTIVARIABLE VSS CONTROLLER IN THE FLOTATION RIG SYSTEM

It was seen in the previous section, that whilst the single input variable structure controller provided a good means of controlling the level of the rougher tank, there was no control over the level of the scavenger tank! The interaction of the two tank system makes this a multivariable control problem.

In chapter 4, robust design schemes to implement multivariable variable structure controllers were developed.

One problem with the existing VSS theory, and indeed all state space techniques, is that all controller designs are based purely on the dynamics of the system with total disregard for steady state conditions. Unlike input - output techniques, where the steady state conditions have little bearing on the design of the controller, it is imperative to understand the steady state conditions of a system for the practical implementation of multivariable controllers using VSS theory. This practical problem, is by and large ignored by the existing theory.

For single input systems, implementation of a setpoint was easily facilitated by ensuring that the system was written in phase canonical form and that state one equalled the system output. If error state one was defined as the difference between the setpoint and the output, by differentiating this error state, the remaining error states could be obtained and the system would then switch on these error states.

The implementation of a system setpoint for multivariable processes is not that simple. Since all the present theory on VSS deals exclusively with illustrating (in simulation) how the dynamics of the system go to zero, it in effect ignores the whole issue of implementing a setpoint.

This section will investigate the implementation of practical multivariable VSS controllers and in so doing, look at the two major problem areas outlined above. In addition, a practical state observer design will be investigated.

5.4.1 SYSTEM MODELLING AND VERIFICATION

The general transfer function for a two input, two output system can be written in the form:

$$G(s) = \begin{bmatrix} g_{11}(s) & g_{12}(s) \\ g_{21}(s) & g_{22}(s) \end{bmatrix} \qquad \ldots(5.29)$$

where

$$\mathbf{y}(s)/\mathbf{u}(s) = G(s) \qquad \ldots(5.30)$$

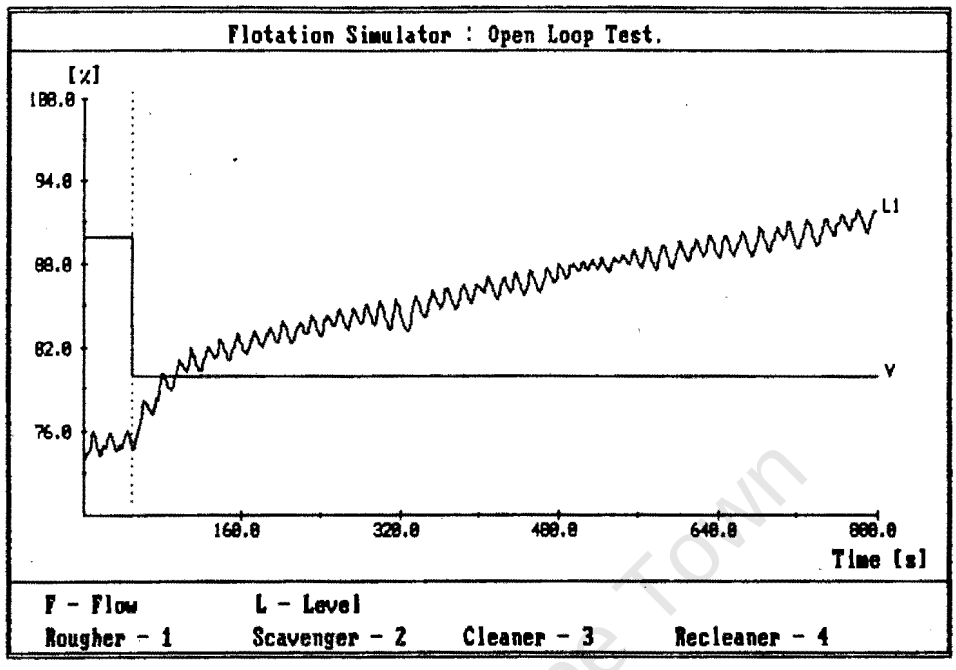
and

$g_{11}(s)$ relates the rougher level to the rougher valve setting.
$g_{21}(s)$ relates the scavenger level to the rougher valve setting.
$g_{12}(s)$ relates the rougher level to the scavenger valve setting.
$g_{22}(s)$ relates the scavenger level to the scavenger valve setting.

$\mathbf{y} = [y_1\ y_2]^T$ = [(rougher level), (scavenger level)]
$\mathbf{u} = [u_1\ u_2]^T$ = [(rougher valve), (scavenger valve)]

Thus, by stepping the rougher valve and recording the rougher and scavenger output levels, $g_{11}(s)$ and $g_{21}(s)$ were obtained. The respective output levels may be seen in figure 5.16.

Similarly, by stepping the scavenger valve and recording the scavenger and rougher output levels, $g_{22}(s)$ and $g_{12}(s)$ were obtained. The results of this step test are illustrated in figure 5.17.



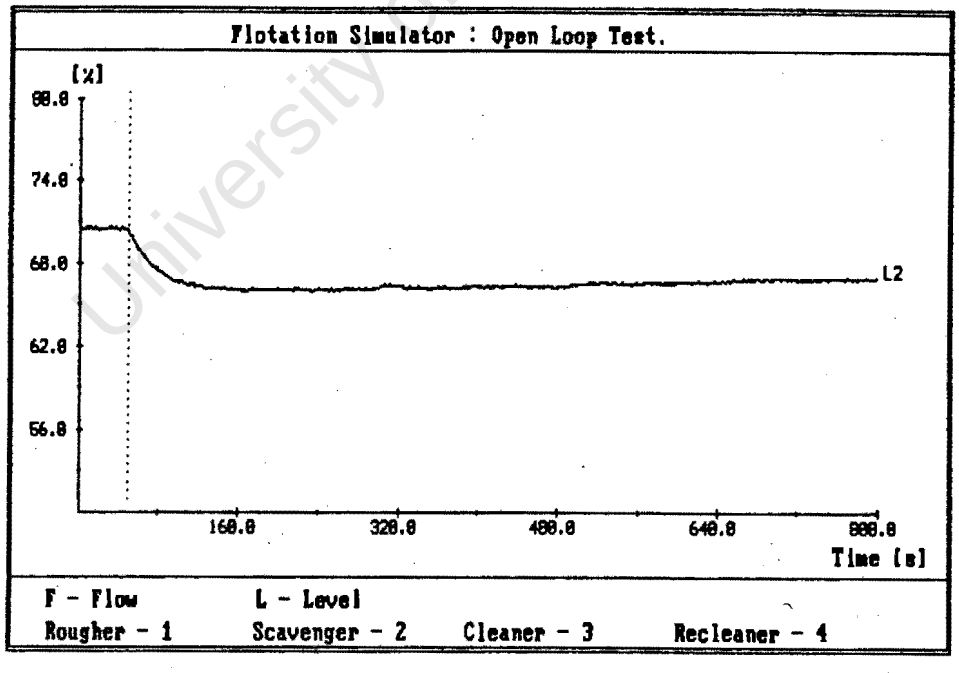
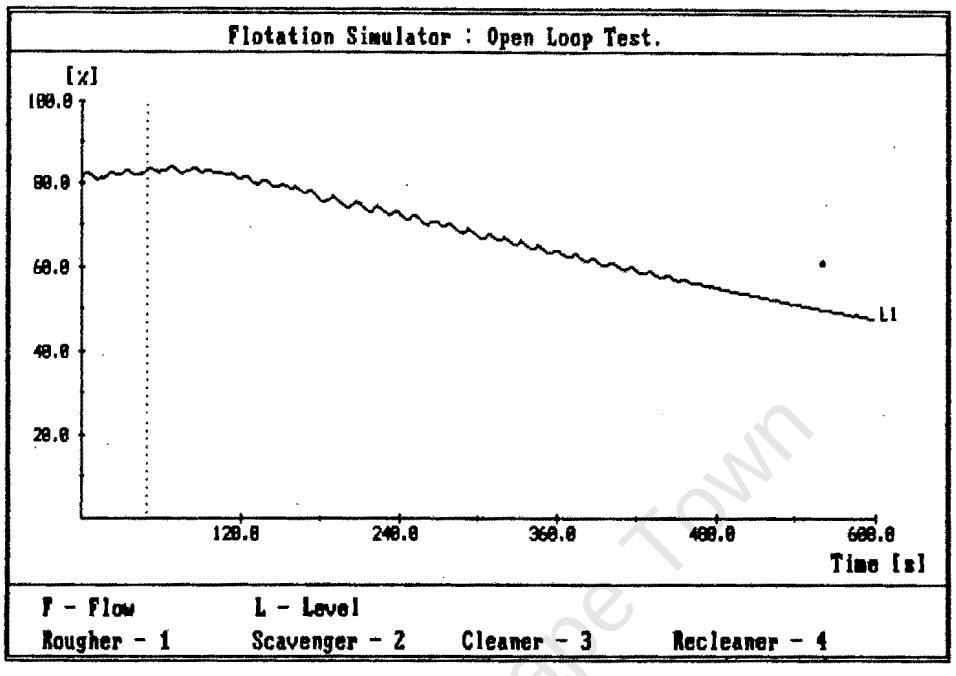


Figure 5.16: (a)Rougher level and (b)Scavenger level for Open loop step of Rougher valve



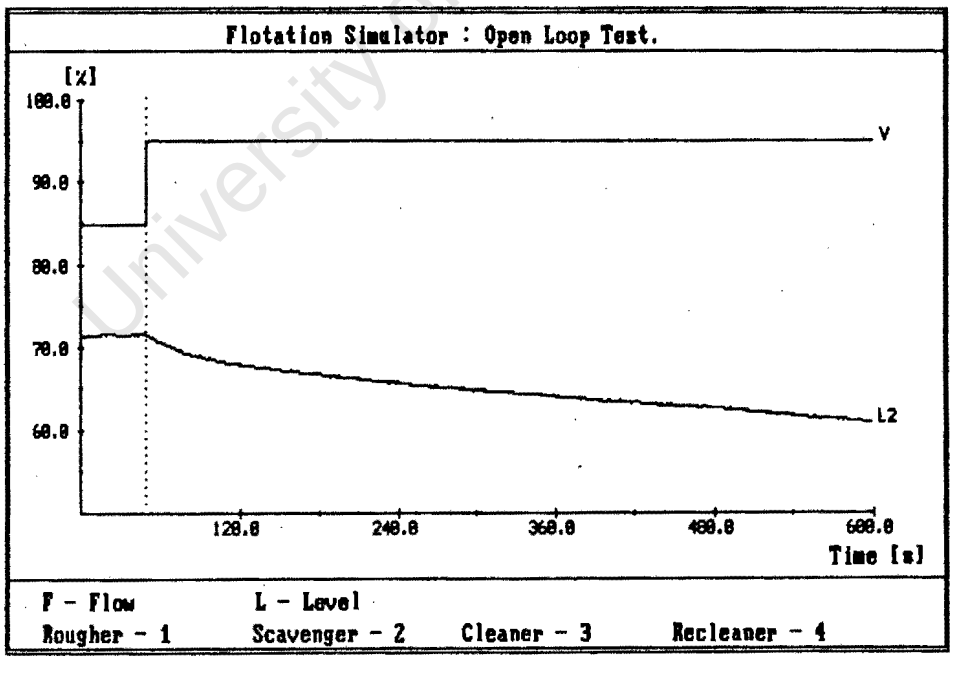


Figure 5.17: (a)Rougher level and (b)Scavenger level for Open loop step of Scavenger valve

From the step tests, it was estimated that:

$$g_{11}(s) = (0.63765*s + 0.0015)/(s + 25.09804*s^{\wedge}2) \quad \ldots(5.31)$$

$$g_{21}(s) = -0.45714/(1 + 28.23529*s) \quad \ldots(5.32)$$

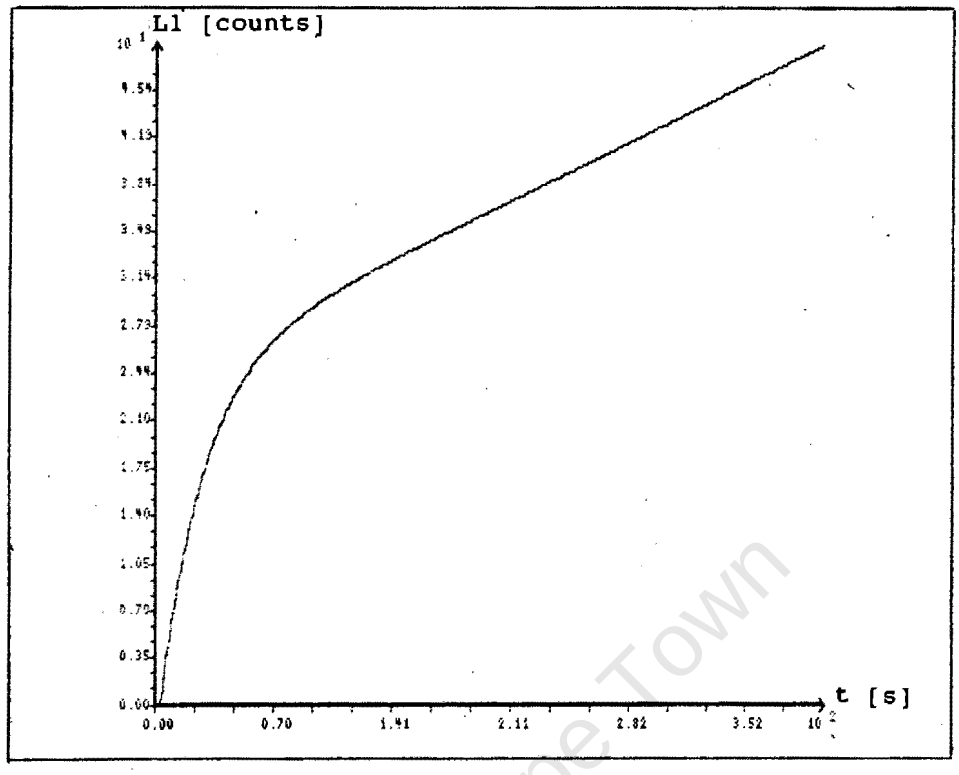
$$g_{12}(s) = 0.0065/s \quad \ldots(5.33)$$

$$g_{22}(s) = (0.35139*s + 0.00156)/(s + 32.94118*s^{\wedge}2) \quad \ldots(5.34)$$

Note that a system dead time was observed in the step tests, but has been ignored. If the controller gain values are big enough, a dead time does not break the sliding mode down; as mentioned in chapter 3.

The individual transfer functions, $g_{ij}(s)$, were converted to equivalent state space models and the step tests repeated in simulation to verify the models.

Figures 5.18 and 5.19 show the results of the open loop simulations,which were run to verify the model. The plots compare well with the actual open loop step tests.



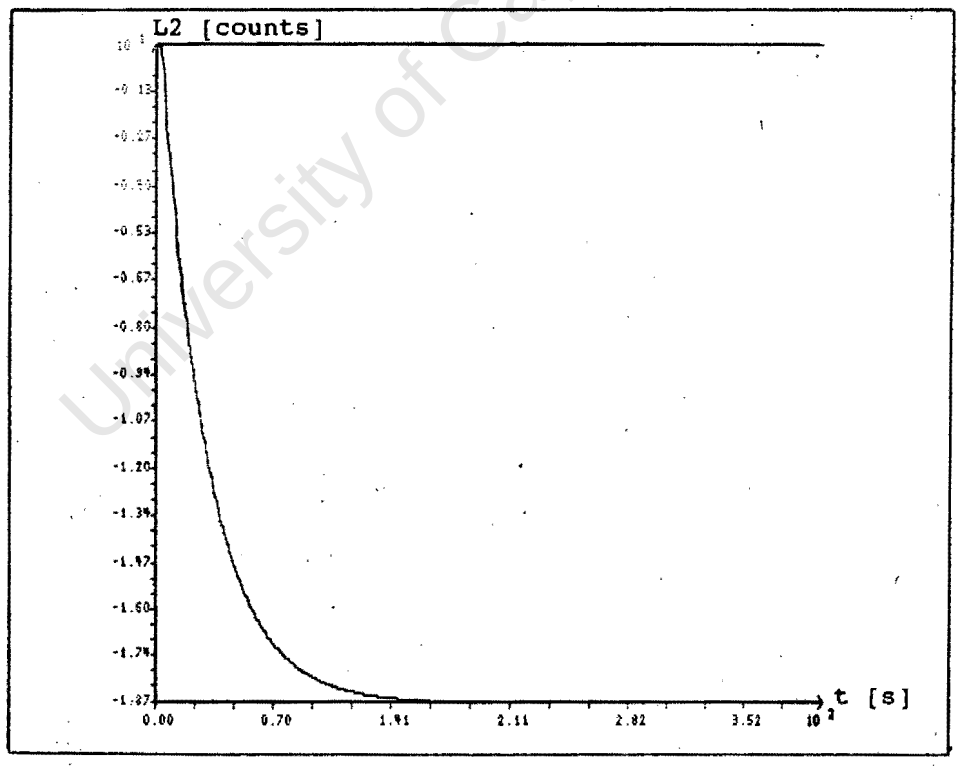
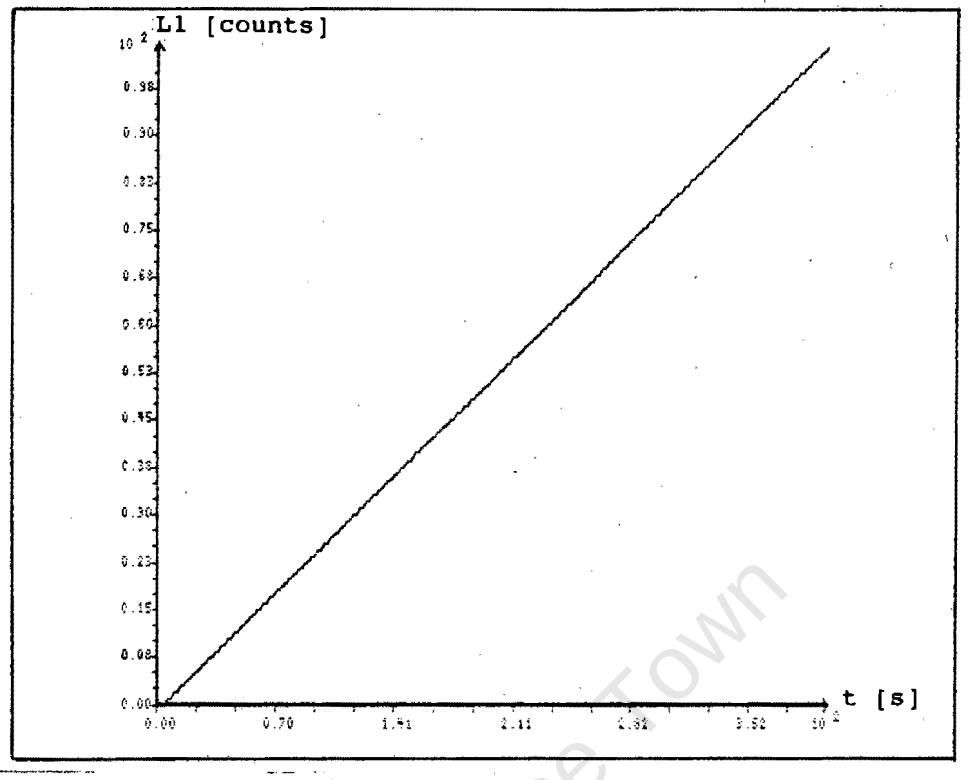


Figure 5.18: (a)Rougher level and (b)Scavenger level in simulation for Open loop step of Rougher valve



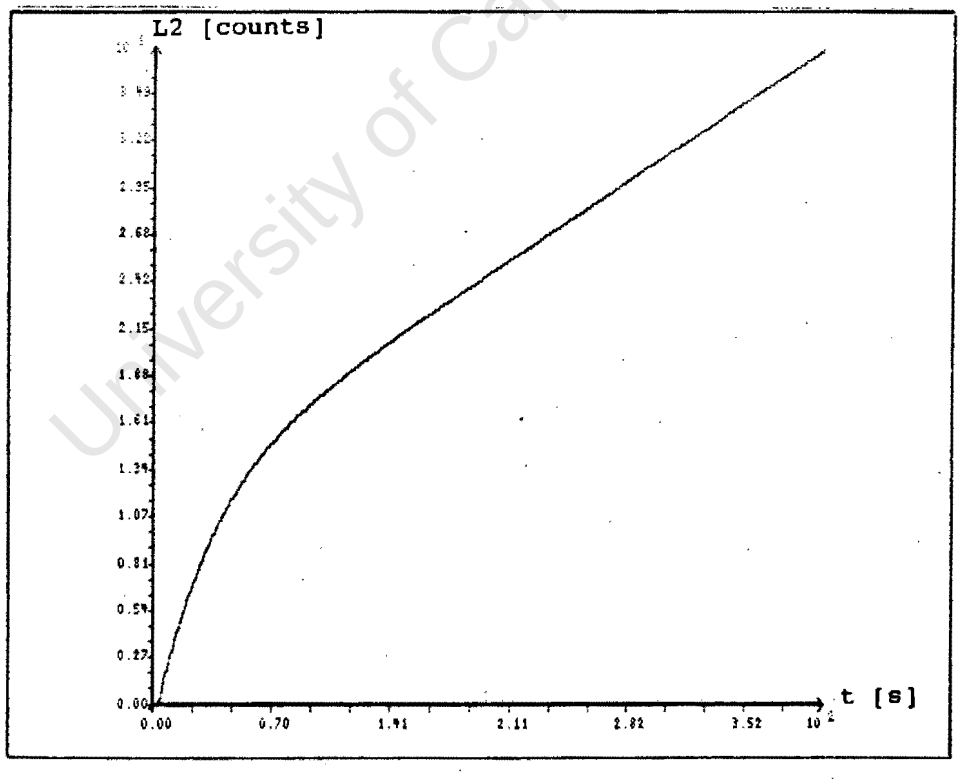


Figure 5.19: (a)Rougher level and (b)Scavenger level in simulation for Open loop step of Scavenger valve

A sixth order state space model was synthesised from the individual state space models.

The model was checked for controllability and state observability (see appendix 6). Unfortunately, both tests failed.

On closer inspection of the system transfer function, it is evident that both $g_{12}(s)$ and $g_{22}(s)$ have a "1/s" term in the denominator; meaning that a redundant state had been defined.

A new fifth order state space model was then defined. The open loop model of the system dynamics is:

$$\mathbf{dx/dt} = \mathbf{Ax} + \mathbf{Bu}$$

$$\mathbf{y} = C_{\mathbf{sys}}\mathbf{x} \qquad \ldots(5.35)$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & -0.03984 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.03542 & 0 \\ 0 & 0 & 0.03036 & 0 & -0.03036 \end{bmatrix} \qquad \ldots(5.36)$$

$$B = \begin{bmatrix} 0 & 0 \\ 0.03984 & 0 \\ 0 & 1 \\ 0.03542 & 0 \\ 0 & 0 \end{bmatrix} \qquad \ldots(5.37)$$

$$C_{sys} = \begin{bmatrix} 0.0015 & 0.63765 & 0.0065 & 0 & 0 \\ 0 & 0 & 0.01067 & -0.45714 & -0.00911 \end{bmatrix} \qquad \ldots(5.38)$$

Now:

$$\text{Rank}[B|AB|A\hat{}2B|A\hat{}3B|A\hat{}4B] = 5 \qquad \ldots(5.39)$$

Thus, the present state space model is completely controllable.

$$\text{Rank}\begin{bmatrix} C_{sys} \\ C_{sys}A \\ C_{sys}A\hat{}2 \\ C_{sys}A\hat{}3 \\ C_{sys}A\hat{}4 \end{bmatrix} = 5 \qquad \ldots(5.40)$$

All the system states are observable.

A state space model has been obtained for the two tank flotation system, which is completely observable and controllable. The model has been verified in simulation.

The model has open loop eigenvalues corresponding to the open loop poles of the transfer function. They are:

$$\begin{aligned} \delta_1 &= 0.0 \\ \delta_2 &= -0.03984 \\ \delta_3 &= 0.0 \\ \delta_4 &= -0.03036 \\ \delta_5 &= -0.03542 \end{aligned} \qquad \ldots(5.41)$$

The system has two unstable eigenvalues at the origin.

It should be noted here that state space techniques provide little or no insight into what happens to the system zeros in closed loop. They are thus, effectively ignored.

5.4.2 DESIGN OF THE HYPERPLANES WHICH DEFINE THE CLOSED LOOP SLIDING MODE

The two tank system is seen to be unstable owing to its two poles at the origin. A multivariable VSS controller is to be designed that firstly stabilises the system. In addition, some desired closed loop behaviour must be achieved.

The speed of the transient response in the sliding mode is defined by the choice of C matrix. Thus, as in chapter 4, three (n-m) closed loop eigenvalues are specified by the choice of the C matrix.

In general, for a 5 state system with two inputs:

$$\sigma_1 = c_{11}*x_1 + c_{12}*x_2 + c_{13}*x_3 + c_{14}*x_4 + c_{15}*x_5 \quad ...(5.42)$$

$$\sigma_2 = c_{21}*x_1 + c_{22}*x_2 + c_{23}*x_3 + c_{24}*x_4 + c_{25}*x_5 \quad ...(5.43)$$

Differentiating equations 5.42 and 5.43 with respect to time and inserting the system equations of the open loop model (equations 5.35 - 5.38) into the new expressions:

$$d\sigma_1/dt = c_{11}*x_2 + c_{12}*a_{22}*x_2 + c_{12}*b_{21}*u_1 + c_{13}*b_{32}*u_2 + c_{14}*a_{44}*x_4 + c_{14}*b_{41}*u_1 + c_{15}*a_{53}*x_3 + c_{15}*a_{55}*x_5 \quad ...(5.44)$$

$$d\sigma_2/dt = c_{21}*x_2 + c_{22}*a_{22}*x_2 + c_{22}*b_{21}*u_1 + c_{23}*b_{32}*u_2 + c_{24}*a_{44}*x_4 + c_{24}*b_{41}*u_1 + c_{25}*a_{53}*x_3 + c_{25}*a_{55}*x_5 \quad ...(5.45)$$

In order to make one of equations 5.44,5.46 dependent on one input only, choose:

$$c_{13} = 0.0 \qquad \ldots(5.46)$$

Since without loss of generality, one of each c_{1j} and c_{2j} (j = 1 to 5) may equal one, for convenience choose:

$$c_{12} = 1.0 \qquad \ldots(5.47)$$

$$c_{25} = 1.0 \qquad \ldots(5.48)$$

These values allow the switching functions to be easily "massaged" into equations 5.44 and 5.45. It is only important that an expression for σ_1 be in equation 5.45 since equation 5.44 is now only dependent on one input.

The switching functions are now:

$$\sigma_1 = c_{11}*x_1 + x_2 + c_{14}*x_4 + c_{15}*x_5 \qquad \ldots(5.49)$$

$$\sigma_2 = c_{21}*x_1 + c_{22}*x_2 + c_{23}*x_3 + c_{24}*x_4 + x_5 \qquad \ldots(5.50)$$

Equations 5.44 and 5.45 become:

$$\begin{aligned} d\sigma_1/dt = {} & (-c_{11}\hat{}2)*x_1 + (a_{22})*x_2 + (c_{15}*a_{53})*x_3 + \\ & (c_{14}*a_{44} - c_{11}*c_{14})*x_4 + (c_{15}*a_{55} - c_{11}*c_{15})*x_5 \\ & + (c_{11})*\sigma_1 + (b_{21} + c_{14}*b_{41})*u_1 \end{aligned} \qquad \ldots(5.51)$$

$$\begin{aligned} d\sigma_2/dt = {} & (-c_{11}*c_{21})*x_1 + (c_{22}*a_{22})*x_2 + (a_{53})*x_3 + \\ & (c_{24}*a_{44} - c_{21}*c_{14})*x_4 + (a_{55} - c_{21}*c_{15})*x_5 \\ & + (c_{21})*\sigma_1 + (c_{22}*b_{21} + c_{24}*b_{41})*u_1 + \\ & (c_{23}*b_{32})*u_2 \end{aligned} \qquad \ldots(5.52)$$

The choice of the remaining c_{ij} values will specify the closed loop eigenvalues.

The method of equivalent control (discussed in chapter 4) is used to choose the C matrix. This is done by choosing the eigenvalues of the system defined by equation 4.15:

$$d\mathbf{x}/dt = [I - B(CB)^{-1}C]A\mathbf{x} \qquad \ldots(5.53)$$

Suitable values for the C matrix were chosen, using Matlab to solve equation 5.53. This placed m eigenvalues at the origin. The remaining (n-m) closed loop eigenvalues (Remember, the sliding mode reduces the system order to (n-m)) were seen to be satisfactory.

Thus, suitable values for the remaining seven elements of the C matrix were in effect chosen by trial and error. In general, this does not constitute a design problem.

The C matrix was chosen (with the aid of Matlab) to be:

$$C = \begin{bmatrix} 0.1 & 0.1 \\ 1.0 & 0.1 \\ 0.0 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 1.0 \end{bmatrix} \qquad \ldots(5.54)$$

Thus, by equation 5.53, the closed loop eigenvalues are:

$$\begin{aligned} \delta_1 &= 0.0 \\ \delta_4 &= 0.0 \\ \delta_2 &= -0.3381 \\ \delta_3 &= -0.083 \\ \delta_5 &= -0.0352 \end{aligned} \qquad \ldots(5.55)$$

m Eigenvalues, δ_1 and δ_4 , are equal to zero, as desired. The remaining (n-m) eigenvalues are all stable.

The open loop system has time constants of:

O/L time constants = 25.1, 28.2 and 32.9 [seconds] ...(5.56)

The closed loop system has time constants of:

C/L time constants = 2.9, 12.04, 28.4 [seconds] ...(5.57)

From equations 5.56 and 5.57, it is evident that the dominant pole of the closed loop system is only marginally faster than the dominant pole of the open loop system. The remaining closed loop poles are much faster (approximately 10 times and twice as fast) than the remaining open loop poles.

Since nothing is known about the position of the system zeros in the s plane, it is not possible to estimate the closed loop response time.

This choice of C matrix ensures a sliding mode to stabilise the system. In addition, superior transient performance (in comparison with the open loop response) in terms of speed is also expected. The extent of the superior speed in transients is at this stage unknown.

5.4.3 CHOICE OF CONTROL FUNCTIONS

The choice of control functions, u_i ($i = 1,2$), has to satisfy existence of a SM conditions, given by equation 4.18:

$$\lim_{\sigma_i \to 0} \sigma_i * d\sigma_i/dt < 0 \quad (i = 1,2) \qquad \ldots(5.58)$$

Thus:

$$\lim_{\sigma_1 \to 0} \sigma_1 * d\sigma_1/dt = (-c_{11}^{\wedge 2}) * x_1 * \sigma_1 + (a_{22}) * x_2 * \sigma_1 + (c_{15} * a_{53}) * x_3 * \sigma_1 + (c_{14} * a_{44} - c_{11} * c_{14}) * x_4 * \sigma_1 + (c_{15} * a_{55} - c_{11} * c_{15}) * x_5 * \sigma_1 + (c_{11}) * \sigma_1^{\wedge 2} + (b_{21} + c_{14} * b_{41}) * u_1 * \sigma_1 < 0 \quad \ldots(5.59)$$

$$\lim_{\sigma_2 \to 0} d\sigma_2/dt = (-c_{11} * c_{21}) * x_1 * \sigma_2 + (c_{22} * a_{22}) * x_2 * \sigma_2 + (a_{53}) * x_3 * \sigma_2 + (c_{24} * a_{44} - c_{21} * c_{14}) * x_4 * \sigma_2 + (a_{55} - c_{21} * c_{15}) * x_5 * \sigma_2 + (c_{21}) * \sigma_1 * \sigma_2 + (c_{22} * b_{21} + c_{24} * b_{41}) * u_1 * \sigma_2 + (c_{23} * b_{32}) * u_2 * \sigma_2 < 0 \quad \ldots(5.60)$$

To satisfy equation 5.58, the control functions must have the form:

$$u_1 = -\Gamma_{11} * x_1 - \Gamma_{12} * x_2 - \Gamma_{13} * x_3 - \Gamma_{14} * x_4 - \Gamma_{15} * x_5 \quad \ldots(5.61)$$

$$u_2 = -\Gamma_{21} * x_1 - \Gamma_{22} * x_2 - \Gamma_{23} * x_3 - \Gamma_{24} * x_4 - \Gamma_{25} * x_5 - \Gamma_2 * \sigma_1 \quad \ldots(5.62)$$

To ensure the existence of a sliding mode (by equations 5.59 and 5.60):

$$\Gamma_{11} = \alpha_{11} > -0.23052 \text{ if } x_1 * \sigma_1 > 0$$
$$= \beta_{11} < -0.23052 \text{ if } x_1 * \sigma_1 < 0 \quad \ldots(5.63)$$

$$\Gamma_{12} = \alpha_{12} > -0.91840 \text{ if } x_2 * \sigma_1 > 0$$
$$= \beta_{12} < -0.91840 \text{ if } x_2 * \sigma_1 < 0 \quad \ldots(5.64)$$

$$\Gamma_{13} = \alpha_{13} > 0.07008 \text{ if } x_3 * \sigma_1 > 0$$
$$= \beta_{13} < 0.07008 \text{ if } x_3 * \sigma_1 < 0 \quad \ldots(5.65)$$

$$\Gamma_{14} = \alpha_{14} > -0.31213 \text{ if } x_4 * \sigma_1 > 0$$
$$= \beta_{14} < -0.31213 \text{ if } x_4 * \sigma_1 < 0 \quad \ldots(5.66)$$

$$\Gamma_{15} = \alpha_{15} > -0.30060 \text{ if } x_5*\sigma_1 > 0$$
$$= \beta_{15} < -0.30060 \text{ if } x_5*\sigma_1 < 0 \quad \ldots(5.67)$$

$$\Gamma_{21} = \alpha_{21} > -0.10000 - 0.07530*\Gamma_{11} \text{ if } x_1*\sigma_2 > 0$$
$$= \beta_{21} < -0.10000 - 0.07530*\Gamma_{11} \text{ if } x_1*\sigma_2 < 0 \quad \ldots(5.68)$$

$$\Gamma_{22} = \alpha_{22} > -0.03980 - 0.07530*\Gamma_{12} \text{ if } x_2*\sigma_2 > 0$$
$$= \beta_{22} < -0.03980 - 0.07530*\Gamma_{12} \text{ if } x_2*\sigma_2 < 0 \quad \ldots(5.69)$$

$$\Gamma_{23} = \alpha_{23} > 0.30360 - 0.07530*\Gamma_{13} \text{ if } x_3*\sigma_2 > 0$$
$$= \beta_{23} < 0.30360 - 0.07530*\Gamma_{13} \text{ if } x_3*\sigma_2 < 0 \quad \ldots(5.70)$$

$$\Gamma_{24} = \alpha_{24} > -0.13540 - 0.07530*\Gamma_{14} \text{ if } x_4*\sigma_2 > 0$$
$$= \beta_{24} < -0.13540 - 0.07530*\Gamma_{14} \text{ if } x_4*\sigma_2 < 0 \quad \ldots(5.71)$$

$$\Gamma_{25} = \alpha_{25} > -0.40360 - 0.07530*\Gamma_{15} \text{ if } x_5*\sigma_2 > 0$$
$$= \beta_{25} < -0.40360 - 0.07530*\Gamma_{15} \text{ if } x_5*\sigma_2 < 0 \quad \ldots(5.72)$$

$$\Gamma_2 = \alpha_2 > 1.00000 \text{ if } \sigma_1*\sigma_2 > 0$$
$$= \beta_2 < 1.00000 \text{ if } \sigma_1*\sigma_2 < 0 \quad \ldots(5.73)$$

With this control arrangement, u_1 drives the RP to σ_1. The other hyperplane, σ_2, is reached by the action of u_2. Note that the gains of u_2 directly take into account u_1 and σ_1, ensuring that there is no fight for control between the two input signals.

One practical design question, which is difficult to answer, is to know how much bigger or smaller to make the specific gain values to satisfy the conditions above.

Making the specific gain values too small is clearly undesirable. Practical systems are sampled at a finite frequency, have nonlinear regions and noise is ever present. These are a few areas of concern which could cause a loss of sliding.

Making the specific gain values too large also leads to problems. There is a limited range within which the control functions, u_1 and u_2, have to operate. Other problems include loss of sliding and the RP cycling in the phase plane.

The specific gain values chosen to satisfy equations 5.63 - 5.73 are:

$\alpha_{11} = 0.35$	$\beta_{11} = -0.80$
$\alpha_{12} = 1.60$	$\beta_{12} = -3.50$
$\alpha_{13} = 0.30$	$\beta_{13} = -0.20$
$\alpha_{14} = 0.60$	$\beta_{14} = -1.20$
$\alpha_{15} = 0.60$	$\beta_{15} = -1.20$
$\alpha_{21} = 0.10$	$\beta_{21} = -0.50$
$\alpha_{22} = 0.805$	$\beta_{22} = -0.60$
$\alpha_{23} = 0.90$	$\beta_{23} = -0.40$
$\alpha_{24} = 0.80$	$\beta_{24} = -0.80$
$\alpha_{25} = 0.60$	$\beta_{25} = -1.60$
$\alpha_2 = 4.00$	$\beta_2 = -3.00$

...(5.74)

5.4.4 THE DESIGN OF A STATE OBSERVER

In general, the states of a system are not available for measurement and thus a state estimator needs to be incorporated into the system.

The state estimator is essentially an internal model of the system which runs in parallel with it.

There are a number of popular state estimators, such as the asymptotic state estimator and the Kalman filter (see [1]). Essentially, the mechanisms are the same; a gain matrix is defined which determines the speed that the estimated states converge to the values of the actual states.

It was decided to use an asymptotic state estimator.

For a system whose dynamics are described by equation 5.35, the estimated states for the sampled data version of this system (the continuous system "viewed" through D.A.C.s and A.D.C.s) are:

$$\mathbf{x_{est}}(n+1) = (\varnothing - MC_{sys})\mathbf{x_{est}}(n) + \Phi\mathbf{u}(n) + M\mathbf{y}(n) \qquad \ldots(5.75)$$

where

$\mathbf{x_{est}}(n+1)$ is the new estimated state vector.
$\mathbf{x_{est}}(n)$ is the previously estimated state vector.
M is the gain matrix.
$\varnothing = I + AT$ (for small sampling time T).
$\Phi = TB$ (for small sampling time T).

The estimator error behaves according to the relationship:

$$(\mathbf{x}(n+1) - \mathbf{x_{est}}(n+1)) = (\varnothing - MC_{sys})(\mathbf{x}(n) - \mathbf{x_{est}}(n)) \qquad \ldots(5.76)$$

Thus, the design problem is to find an M matrix which will ensure that the matrix, ($\varnothing$ - MCsys), has eigenvalues which correspond to a suitably fast decay of any estimated error.

A number of methods, which are aimed at finding the M matrix for specified eigenvalues of ($\varnothing - MC_{sys}$), were investigated. Usually the method involves solving a parallel feedback problem and choosing the feedback constants to satisfy the desired choice of eigenvalues. Specific ways to choose the feedback constants are shown in [1] and [2].

It was found that the easiest way to obtain a suitable gain matrix, was simply to make an educated guess at a particular M matrix and then check that it produced suitable eigenvalues.

The gain matrix was chosen to be:

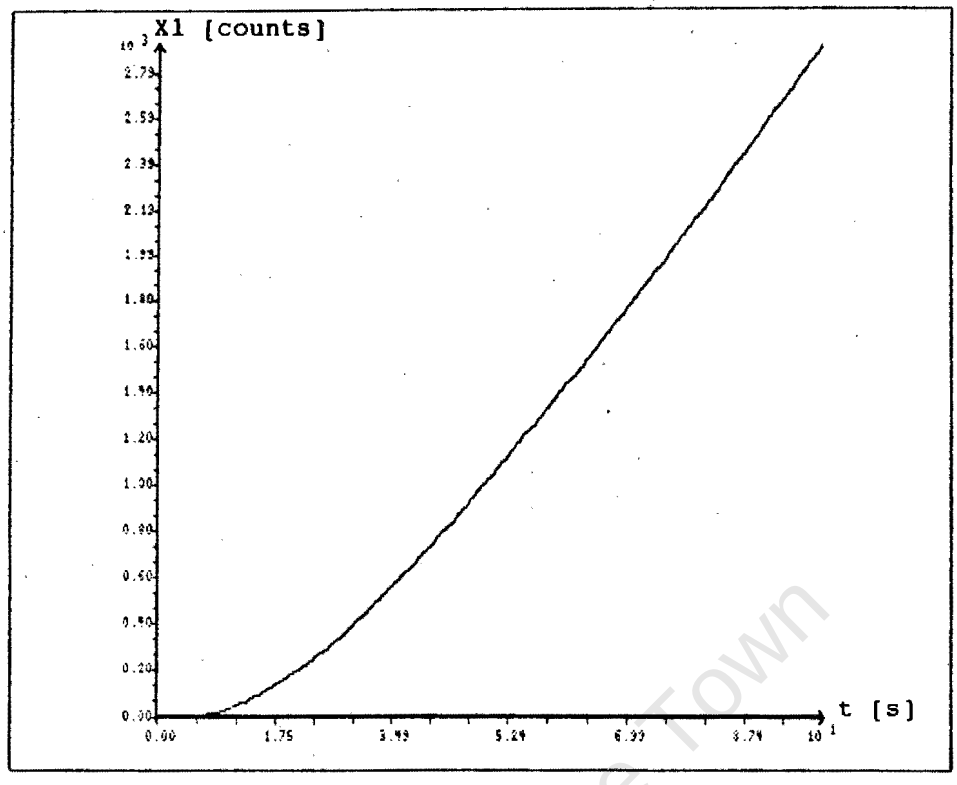
$$M = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \ldots(5.77)$$

The eigenvalues of the matrix (ø - MCsys), were found to be, using Matlab:

$$\begin{aligned} \delta_1 &= -0.0023 \\ \delta_2 &= -0.6767 \\ \delta_3 &= -0.0016 \\ \delta_4 &= -0.0304 \\ \delta_5 &= -0.0354 \end{aligned} \qquad \ldots(5.78)$$

Based on these values, the estimator is expected to provide satisfactory results since the eigenvalues are in general faster than those of the corresponding closed loop system (given in equation 5.55). In addition, their associated time constants are all slower than the system sampling time of one second.

Figures 5.20 to 5.24 illustrate plots of the actual states and the estimated states in open loop.



