AN IMPLEMENTATION OF A SELF TUNING CONTROLLER

Mark Alfred Heilbrunn

A Dissertation Submitted to the Faculty of Engineering,
University of the Witwatersrand, Johannesburg,
for the Degree of Master of Science.

Johannesburg 1982

DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted for the degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

M. Heilbrunn

MARK ALFRED HEILBRUNN

this 20th day of December, 1982.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

1) To Dr. I.J. Barker and Professor [illegible]. Rodd for their guidance and encouragement.

2) To the Council for Mineral Technology for their generous sponsorship of this project.

3) To the Electrical Engineering Department of WITS for the generous use of their equipment.

4) To Mrs. G. Sklar for typing this manuscript.

SYNOPSIS

An algorithm for a non-parametric Self Tuning Controller is derived. The controller equates open and closed loop dynamics, which precludes the necessity of pre-specifying a desired closed loop response. A simplified least squares method is used for estimation of the model. The stability and convergence properties of the controller are shown by computer simulation and by a microprocessor based implementation on a test flow rig.

LIST OF SYMBOLS

A, B, C	Process Parameter polynomials
$\underset{\sim}{\beta}$	Vector of Process Coefficients
a_i, b_i	Process Parameters
c_i	Process Coefficients
$d(k)$, $d(t)$, $D(z)$	Disturbance Signal
d	Process Steady State Offset
$e(k)$, $e(t)$, $E(z)$	Set Point Error
e_p	Prediction Error
ess	Steady State Set Point Error
E	Expectation Operator
fa	Process Frequency
fs	Sampling Frequency
F, G	Controller Parameter Polynomials

F_g	Filter Steady State Gain
G_c	Controller Transfer Function
G_f	Filter Transfer Function
G_p	Process Transfer Function
G_p'	Unity Gain Process Transfer Function
Li	Process Coefficients
N	Amount of Coefficients
P, Q, R	Polynomials in z^{-1}
Qi	Controller Coefficients
R1, R2	Filter Parameters
s	Laplacian Operator
Ts	Sampling Period
Tc	Process Time Constant
T	Parameter Polynomial in z^{-1}

$T1d$, $T1g$	Filter Time Constants
$U(k)$, $u(t)$, $U(z)$	Manipulated Variable (Pre Filter)
$\underset{\sim}{U}$	Vector of Process Inputs
$v(k)$, $v(t)$, $V(z)$	Manipulated Variable (Post Filter)
$V(y, y_m)$	Loss Function
$w(k)$, $w(t)$, $W(z)$	Set Point
$y(k)$, $y(t)$, $Y(z)$	Controlled Variable
Ym, $\hat{y}(t)$	Model Output
z^{-1}	Backward Shift Operator
θ	Vector of Process Parameters
ϕ	Vector of Process States
$\bar{\theta}_{LS}$	Vector of Least Squares Estimate of Process Parameters
Γ	Gain Factor

Tld, Tlg	Filter Time Constants
U(k), u(t), U(z)	Manipulated Variable (Pre Filter)
$\underset{\sim}{U}$	Vector of Process Inputs
v(k), v(t), V(z)	Manipulated Variable (Post Filter)
$V(y,y_m)$	Loss Function
w(k), w(t), W(z)	Set Point
y(k), y(t), Y(z)	Controlled Variable
Ym, $\hat{y}(t)$	Model Output
z^{-1}	Backward Shift Operator
θ	Vector of Process Parameters
ϕ	Vector of Process States
$\hat{\theta}_{LS}$	Vector of Least Squares Estimate of Process Parameters
Γ	Gain Factor

$1/K$ Process Steady State Gain

Δy, Δu, Δv, etc. Incremental Variables

$\hat{y}$, $\hat{L}i$, etc. Estimated or Predicted Values

1. INTRODUCTION

Up until the late 1950's, most control devices found on a typical plant were invariably pneumatic, as these were safer and more reliable than their vacuum tube counterparts. However, with the advance of electronic technology, these were gradually replaced by solid state controllers.

Nevertheless, whether pneumatic or electronic, conventional analog control systems suffer from extreme inflexibility. Each control loop function requires associated hardware to perform this function, and any desired control strategy has to be implementable in analog hardware, and strategy modification invariably requires hardware modification, a difficult and sometimes impossible constraint.

To overcome these problems, control system designers looked at the digital computer as early as the mid 1950's. Since then applications have mushroomed throughout industry. However, by far the majority of computers used have been in a supervisory capacity or for direct digital control performing a discreet equivalent of conventional analog control. Only in the last decade or so have control system researchers and designers sought to implement strategies which are uniquely suited for digital computers. The advent of the microprocessor in the mid 1970's, which has made computing power readily available, reliable and above all cheap, has clearly accelerated this process.

1.1 Industry Standard

Proportional Integral Derivative Control

A controller which has found widespread use in industry over a period of many years is the so called proportional-integral-derivative (PID) controller. These are usually placed in a conventional feedback loop as shown below.

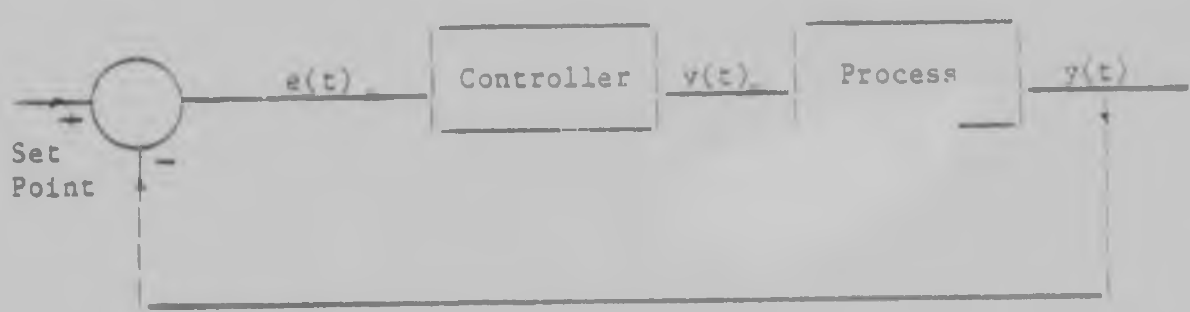


Figure 1
CONVENTIONAL FEEDBACK CONTROL LOOP

The controlled variable y(t) is measured by means of a sensor and the signal is fed back to the controller. Here it is subtracted from the set point (the desired value of y(t)) generating the error e(t). The control law, the defining element of the controller acts on this to generate the manipulated variable v(t). This manipulates the actuator to drive the error e(t) to zero. In this way the controlled variable is forced to the set point.

For a PID Controller, the manipulated variable $v(t)$ is related to the error $e(t)$ by the control law

$$v(t) = Kc \cdot \left(e(t) + \frac{1}{T_i} \cdot \int e(t)\, dt + Td \frac{de(t)}{dt} \right)$$

where Kc = Proportional Gain

T_i = Integral or Reset Time

Td = Derivative Time

The above three values generally appear as adjustments on the rear of the controller. The selection of their proper values is called tuning and is usually accomplished in one of 3 way:-

1) Trial and error.

2) Empirical methods based on some simple measurements taken from the controlled process.

3) Prediction on the basis of frequency response measurements made on the uncontrolled process.

The objectives in setting up and tuning a PID controller can include the following:-

a) Minimization of the error $e(t)$ following a disturbance wherever injected into the system.

b) Maximum rate of recovery to the set point after a disturbance.

c) Minimum steady state error both initially and due to changes in operating conditions.

Generally c) is satisfied if a sufficiently high gain constant Kc can be achieved consistent with process stability and speed of response following a disturbance or change in set point. Derivative control is advantageous in improving a) and b), while integral action may be used when c) is not satisfied without it.

1.2 Adaptive Controllers - An Overview

Tuning a P.I.D. can however be a difficult task, especially when the plant involved portrays the following characteristics:-

a) **Unknown Parameters** - When commissioning a control system or a new plant, the controller has to be tuned to suit the plant. The control action must be neither too sluggish and slow, nor must it be too rapid, causing saturation of variables, or even instability. To be able to properly tune a controller, at least the rudiments of the process dynamics must be known. If they are not, it is usually necessary to disturb the plant in some way in order to find them out. Thereafter, the tuned settings may be predicted. However, some plants may not be amenable to such disturbance, especially where financial or other loss may be incurred.

b) **Time Varying Parameters** - The above problem may be exacerbated in a situation where the process to be controlled has a transfer function that varies with time; i.e. the plant dynamics vary due to such things as changes in raw material or plant throughput etc. Should such a thing occur, a P.I.D. or any non-varying controller may fall out of tune. Inferior control would be the result. The problem then reverts to the point made above, i.e. the controller would need to be retuned.

c) **Non Linear Behaviour** - Process non linearity may be generally identified by the fact that Superposition does not hold, i.e. if a process input $u_1(t)$ produces output $y_1(t)$ and similarly $u_2(t)$ produces $y_2(t)$, then the input $u_1(t) + u_2(t)$ will in general not produce $y_1(t) + y_2(t)$ for a non linear plant, i.e. the process characteristics differ at different operating points If the plant is to operate in a different region, (caused for example, by a set point change) retuning may be necessary.

d) **Process Dead Time** - This is an important consideration when tuning. Suppose the plant contains a delay time T, then no matter what input to the plant occurs, there will be no response for time T. Therefore, the process engineer must choose the desired closed loop response to contain a delay time of at least T time units. Otherwise, the controller will require future values of the controlled variable in order to calculate the current value of the manipulated variable. This is obviously not physically realizable. For processes with significant dead times, even physically realizable controllers may result in an unstable closed loop response where the associated zero delay time plant would be stable. A process engineer's nightmare is of course the time varying delay time.

P.I.D. controllers are widely used because they are cheap, reliable and are remarkably effective in many processes. However, to cope with the abovementioned problems, detuning is often necessary to ensure stability over a wide range of conditions. The result is generally mediocre control.

This has provided the impetus for research into a form of controller which can adapt itself to its environments. The environment includes the system's input signals, the noise against which the system should discriminate and the factors which vary the system's parameters.

There is still much confusion concerning terminology in the area globally referred to as "adaptive control". Names such as adaptive, self organizing, self optimizing, self tuning and learning controllers are used - loosely and interchangeably. Definitions are vague and it is often difficult to draw the boundary lines between different types of controllers. It is even difficult to determine if a controller is adaptive or not, since many adaptive controllers can be regarded as non linear or time varying controllers. Wittenmark (1) notes the following points as basic functions common to most adaptive regulators:-

a) Identification of unknown parameters

OR

Measurement of a Performance Index.

b) Decision of the control strategy.

c) On-line modification of the parameters of the controller.

Differing methods of synthesizing the above functions result in different types of regulators.

Most adaptive control systems could be schematically drawn as in Figure 2 (overleaf).

This has provided the impetus for research into a form of controller which can adapt itself to its environments. The environment includes the system's input signals, the noise against which the system should discriminate and the factors which vary the system's parameters.

There is still much confusion concerning terminology in the area globally referred to as "adaptive control". Names such as adaptive, self organizing, self optimizing, self tuning and learning controllers are used - loosely and interchangeably. Definitions are vague and it is often difficult to draw the boundary lines between different types of controllers. It is even difficult to determine if a controller is adaptive or not, since many adaptive controllers can be regarded as non linear or time varying controllers. Wittenmark (1) notes the following points as basic functions common to most adaptive regulators:-

a) Identification of unknown parameters

OR

Measurement of a Performance Index.

b) Decision of the control strategy.

c) On-line modification of the parameters of the controller.

Differing methods of synthesizing the above functions result in different types of regulators.

Most adaptive control systems could be schematically drawn as in Figure 2 (overleaf).

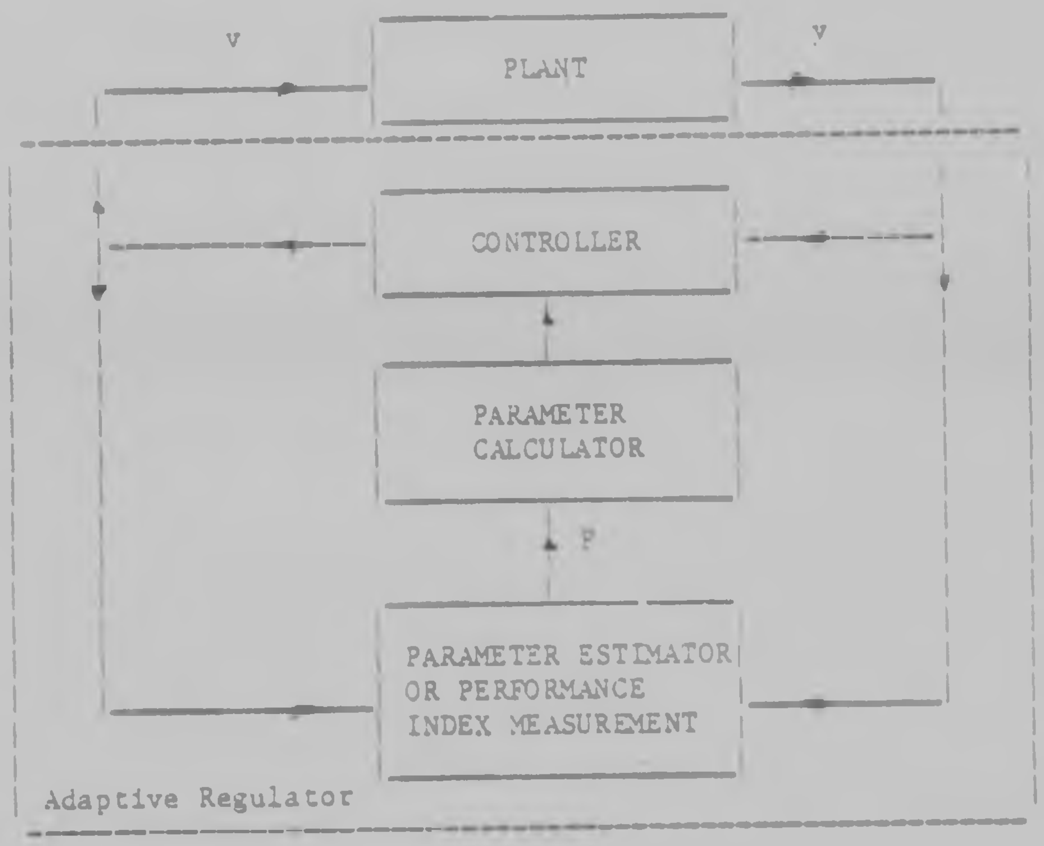


Figure 2

ADAPTIVE CONTROL SYSTEM - SCHEMATIC

The process to be controlled is modelled. The estimator attempts to find those parameters which define the model. Alternatively a Performance Index is evaluated. The vector P is the information that is passed to the parameter calculator which evaluates the nature of the controller itself. Finally the controller determines the input signal v to the process. The divisions as noted in the diagram are important and differentiate between types of adaptive control systems.

The rapid progress in micro-electronics, especially in the last decade, has made it possible to implement controllers simply and cheaply. There is now vigorous development of the field both at universities and in industry. However, no 'best' solution has yet been found, if it exists at all. What is clear however, is that 'better' solutions are being proposed as the understanding of the theory and practice of adaptive controllers advance.

It is the objective of this research project to implement a computer based adaptive controller as a direct replacement of a standard, industrial P.I.D. controller. It is hoped that the experience gained in doing so will enhance our perception of this exciting field.

The process to be controlled is modelled. The estimator attempts to find those parameters which define the model. Alternatively a Performance Index is evaluated. The vector P is the information that is passed to the parameter calculator which evaluates the nature of the controller itself. Finally the controller determines the input signal v to the process. The divisions as noted in the diagram are important and differentiate between types of adaptive control systems.

The rapid progress in micro-electronics, especially in the last decade, has made it possible to implement controllers simply and cheaply. There is now vigorous development of the field both at universities and in industry. However, no 'best' solution has yet been found, if it exists at all. What is clear however, is that 'better' solutions are being proposed as the understanding of the theory and practice of adaptive controllers advance.

It is the objective of this research project to implement a computer based adaptive controller as a direct replacement of a standard, industrial P.I.D. controller. It is hoped that the experience gained in doing so will enhance our perception of this exciting field.

1.3 Approaches to Adaptive Control

In his paper 'Theory and Applications of Adaptive Control', Aström (2) discusses three general methods of adaptive control which are most prevalent in current applications.

These are:-

a) Gain scheduling.
b) Model Reference Control.
c) Self Tuning Control.

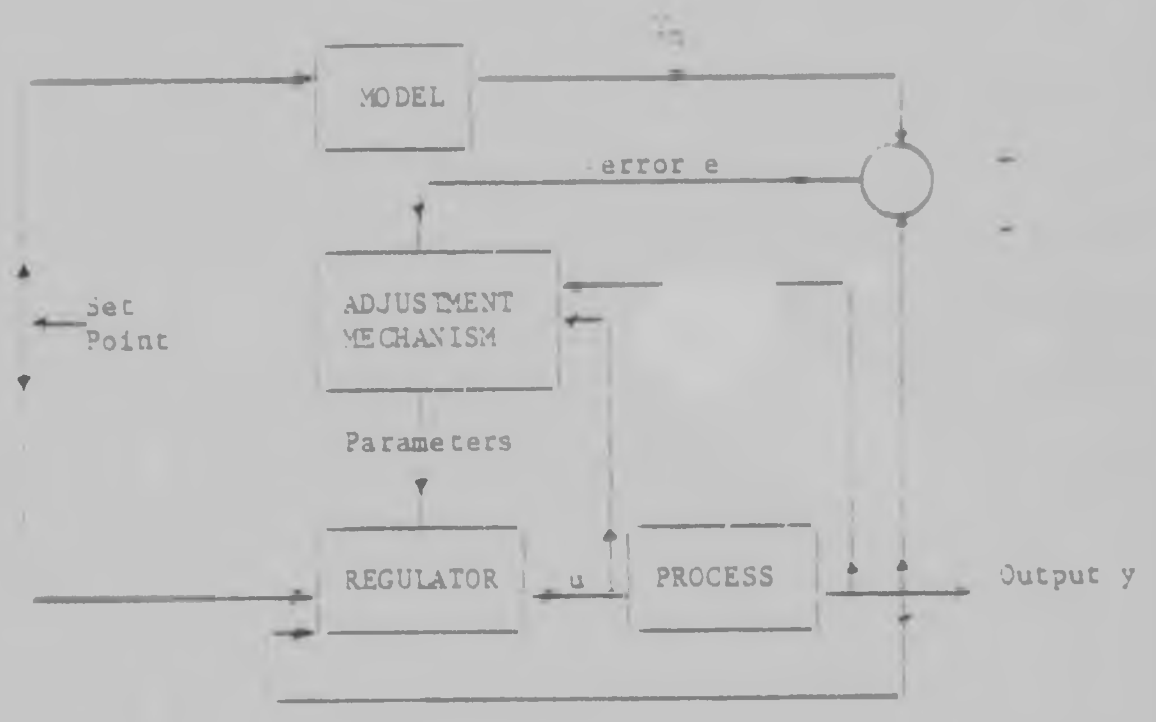
a) Gain Scheduling

This is the simplest method of the three. Here the regulator parameters are varied as functions of auxiliary variables which indicate changes in the process dynamics. The filling and emptying of a spherical tank is one such example, where level control would need a higher gain at mid-level than at either top or bottom extremes due to the non linearity involved. However, gain scheduling is considered by many to be a non linear controller rather than an adaptive one. This is due to the fact that the regulator parameters are changed in open loop, i.e. there is no feedback to compensate for an incorrect schedule. Nevertheless, gain scheduling appears to be the only 'adaptive control' widely available on a commercial basis. ('Operator Convenience is Key as Process Controllers Evolve' (3)).

3) Model Reference Adaptive Systems

Here the control specifications are given in terms of a reference model. The set point is applied to both the model and the closed loop regulator - process combination. The error between the model output y_m and the process output y drives the parameter adjustment mechanism of the regulator. In other words the intention is to force the regulator - process loop to behave in the same fashion as the chosen model. The idea is shown graphically in Figure 3.

Figure 3

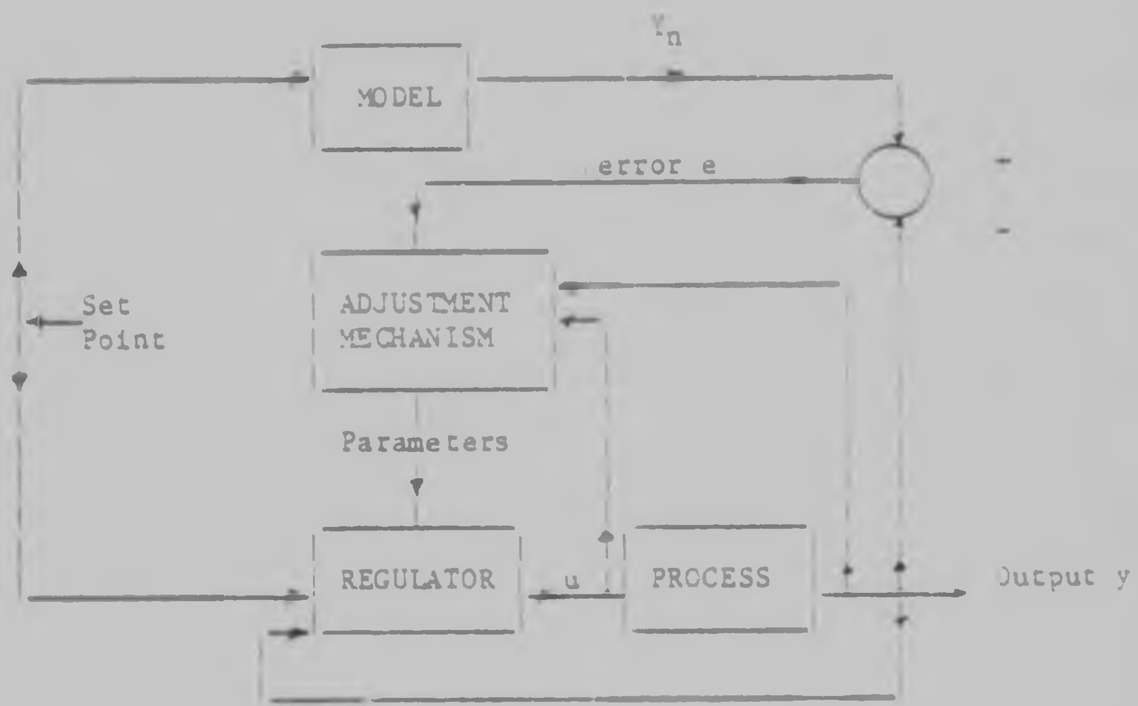


MODEL REFERENCE ADAPTIVE CONTROL

b) Model Reference Adaptive Systems

Here the control specifications are given in terms of a reference model. The set point is applied to both the model and the closed loop regulator - process combination. The error between the model output y_n and the process output y drives the parameter adjustment mechanism of the regulator. In other words the intention is to force the regulator - process loop to behave in the same fashion as the chosen model. The idea is shown graphically in Figure 3.

Figure 3



MODEL REFERENCE ADAPTIVE CONTROL

The problem is to determine the parameter adjustment mechanism so that a stable system results which forces the error to zero. The solution is however, non trivial and a number of papers are available on this topic, see (2) for references. Regulator realizability in the context of the process to be controlled must also be considered when choosing the reference model.

c) Self Tuning Regulation

Another means to adjust the parameters of a regulator is the method of self tuning. This involves choosing a control law as if the parameters of the process were known. The process paramters are then estimated by some identification scheme. The estimated parameters are then used in the control law to derive the control signal. Figure 4 overleaf, shows this schematically.

The problem is to determine the parameter adjustment mechanism so that a stable system results which forces the error to zero. The solution is however, non trivial and a number of papers are available on this topic, see (2) for references. Regulator realizability in the context of the process to be controlled must also be considered when choosing the reference model.

c) Self Tuning Regulation

Another means to adjust the parameters of a regulator is the method of self tuning. This involves choosing a control law as if the parameters of the process were known. The process paramters are then estimated by some identification scheme. The estimated parameters are then used in the control law to derive the control signal. Figure 4 overleaf, shows this schematically.

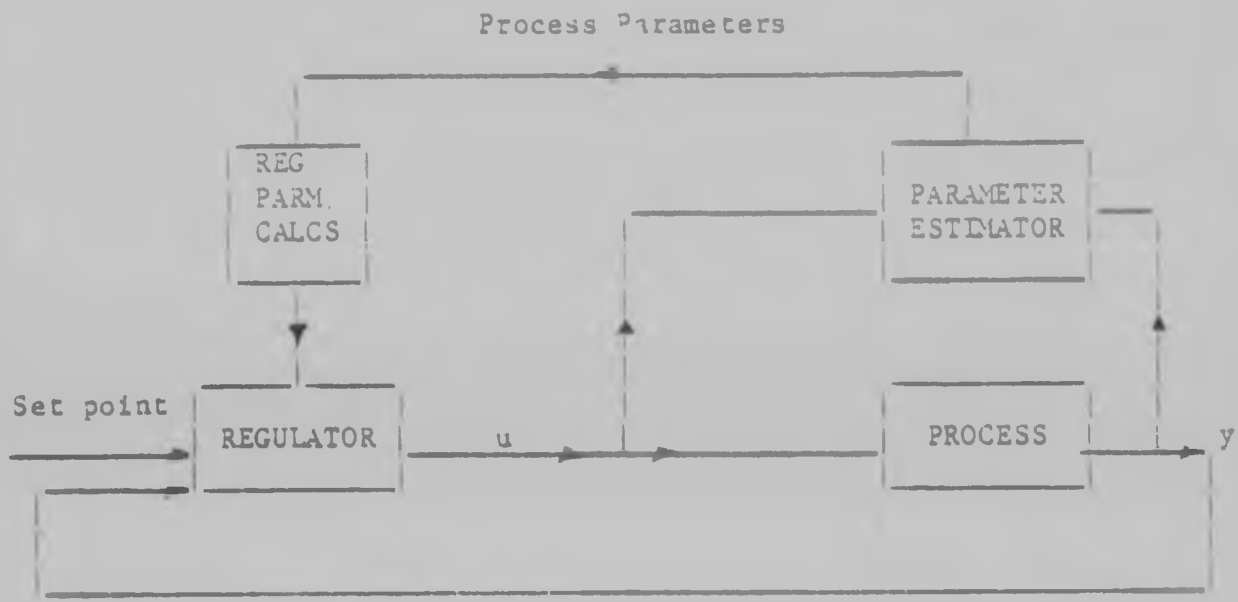


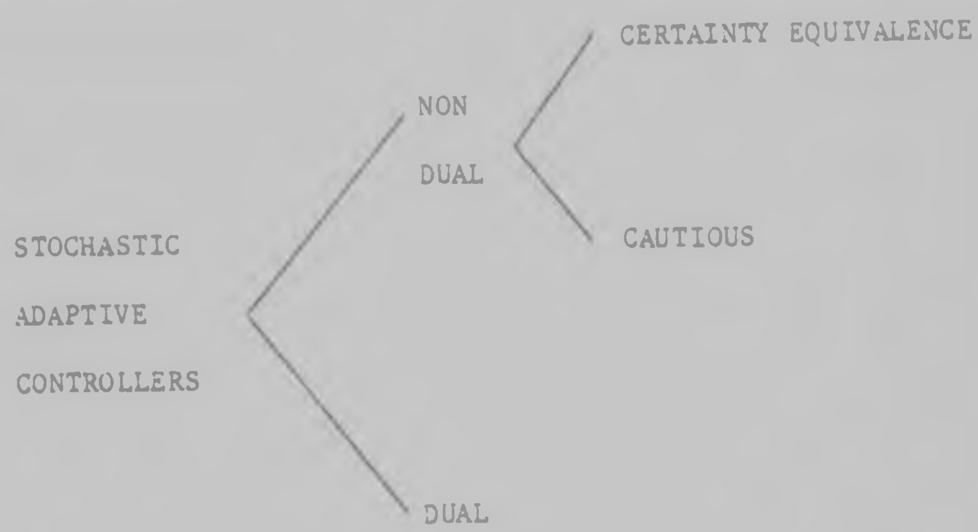
Figure 4

SELF TUNING CONTROL

This is the most flexible method of those discussed so far. Virtually any control law can be combined with some means of identification to provide a self tuning regulator.

Wittenmark Ref (1), in a highly illuminating article entitled 'Stochastic Adaptive Control Methods - A Survey' discusses those controllers which take into consideration the statistical nature of the fluctuations of the parameters and the disturbances acting on the system.

He classifies stochastic adaptive controllers according to the following diagram.



Feldbaum (Ref 4) postulated the 'dual' controller which effectively compromises between the two opposing actions involved in adaptive control. On the one hand 'good' control means minimum control effort to achieve minimum variation of a controlled variable from some desired value. While good plant identification requires 'large' control signals to excite the plant to be identified. The dual controller attempts to find the middle path between a probing action (to set the plant in motion) and a controlling action (to force the plant to some stationary state). The formal solution of the dual control problem has been postulated but leads to a functional equation which is difficult to solve in all but the simplest of cases.

The non dual controllers fall into two categories, i.e. certainty equivalence controllers and cautious controllers. Here, no action is taken to excite the process, bar those that are necessary to control. In other words, the identificaton aspect takes a back seat.

Cautious control is based on the separation principle which holds if it is possible to make a separation between the identification of the parameters of the process on one hand, and the determination of the parameters of the controller on the other. This has significance in that the controller parameters may be functions both of the estimated process parameters as well as the uncertainties of the parameters, i.e. the controller takes cautious action if the identification scheme produces poor results. Unfortunately this could lead to a problem situation because of the control-identification interaction mentioned in the previous paragraph. Poor identification may lead to cautious control, producing worse identification results and so on. In this way, the controller may inadvertently be switched off for some time until noise excites the system improving the identification accuracy.

The certainty equivalence controller is based on the certainty equivalence principle which holds if it is possible to first determine the controller as if the process to be controlled was completely known, and correctly identified, and then substitute the estimated process parameters as if they were the correct ones. Certainty equivalence has been successfully used as a simple design philosophy. The self tuning controller discussed beforehand falls directly under this category.

At this point it seems appropriate to reflect on those controllers already mentioned in order to adopt some or other implementable policy. In the case of dual, cautious and model reference adaptive controllers, there are difficulties in computing the true optimal controllers as the problem is highly non linear. Hence the most promising algorithms are those which, to save computation, approximate to the optimal in some way. One such approximation is the certainty-equivalent control which ignores interaction between estimation and control. The self tuning controller is one such certainty-equivalent law. With this in mind, the rest of this report will be devoted to the design of a self tuning controller based on the certainty-equivalence principle as a direct replacement for a P.I.D. controller.

2. APPROACHES TO SELF TUNING CONTROL

The certainty equivalence principle, when used as an ad hoc design basis suggests the use of identification and control as two seperate entities with a direct transfer of information between the model thus identified to the controller. With this in mind, it is worthwhile considering process identification and process control separately with a view both to choosing some identification - control combination (to be implemented), as well as to compare the self tuner thus chosen, with those methods noted in the literature.

2.1 Identification

Zadeh gives the following formulation of identification:

> 'Identification is the determination on the basis of input and output of a system within a specified class of systems to which a system under test is equivalent.' (7)

Using this definition we need to specifiy

i) A class of systems (usually called models).

ii) A class of input signals.

iii) The meaning of equivalence.

The system under test is usually termed the 'process' or the 'plant'. Equivalence is often defined in terms of a loss function or criterion which is a function of the process output y and the model output y_m, i.e.

$$V = V(y, y_m)$$

Two models M_1 and M_2 are said to be equivalent if the value for the loss function is the same for both models, i.e.

$$V(y, y_{m1}) = V(y, y_{m2})$$

When equivalence is defined by means of a loss function the identification is merely an optimisation problem, i.e. find a model M such that the loss function is as small as possible.

Automatically the following questions arise:-

Is the minimum achieved?
Is there a unique solution?
Can we restrict the choice of model to ensure uniqueness?

These questions are not always answerable, as the complexity of the systems involved may preclude analysis.

The models generally used fall into two distinct categories:-

i) Non-parametric representations, i.e. impulse responses, transfer functions, covariance functions, spectral densities, etc.

ii) Parametric models, such as state space models.

The difference between the two exists in that a non-parametric model has in principle no finite number of parameters which describe its input/output characteristics.

It is known that parametric models can give results with significant errors if the order of the model does not agree with the order of the process, (Ref 7).

The input signals can also be characterised:-

Impulse functions, step functions, white noise, sinusoidal signal, pseudo random binary noise (PRBS).

Whatever input signal is chosen, it must be capable of exciting all the modes of the process to be identified.

There exists no clearly defined method of choosing the model, the input signal and the criterion other than to state that the final aim of the identification, the process to be identified and the computational facilities available should influence the choice.

Some methods which have been used for those self tuners mentioned in the literature will be mentioned here.

2.1.1 One Shot Techniques

A very useful reference is Davis (8) 'System Identification for Self Adaptive Control' where he discusses the methods of random signal testing, concentrating on pseudo random noise and excitation signals in the identification of processes. He enumerates various methods for obtaining the impulse response curve and hence the frequency response curve and transfer function. These methods are well understood and tested, and are useful when non parametric representations are required. However they require plant disturbance and due to their 'one shot' nature, a further decision element is needed as to when reidentification is necessary.

There is one method of identification that warrants going into in some depth. It is the least square identification method usually used with parametric models. It is noteworthy in that by far the majority of self tuners based on parametric models mentioned in the literature use the least squares method or some extension of it, (see for example, References 9 to 17).

The form of model used is the so called generalised ARMA model (Auto Regressive Moving Average) which is a state space representation of the process to be identified, and has the remarkable property that the state is exactly given by the past inputs and outputs to/of the plant. Hence the model has parameters which are uniquely determined by the observed data.

The model is expressed in the following equation in discrete form:-

$$y(k) = a_1\, y(k-1) + a_2\, y(k-2) + \ldots a_n\, y(k-n)$$

$$+ b_1\, u(k-1) + b_2\, u(k-2) + b_3\, u(k-3) + \ldots b_n\, u(k-n)$$

where $y(k)$ = plant output at kth instant of time

$u(k)$ = plant input at kth instant of time

a_i, b_i = associated plant parameters

n = order of the system

The criterion chosen to be minimised is the output prediction error squared, (see Ref. [illegible] for some others).

i.e. If 1 is used as a predictor, using past values of y and u as measured.

$$\hat{y}(k) = \sum a_i\, y(k-i) + \sum b_i\, u(k-i)$$

where $\hat{y}(k)$ = predicted plant output

The prediction will be in error by an amount $e_p(k)$ such that

$$y(k) - \hat{y}(k) = e_p(k)$$

$$= y(k) - (\sum a_i \cdot y(k-i) + \sum b_i \cdot u(k-i)) \qquad \text{II (a)}$$

Therefore assume we have a set of N input and output data.

$$[u(o), u(1). \quad u(N), \; y(o), \; y(1) \quad y(N)]$$

and we wish to compute values for

$$\theta = (a_1, a_2, \ldots a_n, b_1 \ldots b_n)^T$$

which will best fit the observed data such that

$$V(\theta) = \sum_{k=n}^{N} e_p^{\,2}(k,\theta) \text{ a minimum} \qquad \text{II (b)}$$

To do this we introduce some matrix notation

$$\text{Let } \phi(k) = \{y(k-1), y(k-2), \ldots u(k-1) \ldots u(k-n)\}^T$$

$$\text{and } Y(N) = [y(n), \ldots y(N)]^T$$

$$\Psi(N) = \{\phi(n), \phi(n+1) \ldots \phi(N)\}^T$$

$$E(N;\theta) = \{e_p(n) \ldots e_p(N)\}^T \qquad \text{III}$$

$$\theta = \{a_1 \ldots a_n, b_1 \ldots b_n\}^T$$

We can then write $Y = \Psi\theta + E(N;\theta)$ which is another way ˆ saying, take the N sets of data n at a time and substitute them into equation III.

Therefore II (b) can be written

$V(\theta) = E^T(N;\theta)\ E(N;\theta)$ IV

Taking partial derivatives of equation IV with respect to θ and equating to zero we get the least square estimares of the parameters.

$\hat{\theta}LS = (\Psi^T\Psi)^{-1}\ \Psi^TY$ V

see Ref 7 for more detail.

The system is said to be parameter identifiable if one, and only one value of θ makes $V(\theta)$ a minimum.

There is an interesting addendum to the above method. Suppose that while taking the N sets of input/output data we realise that the process parameters may have drifted slightly. We may therefore wish to weight the recent data more than the older ones. This is easily accomplished by specifying the criterion as:-

$$V(\theta) = \sum_{k=n}^{N} W(k)e_p{}^2(k;\theta) = E^TWE$$

where $W(k)$ is some positive weighting function. The estimated parameters then become

$\hat{\theta}WLS = (\Psi^TW\Psi)\Psi^TWY$

A common choice for $W(k)$ is $(1-\gamma)\,\gamma^{N-k}$ where γ near 1 causes a long filter memory while smaller γ can track faster changes in process parameters.

The above method still suffers from the fact that it is 'one shot' or 'batch' in nature, since the formula presumes that one has a batch of data of length N. A self tuner using this method would also need to decide when retuning would be necessary.

The question then arises:-

Is there some way that the least square algorithm can be restructured to cater for sequentially available data such that the estimate can track the changes which may occur in θ if the computation is done over and over as N increases?

The recursive techniques of the following section deal with this problem.

2.1.2 Recursive Techniques

Fortunately the above equation can be manipulated to obtain identification recursively as the process develops such that the entire string of input/output data need not be brought in at each step. It can easily be shown that the least square estimate satisfies the following recursive equation, (see Ref 7 or 18).

A common choice for W(k) is $(1-\gamma)\gamma^{N-k}$ where γ near 1 causes a long filter memory while smaller γ can track faster changes in process parameters.

The above method still suffers from the fact that it is 'one shot' or 'batch' in nature since the formula presumes that one has a batch of data of length N. A self tuner using this method would also need to decide when retuning would be necessary.

The question then arises:-

Is there some way that the least square algorithm can be restructured to cater for sequentially available data such that the estimate can track the changes which may occur in θ if the computation is done over and over as N increases?

The recursive techniques of the following section deal with this problem.

2.1.2 Recursive Techniques

Fortunately the above equation can be manipulated to obtain identification recursively as the process develops such that the entire string of input/output data need not be brought in at each step. It can easily be shown that the least square estimate satisfies the following recursive equation, (see Ref 7 or 18).

$\hat{\theta}WLS(N+1) = \hat{\theta}WLS(N) + L(N+1)\{y(N+1) - \emptyset^T\hat{\theta}WLS(N)\}$ (a)

where

$P(N+1) = \frac{1}{\gamma}(I-L(N+1)\emptyset^T)P$ (b) V

and

$L(N+1) = \frac{P}{\gamma}\emptyset(a^{-1}+\emptyset^T\frac{P\emptyset}{\gamma})^{-1}$ (c)

where the weighting function $W = a\gamma^{N-k}$

These are the least square recursive algorithims and are calculated as follows:-

1) Select a, γ and N (a = γ = 1 is ordinary least square a = 1-γ, 0<γ<1 is exponentially weighted least squares).

2) Select initial values for P(N) and $\hat{\theta}(N)$

3) Collect y(o) ...y(N) and u(o)...u(N) and form $\emptyset^T(N+1)$

4) Let k $\leftarrow$ N

5) Solve L(k+1) using V (c)

6) Collect next input output values $y(k+1)$ and $u(k+1)$

7) Solve for $\hat{\theta}(k+1)$ using V (a)

8) Solve for $P(k+1)$ using V (b)

9) Form $\emptyset(k+2)$

10) Let $k=k+1$

11) Go to step 6

Note that we could have intuitively expected the form of equation V a since the next estimate of θ is given by the old estimate corrected by a term linear in the error between the observed output $y(N+1)$ and the predicted output $\emptyset^T\hat{\theta}(N)$.

We still have the problem of how to choose the initial conditions. Two methods are mentioned (Ref 7):-

1) Collect a batch of $N>2n$ data values and solve the batch formula once for $P(N)$, $L(N+1)$ and $\hat{\theta}(N)$.

2) Set $\hat{\theta}(N) = 0$, $P(N) = \alpha I$, where α is a large scalar

So far, the least square method has been presented with no comment about the possibility that the data may be subject to random effects i.e. in the real world most processes are stochastic in nature. The true prediction model is more likely to be

6) Collect next input output values $y(k+1)$ and $u(k+1)$

7) Solve for $\hat{\theta}(k+1)$ using V (a)

8) Solve for $P(k+1)$ using V (b)

9) Form $\phi(k+2)$

10) Let $k=k+1$

11) Go to step 6

Note that we could have intuitively expected the form of equation V a since the next estimate of θ is given by the old estimate corrected by a term linear in the error between the observed output $y(N+1)$ and the predicted output $\phi^T\hat{\theta}(N)$.

We still have the problem of how to choose the initial conditions. Two methods are mentioned (Ref 7):-

1) Collect a batch of $N>2n$ data values and solve the batch formula once for $P(N)$, $L(N+1)$ and $\hat{\theta}(N)$.

2) Set $\hat{\theta}(N) = 0$, $P(N) = \alpha I$, where α is a large scalar

So far, the least square method has been presented with no comment about the possibility that the data may be subject to random effects i.e. in the real world most processes are stochastic in nature. The true prediction model is more likely to be

$$\hat{y}(k) = \sum_{i=1}^{n} a_i y(k-i) + \sum_{i=1}^{n} b_i u(k-i) + \sum_{i=1}^{n} c_i e(k)$$

where $e(k)$ = a disturbance which is a sequence of independant random variables.

This equation is often written in terms of the polynomials A, B, and C and the backward shift operator Z^{-1}

i.e. $A(Z^{-1})\, y(t) = B(Z^{-1})\, u(t-1) + C(Z^{-1})\, e(t)$ VI

However, the least squares method gives biased estimates unless the true system can be described by equation VI with $C(Z^{-1}) = 1$.

This means that when the ARMA model has noise terms which are correlated from one equation to the next, least square will result in a set of estimate parameters $\hat{\theta}$ such that the mean value of $\hat{\theta}$ differs from the true value, θ_o, i.e. $E\hat{\theta}(N) - \theta_o = b$ where $b \neq o$ and E = Expectation Operator.

To overcome this problem of bias, many other schemes have been introduced, e.g. Extended Least Square Method, Maximum Liklihood Method, Levin's Method, etc. Further information on these can be found in References 7 and 18. All involve more computation or more prior knowledge about the process to be identified (or both) than simple least squares.

There are, of course, many different ways to obtain algorithms for real time recursive identification. But those that are computationally easy to implement are of chief interest.

Practically all methods yield algorithms with the structure:-

$$\hat{\theta}(N+1) = \hat{\theta}(N) + \Gamma(N) \quad e(N)$$

where $\Gamma(N)$ is a gain factor of varying complexity

$e(N)$ is a generalised error such as output prediction error

Compare the above equation with equation V (a).

Some of the better known methods (see Ref 7) are

Steepest Descent
Newton's Method
Stochastic Approximation
Gradient Method

The stochastic approximation method will be taken as an example.

Here $\hat{\theta}(k)$ represents the constant model parameters during the kth measurement interval. The next parameter vector is chosen as the old vector, corrected with a quantity proportional to the gradient of the error function. A normal regressive function can be taken as the model with

$$\hat{y}(k) = b_1u(k-1) + b_2u(k-2) \; ... \; b_mu(k-m)$$

i.e. based on the parameters and inputs only.

The following matrics can be formed:-

$$\hat{\theta}(k) = \begin{bmatrix} b_1(k) \\ \vdots \\ b_m(k) \end{bmatrix} \qquad U(k) = \begin{bmatrix} u(k-1) \\ \vdots \\ u(k-m) \end{bmatrix}$$

parameter vector   input vector

Then $\hat{y}(k) = U^T(k)\,\hat{\theta}(k)$

and output prediction error $e_p(k) = y - U^T(k)\,\hat{\theta}(k)$

where y is as measured.

Eykhoff (18) solves the stochastic approximation formula

$$\hat{\theta}(k+1) = \hat{\theta}(k) - (1/2)\cdot \Gamma(k) \nabla_{\hat{\theta}} (e_p^2(k))$$

(where $\nabla_{\hat{\theta}}$ represents the gradient with respect to $\hat{\theta}$.)

to get

$$\hat{\theta}(k+1) = \hat{\theta}(k) + \Gamma(k)\cdot U(k+1)\cdot\{y(k+1) - U^T(k+1)\cdot\hat{\theta}(k)\} \qquad \text{VII}$$

which is the stochastic approximation algorithm.

For convergence $\Gamma(k)$ has to fulfill the conditions

$$\Gamma(k) > 0\ , \quad \sum_{k=0}^{\infty}\Gamma(k) = \infty \quad \text{and} \quad \sum_{k=0}^{\infty}\Gamma^2(k) < \infty$$

This can be fulfilled by $\Gamma(k) = c/k^{\alpha}$

with $c>0$ and $0,5 < \alpha \leq 1$

If the process is deterministic, i.e. any noise corrupting the ideal model output is negligible, Γ can be chosen as a constant.

2.2 Control

Having studied identification in general, and no explicit methods in particular, we can now turn our attention to the problem of finding suitable control algorithms. Though there are obviously many varied ways to go about this, this section will concentrate specifically on those algorithms mentioned in research papers of the last few years as having been used for self tuning control.

A natural step would seem to be the self tuning of a PID controller. There has been mention of this see e.g. References 5 and 6. Most of these methods use some perturbation signal to produce a non-parametric model and then use standard error criterion (e.g. integral squared error) to evaluate PID tuning parameters. To my knowledge, none have been an unqualified success. At any rate, given the computing power available today, at low cost, there is no reason why more complex algorithms (involving more than three tuning parameters) may not be implemented.

In the late [illegible] Åström and his co-workers produced a series of excellent theoretical and applications papers on the subject of self tuning controllers. Since then there has been widespread interest in the subject, and there is at present (1982) much on-going research in this exciting area.

Before discussing the control algorithms themselves, some terminology should be defined.

Controllers usually accomplish one of two tasks or both. These are:-

1) Control against random noise whenever introduced into the system. This is usually termed 'the regulator problem' and the associated machine is called a 'regulator'.

2) Follow a time varying reference value (set point) usually called the 'servo problem'. The associated machine is usually called a servo controller.

3) The word controller seems to cover either or both of the above two.

Furthermore self tuning controllers may be either explicit or implicit.

1) An explicit controller first identifies the parameters of the model of the process, then further calculation is necessary to calculate the parameters of the controller.

2) An implicit self tuner identifies the parameters of the controller directly.

2.2.1 The Minimum Variance Regulator and Adaptations

References 16, 17 and 20.

Aström showed that if the prediction model is assumed as previously:-

$$A(z)\ y(t) = B(z)\ u(t-k) + C(z)\ e(t) \qquad \text{VIII}$$

where A, B and C are polynomials in the forward shift operator z, e.g.

$$A(z) = z^n + a_1 z^{n-1} \ldots a_n$$

and k represents the time delay.

Then one can postulate and calculate control laws which minimise the criterion:-

$$V = \lim_{N \to \infty} E\left(\frac{1}{N} \sum_{i=1}^{N} y^2(i) + gu^2(i)\right)$$

However, the optimal solution requires solving steady state Ricatti equations. The situation can be simplified if there is no cost on the control, i.e. g=o. The criterion then reduces to minimising the variance of the output, i.e. the resulting controller is called a minimum variance regulator.

Aström proposed that the manipulated variable u(t) be calculated as:-

$$u(t) = \frac{-z^k G(z)}{B(z)F(z)} y(t) \quad \text{IX}$$

where $F(z) = z^k + f_1 z^{k-1} \ldots f_k$

$$G(z) = g_o\, z^{n-1} + g_1\, z^{n-2} \ldots g_{n-1}$$

which are determined from

$$Z^k\, C(z) = F(z)A(z) + G(z) \qquad \text{X}$$

Substitution of IX and X into VIII verifies that the minimum variance regulator can be interpreted as choosing the control signal such that the predicted value k+1 steps ahead will be equal to zero.