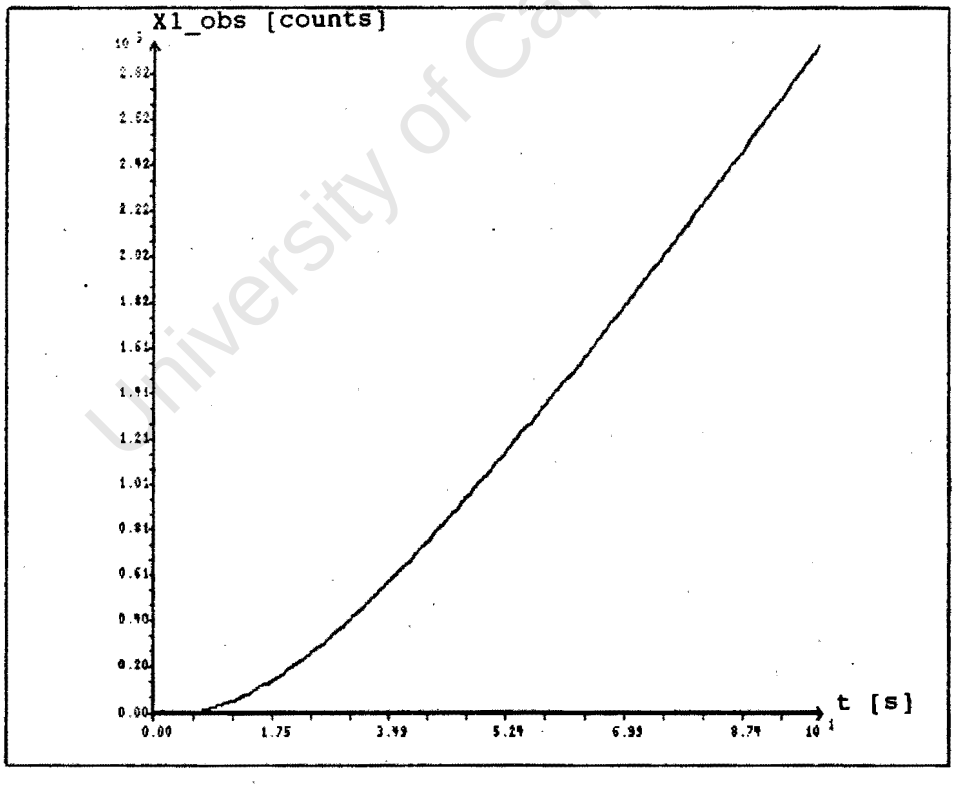
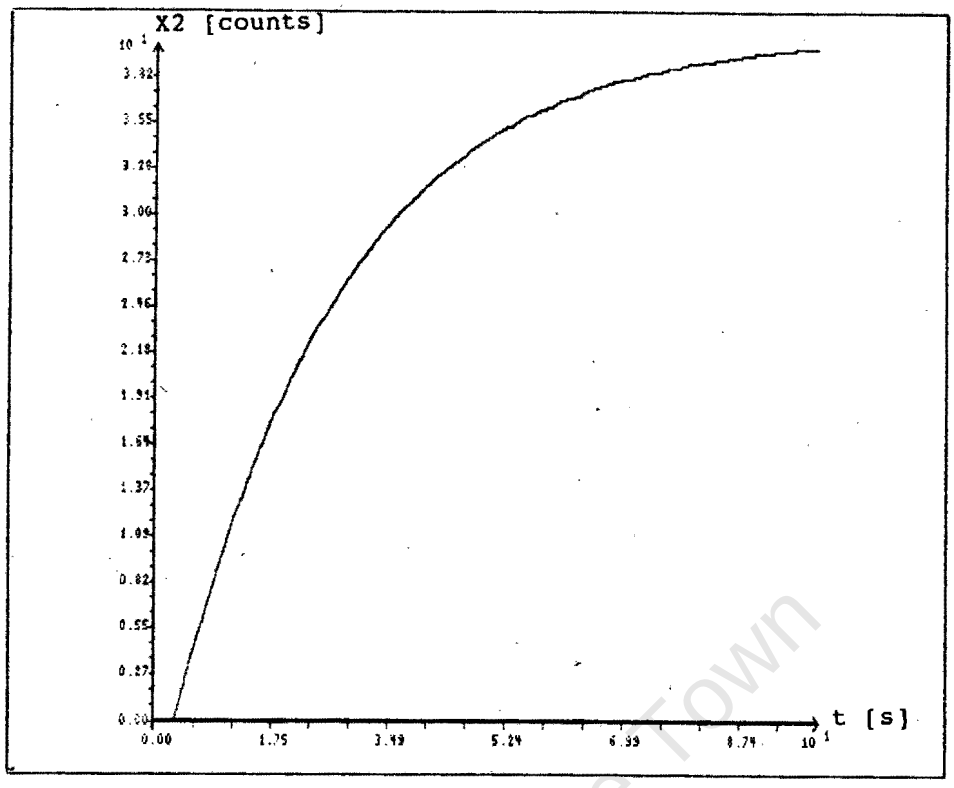


Figure 5.20: (a)Actual state 1 and (b)Estimated state 1.



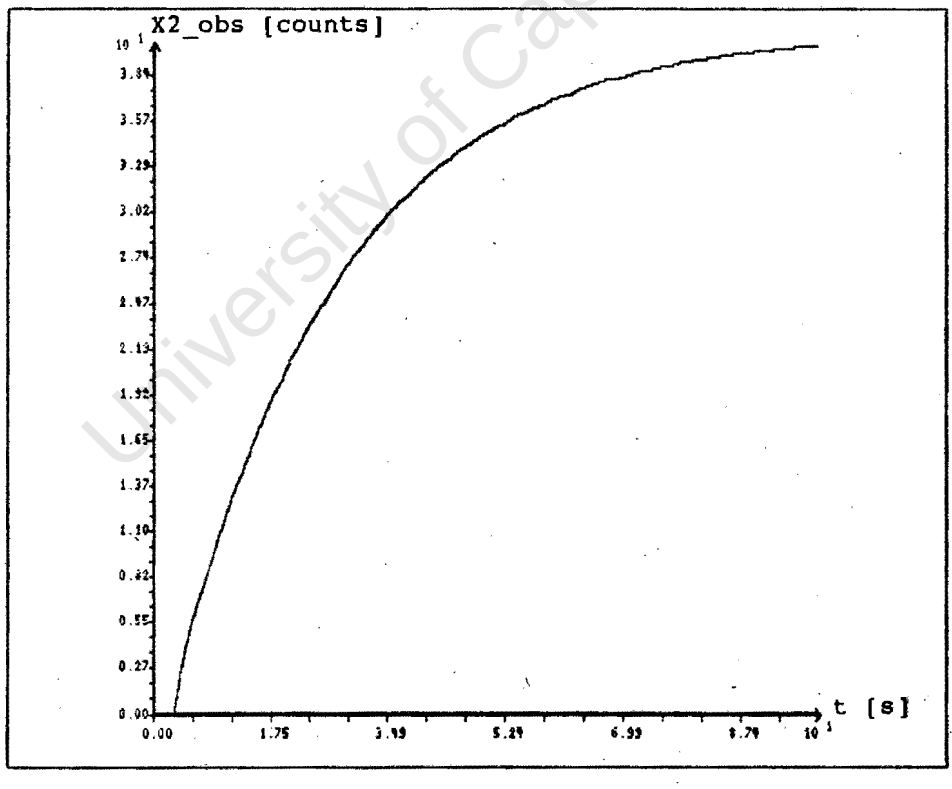
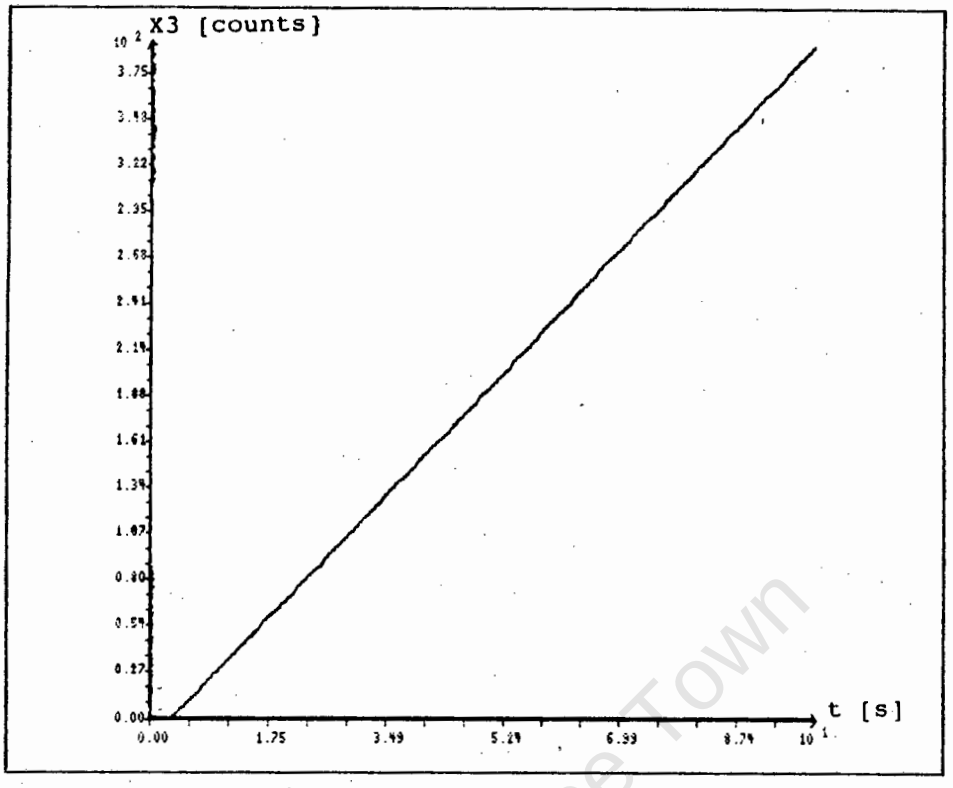


Figure 5.21: (a)Actual state 2 and (b)Estimated state 2.



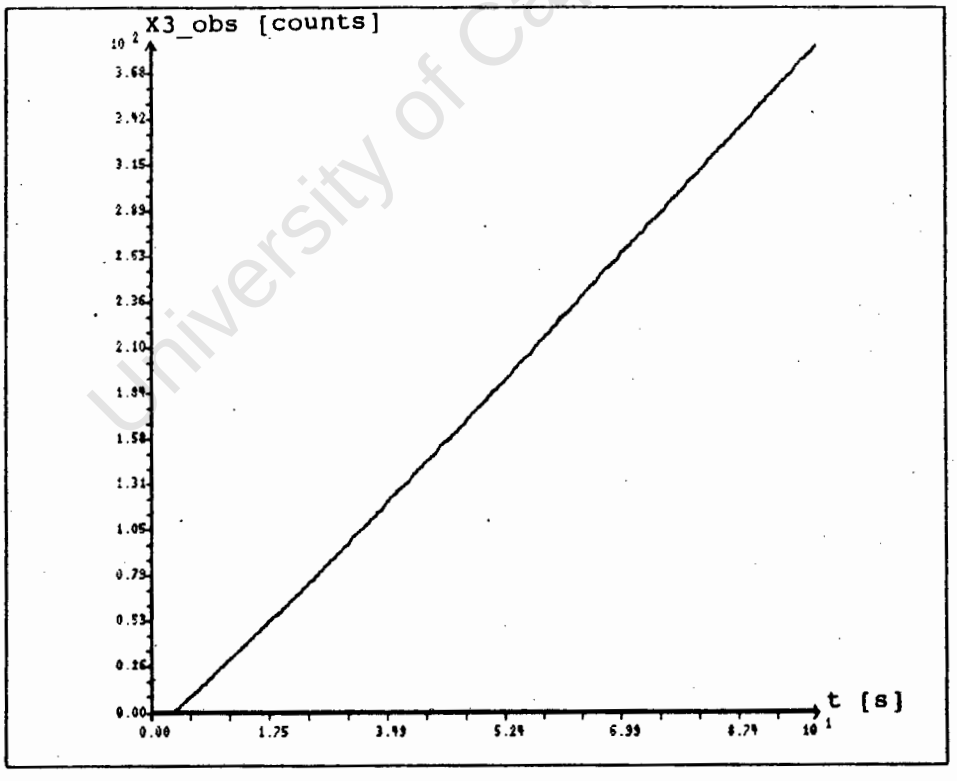
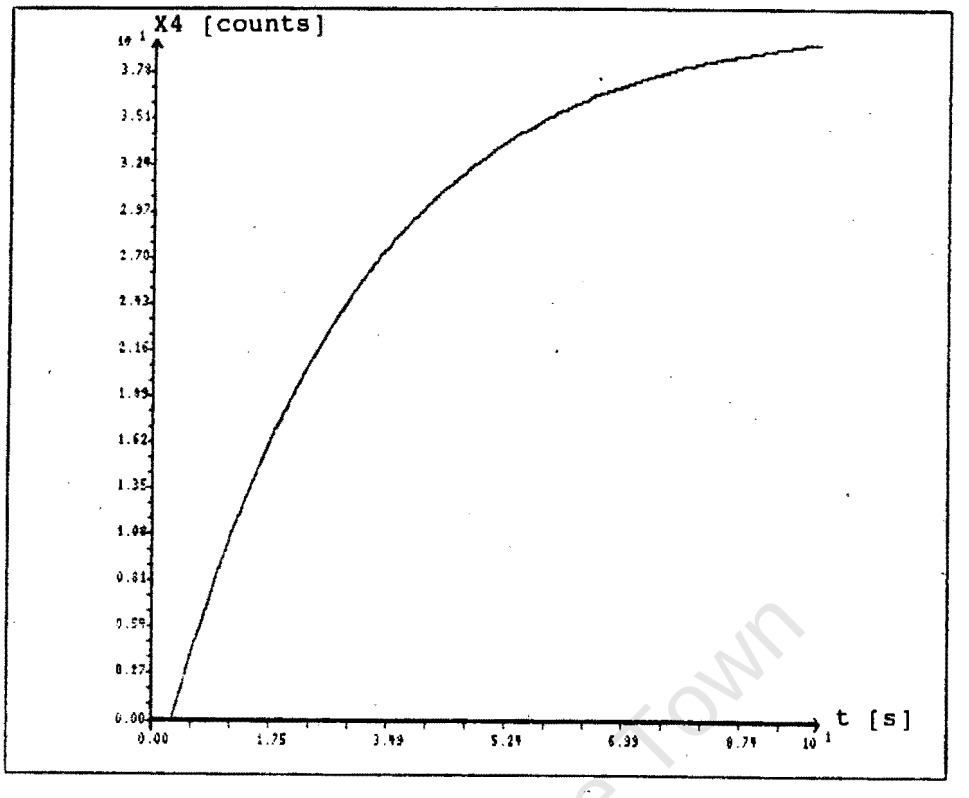


Figure 5.22: (a)Actual state 3 and (b)Estimated state 3.



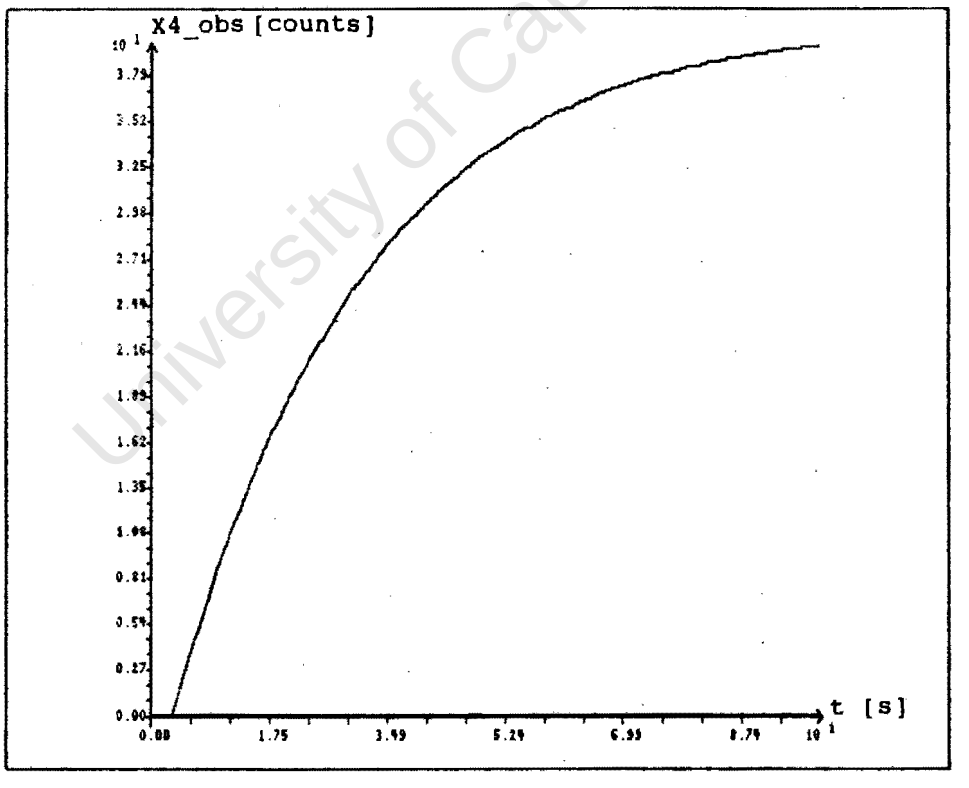


Figure 5.23: (a)Actual state 4 and (b)Estimated state 4.

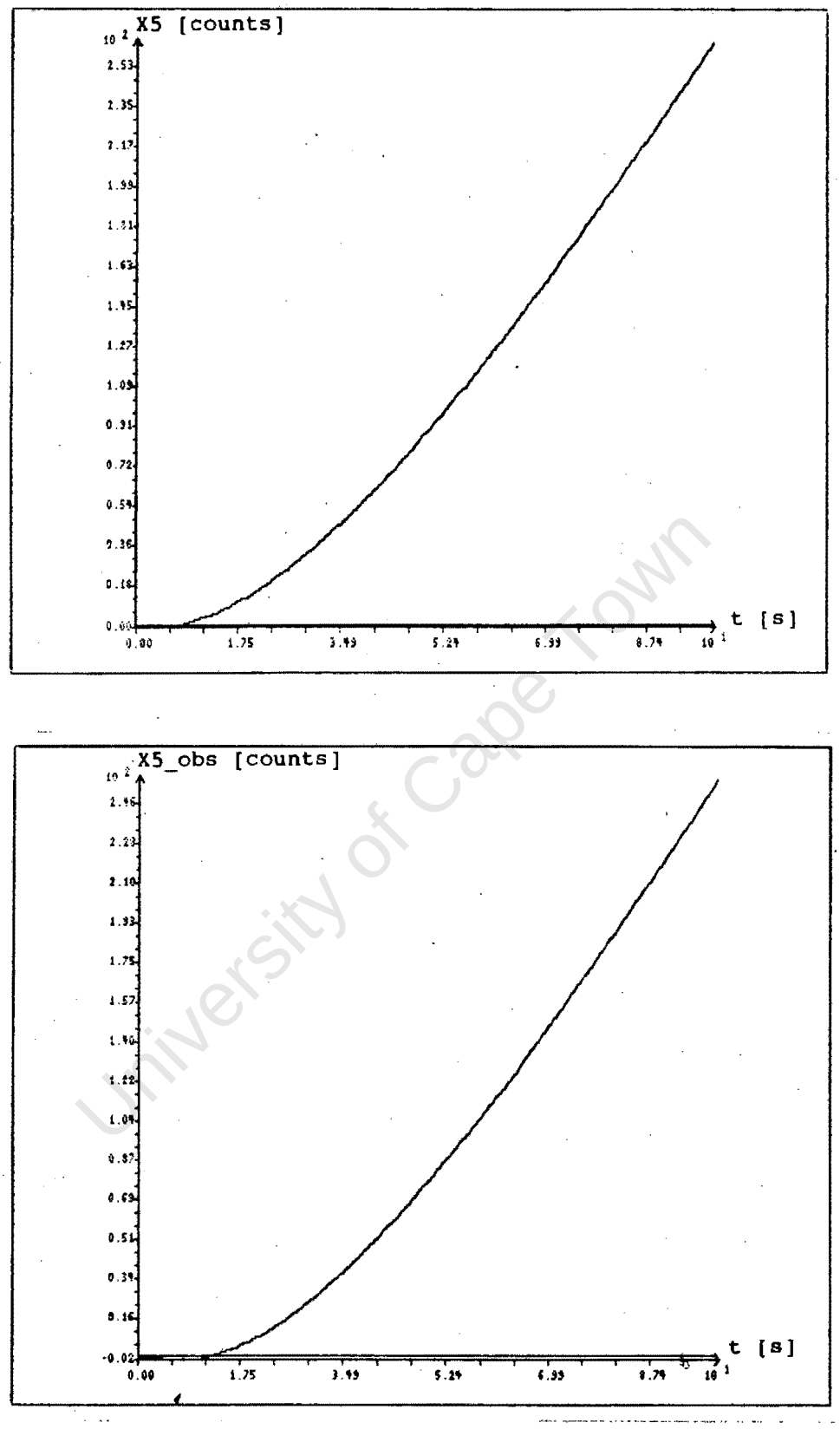


Figure 5.24: (a)Actual state 5 and (b)Estimated state 5.

5.4.5 IMPLEMENTATION OF A SYSTEM SETPOINT

At this stage, it is imperative to review the action taken by the variable structure controller (VSC).

The control functions are a linear combination of the system states. The VSC drives all the system states to zero and in so doing, the control functions are also driven to zero.

It is clear that such a scheme is really suited to systems which operate about some zero point (with zero steady state characteristics).

The control of a system such as the Flotation plant, clearly needs to operate about some non - zero input valve settings and output levels.

One way to overcome this problem, is to make a transformation of states by differentiating the system states. The VSC is based on these transformed states. The variable structure control functions are then integrated before being applied to the valves.

With this scheme, the dynamics of the system are driven to zero, but the system can operate at non - zero operating levels. Any disturbance or plant parameter change will thus be rejected by the control scheme; which also serves to stabilise the plant in question.

This transformation of states is illustrated in figure 5.25.

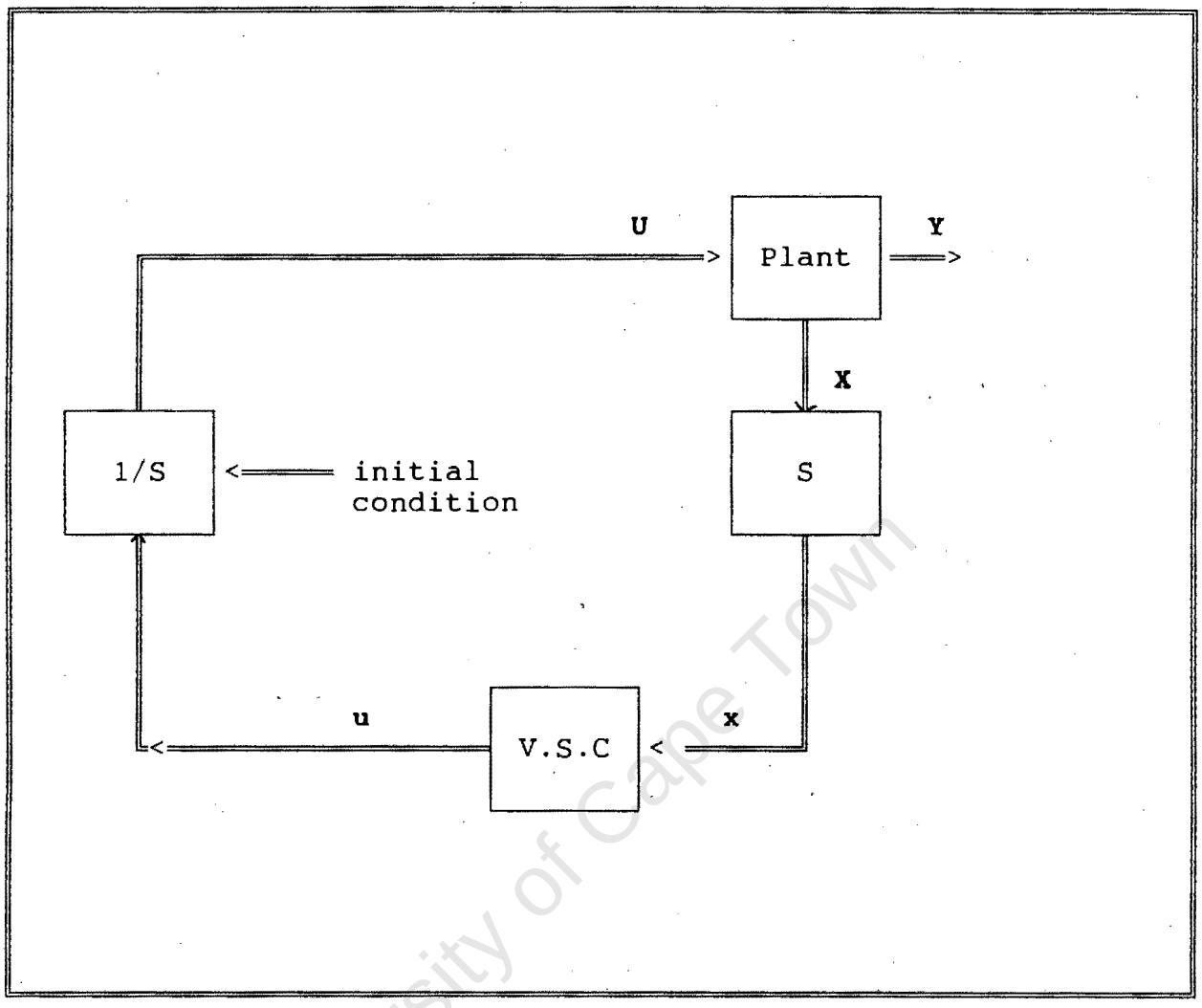


Figure 5.25:Transformation of states to implement VSC.

The transformation of states strategy was used to make the real plant fit in with the existing theory, which is exclusively based on the dynamic state space equations. Existing theory on VSS, however, does not include a general strategy to implement a setpoint or some external signal to the system.

The variable structure control functions are a linear combination of the system states. It must be possible for the system to be written in the form:

$$d\mathbf{x}/dt = A_{eq}\mathbf{x} \qquad \ldots(5.79)$$

An external signal would thus be treated as noise.

Various schemes were considered to overcome this problem.

One scheme is based on a transformation to make each output exclusively equal to one state. The system A matrix is also transformed to ensure that if error states are defined to be the difference between the system setpoints and states which equal the outputs, the remaining states can be obtained by differentiating the error states. This is an attempt to expand the setpoint strategy used in single input systems.

The transformations can be made using the transformation strategy of appendix 2. One problem to consider, is that the transformations usually fill the system B matrix. The control method adopted removes the influence of one input (in the two input case) from the differential of a specific switching function. This would mean that an unwanted number of c_{ij} elements would be set to zero.

The novel scheme adopted for implementing a system setpoint is outlined here. This method of setpoint implementation has not been rigorously proved, but has been successful in practice.

The fact that the state space model and its observer are based entirely on system dynamics, means that the estimated states are based on the dynamic changes of the input and output functions.

This means that in reality, the dynamic changes of the estimated states are obtained by (from equation 5.75):

$$\mathbf{x_{est}}(n+1) = (\varnothing - MC_{sys})\mathbf{x_{est}}(n) + \Phi(\mathbf{U}(n) - \mathbf{U_{init}}) + M(\mathbf{Y}(n) - \mathbf{Y_{init}}) \quad \ldots(5.80)$$

where

$\mathbf{U_{init}}$ are the initial valve settings.

$\mathbf{Y_{init}}$ are the initial output levels.

The dynamics of the states only change if the current input and output values change from their initial settings.

Setpoints for the output levels are facilitated by replacing $\mathbf{Y_{init}}$ with $\mathbf{r}$.

The initial values of the estimated states can be effectively ignored, since the states are differentiated to facilitate non - zero operating regions.

The entire control scheme can be seen in figure 5.26.

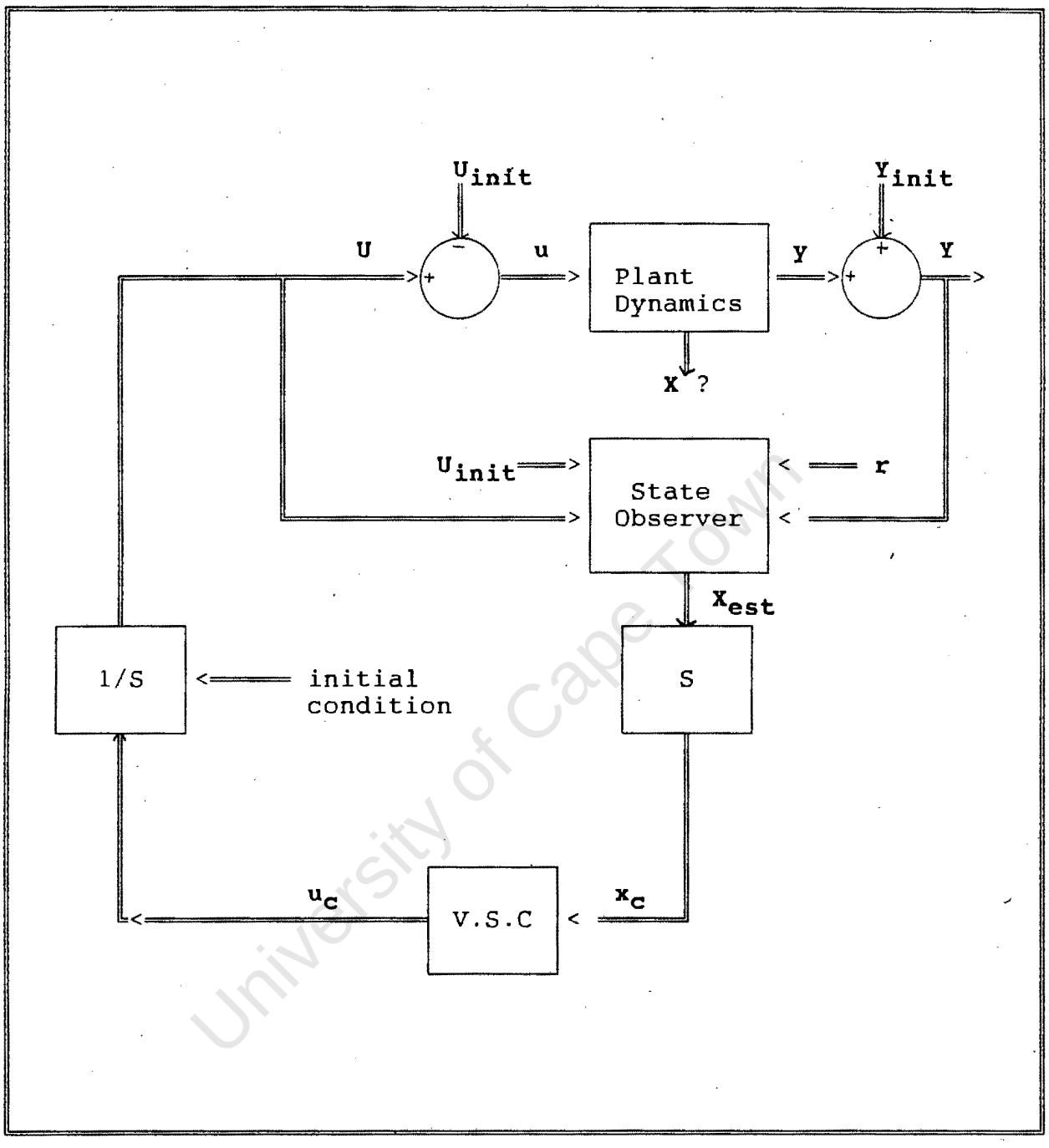


Figure 5.26: Control scheme.

5.4.6 RESULTS

In this section, the results of the outlined control schemes are illustrated.

EXPERIMENT 1:

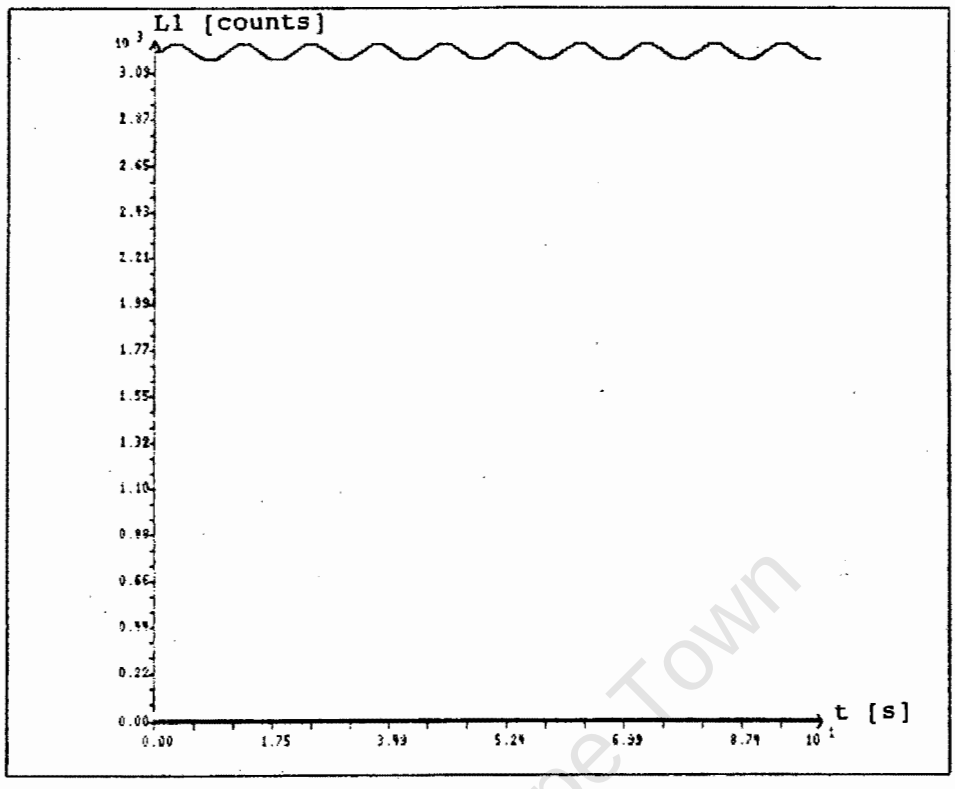
The aim of this experiment, is to implement the multivariable VSC which was designed for the flotation rig.

The control strategy is based on the one outlined in figure 5.25 (including a state observer).

It is expected that this controller will stabilise the system, reject disturbances and be invariant to model parameter changes.

Figure 5.27 illustrates the rougher and scavenger output levels in simulation. The noise function, described by equation 5.23, has been added to the rougher output.

Figure 5.28 illustrates the associated input functions.



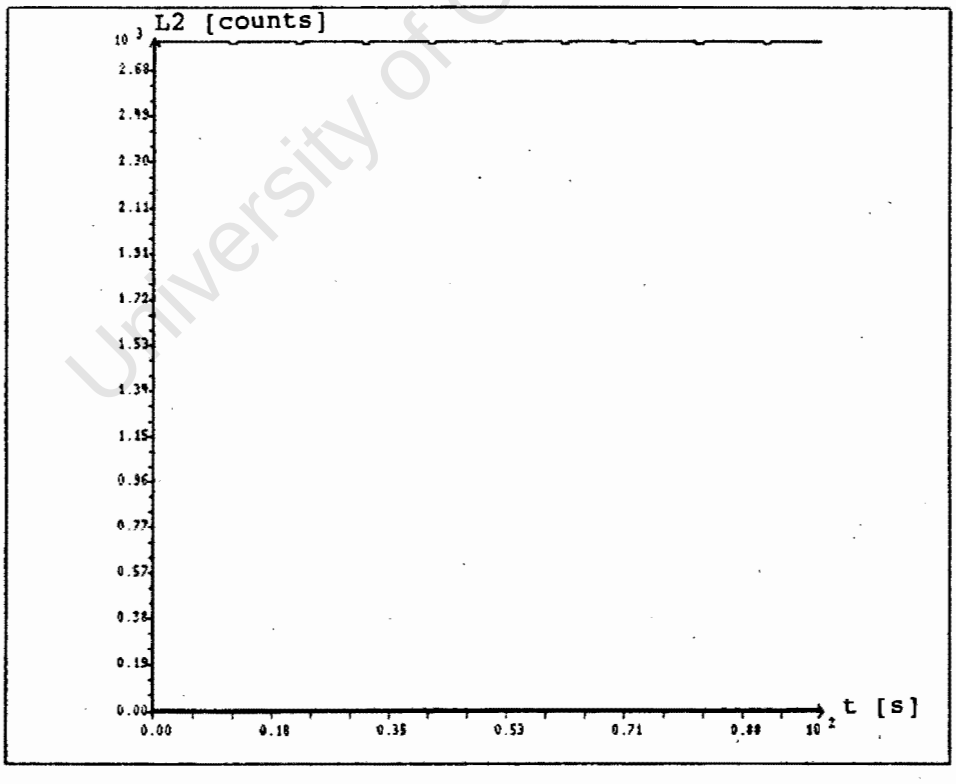


Figure 5.27: (a)Rougher level and (b)Scavenger level vs Time.

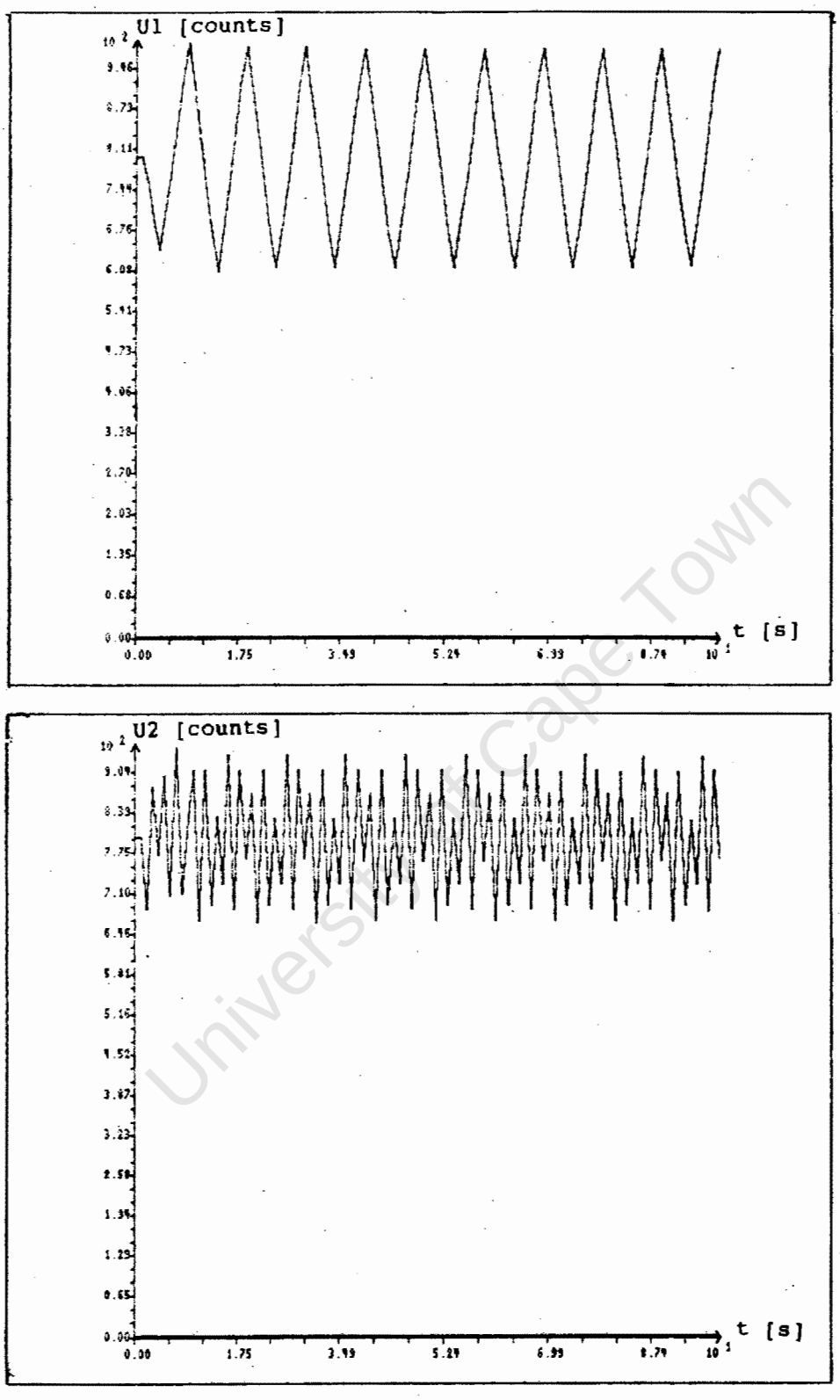


Figure 5.28: (a)Rougher input and (b)Scavenger input vs Time.

Despite the presence of the noise function, the multivariable VSC ensures that the output levels remain at their respective initial levels. Thus, for practical implementation of this controller, a start up control scheme (such as two single input P.I.D. controllers) is used. Once the specified levels have been reached, control is transferred to the VSC scheme.

Figure 5.29 illustrates the output levels of the rougher and scavenger tanks on the flotation rig.

Figure 5.30 illustrates the associated input functions and figure 5.31 illustrates the switching functions.

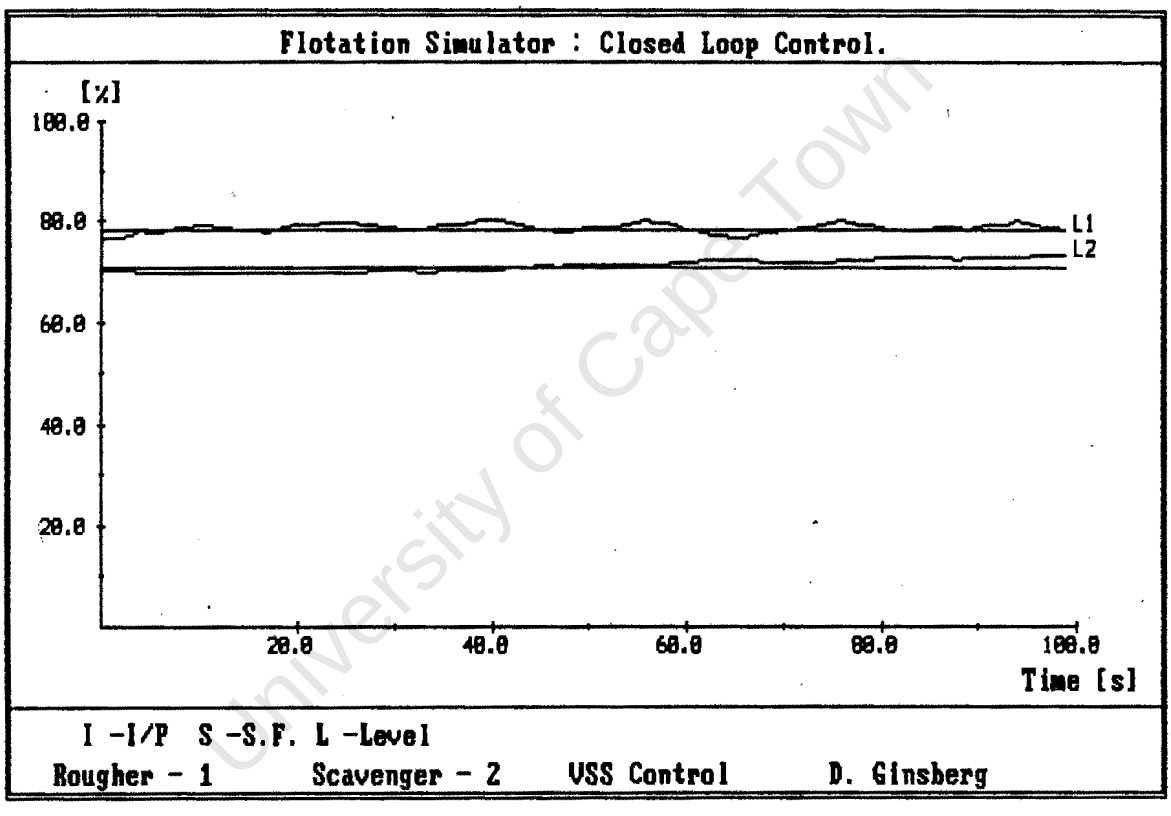
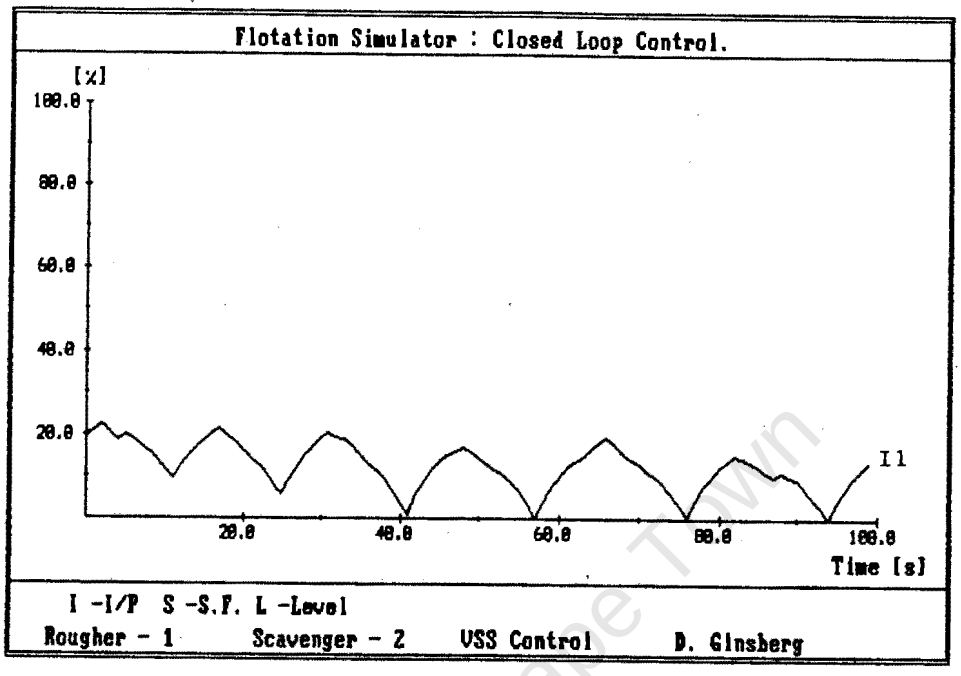


Figure 5.29: (a)Rougher level and (b)Scavenger level vs Time.