Aström also showed that using equation X, the predictor equation can be written as

$$y(t+k+1) + \alpha_1 y(t) + \alpha_m (t-m+1)$$
$$= \beta_o \{u(t) + \beta_1 u(t-1) \ldots + \beta_j u(t-j)\}$$
$$+ \quad \mathcal{E}(t+k+1) \qquad \text{XI}$$

and the controller equation can be written directly in terms of α_i and β_i

$$u(t) = \frac{1}{\beta_o} \{\alpha_1 y(t) + \dots \alpha_m y(t-m+1)\}$$

$$- \beta_1 u(t-1) - \dots - \beta_j u(t-j) \qquad \text{XII}$$

i.e. in implicit format.

Summing up, the algorithm involves:-

At the sampling period Ts determine the model parameters XI, α_i, β_i using a recursive least squares estimator.

Then determine the control variable from equation XII. These are repeated at every sampling period.

In order to evaluate the minimum variance regulator, Aström et al consider three areas for analysis:-

1) Overall stability of the closed loop system

2) Convergence of the regulator

3) Identifiability aspects.

1) Stability is obviously the most important property for an applied system. Aström uses the heuristic argument that if the estimated parameters at any one time are so bad that an unstable closed loop system results, then the resulting increase in input and output signals causes the estimates to rapidly approach their true values. The system will then restabilize. He also shows that provided

i) The time delay k of the process is known;

ii) The order of the system is not underestimated;

iii) The process to be controlled is minimum phase;

the least square estimator plus minimum variance controller will stabilize any linear time invariant process.

2) Aström proves that if the parameter estimates converge, the control law obtained is the minimum variance control law that could be computed if the parameters of the system were known. However, general results giving conditions for convergence are not available but simulations have shown that convergence is attainable in many instances (see Ref 16).

3) It is known that certain problems exist if identification is performed while the system is in closed loop, (see Ref 21, Survey Paper - Identification of Processes in Closed Loop - Identifiability and Accuracy Aspects).

The problem is overcome here in two ways:-

i) The feedback is time varying. (Constant feedback would cause identifiability problems).

ii) The first non-zero parameter is fixed to a given value (β_o in equation XI) such that

$0{,}5b < \beta_o < \alpha$, b = actual plant parameter

where β_o <0,5b gives an unstable algorithm

β_o too large gives slow convergence

A number of implementations of the minimum variance controller (self tuned) have been noted, see Ref 20. An industrial application of a self tuning regulator, by Borrison and Wittenmark. These have been generally successful, though some problems have been noted:-

i) The controller does not penalise excessive control action.

ii) Non-minimum phase systems may cause unstable closed loop systems.

iii) Set point following has not been included. (Though this has been added in Aström's later papers with the associated computational difficultie see Ref 16).

iv) The necessity to choose one parameter (β_o) for identifiability purposes detracts slightly from the 'self tuning' philosophy.

v) The order of the system must not be underestimated.

vi) The time delay k must be known as a' Priori.

To solve some of these problems, Clarke and Gawthrop of the University of Oxford designed and implemented a variation of the minimum variance regulator. The results are published in a series of articles, the seminal work being the report of the Department of Engineering, University of Oxford, entitled 'Feasibility Study of the Application of Microprocessors to Self Tuning Controllers', Report No. 1137/75 (Ref 23), see also Refs 14 and 15.

Instead of minimizing the output variance only, this method minimises the variance of an auxiliary output function given by $\emptyset(k)$

$$\emptyset(t) = P(z^{-1})\, y(t) + Q(z^{-1})\, u(t-k) - R(z^{-1})\, w(t-k)$$

where P, Q and R are polynomials in z^{-1} and w(t) is the set point.

This is another way of saying, minimise the criterion

$$I = E\{(\sum p_i y(t+k-i) - \sum r_i w(t-i))^2$$

$$+ (\sum q^1{}_i u(t-i))^2\}$$

v) The order of the system must not be underestimated.

vi) The time delay k must be known as a' Priori.

To solve some of these problems, Clarke and Gawthrop of the University of Oxford designed and implemented a variation of the minimum variance regulator. The results are published in a series of articles, the seminal work being the report of the Department of Engineering, University of Oxford, entitled 'Feasibility Study of the Application of Microprocessors to Self Tuning Controllers', Report No. 1137/75 (Ref 23), see also Refs 14 and 15.

Instead of minimizing the output variance only, this method minimises the variance of an auxiliary output function given by $\emptyset(k)$

$$\emptyset(t) = P(z^{-1})\, y(t) + Q(z^{-1})\, u(t-k) - R(z^{-1})\, w(t-k)$$

where P, Q and R are polynomials in z^{-1} and w(t) is the set point.

This is another way of saying, minimise the criterion

$$I = E\{(\sum p_i y(t+k-i) - \sum r_i w(t-i))^2 + (\sum q^1{}_i u(t-i))^2\}$$

Note that this includes set point following and a control cost on the plant input u(t). P, Q and R are specified by the user to obtain more general closed loop behaviour.

Note also that Q = R = 0 is the minimum variance regulator discussed before.

The problem reduces to predicting the output $\hat{\emptyset}(t)$ at time t+k and evaluating the input u(t) such that this prediction is set to zero.

Since QU(t) and RW(t) are known at time t, the problem is to predict the component of $\hat{\emptyset}(t+k)$ due to the output y(t), i.e. $\hat{\emptyset}y(t+k)$, where $\hat{\emptyset}y(t) = Py(t)$. The suggested estimation procedure is some form of recursive least squares, usually extended least square acting on $\hat{\emptyset}y(t)$.

The explicit expression for the plant input is then

$$u(t) = \frac{Rw(t) - \hat{\emptyset}y(t+k)}{Q}$$

and the closed loop properties are defined by

$$y(t) = \frac{z^{-k}B \quad R\,w(t)}{PB + QA} \qquad \text{XIII}$$

In terms of the set point only

The stability of the system is determined by the roots of

$$PB + QA = 0$$

i.e. when Q is negligible, the closed loop behaviour is determined by B, i.e. for a non-minimum phase plant we have unstable poles including the case where Q = 0 for the minimum variance regulator. The closed loop poles can therefore be modified by ensuring that the QA term dominates and even non-minimum phase system can be accommodated.

Clarke et al, Ref. 15, also discuss two important practical aspects. The first arises from the fact that system models assumed for self tuning are local linearizations to typically non-linear responses. The input/output signals are generally perturbations around non-zero mean levels. These are denoted by $\overline{W}$, $\overline{U}$, $\overline{Y}$ which are set point, manipulated variable (input), controlled variable (output) mean levels, respectively. These are not usually related only by the steady state gain of the model, so a control offset must be added for generality. This extra term is just a further parameter which may be evaluated with all the rest. However, there are ways of getting rid of it.

The following gives a pictorial example of this.

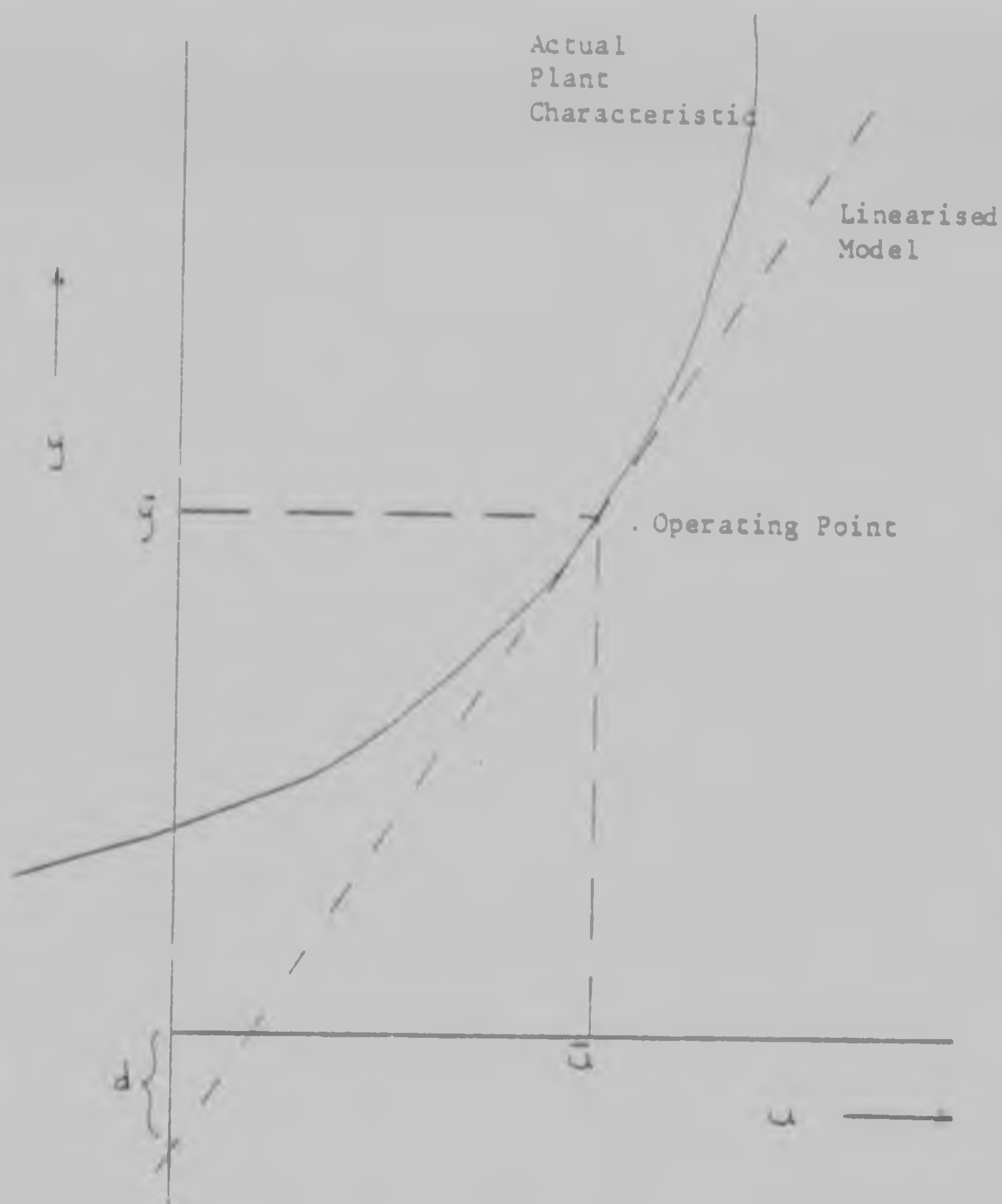


Figure 5

PROCESS CHARACTERISTICS

The previous model equation VI would become

$$A(z^{-1})\, y(t) = B(z^{-1})\, u(t-1) + C(z^{-1})\, e(t) + d$$

where d is the constant offset.

The second aspect is to ensure zero steady state error, i.e. that the plant output equals the set point at steady state.

Clarke mentions the following (Ref. 15):-

1) Insert an integrator into the loop after the self tuner, i.e. effectively compute increments in control signal. This allows d to be omitted from the estimation of parameters, but it detracts from the closed loop performance and in fact can seriously affect convergence of the self tuner.

2) Setting $Q = \lambda(1-z^{-1})$ ensures zero steady state error but d must be estimated as a further parameter. This method can be unstable for non-minimum phase systems.

3) Adding integrators into both P and R polynomials and d is (once more) eliminated. Again, however, the system fails for certain non-minimum phase systems.

4) The fourth method cascades a self tuner in the inner loop and an integrator in the outer loop. The integrator gain must be chosen.

These practical aspects are [illegible] here and the associated difficulties in solving them in order to point out some of the considerations to be taken into account when implementing a self tuner. The following points are noteworthy on the Clarke, Gawthrop controller [illegible].

1) Non-minimum phase systems can be handled by correct choice of polynomial Q.

2) The cost on the input [illegible] ensures that input actuators are not damaged, i.e. excessive control action is prevented.

3) Set point following is included.

4) Fixing parameter β_0 is not necessary but improves robustness.

5) The order of the system must not be underestimated.

6) The time delay k must be known.

7) P, the cost in the output should be chosen of the same order as the process to prevent inferior results.

8) Solutions to the constant offset parameter d, and zero steady state problems may also produce inferior results.

9) The control algorithm is suboptimal in comparison to the minimum variance controller in that minimization of the variance of the output is compromised to prevent excessive control action.

2.2.2 Pole/Zero Assignment Regulators

Another method of controller design is that of pole/zero placement based on classical control methods. Here the control objective is to move the closed loop poles/zeros to prespecified positions which define a transient response. Both regulator and servo problems have been tackled. One such regulator is discussed by Welstead Prager and Zanker, Ref 13 'Pole Assignment Self Tuning Regulator'.

They postulate a model:

$$y(t) = z^{-k}\frac{B(z^{-1})}{1 + A(z^{-1})}u(t) + \frac{1 + C(z^{-1})}{1 + A(z^{-1})}e(t) \qquad \text{XIV}$$

which is identical to the model described by equation VI with the first parameters a_1 and c_1 equal to unity.

The feedback regulator is given by

$$u(t) = \frac{G(z^{-1})}{1 + F(z^{-1})}y(t) \qquad \text{XV}$$

and the object of the self tuning pole assignment regulator is to automatically move the closed loop system poles from their open loop locations to the values specified by the polynomial $1 + T(z^{-1})$ where the zeros of $T(z^{-1})$ are preselected by the process engineer taking into consideration the process at hand.

Substituting XV into XIV gives the closed loop eq ation and equating with the required response

$$y(t) = \frac{1 + F(z^{-1})}{1 + T(z^{-1})} \; e(t)$$

the regulator parameters can be solved for (polymonials G and F) by solving

$$\{1 + A(z^{-1})\}\{1 + F(z^{-1})\} - z^{-k} B(z^{-1}) G(z^{-1}) \}$$
$$= \{1 + T(z^{-1})\}\{1 + C(z^{-1})\} \qquad \text{XVI}$$

where A, B and C are assumed known and T is chosen.

Self tuning then proceeds as follows:-

1) At each sample interval, the parameters of equation XIV are estimated by recursive least squares. (C is usually chosen as zero).

2) The estimated polynomials $\hat{A}$ and $\hat{B}$ are used to calculate F and G by equation XVI.

3) The control input u(t) is obtained from equation XV using F and G evaluated in 2) above.

The following points may be noted about the above mentioned self tuner:-

1) The principle of pole assignment self tuning, proved in Reference 13 states that if the system converges, it will converge to the desired closed loop configuration.

2) Process zeros are not cancelled, only the poles are shifted so the system does not suffer from instability due to the presence of non-minimum phase zeros.

3) Varying and unknown time delays can be accommodated but the numerator polynomial must be extended to ensure that the maximum transport delay expected is catered for. THis may cause problems with over parametization and the self tuning properties may be lost. However it is claimed by the author (Ref 13) that simulation has shown that successful regulation may still be achieved.

4) By the very nature of the controller, excessive control effort may be avoided.

5) Set point tracking can be included (See e.g. Ref 10 'Servo Self Tuners') but there is 'a significant increase in computational effort' (the author's own words).

Aström and Wittenmark (Ref 12 'Self Tuning Controllers Based on Pole Zero Placement') concentrates specifically on the servo problem. Both explicit and implicit algorithms are given. They discuss the problem of choosing a closed loop transfer function as this cannot be specified arbitrarily. Specifically, the delay time of the closed loop response must be at least as long as the processes. They note that open loop process zeros in the right hand plane cannot be cancelled and remain zeros of the closed loop system.

1) The principle of pole assignment self tuning, proved in Reference 13 states that if the system converges, it will converge to the desired closed loop configuration.

2) Process zeros are not cancelled, only the poles are shifted so the system does not suffer from instability due to the presence of non-minimum phase zeros.

3) Varying and unknown time delays can be accommodated but the numerator polynomial must be extended to ensure that the maximum transport delay expected is catered for. THis may cause problems with over parametization and the self tuning properties may be lost. However it is claimed by the author (Ref 13) that simulation has shown that successful regulation may still be achieved.

4) By the very nature of the controller, excessive control effort may be avoided.

5) Set point tracking can be included (See e.g. Ref 10 'Servo Self Tuners') but there is 'a significant increase in computational effort' (the author's own words).

Aström and Wittenmark (Ref 12 'Self Tuning Controllers Based on Pole Zero Placement') concentrates specifically on the servo problem. Both explicit and implicit algorithms are given. They discuss the problem of choosing a closed loop transfer function as this cannot be specified arbitrarily. Specifically, the delay time of the closed loop response must be at least as long as the processes. They note that open loop process zeros in the right hand plane cannot be cancelled and remain zeros of the closed loop system.

Two basic types of controllers/regulators have been discussed. These are:-

1) 'Optimal' controllers based on linear quadratic gaussian control theory. Specifically mentioned were the minimum variance regulator by Aström and co-workers, and the Clarke Gawthrop extension.

2) Pole Zero Assignment Controller based on classical control theory. Here, work has been done by Welstead and company, Aström and Company and the Clarke, Gawthrop team.

The above mentioned are certainly not an exhaustive review of the work that has been done in this sphere. However it is felt that most self tuners do fall into one of the two basic categories mentioned above. The link between these two, and model reference adaptive systems (MRAs) is the subject of current research (e.g. the Clarke Gawthrop controller may be considered as a model reference adaptive controller under certain conditions, see Ref. 19). All the self tuners mentioned have used some form of recursive least square as the identification procedure combined with a parametric model. The problems associated with each type of controller have also been mentioned.

3. AN EASIER METHOD

The ideal controller is a "black box". It has input connections, output connections, possibly some means of displaying controlled and manipulated variables anu that is all. There are no dials or knobs or switches to allow human interference. It is universaly transportable, handling all types of process under all conditions.

The ideal is approached to some extent by those self tuners already mentioned. Simulation and practice has shown that they do provide superior performance where constant controllers fail.

However they do need the following when commissioned:-

i) A rigid specification of the required closed loop characteristics.

ii) Certain information imbedded in the prediction model used.

That is to say, when these ST controllers are first placed within the structure of a process loop, certain control parameters have to be preset in terms of the above two points. This implies a certain a Priori knowledge of the process at hand, which detracts slightly from the self tuning philosophy. Prior knowledge of model order, plant dead time and controller realizability is needed before use on any plant. If self tuners are to replace P.I.D.'s on a broad scale, they must not only improve performance, but also minimise the human effort involved.

3.1 Motivation and Derivation of a Self Tuner

Considering some prime characteristics which should be sought after, when replacing a P.I.D. by a self tuning controller.

From a user's point of view:-

i) Transportability between differing plants;

ii) Compatability with the skills of the plant operator;

iii) Simplicity of implementation and maintenance;

iv) Plant disturbance should be avoided.

From a designer's point of view a self tuner should capable of:-

i) Identifying and controlling a plant when commissioning;

ii) Handling non-linear plant behaviour;

iii) Handling time varying plant characteristics;

iv) Solving both servo and regulator problems;

which means to

v) Ensure zero steady state error at all times;

vi) Cope with transport delays automatically;

vii) Cope with non-minimum phase plants;

viii) Ensure system stability at all times.

With the above discussion in mind, a control algorithm, the basis of which can be found in reference 24 ('Synthesizng a Digital Algorithm for Optimised Control') by Tu and Tsing, is thought suitable for implementation. It will be shown that this controller has a number of interesting characteristics while some slight modification to it, allows for greater flexibility.

The basis of the algorithm is the performance criterion that equates open and closed loop dynamics. This automatically ensures physical realizability of the closed loop system. A short derivation now follows.

Assume the following closed loop system:-

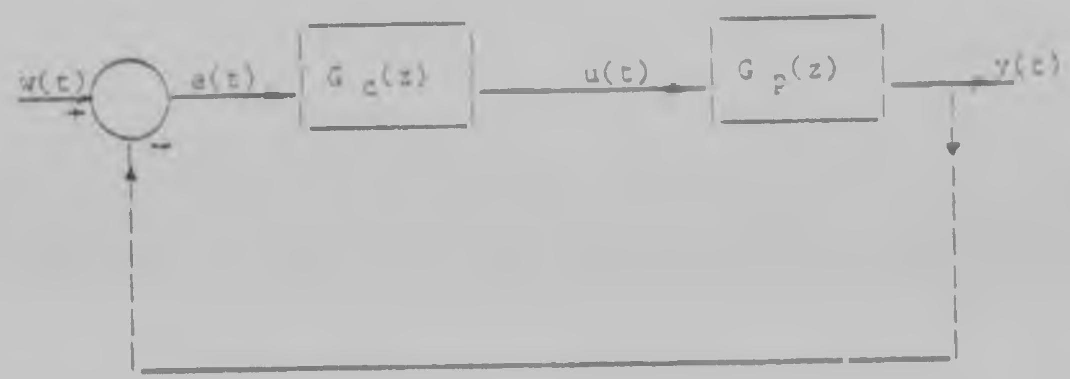


Figure 6
CLOSED LOOP SYSTEM

w(t) is the set point

where $G_p(z)$ is the plant transfer function

and $G_p(z) = \frac{1}{K} G'_p(z)$ such that $G'_p(z)$ has unity steady state gain

while $\frac{1}{K}$ denotes the steady state gain of $G_p(z)$.

$G_c(z)$ is the controller transfer function.

The feedback system will have a closed loop transfer function of

$$\frac{Y(z)}{W(z)} = \frac{G_p(z)\,G_c(z)}{1 + G_p(z)\,G_c(z)}$$

Tu and Tsing now equate open and closed loop dynamics and substituting

$G_p = \frac{1}{K} G_p'$ (dropping the z).

$$G_p' = \frac{(1/K)\,G_p'\,G_c}{1 + (1/K)\,G_p'\,G_c}$$

Solving for G_c gives

$$G_c = \frac{K}{1 - G_p'}$$

which leaves G_p' and K to be synthesized.

Now if a step input is applied to the plant the output is given as

$$Y(z) = \left(\frac{1}{K} - c_1\right)z^{-1} + \left(\frac{1}{K} - c_2\right)z^{-2} + \text{etc.}$$

where c_i is defined as below.

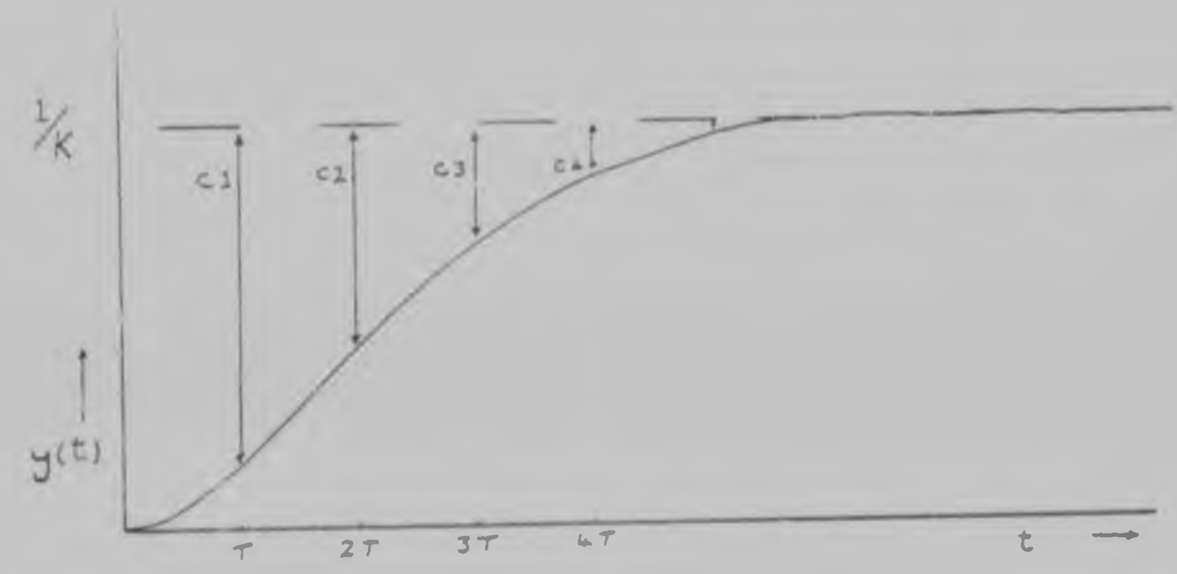


Fig 6a <u>Generalised Process Step Response Sampled at T Interval</u>

where the plant input $= \dfrac{z}{z-1} = u(z) =$ step

Therefore the plant dynamics denoted by G_p' is given as

$$G_p' = \frac{KY(z)}{U(z)} = \frac{z-1}{z} .K. \left\{\left(\frac{1}{K} - c_1\right) z^{-1} + \left(\frac{1}{K} - c_2\right) z^{-2} + \dots\right\}$$

$$= K\left\{\left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} + (c_2 - c_3)z^{-3} \dots\right\} \qquad \text{XVIII}$$

now

$$G_c = \frac{K}{1-G_p'} = \frac{U(z)}{E(z)}$$

Substituting [illegible] leads to

$$U(z) = U(z)z^{-1} - K\{c_1 z^{-1} + (c_2 - c_1)z^{-2} + (c_3 - c_2)z^{-3} \ldots\}U(z) + KE(z)$$

$$U(z) - U(z)z^{-1} = KE(z) - K\{c_1 z^{-1} + c_2 z^{-2} \ldots\}U(z)(1-z^{-1})$$

which in the time domain

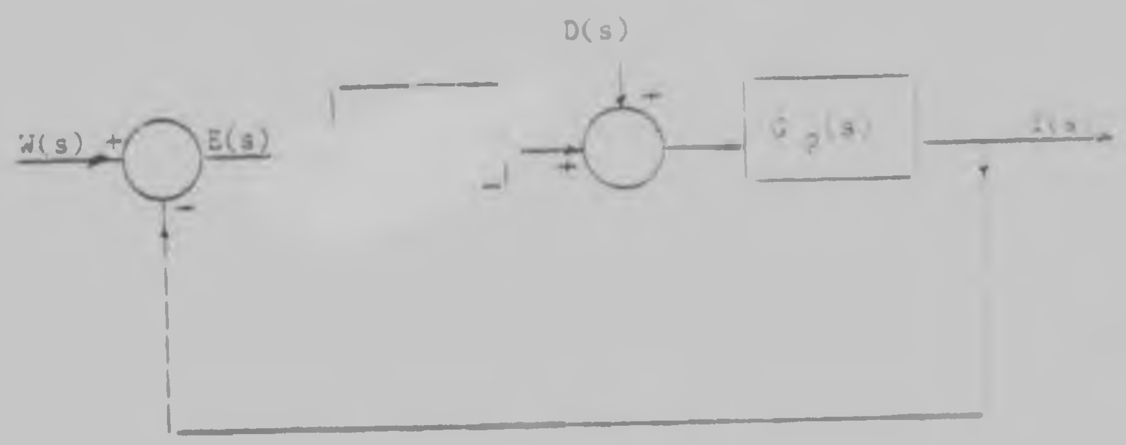
$$\Delta u(t) = u(nt) - u(n(t-1))$$
$$= Ke(t) - K\{c_1 \Delta u(t-1) + c_2 \cdot \Delta u(t-2) + \ldots\} \qquad \text{XVIII}$$

This equation is the Tu Tsing controller in directly usable form for a computer based application. In other words, given the values c_i and K, the controller will ensure that open loop and closed loop dynamics are the same. This has the important result of relieving the process engineer of specifying a desired closed loop response, i.e. one step closer to our ideal 'black box'. At any rate, in most industrial applications exact specifications of such things as damping factor, rise time, number of overshoots, etc. are not necessary. Rather such things as overall system stability and ensuring zero steady state error are more important aspects. Closed loop dynamics equal to open loop dynamics is a sufficient criterion for most applications particularly in the process control field (as compared to the aerospace industry for instance.)

Another important aspect is that the controller is physically realizable, i.e. contains no predictive modes. It has as many poles as zeros, and the closed loop transport delay equals that of the open loop, the minimum required.

Furthermore, process zeros are not cancelled, so zeros in the right hand plane of the root locus plot can not be cancelled by potentially unstable poles. Hence non minimum phase systems are controllable by the algorithm mentioned.

However some questions remain. What about ensuring zero steady state errors? Under what steady state conditions will the controlled variable y(t) and the set point be equal? Consider again Figure 5 with disturbance D(s) added in, i.e.



Figure

FEEDBACK CONTROL SYSTEM

It is well known that for a system in single loop feedback configuration, the following holds:-

1) For zero steady state error with respect to a step change in set point, there must be a forward loop pole at the origin (free integrator), i.e. an integrator in either controller or plant. The disturbance is assumed zero here.

2) For zero steady state error with respect to the application of a step change in disturbance, the controller alone must have at least one free integrator.

For a linear system these two cases are additive due to superposition (As they are for linearized systems. The problem of constant offset will be dealt with later). So in general, the steady state error in response to an input function, disturbance or load, of the form

$$F(t) = Bt^n$$

is zero if the controller has a multiplicity of poles of order m such that $m>n$. Of primary interest here is the steady state error due to step change in set point and (a worst case) step change in disturbance. Therefore the controller needs at least one free integrator to ensure zero steady state error.

It will shortly be shown that the controller does in fact insert a single forward loop integrator. This means inter alia that the controller mentioned solves both the regulator and servo problems, i.e. controls against both set point changes and disturbances. An important factor to note is that the inserted integrator does not detract from the desired closed loop response (as in some cases mentioned earlier) but is an inherent part of the controller.

The following problem now presents itself. When using a self tuning controller, the actual plant dynamics G_p may not be known accurately at some time t. Instead, an estimate denoted by $\hat{G}_p$ is the only available information. This may continue for some time until the plant is sufficiently excited to improve the estimate $\hat{G}_p$. In fact it may take a number of set point changes or disturbance applications to improve the estimate of G_p. Will zero steady state error still be achieved? If the estimate of G_p is given by $\hat{G}_p$, and of K is given by $\hat{K}$, these are 'incorrect' representations but nevertheless provide stable control. The following analysis may be considered with respect to the disturbance input D(z).

$$\text{The error } E(z) = \frac{G_p(z)\, D(z)}{1 + G_p(z)\, G_c(z)}$$

now let G_c be defined in terms of $\hat{G}_p$ and $\hat{K}$, i.e.

$$G_c = \frac{\hat{K}}{1 - \hat{K}\hat{G}_p}$$

$$\text{Therefore } E(z) = \frac{G_p D(z)\{1 - \hat{K}\hat{G}_p\}}{1 - \hat{K}\hat{G}_p + G_p\hat{K}}$$

$$\text{If } D(z) = \frac{1}{z - 1} \quad \text{(step)}$$

then the steady state error ess

$$= \lim_{z \to 1} \frac{z}{(z - 1)} \cdot (z - 1) \frac{G_p\{1-\hat{K}\hat{G}_p\}}{1-\hat{K}\hat{G}_p+G_p\hat{K}}$$

Now since

$$\lim_{z \to 1} \frac{z}{(z-1)} \cdot (z-1)\,\hat{G}_p = \frac{1}{\hat{K}}$$

by definition, (which is the estimated steady state gain of G_p).

The numerator $\{1 - \hat{K}\hat{G}_p\}$ goes to $\{1 - \hat{K} \times \frac{1}{\hat{K}}\}$

and hence ess = 0.

Obviously, if G_c is defined in terms of the actual plant G_p, and not the estimated $\hat{G}_p$, then ess still equals 0.

The controller equation having been chosen, we need an identification scheme to tune it. As always, the identification method must be chosen with the final aim of the identification in mind.

A class of systems i.e. a model is needed first. The derivation of the controller algorithm is helpful here.

From equation XVII, the plant G_p was given as

$$G_p = \frac{1}{K} G_p' = \{(\frac{1}{K} - c_1) z^{-1} + (c_1 - c_2) z^{-2} +$$

$$+ (c_2 - c_3) z^{-3} \ldots\} + d \qquad \text{XIX}$$

Where all variables are defined as previously, but d, a constant offset has been added for generality. The reasons have been given in Section 2.2.1.

Equation XIX contains all the necessary information needed by the controller of equation XVIII.

It is interesting to note that equation XIX is in fact a non-parametric model since $(\frac{1}{K} - c_1)$, $(c_1 - c_2)$ etc. forms an infinite series.

However, as can be seen from Figure 6. the values c_i become negligible for large i and a stable plant. The series may therefore be truncated and equation XIX may be rewritten.

$$G_p = \sum_{i=1}^{N} \beta_i z^{-i} + d \quad \text{where } \beta_i = c_{i-1} - c_i$$

and β_i for $i>N$ are considered negligible.

To avoid confusion from here onward, a discrete parametric model will be considered to be represented by 'parameters' while a discrete non-parametric model will be represented by 'coefficients'.

The simplest method of evaluating the coefficients c_i (or β_i) would, of course, be to apply a step input to the open loop plant and calculate c_i directly from the response. However, this would violate the 'do not disturb' constraint mentioned previously.

A better path to take would be to use equation XIX as the basis of a prediction model which could be applied recursively together with the controller at every sampling instant. This is obviously in similar vein to the methods mentioned in Section 2.2.1, i.e.

1) Predict the controlled variable $\hat{y}(t)$ at some time t using some form of equation XIX.

2) Measure the actual controller variable $y(t)$ and generate a prediction error $e_p(t) = y(t) - \hat{y}(t)$.

3) Update the coefficients by using some form of

$$\hat{\underset{\sim}{\beta}}(t+1) = \hat{\underset{\sim}{\beta}}(t) + \Gamma(t)e_p(t)$$

where $\hat{\underset{\sim}{\beta}}(t)$ = Vector of estimated coefficients $\hat{\beta}_i$

$\Gamma(t)$ = Gain factor yet to be determined.

4) Calculate the controller coefficients from the vector $\hat{\underset{\sim}{\beta}}(t)$.

5) Calculate the manipulated variable using some form of equation XVIII.

6) Return to point 1) at the next sampling interval.

The peculiar aspect of this method is the use of an update procedure (point 3) which is specifically devised for parametic models for a non-parametric formulation.

The discrete nature of the non-parametric model, nevertheless, lends itself to this type of use.

Due to the large number of coefficients necessary, an update algorithm as complex as the recursive least squares is out of the question due to the vast computation involved. However, the stochastic approximation method in Section 2.1 looks appealing.

Point 3) above could then be re-written as

$$\hat{\underset{\sim}{\beta}}(t+1) = \hat{\underset{\sim}{\beta}}(t) + \Gamma\, U(t)e_p(t) \qquad \text{XX}$$

where U(t) = Vector containing a history of the plant input at the sampling instants.

$$U(t) = \begin{bmatrix} u(t-1) \\ u(t-2) \\ \vdots \\ u(t-N) \end{bmatrix} \qquad \& \quad \hat{\underset{\sim}{\beta}}(t) = \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_N \end{bmatrix}$$

In order to ensure that the identification action never dies out, Gamma Γ must be chosen as a constant. The exact value of Gamma must still be decided.

The algorithm mentioned is appealing because it is conceptually simple and easy to implement. Furthermore, the non-parametric model does not require the prior specification of expected plant order, or the presetting of certain parameters to zero to account for process dead time.

From a practical point of view it is worthwhile modifying equation XIX by differencing on both sides. This provides a number of benefit .

The model now becomes:-

$$G_p = \frac{\Delta Y(z)}{\Delta U(z)} = \left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots \qquad \text{(a)}$$

or in the time domain

$$\Delta y(t) = \left(\frac{1}{K} - c_1\right) \Delta u(t-1) + (c_1 - c_2)\Delta u(t-2) \ldots$$

$$+ (c_{N-1} - c_N)\Delta u(t-N)$$

which can be compared with the controller

$$\Delta u(t) = Ke(t) - K\{c_1 \cdot \Delta u(t-1) + c_2 \cdot \Delta u(t-2) \ldots$$

$$+ c_N \cdot \Delta u(t-N)\} \qquad \text{XXI} \quad \text{(b)}$$

The advantages become immediately obvious:-

1) The array $\Delta u(t-i)$ is common to both equations, saving memory space and programming effort in a computer based application.

2) The offset d is cancelled out and need not be estimated.

The result however, is an explicit self tuner as the controller coefficients must be evaluated from the estimated model coefficients $\hat{\beta}_i$ by:-

$$\frac{1}{\hat{K}} = \sum \hat{\beta}_i$$

and $\tilde{c}_i = \tilde{c}_{i-1} - \hat{\beta}_i$

The basic controller algorithm may be manipulated even further. The implementation should allow for the variation of the closed loop response to something other than that of open loop should the process engineer so require it.

The idea is not to destroy the 'black box' constructed in the last few sections. Rather the point is to implement a system which allows a certain amount of adaptation, should this be thought necessary. This can be implemented by including a preplant 'filter' as part of the software as shown below.

The advantages become immediately obvious:-

1) The array $\Delta u(t-i)$ is common to both equations, saving memory space and programming effort in a computer based application.

2) The offset d is cancelled out and need not be estimated.

The result however, i an explicit self tuner as the controller coefficients must be evaluated from the estimated model coefficients $\hat{\beta}_i$ by:-

$$\frac{1}{\hat{K}} = \sum^{N} \hat{\beta}_i$$

and $\hat{c}_i = \hat{a}_{i-1} - \hat{\beta}_i$

The basic contr ler algorithm may be manipulated even further. The implementation should allow for the variation of the closed loop response to something other than that of open loop should the process engineer so require it.

The idea is not to destroy the 'black box' constructed in the last few sections. Rather the point is to implement a system which allows a certain amount of adaptation, should this be thought necessary. This can be implemented by including a preplant 'filter' as part of the sof are as shown below.

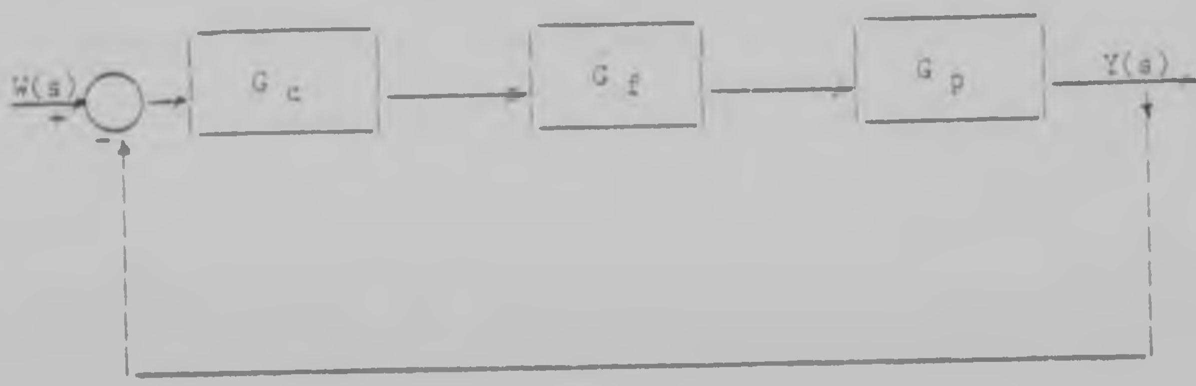


Figure 8

CONTROL LOOP WITH ADDITIONAL FILTER

The filter G_f is physically part of the controller but effectively part of the plant, i.e. since the controller equates open and closed loop dynamics, G_f can be included so as to appear directly in the closed loop response.

$$\text{i.e. } \frac{Y}{W} = \frac{G_c G_f G_p}{1 + G_c G_f G_p} = G_f G_p'$$

The process engineer can prespecify G_f so as to cancel/shift the poles/zeros of G_p to attain a required response. (Zeros in the right hand plane of the pole zero plane may of course not be cancelled).

Should this not be required G_f will default to cancel out, i.e. if G_f is chosen to be a lead-lag compensator

$$G_f(s) = \frac{Tld\ s + 1}{Tlg\ s + 1}$$

Tld, Tlg = filter time constants

which is equivalent to

$$G_f(z) = Fg\ \frac{(R1 - z^{-1})}{(R2 - z^{-1})}$$

where R1, R2, Fg are defined in terms of the time constants of $G_f(s)$

i.e. $R1 = 1 + \frac{Ts}{Tld}$, $R2 = 1 + \frac{Ts}{Tlg}$

$Fg = \frac{Tld}{Tlg}$ ensures unity steady state gain

Ts = sampling period

The process engineer may then do the following. After due consideration of the process at hand, Tld may be chosen to cancel the dominant plant pole and Tlg may then be chosen to attain required dynamics. Various other options are available. However, it must be stressed again that this action detracts from the 'black box' philosophy and must only be used if considered necessary. Otherwise the controller will default to Tld = Tlg, i.e. effectively cancelling the filter action.

Should this not be required G_f will default to cancel out, i.e. if G_f is chosen to be a lead-lag compensator

$$G_f(s) = \frac{Tld\ s + 1}{Tlg\ s + 1} \qquad Tld,\ Tlg = \text{filter time constants}$$

which is equivalent to

$$G_f(z) = Fg\ \frac{(R1 - z^{-1})}{(R2 - z^{-1})}$$

where R1, R2, Fg are defined in terms of the time constants of $G_f(s)$

i.e. $R1 = 1 + \frac{Ts}{Tld}$, $R2 = 1 + \frac{Ts}{Tlg}$

$Fg = \frac{Tld}{Tlg}$ ensures unity steady state gain

Ts = sampling period

The process engineer may then do the following. After due consideration of the process at hand, Tld may be chosen to cancel the dominant plant pole and Tlg may then be chosen to attain required dynamics. Various other options are available. However, it must be stressed again that this action detracts from the 'black box' philosophy and must only be used if considered necessary. Otherwise the controller will default to Tld = Tlg, i.e. effectively cancelling the filter action.

In fact G_f can be directly combined with G_c to form a single equation but it must be remembered that G_f is effectively part of the plant. This final reformatting of the controller equation and the corresponding model equation can be found in the Appendix Section A and will not be repeated here.

The final implementable version is then as follows:-

(The result of the manipulations of Appendix Section A).

At time t, where t coincides with sample interval do the following:-

1) Sample the plant i.e. measure $y(t)$.

Then predict $$\Delta \hat{y}(t) = \frac{1}{R1} \cdot \{\frac{1}{F_g} \cdot [\hat{L}_1(t-1) \cdot \Delta v(t-1) + \hat{L}_2(t-1) \cdot \Delta v(t-2)$$

$$\ldots + \hat{L}_N(t-1) \cdot \Delta v(t-N)] + \Delta y(t-1)\}$$

where R1, R2 and Fg are as described above

$\hat{L}_1(t-1)$, ... $\hat{L}_N(t-1)$ are N coefficients describing the plant dynamics updated at the previous interval.

$\Delta v(t-i)$ is the history of the incremental values of manipulated variable.

$\Delta y(t-1)$ the incremental controlled variable as measured at the (t-1)th sampling interval.

2) Update the coefficients proportional to the prediction error

$$\hat{L}_i(t) = \tilde{L}_i(t-1) + \Gamma \cdot \Delta v(t-i)\{\Delta y(t) - \Delta \hat{y}(t)\} \quad i = 1, N$$

Γ = constant weighting factor

3) Calculate the incremental manipulated variable

$$\Delta v(t) = \frac{\hat{K}}{R2} \cdot \{Q_1(t) \cdot \Delta v(t-1) + Q_2(t) \times \Delta v(t-2) \ldots$$

$$+ Q_N(t) \times \Delta v(t-N)\}$$

$$+ \hat{K}Fg \cdot \{R1 \cdot e(t) - e(t-1)\}$$

and output v(t) to plant actuator.

where $\frac{1}{\hat{K}} = \frac{\sum \tilde{L}_i(t)}{R2-1}$ is the estimated plant steady state gain

and $Q_i(t)$ are the controller coefficients defined by

$$Q_1(t) = \frac{1 - R2}{\hat{K}} + \hat{L}_1(t)$$

and $Q_i(t) = \hat{L}_i(t) + Q_{i-1}(t) \quad i = 1 \ldots N$

$e(t)$ = set point error = $w(t) - y(t)$

4) At the next sampling interval $(t + 1)$, repeat steps 1 to 3.

The above equations will be collectively known as equation XXIII, points 1) to 4) above.

In order to start the algorithm, the following variables must be chosen:

1) The coefficients $\hat{L}_i$ for $i = 1$ to N.

2) N, the number of coefficients to be used.

3) Sampling time.

4) The weighting function Γ.

5) R1 and R2, the filter parameters, if deemed necessary else they default to R1 = R2.

Both the simulation and the implementation sections will discuss these choices further, however, two can be tackled immediately.

When choosing the sampling period, it is common to choose, as a rule of thumb, a sampling rate ten times faster than the fastest mode in the system, i.e. if the fastest mode is given by

$$\frac{fa}{s + fa}$$

The sampling frequency may be chosen as $fs = 10fa$ to satisfy Nyquist criterion.

Furthermore, the number of coefficients Li used, represented by N must span the time response of the plant (e.g. to a step input).

Since, in four time constants Tc, a plant has reached 1,8% of its final value after a step change in input, we may use this as a criterion, i.e.

$$Ts \text{ (seconds per sample)} = \frac{1}{fs}$$

can be chosen such that N x Ts = 4 Tc

Therefore if the fastest mode of the open loop plant Ta = 1/fa is approximately known, the sampling period can be chosen by

$$Ts = \frac{Ta}{10}$$

Then if the time constant Tc of the open loop plant is approximately known, choose

$$N = \frac{4\ Tc}{Ts} = \frac{4\ Tc \times 10}{Ta}$$

Note that if the plant is first order dominant and other modes are neglected, Ta = Tc, and we have the remarakable result that

$$\underline{N = 40}$$

The noteworthy characteristics of the self tuner may be summarized:-

1) The controller equates open and closed loop dynamics. This precludes the necessity of specifying a different desired response for different plants.

2) The model used is non-parametric. Hence the problems associated with parametric models do not arise (i.e. need to specify number of parameters and maximum expected dead time).

3) The self tuner should provide stable control for non-minimum phase systems.

4) Dead time is handled automatically.

5) Time varying and non-linear plant behaviour is handled automatically.

6) Set point (reference) tracking and regulation against noise are included.

7) The control criterion should not cause excessive control action.

8) Zero steady state error is achieved at all times even when the estimated coefficients of the model are not the 'correct' ones.

But also,

9) Open loop unstable plants are not controllable by this method.

10) The algorithm needs more memory space in a computer implementation than those based on parametric models.

When using the controller just described (or any adaptive controller) the resultant closed loop system is usually time varying and non-linear. Analysis of the behaviour of the self tuner is therefore far from trivial.

The main areas of interest are:-

I Overall stability of the system.

II Convergence of the model coefficients.

III The properties of the resulting controller.

To expand:-

I Overall stability is obviously of prime importance. Without it, the controller is useless. Of particular interest are the initial choices of the following to system stability:-

a) The coefficients $\hat{L}i(o)$

b) N, the number of coefficients to be used.

c) The sampling period Ts.

d) The weighting function.

By looking at the controller equation of XXIII, some intuitively dangerous pitfalls may be avoided:-

i) Since the manipulated variable is given by

$$\Delta v(t) = \frac{\hat{K}}{R2} \sum Qi(t) \cdot \Delta v(t-i) + \hat{K} \cdot F_g \{R1\ e(t) - e(t-1)\}$$

negative $\hat{K}$ would drive the system away from the set point.

Since $\hat{K} = \dfrac{R2 - 1}{\sum \hat{L}i}$

then $\hat{L}i(o)$, i = 1, N must be chosen at startup to ensure $\hat{K} > 0$

(R2 is always > 1).

ii) 'Very small' values of $\hat{L}i(o)$ may lead to a large controller forward gain $\hat{K}$. The controller would then be very sensitive to even small set point errors. Though the resulting plant activity would improve the estimates of $\hat{L}i$ very quickly, process variables may saturate first.

iii) The weighting constant Γ is important. The update equation is

$$\hat{L}i(t) = \hat{L}i(t-1) + \Gamma \Delta v(t-i) \{e_p(t)\}$$

If Γ is 'too large' over compensation may occur, i.e. either

a) Negative Over Compensation -

Causing small or negative values of $\hat{L}i$. $\hat{K}$ would become large or negative resulting in instability.

b) Positive Over Compensation -

Causing large values of $\hat{L}i$. $\hat{K}$ would become very small and the control action would switch off.

iv) 'Too large' a sampling period may cause instability, while too small a period may cause excessive control action.