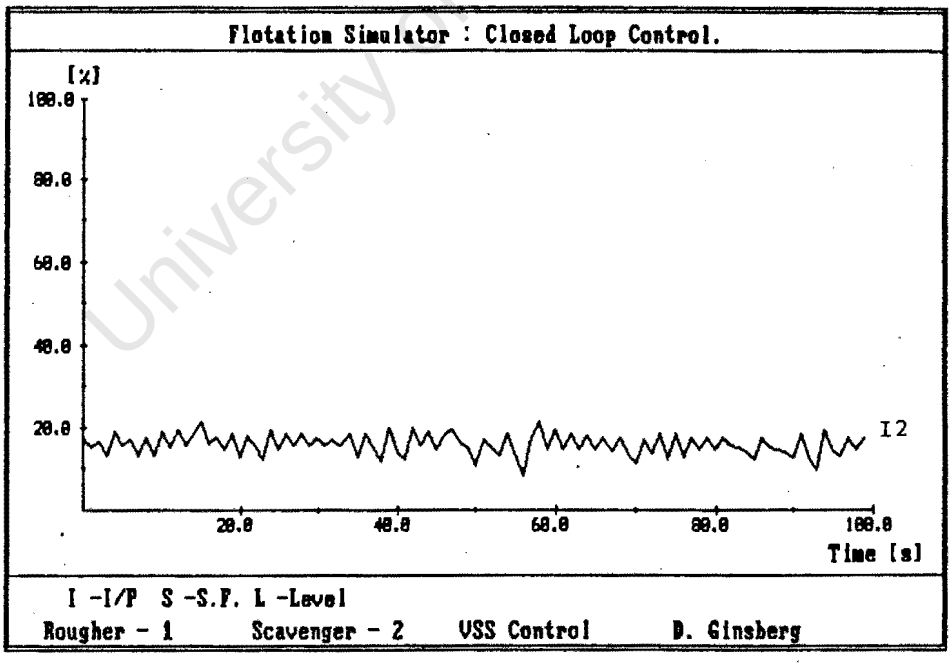


Figure 5.30: (a)Rougher input and (b)Scavenger input vs Time.

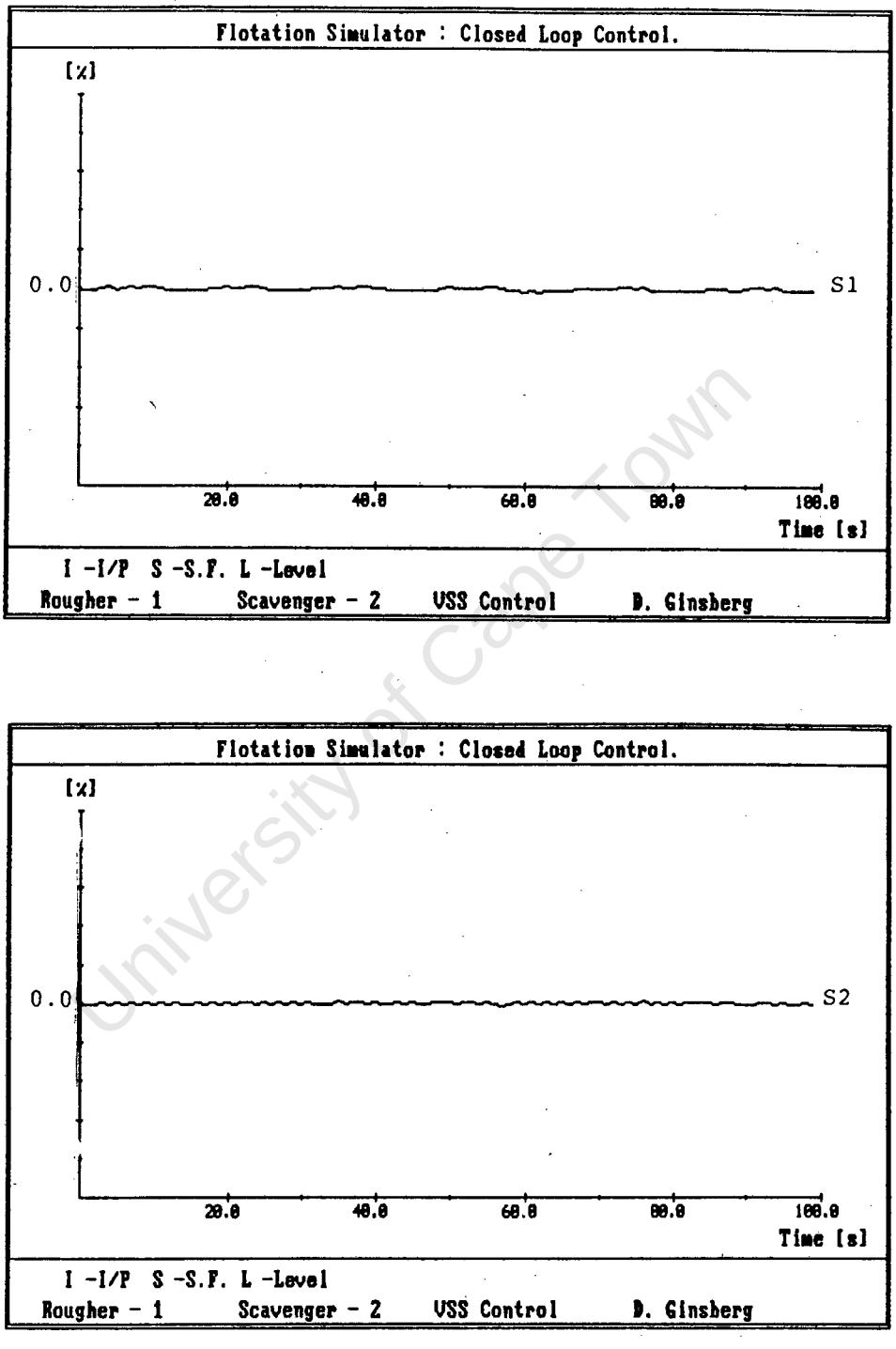


Figure 5.31: (a)Switching function 1 and (b)Switching function 2 vs Time.

The control scheme succesfully keeps the output levels at their respective operating levels.

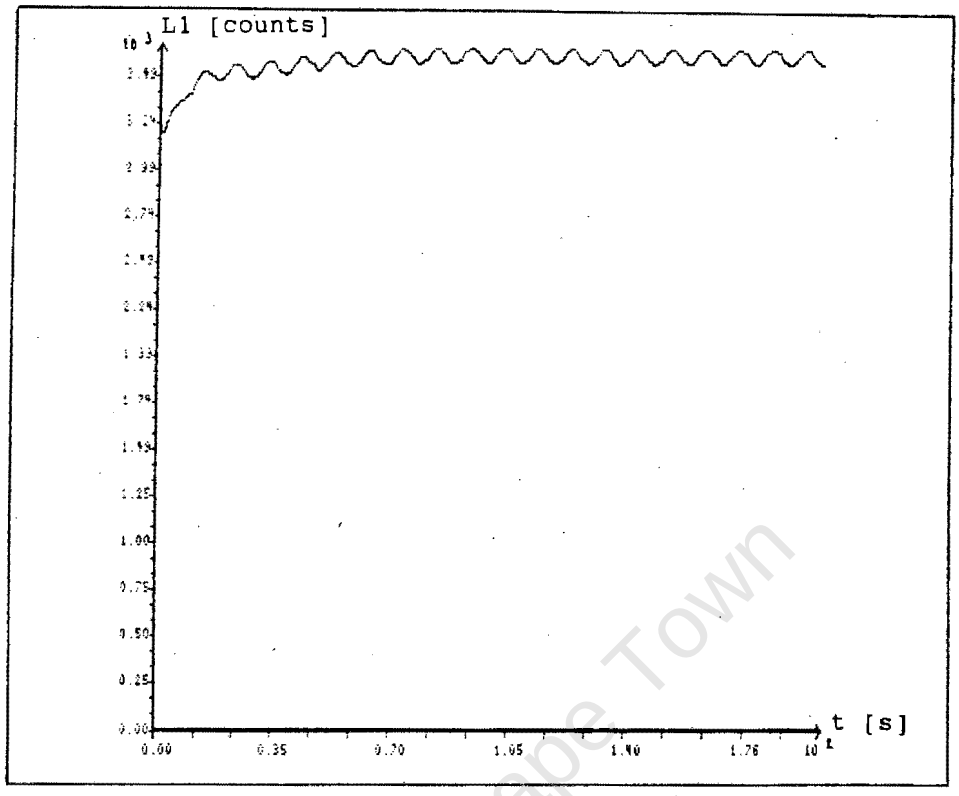
The small variations about zero of the switching functions can be seen in figure 5.31. At no time during the experiment, does the RP leave the sliding surfaces.

EXPERIMENT 2:

In this experiment, the control scheme outlined in figure 5.26 was implemented in simulation and in the actual flotation system.

To verify that a setpoint could be specified, the rougher tank is stepped by 10%.

Figure 5.32 illustrates the rougher output level in simulation and in the flotation system.



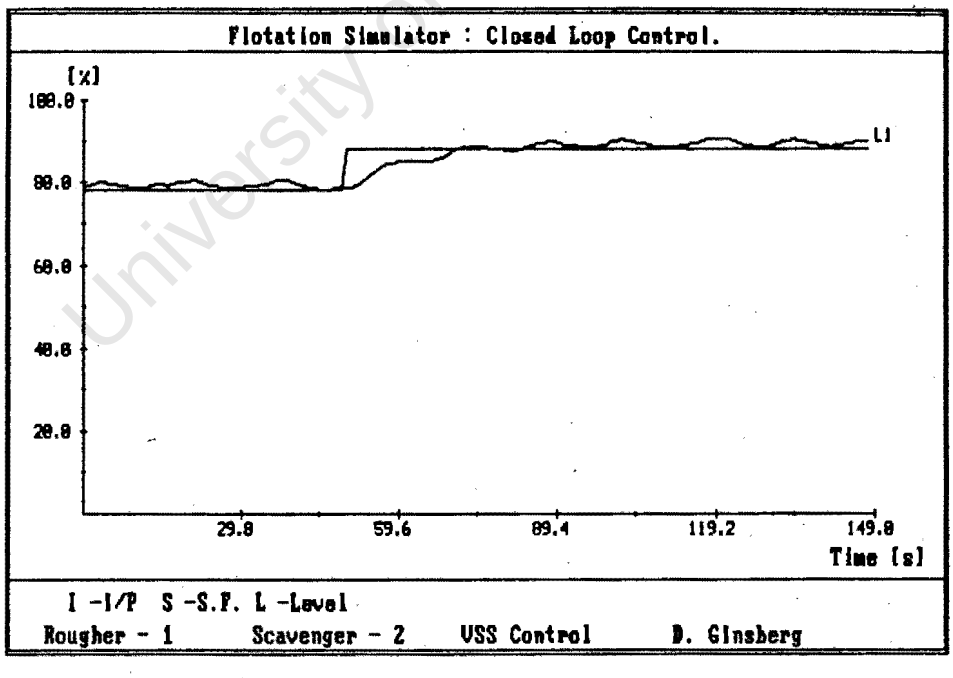


Figure 5.32: Rougher level vs Time (a) in simulation and (b) in the Flotation system.

The results of the simulated and practically implemented control scheme are very similar.

The level takes about 20 seconds to reach the specified setpoint. This can be compared to times in excess of 60 seconds when controllers designed using characteristic loci and I.N.A. methods were implemented (see results in [7] and [13]).

Figure 5.33 illustrates the scavenger level for the duration of the experiment.

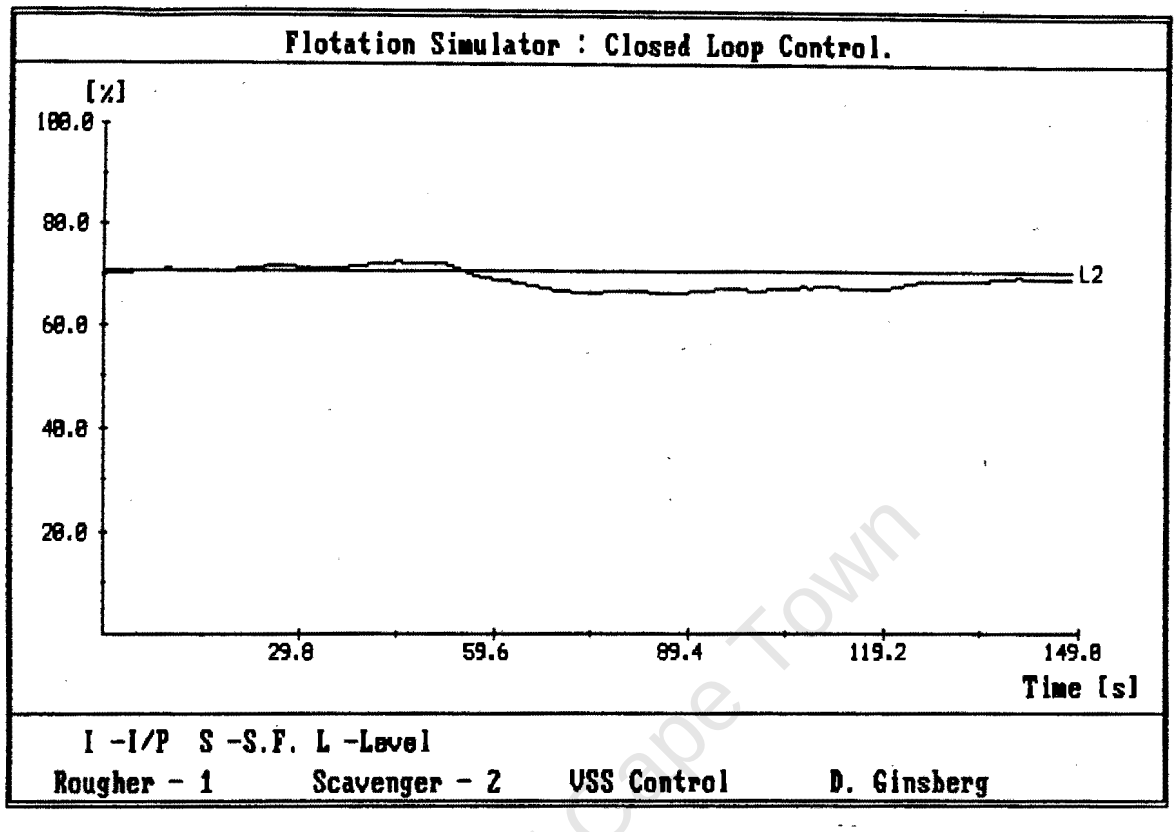


Figure 5.33: Scavenger level vs Time.

Despite the large step of the rougher valve, the scavenger level returns to its specified operating level.

The associated input functions are illustrated in figure 5.34.

The associated switching functions are illustrated in figure 5.35.

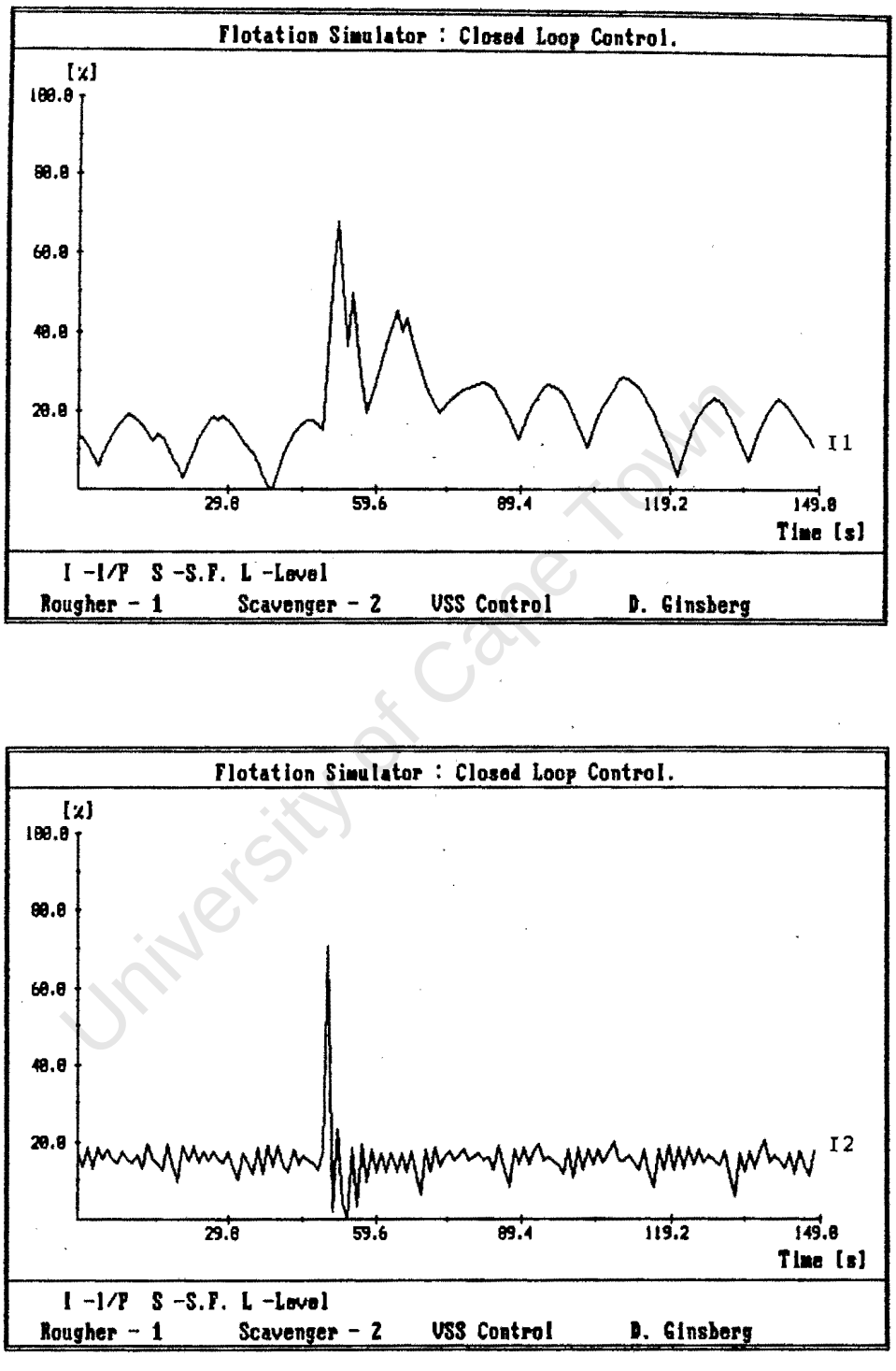
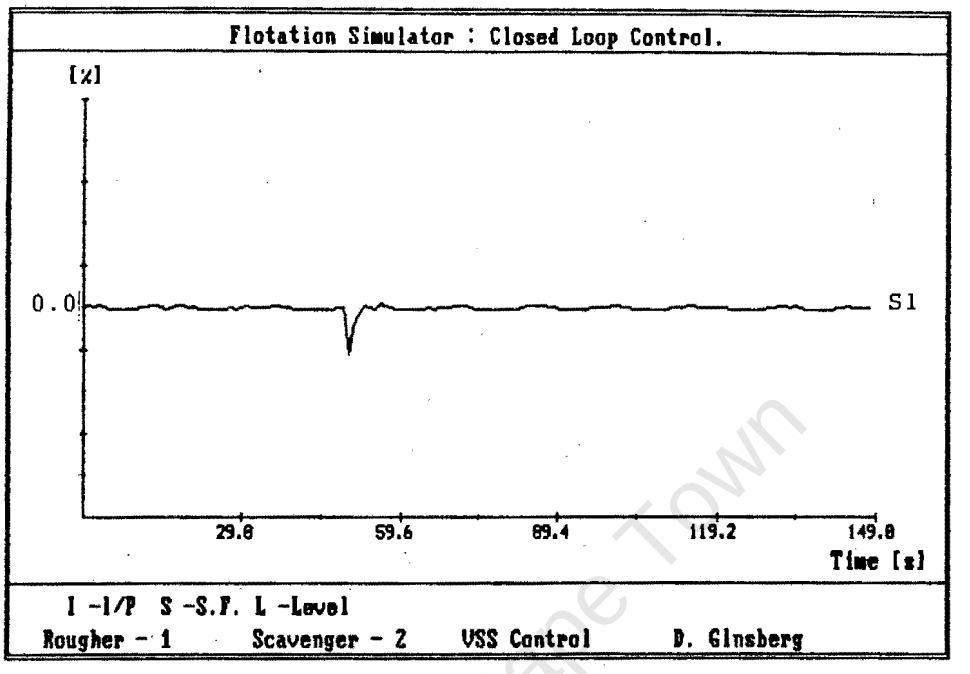


Figure 5.34: (a)Rougher input and (b)Scavenger input vs Time.



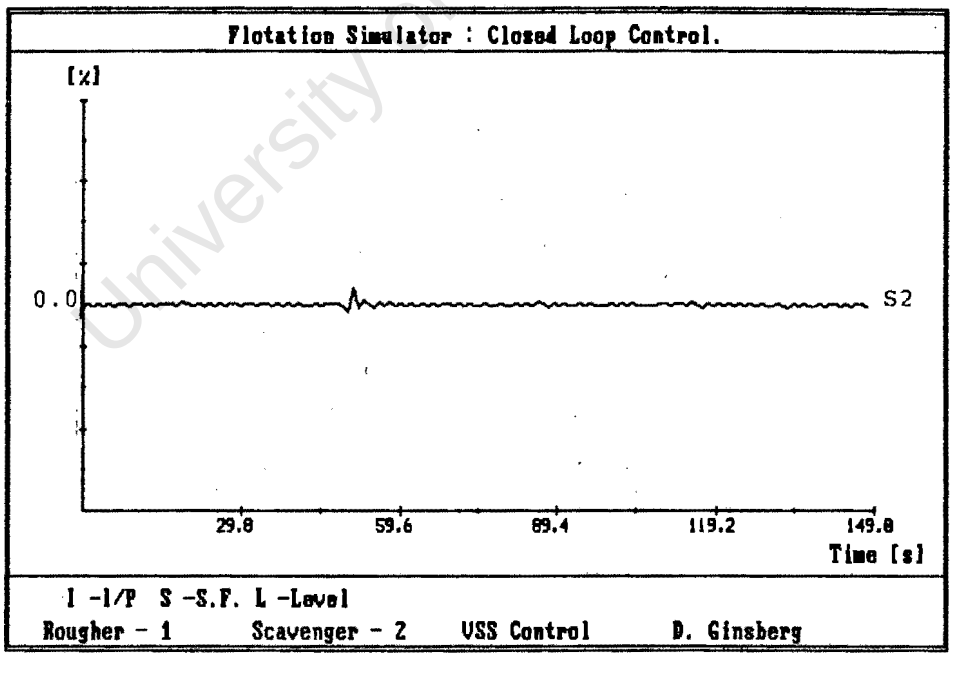


Figure 5.35: (a)Switching function 1 and (b)Switching function 2 vs Time.

It has been shown that a VSC, incorporating a setpoint facility, has been successfully implemented.

The system responds very quickly to track the specified setpoint. It has been noted that the rougher level reaches its setpoint about three times faster using a VSC, than when controllers based on I.N.A. and characteristic loci design methods were used.

The cost of this high speed transient response, is the dynamic range of the input functions; seen in figure 5.34.

Sliding is maintained throughout the experiment, as illustrated in figure 5.35.

REVIEW OF THE CHAPTER

The chapter began by describing the flotation process and why it is used for mineral extraction.

A two tank flotation plant simulator was then described. The studied system has a number of associated problem areas. These include plant parameter variations, noise signals and small linear regions.

A single input VSC was then implemented for closed loop control of the rougher tank. All practical problems, such as system modelling and setpoint implementation, were discussed and practical solutions were found for them.

The design of the controller made provision for specifying the speed of the system response. High speed system responses were obtained in practice, but the cost for this speed is an input function with a large dynamic range.

Practical obstacles, such as a limited input range (0 - 4095 counts), placed constraints on the choice of the system speed of response.

In order to facilitate control of both rougher and scavenger levels, multivariable variable structure controllers were studied.

The system was modelled from open loop step test results and the ensuing state space model was verified in simulation. The model was then checked for controllability and state observability.

The design of the multivariable VSS was based on the procedures described in chapter 4. The ensuing design is to stabilise the system and give it a suitable transient response.

Since all the system states are not available for measurement, a state observer was designed. The ensuing design was run in open loop simulation of the flotation plant and the actual states compared with the respective estimated states. The design was deemed to be satisfactory.

It was noted that VSS theory is based on system dynamics and that to implement a design in practice, non - zero steady state characteristics cannot be ignored. In addition, the theory does not make provision for setpoint tracking.

Two control schemes were described to implement multivariable VSC.

The first scheme ensured that the rougher and scavenger output levels remained at specified operating values, despite the presence of noise. Practical and simulated results supported the theory.

The second scheme modified the first scheme to facilitate the implementation of a setpoint. High speed system responses were recorded for closed loop step tests carried out in simulation and in practice.

In addition, the system becomes invariant to noise in closed loop control.

The cost of this high speed response is illustrated in the figures of the corresponding control functions. The large changes in input values, however, soon diminish.

Controllers, based on the theory of VSS, have been successfully applied to a flotation system. High quality closed loop control has been achieved for both SISO and MIMO flotation systems.

CONCLUSION

Variable structure system (VSS) theory has been investigated and applied to various systems; both in simulation and in practice.

In chapter one, the concept of combining single structure systems to form a superior, variable structure system, was introduced. This concept is made practically viable by the introduction of a sliding mode.

The sliding mode ensures insensitivity to parameter variations and external disturbances in VSS. It can thus be concluded, that VSS theory is an attractive method of controller design.

The theoretical concepts introduced in chapter one, were applied to a servo motor system. Although the theory does not make specific provision for introducing a setpoint into the system, a simple transformation of states circumvented this problem.

It would thus appear that in the case of single input single output systems, the implementation of a setpoint does not constitute a problem .

The design to control the servo motor facilitated easy specification of the closed loop response.

Results of the servo motor in closed loop control in simulation and in practice corresponded closely. Thus, for such systems, it is possible to predict accurately the closed loop behaviour of the system.

A number of non - ideal system characteristics were investigated.

In the two - dimensional phase plane, the area about the origin (the target area) and the area about the switching line (the switching surface) were isolated as areas of concern. If the phase plane representative point (RP) is in either of these two areas, correct structure switching cannot be guaranteed.

The RP might cycle about the origin or the system could even lose sliding.

It was illustrated that these areas of concern are usually only a problem for high speed applications. This places a practical constraint on the choice of design parameters.

A novel design modification, for high speed applications of the servo motor, was put forward. This method ensured no loss of sliding and had the added feature of tracking a rapidly changing setpoint.

Results of the closed loop servo motor system (which incorporated the modified controller) illustrate a high quality system response.

The servo motor has been successfully controlled and the following positive features of VSS theory have been highlighted:

1. High speed system response has been achieved. Since VSS should always be able to out - perform its component structures, it can be concluded that if speed is an important requirement for closed loop control, VSS theory offers an attractive method of control.

2. The model of the servo motor, with velocity output, was described by a first order transfer function, even though it should be described by a second order transfer function. Thus, an accurate model of the plant is not necessary when using VSS theory to design the controller.

3. All external disturbances, conveniently lumped together and termed noise, did not appreciably effect the system behaviour in closed loop control.

The major negative feature of variable structure controllers (VSC), is the large dynamic range required by the plant input. The large swing of the input function, however, disappears quickly.

The size of the gain constants are limited by the practical dynamic range of the system inputs.

In chapter 4, multivariable systems of variable structure were investigated. Practical design criteria were proposed and illustrated by way of an example.

Results from the ensuing simulations illustrate that sliding can be achieved for multivariable systems. Hence, such systems are also insensitive to plant parameter changes and external disturbances.

The focus of the thesis is VSS theory applied to flotation systems.

A VSC was designed for the closed loop control of the rougher tank part of the system. It was assumed that the level of the scavenger tank was constant throughout the experimentation.

This novel application of VSS theory highlighted a number of factors:

1. High quality control of the level of the rougher tank was achieved.

2. Despite the large variation in plant parameters, sliding was not lost.

3. Sliding was not lost, despite the superimposed noise signal on the rougher level. The noise, however, prevented the RP from reaching the origin. Thus, the RP was forced to cycle about the origin.

4. The transfer function, describing the relationship between the valve input and rougher output level, was of first order. To achieve the minimum second order transfer function, a cascaded digital filter was placed before the input to the system. Since the filter was included in the system model, the plant input was (to an extent) smoothed, without losing the robust qualities that the high frequency switching input gives the system.

The level of the scavenger tank did not remain constant throughout the experimentation on the control of the level of the rougher tank. Thus, it is concluded, that a multivariable controller is needed to control the two - tank system adequately.

A number of problems were highlighted when attempting to implement a multivariable controller to the system. These include:

1. Non - zero steady state characteristics cannot be ignored.

2. The implementation of a system setpoint is generally not facilitated by existing theory on VSS.

3. A state observer needs to be incorporated into the system.

The multivariable VSS design was synthesized using the methods and criteria highlighted in chapter four. As with all synthesis techniques, design criteria can be satisfied, but it is impossible to pinpoint the exact factor which leads to unwanted system behaviour.

The effect of zeros on system behaviour is ignored in the design stage.

It can thus be concluded, that a multivariable VSS design is generally synthesized from specified closed loop performance criteria, but needs to be implemented before the effects of factors such as system zeros can be assessed.

The design of a state observer does not constitute a design problem. The quality of the design, however, can only be assessed in simulation.

Two closed loop control strategies were implemented in the flotation system.

The first strategy involved maintaining specified levels for the two tanks. The VSC stabilised the unstable open loop system and rejected disturbances in the plant. This strategy involves a start up control scheme to arrive at the specified levels.

Thus, the design has proved to be successful and is practically viable. It is limited by not facilitating level setpoints.

The second control strategy overcomes this problem. This novel method of setpoint implementation has had practical success, but has not been rigorously proved to be a general method of setpoint implementation.

Results from this second multivariable control strategy are compared with those from similar experiments using controllers designed by I.N.A. and Characteristic Loci methods.

In both cases the variable structure controller provides the system with superior closed loop performance in terms of speed. The cost for this speed is a large dynamic range for the input control functions.

It is thus concluded, that controllers designed using VSS theory, provide an attractive means of controlling flotation systems.

The controllers ensure:

1. High speed system response.

2. Rejection of external disturbances.

3. Quality control of the system, despite plant parameter changes.

RECOMMENDATIONS FOR FURTHER RESEARCH

Based on the findings of this thesis, the following recommendations are made:

1. Variable structure system theory should be considered to design the controller for any high speed closed loop application.

2. Further research should be made into rigorously proving the design strategy to incorporate a setpoint vector into multivariable VSS, which have non - zero steady state characteristics.

REFERENCES

[1]. Borrie J.A., **Modern Control Methods:A Manual Of Design Methods**, Prentice - Hall, 1986.

[2]. Chen Chi - Tsong, **Linear Systems Theory And Design**, CBS College Publishing, 1984.

[3]. Dierks A., M.sc. Thesis in progress, University of Cape Town, 1989.

[4]. Drazenovic B., **The Invarience Conditions In VSS**, Automatica, vol.5, pp.287 - 295, 1969.

[5]. El-Ghezawi O.M.E, Zinober A.S.I.and Billings S.A., **Analysis And Design Of VSS Using A Geometric Approach**, Int. J. Control, vol.38, no.3, pp.657 - 671, 1983.

[6]. Evans M., **Self Tuning Regulation Of Fluid Level In A Flotation Cell**, B.sc. Thesis, University of Cape Town, 1988.

[7]. Fisher I.P., **Multivariable Control Of A Flotation Plant Simulator**, M.sc. Thesis, University of Cape Town, 1988.

[8]. Itkis U., **Control Systems Of Variable Structure**, Israel Universities Press, Jerusalem, 1976.

[9]. Leonhard W.,**Control Of Electrical Drives**, Springer Verlag, 1985.

[10]. Naunin (Prof Electrical Engineering Technical University of Berlin), **Control Of A.C. And D.C. Drives**, lecture notes for EEE571Z, University of Cape Town, 1989.

[11]. Sira-Ramirez H. and Dwyer III T.A.W., **Variable Structure Controller Design For Spacecraft Nutation Damping**, IEEE Transactions on Automatic Control, vol.ac-32, no.5, pp.435 - 438, May 1987.

[12]. Utkin V.I., **Survey Paper - Variable Structure Systems With Sliding Modes**, IEEE Transactions on Automatic Control, vol.ac-22, no.2, April 1977.

[13]. Venzke R., **A Comparison Of Inverse Nyquist Array And Pole Assignment Techniques**, M.sc. Thesis, University of Cape Town, 1988.

[14]. Young K.D., **Design Of Variable Structure Model - Following Control Systems**, IEEE Transactions On Automatic Control, vol.ac-23, no.6, December 1978.

[15]. Zinober A.S.I., **Controller Design Using The Theory Of VSS**, chapter 9 of **Self Tuning and Adaptive Control: Theory and Applications**, edited by C.J. Harris and S.A. Billings, 1981.

BIBLIOGRAPHY

[1]. Astrom K.J. and Wittenmark B., **Computer Controlled Systems Theory And Design**, Prentice - Hall, 1984.

[2]. Borrie J.A., **Modern Control Methods:A Manual Of Design Methods**, Prentice - Hall, 1986.

[3]. Bengiamin N.N. and Kauffmann B., **Variable Structure Position Control**, IEEE Control Systems magazine, pp.3 - 8, August 1984.

[4]. Bezvodinskaya T.A. and Sabaev E.F., **Stability Conditions In The Large For VSS**, Translated from Avtomatika i Telemekhanika, no.10, pp.64 - 68, October 1974.

[5]. Bezvodinskaya T.A. and Sabaev E.F., **Study Of State Space Features In Variable Structure Automatic Control Systems**, Translated from Avtomatika i Telemekhanika, no.7, pp.76 - 79, July 1973.

[6]. Chen Chi - Tsong, **Linear Systems Theory And Design**, CBS College Publishing, 1984.

[7]. Devaud F.M. and Caron J.Y., **Asymptotic Stability Of Model Reference Systems With Bang Bang Control**, IEEE Transactions On Automatic Control, October 1975.

[8]. Dierks A., M.sc. Thesis in progress, University of Cape Town, 1989.

[9]. Drazenovic B., **The Invarience Conditions In VSS**, Automatica, vol.5, pp.287 - 295, 1969.

[10]. Dudin Ye. B., **Switching Hyperplane In Variable Structure Tracking System**, 1963.

[20]. Mathews G.P., Decarlo R.A., Hawley P. and Lefebvre S., **Towards A Feasible Variable Structure Control Design For A Synchronous Machine Connected To An Infinite Bus**, IEEE Transactions on Automatic Control, vol.ac-31, no.12, December 1986.

[21]. Naunin (Prof Electrical Engineering Technical University of Berlin), **Control Of A.C. And D.C. Drives**, lecture notes for EEE571Z, U.C.T., 1989.

[22]. Patel R.V., Munro N., **Multivariable System Theory And Design**, Pergamon Press, Great Britain, 1982.

[23]. Rainville E.D. and Bedient P.E., **Elementary Differential Equations (Sixth Edition)**, Macmillan Publishers, 1981.

[24]. Siljak D.D., **Large - Scale Dynamic Systems Stability And Structure**, North Holland, 1978.

[25]. Sira- Ramirez H., **Harmonic Response Of Variable Structure Controlled Van der Pol Oscillators**, IEEE Transactions on Circuits and Systems, vol.cas-34, no.1, January 1987.

[26]. Sira-Ramirez H. and Dwyer III T.A.W., **Variable Structure Controller Design For Spacecraft Nutation Damping**, IEEE Transactions on Automatic Control, vol.ac-32, no.5, pp.435 - 438, May 1987.

[27]. Utkin V.I., **Survey Paper - Variable Structure Systems With Sliding Modes**, IEEE Transactions on Automatic Control, vol.ac-22, no.2, April 1977.

[28]. Venzke R., **A Comparison Of Inverse Nyquist Array And Pole Assignment Techniques**, M.sc. Thesis, University of Cape Town, 1988.

[29]. Yemelyanov S.V. and Taran V.A., **Problem Of Constructing A Variable Structure Control System For Linear Plants**, 1962.

[30]. Yemelyanov S.V. and Kostyleva N.Ye., **Synthesis Of Variable Structure Automatic Control Systems With Discontinuous Switching Function**, 1963.

[31]. Yemelyanov S.V., Bermant M.A., Kostyleva N.E., Taran V.A. and Utkin V.I., **Design Principles In VSS For Control Of Nonstationary Plants**, Session 40 paper 40.c., Control of complex systems, 24 June 1966.

[32]. Young Kar-Keung D., **Asymptotic Stability Of Model Reference Systems With Variable Structure Control**, IEEE Transactions on Automatic Control, April 1977.

[33]. Young K.D., **Design Of Variable Structure Model - Following Control Systems**, IEEE Transactions On Automatic Control, vol.ac-23, no.6, December 1978.

[34]. Zinober A.S.I., **Self - Adaptive Near Optimal Control Of Diffusion Equations**, IEE Proc., vol.127, no.6, November 1980.

[35]. Zinober A.S.I., **Controller Design Using The Theory Of VSS**, chapter 9 of **Self Tuning and Adaptive Control: Theory and Applications**, edited by C.J. Harris and S.A. Billings, 1981.

APPENDICES

APPENDIX 1: HIERARCHY OF CONTROL METHOD

This method of control was formulated by Utkin and can be found in [8].

Consider the open loop system given by:

$$\mathbf{dx}/dt = \mathbf{Ax} + \mathbf{Bu} \qquad \ldots(a1)$$

where

$\mathbf{x} \in R^n$.
$\mathbf{u} \in R^m$.

The variable structure control is of the form:

$$\begin{aligned} u_i &= u_i^+(\mathbf{x}) \quad \text{if } \sigma_i(\mathbf{x}) > 0 \\ &= u_i^-(\mathbf{x}) \quad \text{if } \sigma_i(\mathbf{x}) < 0 \end{aligned} \qquad \ldots(a2)$$

where

u_i is the ith component of $\mathbf{u}$ and $\sigma_i(\mathbf{x})$ is the ith component of the switching hyperplane:

$$\boldsymbol{\sigma}(\mathbf{x}) = C\mathbf{x} = \mathbf{0} \qquad \ldots(a3)$$

Definition 1: The open loop system in equation a1 is in the sliding mode if:

$$\lim_{\sigma_i \to 0} \sigma_i * d\sigma_i/dt < 0 \qquad \ldots(a4)$$

Definition 2: If the sliding mode occurs first on $\sigma_p(\mathbf{x}) = 0$ and then on $\sigma_p(\mathbf{x}) = 0$ and $\sigma_q(\mathbf{x}) = 0$, then the hierarchy of switching hyperplanes is denoted by $\sigma_p \rightarrow \sigma_q$.

The hierarchy of control method consists of the following steps:

STEP 1: Suppose a hierarchy of switching planes is specified by $\sigma_1 \rightarrow \sigma_2 \rightarrow \ldots \sigma_m$.

STEP 2: Begin with the bottom switching plane, σ_m, in the hierarchy, by letting $i = m$.

STEP 3: Suppose sliding occurs on the first i-1 switching planes; that is $\sigma_j = 0$ for $j = 1,..,i-1$. Solve for the equivelant control, $\mathbf{u_{eq}^{i-1}}$, of the variable structure control, $(u^{i-1})^T = (u_1,\ldots,u_{i-1})$, as a function of u_i and u^{i+1} from the algebraic equations, $\sigma_j = 0$ for $j = 1,\ldots,i-1$. $\mathbf{u^{i+1}} = (u_{i+1},\ldots, u_m)^T$.

Since the system of equation a1 is linear, $\mathbf{u_{eq}^{i-1}}$ is linear in $\mathbf{x}$, u_i and u^{i+1}.

$$\mathbf{u_{eq}^{i+1}} = P_{i-1}\mathbf{x} + Q_{i-1}\mathbf{u^{i+1}} + \alpha_i * u_i \qquad \ldots(a5)$$

where

matrices P_{i-1}, Q_{i-1} and scalar α_i depend on the first i-1 rows of matrices CA and CB. Note that $\mathbf{u^{i+1}}$ is known, since u_k^+, u_k^-, $k = 1,\ldots,m$ have been determined previously.

STEP 4: For the ith switching plane, $\sigma_i = 0$, find u_i^+ and u_i^- such that equation a4 is satisfied.

STEP 5: Let $i = i-1$. If $i > 0$ then go to step 3, else stop.

APPENDIX 2: TRANSFORMATION TO OBTAIN AN EQUIVALENT STATE SPACE SYSTEM

The following transformation can be found in [1].

For the state space system described by:

$$d\mathbf{x}/dt = A\mathbf{x} + B\mathbf{u}$$

$$\mathbf{y} = C_{sys}\mathbf{x} + E\mathbf{u} \qquad \ldots(a6)$$

Let P be an nxn nonsingular matrix with coefficients in the field of complex numbers.

If:

$$\mathbf{x_{eq}} = P\mathbf{x} \qquad \ldots(a7)$$

Then an equivalent state space model to the one defined in equation a6 can be written:

$$d\mathbf{x}_{eq}/dt = A_{eq}\mathbf{x}_{eq} + B_{eq}\mathbf{u}$$

$$\mathbf{y} = C_{syseq}\mathbf{x}_{eq} + E_{eq}\mathbf{u} \qquad \ldots(a8)$$

where

$$A_{eq} = PAP^{-1} \qquad \ldots(a9)$$

$$B_{eq} = PB \qquad \ldots(a10)$$

$$C_{syseq} = CP^{-1} \qquad \ldots(a11)$$

$$E_{eq} = E \qquad \ldots(a12)$$

P is said to be an equivalence transformation matrix.