II A host of questions arise as to the convergence of the model coefficients:-

i) Under what conditions will $\hat{L}i(t)$ converge from those values given at startup to some final value?

ii) If they converge, how fast will they converge?

iii) If the plant parameters vary with time, will the self tuner be able to 'keep up'?

iv) Will the coefficients converge to the same values (for the same plant) no matter what initial conditions (e.g. $\hat{L}i(o)$ and Γ at $t = 0$)?

III If the coefficients converge will the resulting controller be the required one?

Obviously the ideal situation is to find some analytical solution to the abovementioned problems. However, due to the difficulties involved in such a task, a second best approach must be considered, i.e. computer simulation.

4. SIMULATION STUDY

The objectives of the simulation carried out were:-

i) To ascertain whether the self tuning algorithm (equation XXIII) is at all practical to implement in the light of the problems mentioned in the previous chapter.

ii) To gain an intuitive 'feel' of the self tuner characteristics in order to ease implementation.

A number of simulations were run using ACSL (Automatic Continuous Simulation Language) on the IBM 370 mainframe at the University of the Witwatersrand. ACSL is a Fortran like, high level language, specifically designed for simulation in the control field and other related subjects. Transfer functions and time varying signals are easily implemented in a single line of code.

The plant to be controlled was chosen to represent a gas cleaning plant on a submerged arc-furnace, the intended target plant for the self tuner.

$$G_p = \frac{1,7}{(100S+1)\ (15S+1)\ (3S+1)}$$

The filter $G_f = \frac{100S+1}{15S+1}$ was chosen

to give a stable second order dominant

$$G_f G_p = \frac{1,7}{(15S+1)^2\ (3S+1)}$$

Unobservable noise was added to the manipulated variable prior to application to the plant. The noise was simulated using the Ornstein Uhlenbeck, zero mean, band limited noise generator provided by ACSL. The sampling rate was chosen to be at one second intervals to satisfy the Nyquist Criterion.

A number of simulations were run with differing values of, initial coefficients $\hat{L}i(o)$, weighting constant Gamma and number of coefficients N.

The most noteworthy characteristic overall appears to be the insensitivity of system stability to variations in the above. Although the choices were made with due consideration of the facts mentioned in point I in the previous chapter . All in all, it appears that, given stable initial conditions, the coefficients appear to converge and the system is then stable at all times.

As an example, consider the results of the following run. To cover the time response of the system, N, the number of coefficients used, was chosen to be 80.

Gamma was set constant at 0,01 throughout the run. Initial coefficients $\bar{L}i(o)$ were chosen = 0,001 (all eighty). This deserves some discussion. Prespecifying 80 coefficients so that the model in fact approximates the plant $G_f G_p$ in some way is both arduous and self defeating in this context. Specifying all initial coefficients equal, is a much easier task and provides a reference for comparison with the converged coefficients.

The set point was fixed at unity arbitrary units (throughout) while at zero time the plant was set at a steady state value of 1,7 units. The run lasts for 3 000 seconds.

The variation, with time of the tenth coefficient $L_{10}(t)$ is shown in Figure 9.

We can note the following

i) The coefficient converge t to a constant value, but with variation about a mean.

ii) Convergence is fast, ained in a mean sense within 300 seconds.

The controlled variable Y(t) is shown in Figure 10(a) over the same time span.

i) Stability is maintained at all times.

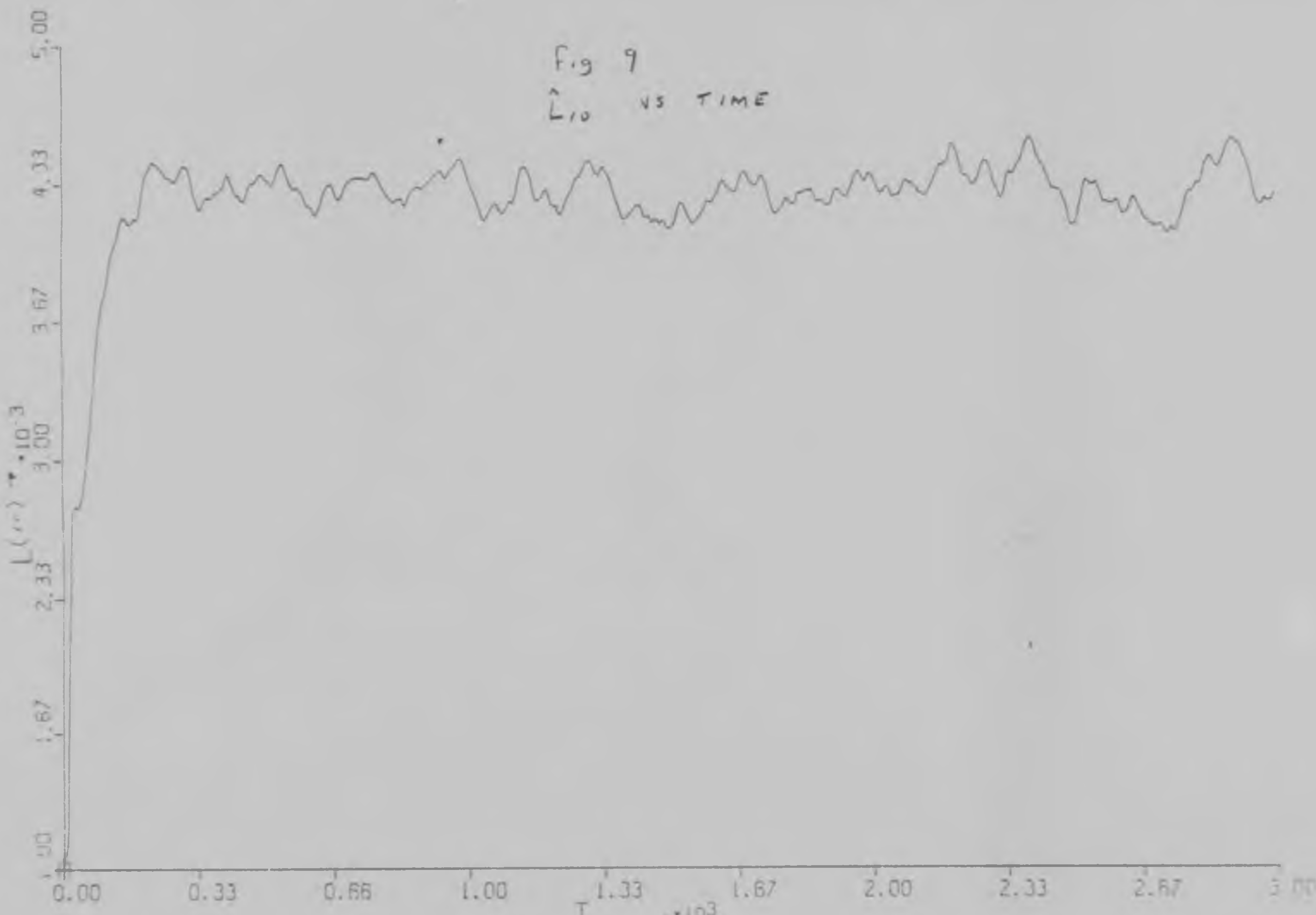
ii) Initial control is erratic as would be expected while the initial coefficients $\bar{L}i(o)$ are "incorrect".

iii) The controlled variable converges to the set point.

As a comparison, an identical re-run (even the noise is repeatable) but with Gamma = 0,001 gave the results in Figure 11.

Fig 9

$\hat{L}_{10}$ vs TIME



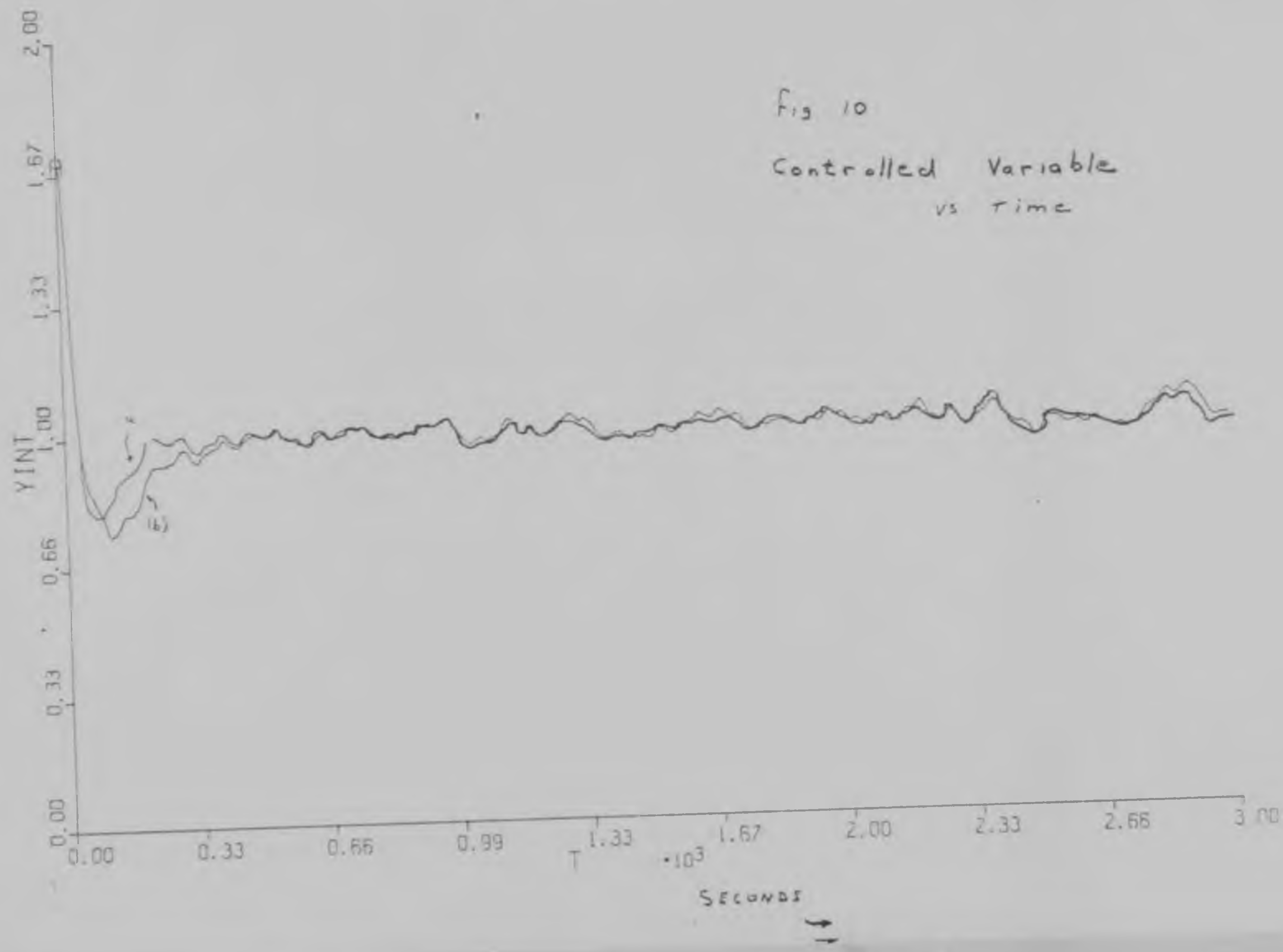


Fig 10
Controlled Variable
vs Time
YINT
2.00
1.67
1.33
1.00
0.66
0.33
0.00
(b)
0.00
0.33
0.66
0.99
1.33
1.67
2.00
2.33
2.66
3.00
T
$\cdot 10^3$
SECONDS

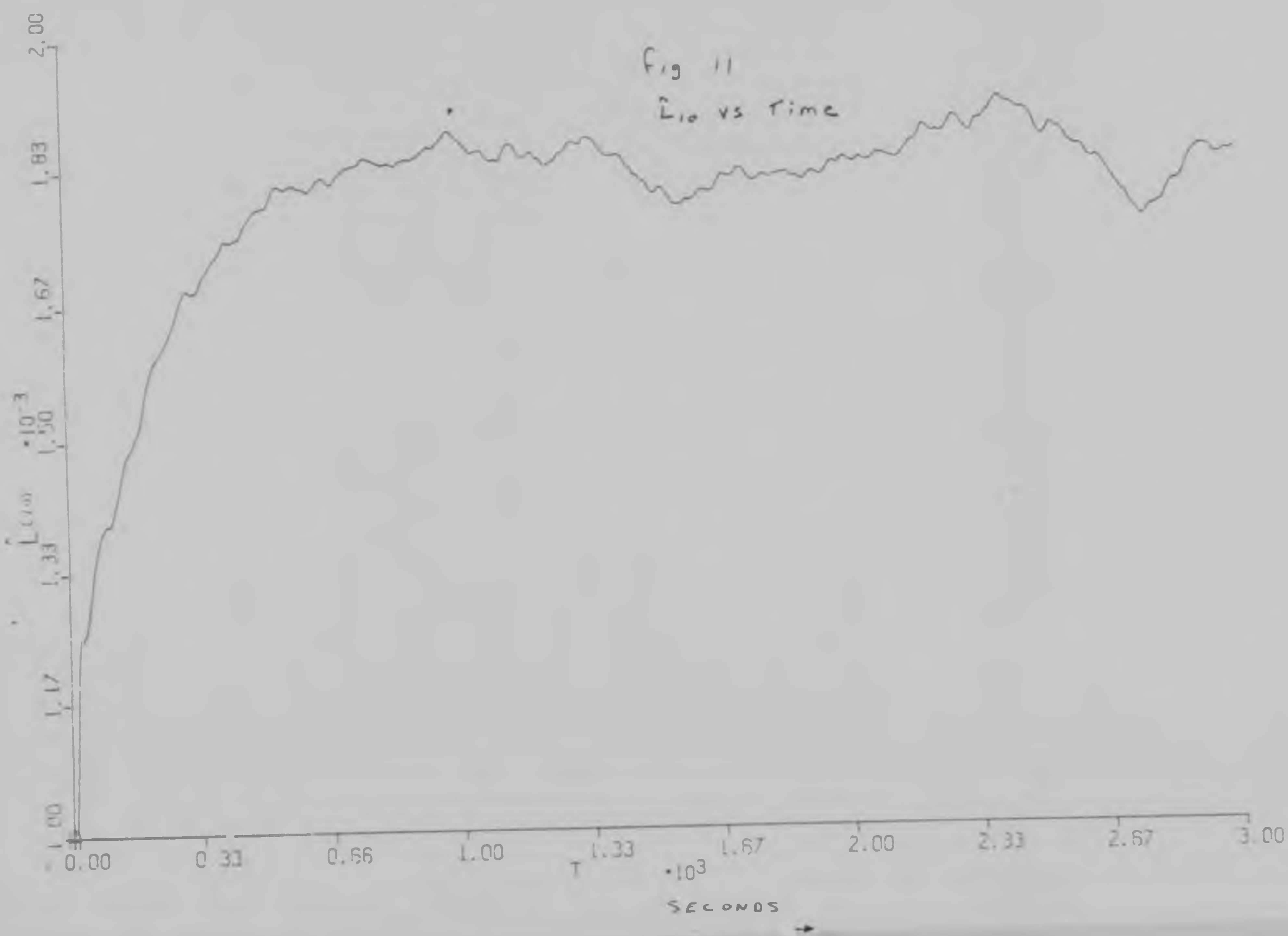


Fig 11
$\hat{L}_{10}$ vs Time

Where for $\hat{L}_{10}(t)$

i) Convergence is slower than for Gamma = 0,01

ii) There is less variation about the mean.

iii) The coefficients converge to different values than for the run Gamma = 0,01.

Points i) & ii) are expected as a lower Gamma reduces the effect of the prediction error, while simultaneously reducing the effect of the unpredictable noise on any new estimate of $\hat{L}i$. Point iii) poses a problem, and will be discussed shortly.

In Figure 10(b) the controlled variable is shown for this run.

i) As expected, due to slower convergence of the coefficients, the initial control is worse than for Gamma = 0,01.

ii) Once coefficients have converged, the associated regulation properties appear identical to case (a).

Now from the simulation it appears that the values to which the coefficients converge are dependant on:-

a) Initial state of the plant.

b) Initial coefficients $\hat{L}i(o)$.

c) Driving noise.

d) Gamma - the estimator weighting constant.

Change just one of these, and the coefficients appear to converge somewhere else. This is most distressing as the original Tu Tsing controller is evaluated via the impulse reponse (which is unique) from the step response of a plant.

Many attempts at rectifying this situation proved fruitless. It was noted, however, that the coefficients $\bar{L}i$ as calculated from the actual step response, if used as the initial condition, did not converge elsewhere.

Aström et al (Ref 16) presets a single parameter in order to ensure identifiability. In this case, presetting up to three coefficients to fixed values had no affect on the final (converged) coefficient estimates. This was possibly due to the large amount of coefficients involved. Presetting any more than this defeats the object of the exercise.

The best that can be done in this situation is, to quote Astrom (Ref 16) on the non uniqueness of parameters:-

> 'We must, however, remark that in the present context we do not bother very much about the behaviour of the parameter extimates. They are only used as an intermediary step to compute the controller parameters, and our main concern is the convergence of the regulator'.

However we do need to measure the achievement of the self tuner in some manner. As with other identification methods, the best test is that which has the ultimate aim of the identification in mind. In this case, open and closed loop dynamics should be identical upon convergence of the coefficients $\tilde{L}i(t)$.

Figure 12(a) shows the open loop step response of

$$G_f G_p = \frac{1,7}{(15S+1)^2 \ (3S+1)}$$

to a unit step input over 200 seconds of real time. The result has been normalised to unity. Superimposed on this (b) is the closed loop response of the self tuned system to a unit step change in set point. For this run the additive noise has been removed, the estimator bypassed, i.e. the controller coefficients fixed for the duration of the run at the values arrived at during the first simulation run mentioned, (i.e. Gamma = 0,01). These figures are as good as identical. Not all runs produced such excellent results. Using converged values of Li from the second run mentioned (Gamma = 0,001), the closed loop, unit step change (in set point) response is shown in Figure 13. Compared to the open loop step response Figure 12(a), is identical for the first 25 seconds, after that, a distinct 'dent' is noted which affects the rest of the response.

SECONDS →

However we do need to measure the achievement of the self tuner in some manner. As with other identification methods, the best test is that which has the ultimate aim of the identification in mind. In this case, open and closed loop dynamics should be identical upon convergence of the coefficients $\bar{L}i(t)$.

Figure 12(a) shows the open loop step response of

$$G_f G_p = \frac{1,7}{(15S+1)^2 \ (3S+1)}$$

to a unit step input over 200 seconds of real time. The result has been normalised to unity. Superimposed on this (b) is the closed loop response of the self tuned system to a unit step change in set point. For this run the additive noise has been removed, the esti[illegible]tor bypassed, i.e. the controller coefficients fixed for the dura[illegible] of the run at the values arrived at during the first simulat[illegible] run mentioned, (i.e. Gamma = 0,01). These figures [illegible] identical. Not all runs produced such excellen[illegible] r[illegible] converged values of Li from the second run mentio[illegible]. (Gam[illegible] = 0,001), the closed loop, unit step change (in set point) response is shown in Figure 13. Compared to the open loop step response Figure 12(a), is identical for the first 25 seconds, after that, a distinct 'dent' is noted which affects the rest of the response.

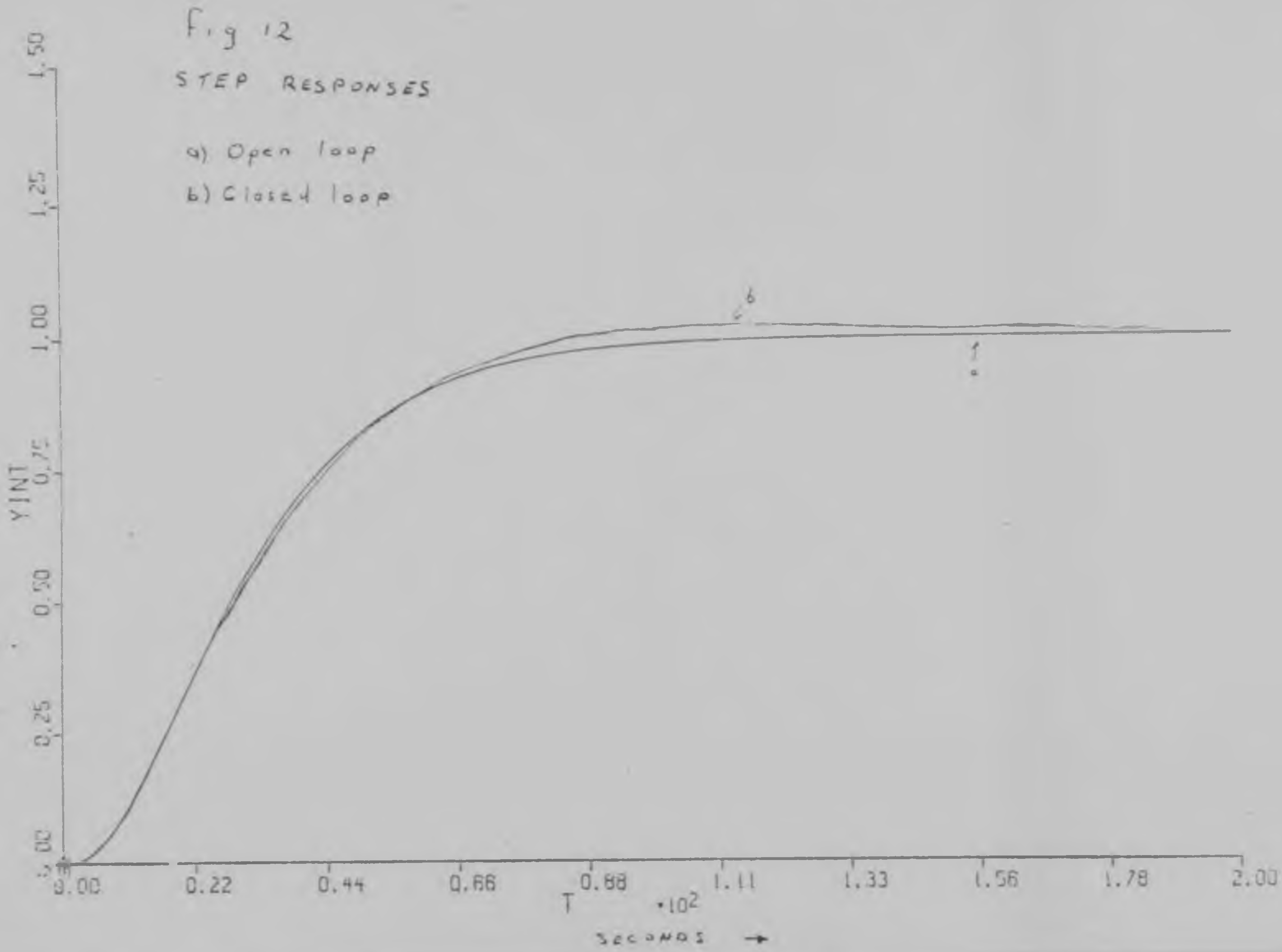


Fig 12
STEP RESPONSES
a) Open loop
b) Closed loop
YINT
0.00
0.25
0.50
0.75
1.00
1.25
1.50
0.00
0.22
0.44
0.66
0.88
1.11
1.33
1.56
1.78
2.00
T
*10^2
SECONDS →
b
a