APPENDIX 3: SIMULATOR PROGRAM

```
Program Cellsim2;

{=====================================================================}
{                                                                     }
{C  FILE              : CELLSIM2.PAS                                  }
{                                                                     }
{CA FUNCTION          : This program uses the Runge Kutte fourth order}
{                       method to simulate a linear system described  }
{                       by a set of differential equations .          }
{                                                                     }
{CM FILES CALLED      : PAD1.DOC ,PAD2.DOC,PAD3.DOC,                  }
{                       TURBO GRAPHICS TOOLBOX                        }
{                                                                     }
{CE ERROR CONDITIONS : number of points to plot and the size of       }
{                       matrices are specified in the Constant        }
{                       Definitions .                                 }
{                       The maximum number of points is 500, but      }
{                       this can be changed in the graphics files.    }
{                       The prompts and I/O are configured for a      }
{                       two drive P.C. without a hard drive .         }
{                                                                     }
{                                                                     }
{CW WRITTEN BY        : D.W. Ginsberg                                 }
{                                                                     }
{CC COMMENTS          : M.sc Thesis 1988/89                           }
{                       Supervisor : M. Braae                         }
{                       University of Cape Town Control Laboratory    }
{                                                                     }
{=====================================================================}

{=====================================================================}
{                                                                     }
{                    GLOBAL DECLARATIONS                              }
{                                                                     }
{=====================================================================}

Const

fn           = #27;                    {FUNCTION KEY CODES}
f1           = #59;
f2           = #60;
f3           = #61;
f4           = #62;
f5           = #63;
f6           = #64;
f7           = #65;
f8           = #66;

array_size   = 5;                      {SIZE OF THE A MATRIX}
array_size2  = 2;                      {NUMBER OF INPUTS AND OUTPUTS}
num_plot     = 300;                    {MAXIMUM NUMBER OF POINTS TO PLOT}
                                       {CHANGE IF EXPERIENCE MEMORY     }
                                       {OVERFLOW.                       }

Type

array_type1  = array[1..array_size] of real;
array_type2  = array[1..array_size,1..array_size]of real;
array_type3  = array[1..array_size2,1..array_size]of real;
array_type4  = array[1..array_size2] of real;
array_type5  = array[1..array_size2] of integer;
array_type6  = array[1..num_plot]of real;
array_type7  = array[1..array_size,1..num_plot]of real;
array_type8  = array[1..array_size2,1..num_plot]of real;
array_type9  = array[1..4] of real;
array_type10 = array[1..3] of real;
array_type11 = array[1..array_size,1..array_size2]of real;

    Details  = Record
                 ma       : array_type2;
                 mb       : array_type2;
```

```
                mc       : array_type3;
                sys_order:integer;
                num_inps :integer;
              End;

name         = string[25];

{$I typedef.sys}                        {GRAPHICS TOOLBOX FILES}
{$I graphix.sys}
{$I kernel.sys}
{$I windows.sys}
{$I findwrld.hgh}
{$I axis.hgh}
{$I polygon.hgh}
{$I spline.hgh}

Var

x              :array_type1;
x_sys          :array_type1;
x_init         :array_type1;
xo             :array_type1;
dx             :array_type1;
xc             :array_type1;
x_observer     :array_type1;
x_observerprev:array_type1;
x_prev         :array_type1;
sx             :array_type1;
u              :array_type1;
u_sys          :array_type1;
u_init         :array_type1;
uc             :array_type1;
up             :array_type1;
u_prev         :array_type1;
y              :array_type1;
y_sys          :array_type1;
y_init         :array_type1;
sp             :array_type1;
Matrix1        :array_type2;
Temp_matrix    :array_type2;            {(Φ-MC) OF OBSERVER}
c              :array_type3;
alpha          :array_type3;
beta           :array_type3;
gammap         :array_type4;
gamman         :array_type4;
step           :array_type4;
sum            :array_type4;
step_count     :array_type5;
pt             :array_type6;
px             :array_type7;  {Plant    State variables plotted in here}
px_observer    :array_type7;  {Observer State variables plotted in here}
py             :array_type8;  {Output data stored in here for plotting}
pu             :array_type8;
puc            :array_type8;
c1             :array_type9;
c2             :array_type10;
Matrix2        :array_type11;
G_Matrix       :array_type11;           {GAIN MATRIX M OF OBSERVER}

ch             :char;

datarec        :details;

datamaster     :file of details;

max_pts        :integer;
f_filter       :integer;
rep_flag       :integer;
f_observer     :integer;
f_noise        :integer;
counter        :integer;
control_type   :integer;
hardcopy_flag :integer;
test_type      :integer;
precedence     :integer;
code           :integer;
```

```
  Enter_flag    :integer;

  A             :Plotarray;              {USED BY TOOLBOX FOR PLOTTING}

  x_min         :real;
  x_max         :real;
  temp          :real;
  filter_const  :real;
  ob_const      :real;
  t             :real;
  dt            :real;
  {y_init        :real;}

  Line          :string[80];
  File_name     :string[15];
  File_name2    :string[15];
  Input_value   :string[10];
  header        :string[15];
  header_graph  :string[25];

  out           :text;
  Filvar        :text;


  {=============================================================}
  {                                                             }
  {                       PROCEDURES                            }
  {                                                             }
  {=============================================================}

  Procedure Menu;                          {**********MENU**********}

 Var i:integer;
  Begin
    Assign(Filvar,File_name);
    Reset(Filvar);
    i := 1;
    clrscr;
    While not Eof(Filvar) do
    Begin
      Readln(Filvar,Line);
      if( (i > 1) and (i < 23)) then Writeln(Line);
      i := i + 1;
    End;
    Close(Filvar);
    gotoxy(1,1);
 End;  {Procedure Menu}

  Procedure Datahold;                      {**********DATAHOLD**********}
  Begin
    clrscr;
    writeln('Place storage disk in drive "B" and Enter name of file .');
    gotoxy(35,5);
    textcolor(cyan);
    write('╔═══════════════════════════════════╗');
    gotoxy(35,6);
    write('‖');
    textcolor(white);
    write('B:XXXXXXXX.Dat');
    textcolor(cyan);
    write('                     ‖');
    gotoxy(35,7);
    write('╚═══════════════════════════════════╝');
    gotoxy(38,6);
    readln(file_name2);
    file_name2 := concat('B:',file_name2,'.dat');
    assign(datamaster,file_name2);
    rewrite(datamaster);
    write(datamaster,datarec);
  End; {Procedure Datahold}

  Procedure Readin;                        {**********READIN**********}

  Var legal:Boolean;
      i,j  :integer;
```

```
Begin
  legal := False;
  clrscr;
  with datarec do
  Begin
    Repeat
      clrscr;
      writeln('Enter system order:',sys_order);
      read(Input_value);
      Val(Input_value,sys_order,code);
      if code = 0 then legal := True;
    Until legal = True;
    legal := False;
    Repeat
      clrscr;
      writeln('Enter number of inputs :',num_inps);
      read(Input_value);
      Val(Input_value,num_inps,code);
      if code = 0 then legal := True;
    Until legal = True;
    clrscr;
    For i := 1 to sys_order do
    Begin
      For j := 1 to sys_order do
      Begin
        legal := False;
        Repeat
          clrscr;
          writeln('Enter value for A matrix at position:',i,j);
          read(Input_value);
          Val(Input_value,ma[i,j],code);
          if code = 0 then legal := True;
        Until legal = True;
      End;
    End;
    For i := 1 to sys_order do
    Begin
      For j := 1 to num_inps do
      Begin
        legal := False;
        Repeat
          clrscr;
          writeln('Enter value for b matrix at position:',i,j);
          read(Input_value);
          Val(Input_value,mb[i,j],code);
          if code = 0 then legal := True;
        Until legal = True;
        clrscr;
      End;
    End;
    For i := 1 to num_inps do
    Begin
      For j := 1 to sys_order do
      Begin
        legal := False;
        Repeat
          clrscr;
          writeln('Enter value for cT matrix at position:',i,j);
          read(Input_value);
          Val(Input_value,mc[i,j],code);
          if code = 0 then legal := True;
        Until legal = True;
        clrscr;
      End;
    End;
  End;
End; {Procedure Readin}

Procedure Fetchdata;                    {**********FETCHDATA**********}

Var ok :boolean;

Begin
  clrscr;
Repeat
  writeln('Place storage disk in drive "B" and Enter name of file');
```

```
  gotoxy(35,5);
  textcolor(cyan);
  write('╔═══════════════════════════════════╗');
  gotoxy(35,6);
  write('║');
  textcolor(white);
  write('B:XXXXXXXX.Dat');
  textcolor(cyan);
  write('                          ║');
  gotoxy(35,7);
  write('╚═══════════════════════════════════╝');
  gotoxy(38,6);
  readln(file_name2);
  file_name2 := concat('B:',file_name2,'.dat');
  assign(datamaster,file_name2);
  {$I-} reset(datamaster) {$I+};
  ok := (ioresult = 0);
  if not ok then writeln('cannot find file',file_name2);
until ok;
    while not eof(datamaster)do
    Begin
      read(datamaster,datarec);
    End;
    close(datamaster);
    clrscr;
End; {Procedure Fetchdata}

Procedure Enter_sys;                    {**********ENTER_SYS**********}

Label Error,Error1;

Begin
  clrscr;
  File_name := 'b:pad1.doc';
  Menu;
  gotoxy(3,6);
  write('POINTS');
  gotoxy(22,6);
  write('FILE');
  gotoxy(3,11);
  write('HALT');
  gotoxy(39,10);
  write('Enter system by');
  gotoxy(39,11);
  write('Points or off File?');
  Error:
  Repeat until keypressed;
  read(kbd,ch);
  If ch = fn then read(kbd,ch)
  Else goto Error;
  Case ch of
    f1:Begin
         clrscr;
         Readin;
         File_name := 'b:pad1.doc';
         Menu;
         gotoxy(3,6);
         write('YES');
         gotoxy(22,6);
         write('NO');
         gotoxy(39,10);
         write('Store system on File?');
         Error1:
         Repeat until keypressed;
         read(kbd,ch);
         If ch = fn then read(kbd,ch)
         Else goto Error1;
         if (ch = f1) then datahold;
       End;
    f2:fetchdata;
    f3:Halt;
  End;
End; {Enter_sys}

Procedure Sim_type;                     {**********SIM_TYPE**********}
```

```
Label Error;

Begin
  With datarec do
  Begin
    clrscr;
    File_name := 'b:pad1.doc';
    Menu;
    gotoxy(3,6);
    write('Open Loop');
    gotoxy(22,6);
    write('VSS');
    gotoxy(3,10);
    write('O/L + VSS');                {UNUSED AT PRESENT}
    gotoxy(22,10);
    write('SET TEST');
    gotoxy(39,10);
    write('Type of Simulation');
    gotoxy(3,16);
    write('HALT');
    Error:
    Repeat until keypressed;
    read(kbd,ch);
    If ch = fn then read(kbd,ch)
    Else goto Error;
    Case ch of
      f1:control_type := 1;
      f2:control_type := 2;
      f3:control_type := 3;
      f4:control_type := 4;
      f5:Halt;
    End;
  End;
End; {Procedure Sim_type}

Procedure Step_tests;              {**********STEP_TESTS**********}

Var legal:Boolean;
    i    :integer;

Begin
  With Datarec do
  Begin
    For i := 1 to num_inps do
    Begin
      legal := False;
      Repeat
        clrscr;
        writeln('Enter step size for input :',i);
        read(Input_value);
        Val(Input_value,step[i],code);
        If code = 0 then legal := True;
      Until legal = True;
      legal := False;
      Repeat
      clrscr;
      writeln('Enter when to step input[',i,']', step_count[i]);
      read(Input_value);
        Val(Input_value,step_count[i],code);
        If code = 0 then legal := True;
      Until legal = True;
    End;
  End;
End; {Procedure Step_tests}

Procedure VSS_parameters;          {**********VSS_PARAMETERS**********}

Var legal :Boolean;
    i,j,k :integer;
Begin
  With Datarec do
  Begin
    If sys_order = 1 then k := 2
    Else k := sys_order;
    For i := 1 to num_inps do
    Begin
```

```
        For j := 1 to k do
        Begin
          legal := False;
          Repeat
            clrscr;
            writeln('Enter α[',i,j,'] value:',alpha[i,j]:6);
            read(Input_value);
            Val(Input_value,alpha[i,j],code);
            If code = 0 then legal := True;
          Until legal = True;
          legal := False;
          Repeat
            clrscr;
            writeln('Enter β[',i,j,'] value:',beta[i,j]:6);
            read(Input_value);
            Val(Input_value,beta[i,j],code);
            If code = 0 then legal := True;
          Until legal = True;
        End;
      End;
      If num_inps > 1 then
      Begin
        legal := False;
        Repeat
          clrscr;
          writeln('Enter which hyperplane takes precedence ?',precedence);
          read(Input_value);
          Val(Input_value,precedence,code);
          If code = 0 then legal := True;
        Until legal = True;
        legal := False;
        Repeat
          clrscr;
          writeln('Enter gamma+[',precedence,'] value:',gammap[precedence]:6);
          read(Input_value);
          Val(Input_value,gammap[precedence],code);
          If code = 0 then legal := True;
        Until legal = True;
        legal := False;
        Repeat
          clrscr;
          writeln('Enter gamma-[',precedence,'] value:',gamman[precedence]:6);
          read(Input_value);
          Val(Input_value,gamman[precedence],code);
          If code = 0 then legal := True;
        Until legal = True;
      End;
      For i := 1 to num_inps do
      Begin
        For j := 1 to k do
        Begin
          legal := False;
          Repeat
            clrscr;
            writeln('Enter c[',i,j,']:',c[i,j]:6);
            read(Input_value);
            Val(Input_value,c[i,j],code);
            If code = 0 then legal := True;
          Until legal = True;
        End;
      End;
      For i := 1 to k do
      Begin
        legal := False;
        Repeat
          clrscr;
          writeln('Enter setpoint for state[',i,']',sp[i]:6);
          read(Input_value);
          Val(Input_value,sp[i],code);
          If code = 0 then legal := True;
        Until legal = True;
        clrscr;
      End;
    End;
  End; {Procedure VSS_parameters}
```

```
Label Error;

Begin
  clrscr;
  File_name := 'b:pad1.doc';
  Menu;
  gotoxy(3,6);
  write('YES');
  gotoxy(22,6);
  write('NO');
  gotoxy(3,11);
  write('HALT');
  gotoxy(39,10);
  write('ADD NOISE ?');
  Error:
  Repeat until keypressed;
  read(kbd,ch);
  If ch = fn then read(kbd,ch)
  Else goto Error;
  Case ch of
  f1:F_noise := 1;
  f2:F_noise := 0;
  End;
  clrscr;
End; {Procedure Noise}

Procedure filter;                    {**********FILTER**********}

Label Error,Error1;

Var legal :boolean;

Begin
  clrscr;
  File_name := 'b:pad1.doc';
  Menu;
  gotoxy(3,6);
  write('YES');
  gotoxy(22,6);
  write('NO');
  gotoxy(3,11);
  write('HALT');
  gotoxy(39,10);
  write('Is a Filter required?');
  Error:
  Repeat until keypressed;
  read(kbd,ch);
  If ch = fn then read(kbd,ch)
  Else goto Error;
    if(ch = f1)then
    Begin
      clrscr;
      File_name := 'b:pad1.doc';
      Menu;
      gotoxy(3,6);
      write('Integrator');
      gotoxy(22,6);
      write('L.P.F.');
      Error1:
      Repeat until keypressed;
      read(kbd,ch);
      If ch = fn then read(kbd,ch)
      Else goto Error1;
      if ch = f1 then f_filter := 1;
      if ch = f2 then
      Begin
        legal := False;
        Repeat
          clrscr;
          writeln('Enter filter constant ');
          read(Input_value);
          Val(Input_value,filter_const,code);
          if code = 0 then legal := True;
        Until legal = True;
        f_filter := 2;
      End;
```

```
    End
    Else f_filter := 0;
    if ch = f3 then Halt;
End; {Procedure Filter}

Procedure Calc_input;                    {**********CALC_INPUT**********}

Var i,j,k:integer;

Begin
  With Datarec do
  Begin
    If sys_order = 1 then k:= 2
    Else k := sys_order;
    Case control_type of
    2,4:Begin

{********** actual states or the difference between states **********}
{           Change Manually ! }

{          If(F_observer = 0)then For i := 1 to k do xc[i] := sp[i] - x[i]
          Else For i := 1 to k do xc[i] := sp[i] - x_observer[i];}

          For i := 1 to k do
          Begin
            If(F_observer = 1)then
            Begin
              xc[i] := x_observer[i] - x_observerprev[i];
              x_observerprev[i] := x_observer[i];
            End
            Else
            Begin
              xc[i] := x_sys[i] - x_prev[i];
              x_prev[i] := x_sys[i];
            End;
          End;
{*************************************************************************}

          For i := 1 to num_inps do sum[i] := 0;
          For i := 1 to num_inps do
          Begin
            For j := 1 to k do sum[i] := sum[i] + c[i,j]*xc[j];
          End;
          For i := 1 to num_inps do
          Begin
            uc[i] := 0.0;
            For j := 1 to k do
            Begin
              if(sum[i]*xc[j] > 0.0)then
              Begin
                uc[i] := uc[i] - alpha[i,j]*xc[j];
              End
              Else
              Begin
                uc[i] := uc[i] - beta[i,j] *xc[j];
              End;
            End;
            If num_inps > 1 then
            Begin
              If (precedence = 1)then
              Begin
                if(sum[1]*sum[2] > 0)then
                Begin
                  uc[1] := uc[1] - gammap[1]*sum[2];
                End
                Else
                Begin
                  uc[1] := uc[1] - gamman[1]*sum[2];
                End;
              End;
              If (precedence = 2)then
              Begin
                if(sum[1]*sum[2] > 0)then
                Begin
                  uc[2] := uc[2] - gammap[2]*sum[1];
                End
```

```
                Else
                Begin
                   uc[2] := uc[2] - gamman[2]*sum[1];
                End;
              End;
            End;
            u_sys[i] := 0.0;
            Case f_filter of
              0:u_sys[i] := uc[i];
              1:u_sys[i] := u_prev[i] + dt*uc[i];
              2:u_sys[i] := exp(-dt/filter_const)*u_prev[i] +
                        (1 - exp(-dt/filter_const))*uc[i];
            End;

            u_sys[i] := u_sys[i] + u_prev[i]; {Integrate the I/P}
            u_prev[i] := u_sys[i];
{            u_sys[i] := u_sys[i] + sp[i];  }   {Bump the system}

            If u_sys[i] > 4095 then u_sys[i] := 4095; {I/P Limiter}
            If u_sys[i] < 0    then u_sys[i] := 0;
            up[i] := u_sys[i];
            u[i] := u_sys[i] - u_init[i];     {Dynamic Input}
          End;
        End;
        1:Begin
            For i := 1 to num_inps do
            Begin
              if(counter = step_count[i])then u_sys[i] := u_sys[i] + step[i];
              up[i] := u_sys[i];
              u[i] := u_sys[i] - u_init[i];
            End;
            For j := 1 to sys_order do x_observerprev[j] := x_observer[j];
          End;
    End;
  End;
End; {Procedure Calc_input}


Procedure Plant;                          {**********PLANT**********}

Var i,j,k:integer;
Begin
  With datarec do
  Begin
    For i := 1 to sys_order do
    Begin
      sx[i] := 0;
      xo[i] := x[i];
    End;
    For k := 1 to 4 do
    Begin
      For i := 1 to sys_order do
      Begin
        dx[i] := 0;
        For j := 1 to sys_order do dx[i] := dx[i] + ma[i,j]*x[j];
        For j := 1 to num_inps  do dx[i] := dx[i] + mb[i,j]*u[j];
      End;
      For i := 1 to sys_order do
      Begin
        sx[i] := sx[i] + c1[k]*dx[i];
        if(k < 4)then x[i] := xo[i] + c2[k]*dx[i]*dt
        Else
        x[i] := xo[i] + sx[i]*dt/6;
        If ((f_noise = 1) and (f_observer = 0))then
        Begin
          x[1] := x[1] + 5*sin(0.62832*t);    {Manually implemented noise}
          x[5] := x[1] + 5*sin(0.62832*t);    {Functions .               }
        End;
        x_sys[i] := x_init[i] + x[i];
      End;
    End;
  End;
End; {Procedure Plant}

Procedure Observer; {************(Manually Implemented) OBSERVER**************}
```

```
  Var i,j,k :integer;

  Begin
    With datarec do
    Begin
      {***********Calculate Output**************}

      For i := 1 to num_inps do
      Begin
        y_sys[i] := y_init[i];
        For j := 1 to sys_order do
        Begin
          y_sys[i] := y_sys[i] + mc[i,j]*x[j];
        End;
        if f_noise = 1 then y_sys[1] := y_sys[1] + 18*sin(0.62832*t);
      End;

      {**********Calculate new states**********}

      For i := 1 to sys_order do
      Begin
        x_observer[i] := 0.0;
        For j:= 1 to sys_order do
        Begin
          x_observer[i] := x_observer[i] + Temp_Matrix[i,j]*x_observerprev[j];
        End;
        For j:= 1 to num_inps do
        Begin
          x_observer[i] := x_observer[i] + mb[i,j]*dt*(u_sys[j] - u_init[j]);
        End;
        For j:= 1 to num_inps do
        Begin
          x_observer[i] := x_observer[i] + G_Matrix[i,j]*(y_sys[j] - sp[j]);
        End;
      End;
    End;
  End; {Procedure Observer}

  Procedure Graph_op;                    {**********GRAPH_OP**********}

  Label Error,Error1;

  Var legal  :Boolean;
      i,j,k,l,m:integer;
  Begin
    With datarec do
    Begin
      If sys_order = 1 then i := 2
      Else i := sys_order;
      For m := 1 to 2 do
      Begin
        File_name := 'b:pad3.doc';
        Menu;
        gotoxy(3,6);
        write('TIME');
        gotoxy(22,6);
        write('PLANT STATES');
        gotoxy(41,6);
        write('OBSERV STATES');
        gotoxy(3,11);
        write('ERROR STATES');
        gotoxy(22,11);
        write('OUTPUT');
        gotoxy(41,11);
        write('VSS INPUT');
        gotoxy(3,16);
        write('PLANT INPUT');
        gotoxy(22,16);
        write('HYPERPLANE');
        gotoxy(40,19);
        if m = 1 then write('X');
        if m = 2 then write('Y');
        Error:
        Repeat until keypressed;
        read(kbd,ch);
        If ch = fn then read(kbd,ch)
```

```
                A[j,m] := px_observer[k,j] - sp[k];
          If(A[(counter-1),m] = 0.0)then A[(counter-1),m] := 0.001;
                                {Default For graphics}
          For j := counter to num_plot do A[j,m] := 0.0;
        End;
      f5:Begin
          legal := False;
          Repeat
            clrscr;
            writeln('Enter Output to plot. ');
            read(Input_value);
            Val(Input_value,k,code);
            if code = 0 then legal := True;
          Until legal = True;
          For j := 1 to (counter - 1) do  A[j,m] := py[k,j];
          If(A[(counter-1),m] = 0.0)then A[(counter-1),m] := 0.001;
                                {Default For graphics}
          For j := counter to num_plot do A[j,m] := 0.0;
        End;
      f6:Begin
          legal := False;
          Repeat
            clrscr;
            writeln('Enter VSS Input to plot. ');
            read(Input_value);
            Val(Input_value,k,code);
            if code = 0 then legal := True;
          Until legal = True;
            For j := 1 to (counter - 1) do A[j,m] := puc[k,j];
            If(A[(counter-1),m] = 0.0)then A[(counter-1),m] := 0.001;
                                {Default For graphics}

            For j := counter to num_plot do A[j,m] := 0.0;
        End;
      f7:Begin
          legal := False;
          Repeat
            clrscr;
            writeln('Enter Plant Input to plot. ');
            read(Input_value);
            Val(Input_value,k,code);
            if code = 0 then legal := True;
          Until legal = True;
            For j := 1 to (counter - 1) do A[j,m] := pu[k,j];
            If(A[(counter-1),m] = 0.0)then A[(counter-1),m] := 0.001;
                                {Default For graphics}

            For j := counter to num_plot do A[j,m] := 0.0;
        End;
      f8:Begin
          legal := False;
          Repeat
            clrscr;
            writeln('Enter Hyperplane to plot. ');
            read(Input_value);
            Val(Input_value,k,code);
            if code = 0 then legal := True;
          Until legal = True;
            For j := 1 to (counter - 1) do
            Begin
              A[j,m] := 0;
              If F_observer = 0 then
                   For l := 1 to i do
                        A[j,m] := A[j,m] + c[k,l]*(px[l,j]-sp[l])
              Else For l := 1 to i do
                        A[j,m] := A[j,m] + c[k,l]*(px_observer[l,j]-sp[l]);
            End;
            If(A[(counter-1),m] = 0.0)then A[(counter-1),m] := 0.001;
                                {Default For graphics}

            For j := counter to num_plot do A[j,m] := 0.0;
        End;
    End;
End;
clrscr;
writeln('Enter Title of graph');
```

```
    gotoxy(35,5);
    write('╔═══════════════════════════╗');
    gotoxy(35,6);
    write('╟                           ╢');
    gotoxy(36,6);
    write('XXXXXXXXXXXXXXXXXXXXXXXXXX');
    gotoxy(35,7);
    write('╚═══════════════════════════╝');
    gotoxy(36,6);
    readln(header_graph);
    clrscr;
  End;
End; {Procedure Graph_op}

Procedure plot_graph;                    {**********PLOT_GRAPH**********}

Var dx,dy,x1,y1,x2,y2,lines,scale :Integer;
    arrows                        :Boolean;


Begin
  arrows := true;
  If Enter_flag = 0 then
  Begin
    Initgraphic;              {initialize the graphics system}
    Enter_flag := 1;
  End
  Else Entergraphic;
  ClearScreen;
  DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
  DefineWindow(2,0,0,XMaxGlb,YMaxGlb);
  DefineHeader(2,Header_graph);
  DefineWorld(2,0,YmaxGlb,XMaxGlb,0);
  SelectWindow(2);
  SetHeaderOn;
  FindWorld(2,A,counter,1,1);
  SelectWorld(2);
  SelectWindow(2);
  DrawBorder;
  Dx := -7;
  Dy := -7;
  X1 := 10;
  Y1 := 10;
  X2 := 10;
  Y2 := 10;
  Lines := 0;
  Scale := 0;
  SetLineStyle(0);
  DrawAxis(dx,dy,x1,y1,x2,y2,lines,scale,Arrows);
  DrawPolygon(A,1,(counter-1),0,0,0);      {draw the polygon}
  Case Hardcopy_flag of
    1:Hardcopy(False,1);
    2:repeat until keypressed;
  End;
  LeaveGraphic;    {leave the graphics system}
End; {Procedure Plot_graph}

Procedure User_graphics;                 {**********USER_GRAPHICS*******}

Label Error2,Error3,Rep_graph;

Begin
  Rep_graph:
  Graph_op;
  plot_graph;
  clrscr;
  File_name := 'b:pad1.doc';
  Menu;
  gotoxy(3,6);
  write('YES');
  gotoxy(22,6);
  write('NO');
  gotoxy(39,10);
  write('HARDCOPY OF GRAPH?');
  Error2:
  Repeat until keypressed;
```

```
    read(kbd,ch);
    If ch = fn then read(kbd,ch)
    Else goto Error2;
    if ch =  f1 then
    Begin
      hardcopy_flag := 1;
      plot_graph;
    End;
    hardcopy_flag := 2;
    clrscr;
    File_name := 'b:pad1.doc';
    Menu;
    gotoxy(3,6);
    write('YES');
    gotoxy(22,6);
    write('NO');
    gotoxy(39,10);
    write('SEE ANOTHER GRAPH?');
    Error3:
    Repeat until keypressed;
    read(kbd,ch);
    If ch = fn then read(kbd,ch)
    Else goto Error3;
    if ch =  f1 then goto Rep_graph;
  End;{Procedure User_graphics}

  Procedure Initialise;                    {**********INITIALISE**********}

  Var i,j:integer;
  Begin
    With datarec do
    Begin
      If rep_flag = 0 then
      Begin
        c1[1] := 1;                        {R.K.4 CONSTANTS }
        c1[2] := 2;
        c1[3] := 2;
        c1[4] := 1;
        c2[1] := 0.5;
        c2[2] := 0.5;
        c2[3] := 1.0;

        counter := 1;
        precedence := 2;
        Enter_flag := 0;

        For i := 1 to array_size do
        Begin
          For j := 1 to array_size do
          Begin
            ma[i,j] := 0;
            mb[i,j] := 0;
            Temp_Matrix[i,j] :=0.0;
          End;
          For j := 1 to array_size2 do
          Begin
            G_Matrix[i,j] := 0.0;
          End;
            xo[i] := 0;
            dx[i] := 0;
            xc[i] := 0;
            sx[i] := 0;
            u[i] := 0;
            uc[i] := 0;
            y[i]  := 0;
            sp[i] := 0;
            x_init[i] := 0.0;
            u_init[i] := 0.0;
            y_init[i] := 0.0;
            x_sys[i]  := 0.0;
            u_sys[i]  := 0.0;
            y_sys[i]  := 0.0;
        End;
        f_filter := 0;
        f_observer := 0;
        hardcopy_flag := 2;
```

```
      alpha[1,1]:= 0.35;                {TEST VALUES }
      alpha[1,2]:= 1.6;
      alpha[1,3]:= 0.3;
      alpha[1,4]:= 0.6;
      alpha[1,5]:= 0.6;
      alpha[2,1]:= 0.1;
      alpha[2,2]:= 0.8;
      alpha[2,3]:= 0.9;
      alpha[2,4]:= 0.8;
      alpha[2,5]:= 0.6;
      beta[1,1] := -0.8;
      beta[1,2] := -3.5;
      beta[1,3] := -0.2;
      beta[1,4] := -1.2;
      beta[1,5] := -1.2;
      beta[2,1] := -0.5;
      beta[2,2] := -0.6;
      beta[2,3] := -0.4;
      beta[2,4] := -0.8;
      beta[2,5] := -1.6;
      gammap[2] := 4;
      gamman[2] := -3;
      For i := 1 to array_size2 do
      Begin
        For j := 1 to array_size do c[i,j] :=0.1;
      End;
      c[1,2] := 1.0;
      c[1,3] := 0.0;
      c[2,5] := 1.0;
      G_Matrix[1,1] := 1.0;             {OBSERVER MATRICES DEFINED}
      G_Matrix[2,1] := 1.0;
      G_Matrix[3,2] := 1.0;
      G_Matrix[5,2] := 1.0;
      Temp_Matrix[1,1] := 0.9985;
      Temp_Matrix[1,2] := 0.3623;
      Temp_Matrix[1,3] :=-0.0065;
      Temp_Matrix[2,1] :=-0.0015;
      Temp_Matrix[2,2] := 0.3225;
      Temp_Matrix[2,3] :=-0.0065;
      Temp_Matrix[3,3] := 0.9893;
      Temp_Matrix[3,4] := 0.4571;
      Temp_Matrix[3,5] := 0.0091;
      Temp_Matrix[4,4] := 0.9646;
      Temp_Matrix[5,3] := 0.0197;
      Temp_Matrix[5,4] := 0.4571;
      Temp_Matrix[5,5] := 0.9787;
    End;
    For i := 1 to num_plot do pt[i]:=0.0;
    For i := 1 to array_size do
    Begin
      For j := 1 to num_plot do
      Begin
        px[i,j] := 0.0;
        px_observer[i,j] := 0.0;
      End;
    End;
    For i := 1 to array_size2 do
    Begin
      For j := 1 to num_plot do
      Begin
        pu[i,j] := 0.0;
        puc[i,j] := 0.0;
      End;
    End;
  End;
End;{Procedure Initialise}

Procedure Repeat_p;                     {**********REPEAT_P**********}

Label Error;

Begin
    clrscr;
    File_name := 'b:pad1.doc';
    Menu;
```

```
    gotoxy(3,6);
    write('REPEAT');
    gotoxy(22,6);
    write('QUIT');
    gotoxy(39,10);
    write('REPEAT OR QUIT ?');
    Error:
    Repeat until keypressed;
    read(kbd,ch);
    If ch = fn then read(kbd,ch)
    Else goto Error;
    if(ch = f1)then rep_flag := 1
    Else rep_flag := 0;
    clrscr;
End;

Procedure Simulate;                    {**********SIMULATE**********}

Label Error;

Var i,j,k:integer;

Begin
  With Datarec do
  Begin
    If sys_order = 1 then j := 2
    Else j := sys_order;
    clrscr;
    File_name := 'b:pad1.doc';
    Menu;
    gotoxy(3,6);
    write('OBSERVER');
    gotoxy(22,6);
    write('PLANT');
    gotoxy(39,10);
    write('ESTIMATE STATES BY');
    gotoxy(39,11);
    write('OBSERVER OR DIRECTLY?');
    Error:
    Repeat until keypressed;
    read(kbd,ch);
    If ch = fn then read(kbd,ch)
    Else goto Error;
    Case ch of
      f1:F_observer := 1;
      f2:F_observer := 0;
    End;
    clrscr;
    Case control_type of
      1:Step_tests;
      2:Vss_parameters;
      4:Vss_parameters;
    End;
    clrscr;
    pt[1] := 0.0;
    For i := 1 to j do
    Begin
      px[i,1] := x_init[i];
      if f_observer = 1 then px_observer[i,1] := x_init[i];
    End;
    For i := 1 to num_inps do
    Begin
      pu[i,1] := u_init[i];
      puc[i,1] := 0.0;                 {**?**}
      py[i,1] := y_init[i];
    End;
    counter := 2;
    t := 0.0;
    If control_type = 4 then
    Begin
      While counter <= 50 do
      Begin
        t := t + dt;
        pt[counter] := t;
        For i := 1 to num_inps do u[i] := 0.0;
        For i := 1 to num_inps do
```

```
        Begin
          pu[i,counter] :=  up[i];
          puc[i,counter] := uc[i];
        End;
        Plant;
        if f_observer = 1 then Observer;
        For i := 1 to j do
        Begin
          px[i,counter] :=  x[i]+x_init[i];
          px_observer[i,counter] := x_observer[i] + x_init[i];
        End;
        counter := counter + 1;
      End;
    End;
    If control_type = 4 then sp[1] := 5;
    If (f_observer = 1 ) then Observer;
    For i := 1 to num_inps do py[i,counter] := y_sys[i];
    While(counter <= max_pts)do
    Begin
      If control_type = 4 then
      Begin
        If(( counter > 200) and (counter <= 400)) then sp[1] := 10;
        If counter > 400 then sp[1] := sp[1] - 0.03;
      End;
      t := t + dt;
      pt[counter] := t;
      Calc_input;
      For i := 1 to num_inps do
      Begin
        pu[i,counter] :=  up[i];
        puc[i,counter] := uc[i];
      End;
      Plant;
      if f_observer = 1 then Observer;
      For i := 1 to j do
      Begin
        px[i,counter] :=  x[i]+x_init[i];
        px_observer[i,counter] := x_observer[i] + x_init[i];
      End;
      counter := counter + 1;
      If(counter <= max_pts) then
      For i := 1 to num_inps do py[i,counter] := y_sys[i];
    End;
    clrscr;
  End;
End; {Procedure Simulate}

{=================================================================}
{                                                                 }
{                          MAIN PROGRAM                           }
{                                                                 }
{=================================================================}

Begin
  rep_flag :=0;
  initialise;
  Enter_sys;
  rep_flag := 1;
  Repeat
    Filter;
    Noise;
    Sim_constants;
    Sim_type;
    Simulate;
    User_graphics;
    Repeat_P;
    initialise;
    counter := 1;
  Until (rep_flag = 0);
End.

{*********************************************************************}
{*****************************File end********************************}
{*Edited last by : D. Ginsberg                                       }
{*Date : 21/08/89                                                    }
```