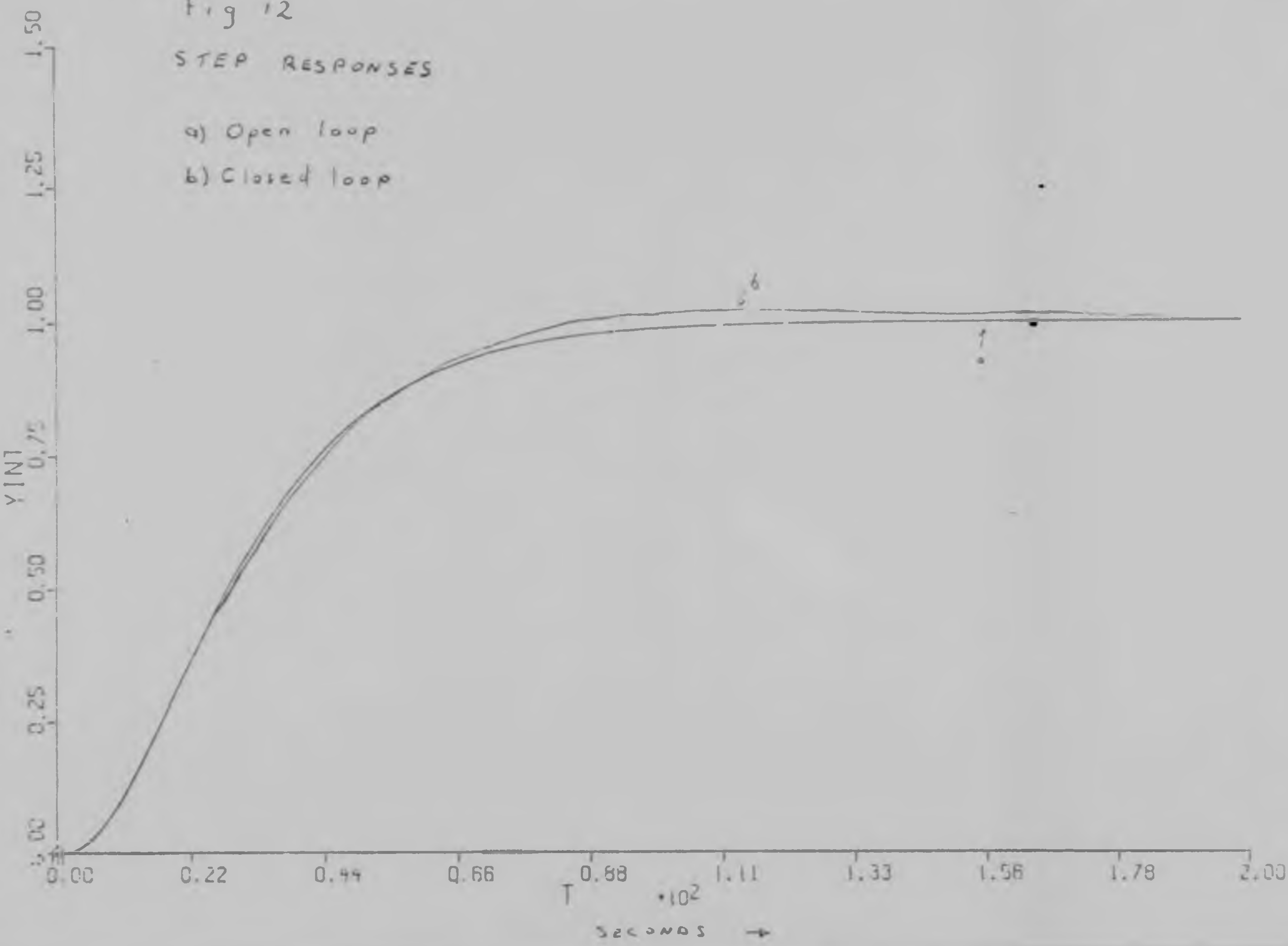


Fig 12

STEP RESPONSES

a) Open loop

b) Closed loop

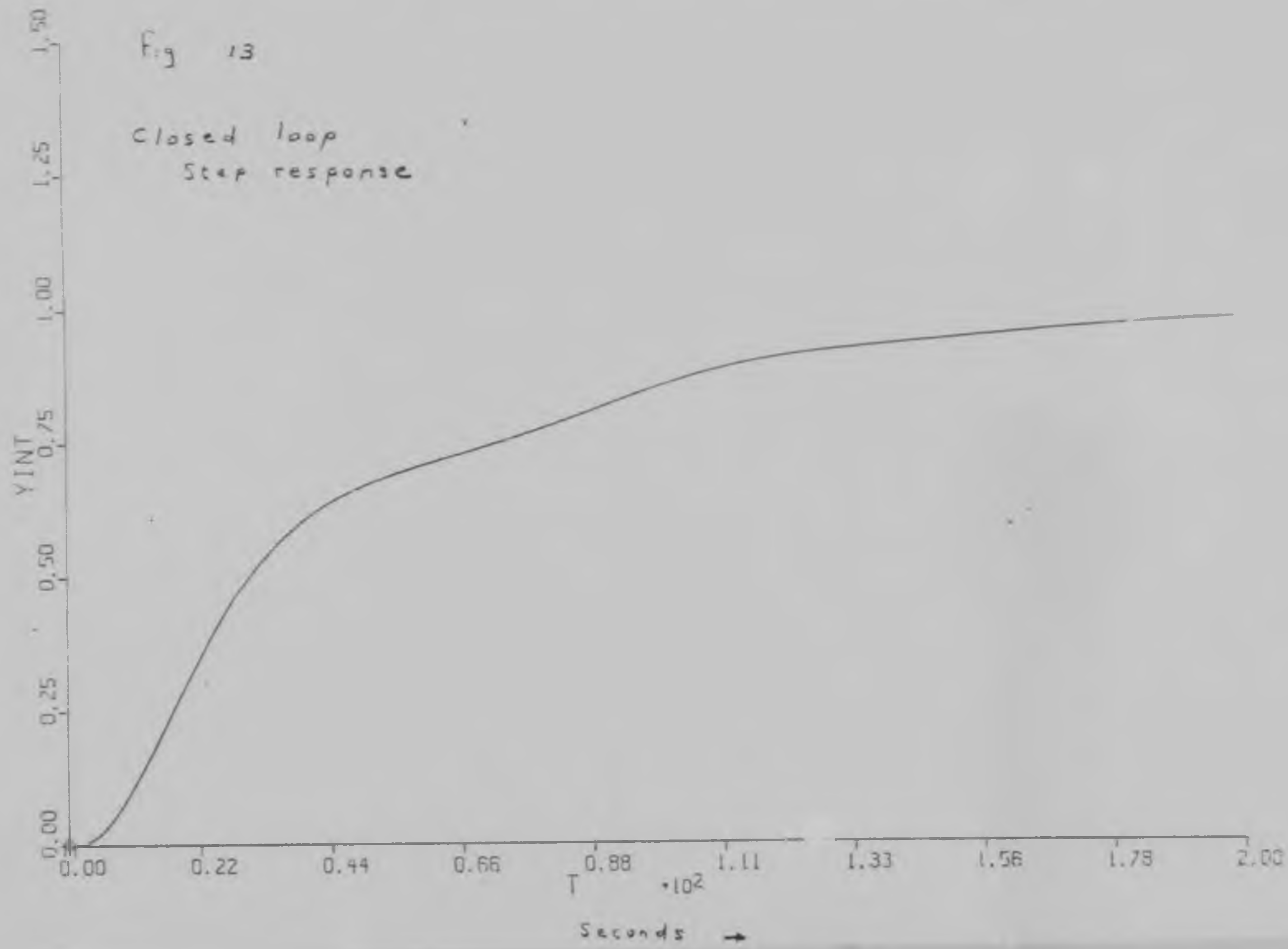


Fig 13
Closed loop
Step response
YINT
0.00
0.25
0.50
0.75
1.00
1.25
1.50
0.00
0.22
0.44
0.66
0.88
1.11
1.33
1.56
1.79
2.00
T *10²
Seconds →

Nevertheless, the closed loop response is still quite similar to the open loop response. One reason for the apparent discrepancy could be the affect of the unpredictable noise in the estimates coefficients, i.e. the coefficients converge in a mean sense about a constant value, the variation about the mean depends on the driving noise, and Gamma the weighting constant. Furthermore the system excitation may not be sufficient to identify all the system modes.

In general the coefficients converged such that if $\hat{L}i$ were plotted against i for a number of runs, the results are as shown overleaf.

The following can be noted:-

i) All runs resulted in the typical bell shaped curve (similar to the impulse response).

ii) The peaks in the bell shape occur at approximately the same value of i for any run regardless.

iii) The estimated plant steady state gain $1/\hat{K}$ was consistent in all cases (regardless of other conditions) within 4% to the actual plant gain.

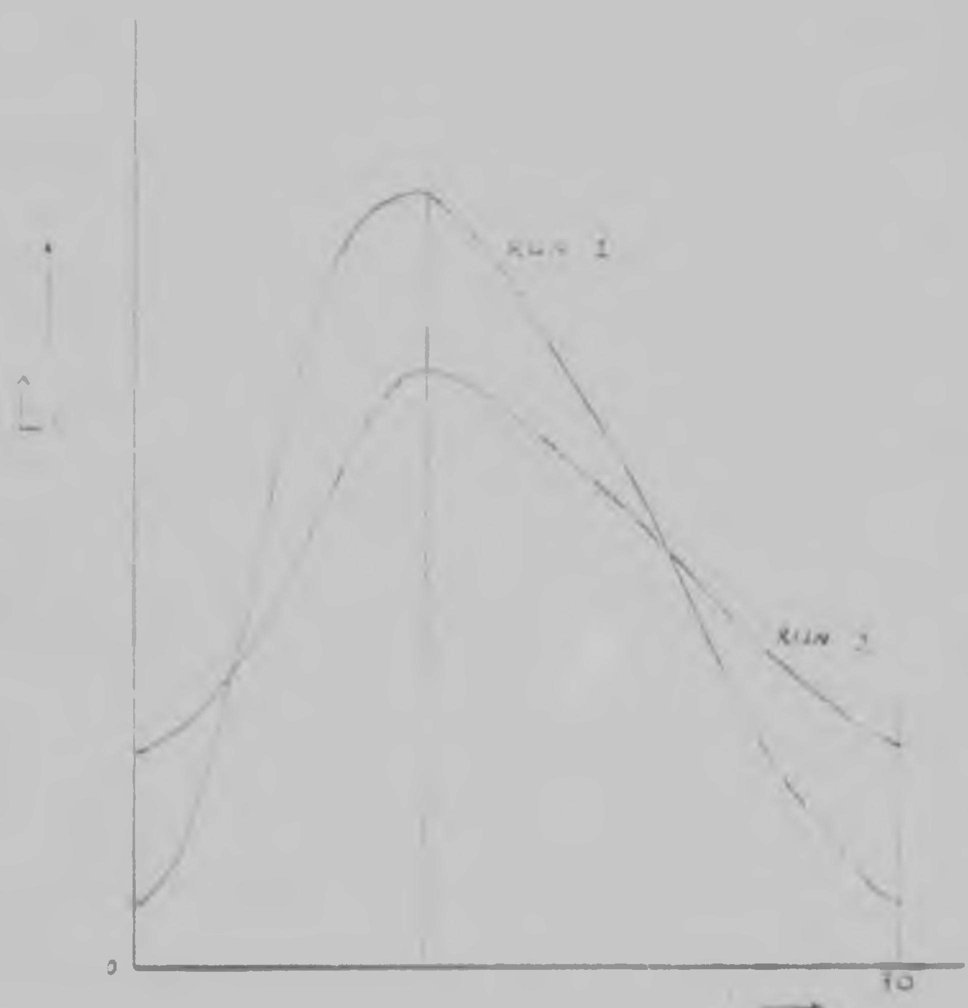


Figure [illegible]

CONVERGED COEFFICIENTS [illegible]

As a further example, another run is shown in Figures 15 and 16. Here initial conditions are identical to one mentioned previously (Gamma = 0,01) but at 1 000 seconds, the plant gain was changed from 1,7 to 3,· instantaneously, simulating a time varying plant. The effect on Li(t) can be seen in Figure 15 and on the controlled variable y(t) in Figure 16. The system is thrown 'off balance' for a while but soon recovers to normal operation.

To sum up the simulation results:-

i) The algorithm for the self tuner appears to be very robust. Stability is achievable under a wide range of initial conditions

ii) The coefficients appear to converge whenever there is system stability.

iii) If the coefficients converge, the resulting controller is a good approximation to the required controller.

However, it must be stressed here that the above sort of simulation can never give global results of stability and convergence properties. Nevertheless, to the question, 'Is the self tuning algorithm worthwhile implementing?', the answer must be .nqualified 'YES'.

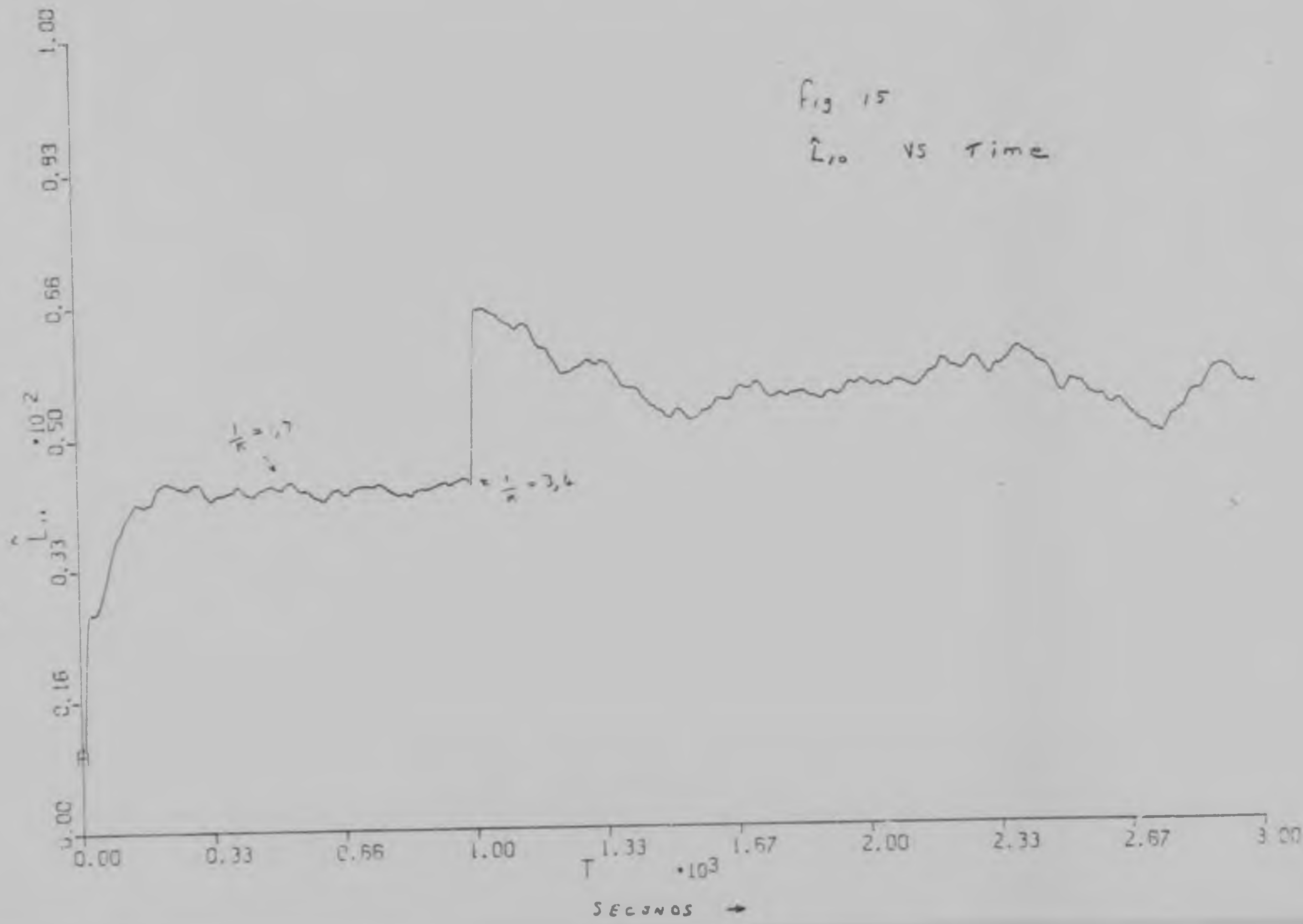


Fig 15
$\hat{L}_{10}$ VS Time
$\frac{1}{R} = 1.7$
$\frac{1}{R} = 3.6$
1.00
0.83
0.66
0.50
0.33
0.16
0.00
$\cdot 10^{-2}$
0.00
0.33
0.66
1.00
1.33
1.67
2.00
2.33
2.67
3.00
T
$\cdot 10^{3}$
SECONDS

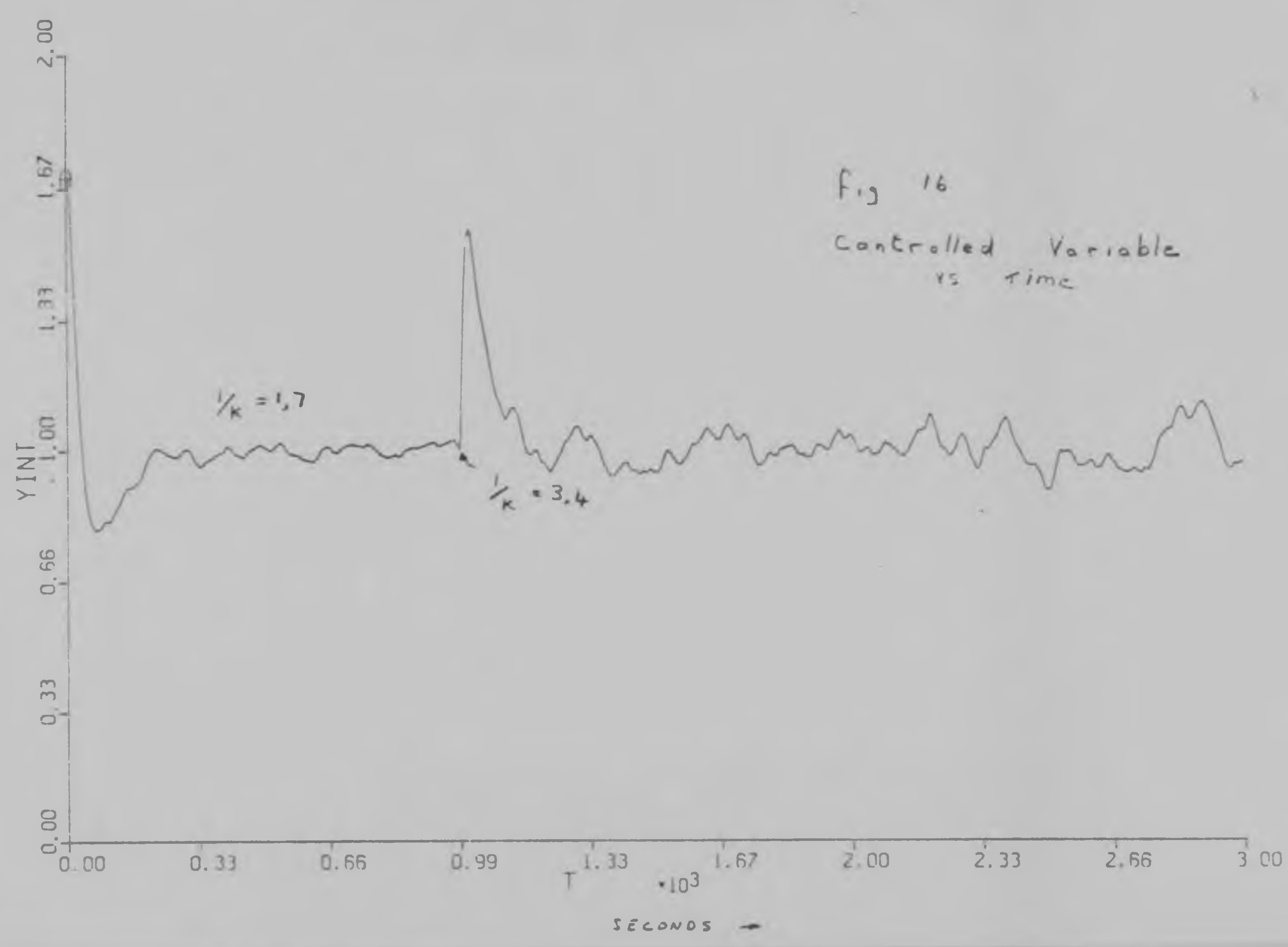


Fig 16
Controlled Variable
vs time
1/k = 1,7
1/k = 3.4
YINT
2.00
1.67
1.33
1.00
0.66
0.33
0.00
0.00
0.33
0.66
0.99
1.33
1.67
2.00
2.33
2.66
3.00
T
*10^3
SECONDS

5. SELF TUNER IMPLEMENTATION - THE NUTS AND BOLTS

5.1 Practical Requirements

A self tuner of general industrial applicability must appear, to both operator and plant, to be a standard PID type controller minus a manual tuning ability. To this end the controller must have the following functions:-

i) Automatic operation (self tuning).

ii) Manual operation (allowing operator intervention) and an added feature, here termed;

iii) Interrogation operation.

The latter entails the ability to review and manipulate all variables pertinent to the operation of the self tuner at any time. This is a necessary feature in a research project of this nature.

To achieve the above, hardwa e and software were developed and are described in the following sections.

5.2 Hardware

Considering the arithmetic manipulations, it is a foregone conclusion that the self tuner must be computer based. learly, a microprocessor is ideal for this type or application.

The heart of the system was chosen to be the Intel single board computer based on the 8080A microprocessor. The board, called the SBC 80-10 by Intel was a natural choice due to its availability. Furthermore, both the University of the Witwatersrand Electrical Engineering Department, and the National Institute for Metallurgy have extensive hardware and software development aids for Intel microprocessors. These take the form of emulation stations and PLM 80 compilers respectively. From a user's point of view, the controller is as shown in the figure below.

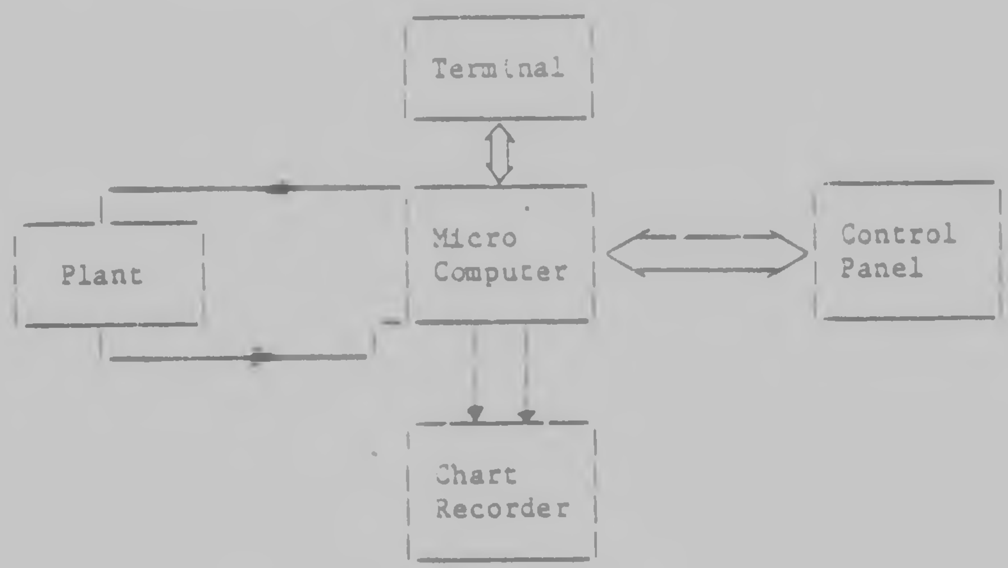


Figure 17

CONTROLLER SCHEMATIC

where:-

i) The control panel allows the operator to specify different modes of operation.

ii) The terminal enables controller variables to be examined and modified.

iii) The chart recorder enables the recording of process and controller variables for post operative analysis.

A more detailed view is shown in Figure 19.

The central unit contains four boards:-

i) SBC 80-10 Microcomputer which includes

1 x 8080, CPU
1Kb Random Access Memory (RAM)
4Kb EPROM
1 x SERIAL I/O Interface
48 x Parallel I/O Lines

ii) SBC 104 Memory and I/O Extension Board which includes

4Kb RAM
4Kb EPROM
1 x SERIAL I/O Interface
48 x Parallel I/O Lines

iii) RTI 1200 Board which contains

2 x 12 bit Digital to Analogue Converters with associated current outputs (4-20 mA)

1 x 12 bit Analogue to Digital Converter (32 singled ended, multiplexed input Channels)

1 x Real Time Clock used as an interrupt

iv) 1x Custom wire wrapped board which buffers, filters and amplifies the controlled variable signal. An extra voltage to current converter is also available on this board.