APPENDIX 4: CONTROL PROGRAM FOR SERVO MOTOR

```
Program PJ417_88 (input,output);

{*************************************************************}
{                                                             }
{CF FILE          : PJ417_88.PAS                              }
{                                                             }
{CC DESCRIPTION   :Process variable, y(t), connected to ADC#0 }
{                  range: -10..+10[Volts] = -2048..+2047      }
{                  [Counts]. Process input, u(t), connected to }
{                  DAC#0.                                     }
{                  range: -2048..+2047[Counts] = -10.+10[Volts]}
{                                                             }
{                  This program is an adaption of the control }
{                  program used to run the EEE417 project in  }
{                  1988.                                      }
{                                                             }
{                  It is a real - time program to implement a }
{                  discrete controller for a servo motor .    }
{                                                             }
{CW WRITTEN BY    :M.Braae, Oct 1987                          }
{                                                             }
{CE EDITED BY     :D.GINSBERG 1988                            }
{                                                             }
{*************************************************************}

{$C-}             {Allow keyboard control of program}
{$I GXINC1.PAS}   {Graphics drivers}
{$I PLOT.PAS}     {Plot drivers for user co-ordinates}
{$I DT2801.PAS}   {DT2801 card analog I/O drivers}
{$I BEEP.PAS}

type
  string20= string[20];
const
  PageLength=24;
  a= -7.27;       {Obtain the system states from the output}
  b= -0.99;
  HeaderFile='PJ417_88.HDG';
  ProcessFile='PJ417_88.DAT';
  StatesFile='PJ417_88.PRM';
  DisplayFile='PJ417_88.PNL';
  HelpFile='PJ417_88.HLP';
  nk=4;mk=4; {Order of numerator & denominator polynomials}
var
  AutoMode, Graphics, Alarm, LogMode, RampMode,
  TestMode, PlantStarted, InitializeController: boolean;
  Key_Pressed, Key, CONTYPE: char;
  RCondition, YCondition, PCONDITION, UCondition: string[2];
  Frame: array[1..25] of string[80];
  FrameLength,
  TimeCounter, ControlCycles, i, j, MaxNumberOfSamples,
  Time, DeltaTime,
  SamplePntr, graph, v, P, SamplingTime, LastTime: integer;
  Data: array[1..4, 1..600] of integer;
  Setpointr, SetpointMin, SetpointMax, SetpointLO, SetpointHI,
  Outputy, OUTPUTP, OutputMin, OutputMax, OutputLO, OutputHI,
  Inputu, InputMin, InputMax, InputLO, InputHI, InputLOLO,
  Speed, POSITION, e, eLast,
  StepSize, Temp,
  Parametery1, PARAMETERY2, ParameterU0,
  Outputy1, OUTPUTY2, T, PNONL, NNONL, K1, K2, X1, X2 : real;
  kn, kd, du, de: array[0..4] of real;

  vsserror,sum,c1,alpha,beta,torf,uprev,vinp : real;
  sum2,sum3,c2,c3,gamma,kappa,               :real;

Procedure PagePause;
begin
  gotoxy(20,25);
```

```
  write('==>> Press any key to continue <<==');
  Beep;
  read(kbd,key);
  ClrScr;
  end;{PagePause proc}

Procedure ReadDisplayFrame;
var
  Filvar: text;
begin
  Assign(Filvar,DisplayFile);
  Reset(Filvar);
  FrameLength:=0;
  while not Eof(Filvar) do begin
    FrameLength:=FrameLength+1;
    readln(Filvar,Frame[FrameLength]);
    end; {while not eof}
  close(Filvar);
  end; {ReadDisplayFrame proc}

Procedure DisplayFrame;
begin
  ClrScr;
  for i:=1 to FrameLength do begin
    writeln(Frame[i]);
    end;
  if InitializeController then begin
    gotoxy(19,15);
    write('[z]');
    end; {if InitializeController}
  end; {DisplayFrame proc}

Procedure Display;
begin
  gotoxy(10,7);if PlantStarted then write('ON ')
                               else write('OFF');
  gotoxy(21,7);write(Inputu:9:1);
  gotoxy(50,7);write(Outputy:9:1);
  GOTOXY(50,11);WRITE(OUTPUTP:9:1);
  gotoxy(12,15);if AutoMode then write('Auto  ')
                            else write('Manual');
  gotoxy(31,15);write(10*ControlCycles*DeltaTime:3);
  gotoxy(50,15);write(Setpointr:9:1);
  gotoxy(12,19);write(SamplePntr:3);
  gotoxy(68,15);write(RCondition);
  gotoxy(68,7);write(YCondition);
  GOTOXY(68,11);WRITE(PCONDITION);
  gotoxy(39,7);write(UCondition);
  end; {Display proc}

Procedure AnalogInput;
begin
  adc_read(0,v);
  adc_read(1,P);
  Speed:=v;
  POSITION:=P;
  OUTPUTP:=P;
  v:=0;
  P:=0;
  end; {AnalogInput}

Procedure AnalogOutput;
begin
  IF V>0 THEN V:=V+ROUND(PNONL);
  IF V<0 THEN V:=V-ROUND(NNONL);
  dac_write(0,v);
  end; {AnalogOutput}

procedure StopPlant;
begin
  v:=0;
  AnalogOutput;
  Inputu:=0;
  AutoMode:=false;
  PlantStarted:=false;
  end; {StopPlant proc}
```

```
procedure StartPlant;
begin
  Inputu:=0;
  PlantStarted:=true;
  gotoxy(28,19);write(' ':40);
  Display;
  v:=round(Inputu);
  AnalogOutput;
  Delay(50);
  AnalogInput;
  {Model;}
  end; {StartPlant proc}

procedure ZeroControllerStates;
begin
  if InitializeController then begin
    eLast:=0;
    for i:=0 to nk do begin
      du[i]:=0;
      end; {for i loop}
    for i:=0 to mk do begin
      de[i]:=0;
      end; {for i loop}
    end; {if InitializeController}
  end; {ZeroControllerStates proc}

Procedure Initialize;
var
  Filvar: text;
  Line: string[80];
  LineCounter: integer;
begin
  set_up; {Initialize DT2801 Analog I/O card}
  StopPlant;
 {==>> Write Introductory Header Page <<==}
  ClrScr;
  ReadDisplayFrame;
  Assign(Filvar,HeaderFile);
  Reset(Filvar);
  LineCounter:=1;
  ClrScr;
  while not Eof(Filvar) do begin
    readln(Filvar,Line);
    writeln(Line);
    LineCounter:=LineCounter+1;
    if LineCounter>PageLength then begin
      LineCounter:=1;
      PagePause;
      end;{if LineCounter>PageLength}
    end;{while not eof}
  close(Filvar);
 {==>> Read Initial Values for Program States <<==}
  Assign(Filvar,StatesFile);
  Reset(Filvar);
  readln(Filvar, LastTime, ControlCycles, MaxNumberOfSamples);
  readln(Filvar, kn[0], kn[1], kn[2], kn[3], kn[4]);
  readln(FilVar, kd[0], kd[1], kd[2], kd[3], kd[4]);
  readln(Filvar, Setpointr, SetpointMin, SetpointMax, SetpointLO, SetpointHI);
  readln(Filvar, Outputy, OutputMin, OutputMax, OutputLO, OutputHI);
  readln(Filvar, Inputu, InputMin, InputMax, InputLO, InputHI);
  readln(Filvar, SamplePntr, StepSize, SamplingTime, DeltaTime);
  readln(Filvar, X_Min, X_Max, X_Tick, X_Axis_Label);
  readln(Filvar, Y_Min, Y_Max, Y_Tick, Y_Axis_Label);
  close(Filvar);
  InitializeController:=true;
  ZeroControllerStates;
  InitializeController:=false;
  AutoMode:=false;
  Graphics:=false;
  TestMode:=false;
  LogMode:=false;
  PlantStarted:=false;
  INCSN:='Y';
  INCDIST:='Y';
  SPNMAX:=0;
```

```
  FSPN:=0.005;
  SVNMAX:=0;
  FSVN:=0.005;
  SENSPN:=0;
  SENSVN:=0;
  PDISTMAX:=0;
  FPDIST:=0.005;
  VDISTMAX:=0;
  FVDIST:=0.005;
  VDIST:=0;
  PDIST:=0;
  TimeCounter:=0;
  PNONL:=140;
  NNONL:=140;
  CONTYPE:='S';
  NSINPS:=0;
  NSINVS:=0;
  NSINPSMAX:=0;
  NSINVSMAX:=0;
  NSINPD:=0;
  NSINVD:=0;
  NSINPDMAX:=0;
  NSINVDMAX:=0;
  K1:=16.1;
  K2:=17.3;
  RCondition:=''; YCondition:='';PCONDITION:=''; UCondition:='';
  Outputy1:=Outputy;
 {==>> Zero Data[1..3, 1..300] Array <<==}
  for i:=1 to 4 do begin
    for j:=1 to 600 do begin
      Data[i,j]:=0;
      end; {for j loop}
    end; {for i loop}
  end; {Initialize proc}

Procedure KeyStatus(Var KeyChar: Char);
  begin
    KeyChar:=' ';
    if KeyPressed then read(kbd, KeyChar);
    end; {KeyStatus proc}

Procedure ValueUpdate(var Value: real; Increment, ValueMin, ValueMax: Real);
begin
  Value:=Value+Increment;
  if Value<ValueMin then Value:=ValueMin;
  if Value>ValueMax then Value:=ValueMax;
  if keypressed then begin
    repeat
      Value:=Value+Increment;
      if Value<ValueMin then Value:=ValueMin;
      if Value>ValueMax then Value:=ValueMax;
      read(kbd,key);
      Display;
      until not keypressed;
    end; {if keypressed}
  end; {ValueUpdate proc}

procedure TimeTrigger;
type
  regtype = record
              ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
              end;
var
  regs: regtype;
begin
  repeat
    Regs.AX:=$2C00;
    MSDOS(Regs);
    Time:=lo(regs.DX); {Centiseconds}
    DeltaTime:=Time-LastTime;
    if DeltaTime < 0 then DeltaTime:=DeltaTime+100
                     else DeltaTime:=DeltaTime+0;
    until DeltaTime>=SamplingTime;
  LastTime:=Time;
  end; {TimeTrigger}
```

```
Procedure TypeParameters;
begin
  ClrScr;
  gotoxy(1,1);
  writeln('                    PROGRAM  PARAMETERS');
  writeln;
  writeln('SAMPLING:');
  writeln('        ControlCycles [ ], T (',ControlCycles:10,') ');
  writeln('        MaxNumberOfSamples (',MaxNumberOfSamples:10,') ');
  writeln('        StepSize (',StepSize:10:1,') ');
  writeln('ALARMS:');
  writeln('        SetpointMin Limit [C] (',SetpointMin:10:0,') ');
  writeln('        SetpointMax Limit [C] (',SetpointMax:10:0,') ');
  writeln('        OutputMin Limit [C] (',OutputMin:10:0,') ');
  writeln('        OutputMax Limit [C] (',OutputMax:10:0,') ');
  writeln('        InputMin Limit [C] (',InputMin:10:0,') ');
  writeln('        InputMax Limit [C] (',InputMax:10:0,') ');
  writeln('GRAPH AXES:');
  writeln('        X_Min [C] (',X_Min:5:0,') ');
  writeln('        X_Max [C] (',X_Max:5:0,') ');
  writeln('        X_Tick [C] (',X_Tick:5:0,') ');
  writeln('        X_Axis_Label (',X_Axis_Label,') ');
  writeln('        Y_Min [C] (',Y_Min:5:0,') ');
  writeln('        Y_Max [C] (',Y_Max:5:0,') ');
  writeln('        Y_Tick [C] (',Y_Tick:5:0,') ');
  writeln('        Y_Axis_Label (',Y_Axis_Label,') ');
  end;{ParamUpdate}

Procedure UpdateParameters;
var
  Buffer: string[80];
begin
  ClrScr;
  repeat
    TypeParameters;
    gotoxy(1,1);
    writeln('           Update  PROGRAM  PARAMETERS');
    writeln;
    writeln;
    write('  Enter ControlCycles [ ], T (',ControlCycles:10,') ');
      readln(ControlCycles);
      if ControlCycles<1 then ControlCycles:=1;
    write('  Enter MaxNumberOfSamples (',MaxNumberOfSamples:10,') ');
      readln(MaxNumberOfSamples);
      if MaxNumberOfSamples<10 then MaxNumberOfSamples:=10;
      if MaxNumberOfSamples>600 then MaxNumberOfSamples:=600;
    write('  Enter StepSize (',StepSize:10:1,') ');
      readln(StepSize);
    writeln;
    write('  Enter SetpointMin Limit [C] (',SetpointMin:10:0,') ');
      readln(SetpointMin);
    write('  Enter SetpointMax Limit [C] (',SetpointMax:10:0,') ');
      readln(SetpointMax);
    write('  Enter OutputMin Limit [C] (',OutputMin:10:0,') ');
      readln(OutputMin);
    write('  Enter OutputMax Limit [C] (',OutputMax:10:0,') ');
      readln(OutputMax);
    write('  Enter InputMin Limit [C] (',InputMin:10:0,') ');
      readln(InputMin);
    write('  Enter InputMax Limit [C] (',InputMax:10:0,') ');
      readln(InputMax);
    writeln;
    write('  Enter X_Min [C] (',X_Min:5:0,') ');
      readln(X_Min);
    write('  Enter X_Max [C] (',X_Max:5:0,') ');
      readln(X_Max);
    write('  Enter X_Tick [C] (',X_Tick:5:0,') ');
      readln(X_Tick);
    write('  Enter X_Axis_Label (',X_Axis_Label,') ');
      readln(Buffer); if Buffer<>'' then X_Axis_Label:=Buffer;
    write('  Enter Y_Min [C] (',Y_Min:5:0,') ');
      readln(Y_Min);
    write('  Enter Y_Max [C] (',Y_Max:5:0,') ');
      readln(Y_Max);
    write('  Enter Y_Tick [C] (',Y_Tick:5:0,') ');
      readln(Y_Tick);
```

```
    write('  Enter Y_Axis_Label (',Y_Axis_Label,') ');
      readln(Buffer); if Buffer<>'' then Y_Axis_Label:=Buffer;
    write('Okay? '); read(Key);
    until upcase(Key)<>'N';
    ClrScr;
  end;{ParamUpdate}

procedure TypeController;
begin
  ClrScr;
  gotoxy(1,1);
  writeln('                    CONTROLLER  CONSTANTS');
  writeln(contype);
  writeln('Numerator:');
  for i:=0 to mk do begin
    writeln('        z^-',i:1,' coeff: (',kn[i]:10:4,') ');
    end; {for i loop}
    writeln;
    writeln('Denominator:');
    for i:=0 to nk do begin
      writeln('        z^-',i:1,' coeff: (',kd[i]:10:4,') ');
      end; {for i loop}
    WRITELN;
    WRITELN('  K1=',K1);
    WRITELN('  K2=',K2);
  WRITELN;
  WRITELN('Positive Nonlinearity =',PNONL);
  WRITELN('Negative Nonlinearity =',NNONL);
  end; {TypeController}

procedure UpdateController;
begin
  ClrScr;
  repeat
    TypeController;
    gotoxy(1,1);
    writeln('           Update  CONTROLLER  CONSTANTS');
    write('Output (O) ,VSS(V) State F/B (sf) or State (S)?');
    READLN(CONTYPE);
    CONTYPE:=UPCASE(CONTYPE);
    gotoxy(1,1);
    IF CONTYPE='O' THEN BEGIN
      WRITELN;
      WRITELN;
      WRITELN;
    for i:=0 to mk do begin
      write('  Enter z^-',i:1,' coeff: (',kn[i]:10:4,')');
      readln(kn[i]);
      end; {for i loop}
    writeln;
    writeln;
    for i:=0 to nk do begin
      write('  Enter z^-',i:1,' coeff: (',kd[i]:10:4,')');
      readln(kd[i])
      end; {for i loop}
    WRITELN;
    WRITELN;
    WRITELN;
    WRITELN;
    END; {CONTYPE = OUTPUT}
    IF CONTYPE <> 'O' THEN BEGIN
      WRITELN;
      WRITELN;
      WRITELN;
      WRITELN;
    FOR I:=0 TO MK DO WRITELN;
    WRITELN;
    WRITELN;
    FOR I:=0 TO NK DO WRITELN;
    WRITE('  ENTER K1=(',K1:10:4,')');
    READLN(K1);
    WRITE('  ENTER K2=(',K2:10:4,')');
    READLN(K2);
    WRITELN;
    END; {CONTYPE = STATE}
```

```
    If contype = 'V' then
    Begin
      clrscr;
      write( Enter c1 ');
      readln(c1);
      write( Enter c2 ');
      readln(c2);
      write( Enter c3 ');
      readln(c3);
      write( Enter alpha ');
      readln(alpha);
      write( Enter beta ');
      readln(beta);
      write( Enter gamma ');
      readln(gamma);
      write( Enter kappa ');
      readln(kappa);
      write( Enter torf ');     {FILTER CONSTANT }
      readln(torf);
      write( Enter initial Uprev ');
      readln(Uprev);
      write( Enter Ro value ');
      readln(Ro);
      write( Enter k1 value ');
      readln(k1);
      write( Enter k2 value ');
      readln(k2);
    End; {contype = VSS}
    WRITE('  Enter positive Nonlinearity =(',PNONL:10:4,')');
    READLN(PNONL);
    WRITE('  Enter Negative Nonlinearity =(',NNONL:10:4,')');
    READLN(NNONL);
    write('Okay? '); read(Key);
    until upcase(Key)<>'N';
    ClrScr;
  end; {UpdateController proc}

PROCEDURE NOISEDIST;
BEGIN
ClrScr;
gotoxy(1,1);
INCSN:='Y';
INCDIST:='Y';
WRITE('Include Sensor Noise? (Y/N : Def = Y)');
READLN(INCSN);
INCSN:=UPCASE(INCSN);
IF INCSN<>'N' THEN BEGIN
  INCSN:='Y';
  WRITELN('Position Sensor Noise Status');
  WRITE('  Maximum Deviation = (',SPNMAX:10:4,')');
  READLN(SPNMAX);
  WRITE('  Sensor Noise Frequency(Hz) = (',FSPN:10:4,')');
  READLN(FSPN);
  IF FSPN=0.0 THEN FSPN := 0.001;
  WRITELN;
  WRITELN('Velocity Sensor Noise Status');
  WRITE('  Maximum Deviation = (',SVNMAX:10:4,')');
  READLN(SVNMAX);
  WRITE('  Sensor Noise Frequency(Hz) = (',FSVN:10:4,')');
  READLN(FSVN);
  IF FSVN=0.0 THEN FSVN := 0.001;
END; {INCSN = Y}
WRITELN;
WRITELN;
WRITE('Include Plant Disturbances? (Y/N : Def = Y)');
READLN(INCDIST);
INCDIST:=UPCASE(INCDIST);
IF INCDIST<>'N' THEN BEGIN
  INCDIST:='Y';
  WRITELN('Position Disturbance Status');
  WRITE('  Maximum Deviation = (',PDISTMAX:10:4,')');
  READLN(PDISTMAX);
  WRITE('  Disturbance Frequency(Hz) = (',FPDIST:10:4,')');
  READLN(FPDIST);
  IF FPDIST=0.0 THEN FPDIST := 0.001;
  WRITELN;
```

```
  WRITELN('Velocity Disturbance Status');
  WRITE('  Maximum Deviation = (',VDISTMAX:10:4,')');
  READLN(VDISTMAX);
  WRITE('  Disturbance Frequency(Hz) = (',FVDIST:10:4,')');
  READLN(FVDIST);
  IF FVDIST=0.0 THEN FVDIST := 0.001;
END; {INCDIST = Y}
END; {NOISEDIST PROC}

Procedure ServiceKeyboard(var Key_Pressed: char);
var
  Filvar: text;
  Line: string[80];
  LineCounter: integer;
begin
  read(kbd,Key_Pressed);
  case Key_Pressed of
    '?': begin
      Assign(Filvar,HelpFile);
      Reset(Filvar);
      ClrScr;
      LineCounter:=1;
      repeat
        readln(Filvar,Line);
        writeln(Line);
        LineCounter:=LineCounter+1;
        if LineCounter>PageLength then begin
          LineCounter:=1;
          PagePause;
          end; {if LineCounter>PageLength}
        until (Line='.') or (Eof(Filvar));
      close(Filvar);
      PagePause;
      DisplayFrame;
      end;{option ?}
    'H': begin
      Assign(Filvar,HelpFile);
      Reset(Filvar);
      ClrScr;
      LineCounter:=1;
      while not Eof(Filvar) do begin
        readln(Filvar,Line);
        writeln(Line);
        LineCounter:=LineCounter+1;
        if LineCounter>PageLength then begin
          LineCounter:=1;
          PagePause;
          end;{if LineCounter>PageLength}
        end;{while not eof}
      close(Filvar);
      PagePause;
      DisplayFrame;
      end;{option H}
    'A': begin
      if PlantStarted then begin
        AutoMode:=not AutoMode;
        if AutoMode then ZeroControllerStates;
        end; {if PlantStarted}
      end; {option A}
    'D': begin
      TypeParameters;
      PagePause;
      DisplayFrame;
      end; {option D}
    'E': begin
      UpdateParameters;
      DisplayFrame;
      end;{option 'E'}
    'd': begin
      TypeController;
      PagePause;
      DisplayFrame;
      end; {option 'd'}
    'e': begin
      UpdateController;
      DisplayFrame;
```

```
  end; {option 'e'}
'r':begin
  ValueUpdate(Setpointr,+1, SetpointMin, SetpointMax);
  end; {option r}
'R': begin
  ValueUpdate(Setpointr,+20, SetpointMin, SetpointMax);
  end; {option R}
'f':begin
  ValueUpdate(Setpointr,-1, SetpointMin, SetpointMax);
  end; {option f}
'F': begin
  ValueUpdate(Setpointr,-20, SetpointMin, SetpointMax);
  end; {option F}
'u':begin
  if not AutoMode then
    if PlantStarted then ValueUpdate(Inputu,+1, InputMin, InputMax);
  end; {option u}
'U':begin
  if not AutoMode then
    if PlantStarted then ValueUpdate(Inputu,+20, InputMin, InputMax);
  end; {option U}
'j': begin
  if not AutoMode then
    if PlantStarted then ValueUpdate(Inputu,-1, InputMin, InputMax);
  end; {option j}
'J': begin
  if not AutoMode then
    if PlantStarted then ValueUpdate(Inputu,-20, InputMin, InputMax);
  end; {option J}
'I': begin
  PlantStarted:=not PlantStarted;
  if PlantStarted then StartPlant
                  else StopPlant;
  end; {option I}
'G': begin
  graphics:=true;
  end; {option G}
'L': begin
  LogMode:=not LogMode;
  if LogMode then SamplePntr:=0;
  end; {option L}
's': begin
  if AutoMode then Setpointr:=Setpointr-StepSize
    else Inputu:=Inputu-StepSize;
  end; {option s}
'S': begin
  if AutoMode then Setpointr:=Setpointr+StepSize
    else Inputu:=Inputu+StepSize;
  end; {option s}
'T': begin
  TestMode:=true;
  AutoMode:=false;
  LogMode:=true;
  RampMode:=false;
  Setpointr:=500;
  SamplePntr:=1;
  MaxNumberOfSamples:=600;
  Y_Min:=-100;
  Y_Max:=1500;
  X_Max:=600;
  if PlantStarted then StopPlant;
  StartPlant;
  gotoxy(28,19);
  write('ACCEPTANCE TEST RUNNING          ':40);
  end; {option T}
'Z': begin
  InitializeController:=true;
  gotoxy(19,15);write('[z]');
  end; {option Z}
'N': BEGIN
  NOISEDIST;
  NSINPSMAX:=ROUND(1/(FSPN*SAMPLINGTIME));
  NSINVSMAX:=ROUND(1/(FSVN*SAMPLINGTIME));
  NSINPDMAX:=ROUND(1/(FPDIST*SAMPLINGTIME));
  NSINVDMAX:=ROUND(1/(FVDIST*SAMPLINGTIME));
  DISPLAYFRAME;
```

```
      END; {OPTION N}
      'n': BEGIN
       NOISEDIST;
       NSINPSMAX:=ROUND(1/(FSPN*SAMPLINGTIME));
       NSINVSMAX:=ROUND(1/(FSVN*SAMPLINGTIME));
       NSINPDMAX:=ROUND(1/(FPDIST*SAMPLINGTIME));
       NSINVDMAX:=ROUND(1/(FVDIST*SAMPLINGTIME));
       DISPLAYFRAME;
       END; {OPTION n}
    end;{case Key_Pressed}
  end;{ServiceKeyboard}

Procedure LimitAlarms;
begin
  UCondition:='  '; YCondition:='  '; PCONDITION:='  '; RCondition:='  ';
  Alarm:=false;
  if PlantStarted then begin
    if Inputu<InputLO then begin
      UCondition:='LO'; Alarm:=FALSE;
      end;
    if Inputu>InputHI then begin
      UCondition:='HI'; Alarm:=FALSE;
      end;
    v:=round(Inputu);
    if Outputy<OutputLO then begin
      YCondition:='LO'; Alarm:=FALSE;
      end;
    if Outputy>OutputHI then begin
      YCondition:='HI'; Alarm:=FALSE;
      end;
    end; {if PlantStarted;}
  if Setpointr<SetpointLO then begin
    RCondition:='LO'; Alarm:=FALSE;
    end;
  if Setpointr>SetpointHI then begin
    RCondition:='HI'; Alarm:=FALSE;
    end;
  if Alarm then NOSOUND
           else NoSound;
  end; {LimitAlarm proc}


Procedure ControlAlgorithm;
begin
IF CONTYPE='O' THEN BEGIN
  e:=Setpointr-(OUTPUTP + SENSPN);
  de[0]:=e-eLast;
  du[0]:=0;
  for i:=1 to nk do begin
    du[0]:=du[0]-kd[i]*du[i];
    end; {for i loop}
  for i:=0 to mk do begin
    du[0]:=du[0]+kn[i]*de[i];
    end; {for i loop}
  du[0]:=du[0]/kd[0];
  for i:=0 to nk-1 do begin
    du[nk-i]:=du[nk-1-i];
    end; {for i loop}
  for i:=0 to mk-1 do begin
    de[mk-i]:=de[mk-1-i];
   end; {for i loop}
  Inputu:=Inputu+du[0];
  eLast:=e;
END; {CONTYPE = OUTPUT}
IF CONTYPE <> 'O' THEN BEGIN
  X1 := (OUTPUTY + SENSVN)/a;
  X2 := (OUTPUTP + SENSPN)/(a*b);
  DU[0] := (SETPOINTR - ROUND(K1*X1 + K2*X2)) - INPUTU;
  INPUTU := INPUTU + DU[0];
END; {CONTYPE = STATE}

If(contype = 'V')then
Begin
  x1 := outputy/b;
  x2 := outputp/a;
  vsserror := (x2 - setpointr/a);
```

```
  kappa := x1*x1 + vsserror*vsserror;
  sum  := vsserror*(vsserror*c1 + x1);
  sum2 := vsserror*(vsserror*c2 + x1);
  If(kappa > Ro*Ro)then
  Begin
    If(sum > 0.0 )then vinp := alpha*vsserror;
    If(sum < 0.0 )then
    Begin
      If(sum2 > 0.0 )then Vinp := beta*vsserror;
      If(sum2 < 0.0 )then Vinp := gamma*vsserror;
    End;
  End;
  If(kappa < Ro*Ro)then vinp := (setpointr - k1*x1 - k2*x2);
  Inputu := Uprev*(exp(-deltatime/torf))+(1-exp(-deltatime/torf))*vinp;
  Uprev := inputu;
End;{Contype = VSS}
end; {ControlAlgorithm}

Procedure DataStorage;
begin
  SamplePntr:=SamplePntr+1;
  if SamplePntr>MaxNumberOfSamples then begin
    SamplePntr:=MaxNumberOfSamples;
    LogMode:=false;
    RampMode:=false;
    TestMode:=false;
    end; {if SamplePntr...}
  Data[1,SamplePntr]:=0;
  if AutoMode then Data[1,SamplePntr]:=round(Setpointr);
  Data[2,SamplePntr]:=0; {round(Outputy);}
  DATA[3,SamplePntr]:=ROUND(OUTPUTP);
  Data[4,SamplePntr]:=round(Inputu);
  end; {DataStorage}

Procedure DataPlot;
begin
  R_Start;gmode;outline;
 {Plot background}
  gmode;ClrScn;Outline;
  HText(10,20,'M.SC. CONTROL ENGINEERING THESIS 88');
  Htext(480,20,'D.GINSBERG');
  Htext(10,335,'SampleTime =      [ms]');
  write_num(10*DeltaTime,2,4,140,330);
  Htext(250,335,'ControlTime =      [ms]');
  write_num(10*ControlCycles*DeltaTime,2,4,390,330);
  Htext(490,335,'Anton de Waal');
    R_Border(X_Min,Y_Min,X_Max,Y_Max);
  R_Axes(X_Min,X_Max,X_Tick,Y_Min,Y_Max,Y_Tick);
 {Plot data}
  for graph:=1 to 4 do begin
    R_move(1,Data[graph,1]);
    for i:=2 to MaxNumberOfSamples do begin
      R_Draw(i,Data[graph,i-1]);
      R_Draw(i,Data[graph,i]);
      end; {for i loop}
    end; {for graph loop}
    repeat until keypressed;
    tmode;ClrScr;DisplayFrame;
    Graphics:=false;
  end; {DataPlot proc}

Begin {PJ417_88 Main Program}
  ClrScr;
  Initialize;
  repeat
    DisplayFrame;
    repeat
      NoSound;
      if keypressed then ServiceKeyboard(Key);
      AnalogInput;
      OUTPUTY:=Speed;
      OUTPUTP:=POSITION;
      if TestMode then begin
        if SamplePntr = 50 then begin
          AutoMode:=true;
          ZeroControllerStates;
```

```
        end; {if SamplePntr = 150}
      if SamplePntr = 200 then Setpointr:=1000;
      if SamplePntr = 400 then RampMode:=true;
      if SamplePntr = 600 then begin
        TestMode:=false;
        graphics:=true;
        gotoxy(28,19);write(' ':40);
        end; {if SamplePntr > 600}
      if RampMode then Setpointr:=Setpointr-3;
      end; {if TestMode}
    TimeCounter:=TimeCounter+1;
    if TimeCounter>=ControlCycles then begin
      TimeCounter:=0;
      if AutoMode then ControlAlgorithm;
      LimitAlarms;
      V := ROUND(INPUTU);
      AnalogOutput;
      end; {if TimeCounter>...}
    if LogMode then DataStorage;
    if not Graphics then Display
    else DataPlot;
    TimeTrigger;
    until upcase(Key)='Q';
  gotoxy(1,25);write('Are you sure (Y/N)?');readln(Key)
  until Upcase(Key)='Y';
  StopPlant;
end. {Main Program}
```

APPENDIX 5:CONTROL PROGRAM FOR FLOTATION RIG

```
C
C     FILE             : RIGAN.FOR
C ***************************************************************************
CN    MODULE NAME      :
CA    FUNCTION         :
CS    CALL SEQUENCE    :
CI    INPUT PARAMETERS : None.
C
CO    OUTPUT PARAMETERS: None.
C
CG    GLOBAL VARIABLES : None.
CM    MODULES CALLED   : DOMENU (asm),inisys (for), inmenu (for),
C                        INTONE (asm), RETONE (asm),
C                        RSTSCR (asm), WIPSCR(asm), wrtitl (for), WRTSTR (asm),
C                        REDERR (asm).
C
CE    ERROR CONDITIONS : INT10 GRAPHIX package not resident in the system,
C                        HERCULES card not installed.
C
CC    COMMENTS         :
C
C                        following functions :-
C                          i) Initialises the 2 graphics pages.
C                         ii) Initialises the system variables.
C                        iii) Initialises the system timer interrupt for
C                             error tone operation.
C
C                        This package requires the following in order to run :-
C                          i) IBM PC - XT/AT (compatable, preferably an AT)
C                         ii) PC to contain at least 512 K of RAM.
C                        iii) PC to contain a HERCULES graphics card.
C                         iv) PC to have access to at least one disk drive.
C                          v) INT10 GRAPHIX package is also rquired by the CAD
C                             package.
C
C ***************************************************************************

      program RIGAN
      implicit integer*2 (D)

      integer*2 opt2

      call INTONE()
      call inmenu()
      call init()
99    continue
      call wrtitl(0,250,35,23,'Flow Rig Data Analysis.')
      opt2 = DOMENU(2)
      call WRTSTR(0,250,35,28,'                              ')
```

```
      if (opt2.eq.1) then
         call rigan2()
      elseif (opt2.eq.2) then
         call mcont()
      endif
      if (opt2.ne.3) goto 99
      call REDERR()
      call RSTSCR()
      call RETONE()
      stop
      end

C *************************************************************************
      subroutine rigan2()
      implicit integer*2 (D)

      integer*2 opt1

99    continue
      call WIPSCR(0)
      call wrtitl(0,250,35,20,'Flow Rig Step Tests.')
      opt1 = DOMENU(1)
      call WRTSTR(0,250,35,28,'                                ')
      if (opt1.eq.1) then
         call loadf()
      elseif (opt1.eq.2) then
         call savef()
      elseif (opt1.eq.3) then
         call vwdata()
      elseif (opt1.eq.4) then
         call rigstp()
      elseif (opt1.eq.5) then
         call pldata()
      elseif (opt1.eq.6) then
         call mix2()
      elseif (opt1.eq.7) then
         call opts()
      elseif (opt1.eq.8) then
         call daoff()
      elseif (opt1.eq.9) then
         call setda()
      elseif (opt1.eq.10) then
         call docont()
      elseif (opt1.eq.11) then
         call nodly()
      endif
      if (opt1.ne.0) goto 99
      call WIPSCR(0)
      return
      end
C *************************************************************************
CN    MODULE NAME      : wrtitl
CA    FUNCTION         : To write a title on a page.
CS    CALL SEQUENCE    : call wrtitl(page,x,y,length,string)
```

```
CI    INPUT PARAMETERS : page - (integer*2) The page to which the title is to
C                                          be written.
C                           x,y - (integer*2) The x,y co-ords of the title.
C                        length - (integer*2) The length of the title.
C                        string - (character*length) The title string.
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES : None.
CM    MODULES CALLED   : DLINE (asm), MOVE (asm), WRTSTR (asm).
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS         : Writes out the string and the draws a line underneath
C                        the string for the length of the string.
C
C ***************************************************************************
      subroutine wrtitl(pg,x,y,len,str)
      integer*2 pg,x,y,len
      character*80 str
      external DLINE
      external MOVE
      external WRTSTR
      call WRTSTR(pg,x,y,len,str)
      call MOVE(x,y+3)
      call DLINE((len*9+x),y+3)
      return
      end

C ***************************************************************************
      subroutine fixdat(percnt,data,samps,instr)
      real*4 percnt(4000)
      integer*2 data(4000,8)
      integer*2 samps,instr
      integer*2 i
      do 299 i =1,samps
         percnt(i) = real(data(i,instr))/40.95
299   continue
      return
      end

C ***************************************************************************
      subroutine init()
$include: 'rigdat.inc'
$include: 'multi.inc'
$include: 'setp.inc'
      integer*2 i,j

      samps  = 800
      tankno = 0
      stepno = 100
      hertz  = 1.0
      step = 0.0
      commt = '
      do 799 i = 1,4
         invalv(i) = 80.0
799   continue
```

```
      step = 10.0
      do 798 i = 1,4000
         do 797 j = 1,8
            instru(i,j) = 0
797      continue
798   continue

      setp(1) = 3000
      setp(2) = 2200
      setp(3) = 2900
c2000
      setp(4) = 3200
c2900
      setpnt(1) = 3000
      setpnt(2) = 2200
      setpnt(3) = 2900
c2000
      setpnt(4) = 3200
c2900
      uptr(1,1,1) = 1
      uptr(1,1,2) = 1
      uptr(1,2,1) = 1
      uptr(1,2,2) = 1
c      uptr(1,2,2) = 20
      uptr(1,3,1) = 1
      uptr(1,3,2) = 1
      uptr(1,4,1) = 1
      uptr(1,4,2) = 1

      uptr(2,1,1) = 1
      uptr(2,1,2) = 1
      uptr(2,2,1) = 1
      uptr(2,2,2) = 1
      uptr(2,3,1) = 1
      uptr(2,3,2) = 1
c      uptr(2,3,2) = 20
      uptr(2,4,1) = 1
      uptr(2,4,2) = 1

      uptr(3,1,1) = 1
      uptr(3,1,2) = 1
      uptr(3,2,1) = 1
      uptr(3,2,2) = 1
c      uptr(3,2,2) = 32
      uptr(3,3,1) = 1
      uptr(3,3,2) = 1
      uptr(3,4,1) = 1
      uptr(3,4,2) = 1

      uptr(4,1,1) = 1
      uptr(4,1,2) = 1
c      uptr(4,1,2) = 27
      uptr(4,2,1) = 1
      uptr(4,2,2) = 1
```

```
c      uptr(4,2,2) = 70
       uptr(4,3,1) = 1
       uptr(4,3,2) = 1
c      uptr(4,3,2) = 10
       uptr(4,4,1) = 1
       uptr(4,4,2) = 1

       pi(1,1,1) =  7.5
       pi(1,1,2) =  1.2
       pi(1,2,1) =  -6.0*0.01
       pi(1,2,2) =  -2.6*0.01
       pi(1,3,1) =  -0.06*0.01
       pi(1,3,2) =  2.6*0.01
       pi(1,4,1) =  -0.08*0.01
       pi(1,4,2) =  -0.95*0.01

       pi(2,1,1) = 6.0*0.01
       pi(2,1,2) = -0.04*0.01
       pi(2,2,1) =  55.0
       pi(2,2,2) =  5.0
       pi(2,3,1) = 0.6*0.01
       pi(2,3,2) = -0.25*0.01
       pi(2,4,1) = 1.0*0.01
       pi(2,4,2) = 0.2*0.01

       pi(3,1,1) = -0.3*0.01
       pi(3,1,2) = -0.7*0.01
       pi(3,2,1) = -1.2*0.01
       pi(3,2,2) = -0.03*0.01
       pi(3,3,1) =  2.4
       pi(3,3,2) =  0.3
       pi(3,4,1) = 0.065*0.01
       pi(3,4,2) = 0.11*0.01

       pi(4,1,1) = -1.2*0.01
       pi(4,1,2) = -1.0*0.01
       pi(4,2,1) = -6.0*0.01
       pi(4,2,2) =  1.8*0.01
       pi(4,3,1) = -0.5*0.01
       pi(4,3,2) = -4.5*0.01
       pi(4,4,1) =  6.0
       pi(4,4,2) =  2.0

c      pi(1,1,1) =  17.0
c      pi(1,1,2) =  1.6
c      pi(1,2,1) = -16.0
c      pi(1,2,2) = -4.0
c      pi(1,3,1) = -2.0
c      pi(1,3,2) =  2.0
c      pi(1,4,1) = -15.0
c      pi(1,4,2) = -0.95

c      pi(2,1,1) =  7.0
c      pi(2,1,2) = -0.04
```

```
c       pi(2,2,1) = 110.0
c       pi(2,2,2) = 0.6
c       pi(2,3,1) = 1.0
c       pi(2,3,2) = -0.25
c       pi(2,4,1) = 1.5
c       pi(2,4,2) = 0.22

c       pi(3,1,1) = -0.5
c       pi(3,1,2) = -0.67
c       pi(3,2,1) = -4.0
c       pi(3,2,2) = -0.9
c       pi(3,3,1) =  4.5
c       pi(3,3,2) =  0.84
c       pi(3,4,1) =  0.1
c       pi(3,4,2) =  0.11

c       pi(4,1,1) = -2.0
c       pi(4,1,2) = -1.0
c       pi(4,2,1) = -15.0
c       pi(4,2,2) =  3.0
c       pi(4,3,1) = -5.0
c       pi(4,3,2) = -4.0
c       pi(4,4,1) =  9.0
c       pi(4,4,2) =  3.0

        sampl = 1000
        stank = 0
        steplv = 410
        return
        end
C *****************************************************************************
        subroutine daoff()
        implicit integer*2 (O,R)
        integer*2 i,err

        err = RESTDA()
        if (err.eq.04) then
           call OUTP(548,11)
        endif
        do 498 i = 1,4
           err = OUTDA((i-1),0)
498     continue
        return
        end
C *****************************************************************************
        subroutine setda()
        implicit integer*2 (O,R)
$include: 'rigdat.inc'
        integer*2 i,err,val
        err = RESTDA()
        if (err.eq.04) then
           call OUTP(548,11)
        endif
        do 497 i = 1,4
```

```
         val = int((100.0 - invalv(i))*40.95)
         err = OUTDA((i-1),val)
497   continue
      return
      end
C *****************************************************************************
CH    REVISION HISTORY :
C     VERSION       BY                DATE        COMMENT
C     0.00          Ian Fisher        23-11-87    Finally commented.
C     FILEND          :
C
C     FILE            : DODIR
C *****************************************************************************
CN    MODULE NAME     : dodir
CA    FUNCTION        : To print out a default directory.
CS    CALL SEQUENCE   : call dodir(pg,ylimit,len,str)
CI    INPUT PARAMETERS :     pg - (INTEGER*2) The page to which the directory
C                                             will be written.
C                        ylimit - (INTEGER*2) The upper Y limit on the screen.
C                           len - (INTEGER*2) Length of default search string.
C                           str - (CHARACTER*len) The search string.
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES : None.
CM    MODULES CALLED   : FFIRST (asm), FNEXT (asm), WRTSTR (asm).
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS        : Just prints out as many directory entries that can
C                       be found or fitted onto the screen.
C
C *****************************************************************************
      subroutine dodir(pg,ylimit,dirlen,str)
      implicit integer*2 (F,G,L)
      integer*2 pg,ylimit,dirlen
      character*25 str
      external COPYST
      external DLINE
      external FFIRST
      external FNEXT
      external GPAGE
      external MOVE
      external prtnum
      external RJUST
      external wrtitl
      external WRTSTR
      external RSTERR
      external GETERR
      integer*2 len,len2,find,xpos,ypos,fcount,count2,pos,derr
      character*15 retstr

      call GPAGE(pg)
      call MOVE(4,ylimit)
      call DLINE(714,ylimit)
      call wrtitl(pg,20,ylimit+15,15,'Search string :')
      call RJUST(dirlen,str)
```

```
      len = LENSTR(dirlen,str)
      if (len.gt.36) then
         len2 = 36
      else
         len2 = len
      endif
      call WRTSTR(0,165,ylimit+15,len2,str)
      fcount = 0
      xpos = 20
      ypos = ylimit+32
      call RSTERR()
      find = FFIRST(len,str,len2,retstr)
      if (find.eq.0) then
102      continue
         fcount = fcount + 1
         call WRTSTR(pg,xpos,ypos,len2,retstr)
         xpos = xpos + 140
         if (xpos.gt.680) then
            xpos = 20
            ypos = ypos + 15
         endif
         if (ypos.gt.270) goto 104
         find = FNEXT(len2,retstr)
         if (find.eq.0) goto 102
104      continue
         count2 = fcount
106      continue
         find = FNEXT(len2,retstr)
         if (find.eq.0) count2 = count2 + 1
         if (find.eq.0) goto 106
         call prtnum(0,495,ylimit+15,1,4,'(i2)',2,fcount)
         call WRTSTR(0,525,ylimit+15,10,'printed / ')
         call prtnum(0,615,ylimit+15,1,4,'(i3)',3,count2)
         call WRTSTR(0,650,ylimit+15,6,'found.')
      else
         derr = GETERR()
         if (derr.ne.-1) then
            call prderr(pg,40,ylimit+40,derr)
         else
            call WRTSTR(pg,40,ylimit+40,15,'No files found.')
         endif
      endif
      return
      end

C ******************************************************************************
CH    REVISION HISTORY :
C     VERSION       BY                DATE        COMMENT
C     FILEND          :
```

```
      subroutine docont()
      implicit integer*2 (I,O,R)
$include: 'keys.inc'
$include: 'rigdat.inc'
$include: 'pass.inc'
$include: 'setp.inc'
      integer*2 condat(8)
      real*4 intgrl(4),gain(4),err,ival,t1,errs(4)
      integer*2 key,i,j,k,val,daerr,t2
      call WIPSCR(0)
      call wrtitl(0,200,30,33,'Find Steady State Valve Settings.')
      call wrtitl(0,140,60,9,'Valve [%]')
      call wrtitl(0,270,60,8,'Level SP')
      call wrtitl(0,360,60,5,'Level')
      call WRTSTR(0,30,80,12,'Recleaner : ')
      call WRTSTR(0,30,95,12,'Cleaner   : ')
      call WRTSTR(0,30,110,12,'Scavenger : ')
      call WRTSTR(0,30,125,12,'Rougher   : ')
      call prtnum(0,270,80,1,4,'(i4)',4,setpnt(1))
      call prtnum(0,270,95,1,4,'(i4)',4,setpnt(2))
      call prtnum(0,270,110,1,4,'(i4)',4,setpnt(3))
      call prtnum(0,270,125,1,4,'(i4)',4,setpnt(4))
      gain(1) = 4.0
      gain(2) = 6.0
      gain(3) = 5.0
      gain(4) = 2.5
      ival = 10.0

      do 299 i = 1,4
         intgrl(i) = 2000.0
         uuval(i) = 1
299   continue
      i = 600
      key = INKEY(2)
      daerr = RESTDA()
      if (daerr.eq.04) then
         call OUTP(548,11)
      endif
199   continue
      call iniad(condat)
      k = 4
      do 99 j = 1,4
         call prtnum(0,360,(80+(j-1)*15),1,4,'(i4)',4,condat(j+4))
         err = real(setpnt(j) - condat(j+4))
         errs(k) = err
         k = k - 1
         if ((uuval(j).gt.0).and.(uuval(j).lt.4095)) then
            intgrl(j) = intgrl(j) + err/ival
         endif
         uuval(j) = int(gain(j)*err + intgrl(j))
         if (uuval(j).lt.1) then
            uuval(j) = 0
         elseif (uuval(j).gt.4094) then
            uuval(j) = 4095
         endif
         t1 = (4095.0 - real(uuval(j)))/40.95
         call prtnum(0,140,(80+(j-1)*15),3,7,'(f6.1)',6,t1)
99    continue
      call setdly(errs)
      do 498 j = 1,4
         daerr = OUTDA((j-1),uuval(j))
498   continue
      key = INKEY(4)
      if (key.ne.esck) goto 199
      do 500 i = 1,4
         invalv(i) = int(real((4095-uuval(i)))/40.95)
500   continue
      return
      end

C *************************************************************************
      subroutine iniad(condat)
      implicit integer*2 (I,O,R)
      integer*2 condat(8)
$include: 'rigdat.inc'
      integer*4 temper,sams
```

```
      integer*2 i,j,err,temp,clklo,clkhi,samlo,samhi
      integer*2 basadr,comreg,stareg,datreg
      integer*2 waitcm,writwt,readwt,crdclr,crdclk,crdpar,crdrd,crdadi
      integer*2 crdstp,crderr

      basadr = 748
      comreg = basadr + 1
      stareg = basadr + 1
      datreg = basadr
      waitcm = 4
      writwt = 2
      readwt = 5
      crdclr = 1
      crdclk = 3
      crdpar = 13
      crdrd  = 14
      crdadi = 12
      crdstp = 15
      crderr = 2

      call OUTP(comreg,crdstp)
      temp = INP(datreg)
      call crdrdy()
      call OUTP(comreg,crdclr)
      call crdrdy()
      call OUTP(comreg,crdclk)
      call WAITDT(stareg,writwt,0)
      temper = (int (1.0/(8.0*1.5*0.0000025))) + 1
      call CONVRT(clklo,clkhi,temper)
      call OUTP(datreg,clklo)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,clkhi)
      call crdrdy()
      call OUTP(comreg,crdpar)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,0)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,0)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,7)
      call WAITDT(stareg,writwt,0)
      sams = 8
      call CONVRT(samlo,samhi,sams)
      call OUTP(datreg,samlo)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,samhi)
      call crdrdy()
      call OUTP(comreg,crdrd)
      do 99 j = 1,8
         call WAITDT(stareg,readwt,1)
         condat(j) =  (int (INP(datreg)))
         call WAITDT(stareg,readwt,1)
         condat(j) =  (int(INP(datreg)*256)) + condat(j)
99    continue
      return
      end
C *****************************************************************************
C *****************************************************************************
C *****************************************************************************
C
C     FILE             : DOHELP.FOR
C *****************************************************************************
CN    MODULE NAME      : DOHELP
CA    FUNCTION         : To initiate the help facility.
CS    CALL SEQUENCE    : call dohelp()
CI    INPUT PARAMETERS : None.
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES : keys.inc
CM    MODULES CALLED   : GETPG (asm), GETWPG (asm), GETATT (asm), FLIPG1 (asm),
C                        DISP  (asm),  LEVEL  (asm),  GPAGE (asm),  BOX  (asm),
C                        prhelp (for), ERTONE (asm), WRTSTR (asm), INKEY2 (asm)
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS         : The routine  first  saves  all  the attributes of the
C                        current system (eg. displayed page, write page, write
C                        attributes) and then calls a routine to dump graphics
```

```
      implicit integer*2 (I)
      integer*2 funit
$include: 'keys.inc'
      integer*2 key,i,chkio,sect,pg,maxpg,count
      character*1 chkch

      rewind funit
      read(funit,'(a)',iostat=chkio,err=300) chkch
      if (chkch.eq.'*') then
         backspace funit
         read(funit,102,iostat=chkio,err=300) chkch,sect,pg,maxpg
102      format(a1,i4,i4,i4)
         backspace funit
         call prpage(sect,pg,funit)
         count = 1
         call blksec(count)
99       continue
         key = INKEY2()
         if (key.eq.upk) then
            call blksec(count)
            count = count - 1
            if (count.lt.1) count = 10
            call blksec(count)
         elseif (key.eq.downk) then
            call blksec(count)
            count = count + 1
            if (count.gt.10) count = 1
            call blksec(count)
         elseif (key.eq.retk) then
            call drvhlp(count,key,funit)
            if (key.ne.59) then
               rewind funit
               call prpage(sect,pg,funit)
               call blksec(count)
            endif
         endif
         if ((key.ne.esck).and.(key.ne.59)) goto 99
      endif
300   continue
      if (chkio.ne.0) then
         call ERTONE()
      endif
      return
      end

C ***************************************************************************
CN    MODULE NAME       : PRPAGE
CA    FUNCTION          : To print a single page of help information.
CS    CALL SEQUENCE     : call prpage(sect,pg,funit)
CI    INPUT PARAMETERS :
C                 sect - (integer*2) The section number of the help to print.
C                   pg - (integer*2) The page number of the help to print.
C                funit - (integer*2) The unit  number  for  the source of help
C                        information.
C
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES : None.
CM    MODULES CALLED   : WRTSTR (asm), findpg (for)
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS          : The routine prints out the  data from the current
C                         file pointer until a 'help  file control char' is
C                         encountered.
C                         The routine does perform a  'backspace' operation
C                         before exit; thus the file pointer is pointing at
C                         the next page control char.
C ***************************************************************************
      subroutine prpage(sect,pg,funit)
      integer*2 sect,pg,funit

      character*1 chkch
      character*78 hlpstr,cll
      integer*2 tsect,tpg,tmax,chkio,xpos,ypos
      call findpg(sect,pg,funit)
      hlpstr = ' '
      cll = ' '
```

```
      xpos = 10
      ypos = 17
99    continue
      read(funit,100,iostat=chkio,err=300) chkch,hlpstr
100   format(a1,a)
      if ((chkch.ne.'*').and.(chkch.ne.'+')) then
         call WRTSTR(1,xpos,ypos,77,cl1)
         call WRTSTR(1,xpos,ypos,77,hlpstr)
      else
199      continue
         call WRTSTR(1,xpos,ypos,77,cl1)
         ypos = ypos + 14
         if (ypos.le.315) goto 199
      endif
      ypos = ypos + 14
      if ((chkch.ne.'*').and.(chkch.ne.'+')) goto 99
      backspace funit
300   continue
      return
      end

C ***************************************************************************
CN    MODULE NAME       : FINDPG
CA    FUNCTION          : To find a page of information in the help file.
CS    CALL SEQUENCE     : call findpg(sect,pg,funit)
CI    INPUT PARAMETERS :
C                  sect - (integer*2) The section number of the help to find.
C                    pg - (integer*2) The page number of the help to find.
C                 funit - (integer*2) The unit  number  for  the source of help
C                         information.
C
CO    OUTPUT PARAMETERS: File pointer at the start of the requested page.
CG    GLOBAL VARIABLES : None.
CM    MODULES CALLED   : None.
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS          : This routine scans  through  a  file  searching  for a
C                         particular section and page number.
C                         The scan starts at the current file  pointer  position
C                         and checks the first  control  record : If the section
C                         and page number is beyond that desired then a 'rewind'
C                         operation is performed  and  the  scan starts from the
C                         start of the help file.
C ***************************************************************************
      subroutine findpg(sect,pg,funit)
      integer*2 sect,pg,funit

      character*1 chkch
      integer*2 tsect,tpg,tmax,chkio
99    continue
      read(funit,'(a)',iostat=chkio,err=300) chkch
      if ((chkch.ne.'*').and.(chkch.ne.'+')) goto 99
      if (chkch.eq.'*') then
         backspace funit
         read(funit,102,iostat=chkio,err=300) chkch,tsect,tpg,tmax
102      format(a1,i4,i4,i4)
         if (tsect.gt.sect) then
            rewind funit
         elseif (tsect.eq.sect) then
            if (tpg.gt.pg) then
               rewind funit
            elseif (tpg.eq.pg) then
               backspace funit
            endif
         endif
      elseif (chkch.eq.'+') then
         rewind funit
      endif
98    continue
      read(funit,'(a)',iostat=chkio,err=300) chkch
      if ((chkch.ne.'*').and.(chkch.ne.'+')) goto 98
      backspace funit
      read(funit,102,iostat=chkio,err=300) chkch,tsect,tpg,tmax
      if ((tsect.ne.sect).or.(tpg.ne.pg)) goto 98
300   continue
      return
```

```
      end

C ***************************************************************************
CN    MODULE NAME       : DRVHLP
CA    FUNCTION          : To control the help screens within a help section.
CS    CALL SEQUENCE     : call drvhlp(sect,key,funit)
CI    INPUT PARAMETERS :
C                 sect - (integer*2) The section number of the help to find.
C                funit - (integer*2) The unit  number  for  the source of help
C                        information.
C
CO    OUTPUT PARAMETERS:
C                  key - (integer*2) The key that caused this routine to pass
C                        control back to the caller. (If  F1  key, the entire
C                        help facility closes down).
CG    GLOBAL VARIABLES : keys.inc
CM    MODULES CALLED   : findpg (for), INKEY2 (asm), ERTONE (asm).
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS          : This routine controls  the  display  of  information
C                         within a section of the help  file. Each section has
C                         a certain number of pages.  This  routine allows the
C                         user to page back and forth within the section.
C                         The ESC or F1 key causes the routine to pass control
C                         back to the calling routine.
C ***************************************************************************
      subroutine drvhlp(sect,key,funit)
      implicit integer*2 (I)
      integer*2 sect,key,funit

$include: 'keys.inc'
      integer*2 tsect,tpg,tmax,pos,chkio,oldpos
      character*1 chkch

      tpg = 1
      call findpg(sect,tpg,funit)
      backspace funit
      read(funit,102,iostat=chkio,err=300) chkch,tsect,tpg,tmax
102   format(a1,i4,i4,i4)
      pos = 1
      oldpos = 0
      backspace funit
99    continue
      if (oldpos.ne.pos) then
         call prpage(sect,pos,funit)
      endif
      oldpos = pos
      key = INKEY2()
      if (key.eq.pgdnk) then
         pos = pos + 1
         if (pos.gt.tmax) then
            pos = pos - 1
            call ERTONE()
         endif
      elseif (key.eq.pgupk) then
         pos = pos - 1
         if (pos.lt.1) then
            pos = pos + 1
            call ERTONE()
         endif
      endif
      if ((key.ne.esck).and.(key.ne.59)) goto 99
      if (key.eq.esck) key = 0
300   continue
      return
      end

C ***************************************************************************
CN    MODULE NAME       : BLKSEC
CA    FUNCTION          : To back highlight a menu header option.
CS    CALL SEQUENCE     : call blksec(sect)
CI    INPUT PARAMETERS :
C                 sect - (integer*2) The section number.
C
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES : None.
```

```
CM    MODULES CALLED   : LEVEL (asm), BLKFIL (asm).
CE    ERROR CONDITIONS : ?
C
CC    COMMENTS         : The routine fills a block around the option while
C                        the write attribute is set to XOR.
C ***************************************************************************
      subroutine blksec(sect)
      integer*2 sect

      call LEVEL(2)
      call BLKFIL(45,(17+3*14+sect*14+3),320,14)
      call LEVEL(1)
      return
      end

C ***************************************************************************
CH    REVISION HISTORY :
C     VERSION       BY              DATE        COMMENT
C     1.00          Ian Fisher      23/06/88    Creation.
C     FILEND           :
```

```
C ******************************************************************************
CN    MODULE NAME       : INMENU
CA    FUNCTION          : To initialise the menu screens and menus.
CS    CALL SEQUENCE     : call inmenu()
CI    INPUT PARAMETERS  : None.
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES  : None.
CM    MODULES CALLED    : BOX (asm), GPAGE (asm), INTSCR (asm), INDERR (asm).
CE    ERROR CONDITIONS  : ?
C
CC    COMMENTS          : Just clears all the pages, setups the frames around
C                         the screens and prints out the menu headers.
C
C ******************************************************************************

      subroutine inmenu()

      external BOX
      external GPAGE
      external INTSCR
      external INDERR

      call INTSCR()
      call INDERR()
      call BOX(1,271,717,270)
      call BOX(3,269,713,266)
      call GPAGE(1)
      call BOX(1,319,717,318)
      call BOX(3,317,713,314)
      call GPAGE(0)

      return
      end

C ******************************************************************************
CH    REVISION HISTORY :
C     VERSION       BY                DATE        COMMENT
C     0.00          Ian Fisher        23-11-87    Finally commented.
C     0.01          Ian Fisher        14-01-88    Add INT23H vector grab.
C     FILEND           :
C

C
C     FILE            : FDATA.FOR
C ******************************************************************************
      subroutine savef()
      implicit integer*2 (c,G,L,S)
$include: 'keys.inc'
$include: 'rigdat.inc'
      integer*2 key,ferr,derr,serr,len,funit
      character*1 yesno
      character*30 usenam
      usenam = '.dat                    '
      funit = 4
      call WIPSCR(0)
      call wrtitl(0,40,60,24,'Saving Rig Data to File.')
      call wrtitl(0,40,85,14,'Save to file :')
      call WRTSTR(0,40,110,30,'Hit RETURN to accept filename.')
      call WRTSTR(0,40,125,16,'Hit ESC to exit.')
      call dodir(0,180,5,'*.dat')
99    continue
      key = STRIN(0,175,85,30,usenam)
      if ((key.ne.retk).and.(key.ne.esck)) goto 99
      call WRTSTR(0,40,110,30,'                                    ')
      call WRTSTR(0,40,125,16,'                ')
      if (key.eq.retk) then
         call RSTERR()
         open(funit,FILE=usenam,STATUS='OLD',IOSTAT=ferr)
         if (ferr.gt.0) then
            derr = GETERR()
            if (derr.eq.-1) then
               close(funit)
               call WRTSTR(0,40,110,19,'Opening new file : ')
               call WRTSTR(0,215,110,25,usenam)
               open(funit,FILE=usenam,STATUS='NEW',IOSTAT=ferr)
               if (ferr.eq.0) then
```

```
                  serr = savdat(funit)
                  close(funit)
                  if (serr.eq.0) then
                     call WRTSTR(0,40,130,24,'Rig data save completed.')
                  else
                     call WRTSTR(0,40,130,40,
     +                    '*** Error : Rig data save not completed.')
                  endif
               else
                  call ERTONE()
                  call WRTSTR(0,40,130,33,
     +                 '*** Error : Cannot open new file.')
                  close(funit)
               endif
            else
               call prderr(0,40,130,derr)
            endif
         else
            len = LENSTR(30,usenam)
            call ERTONE()
            call WRTSTR(0,40,110,6,'File :')
            call LEVEL(2)
            call BLKFIL(115,113,(len*9),14)
            call WRTSTR(0,115,110,len,usenam)
            call LEVEL(1)
            call WRTSTR(0,(115+(len*9)),110,16,' already exists.')
            call WRTSTR(0,40,140,26,'Overwrite the file (y/n) ?')
            yesno = 'n'
98          continue
            key = STRIN(0,280,140,1,yesno)
            if (key.ne.retk) goto 98
            if ((yesno.eq.'y').or.(yesno.eq.'Y')) then
               call WRTSTR(0,40,170,20,'Saving rig data ....')
               close(funit)
               call RSTERR()
               open(funit,FILE=usenam,STATUS='NEW',IOSTAT=ferr)
               if (ferr.eq.0) then
                  serr = savdat(funit)
                  close(funit)
                  if (serr.eq.0) then
                     call WRTSTR(0,40,170,24,
     +                    'Rig data save completed.')
                  else
                     call WRTSTR(0,40,170,40,
     +                    '*** Error : Rig data save not completed.')
                  endif
               else
                  derr = GETERR()
                  if (derr.eq.-1) then
                     call ERTONE()
                     call WRTSTR(0,40,140,38,
     +                    '*** Error : Cannot overwrite old file.')
                     close(funit)
                  else
                     call prderr(0,40,170,derr)
                  endif
               endif
            else
               call WRTSTR(0,40,170,34,
     +                'Rig data save operation cancelled.')
            endif
         endif
      else
         call WRTSTR(0,40,170,34,
     +             'Rig data save operation cancelled.')
      endif
      close(funit)
      return
      end
C *****************************************************************************
      integer*2 function savdat(funit)
$include: 'rigdat.inc'
      integer*2 funit
      integer*2 chkio,i,j
      write(funit,600,iostat=chkio,err=699) samps,hertz
      write(funit,601,iostat=chkio,err=699)
```

```
     +    (invalv(i), i=1,4),tankno,step,stepno
600   format(bn,i4,f10.4)
601   format(bn,4f5.1,i1,f5.1,i4)
      do 650 i =1,samps
         write(funit,'(8i5)',iostat=chkio,err=699)
     +        (instru(i,j), j=1,8)
650   continue
      write(funit,602,iostat=chkio,err=699) commt
602   format(bn,a50)

      if (chkio.eq.0) goto 651
699   call ERTONE()
651   continue
      savdat = chkio
      return
      end

C ***************************************************************************
CN    MODULE NAME      : loadf
CM    MODULES CALLED   : BLKFIL  (asm) ,  ,  CMPSTR  (asm) ,
C                        ERTONE (asm), LEVEL (asm), lodmat (for), maknam (for),
C                        prderr (for), STRIN (asm), wrtitl (for), WRTSTR (asm).
CE    ERROR CONDITIONS : ?
C
C ***************************************************************************
      subroutine loadf()
      implicit integer*2 (c,G,l,S)
$include: 'keys.inc'
      integer*2 key,ferr,derr,lerr,funit
      character*30 usenam
      usenam = '.dat                          '
      funit = 4
      call WIPSCR(0)
      call wrtitl(0,40,60,27,'Loading Rig Data from File.')
      call wrtitl(0,40,85,16,'Enter filename :')
      call WRTSTR(0,40,110,30,'Hit RETURN to accept filename.')
      call WRTSTR(0,40,125,16,'Hit ESC to exit.')
      call dodir(0,180,5,'*.dat')
99    continue
      key = STRIN(0,190,85,30,usenam)
      if ((key.ne.retk).and.(key.ne.esck)) goto 99
      call WRTSTR(0,40,110,30,'                                   ')
      call WRTSTR(0,40,125,16,'                    ')
      if (key.eq.retk) then
         call RSTERR()
         open(funit,FILE=usenam,STATUS='OLD',IOSTAT=ferr)
         if (ferr.eq.0) then
            call WRTSTR(0,40,110,17,'Loading Data ....')
            lerr = lodmat(funit)
            close(funit)
            if (lerr.eq.0) then
               call WRTSTR(0,200,110,20,'Data load completed.')
            else
               call WRTSTR(0,40,130,36,
     +         '*** Error : Data load not completed.')
            endif
         else
            derr = GETERR()
            if (derr.eq.-1) then
               close(funit)
               call ERTONE()
               call WRTSTR(0,40,110,34,
     +              '*** Error : Cannot load from file.')
            else
               call prderr(0,40,110,derr)
            endif
         endif
      else
         call WRTSTR(0,40,140,30,'Data load operation cancelled.')
      endif
      return
      end

C ***************************************************************************
      integer*2 function lodmat(funit)
$include: 'rigdat.inc'
```

```
      integer*2 funit
      integer*2 chkio,i,j
      read(funit,500,iostat=chkio,err=599) samps,hertz
      read(funit,501,iostat=chkio,err=599)
     +    (invalv(i), i=1,4),tankno,step,stepno
500   format(bn,i4,f10.4)
501   format(bn,4f5.1,i1,f5.1,i4)
      do 550 i =1,samps
         read(funit,'(815)',iostat=chkio,err=599)
     +        (instru(i,j), j=1,8)
550   continue

      if (chkio.eq.0) goto 551
599   call ERTONE()
551   continue
      lodmat = chkio
      return
      end

C ***************************************************************************
CN    MODULE NAME       : prderr
CA    FUNCTION          : To print an appropriate disk error message.
CS    CALL SEQUENCE     : call prderr(page,x,y,derr)
CI    INPUT PARAMETERS  : page - (integer*2) The page to which the message is
C                                            to be written.
C                         x,y - (integer*2) The x,y co-ords of the message.
C                         derr - (integer*2) The disk error number.
CO    OUTPUT PARAMETERS: None.
CG    GLOBAL VARIABLES  : None.
CM    MODULES CALLED    : ERTONE (asm), WRTSTR (asm).
CE    ERROR CONDITIONS  : ?
C
CC    COMMENTS          : The routine interprets the disk drive error and then
C                         prints an  appropriate message at the user specified
C                         location.
C
C ***************************************************************************
      subroutine prderr(pg,x,y,derr)
      integer*2 pg,x,y,derr
      external WRTSTR
      external ERTONE
      call ERTONE()
      if (derr.eq.0) then
         call WRTSTR(pg,x,y,33,'*** Error : Disk write protected.')
      elseif (derr.eq.1) then
         call WRTSTR(pg,x,y,35,'*** Error : Unknown unit specified.')
      elseif (derr.eq.2) then
         call WRTSTR(pg,x,y,33,'*** Error : Disk drive not ready.')
      elseif (derr.eq.3) then
         call WRTSTR(pg,x,y,38,'*** Error : Unknown command for drive.')
      elseif (derr.eq.4) then
         call WRTSTR(pg,x,y,35,'*** Error : Data CRC error on disk.')
      elseif (derr.eq.5) then
         call WRTSTR(pg,x,y,45,
     +              '*** Error : Bad request for structure length.')
      elseif (derr.eq.6) then
         call WRTSTR(pg,x,y,29,'*** Error : Drive seek error.')
      elseif (derr.eq.7) then
         call WRTSTR(pg,x,y,31,'*** Error : Unknown media type.')
      elseif (derr.eq.8) then
         call WRTSTR(pg,x,y,29,'*** Error : Sector not found.')
      elseif (derr.eq.10) then
         call WRTSTR(pg,x,y,29,'*** Error : Disk write fault.')
      elseif (derr.eq.11) then
         call WRTSTR(pg,x,y,28,'*** Error : Disk read fault.')
      elseif (derr.eq.12) then
         call WRTSTR(pg,x,y,33,'*** Error : General disk failure.')
      endif
      return
      end

C ***************************************************************************
CH    REVISION HISTORY :
C     VERSION       BY                DATE        COMMENT
C     0.00          Ian Fisher        23-11-87    Finally commented.
```

```
C     FILEND          :
      subroutine mcont()
      implicit integer*2 (D)

      integer*2 opt3
      call WIPSCR(0)
99    continue
      call wrtitl(0,250,35,17,'Flow Rig Control.')
      opt3 = DOMENU(3)
      call WRTSTR(0,250,35,18,'                    ')
      if (opt3.eq.1) then
         call mloadf()
      elseif (opt3.eq.2) then
         call msave()
      elseif (opt3.eq.3) then
         call multi()
      elseif (opt3.eq.4) then
         call docont()
      elseif (opt3.eq.5) then
         call mldata()
      elseif (opt3.eq.6) then
         call mixdat()
      elseif (opt3.eq.7) then
         call pilod()
      endif
      if (opt3.ne.0) goto 99
      call WIPSCR(0)
      return
      end
C *************************************************************************
      subroutine multi()
      implicit integer*2 (I,O,R)
$include: 'keys.inc'
$include: 'rigdat.inc'
$include: 'multi.inc'

      integer*2 key,i,j,k,val,daerr,logfl,temp,secs,hunds,pos,swit
      integer*2 swval,swstep
      real*4 ctime,difft

      real*4 xprev(5),c(2,5)
      real*4 alpha(2,5),beta(2,5),gammap,gamman,uprev(2)
      real*4 Mat1(5,2),Mat2(5,5),GMat(5,2)
      real*4 sum(2),uinit(2),yinit(2)
      integer*2 sporig(2)

C***** Define Hyperplanes *****

      c(1,1) = 0.1
      c(1,2) = 1.0
      c(1,3) = 0.0
      c(1,4) = 0.1
      c(1,5) = 0.1
      c(2,1) = 0.1
      c(2,2) = 0.1
      c(2,3) = 0.1
      c(2,4) = 0.1
      c(2,5) = 1.0

C***** Define Input Gains *****
      alpha(1,1) =  0.35
      beta(1,1)  = -0.8
      alpha(1,2) =  1.6
      beta(1,2)  = -3.5
      alpha(1,3) =  0.3
      beta(1,3)  = -0.2
      alpha(1,4) =  0.6
      beta(1,4)  = -1.2
      alpha(1,5) =  0.6
      beta(1,5)  = -1.2
      alpha(2,1) =  0.1
      beta(2,1)  = -0.5
      alpha(2,2) =  0.8
      beta(2,2)  = -0.6
      alpha(2,3) =  0.9
      beta(2,3)  = -0.4
```

```
      alpha(2,4) =  0.8
      beta(2,4)  = -0.8
      alpha(2,5) =  0.6
      beta(2,5)  = -1.6
      gammap     =  4.0
      gamman     = -3.0


C***** Define Filter Matrices *****

      Do 1000 i = 1,5
         Do 1010 j = 1,5
            Mat2(i,j) = 0.0
1010     Continue
         Do 1020 j = 1,2
            GMat(i,j) = 0.0
            Mat1(i,j) = 0.0
1020     Continue
1000  Continue

      Mat1(2,1) =  0.03984
      Mat1(3,2) =  1.0
      Mat1(4,1) =  0.03542

      Mat2(1,1) =  0.9985
      Mat2(1,2) =  0.3623
      Mat2(1,3) = -0.0065
      Mat2(2,1) = -0.0015
      Mat2(2,2) = 0.3225
      Mat2(2,3) = -0.0065
      Mat2(3,3) = 0.9893
      Mat2(3,4) = 0.4571
      Mat2(3,5) = 0.0091
      Mat2(4,4) = 0.9646
      Mat2(5,3) = 0.0197
      Mat2(5,4) = 0.4571
      Mat2(5,5) = 0.9787
      GMat(1,1) = 1.0
      GMat(2,1) = 1.0
      GMat(3,2) = 1.0
      GMat(5,2) = 1.0

      sporig(1) = setp(4)
      sporig(2) = setp(3)

*******************************************************************************

      call WIPSCR(0)
      call WIPSCR(1)
      call wrtitl(0,200,30,30,'Multi-Variable Control of Rig.')
      call wrtitl(0,140,60,8,'Setpoint')
      call wrtitl(0,270,60,5,'Value')
      call wrtitl(0,360,60,5,'Error')
      call wrtitl(0,460,60,7,'U_value')
      call WRTSTR(0,30,80,12,'Recleaner : ')
      call WRTSTR(0,30,95,12,'Cleaner   : ')
      call WRTSTR(0,30,110,12,'Scavenger : ')
      call WRTSTR(0,30,125,12,'Rougher   : ')
      call inicon(setp,udly,uptr,ival,pi,u)
      call prtnum(0,140,80,1,4,'(14)',4,setp(1))
      call prtnum(0,140,95,1,4,'(14)',4,setp(2))
      call prtnum(0,140,110,1,4,'(14)',4,setp(3))
      call prtnum(0,140,125,1,4,'(14)',4,setp(4))

      call WRTSTR(0,30,150,6,'Step :')
      call prtnum(0,95,150,1,4,'(16)',6,steplv)
      call WRTSTR(0,300,150,6,'Tank :')
      call prtnum(0,365,150,1,4,'(i1)',1,stank)
      call WRTSTR(0,390,150,24,'(0-Recleaner  1-Cleaner)')
      call WRTSTR(0,390,165,24,'(2-Scavenger  3-Rougher)')
      call WRTSTR(0,30,190,5,'Log :')
      call WRTSTR(0,20,220,60,
     +'TAB to move between fields. ESC to quit (set valves to 85%).')
      call WRTSTR(0,20,235,58,
     +'F7/ALT-F7 to control LOG. ^N to step. ^V/ALT-V to disturb.')
      call WRTSTR(0,500,190,11,'DISTURB OFF')
```

```
      logfl = 0
      pos = 1
      sampl = 1
      swit = 0
      call LEVEL(2)
      call BLKFIL(95,152,54,13)
      call LEVEL(1)
      key = INKEY(2)
      daerr = RESTDA()
      if (daerr.eq.04) then
         call OUTP(548,11)
      endif
      uprev(2) = 4095 - 40.95*invalv(3)
      uprev(1) = 4095 - 40.95*invalv(4)
      uinit(1) = uprev(1)
      uinit(2) = uprev(2)
      call getlev(con)
      yinit(1) = setp(4)
c con(8)
      yinit(2) = setp(3)
c con(7)

C***** Calculate Initial States *****

      Do 1050 i = 1,5
         xprev(i) = 0.0
1050  Continue

*************************************************************

199   continue
      call GETIME(secs,hunds)
      ctime = real(secs) + real(hunds)/100.0
      call getlev(con)
      call docalc(con,setp,udly,uptr,uval,ival,pi,u,ul,uprev,
     +xprev,c,alpha,beta,gammap,gamman,Matl,Mat2,
     +GMat,sum,sporig,uinit,yinit)


      if (logfl.eq.1) then
         do 299 k = 1,8
            instru(sampl,k) = con(k)
299      continue
         do 297 k = 1,4
            instru(sampl,k) = ul(k)
297      continue
         instru(sampl,1) = int(sum(1))+2047
         instru(sampl,2) = int(sum(2))+2047
         do 298 k = 1,4
            spdata(sampl,k) = setp(k)
298      continue
         sampl = sampl + 1
         if (sampl.gt.4000) then
            logfl = 0
            sampl = 1
         endif
      endif
      do 498 j = 1,4
         daerr = OUTDA((j-1),ul(j))
498   continue
      if (swit.eq.1) then
         swval = swval + 1
         if (swval.eq.25) then
            swval = 1
            setp((stank+1)) = setp((stank+1)) + swstep
            if (setp(stank+1).gt.4095) setp(stank+1) = 4095
            if (setp(stank+1).lt.0) setp(stank+1) = 0
            call prtnum(0,140,80,1,4,'(i4)',4,setp(1))
            call prtnum(0,140,95,1,4,'(i4)',4,setp(2))
            call prtnum(0,140,110,1,4,'(i4)',4,setp(3))
            call prtnum(0,140,125,1,4,'(i4)',4,setp(4))
            swstep = -1*swstep
         endif
      endif
399   continue
      if (logfl.eq.0) then
```

```
         call WRTSTR(0,90,190,22,'OFF                    ')
      else
         call WRTSTR(0,90,190,14,'ON    Sample :')
         call prtnum(0,220,190,1,4,'(i4)',4,sampl)
      endif
      call GETIME(secs,hunds)
      difft = real(secs) + real(hunds)/100.0
      if (difft.lt.ctime) difft = difft + 60.0
      difft = abs(difft-ctime)
      if (difft.gt.(0.99)) goto 398
      key = INKEY(4)
      if (key.ne.0) then
         call edtrig(key,pos,logfl,sampl,swit,swval,swstep)
      endif
      if (key.eq.esck) goto 397
      goto 399
398   continue
      call prtnum(0,600,80,3,7,'(f10.2)',10,difft)
      goto 199

397   continue
      do 497 j = 1,4
         daerr = OUTDA((j-1),614)
497   continue
      logfl = 0
      if (sampl.gt.1) sampl = sampl - 1
      call WIPSCR(0)
      return
      end

C ***************************************************************************
      subroutine edtrig(key,pos,logfl,sam,swit,swval,swstep)
      integer*2 key,pos,logfl,sam,swit,swval,swstep
$include: 'keys.inc'
$include: 'multi.inc'
      integer*2 f7,altf7,ctrln
      f7 = 65
      altf7 = 110
      ctrln = 3584
      if ((key.eq.upk).or.(key.eq.downk)) then
         if (pos.eq.1) then
            if (key.eq.upk) then
               if (steplv.le.780) then
                  steplv = steplv + 41
               else
                  call ERTONE()
               endif
            elseif (key.eq.downk) then
               if (steplv.ge.(-780)) then
                  steplv = steplv - 41
               else
                  call ERTONE()
               endif
            endif
            call prtnum(0,95,150,1,4,'(i6)',6,steplv)
            call LEVEL(2)
            call BLKFIL(95,152,54,13)
            call LEVEL(1)
         elseif (pos.eq.2) then
            if (key.eq.upk) then
               if (stank.lt.3) then
                  stank = stank + 1
               else
                  call ERTONE()
               endif
            else
               if (stank.gt.0) then
                  stank = stank - 1
               else
                  call ERTONE()
               endif
            endif
            call prtnum(0,365,150,1,4,'(i1)',1,stank)
            call LEVEL(2)
            call BLKFIL(365,152,9,13)
            call LEVEL(1)
```

```
         endif
      elseif (key.eq.tabk) then
         pos = pos + 1
         if (pos.eq.3) pos = 1
         call LEVEL(2)
         call BLKFIL(95,152,54,13)
         call BLKFIL(365,152,9,13)
         call LEVEL(1)
      elseif (key.eq.f7) then
         if (logfl.eq.0) then
            logfl = 1
            sam = 1
         else
            call ERTONE()
         endif
      elseif (key.eq.altf7) then
         logfl = 0
      elseif (key.eq.ctrln) then
         setp((stank+1)) = setp((stank+1)) + steplv
         if (setp(stank+1).gt.4095) setp(stank+1) = 4095
         if (setp(stank+1).lt.0) setp(stank+1) = 0
         call prtnum(0,140,80,1,4,'(i4)',4,setp(1))
         call prtnum(0,140,95,1,4,'(i4)',4,setp(2))
         call prtnum(0,140,110,1,4,'(i4)',4,setp(3))
         call prtnum(0,140,125,1,4,'(i4)',4,setp(4))
      elseif (key.eq.5632) then
         if (swit.eq.0) then
            swit = 1
            swstep = steplv
            swval = 1
            call WRTSTR(0,500,190,11,'DISTURB ON ')
         endif
      elseif (key.eq.47) then
         swit = 0
         swstep = 0
         swval = 1
         call WRTSTR(0,500,190,11,'DISTURB OFF')
      endif

      return
      end
C *************************************************************************