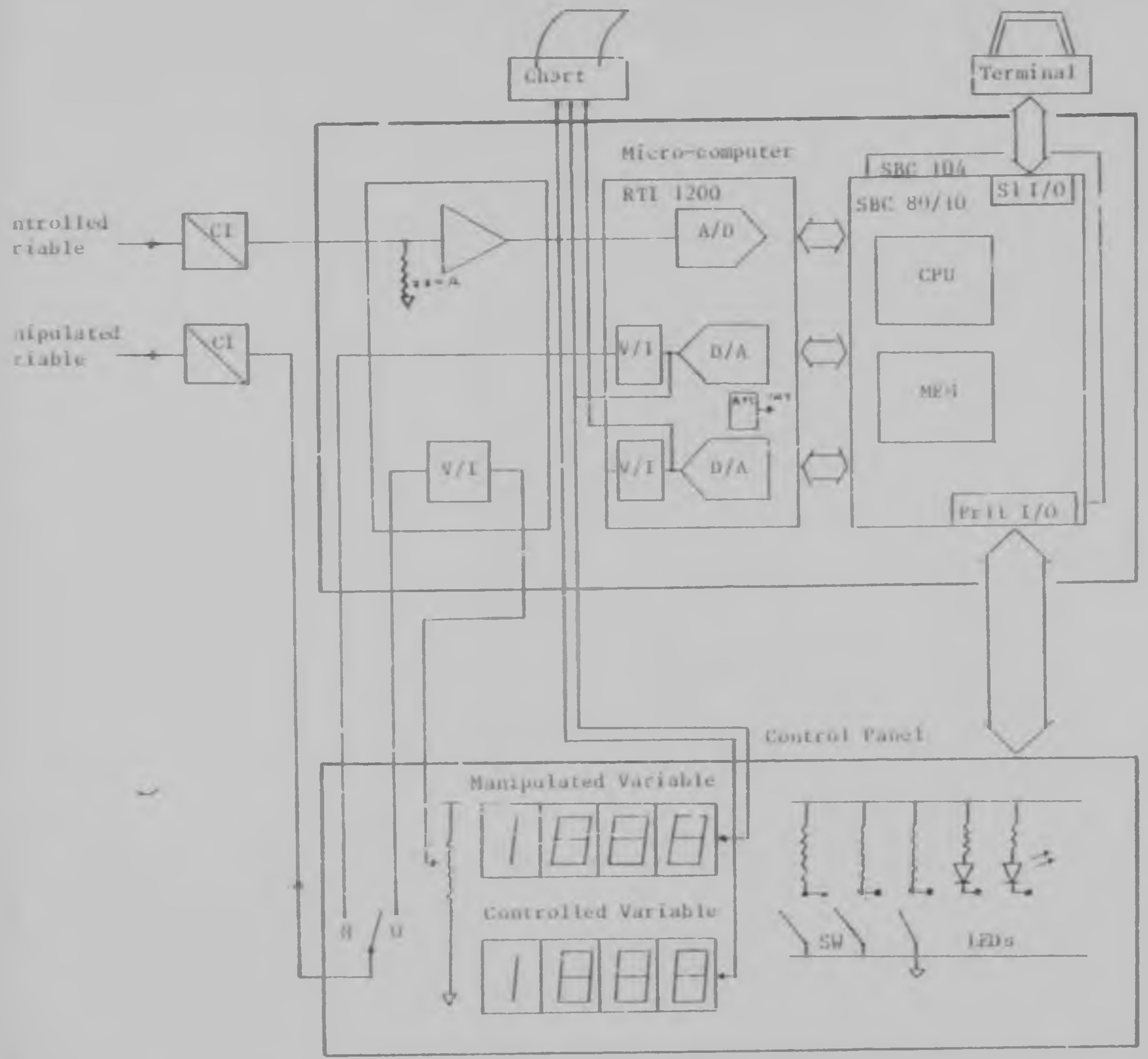
The operator deals mainly with the control panel, a picture of which can be seen in Figure 18.

There are two LED displays marked CV and MV. These display the controlled and manipulated variables respectively in percentage values. The thumbwheel switch enables set point alteraton. There are also a number of switches present. These fulfil the following roles. Three single pole changeover switches are used as hardware flags via an input port and are as marked.

i) MONIT - In the up position, this halts control action. Program flow is directed to the SBC 80-10 monitor program to allow all variables located in memory to be examined and altered if desired.

Figure 18

CONTROL PANEL



N - Normal
O - Override
RTC - Real Time Clock
INT - Interrupt
CI - Current Isolation
SW - Hardware Flags
& Thumbwheel Switch

Figure 19

CONTROLLER CONFIGURATION

ii) MANUAL/AUTO - (Top left). This enables control action. Either in the auto-mode (Down) or manual mode (Up). If manual mode is requested, the single pole changeover switch (biased to centre off) directly beneath (called the ramp flag) enables the operator to vary the manipulated variable via the microcomputer.

iii) PRT - This switch enables or disables the writing of preselected data to the terminal during run time.

The final switch

iv) OVRD - This allows a complete operator override, i.e. the micro is bypassed and the operator has complete control over the manipulated variable.

An array of seven LED's are also present on the control panel. Six of these are used for a qualitative display of the prediction error during run time. The extra LED is used as an alarm indicator.

Additional controller features include

* Galvanic isolation for plant input and output.

* The use of the second D/A channel to display any preselected variable on a Chart Recorder.

[illegible]

[illegible] possible, the [illegible] was written in PLM80, a high level language developed by Intel, the designers of the 8080 microprocessor. The structure of this language is such that programs are written in modular form. Each module performs a specific task which can be tested independently of other modules. A single Assembler [illegible] was also found to be necessary.

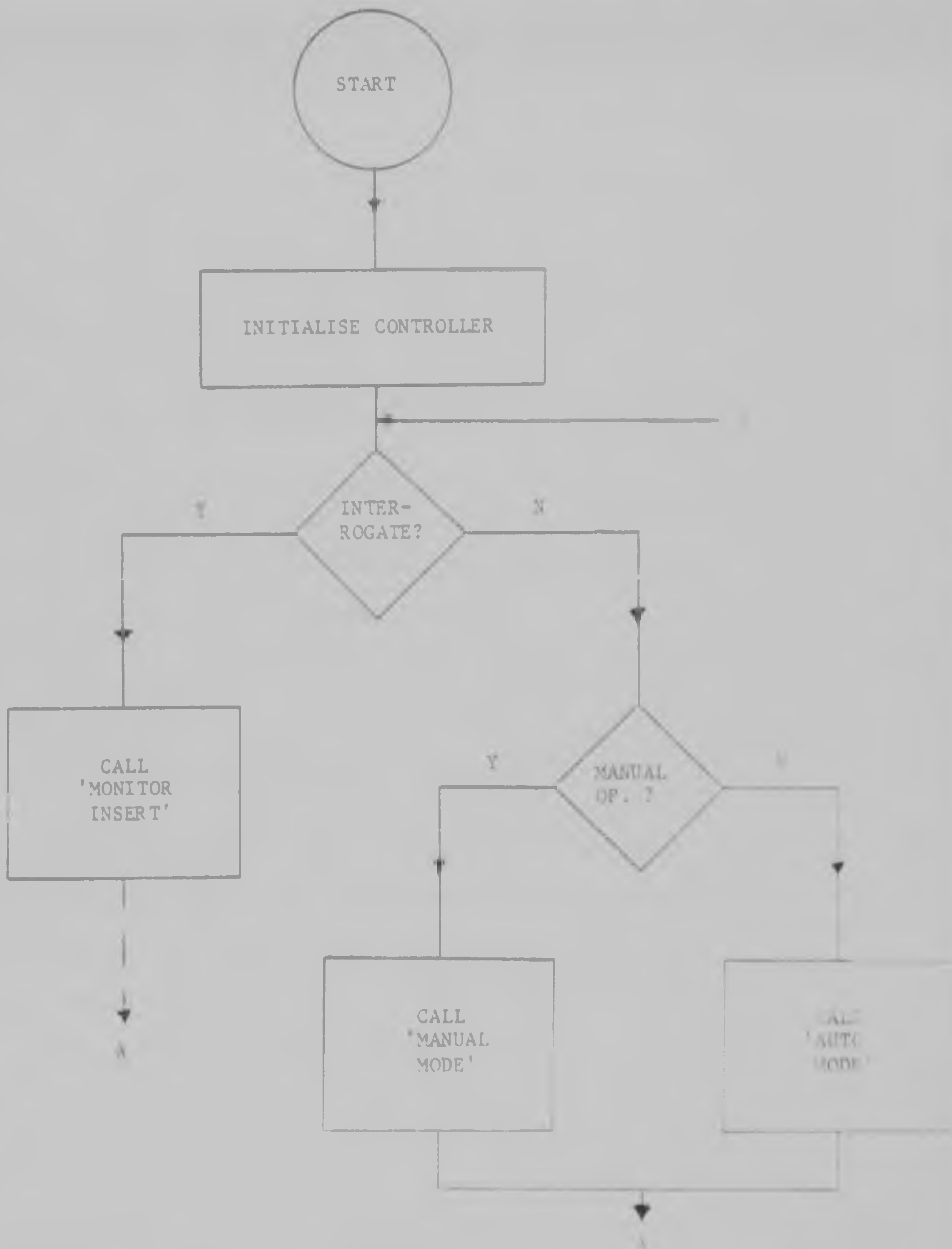
This section describes the software modules comprising the controller. [illegible] the [illegible] to memory are being investigated, the controller [illegible], controlling [illegible] automatic (self tune) mode, or [illegible] mode.

[illegible] The [illegible] Module

[illegible] this module is [illegible] first. Program [illegible] in the following figure. This module is essentially in [illegible].

Figure

Flow Chart - [illegible]uner [illegible]



There are two major sections:-

i) An initialization procedure which initializes all hardware devices and software parameters before self tuning actually begins. Here ports are defined as input or output, variables are preset, etc.

ii) A continual 'do-loop' which checks the operator's requirements via the hardware generated flags on the control panel. The result is one of three operations.

a) Full Automatic Control

b) Manual Control

c) Interrogation Mode

In cases a) and b) program flow returns to the self tuner control module when the associated function has been executed. In case c) program flow is at operator discretion.

2. <u>Monitor Insert Module</u>

Should the operator request interrogation mode, the 'self tuner control module' discussed above directs program flow to this routine, which is the only routine written in Assembler. This module eases transfer to the SBC 80P monitor program which enables the operator to inspect memory, registers etc. via the RS232 link. The self tuner control module is the only place where transfer to interrogation mode can occur, (see previous Flow Chart).

The interrogation request has precedence over the auto or manual modes. Upon entering the monitor program all control processing halts. However, the manual override (OVRD) function allows the operator control over the plant in bypassing the microcomputer.

3. Automatic Mode Module

This module's function is to control program flow during automatic mode. A flow chart is shown in the figure overleaf. This routine is called only by the 'self tuner control module'.

Flow Chart - Automatic Mode Module

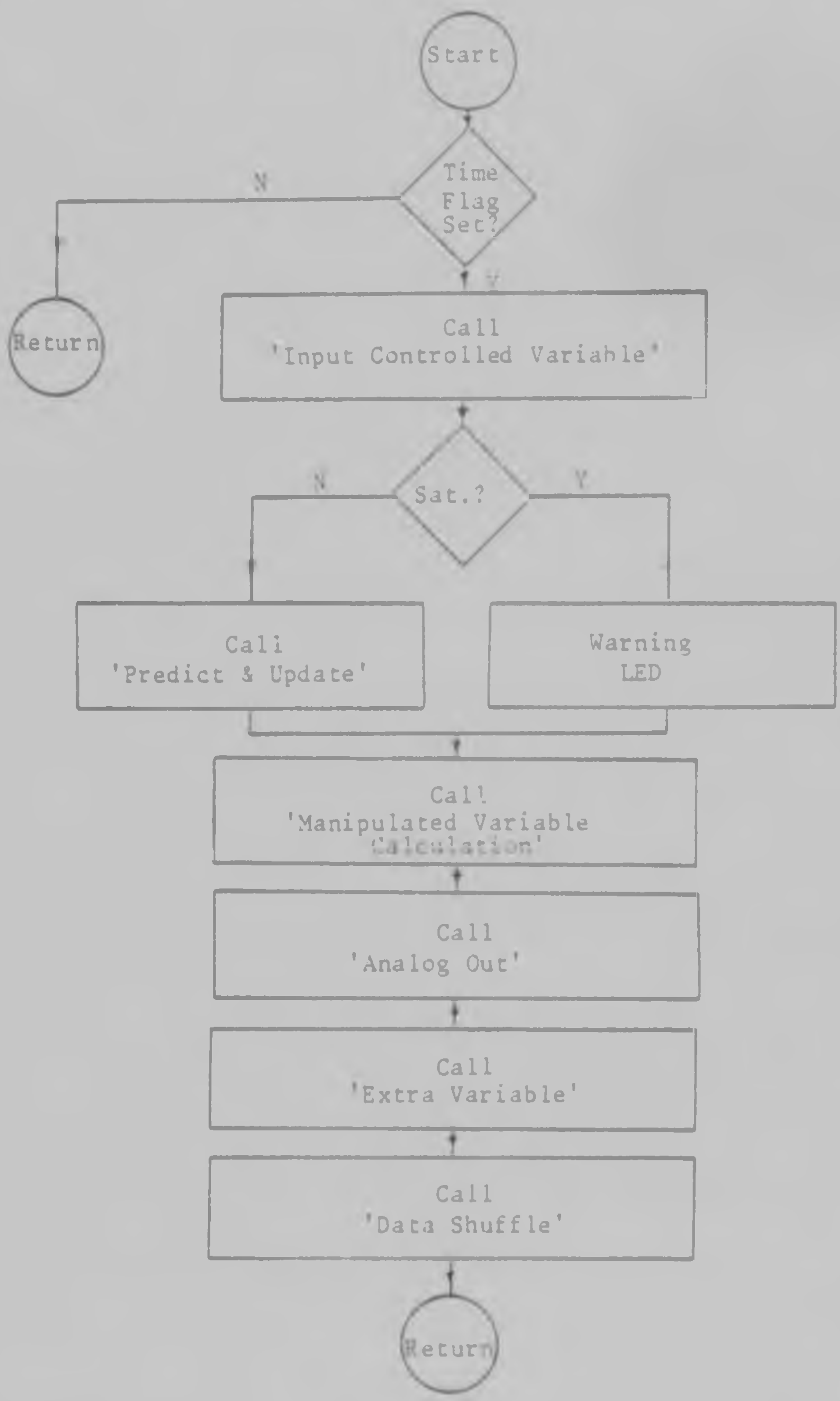


Figure 21

A flag, set by the real time clock is tested to see whether a sampling period has passed. If negative, program execution returns to the calling program. If positive, the controlled variable is measured and checked for saturation at both high and low limits. If there is no saturation, the following modules are called in order

i) Predict and update (model coefficients)
ii) Manipulated variable calculation
iii) Analogue out
iv) Extra variable
v) Data shuffle (prepare for next loop)

and will be described later. If saturation is found to exist, a warning LED (SAT on the control panel) is lit for operator notification, the 'predict and update' module is skipped and execution continues sequentially.

4. Manual Mode Module

This module is similar to the 'automatic mode module' in many respects. It is called by the self tuner control module when the operator desires manual control. If the real time clock generated flag allows, and the controlled variable is not saturated the following modules are called:-

i) Predict and Update
ii) Extra Variable
iii) Data Shuffle

In this mode too, the manipulated variable is handled by the computer, according to the operator's direction request. However, the manipulated variable is not outputted here. Instead a flag is set which notifies the interrupt module that manual operation is under way. The interrupt module then handles timeous outputs to the plant. This ensures much smoother control.

5. Interrupt Processor Module

Once every 40 ms, a hardware generated interrupt causes program flow to revert to this module. Two functions are satisfied here:-

i) Operation of a real time clock.

ii) Handling of manipulated variable during manual operation.

i) At every interrupt a counter is incremented. When reaching a terminal count, the counter is reset and a flag is set. This flag is polled by both 'Auto' and 'Manual' modules in order to keep track of real time.

ii) If conditions are favourable on an interrupt (i.e. manual operation has been requested and the automatic mode is not being interrupted) then the following takes place. The ramp flag is tested to see whether the operator requires ramp up, ramp down or no operation (i.e. increase, decrease manipulated variable or inaction). Dependant on this, an increment in manipulated variable is calculated and passed to 'analog out module' for outputting to the plant. This type of manual operation is useful in that bumpless transfer automatically occurs in either direction and it is compatible with the self tuning algorithms discussed, enabling the controller to keep track of the plant dynamics even in manual mode.

A further feature implemented in this module is the variation of output sensitivity. When manual operation is first requested, the sensitivity is set to a high value. After a set period has passed, sensitivity is decreased and output proceeds much faster. Variables such as sampling period may be changed using interrogation mode.

6. Analogue Out Module

This short module has the function of transferring the controller output to the plant. It accepts a 16 bit binary number representing the incremental change in manipulated variable. This value is added/subtracted to/from the previous manipulated variable and outputted to the plant via one of the two D/A converters on the RTI-1200 board. Both auto and manual modules call this module.

7. Input Controlled Variable Module

This module measures the controlled variable by initiating the A/D conversion, waiting for a period, reading in and storing the variable. Both auto and manual modes call this module.

8. Predict and Update Module

This module essentially attempts to improve on the model of the plant by predicting

$$\Delta\hat{y}(t) = \frac{1}{R1}\left\{\frac{1}{Fg}\left\{\sum \hat{L}i(t-1)\cdot\Delta v(t-i)\right\} + \Delta y(t-i)\right\}$$

where all symbols are defined as before (see Section 3).

The model coefficients are then updated by

$$\hat{L}i(t) = \hat{L}i(t-1) + \Gamma\cdot\Delta v(t-i)\,\{\Delta y(t) - \Delta\hat{y}(t)\}$$

and the estimated steady state gain of the plant is calculated by

$$1/\hat{K} = \frac{\sum \hat{L}i(t)}{R2 - 1}$$

All of the above, i.e. $\hat{L}i(t)$, N, Γ, $1/\hat{K}$ have default values on startup, but they may be altered (at any time) by the operator by an interrogation request. Only auto and manual modules call this routine.

9. Manipulated Variable Calculation Module

This module calculates the desired manipulated variable. The updated model coefficients are passed directly from 'Predict and update module', i.e.

$$\Delta v(t) = \frac{\hat{K}}{R2} \{\sum Qi \cdot \Delta v(t-i)\}$$

$$+ \hat{K} \cdot Fg \{R1 \cdot e(t) - e(t-1)\}$$

where $Q_1 = \frac{1 - R2}{\hat{K}} + \hat{L}_1(t)$

and

$Q_i(t) = \hat{L}i\ (t) + Q_{i-1}(t) \qquad i = 1,N$

Full arrays (N values) of $\hat{L}i(t)$ and $\Delta v(t)$ are kept, however, only one storage space is necessary for $Q_i(t)$ since $Q_{i+1}(t)$ replaces $Q_i(t)$.

$\Delta v(t)$ is then passed to 'Analogue out' module for outputting to the plant. This routine is called by 'auto module' only.

Both 'Predict and Update Module' and 'Manipulated Variable Calculation Module' entail a fair amount of arithmetic manipulation. The Intel supplied FPAL software (floating point arithmetic language) was used to achieve the large variation in number magnitude needed for this application. However, as each floating point number is represented by four bytes (32 bits) the storage capacity needed is quite large.

10. Extra Variable Module

There are two means whereby the operator may be aware of the performance of the self tuner. The first method is an interrogation request. Here all variables may be examined, but the action of the self tuner is stopped during this request. A further method which enables run time information to reach the operator without disturbing the controller, is handled by this module. The functions carried out once every sampling period are:-

i) An output to the terminal of a single preselected estimated model coefficient.

ii) A display of the prediction error visually via a row of six light emitting diodes on the control panel.

iii) The output, via the 2nd D/A channel of the prediction error to a chart recorder.

Function i) entails converting a 32 bit floating point number into decimal (ASCII) format and sending the result to the terminal if the PRT (print) flag is enabled.

Function ii) entails choosing the first prediction error after a reset or an interrogation request as a reference value for those that follow (which are likely to be smaller in magnitude).

The result is a qualitative display which should indicate to the operator whether the self tuner is tuning (i.e prediction error is decreasing with time).

11. Data Shuffle Module

This module is called at the end of any one pass through the controller i.e. once every sampling period. Here data evaluated during the pass is 'shuffled' to prepare for the next pass, i.e. $\Delta v(t)$ is shuffled back one step in time to $\Delta v(t-1)$, etc. Program flow then returns to 'Self Tune Control Module' and the cycle is repeated.

All the above software, including the necessary [illegible] routines and one kilobyte of the 80P monitor program amounted to

9 Kb Code

1/2 Kb Data

170 (decimal) bytes of stack

The execution of a single automatic mode loop was calculated as not exceeding 500 ms when using 50 coefficients.

6. AN APPLICATION OF THE SELF TUNER

A suitable test process, in the form of a flow rig was used to evaluate the self tuner. The criteria used in choosing this were:-

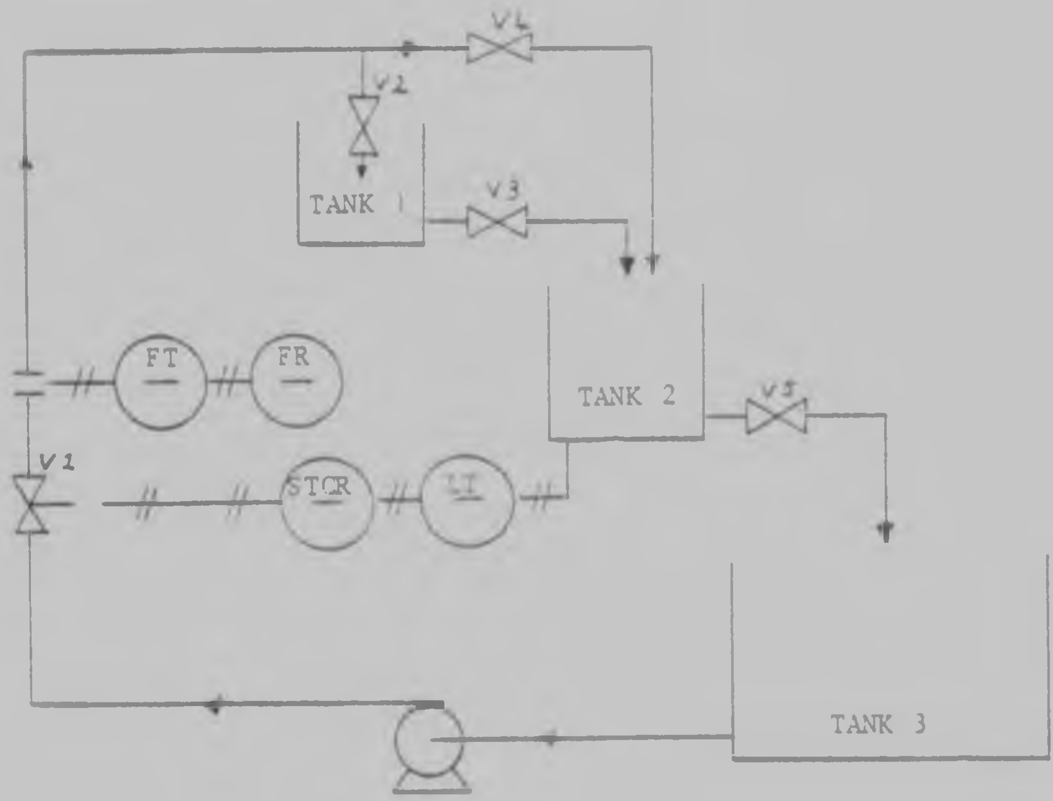
i) Minimal consequences should failure occur.

ii) Similarity to an industrial type plant.

6.1 The Test Plant

The flow rig [illegible] own in the following photograph and schematic. The pho[illegible] y shows the flow rig on the right and the instrume[illegible] sing on the left. The latter contains a standard PI controller and associated displays and chart recorders. On the far left is the self tuner housing linked by flat cable to the control panel. A chart recorder and teletype are also visible.

igure 22



FLOW RIG SCHEMATIC

Figure 23

LT - Level Transmitter

FR - Flow Recorder

FT - Flow Transmitter

STCR - Self Tuning Controller and Recorder

 - Flow Restrictor

The schematic shows the test rig as set up with the self tuning controller. The objective was to control the level of water in tank 2 by manipulation of V1. Tank 3 was merely a reservoir while manipulation of V2, V3, V4 and V5 enabled the configuration of differing plants. In particular V4, V5 open and V2, V3 closed results in a single order system while V2, V3 and V5 open and V4 closed results in a second order system.