      subroutine docalc(con,setp,udly,uptr,uval,ival,pi,u,ul,uprev,
     +xprev,c,alpha,beta,gammap,gamman,Mat1,Mat2,
     +GMat,sum,sporig,uinit,yinit)

      implicit integer*2 (I)
      integer*2 con(8),setp(4),udly(4,4,100)
      integer*2 uptr(4,4,2),uval(4,4)
      real*4 ival(4,4),pi(4,4,2),x(5),xprev(5),xcon(2),c(2,5)
      real*8 u(4),uc(2)
      integer*2 ul(4)
      real*4 alpha(2,5),beta(2,5),gammap,gamman,yprev,uprev(2),e(5)
      real*4 err(4),temp,errs(4),err0,err1,sum(2)
      real*4 Mat1(5,2),Mat2(5,5),GMat(5,2)
      real*4 uinit(2),yinit(2)
      integer*2 i,j,k,key
      integer*2 sporig(2)


C************************* Calculate The States ************************

      yinit(1) = setp(4)
      yinit(2) = setp(3)

      Do 2020 i = 1,5
         x(i) = 0
         Do 2030 j = 1,5
            x(i) = x(i) + Mat2(i,j)*xprev(j)
2030     Continue
         Do 2040 j = 1,2
            x(i) = x(i) + GMat(i,j)*(con(9-j)-yinit(j))
2040     Continue
         Do 2050 j = 1,2
            x(i) = x(i) + Mat1(i,j)*(uprev(j)-uinit(j))
2050     Continue
```

```
2020  Continue

C********************** Calculate The Inputs *************************

      Do 2060 i = 1,5
         e(i) = x(i) - xprev(i)
         xprev(i) = x(i)
2060  Continue
        call prtnum(0,270,80+3*15,1,4,'(i6)',6,con(8))
        call prtnum(0,340,80+3*15,3,7,'(f10.2)',10,e(1))
        call prtnum(0,270,80+2*15,1,4,'(i6)',6,con(7))
        call prtnum(0,340,80+2*15,3,7,'(f10.2)',10,e(3))
        call prtnum(0,270,80+1*15,1,4,'(i6)',6,e(4))
        call prtnum(0,340,80+1*15,3,7,'(f10.2)',10,e(5))
        call prtnum(0,270,80,1,4,'(i6)',6,e(2))
c        call prtnum(0,340,80,3,7,'(f10.2)',10,x(3))

        Do 2070 i = 1,2
           sum(i) = 0.0
2070    Continue
        Do 2080 i = 1,2
           Do 2090 j = 1,5
              sum(i) = sum(i) + c(i,j)*e(j)
2090       Continue
2080    Continue
        Do 2100 i = 1,2
           uc(i) = 0
           Do 2110 j = 1,5
              If(sum(i)*e(j) .gt. 0)uc(i) = uc(i) - alpha(i,j)*e(j)
              If(sum(i)*e(j) .lt. 0)uc(i) = uc(i) - beta(i,j)*e(j)
2110       Continue
           If(sum(1)*sum(2) .gt. 0)uc(2) = uc(2) - gammap*sum(1)
           If(sum(1)*sum(2) .lt. 0)uc(2) = uc(2) - gamman*sum(1)
           uc(i) = uc(i) + uprev(i)
           uprev(i) = uc(i)
           if (uc(i).gt.(4095.)) then
              uc(i) = 4095.0
              uprev(i)=uc(i)
           elseif (uc(i).lt.0.0) then
             uc(i) = 0.0
             uprev(i)=uc(i)
           endif
2100    Continue

        ul(4) = int(uc(1))
        ul(3) = int(uc(2))
c        uprev(1) = ul(4)
c        uprev(2) = ul(3)

        call prtnum(0,460,(80+(4-1)*15),1,4,'(i6)',6,ul(4))
        call prtnum(0,460,(80+(3-1)*15),1,4,'(i6)',6,ul(3))
      return
      end
C ***************************************************************************
      subroutine setdly(err)
      real*4 err(4)
$include: 'multi.inc'
      integer*2 i,j
      do 77 i = 1,4
         do 78 j = 1,4
            uptr(i,j,2) = uptr(i,j,2) + 1
            if (uptr(i,j,2).gt.100) then
               uptr(i,j,2) = 1
            endif
            uptr(i,j,1) = uptr(i,j,1) + 1
            if (uptr(i,j,1).gt.100) then
               uptr(i,j,1) = 1
            endif
78       continue
77    continue
      do 75 i = 1,4
         do 76 j = 1,4
            udly(i,j,uptr(i,j,2)) = int(err(j))
76       continue
75    continue
      return
```

```
      end
C ***************************************************************************
      subroutine gerr(row,col,udly,uptr,err0,err1)
      integer*2 row,col,udly(4,4,100),uptr(4,4,2)
      real*4 err0,err1

      err0 = real(udly(row,col,uptr(row,col,1)))
      if (uptr(row,col,1).eq.1) then
         err1 = real(udly(row,col,100))
      else
         err1 = real(udly(row,col,(uptr(row,col,1)-1)))
      endif
      return
      end
C ***************************************************************************
      subroutine inicon(setp,udly,uptr,ival,pi,u)
      integer*2 setp(4),udly(4,4,100)
      integer*2 uptr(4,4,2)
      real*4 ival(4,4),pi(4,4,2)
      real*8 u(4)

$include: 'pass.inc'
      integer*2 i,j,k

c     do 33 i = 1,4
c        do 34 j = 1,4
c           ival(i,j) = 0.0
c           do 35 k = 1,100
c              udly(i,j,k) = 0
c35          continue
c34       continue
c33    continue
c      do 36 i = 1,4
c         u(i) = 0.0
c        real(uuval(i))
c36    continue
      return
      end
C ***************************************************************************
      subroutine getlev(con)
      implicit integer*2 (I,O,R)
      integer*2 con(8)
$include: 'rigdat.inc'
      integer*4 temper,sams
      integer*2 i,j,err,temp,clklo,clkhi,samlo,samhi
      integer*2 basadr,comreg,stareg,datreg
      integer*2 waitcm,writwt,readwt,crdclr,crdclk,crdpar,crdrd,crdadi
      integer*2 crdstp,crderr

      basadr = 748
      comreg = basadr + 1
      stareg = basadr + 1
      datreg = basadr
      waitcm = 4
      writwt = 2
      readwt = 5
      crdclr = 1
      crdclk = 3
      crdpar = 13
      crdrd  = 14
      crdadi = 12
      crdstp = 15
      crderr = 2

      call OUTP(comreg,crdstp)
      temp = INP(datreg)
      call crdrdy()
      call OUTP(comreg,crdclr)
      call crdrdy()
      call OUTP(comreg,crdclk)
      call WAITDT(stareg,writwt,0)
      temper = (int (1.0/(1600.0*1.5*0.0000025))) + 1
      call CONVRT(clklo,clkhi,temper)
      call OUTP(datreg,clklo)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,clkhi)
```

```
      call crdrdy()
      call OUTP(comreg,crdpar)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,0)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,0)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,7)
      call WAITDT(stareg,writwt,0)
      sams = 8
      call CONVRT(samlo,samhi,sams)
      call OUTP(datreg,samlo)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,samhi)
      call crdrdy()
      call OUTP(comreg,crdrd)
      do 99 j = 1,8
         call WAITDT(stareg,readwt,1)
         con(j) =  (int (INP(datreg)))
         call WAITDT(stareg,readwt,1)
         con(j) =  (int(INP(datreg)*256)) + con(j)
99    continue
      return
      end
C ***************************************************************************
```

```
C
C     FILE            : FDATA.FOR
C ***************************************************************************
      subroutine msave()
      implicit integer*2 (c,G,L,S)
$include: 'keys.inc'
$include: 'multi.inc'
      integer*2 key,ferr,derr,serr,len,funit
      character*1 yesno
      character*30 usenam
      usenam = '.dat
      funit = 4
      call WIPSCR(0)
      call wrtitl(0,40,60,25,'Saving Multi Data to File.')
      call wrtitl(0,40,85,14,'Save to file :')
      call WRTSTR(0,40,110,30,'Hit RETURN to accept filename.')
      call WRTSTR(0,40,125,16,'Hit ESC to exit.')
      call dodir(0,180,5,'*.dat')
99    continue
      key = STRIN(0,175,85,30,usenam)
      if ((key.ne.retk).and.(key.ne.esck)) goto 99
      call WRTSTR(0,40,110,30,'                                    ')
      call WRTSTR(0,40,125,16,'                ')
      if (key.eq.retk) then
         call RSTERR()
         open(funit,FILE=usenam,STATUS='OLD',IOSTAT=ferr)
         if (ferr.gt.0) then
            derr = GETERR()
            if (derr.eq.-1) then
               close(funit)
               call WRTSTR(0,40,110,19,'Opening new file : ')
               call WRTSTR(0,215,110,25,usenam)
               open(funit,FILE=usenam,STATUS='NEW',IOSTAT=ferr)
               if (ferr.eq.0) then
                  serr = msav(funit)
                  close(funit)
                  if (serr.eq.0) then
                     call WRTSTR(0,40,130,24,'Rig data save completed.')
                  else
                     call WRTSTR(0,40,130,40,
     +                    '*** Error : Rig data save not completed.')
                  endif
               else
                  call ERTONE()
                  call WRTSTR(0,40,130,33,
     +                 '*** Error : Cannot open new file.')
                  close(funit)
               endif
            else
               call prderr(0,40,130,derr)
            endif
         else
            len = LENSTR(30,usenam)
            call ERTONE()
            call WRTSTR(0,40,110,6,'File :')
            call LEVEL(2)
            call BLKFIL(115,113,(len*9),14)
            call WRTSTR(0,115,110,len,usenam)
            call LEVEL(1)
            call WRTSTR(0,(115+(len*9)),110,16,' already exists.')
            call WRTSTR(0,40,140,26,'Overwrite the file (y/n) ?')
            yesno = 'n'
98          continue
            key = STRIN(0,280,140,1,yesno)
            if (key.ne.retk) goto 98
            if ((yesno.eq.'y').or.(yesno.eq.'Y')) then
               call WRTSTR(0,40,170,20,'Saving rig data ....')
               close(funit)
               call RSTERR()
               open(funit,FILE=usenam,STATUS='NEW',IOSTAT=ferr)
               if (ferr.eq.0) then
                  serr = msav(funit)
                  close(funit)
                  if (serr.eq.0) then
                     call WRTSTR(0,40,170,24,
     +                    'Rig data save completed.')
```

```
                else
                   call WRTSTR(0,40,170,40,
     +                  '*** Error : Rig data save not completed.')
                endif
             else
                derr = GETERR()
                if (derr.eq.-1) then
                   call ERTONE()
                   call WRTSTR(0,40,140,38,
     +                  '*** Error : Cannot overwrite old file.')
                   close(funit)
                else
                   call prderr(0,40,170,derr)
                endif
             endif
          else
             call WRTSTR(0,40,170,34,
     +               'Rig data save operation cancelled.')
          endif
       endif
      else
         call WRTSTR(0,40,170,34,
     +              'Rig data save operation cancelled.')
      endif
      close(funit)
      return
      end
C *************************************************************************
      integer*2 function msav(funit)
$include: 'multi.inc'
$include: 'rigdat.inc'
      integer*2 funit
      integer*2 chkio,i,j
      write(funit,600,iostat=chkio,err=699) sampl
600   format(bn,i4)
      do 650 i =1,sampl
         write(funit,'(12i5)',iostat=chkio,err=699)
     +        (instru(i,j), j=1,8),(spdata(i,j), j=1,4)
650   continue

      if (chkio.eq.0) goto 651
699   call ERTONE()
651   continue
      msav = chkio
      return
      end

C *************************************************************************
CN    MODULE NAME      : loadf
CM    MODULES CALLED   : BLKFIL  (asm) ,  ,  CMPSTR  (asm) ,
C                        ERTONE (asm), LEVEL (asm), mlod (for), maknam (for),
C                        prderr (for), STRIN (asm), wrtitl (for), WRTSTR (asm).
CE    ERROR CONDITIONS : ?
C
C *************************************************************************
      subroutine mloadf()
      implicit integer*2 (c,G,l,S)
$include: 'keys.inc'
      integer*2 key,ferr,derr,lerr,funit
      character*30 usenam
      usenam = '.dat                          '
      funit = 4
      call WIPSCR(0)
      call wrtitl(0,40,60,28,'Loading Multi Data from File.')
      call wrtitl(0,40,85,16,'Enter filename :')
      call WRTSTR(0,40,110,30,'Hit RETURN to accept filename.')
      call WRTSTR(0,40,125,16,'Hit ESC to exit.')
      call dodir(0,180,5,'*.dat')
99    continue
      key = STRIN(0,190,85,30,usenam)
      if ((key.ne.retk).and.(key.ne.esck)) goto 99
      call WRTSTR(0,40,110,30,'                                  ')
      call WRTSTR(0,40,125,16,'                ')
      if (key.eq.retk) then
         call RSTERR()
         open(funit,FILE=usenam,STATUS='OLD',IOSTAT=ferr)
```

```
         if (ferr.eq.0) then
            call WRTSTR(0,40,110,17,'Loading Data ....')
            lerr = mlod(funit)
            close(funit)
            if (lerr.eq.0) then
               call WRTSTR(0,200,110,20,'Data load completed.')
            else
               call WRTSTR(0,40,130,36,
     +         '*** Error : Data load not completed.')
            endif
         else
            derr = GETERR()
            if (derr.eq.-1) then
               close(funit)
               call ERTONE()
               call WRTSTR(0,40,110,34,
     +              '*** Error : Cannot load from file.')
            else
               call prderr(0,40,110,derr)
            endif
         endif
      else
         call WRTSTR(0,40,140,30,'Data load operation cancelled.')
      endif
      return
      end

C ***************************************************************************
      integer*2 function mlod(funit)
$include: 'rigdat.inc'
$include: 'multi.inc'

      integer*2 funit
      integer*2 chkio,i,j
      read(funit,500,iostat=chkio,err=599) sampl
500   format(bn,i4)
      do 550 i =1,sampl
         read(funit,'(12i5)',iostat=chkio,err=599)
     +        (instru(i,j), j=1,8),(spdata(i,j), j=1,4)
550   continue

      if (chkio.eq.0) goto 551
599   call ERTONE()
551   continue
      mlod = chkio
      return
      end

C ***************************************************************************
      subroutine mldata()
      implicit integer*2 (e,I,g)
$include: 'rigdat.inc'
$include: 'multi.inc'
$include: 'keys.inc'
      integer*2 i,j,key,pos,flflg,levflg,valflg
      real*4 totime,pmax,pmin
c      real*4 ssp,ssporig,alpha,beta,Again,torl,tor2,uprev,uuc,sum,
c     + Ro,k1,k2,kappa,cc(2),xx(2),xcon(2)

      call WIPSCR(0)
      call wrtitl(0,50,20,18,'Plotting Rig Data.')
      call wrtitl(0,20,40,30,'Current rig data parameters :-')
      call WRTSTR(0,20,55,22,'Data from rig sampled ')
      pos = 20 + 22*9
      call prtnum(0,pos,55,1,4,'(i4)',4,sampl)
      call WRTSTR(0,(pos+5*9),55,25,' times at a frequency of ')
      call prtnum(0,(pos+30*9),55,3,7,'(f7.2)',7,(1.0))
      call WRTSTR(0,(pos+38*9),55,6,'Hertz.')

      call wrtitl(0,20,90,19,'Axes Information :-')
      call WRTSTR(0,20,105,30,'Horizontal Axes : Time [secs] :')
      totime = real(sampl)/(1.0)
      call prtnum(0,300,105,3,7,'(f10.4)',10,totime)
      call WRTSTR(0,20,120,43,
     +'Vertical Axes   : Percentage of Full Scale.')
      pmax = 100.0
```

```
      pmin = 0.0
      call WRTSTR(0,210,135,24,'Max :             Min : ')
      call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)
      flflg  = 1
      levflg = 1
      valflg = 1
      call wrtitl(0,20,150,18,'Plot Parameters :-')
      call WRTSTR(0,20,170,8,'Flows  :')
      call prsel(flflg,170,1)
      call WRTSTR(0,20,185,8,'Levels :')
      call prsel(levflg,185,1)
      call WRTSTR(0,20,200,8,'Setpnt :')
      call prsel(valflg,200,1)
      call WRTSTR(0,20,220,31,'Space bar to flip.   F5 to plot.')
      call WRTSTR(0,20,235,50,
     +'RETURN, TAB and cursor keys to move.  ESC to exit.')
      pos = 1
299   continue
      if (pos.eq.1) then
         key = edtflg(flflg,170)
      elseif (pos.eq.2) then
         key = edtflg(levflg,185)
      elseif (pos.eq.3) then
         key = edtflg(valflg,200)
      elseif (pos.eq.4) then
         key = getnum(0,260,135,3,7,'(g10.4)',10,pmax)
         if (pmax.gt.(100.0)) then
            call ERTONE()
            pmax = 100.0
         elseif (pmax.lt.(0.0)) then
            call ERTONE()
            pmax = 0.0
         endif
         call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      elseif (pos.eq.5) then
         key = getnum(0,430,135,3,7,'(g10.4)',10,pmin)
         if (pmin.gt.pmax) then
            call ERTONE()
            pmin = pmax
         elseif (pmin.lt.(0.0)) then
            call ERTONE()
            pmin = 0.0
         endif
         call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)
      endif
      if ((key.eq.tabk).or.(key.eq.downk).or.(key.eq.retk)) then
         pos = pos + 1
         if (pos.gt.5) pos = 1
      elseif ((key.eq.rtabk).or.(key.eq.upk)) then
         pos = pos - 1
         if (pos.lt.1) pos = 5
      elseif (key.eq.63) then
         call mltrig(flflg,levflg,valflg,pmax,pmin)
      endif
      if (key.ne.esck) goto 299
      call WIPSCR(0)
      return
      end
C *************************************************************************
      subroutine mltrig(flflg,levflg,valflg,pmax,pmin)
      integer*2 flflg,levflg,valflg
      real*4 pmax,pmin
$include: 'rigdat.inc'
$include: 'rigplt.inc'
$include: 'multi.inc'
      integer*2 i,j,npts

      call WIPSCR(1)
      call DISP(1)
      xp(1) = real(sampl)/(1.0)
      xp(2) = 0.0
      xp(3) = 0.0
      yp(1) = pmax
      yp(2) = pmin
      yp(3) = pmin
```

```
      do 199 i = 0,(sampl-1)
         time(i+1) = real(i)/(1.0)
199   continue
      call WRTSTR(1,145,158,7,'Rougher')
      call WRTSTR(1,145,297,9,'Scavenger')
      call WRTSTR(1,500,297,7,'Cleaner')
      call WRTSTR(1,500,158,9,'Recleaner')
      call WRTSTR(1,250,313,27,
     +'F - Flow          L - Level')

      call MOVE(4,301)
      call DLINE(715,301)
      call MOVE(4,162)
      call DLINE(715,162)
      call MOVE(365,301)
      call DLINE(365,22)
      call WRTSTR(1,170,18,42,
     +          'Flotation Simulator : Closed Loop Control.')
      call MOVE(4,22)
      call DLINE(715,22)

      grfdim(1) = 10
      grfdim(2) = 150
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,roscal,rocent)
      call WRTSTR(1,35,38,3,'[%]')
      call WRTSTR(1,310,158,3,'[s]')

      grfdim(1) = 10
      grfdim(2) = 290
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,scscal,sccent)
      call WRTSTR(1,35,178,3,'[%]')
      call WRTSTR(1,310,297,3,'[s]')

      grfdim(1) = 375
      grfdim(2) = 290
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,clscal,clcent)
      call WRTSTR(1,400,178,3,'[%]')
      call WRTSTR(1,675,297,3,'[s]')

      grfdim(1) = 375
      grfdim(2) = 150
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,rcscal,rccent)
      call WRTSTR(1,400,38,3,'[%]')
      call WRTSTR(1,675,158,3,'[s]')

      if (flflg.eq.1) then
         call fixdat(percnt,instru,sampl,4)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'f')
         call fixdat(percnt,instru,sampl,3)
         call drawpt(1,1,time,percnt,sampl,sccent,scscal,2)
         call drwlet(1,time(sampl),percnt(sampl),sccent,scscal,'f')
         call fixdat(percnt,instru,sampl,2)
         call drawpt(1,1,time,percnt,sampl,clcent,clscal,2)
         call drwlet(1,time(sampl),percnt(sampl),clcent,clscal,'f')
         call fixdat(percnt,instru,sampl,1)
         call drawpt(1,1,time,percnt,sampl,rccent,rcscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rccent,rcscal,'f')
      endif
      if (levflg.eq.1) then
         call fixdat(percnt,instru,sampl,8)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'l')
         call fixdat(percnt,instru,sampl,7)
         call drawpt(1,1,time,percnt,sampl,sccent,scscal,2)
         call drwlet(1,time(sampl),percnt(sampl),sccent,scscal,'l')
         call fixdat(percnt,instru,sampl,6)
```

```
         call drawpt(1,1,time,percnt,sampl,clcent,clscal,2)
         call drwlet(1,time(sampl),percnt(sampl),clcent,clscal,'l')
         call fixdat(percnt,instru,sampl,5)
         call drawpt(1,1,time,percnt,sampl,rccent,rcscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rccent,rcscal,'l')
      endif
      if (valflg.eq.1) then
         call valfix(percnt,spdata,sampl,4)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call valfix(percnt,spdata,sampl,3)
         call drawpt(1,1,time,percnt,sampl,sccent,scscal,2)
         call valfix(percnt,spdata,sampl,2)
         call drawpt(1,1,time,percnt,sampl,clcent,clscal,2)
         call valfix(percnt,spdata,sampl,1)
         call drawpt(1,1,time,percnt,sampl,rccent,rcscal,2)
      endif
      if (flflg.eq.1) then
         do 249 i = 1,50
            time(i) = int(i*(real(sampl)/1.0)/50.0)
            percnt(i) = real(760)/40.95
249      continue
         call drawpt(1,1,time,percnt,50,rocent,roscal,1)
         call drawpt(1,1,time,percnt,50,sccent,scscal,1)
         call drawpt(1,1,time,percnt,50,clcent,clscal,1)
         call drawpt(1,1,time,percnt,50,rccent,rcscal,1)
      endif
      return
      end
C *************************************************************************
      subroutine valfix(percnt,data,samps,instr)
      real*4 percnt(4000)
      integer*2 data(4000,4)
      integer*2 samps,instr
      integer*2 i
      do 299 i =1,samps
         percnt(i) = real(data(i,instr))/40.95
299   continue
      return
      end

C *************************************************************************
C *************************************************************************
      subroutine mixdat()
      implicit integer*2 (e,I,g)
$include: 'rigdat.inc'
$include: 'multi.inc'
$include: 'keys.inc'
      integer*2 i,j,key,pos,flflg,levflg,valflg
      integer*2 tankpl(4,3)
      real*4 totime,pmax,pmin
      call WIPSCR(0)
      call wrtitl(0,50,20,18,'Plotting Rig Data.')
      call wrtitl(0,20,40,30,'Current rig data parameters :-')
      call WRTSTR(0,20,55,22,'Data from rig sampled ')
      pos = 20 + 22*9
      call prtnum(0,pos,55,1,4,'(i4)',4,sampl)
      call WRTSTR(0,(pos+5*9),55,25,' times at a frequency of ')
      call prtnum(0,(pos+30*9),55,3,7,'(f7.2)',7,(1.0))
      call WRTSTR(0,(pos+38*9),55,6,'Hertz.')

      call wrtitl(0,20,90,19,'Axes Information :-')
      call WRTSTR(0,20,105,30,'Horizontal Axes : Time [secs] :')
      totime = real(sampl)/(1.0)
      call prtnum(0,300,105,3,7,'(f10.4)',10,totime)
      call WRTSTR(0,20,120,43,
     +'Vertical Axes   : Percentage of Full Scale.')
      pmax = 100.0
      pmin = 0.0
      call WRTSTR(0,210,135,24,'Max :              Min : ')
      call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)


      tankpl(1,1) = 1
      tankpl(1,2) = 1
      tankpl(1,3) = 0
```

```
      tankpl(2,1) = 1
      tankpl(2,2) = 1
      tankpl(2,3) = 0

      tankpl(3,1) = 1
      tankpl(3,2) = 1
      tankpl(3,3) = 0

      tankpl(4,1) = 1
      tankpl(4,2) = 1
      tankpl(4,3) = 0

      call wrtitl(0,95,150,7,'Rougher')
      call wrtitl(0,195,150,9,'Scavenger')
      call wrtitl(0,295,150,7,'Cleaner')
      call wrtitl(0,395,150,9,'Recleaner')
      call WRTSTR(0,20,170,7,'Level :')
      call WRTSTR(0,20,190,7,'Setpt :')
      call WRTSTR(0,20,210,7,'Flow  :')
      do 888 i = 1,4
         do 887 j = 1,3
            call prsel2(tankpl(i,j),i,j,1)
887      continue
888   continue
      call WRTSTR(0,20,245,31,'Space bar to flip.   F5 to plot.')
      call WRTSTR(0,20,260,50,
     +'RETURN, TAB and cursor keys to move.  ESC to exit.')

      pos = 1
      i = 1
      j = 1
299   continue
      if (pos.eq.1) then
         key = edtf12(tankpl(i,j),i,j)
         call prsel2(tankpl(i,j),i,j,1)
      elseif (pos.eq.2) then
         key = getnum(0,260,135,3,7,'(g10.4)',10,pmax)
         if (pmax.gt.(100.0)) then
            call ERTONE()
            pmax = 100.0
         elseif (pmax.lt.(0.0)) then
            call ERTONE()
            pmax = 0.0
         endif
         call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      elseif (pos.eq.3) then
         key = getnum(0,430,135,3,7,'(g10.4)',10,pmin)
         if (pmin.gt.pmax) then
            call ERTONE()
            pmin = pmax
         elseif (pmin.lt.(0.0)) then
            call ERTONE()
            pmin = 0.0
         endif
         call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)
      endif
      if ((key.eq.tabk).or.(key.eq.retk)) then
         pos = pos + 1
         if (pos.gt.3) pos = 1
      elseif (key.eq.downk) then
         j = j + 1
         if (j.gt.3) j = 1
      elseif (key.eq.leftk) then
         i = i - 1
         if (i.lt.1) i = 4
      elseif (key.eq.rightk) then
         i = i + 1
         if (i.gt.4) i = 1
      elseif (key.eq.upk) then
         j = j - 1
         if (j.lt.1) j = 3
      elseif (key.eq.63) then
         call mixrig(tankpl,pmax,pmin)
      endif
      if (key.ne.esck) goto 299
```

```
      call WIPSCR(0)
      return
      end
C ***************************************************************************
      subroutine mixrig(tankpl,pmax,pmin)
      integer*2 tankpl(4,3)
      real*4 pmax,pmin
$include: 'rigdat.inc'
$include: 'multi.inc'
$include: 'rigplt.inc'
      integer*2 i,j,npts

      call WIPSCR(1)
      call DISP(1)
      xp(1) = real(sampl)/(1.0)
      xp(2) = 0.0
      xp(3) = 0.0
      yp(1) = pmax
      yp(2) = pmin
      yp(3) = pmin

      do 199 i = 0,(sampl-1)
         time(i+1) = real(i)/(1.0)
199   continue
      call WRTSTR(1,30,313,11,'Rougher - 1')
      call WRTSTR(1,190,313,13,'Scavenger - 2')
      call WRTSTR(1,350,313,11,'VSS Control')
      call WRTSTR(1,510,313,13,'D. Ginsberg')
      call WRTSTR(1,30,297,27,
     +'  I -I/P  S -S.F. L -Level ')
      call MOVE(4,282)
      call DLINE(715,282)
      call WRTSTR(1,170,18,42,
     +            'Flotation Simulator : Closed Loop Control.')
      call MOVE(4,22)
      call DLINE(715,22)
      call WRTSTR(1,47,38,3,'[%]')
      call WRTSTR(1,630,275,8,'Time [s]')
      grfdim(1) = 20
      grfdim(2) = 265
      grfdim(3) = 660
      grfdim(4) = 225
      call dwaxes(1,1,1,xp,yp,grfdim,roscal,rocent)

      if (tankpl(1,1).eq.1) then
         call fixdat(percnt,instru,sampl,8)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
      endif
      if (tankpl(2;1).eq.1) then
         call fixdat(percnt,instru,sampl,7)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'2')
      endif
      if (tankpl(3,1).eq.1) then
         call fixdat(percnt,instru,sampl,6)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'3')
      endif
      if (tankpl(4,1).eq.1) then
         call fixdat(percnt,instru,sampl,5)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'4')
      endif

      if (tankpl(1,2).eq.1) then
         call valfix(percnt,spdata,sampl,4)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
      endif
      if (tankpl(2,2).eq.1) then
         call valfix(percnt,spdata,sampl,3)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
```

```
      endif
      if (tankpl(3,2).eq.1) then
         call valfix(percnt,spdata,sampl,2)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
      endif
      if (tankpl(4,2).eq.1) then
         call valfix(percnt,spdata,sampl,1)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
      endif

      if (tankpl(1,3).eq.1) then
         call fixdat(percnt,instru,sampl,4)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
      endif
      if (tankpl(2,3).eq.1) then
         call fixdat(percnt,instru,sampl,3)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'2')
      endif
      if (tankpl(3,3).eq.1) then
         call fixdat(percnt,instru,sampl,2)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'2')
      endif
      if (tankpl(4,3).eq.1) then
         call fixdat(percnt,instru,sampl,1)
         call drawpt(1,1,time,percnt,sampl,rocent,roscal,2)
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'')
         call drwlet(1,time(sampl),percnt(sampl),rocent,roscal,'1')
      endif

      return
      end
C *************************************************************************
      integer*2 function edtfl2(flg,col,row)
      implicit integer*2 (I)
      integer*2 flg,col,row
      integer*2 key
399   continue
      call prsel2(flg,col,row,0)
      key = INKEY(1)
      if (key.eq.8192) then
         if (flg.eq.1) then
            flg = 0
         else
            flg = 1
         endif
      endif
      if (key.eq.8192) goto 399
      call prsel(flg,col,row,1)
      edtfl2 = key
      return
      end
C *************************************************************************
      subroutine prsel2(flg,col,row,lev)
      integer*2 flg,col,row,lev
      call LEVEL(lev)
      if (flg.eq.0) then
         call WRTSTR(0,(125+(col-1)*100),(170+(row-1)*20),3,'OFF')
      else
         call WRTSTR(0,(125+(col-1)*100),(170+(row-1)*20),3,' ON')
      endif
      call LEVEL(1)
      return
      end
C *************************************************************************
C *************************************************************************
      subroutine mix2()
      implicit integer*2 (e,I,g)
$include: 'rigdat.inc'
$include: 'multi.inc'
$include: 'keys.inc'
```

```
      integer*2 i,j,key,pos,flflg,levflg,valflg
      integer*2 tankpl(4,3)
      real*4 totime,pmax,pmin
      call WIPSCR(0)
      call wrtitl(0,50,20,18,'Plotting Rig Data.')
      call wrtitl(0,20,40,30,'Current rig data parameters :-')
      call WRTSTR(0,20,55,22,'Data from rig sampled ')
      pos = 20 + 22*9
      call prtnum(0,pos,55,1,4,'(i4)',4,sampl)
      call WRTSTR(0,(pos+5*9),55,25,' times at a frequency of ')
      call prtnum(0,(pos+30*9),55,3,7,'(f7.2)',7,(1.0))
      call WRTSTR(0,(pos+38*9),55,6,'Hertz.')

      call wrtitl(0,20,90,19,'Axes Information :-')
      call WRTSTR(0,20,105,30,'Horizontal Axes : Time [secs] :')
      totime = real(sampl)/(1.0)
      call prtnum(0,300,105,3,7,'(f10.4)',10,totime)
      call WRTSTR(0,20,120,43,
     +'Vertical Axes   : Percentage of Full Scale.')
      pmax = 100.0
      pmin = 0.0
      call WRTSTR(0,210,135,24,'Max :              Min : ')
      call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)


      tankpl(1,1) = 1
      tankpl(1,2) = 1
      tankpl(1,3) = 0

      tankpl(2,1) = 1
      tankpl(2,2) = 1
      tankpl(2,3) = 0

      tankpl(3,1) = 1
      tankpl(3,2) = 1
      tankpl(3,3) = 0

      tankpl(4,1) = 1
      tankpl(4,2) = 1
      tankpl(4,3) = 0

      call wrtitl(0,95,150,7,'Rougher')
      call wrtitl(0,195,150,9,'Scavenger')
      call wrtitl(0,295,150,7,'Cleaner')
      call wrtitl(0,395,150,9,'Recleaner')
      call WRTSTR(0,20,170,7,'Level :')
      call WRTSTR(0,20,190,7,'Setpt :')
      call WRTSTR(0,20,210,7,'Flow  :')
      do 888 i = 1,4
         do 887 j = 1,3
            call prsel2(tankpl(i,j),i,j,1)
887      continue
888   continue
      call WRTSTR(0,20,245,31,'Space bar to flip.   F5 to plot.')
      call WRTSTR(0,20,260,50,
     +'RETURN, TAB and cursor keys to move.  ESC to exit.')

      pos = 1
      i = 1
      j = 1
299   continue
      if (pos.eq.1) then
         key = edtf12(tankpl(i,j),i,j)
         call prsel2(tankpl(i,j),i,j,1)
      elseif (pos.eq.2) then
         key = getnum(0,260,135,3,7,'(g10.4)',10,pmax)
         if (pmax.gt.(100.0)) then
            call ERTONE()
            pmax = 100.0
         elseif (pmax.lt.(0.0)) then
            call ERTONE()
            pmax = 0.0
         endif
         call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      elseif (pos.eq.3) then
```

```
         key = getnum(0,430,135,3,7,'(g10.4)',10,pmin)
         if (pmin.gt.pmax) then
            call ERTONE()
            pmin = pmax
         elseif (pmin.lt.(0.0)) then
            call ERTONE()
            pmin = 0.0
         endif
         call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)
      endif
      if ((key.eq.tabk).or.(key.eq.retk)) then
         pos = pos + 1
         if (pos.gt.3) pos = 1
      elseif (key.eq.downk) then
         j = j + 1
         if (j.gt.3) j = 1
      elseif (key.eq.leftk) then
         i = i - 1
         if (i.lt.1) i = 4
      elseif (key.eq.rightk) then
         i = i + 1
         if (i.gt.4) i = 1
      elseif (key.eq.upk) then
         j = j - 1
         if (j.lt.1) j = 3
      elseif (key.eq.63) then
         call mixr2(tankpl,pmax,pmin)
      endif
      if (key.ne.esck) goto 299
      call WIPSCR(0)
      return
      end
C *************************************************************************
      subroutine mixr2(tankpl,pmax,pmin)
      integer*2 tankpl(4,3)
      real*4 pmax,pmin
$include: 'rigdat.inc'
$include: 'multi.inc'
$include: 'rigplt.inc'
      integer*2 i,j,npts

      call WIPSCR(1)
      call DISP(1)
      xp(1) = real(samps)/(hertz)
      xp(2) = 0.0
      xp(3) = 0.0
      yp(1) = pmax
      yp(2) = pmin
      yp(3) = pmin

      do 199 i = 0,(samps-1)
         time(i+1) = real(i)/hertz
199   continue
      call WRTSTR(1,30,313,11,'Rougher - 1')
      call WRTSTR(1,190,313,13,'Scavenger - 2')
      call WRTSTR(1,350,313,11,'Cleaner - 3')
      call WRTSTR(1,510,313,13,'Recleaner - 4')
      call WRTSTR(1,30,297,27,
     +'F - Flow          L - Level')
      call MOVE(4,282)
      call DLINE(715,282)
      call WRTSTR(1,170,18,37,
     +            'Flotation Simulator : Open Loop Test.')
      call MOVE(4,22)
      call DLINE(715,22)
      call WRTSTR(1,47,38,3,'[%]')
      call WRTSTR(1,630,275,8,'Time [s]')
      grfdim(1) = 20
      grfdim(2) = 265
      grfdim(3) = 660
      grfdim(4) = 225
      call dwaxes(1,1,1,xp,yp,grfdim,roscal,rocent)

      if (tankpl(1,1).eq.1) then
         call fixdat(percnt,instru,samps,8)
         call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
```

```
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'1')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'1')
   endif
   if (tankpl(2,1).eq.1) then
      call fixdat(percnt,instru,samps,7)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'1')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'2')
   endif
   if (tankpl(3,1).eq.1) then
      call fixdat(percnt,instru,samps,6)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'1')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'3')
   endif
   if (tankpl(4,1).eq.1) then
      call fixdat(percnt,instru,samps,5)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'1')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'4')
   endif

   if (tankpl(1,2).eq.1) then
      call fixval(percnt,time,npts,3)
      call drawpt(1,1,time,percnt,npts,rocent,roscal,2)
      call drwlet(1,time(npts),percnt(npts),rocent,roscal,'v')
   endif
   if (tankpl(2,2).eq.1) then
      call fixval(percnt,time,npts,2)
      call drawpt(1,1,time,percnt,npts,rocent,roscal,2)
      call drwlet(1,time(npts),percnt(npts),rocent,roscal,'v')
   endif
   if (tankpl(3,2).eq.1) then
      call fixval(percnt,time,npts,1)
      call drawpt(1,1,time,percnt,npts,rocent,roscal,2)
      call drwlet(1,time(npts),percnt(npts),rocent,roscal,'v')
   endif
   if (tankpl(4,2).eq.1) then
      call fixval(percnt,time,npts,0)
      call drawpt(1,1,time,percnt,npts,rocent,roscal,2)
      call drwlet(1,time(npts),percnt(npts),rocent,roscal,'v')
   endif
   do 250 i = 1,50
      time(i) = real(stepno)/hertz
      percnt(i) = real(i-1)*((pmax-pmin)/100.0)*2.0 + 1.0 + pmin
250 continue
   call drawpt(1,1,time,percnt,50,rocent,roscal,1)


   if (tankpl(1,3).eq.1) then
      call fixdat(percnt,instru,samps,4)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'f')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'1')
   endif
   if (tankpl(2,3).eq.1) then
      call fixdat(percnt,instru,samps,3)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'f')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'2')
   endif
   if (tankpl(3,3).eq.1) then
      call fixdat(percnt,instru,samps,2)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'f')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'3')
   endif
   if (tankpl(4,3).eq.1) then
      call fixdat(percnt,instru,samps,1)
      call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'f')
      call drwlet(1,time(samps),percnt(samps),rocent,roscal,'4')
   endif

   return
   end
```

```
            subroutine nodly()
      implicit integer*2 (D)
      integer*2 opt12

      call WIPSCR(0)
      call wrtitl(0,200,40,21,'Modifying Data Files.')
99    continue
      opt12 = DOMENU(12)
      if (opt12.eq.1) then
         call modi1()
      elseif (opt12.eq.2) then
         call modi2()
      endif
      if (opt12.ne.0) goto 99
      call WIPSCR(0)
      return
      end
C     ********************************************************************************
      subroutine modi1()
      implicit integer*2 (I,g,S)
$include: 'keys.inc'
$include: 'rigdat.inc'
      integer*2 key,pos,ctrlt
      character*40 ronam,scnam,clnam,rcnam

      ctrlt = 5120
      ronam = '.lro'
      scnam = '.lsc'
      clnam = '.lcl'
      rcnam = '.lrc'
      call WIPSCR(0)
      call wrtitl(0,50,20,20,'Modify Data Files 1.')
      call WRTSTR(0,20,50,12,'Recleaner : ')
      call WRTSTR(0,20,65,12,'Cleaner   : ')
      call WRTSTR(0,20,80,12,'Scavenger : ')
      call WRTSTR(0,20,95,12,'Rougher   : ')
      call WRTSTR(0,20,130,22,'Hit CTRL-T to process.')
      call WRTSTR(0,130,65,40,clnam)
      call WRTSTR(0,130,80,40,scnam)
      call WRTSTR(0,130,95,40,ronam)
      pos = 1
199   continue
      if (pos.eq.1) then
         key = STRIN(0,130,50,40,rcnam)
      elseif (pos.eq.2) then
         key = STRIN(0,130,65,40,clnam)
      elseif (pos.eq.3) then
         key = STRIN(0,130,80,40,scnam)
      elseif (pos.eq.4) then
         key = STRIN(0,130,95,40,ronam)
      endif
      if ((key.eq.retk).or.(key.eq.tabk).or.(key.eq.downk)) then
         pos = pos + 1
         if (pos.gt.4) pos = 1
      elseif ((key.eq.rtabk).or.(key.eq.upk)) then
         pos = pos - 1
         if (pos.lt.1) pos = 4
      elseif (key.eq.ctrlt) then
         call domod1(rcnam,5)
         call WRTSTR(0,500,50,4,'Done')
         call domod1(clnam,6)
         call WRTSTR(0,500,65,4,'Done')
         call domod1(scnam,7)
         call WRTSTR(0,500,80,4,'Done')
         call domod1(ronam,8)
         call WRTSTR(0,500,95,4,'Done')
         key=esck
      endif
      if (key.ne.esck) goto 199
      call WIPSCR(0)
      return
      end
C ********************************************************************************
      subroutine modi2()
      implicit integer*2 (I,g,S)
$include: 'keys.inc'
```

```
$include: 'rigdat.inc'
      integer*2 key,pos,ctrlt
      character*40 ronam,scnam,clnam,rcnam

      ctrlt = 5120
      scnam = '.lsc'
      call WIPSCR(0)
      call wrtitl(0,50,20,20,'Modify Data Files 2.')
      call WRTSTR(0,20,80,12,'Scavenger : ')
      call WRTSTR(0,20,130,22,'Hit CTRL-T to process.')
      pos = 1
199   continue
      if (pos.eq.1) then
         key = STRIN(0,130,80,40,scnam)
      endif
      if ((key.eq.retk).or.(key.eq.tabk).or.(key.eq.downk)) then
         pos = pos + 1
         if (pos.gt.1) pos = 1
      elseif ((key.eq.rtabk).or.(key.eq.upk)) then
         pos = pos - 1
         if (pos.lt.1) pos = 1
      elseif (key.eq.ctrlt) then
         call domod1(scnam,7)
         call WRTSTR(0,500,80,4,'Done')
         key=esck
      endif
      if (key.ne.esck) goto 199
      call WIPSCR(0)
      return
      end
C *****************************************************************************
      subroutine domod1(fnam,lev)
      implicit integer*2 (C)

      character*40 fnam
      integer*2 lev
$include: 'rigdat.inc'
      real*4 temp(4000),tim(4000)
      real*4 avg
      integer*2 avglim,i,j,avgcnt,stunit,ferr,chkio,err,pos
      character*25 prstr1,prstr2

      avglim = stepno
      avg = 0.0
      do 99 i = 1,avglim
         avg = avg + real(instru(i,lev))
99    continue
      avgcnt = int(avg/(real(avglim)))
      stunit = int(abs(step*40.95))
      j = 1
      do 199 i = (avglim+1),samps
         temp(j) = (real(instru(i,lev)-avgcnt))/(real(stunit))
         tim(j) = real(j)/hertz
         j = j + 1
199   continue
      open(6,FILE=fnam,STATUS='NEW',IOSTAT=ferr)
      j = 1
      do 299 i = (avglim+1),samps
         prstr1 = '                         '
         call numstr(3,8,'(E17.10)',tim(j),17,prstr1)
         call copyst(20,prstr1,0,1,' ',0,1)
         pos = CMPSTR(17,prstr1,1,'.')
         call copyst(20,prstr1,(pos-1),1,'0',0,1)

         prstr2 = '                         '
         call numstr(3,8,'(E17.10)',temp(j),17,prstr2)
         call copyst(20,prstr2,0,1,' ',0,1)
         pos = CMPSTR(17,prstr2,1,'.')
         call copyst(20,prstr2,(pos-1),1,'0',0,1)
         write(6,600,iostat=chkio,err=699) prstr1,prstr2
         j = j + 1
299   continue
600   format(a,' ',a)
      if (chkio.eq.0) goto 651
699   call ERTONE()
651   continue
```

```
      close(6)
      return
      end
C ***************************************************************************
      subroutine pldata()
      implicit integer*2 (e,I,g)
$include: 'rigdat.inc'
$include: 'keys.inc'
      integer*2 i,j,key,pos,flflg,levflg,valflg
      real*4 totime,pmax,pmin
      call WIPSCR(0)
      call wrtitl(0,50,20,18,'Plotting Rig Data.')
      call wrtitl(0,20,40,30,'Current rig data parameters :-')
      call WRTSTR(0,20,55,22,'Data from rig sampled ')
      pos = 20 + 22*9
      call prtnum(0,pos,55,1,4,'(i4)',4,samps)
      call WRTSTR(0,(pos+5*9),55,25,' times at a frequency of ')
      call prtnum(0,(pos+30*9),55,3,7,'(f7.2)',7,hertz)
      call WRTSTR(0,(pos+38*9),55,6,'Hertz.')

      if (tankno.eq.0) then
         pos = 20 + 10*9
         call WRTSTR(0,20,70,9,'Recleaner')
      elseif (tankno.eq.1) then
         pos = 20 + 8*9
         call WRTSTR(0,20,70,7,'Cleaner')
      elseif (tankno.eq.2) then
         pos = 20 + 10*9
         call WRTSTR(0,20,70,9,'Scavenger')
      elseif (tankno.eq.3) then
         pos = 20 + 8*9
         call WRTSTR(0,20,70,7,'Rougher')
      endif
      call WRTSTR(0,pos,70,8,'stepped ')
      call prtnum(0,(pos+9*9),70,3,7,'(f7.2)',7,step)
      call WRTSTR(0,(pos+17*9),70,13,'percent after')
      totime = real(stepno)/hertz
      call prtnum(0,(pos+31*9),70,3,7,'(f10.4)',10,totime)
      call WRTSTR(0,(pos+44*9),70,8,'seconds.')

      call wrtitl(0,20,90,19,'Axes Information :-')
      call WRTSTR(0,20,105,30,'Horizontal Axes : Time [secs] :')
      totime = real(samps)/hertz
      call prtnum(0,300,105,3,7,'(f10.4)',10,totime)
      call WRTSTR(0,20,120,43,
     +'Vertical Axes   : Percentage of Full Scale.')
      pmax = 100.0
      pmin = 0.0
      call WRTSTR(0,210,135,24,'Max :                Min : ')
      call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)

      flflg  = 1
      levflg = 1
      valflg = 1
      call wrtitl(0,20,150,18,'Plot Parameters :-')
      call WRTSTR(0,20,170,8,'Flows  :')
      call prsel(flflg,170,1)
      call WRTSTR(0,20,185,8,'Levels :')
      call prsel(levflg,185,1)
      call WRTSTR(0,20,200,8,'Valves :')
      call prsel(valflg,200,1)
      call WRTSTR(0,20,220,30,'Space bar to flip.  F5 to plot.')
      call WRTSTR(0,20,235,50,
     +'RETURN, TAB and cursor keys to move.  ESC to exit.')
      pos = 1
299   continue
      if (pos.eq.1) then
         key = edtflg(flflg,170)
      elseif (pos.eq.2) then
         key = edtflg(levflg,185)
      elseif (pos.eq.3) then
         key = edtflg(valflg,200)
      elseif (pos.eq.4) then
         key = getnum(0,260,135,3,7,'(g10.4)',10,pmax)
         if (pmax.gt.(100.0)) then
```

```
            call ERTONE()
            pmax = 100.0
         elseif (pmax.lt.(0.0)) then
            call ERTONE()
            pmax = 0.0
         endif
         call prtnum(0,260,135,3,7,'(g10.4)',10,pmax)
      elseif (pos.eq.5) then
         key = getnum(0,430,135,3,7,'(g10.4)',10,pmin)
         if (pmin.gt.pmax) then
            call ERTONE()
            pmin = pmax
         elseif (pmin.lt.(0.0)) then
            call ERTONE()
            pmin = 0.0
         endif
         call prtnum(0,430,135,3,7,'(g10.4)',10,pmin)
      endif
      if ((key.eq.tabk).or.(key.eq.downk).or.(key.eq.retk)) then
         pos = pos + 1
         if (pos.gt.5) pos = 1
      elseif ((key.eq.rtabk).or.(key.eq.upk)) then
         pos = pos - 1
         if (pos.lt.1) pos = 5
      elseif (key.eq.63) then
         call pltrig(flflg,levflg,valflg,pmax,pmin)
      endif
      if (key.ne.esck) goto 299
      call WIPSCR(0)
      return
      end
C *************************************************************************
      subroutine pltrig(flflg,levflg,valflg,pmax,pmin)
      integer*2 flflg,levflg,valflg
      real*4 pmax,pmin
$include: 'rigdat.inc'
$include: 'rigplt.inc'
      integer*2 i,j,npts

      call WIPSCR(1)
      call DISP(1)
      xp(1) = real(samps)/hertz
      xp(2) = 0.0
      xp(3) = 0.0
      yp(1) = pmax
      yp(2) = pmin
      yp(3) = pmin

      do 199 i = 0,(samps-1)
         time(i+1) = real(i)/hertz
199   continue
      call WRTSTR(1,145,158,7,'Rougher')
      call WRTSTR(1,145,297,9,'Scavenger')
      call WRTSTR(1,500,297,7,'Cleaner')
      call WRTSTR(1,500,158,9,'Recleaner')
      call WRTSTR(1,150,313,46,
     +'V - Valve          F - Flow          L - Level')
      call MOVE(4,301)
      call DLINE(715,301)
      call MOVE(4,162)
      call DLINE(715,162)
      call MOVE(365,301)
      call DLINE(365,22)
      call WRTSTR(1,170,18,42,
     +            'Flotation Simulator : Open Loop Step Test.')
      call MOVE(4,22)
      call DLINE(715,22)

      grfdim(1) = 10
      grfdim(2) = 150
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,roscal,rocent)
      call WRTSTR(1,35,38,3,'[%]')
      call WRTSTR(1,310,158,3,'[s]')
```

```
      grfdim(1) = 10
      grfdim(2) = 290
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,scscal,sccent)
      call WRTSTR(1,35,178,3,'[%]')
      call WRTSTR(1,310,297,3,'[s]')

      grfdim(1) = 375
      grfdim(2) = 290
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,clscal,clcent)
      call WRTSTR(1,400,178,3,'[%]')
      call WRTSTR(1,675,297,3,'[s]')

      grfdim(1) = 375
      grfdim(2) = 150
      grfdim(3) = 340
      grfdim(4) = 110
      call dwaxes(1,1,1,xp,yp,grfdim,rcscal,rccent)
      call WRTSTR(1,400,38,3,'[%]')
      call WRTSTR(1,675,158,3,'[s]')

      if (flflg.eq.1) then
         call fixdat(percnt,instru,samps,4)
         call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
         call drwlet(1,time(samps),percnt(samps),rocent,roscal,'f')
         call fixdat(percnt,instru,samps,3)
         call drawpt(1,1,time,percnt,samps,sccent,scscal,2)
         call drwlet(1,time(samps),percnt(samps),sccent,scscal,'f')
         call fixdat(percnt,instru,samps,2)
         call drawpt(1,1,time,percnt,samps,clcent,clscal,2)
         call drwlet(1,time(samps),percnt(samps),clcent,clscal,'f')
         call fixdat(percnt,instru,samps,1)
         call drawpt(1,1,time,percnt,samps,rccent,rcscal,2)
         call drwlet(1,time(samps),percnt(samps),rccent,rcscal,'f')
      endif
      if (levflg.eq.1) then
         call fixdat(percnt,instru,samps,8)
         call drawpt(1,1,time,percnt,samps,rocent,roscal,2)
         call drwlet(1,time(samps),percnt(samps),rocent,roscal,'l')
         call fixdat(percnt,instru,samps,7)
         call drawpt(1,1,time,percnt,samps,sccent,scscal,2)
         call drwlet(1,time(samps),percnt(samps),sccent,scscal,'l')
         call fixdat(percnt,instru,samps,6)
         call drawpt(1,1,time,percnt,samps,clcent,clscal,2)
         call drwlet(1,time(samps),percnt(samps),clcent,clscal,'l')
         call fixdat(percnt,instru,samps,5)
         call drawpt(1,1,time,percnt,samps,rccent,rcscal,2)
         call drwlet(1,time(samps),percnt(samps),rccent,rcscal,'l')
      endif
      if (valflg.eq.1) then
         call fixval(percnt,time,npts,3)
         call drawpt(1,1,time,percnt,npts,rocent,roscal,2)
         call drwlet(1,time(npts),percnt(npts),rocent,roscal,'v')
         call fixval(percnt,time,npts,2)
         call drawpt(1,1,time,percnt,npts,sccent,scscal,2)
         call drwlet(1,time(npts),percnt(npts),sccent,scscal,'v')
         call fixval(percnt,time,npts,1)
         call drawpt(1,1,time,percnt,npts,clcent,clscal,2)
         call drwlet(1,time(npts),percnt(npts),clcent,clscal,'v')
         call fixval(percnt,time,npts,0)
         call drawpt(1,1,time,percnt,npts,rccent,rcscal,2)
         call drwlet(1,time(npts),percnt(npts),rccent,rcscal,'v')
         do 250 i = 1,50
            time(i) = real(stepno)/hertz
            percnt(i) = real(i-1)*2.0 + 1.0
250      continue
         call drawpt(1,1,time,percnt,50,rocent,roscal,1)
         call drawpt(1,1,time,percnt,50,sccent,scscal,1)
         call drawpt(1,1,time,percnt,50,clcent,clscal,1)
         call drawpt(1,1,time,percnt,50,rccent,rcscal,1)
      endif
      if (flflg.eq.1) then
         do 249 i = 1,50
```

```
            time(i) = int(i*(real(samps)/hertz)/50.0)
            percnt(i) = real(760)/40.95
249         continue
            call drawpt(1,1,time,percnt,50,rocent,roscal,1)
            call drawpt(1,1,time,percnt,50,sccent,scscal,1)
            call drawpt(1,1,time,percnt,50,clcent,clscal,1)
            call drawpt(1,1,time,percnt,50,rccent,rcscal,1)
         endif
         return
         end
C ********************************************************************************
         integer*2 function edtflg(flg,pos)
         implicit integer*2 (I)
         integer*2 flg,pos
         integer*2 key
399      continue
         call prsel(flg,pos,0)
         key = INKEY(1)
         if (key.eq.8192) then
            if (flg.eq.1) then
               flg = 0
            else
               flg = 1
            endif
         endif
         if (key.eq.8192) goto 399
         call prsel(flg,pos,1)
         edtflg = key
         return
         end
C ********************************************************************************
         subroutine prsel(flg,pos,lev)
         integer*2 flg,pos,lev
         call LEVEL(lev)
         if (flg.eq.0) then
            call WRTSTR(0,105,pos,3,'OFF')
         else
            call WRTSTR(0,105,pos,3,' ON')
         endif
         call LEVEL(1)
         return
         end
C ********************************************************************************
         subroutine fixval(percnt,time,npts,valv)
         real*4 percnt(4000),time(4000)
         integer*2 npts,valv
$include: 'rigdat.inc'

         if (valv.eq.tankno) then
            npts = 4
            time(1) = 0.0
            time(2) = real(stepno)/hertz
            time(3) = real(stepno)/hertz
            time(4) = real(samps)/hertz
            percnt(1) = invalv(valv+1)
            percnt(2) = invalv(valv+1)
            percnt(3) = invalv(valv+1) + step
            percnt(4) = invalv(valv+1) + step
         else
            npts = 2
            time(1) = 0.0
            time(2) = real(samps)/hertz
            percnt(1) = invalv(valv+1)
            percnt(2) = invalv(valv+1)
         endif

         return
         end

C ********************************************************************************

         subroutine drwlet(pg,x,y,cent,scal,let)
         integer*2 pg
         real*4 x,y
         real*8 cent(4),scal(2)
         character*1 let
```

```
      external GPAGE
      external PLOT
      integer*2 xp,yp

      xp = int((x-cent(3))*scal(1)+cent(1)) + 3
      yp = int(cent(2)-(y-cent(4))*scal(2))
      call GPAGE(pg)
      call LEVEL(1)
      if (let.eq.'f') then
         call MOVE(xp+3,yp)
         call DLINE(xp+3,yp-6)
         call DLINE(xp+7,yp-6)
         call MOVE(xp+4,yp-4)
         call DLINE(xp+5,yp-4)
      elseif (let.eq.'l') then
         call MOVE(xp+3,yp-6)
         call DLINE(xp+3,yp)
         call MOVE(xp+4,yp)
         call DLINE(xp+7,yp)
      elseif (let.eq.'v') then
         call MOVE(xp+3,yp-6)
         call DLINE(xp+6,yp)
         call DLINE(xp+9,yp-6)
      else
         call gnum(pg,(xp+10),yp,let)
      endif
      call LEVEL(1)
      return
      end

      subroutine pilod()
      implicit integer*2 (I)
$include: 'keys.inc'
$include: 'multi.inc'

      integer*2 chkio,i,j,ferr,key

      call WIPSCR(0)
      call prcon(pi)
      call WRTSTR(0,20,220,32,'Hit RETURN to load. ESC to quit.')

199   continue
      key = INKEY(1)
      if (key.eq.retk) then
         open(4,FILE='MRIG.MAT',STATUS='OLD',IOSTAT=ferr)
         if (ferr.eq.0) then
            do 550 i =1,4
               read(4,'(8g10.4)',iostat=chkio,err=599)
     +             (pi(i,1,j), j=1,2),
     +             (pi(i,2,j), j=1,2),
     +             (pi(i,3,j), j=1,2),
     +             (pi(i,4,j), j=1,2)
550         continue
            call prcon(pi)
            call WRTSTR(0,20,220,32,'New Multi-variable controller.  ')
         endif
      endif
      if ((key.ne.esck).and.(key.ne.retk)) goto 199
540   continue
      if (chkio.eq.0) goto 551
599   call ERTONE()
      call WRTSTR(0,20,220,32,'*** Error : No Load.            ')
551   continue
      call WRTSTR(0,20,235,24,'Hit any key to continue.')
      key = INKEY(1)
      call WIPSCR(0)
      return
      end

C *****************************************************************************
      subroutine prcon(pi)
      real*4 pi(4,4,2)
      integer*2 i,j

      do 777 i = 1,4
```

```
         do 778 j = 1,4
            call prtnum(0,(20+(j-1)*130),(20+(i-1)*45),3,7,'(g10.4)',
     +                  10,pi(i,j,1))
            call prtnum(0,(20+(j-1)*130),(36+(i-1)*45),3,7,'(g10.4)',
     +                  10,pi(i,j,2))
778      continue
777   continue
      return
      end
      subroutine rigstp()
      implicit integer*2 (I,O)
$include: 'keys.inc'
$include: 'rigdat.inc'
      integer*2 key,i,j,err,val
      integer*2 lastsa(8),delta(10,8),avgst(8)

      call WIPSCR(0)
      call doda()
      call wrtitl(0,50,20,21,'Performing Step Test.')
      call WRTSTR(0,20,40,23,'  Number of samples   :')
      call prtnum(0,230,40,1,4,'(i4)',4,samps)
      call WRTSTR(0,350,40,23,'Sample frequency [Hz] :')
      call prtnum(0,560,40,3,7,'(f10.4)',10,hertz)
      call WRTSTR(0,20,60,39,' Valve Initial States : Recleaner [%] :')
      call prtnum(0,375,60,3,7,'(f10.4)',10,invalv(1))
      call WRTSTR(0,20,75,39,'                        Cleaner   [%] :')
      call prtnum(0,375,75,3,7,'(f10.4)',10,invalv(2))
      call WRTSTR(0,20,90,39,'                        Scavenger [%] :')
      call prtnum(0,375,90,3,7,'(f10.4)',10,invalv(3))
      call WRTSTR(0,20,105,39,'                        Rougher   [%] :')
      call prtnum(0,375,105,3,7,'(f10.4)',10,invalv(4))
      call WRTSTR(0,20,125,23,'         Tank to step :')
      if (tankno.eq.0) then
         call WRTSTR(0,235,125,9,'Recleaner')
      elseif (tankno.eq.1) then
         call WRTSTR(0,235,125,7,'Cleaner')
      elseif (tankno.eq.2) then
         call WRTSTR(0,235,125,9,'Scavenger')
      elseif (tankno.eq.3) then
         call WRTSTR(0,235,125,7,'Rougher')
      endif
      call WRTSTR(0,350,125,23,'       Valve Step [%] :')
      call prtnum(0,560,125,3,7,'(f10.4)',10,step)
      call WRTSTR(0,20,140,23,'Step at Sample Number :')
      call prtnum(0,230,140,1,4,'(i4)',4,stepno)
      call WRTSTR(0,30,245,48,
     +'Hit RETURN to start test.  Hit ESC to quit test.')

333   continue
      key = INKEY(1)
      if (key.eq.retk) then
         call DISP(0)
         call WRTSTR(0,30,245,48,
     +    'Step test in progress.....                    ')
         call docard(stepno,1)
         val = int((100.0 - (invalv(tankno+1)+step))*40.95)
         err = OUTDA(tankno,val)
         call docard((samps-stepno),(stepno+1))
         call WRTSTR(0,280,245,9,'Finished.')
         call DELAY()
      elseif (key.eq.esck) then
         call DISP(0)
         call WRTSTR(0,30,245,48,
     +    'Step test cancelled.                          ')
         call DELAY()
      endif
      if ((key.ne.retk).and.(key.ne.esck)) goto 333
      call WIPSCR(0)

      return
      end
C ***************************************************************************
      subroutine docard(smpnum,starti)
      implicit integer*2 (I,O,R)
      integer*2 smpnum,starti
$include: 'rigdat.inc'
```

```
      integer*4 temper,sams
      integer*2 i,j,err,temp,clklo,clkhi,samlo,samhi
      integer*2 basadr,comreg,stareg,datreg
      integer*2 waitcm,writwt,readwt,crdclr,crdclk,crdpar,crdrd,crdadi
      integer*2 crdstp,crderr

      basadr = 748
      comreg = basadr + 1
      stareg = basadr + 1
      datreg = basadr
      waitcm = 4
      writwt = 2
      readwt = 5
      crdclr = 1
      crdclk = 3
      crdpar = 13
      crdrd  = 14
      crdadi = 12
      crdstp = 15
      crderr = 2

      call OUTP(comreg,crdstp)
      temp = INP(datreg)
      call crdrdy()
      call OUTP(comreg,crdclr)
      call crdrdy()
      call OUTP(comreg,crdclk)
      call WAITDT(stareg,writwt,0)
      temper = (int (1.0/(8.0*hertz*0.0000025))) + 1
      call CONVRT(clklo,clkhi,temper)
      call OUTP(datreg,clklo)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,clkhi)
      call crdrdy()
      call OUTP(comreg,crdpar)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,0)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,0)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,7)
      call WAITDT(stareg,writwt,0)
      sams = smpnum*8
      call CONVRT(samlo,samhi,sams)
      call OUTP(datreg,samlo)
      call WAITDT(stareg,writwt,0)
      call OUTP(datreg,samhi)
      call crdrdy()
      call OUTP(comreg,crdrd)
      do 100 i = 1,smpnum
         do 99 j = 1,8
            call WAITDT(stareg,readwt,1)
            instru((starti+i-1),j) = (int (INP(datreg)))
            call WAITDT(stareg,readwt,1)
            instru((starti+i-1),j) =
     +           (int (INP(datreg)*256)) + instru((starti+i-1),j)
99       continue
100   continue

      return
      end


C *************************************************************************
      subroutine crdrdy()
      integer*2 stareg,basadr
      integer*2 waitcm,writwt
      basadr = 748
      stareg = basadr + 1
      waitcm = 4
      writwt = 2
      call WAITDT(stareg,writwt,0)
      call WAITDT(stareg,waitcm,1)
      return
      end
```

```
C ********************************************************************************
      subroutine doda()
      implicit integer*2 (O,R)
$include: 'rigdat.inc'
      integer*2 i,err,val
      err = RESTDA()
      if (err.eq.04) then
         call OUTP(548,11)
      endif
      do 498 i = 1,4
         val = int((100.0 - invalv(i))*40.95)
         err = OUTDA((i-1),val)
498   continue
      return
      end

C ********************************************************************************
C ********************************************************************************
      subroutine opts()
      implicit integer*2 (I,g,S)
$include: 'keys.inc'
$include: 'rigdat.inc'
$include: 'setp.inc'
      integer*2 key,pos

      call WIPSCR(0)
      call wrtitl(0,50,20,21,'Step Test Parameters.')
      call WRTSTR(0,30,50,23,'  Number of samples   :')
      call WRTSTR(0,30,70,23,'Sample frequency [Hz] :')
      call WRTSTR(0,30,95,39,' Valve Initial States : Recleaner [%] :')
      call WRTSTR(0,30,110,39,'                       Cleaner   [%] :')
      call WRTSTR(0,30,125,39,'                       Scavenger [%] :')
      call WRTSTR(0,30,140,39,'                       Rougher   [%] :')
      call WRTSTR(0,30,160,55,
     +'         Tank to step :        (0-Recleaner  1-Cleaner)')
      call WRTSTR(0,30,175,55,
     +'                               (2-Scavenger  3-Rougher)')
      call WRTSTR(0,30,195,23,'       Valve Step [%] :')
      call WRTSTR(0,30,220,23,'Step at Sample Number :')
      call WRTSTR(0,500,95,7,'Setp : ')
      call prtnum(0,240,70,3,7,'(f10.4)',10,hertz)
      call prtnum(0,385,95,3,7,'(f10.4)',10,invalv(1))
      call prtnum(0,385,110,3,7,'(f10.4)',10,invalv(2))
      call prtnum(0,385,125,3,7,'(f10.4)',10,invalv(3))
      call prtnum(0,385,140,3,7,'(f10.4)',10,invalv(4))
      call prtnum(0,260,160,1,4,'(i1)',1,tankno)
      call prtnum(0,240,195,3,7,'(f10.4)',10,step)
      call prtnum(0,240,220,1,4,'(i4)',4,stepno)
      call prtnum(0,565,95,1,4,'(i5)',5,setpnt(1))
      call prtnum(0,565,110,1,4,'(i5)',5,setpnt(2))
      call prtnum(0,565,125,1,4,'(i5)',5,setpnt(3))
      call prtnum(0,565,140,1,4,'(i5)',5,setpnt(4))
      call WRTSTR(0,30,235,23,'Enter Comment         :')
      call WRTSTR(0,240,235,50,commt)
      pos = 1
199   continue
      if (pos.eq.1) then
         key = getnum(0,240,50,1,4,'(i4)',4,samps)
         if (samps.lt.1) then
            samps = 1
            call prtnum(0,240,50,1,4,'(i4)',4,samps)
            call ERTONE()
         elseif (samps.gt.4000) then
            samps = 4000
            call prtnum(0,240,50,1,4,'(i4)',4,samps)
            call ERTONE()
          endif
      elseif (pos.eq.2) then
         key = getnum(0,240,70,3,7,'(f10.4)',10,hertz)
         if (hertz.lt.(1.0)) then
            hertz = 1.0
            call prtnum(0,240,70,3,7,'(f10.4)',10,hertz)
            call ERTONE()
         elseif (hertz.gt.(1000.0)) then
            hertz = 1000.0
          . call prtnum(0,240,70,3,7,'(f10.4)',10,hertz)
```

```
      call ERTONE()
   endif
elseif (pos.eq.3) then
   key = getnum(0,385,95,3,7,'(f10.4)',10,invalv(1))
   if (invalv(1).lt.(0.0)) then
      invalv(1) = 0.0
      call prtnum(0,385,95,3,7,'(f10.4)',10,invalv(1))
      call ERTONE()
   elseif (invalv(1).gt.(100.0)) then
      invalv(1) = 100.0
      call prtnum(0,385,95,3,7,'(f10.4)',10,invalv(1))
      call ERTONE()
   endif
elseif (pos.eq.4) then
   key = getnum(0,385,110,3,7,'(f10.4)',10,invalv(2))
   if (invalv(2).lt.(0.0)) then
      invalv(2) = 0.0
      call prtnum(0,385,110,3,7,'(f10.4)',10,invalv(2))
      call ERTONE()
   elseif (invalv(2).gt.(100.0)) then
      invalv(2) = 100.0
      call prtnum(0,385,110,3,7,'(f10.4)',10,invalv(2))
      call ERTONE()
   endif
elseif (pos.eq.5) then
   key = getnum(0,385,125,3,7,'(f10.4)',10,invalv(3))
   if (invalv(3).lt.(0.0)) then
      invalv(3) = 0.0
      call prtnum(0,385,125,3,7,'(f10.4)',10,invalv(3))
      call ERTONE()
   elseif (invalv(3).gt.(100.0)) then
      invalv(3) = 100.0
      call prtnum(0,385,125,3,7,'(f10.4)',10,invalv(3))
      call ERTONE()
   endif
elseif (pos.eq.6) then
   key = getnum(0,385,140,3,7,'(f10.4)',10,invalv(4))
   if (invalv(4).lt.(0.0)) then
      invalv(4) = 0.0
      call prtnum(0,385,140,3,7,'(f10.4)',10,invalv(4))
      call ERTONE()
   elseif (invalv(4).gt.(100.0)) then
      invalv(4) = 100.0
      call prtnum(0,385,140,3,7,'(f10.4)',10,invalv(4))
      call ERTONE()
   endif
elseif (pos.eq.7) then
   key = getnum(0,565,95,1,4,'(i5)',5,setpnt(1))
   if (setpnt(1).lt.0) then
      setpnt(1) = 0
      call prtnum(0,565,95,1,4,'(i5)',5,setpnt(1))
      call ERTONE()
   elseif (setpnt(1).gt.4095) then
      setpnt(1) = 4095
      call prtnum(0,565,95,1,4,'(i5)',5,setpnt(1))
      call ERTONE()
   endif
elseif (pos.eq.8) then
   key = getnum(0,565,110,1,4,'(i5)',5,setpnt(2))
   if (setpnt(2).lt.0) then
      setpnt(2) = 0
      call prtnum(0,565,110,1,4,'(i5)',5,setpnt(2))
      call ERTONE()
   elseif (setpnt(2).gt.4095) then
      setpnt(2) = 4095
      call prtnum(0,565,110,1,4,'(i5)',5,setpnt(2))
      call ERTONE()
   endif
elseif (pos.eq.9) then
   key = getnum(0,565,125,1,4,'(i5)',5,setpnt(3))
   if (setpnt(3).lt.0) then
      setpnt(3) = 0
      call prtnum(0,565,125,1,4,'(i5)',5,setpnt(3))
      call ERTONE()
   elseif (setpnt(3).gt.4095) then
      setpnt(3) = 4095
```

```
            call prtnum(0,565,125,1,4,'(i5)',5,setpnt(3))
            call ERTONE()
         endif
      elseif (pos.eq.10) then
         key = getnum(0,565,140,1,4,'(i5)',5,setpnt(4))
         if (setpnt(4).lt.0) then
            setpnt(4) = 0
            call prtnum(0,565,140,1,4,'(i5)',5,setpnt(4))
            call ERTONE()
         elseif (setpnt(4).gt.4095) then
            setpnt(4) = 4095
            call prtnum(0,565,140,1,4,'(i5)',5,setpnt(4))
            call ERTONE()
         endif
      elseif (pos.eq.11) then
         key = getnum(0,260,160,1,4,'(i1)',1,tankno)
         if (tankno.lt.0) then
            tankno = 0
            call prtnum(0,260,160,1,4,'(i1)',1,tankno)
            call ERTONE()
         elseif (tankno.gt.3) then
            tankno = 3
            call prtnum(0,260,160,1,4,'(i1)',1,tankno)
            call ERTONE()
         endif
      elseif (pos.eq.12) then
         key = getnum(0,240,195,3,7,'(f10.4)',10,step)
         if (step.lt.(-100.0)) then
            step = -100.0
            call prtnum(0,240,195,3,7,'(f10.4)',10,step)
            call ERTONE()
         elseif (step.gt.(100.0)) then
            tankno = 100.0
            call prtnum(0,240,195,3,7,'(f10.4)',10,step)
            call ERTONE()
         endif
      elseif (pos.eq.13) then
         key = getnum(0,240,220,1,4,'(i4)',4,stepno)
         if (stepno.lt.1) then
            stepno = 1
            call prtnum(0,240,220,1,4,'(i4)',4,stepno)
            call ERTONE()
         elseif (stepno.gt.samps) then
            stepno = samps
            call prtnum(0,240,220,1,4,'(i4)',4,stepno)
            call ERTONE()
         endif
      elseif (pos.eq.14) then
         key = STRIN(0,240,235,50,commt)
      endif
      if ((key.eq.retk).or.(key.eq.tabk).or.(key.eq.downk)) then
         pos = pos + 1
         if (pos.gt.14) pos = 1
      elseif ((key.eq.rtabk).or.(key.eq.upk)) then
         pos = pos - 1
         if (pos.lt.1) pos = 14
      endif
      if (key.ne.esck) goto 199
      call WIPSCR(0)
      return
      end
C *****************************************************************************
C *****************************************************************************
      subroutine vwdata()
      implicit integer*2 (I)
$include: 'keys.inc'
$include: 'rigdat.inc'
      integer*2 i,j,key,lineno

      call WIPSCR(1)
      call DISP(1)
      call wrtitl(1,20,20,17,'Viewing Rig Data.')
      call WRTSTR(1,300,20,10,'Samples : ')
      call prtnum(1,390,20,1,4,'(i4)',4,samps)
      call WRTSTR(1,495,20,12,'Frequency : ')
      call prtnum(1,605,20,3,7,'(f7.2)',7,hertz)
```

```
      call WRTSTR(1,20,60,23,'Initial States : Rec = ')
      call prtnum(1,230,60,3,7,'(f7.2)',7,invalv(1))
      call WRTSTR(1,323,60,5,'Cl = ')
      call prtnum(1,368,60,3,7,'(f7.2)',7,invalv(2))
      call WRTSTR(1,461,60,5,'Sc = ')
      call prtnum(1,506,60,3,7,'(f7.2)',7,invalv(3))
      call WRTSTR(1,599,60,5,'Ro = ')
      call prtnum(1,644,60,3,7,'(f7.2)',7,invalv(4))

      call WRTSTR(1,20,40,17,'Tank stepped   : ')
      if (tankno.eq.0) then
         call WRTSTR(1,173,40,9,'Recleaner')
      elseif (tankno.eq.1) then
         call WRTSTR(1,173,40,7,'Cleaner')
      elseif (tankno.eq.2) then
         call WRTSTR(1,173,40,9,'Scavenger')
      elseif (tankno.eq.3) then
         call WRTSTR(1,173,40,7,'Rougher')
      endif
      call WRTSTR(1,300,40,10,'Step    : ')
      call prtnum(1,390,40,3,7,'(f7.2)',7,step)
      call WRTSTR(1,495,40,12,'At sample : ')
      call prtnum(1,605,40,1,4,'(i4)',4,stepno)
      call MOVE(4,64)
      call DLINE(715,64)

      call wrtitl(1,20,92,3,'No.')
      call wrtitl(1,100,77,7,'Rougher')
      call wrtitl(1,80,92,4,'Flow')
      call wrtitl(1,140,92,5,'Level')
      call wrtitl(1,250,77,9,'Scavenger')
      call wrtitl(1,240,92,4,'Flow')
      call wrtitl(1,300,92,5,'Level')
      call wrtitl(1,420,77,7,'Cleaner')
      call wrtitl(1,400,92,4,'Flow')
      call wrtitl(1,460,92,5,'Level')
      call wrtitl(1,570,77,9,'Recleaner')
      call wrtitl(1,560,92,4,'Flow')
      call wrtitl(1,620,92,5,'Level')
      call WRTSTR(1,20,312,52,
     +'PgUp, PgDn, Home and End to view data.  ESC to exit.')

      lineno = 1
100   continue
      do 199 i = 0,13
         if (lineno.le.samps) then
            call prtnum(1,10,(114+i*14),1,4,'(i4)',4,lineno)
            call prtnum(1,80,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,4))
            call prtnum(1,140,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,8))
            call prtnum(1,240,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,3))
            call prtnum(1,300,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,7))
            call prtnum(1,400,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,2))
            call prtnum(1,460,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,6))
            call prtnum(1,560,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,1))
            call prtnum(1,620,(114+i*14),1,4,'(i4)',4,
     +                  instru(lineno,5))
            lineno = lineno + 1
         else
            call WRTSTR(1,10,(114+i*14),72,
     +'
     +                  ')
         endif
199   continue
198   continue
      key = INKEY(1)
      if (key.eq.pgupk) then
         if (lineno.eq.15) then
            call ERTONE()
```

```
            endif
            lineno = lineno - 28
            if (lineno.lt.1) lineno = 1
         elseif (key.eq.pgdnk) then
            if (lineno.eq.(samps+1)) then
               call ERTONE()
            endif
         elseif (key.eq.homek) then
            if (lineno.gt.15) lineno = 1
         elseif (key.eq.endk) then
            lineno = samps - 13
         endif
         if ((key.eq.pgdnk).and.(lineno.gt.samps)) goto 198
         if ((key.eq.homek).and.(lineno.eq.15)) goto 198
         if ((key.ne.pgdnk).and.(key.ne.pgupk).and.(key.ne.endk).and.
        +    (key.ne.homek).and.(key.ne.esck)) goto 198
         if (key.ne.esck) goto 100
         call WIPSCR(1)
         return
         end
C ********************************************************************************
C ********************************************************************************
C ********************************************************************************
C ********************************************************************************
```

```
      subroutine prtu(ival,err,u1)
      real*4 ival(4,4),err(4)
      integer*2 u1(4)

      integer*2 i,j,k

      k = 4
      do 99 i = 1,4
         do 98 j = 1,4
            call prtnum(1,(20+(j-1)*110),(25+(i-1)*20),3,7,'(g10.4)',
     +                  10,ival(i,j))
98       continue
         call prtnum(1,(30+4*110),(25+(i-1)*20),1,4,'(i6)',6,u1(k))
         call prtnum(1,(40+5*110),(25+(i-1)*20),3,7,'(g10.4)',
     +                  10,err(i))
         k = k - 1
99    continue
      return
      end
PAGE 56,80

; MODULE : Macro routines.
;          ***************

TITLE Sets up the menu data segments.
;     *******************************

; Termination characters.
; ***********************
EOM     EQU     02      ; End of menu.
EOS     EQU     00      ; End of string.
EOT     EQU     04      ; End of table.
ESC     EQU     1BH     ; ESC character.
HLP     EQU     03      ; Start of help.

; Macro definitions.
; ******************
; Set up menu data area.
; **********************

STARTMENU       MACRO   MENU_NUM
        MENU_DATA       SEGMENT PUBLIC 'FAR_DATA'
MENU&MENU_NUM   LABEL BYTE
        MENU_DATA       ENDS
        TABLE_DATA      SEGMENT PUBLIC 'FAR_DATA'
        DW      MENU_NUM
        DW      MENU&MENU_NUM
        TABLE_DATA      ENDS
        MENU_DATA       SEGMENT PUBLIC 'FAR_DATA'
        ENDM

; End menu data area.
; *******************

ENDMENU MACRO
        MENU_DATA       ENDS
        ENDM

; ****************************************************************************

; Menu data areas.
; ****************

TABLE_DATA      SEGMENT WORD PUBLIC 'FAR_DATA'

TABLE_DATA      ENDS

MENU_DATA       SEGMENT WORD PUBLIC 'FAR_DATA'

MENU_DATA       ENDS

; ****************************************************************************
; The structure of a menu is described :-
;
;     STARTMENU   number
;                 DB    'option1',HLP,'Description of option1',0
```

```
;                DB     'option2',HLP,'Description of option2',0
;                .
;                .
;                DB     'option(n-1)',HLP,'Description of option(n-1)',0
;                DB     'option(n)',HLP,'Description of option(n)',0
;                DB     EOM
;     ENDMENU
;
; STARTMENU and ENDMENU are macros defined above, 'number' is parameter of
; the macro : STARTMENU.
; HLP and EOM have been defined in the equates above.
;
; 'number' defines the menu to be used when calling DOMENU from FORTRAN.
;
; The 'number' can be any value except 0, this is used by the system to
; define the end of the menu data area and tables.
;
; For a description of how the menu is displayed, see the routine : DOMENU
; which resides in the module DOMENU.ASM.
;
; ***************************************************************************
;
; Menu definition.
; ****************
STARTMENU       3
        DB      'Load',HLP,'Load control data file.',0
        DB      'Save',HLP,'Save control data to file.',0
        DB      'Control',HLP,'Multivariable control of rig.',0
        DB      'Yuk',HLP,'Single-variable control the rig.',0
        DB      'Group_plot',HLP,'Plot all the graphs.',0
        DB      'One_plot',HLP,'Plot on a single set of axes.',0
        DB      'PI_load',HLP,'Load new controller values.',0
            DB          'VSS',HLP,'Load VSS controller values.',0
            DB          'VSS_Control',HLP,'VSS control of the rig.',0
        DB      EOM
ENDMENU

STARTMENU       1
        DB      'Load',HLP,'Load data disk file into memory.',0
        DB      'Save',HLP,'Save data in memory to disk file.',0
        DB      'View',HLP,'To view the test data.',0
        DB      'Test',HLP,'Perform step test on the rig.',0
        DB      'Group_plot',HLP,'Plot data from all the tanks.',0
        DB      'One_plot',HLP,'Plot data on a single set of axes.',0
        DB      'Format',HLP,'The step test format.',0
        DB      'Reset_DA',HLP,'Set all D/A converters to zero.',0
        DB      'D/A_set',HLP,'Set all D/A converters to preset values.',0
        DB      'Control',HLP,'Find valve steady state operating settings.',0
        DB      'Normalise',HLP,'Modify data files.',0
        DB      EOM
ENDMENU

STARTMENU       12
        DB      '1_Modify',HLP,'Normalise data and generate data files.',0
        DB      '2_Modify',HLP,'Generate data for pretty print.',0
        DB      EOM
ENDMENU

STARTMENU       2
        DB      'Step_test',HLP,'To perform step test on the rig.',0
        DB      'Multi',HLP,'Execute multi-var. control.',0
        DB      'Quit',HLP,'End test software.',0
        DB      EOM
ENDMENU

STARTMENU       0
ENDMENU

; ***************************************************************************

        END

PAGE 56,132

; MODULE : ADIN
```

```
; ************

TITLE To read in data from a DT2801 A/D/A card.

DGROUP  GROUP   CONST, _BSS, _DATA
        ASSUME  DS: DGROUP
        ASSUME  SS: DGROUP
        ASSUME  ES: DGROUP

;--------------------------------------------------------------------
;                   EXTERNAL DECLARATIONS.                          :
;--------------------------------------------------------------------

;-----------   Subroutines.
        PUBLIC  RESTDA
        PUBLIC  OUTDA
        PUBLIC  WAITDT
        PUBLIC  INP
        PUBLIC  OUTP
        PUBLIC  DELAY
           PUBLIC       CONVRT
;--------------------------------------------------------------------
;                       EQUATES                                     :
;--------------------------------------------------------------------

;-----------   Card addresses.
BASE_ADR EQU     02ECH    ; Card base address.
COMMAND_REG      EQU     BASE_ADR+1      ; Command register address.
STATUS_REG       EQU     BASE_ADR+1      ; Status register address.
DATA_REG EQU     BASE_ADR ; Data register address.

;-----------   Card commands.
WAIT_CMD EQU     04H      ; Command wait.
WRITE_WAIT       EQU     02H      ; Write wait.
READ_WAIT        EQU     05H      ; Read wait.
CARD_CLEAR       EQU     01H      ; Clear the card.
CARD_CLOCK       EQU     03H      ;
CARD_PARAM       EQU     0DH      ;
CARD_READ        EQU     0EH      ;
CARD_ADIN        EQU     0CH      ; Read AD in.
CARD_STOP        EQU     0FH      ; Stop the card.
CARD_ERROR       EQU     02H      ;
ADIN_GAIN        EQU     00H      ; Zero gain on the card.
CHAN0   EQU      00H      ; Channel 0.

;--------------------------------------------------------------------
_BSS    SEGMENT WORD PUBLIC 'BSS'

_BSS    ENDS

CONST   SEGMENT WORD PUBLIC 'CONST'

CONST   ENDS

_DATA   SEGMENT WORD PUBLIC  'DATA'

_DATA   ENDS


_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS: _TEXT
        ASSUME  DS: DGROUP
        ASSUME  SS: DGROUP
        ASSUME  ES: DGROUP

;-------------------------------------------------------------------------
;
; Function:
; Inputs:       None.
; Outputs:
; Calls:
; Destroys: Flags.
;
; Description:
;
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------------
;
; Function: RESTDA
; Inputs:       None.
; Outputs:      To D/A channels.
;          AX - status of reset.
; Calls:WAITCRD
; Destroys: Flags.
;
; Description: Resets the D/A card.
;
;-------------------------------------------------------------------------------
RESTDA  PROC FAR

        PUSH    BX              ;
        PUSH    CX              ;
        PUSH    DX              ;
        MOV     BX,0            ;
NEXT_RESET:                             ;
        CMP     BX,5            ;
        JE      RESET_FAIL              ;
        MOV     CX,0            ;
        MOV     DX,0225H        ;
        MOV     AL,0            ;
        OUT     DX,AL           ;
        CALL    FAR PTR DELAY   ;
NEXT_CHK:       CMP     CX,5000         ;
        JE      NEXT_RESET              ;
        INC     CX              ;
        MOV     DX,0225H        ;
        IN      AL,DX           ;
        CMP     AL,04           ;
        JE      DA_RESET        ;
        JMP     NEXT_CHK        ;
RESET_FAIL:     XOR     AL,AL           ;
DA_RESET:       XOR     AH,AH           ;
        POP     DX              ;
        POP     CX              ;
        POP     BX              ;
        RET
RESTDA  ENDP

;-------------------------------------------------------------------------------
;
; Function: Output a value to D/A channel
; Inputs:       err = OUTDA(channel,value)
; Outputs:      err = 0 - no errors
;                 1 - low byte error
;                 2 - high byte error
; Calls:
; Destroys: Flags.
;
; Description:
;
;-------------------------------------------------------------------------------
OUTDA   PROC    FAR
        PUSH    BP              ;
        PUSH    ES              ;
        PUSH    BX              ;
        PUSH    CX              ;
        PUSH    DX              ;
        MOV     BP,SP           ;
        LES     BX,DWORD PTR [BP+14]    ;
        MOV     AX,WORD PTR ES:[BX]     ;
        MOV     CL,4            ;
        SHL     AX,CL           ;
        LES     BX,DWORD PTR [BP+18]    ;
        MOV     DX,WORD PTR ES:[BX]     ;
        SHL     DX,1            ;
        AND     DX,000EH        ;
        OR      DX,0001H        ;
        ADD     AX,DX           ;
        MOV     BX,AX           ;
        MOV     DX,0225H        ;
CHKDA1: IN      AL,DX           ;
        CMP     AL,0            ;
```

```
        JNE     CHKDA1          ;
        MOV     DX,0224H        ;
        MOV     AL,BL           ;
        CALL    FAR PTR DELAY   ;
        OUT     DX,AL           ;
        MOV     DX,0225H        ;
CHKDA2: IN      AL,DX           ;
        CMP     AL,010H         ;
        JNE     CHKDA2          ;
        MOV     DX,0224H        ;
        MOV     AL,BH           ;
        CALL    FAR PTR DELAY   ;
        OUT     DX,AL           ;
        POP     DX              ;
        POP     CX              ;
        POP     BX              ;
        POP     ES              ;
        POP     BP              ;
        RET     08
OUTDA   ENDP

;-------------------------------------------------------------------------------
;
; Function: Given a set of conditions ,wait for the conditions to be
;           met.
; Inputs:       call WAITDT(address,mask,case)
;            address - address of port
;               mask - condition flag (ANDed with input)
;           function - Function: 00 - FLAG TO BE CLEAR
;                                01 - FLAG TO BE SET.
; Outputs:      None.
; Calls: None.
; Destroys: Flags.
;
; Description:
;
;-------------------------------------------------------------------------------

WAITDT  PROC    FAR

        PUSH    BP              ;
        PUSH    ES              ;
        PUSH    AX              ;
        PUSH    BX              ;
        PUSH    CX              ;
        PUSH    DX              ;
        MOV     BP,SP           ;
        LES     BX,DWORD PTR [BP+24]    ;
        MOV     DX,WORD PTR ES:[BX]     ;
        LES     BX,DWORD PTR [BP+20]    ;
        MOV     CX,WORD PTR ES:[BX]     ;
        LES     BX,DWORD PTR [BP+16]    ;
        MOV     BX,WORD PTR ES:[BX]     ;
        MOV     BH,BL           ;
        MOV     BL,CL           ;
WAITNEXT:                               ;
        IN      AL,DX           ;
        AND     AL,BL           ;
        CMP     BH,0            ;
        JE      CHK_CLR         ;
        CMP     AL,0            ;
        JE      WAITNEXT        ;
        JMP     EXITWAIT        ;
CHK_CLR: CMP    AL,0            ;
        JE      EXITWAIT        ;
        JMP     WAITNEXT        ;
EXITWAIT:                               ;
        POP     DX              ;
        POP     CX              ;
        POP     BX              ;
        POP     AX              ;
        POP     ES              ;
        POP     BP              ;
        RET     0CH             ;
WAITDT  ENDP
```

```
;-------------------------------------------------------------------------------
;
; Function: To input data from a port in the I/O map.
; Inputs:       data = INP(address)
;           data - (integer*2) Data from port.
;           adress - (integer*2) adress of port.
; Outputs:
; Calls: None.
; Destroys: Flags.
;
; Description:
;
;-------------------------------------------------------------------------------
INP     PROC    FAR
        PUSH    BP              ;
        PUSH    ES              ;
        PUSH    BX              ;
        PUSH    DX              ;
        MOV     BP,SP           ;
        LES     BX,DWORD PTR [BP+12]    ;
        MOV     DX,WORD PTR ES:[BX]     ;
        IN      AL,DX           ;
        XOR     AH,AH           ;
        POP     DX              ;
        POP     BX              ;
        POP     ES              ;
        POP     BP              ;
        RET     04              ;
INP     ENDP

;-------------------------------------------------------------------------------
;
; Function: To output a byte to a port in the I/O map.
; Inputs:       call OUTP(address,data)
;           address - (integer*2) Address of port.
;              data - (integer*2) Data : truncated to a byte.
; Outputs:
; Calls:    None.
; Destroys: Flags.
;
; Description:
;   .
;-------------------------------------------------------------------------------
OUTP    PROC    FAR
        PUSH    BP              ;
        PUSH    ES              ;
        PUSH    AX              ;
        PUSH    BX              ;
        PUSH    DX              ;
        MOV     BP,SP           ;
        LES     BX,DWORD PTR [BP+14]    ;
        MOV     AX,WORD PTR ES:[BX]     ;
        LES     BX,DWORD PTR [BP+18]    ;
        MOV     DX,WORD PTR ES:[BX]     ;
        OUT     DX,AL           ;
        POP     DX              ;
        POP     BX              ;
        POP     AX              ;
        POP     ES              ;
        POP     BP              ;
        RET     08H             ;
OUTP    ENDP

;-------------------------------------------------------------------------------
DELAY   PROC    FAR

        PUSH    AX              ;
        MOV     AX,02710H               ;
CHKDELAY:       DEC     AX              ;
        CMP     AX,0            ;
        JNE     CHKDELAY        ;
        POP     AX              ;
        RET                     ;
DELAY   ENDP
;-------------------------------------------------------------------------------
```

```
CONVRT  PROC    FAR

        PUSH    BP              ;
        PUSH    ES              ;
        PUSH    AX              ;
        PUSH    BX              ;
        PUSH    CX              ;
        MOV     BP,SP           ;
        LES     BX,DWORD PTR [BP+14]    ;
        MOV     CX,WORD PTR ES:[BX]     ;
        LES     BX,DWORD PTR [BP+18]    ;
        MOV     AL,CH           ;
        XOR     AH,AH           ;
        MOV     WORD PTR ES:[BX],AX     ;
        LES     BX,DWORD PTR [BP+22]    ;
        MOV     AL,CL           ;
        XOR     AH,AH           ;
        MOV     WORD PTR ES:[BX],AX     ;
        POP     CX              ;
        POP     BX              ;
        POP     AX              ;
        POP     ES              ;
        POP     BP              ;
        RET     0CH             ;
CONVRT  ENDP


_TEXT   ENDS                    ;
                                ;
        END                     ;

;-------------------------------------------------------------------------------
PAGE 56,132

; MODULE : HELP routines.
; ***********************

TITLE HELP routines and utilities.
;     ***********************

; ******************************************************************************
; REVISION HISTORY :
; VERSION       BY              DATE        COMMENT
;    1.00       Ian Fisher      23/06/88    Creation.
;
; ******************************************************************************

; Procedures.
; ***********
        PUBLIC  INKEY2  ; HELP function keyboard function.
        PUBLIC  FLIPG1  ; Flip screen memory into buffer.

; INT 21H : Function calls and returns.
; *************************************
CHAR_TRUE       EQU     0FFH    ; Return if a char in buffer.
FLUSH_KB EQU    00CH    ; Function : flush buffer.
NO_PROCESS      EQU     000H    ; Function : no function,
                        ;            while flushing.
READ_KB  EQU    008H    ; Function : read ASCII char.
STATUS_KB       EQU     00BH    ; Function : read keyb. status.

; General data segment.
; *********************
DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS

; Menu data areas.
; ****************
MENU_DATA       SEGMENT WORD PUBLIC 'FAR_DATA'
PAGE1   DB      32*1024 DUP(0)
        DB      0
MENU_DATA       ENDS

DGROUP  GROUP DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE
```

```
        ASSUME DS:DGROUP
        ASSUME ES:DGROUP

;-------------------------------------------------------------------------
FLIPG1  PROC FAR
; Function: To flip page out/in of graphics page 1.
; Inputs:       FORTRAN : call FLIPG1(op)
;               op - (integer*2) The operation : 1 - flip page 1 into memory.
;                                                2 - flip memory into page 1.
; Outputs:      None.
; Calls:    None.
; Destroys: Flags.
;
; Description: The routine does a 32K dump from screen memory into a buffer
;              or vica-versa depending on user request.
;
;-------------------------------------------------------------------------
        PUSH    BP              ; Save the registers.
        PUSH    DS              ;
        PUSH    ES              ;
        PUSH    AX              ;
        PUSH    BX              ;
        PUSH    CX              ;
        PUSH    DX              ;
        MOV     BP,SP           ;
        LES     BX,DWORD PTR [BP+18]    ; Load the operation number.
        MOV     AX,WORD PTR ES:[BX]     ;
        CMP     AX,2            ;
        JE      FLIP_IN         ; If (operation = 1) then
        MOV     BX,0B800H               ;    Set source address in
        MOV     ES,BX           ;        ES:BX
        MOV     BX,0            ;
        MOV     AX,MENU_DATA    ;    Set destination address in
        MOV     DS,AX           ;        DS:AX
        MOV     AX,OFFSET MENU_DATA:PAGE1;
        JMP     FL_EXIT         ;
FLIP_IN:                        ; Elseif (operation = 2) then
        MOV     BX,MENU_DATA    ;    Set source address in
        MOV     ES,BX           ;        ES:BX
        MOV     BX,OFFSET MENU_DATA:PAGE1;
        MOV     AX,0B800H               ;    Set destination address in
        MOV     DS,AX           ;        DS:AX
        MOV     AX,0            ;
FL_EXIT:                        ; Endif
        MOV     CX,(16*1024)    ; Set count to 32K.
NEXT_FLIP:                              ; While (count > 0)
        MOV     DX,WORD PTR ES:[BX]     ;   Load word from source.
        MOV     WORD PTR ES:[BX],0      ;   Set source to zero.
        ADD     BX,2            ;  Increment source ptr.
        XCHG    AX,BX           ;
        MOV     WORD PTR DS:[BX],DX     ;   Load word into destination.
        ADD     BX,2            ;  Increment destination ptr.
        XCHG    AX,BX           ;
        LOOP    NEXT_FLIP               ; Endwhile.
                                ;
        POP     DX              ; Restore registers.
        POP     CX              ;
        POP     BX              ;
        POP     AX              ;
        POP     ES              ;
        POP     DS              ;
        POP     BP              ;
        RET     04H             ; Pop parameters and exit.
FLIPG1  ENDP                    ;

;-------------------------------------------------------------------------
INKEY2  PROC FAR
; NOTE : This routine is a duplicate of GETKEY. This routine was created for
;        the HELP facility as the GETKEY routine  is  not re-entrant and the
;        HELP facility requires keyboard inputs.
;
; Function: FORTRAN Version :To read in a single character.
; Inputs:       FORTRAN: key = INKEY2()
;       key - (integer*2) Return value from routine.
; Outputs:      AX  - contains  the  return  code  or return character for
;                 the calling routine.
```

```
; Calls:    INT21H.
; Destroys: Flags.
;
; Description: Routine returns the ASCII (upper byte or shifted left 8 times)
;              code of the key typed (the char is  not echoed to the screen).
;              The  routine  first flushes the keyboard buffer before waiting
;              for a key to be typed.
;
;-------------------------------------------------------------------------------
        PUSH    BP              ; Save the registers.
        PUSH    ES              ;
        PUSH    BX              ;
        MOV     BP,SP           ;
                                ;------------------------------
        MOV     AH,FLUSH_KB             ;   Set function to flush the
        MOV     AL,NO_PROCESS   ;   buffer and do nothing else.
        INT     21H             ;   Execute the function.
                                ;
        MOV     AH,READ_KB              ;   Set function to read char.
        INT     21H             ;   Execute function.
                                ;   (Function waits for key to
                                ;    entered before returning)
READ_CH: CMP    AL,0            ;   Has ALT, Function or cursor
        JNE     ONE_CH          ;   key been entered ?
        MOV     AH,STATUS_KB    ;    Keyboard status function.
        INT     21H             ;    Get status.
        CMP     AL,CHAR_TRUE    ;    If a key has been hit ?
        JNE     NO_CHAR         ;       then : read char,
        MOV     AH,READ_KB              ;              return = char
        INT     21H             ;       else : return = 0;
        XOR     AH,AH           ;
        JMP     CHKF            ;
NO_CHAR: XOR    AX,AX           ;    Zero rest of return value.
        JMP     CHKF            ;
ONE_CH:  MOV    AH,AL           ;
        XOR     AL,AL           ;
        JMP     CHKF            ; Exit.
CHKF:   MOV     SP,BP           ; Restore registers.
        POP     BX              ;
        POP     ES              ;
        POP     BP              ;
        RET     0H              ;
INKEY2  ENDP

CODE    ENDS
        END                     ;
;-------------------------------------------------------------------------------
PAGE 56,80

TITLE Get Time
;     *********

; Routines.
; *********
        PUBLIC  GETIME

; Variables.
; **********


; General data segment.
; *********************
DATA    SEGMENT PUBLIC 'DATA'

DATA    ENDS

; ******************************************************************************

DGROUP  GROUP DATA

CODE    SEGMENT 'CODE'

        ASSUME CS:CODE
        ASSUME DS:DGROUP
        ASSUME ES:DGROUP
```

```
;-------------------------------------------------------------------------------
GETIME   PROC FAR
; Function: To get current time in seconds.
; Inputs:       FORTRAN : call GETIME(secs,hundreds)
; Outputs:      secs - (integer*2) The current seconds.
;           hundreds - (integer*2) The current hundredths.
;           No hours or minutes.
; Calls:    INT 21H
; Destroys: Flags.
;
; Description:
;-------------------------------------------------------------------------------
        PUSH    BP
        PUSH    DI
        PUSH    SI
        PUSH    DS
        PUSH    ES
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     BP,SP
        MOV     AH,02CH
        XOR     AL,AL
        INT     21H
        LES     BX,DWORD PTR [BP+22]
        MOV     AL,DL
        XOR     AH,AH
        MOV     WORD PTR ES:[BX],AX
        LES     BX,DWORD PTR [BP+26]
        MOV     AL,DH
        XOR     AH,AH
        MOV     WORD PTR ES:[BX],AX
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        POP     ES
        POP     DS
        POP     SI
        POP     DI
        POP     BP
        RET     08H
GETIME   ENDP


;-------------------------------------------------------------------------------

CODE    ENDS                    ;
                                ;
        END                     ;
                                ;
```

APPENDIX 6: STATE SPACE SYSTEM CONTROLLABILITY AND OBSERVABILITY

The following two mathematical tests check for system controllability and state observability and can be found in [1].

Consider the following nth order state space system:

$$\mathbf{dx}/dt = \mathbf{Ax} + \mathbf{Bu}$$

$$\mathbf{y} = C_{sys}\mathbf{x} \qquad \text{...(a13)}$$

The system is said to be fully controllable if and only if:

$$P = [B\ AB\ A^{\wedge}2B\ \ldots\ A^{\wedge}n\text{-}1B] \qquad \text{...(a14)}$$

$$\text{Rank}\{P\} = n \qquad \text{...(a15)}$$

The system states are said to be fully observable if and only if:

$$Q = [C_{sys}T\ (C_{sys}A)^T\ (C_{sys}A^{\wedge}2)^T\ \ldots\ (C_{sys}A^{\wedge}n\text{-}1)^T] \qquad \text{...(a16)}$$

$$\text{Rank}\{Q\} = n \qquad \text{...(a17)}$$