Objectives

It must be emphasised once more that no global conclusions can result from an implementation of this sort. There is no substitute for a thorough theoretical analysis in this regard. However, one can look for general trends and apparent characteristics. Again, chief areas of interest are:-

i) Overall system stability.
ii) Convergence of the model coefficients.
iii) Properties of the resulting controller.

Here, the simulation study mentioned previously and the implementation must surely complement one another. Simulation has provided an insight into the properties of the self tuner. However, certain assumptions were made when mathematically modelling the system. In particular, the simulation used a linear plant. The test rig, as with most practical plants, has non-linear characteristics. The self tuner assumes a linearized model about an operating point. It is in the light of such 'real' plant characteristics that the self tuning properties must be evaluated.

6.2 Tests Undertaken and Results

Stability

The two important values on which stability depends are

i) The initial coefficients $\hat{L}i(o)$.

ii) The updating constant Gamma.

The plant was configured as a second order process because of the increased difficulty in controlling and identifying it compared to a first order system.

A major problem may occur when $\hat{L}i(o)$ are chosen to give an unstable closed loop system. Though the self tuning would act to stabilize the system, catastrophic results may occur before this. With this in mind, the weighting constant Gamma was set to zero, effectively cancelling out the self tuning action and the closed loop system was subjected to a 10% change in set point for varying values of $\hat{L}i(o)$. When actually implementing the self tuner as much information as is known should be used to arrive at the initial coefficient values $\hat{L}i(o)$. However to gain an insight into the insensitivity of system stability to $\hat{L}i(o)$, all $\hat{L}i(o)$ were chosen equal, i = 1 to N. The following table depicts the results. For all runs 40 coefficients were used (N = 40), Sampling Period = 30 seconds and Gamma = 0.

$\hat{\omega}(o)$, i = 1 to 40	RESULT
$6{,}0 \times 10^{-4}$	Control Variable unstable, tank overflowed
$1{,}2 \times 10^{-3}$	Manipulated variable (valve position) saturates at both ends. Tank level highly oscillatory but finally settles to set point.
$2{,}5 \times 10^{-3}$	No saturation, response oscillatory See Figure 24.
	:
1×10^{-2}	Damped Response
	:
	Response slows down
2×10^{-1}	No cont .ol action apparent

Figure 24

Step Response

$\hat{L}_1(0) = 2{,}5 \times 10^{-3}$

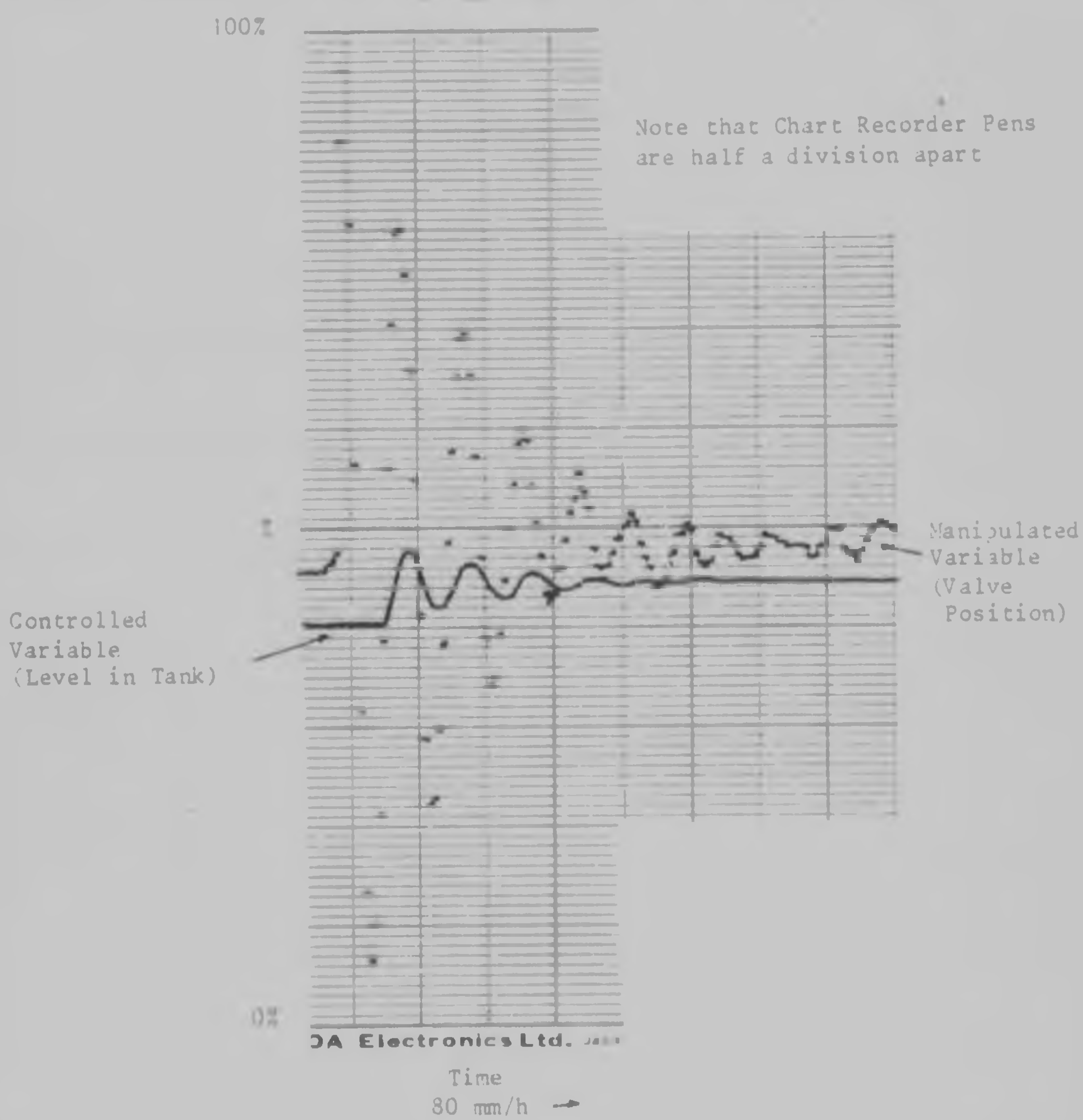
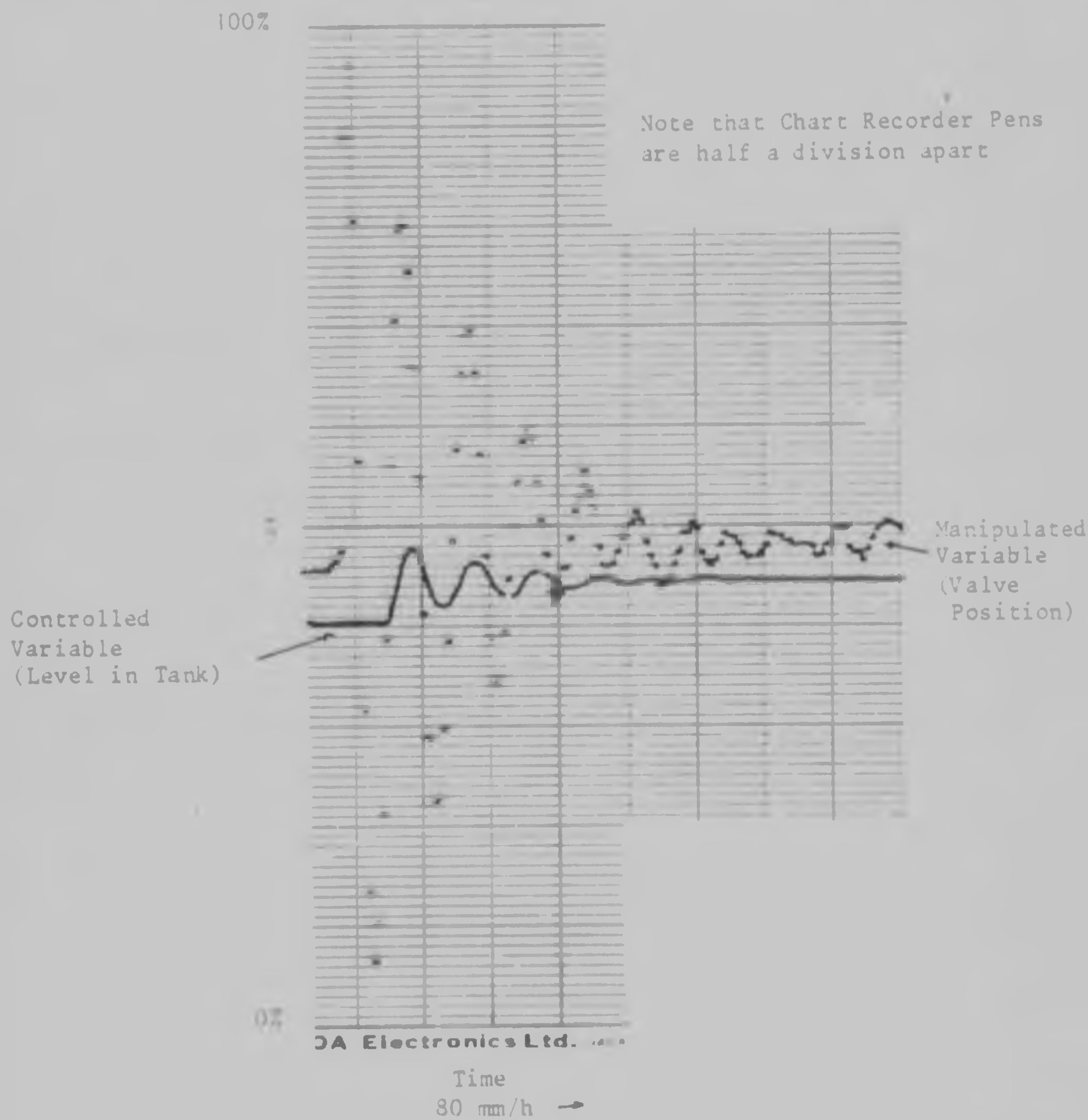


Figure 24

Step Response

$\dot{L}_i(0) = 2{,}5 \times 10^{-3}$



Negative values of $\hat{Li}(t)$ were also found to caus instability. This occurred when Gamma was large enough to cause over compensation in the update equation when self tuning was applied (Gamma non zero)

$$\hat{Li}(t) = \hat{Li}(t-1) + \Gamma \cdot \Delta v(t-1) \{\Delta y(t) - \Delta \hat{y}(t)\}$$

This was particularly apt to happen at startup if the initial values of $\hat{Li}(o)$ were chosen small. Larger values chosen for $\hat{Li}(o)$ often produced stable results for the same Gamma.

Convergence

Points looked for here were

i) The variation of the estimated coefficients fr ome constant initial value $\hat{Li}(o)$ to same final value $\hat{Li}(\alpha)$

ii) The speed of convergence from $\hat{Li}(o)$ to $\hat{Li}(\alpha)$.

iii) The final values of $\hat{Li}(\alpha)$.

To evaluate all of these points the following two runs were undertaken:-

	RUN 1	RUN 2
COEFFICIENTS AT T = 0	$\hat{L}i(o) = 5 \times 10^{-3}$	$\hat{L}i(o) = 1 \times 10^{-2}$
NO. OF COEFFICIENTS	N = 40	N = 40
SAMPLING PERIOD	Ts = 30 Seconds	Ts = 30 Seconds
UPDATE CONSTANT	Gamma = 1×10^{-9}	Gamma = 5×10^{-7}
FILTER PARAMETERS	R1 = R2	R1 = R2

RUN 1

The system was allowed to run night and day for over three weeks. As coefficient convergence was found to be slow, set point changes were introduced during daytime to improve system excitation. The tenth coefficient was printed out hourly to the teletype to serve as an indicator as to the convergence of the model. At the end of the period mentioned, the coefficients had apparently not reached steady values. The run was therefore halted and Run 2 initiated.

RUN 2

Here the initial coefficients $\hat{L}i(o)$ were increased to ensure stability with increased weighting constant Gamma, to speed up convergence. The results were remarkably different to those of Run 1. Within four days and four nights the coefficients had apparently converged. Again set point changes over a ten percent operating region were used to excite the system during the day. At the end of four days, set point changes caused no marked changes in the coefficient values. (Though the values were noted to vary slightly about a mean). The resulting plot of $\hat{L}i(\infty)$ against i is shown in Figure 25.

	RUN 1	RUN 2
COEFFICIENTS AT T = 0	$\hat{L}i(o) = 5 \times 10^{-3}$	$\hat{L}i(o) = 1 \times 10^{-2}$
NO. OF COEFFICIENTS	N = 40	N = 40
SAMPLING PERIOD	Ts = 30 Seconds	Ts = 30 Seconds
UPDATE CONSTANT	Gamma = 1×10^{-9}	Gamma = 5×10^{-7}
FILTER PARAMETERS	R1 = R2	R1 = R2

RUN 1

The system was allowed to run night and day for over three weeks. As coefficient convergence was found to be slow, set point changes were introduced during daytime to improve system excitation. The tenth coefficient was printed out hourly to the teletype to serve as an indicator as to the convergence of the model. At the end of the period mentioned, the coefficients had apparently not reached steady values. The run was therefore halted and Run 2 initiated.

RUN 2

Here the initial coefficients $\hat{L}i(o)$ were increased to ensure stability with increased weighting constant Gamma, to speed up convergence. The results were remarkably different to those of Run 1. Within four days and four nights the coefficients had apparently converged. Again set point changes over a ten percent operating region were used to excite the system during the day. At the end of four days, set point changes caused no marked changes in the coefficient values. (Though the values were noted to vary slightly about a mean). The resulting plot of $\hat{L}i(\infty)$ against i is shown in Figure 25.

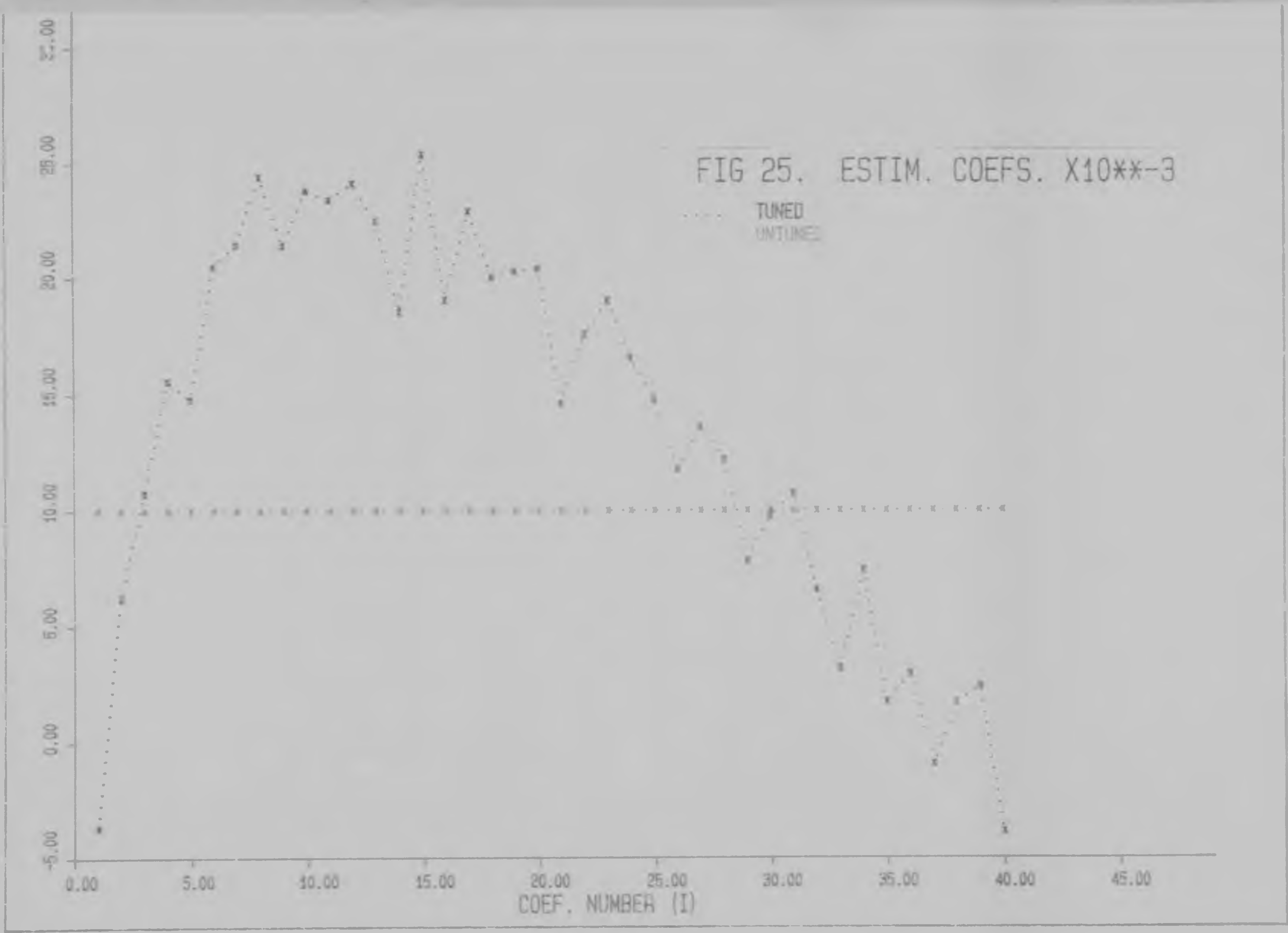


FIG 25. ESTIM. COEFS. X10**-3
TUNED
UNTUNED
-5.00
0.00
5.00
10.00
15.00
20.00
25.00
27.00
0.00
5.00
10.00
15.00
20.00
25.00
30.00
35.00
40.00
45.00
COEF. NUMBER (I)

Here the bell shaped curve, similar to the impulse response curve expected, can be clearly seen. Note that for both large and small i, coefficients have actually taken on negative values, (i = 1, 37, 40).

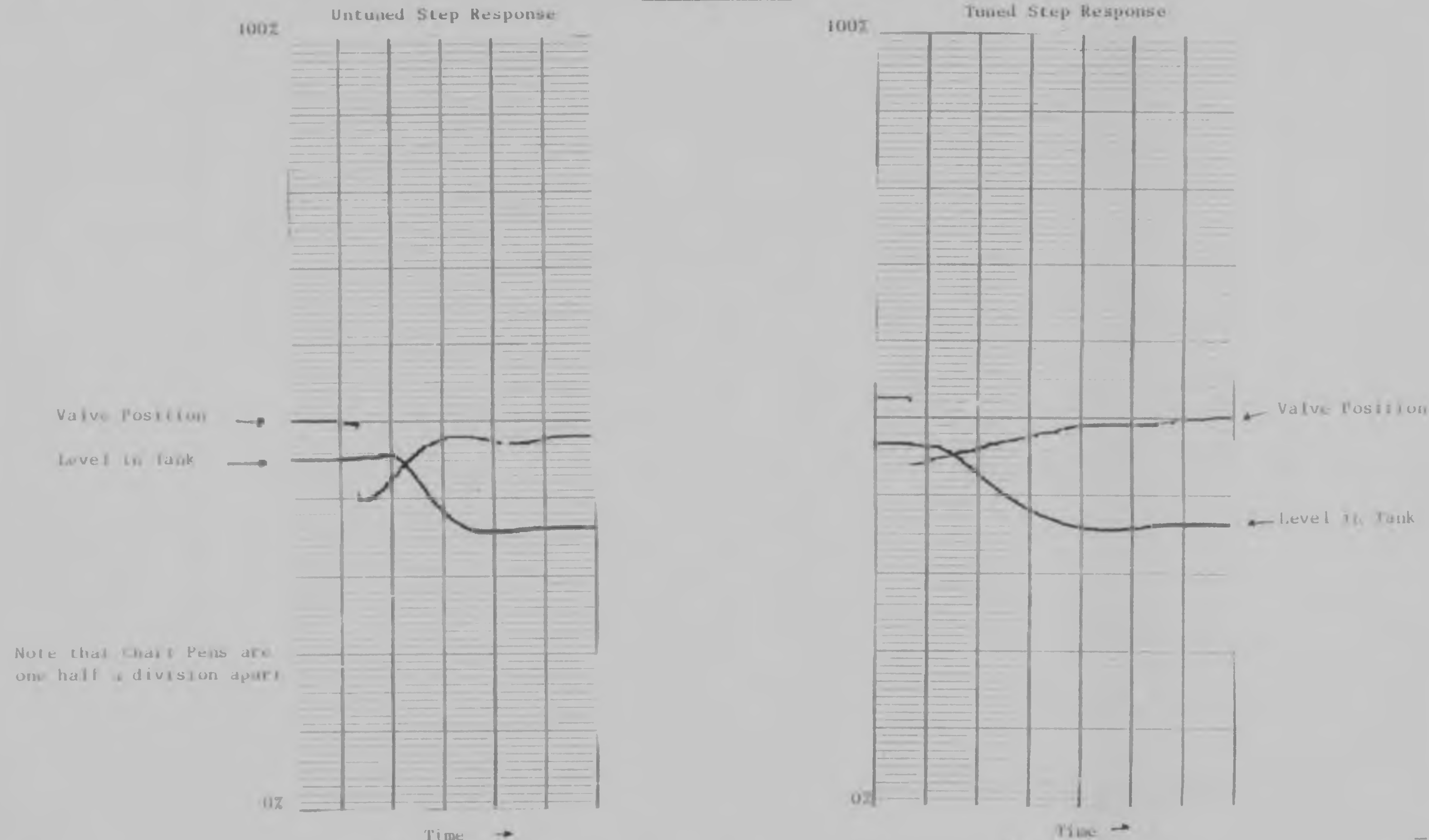
Resulting Controller

Figure 26 shows the step response of the system corresponding to Run 2. Figure 26(a) represents the closed loop set point response (step change) with initial conditions only and no self tuning (Gamma = 0). Figure 26(b) represents the set point step response after the model coefficients [illegible]rently converged. The self tuned system has a much slower [illegible]se with a dominant time constant of 7,5 minutes. In order to compare this with the open loop response a number of open loop step tests were undertaken. Unfortunately due to the inherent non linearity of the plant it was found to be impossible to produce repeatable results for the tests and open loop dominant time constants were measured between 6 and 10 minutes.

The plant was also controlled by the PI Controller available. Here the Ziegler Nichols method was used to tune the controller over an operating region of 10%. It was found that set point changes out of this region caused severe limit cycling. The self tuner however, was capable of withstanding a [illegible]0% set point change (once self tuned) without instability.

Figure 26

STEP RESPONSES



(a) 80 mm/h

(b) 80 mm/h

7. DISCUSSION OF THE SELF TUNER PROPERTIES

7.1 Stability

The self tuner showed remarkable stability properties, both at the crucial initial stage and thereafter. Although the dynamics of the plant were completely neglected in specifying the initial model coefficients, system stability was attainable for a wide choice of these coefficients.

The overall system was found to become unstable when a large number of coefficients became negative. The instability is probably due to the fact that the estimated steady state gain $1/\bar{K}$ then becomes negative as well. The result is effectively positive feedback. In general the combination of unpredictable noise acting on the system and a large weighting constant Gamma were responsible for the negating of the model coefficients. easy solution to this problem is the minimization of Gamma.

The above discussion suggests that while it is not crucial to specify 'correct' values of model coefficients $\hat{L}i(o)$ and Gamma to start the self tuner, these choices must be a fair approximation. To achieve this, consider the following equation of the estimated steady state gain:-

$$\frac{1}{\bar{K}} = \frac{\sum \hat{L}i(t)}{R2 - 1}$$

Since both simulation and implementation suggest that all model coefficients can be chosen equal to one another at start up, then choose

$$\hat{L}i(o) = \frac{R2 - 1}{\hat{K} \cdot N}$$

where N represents the number of coefficients used (this choice has been discussed before in Chapter 3). R2 is chosen by the process engineer, or the default value may be used. However, an estimate of the process, steady state gain needs to be specified. If this is known accurately, good and well. If not, an approximate guess is likely to suffice. As can be seen from Section 6.2 where $\hat{L}i(o)$ were all chosen = $1{,}2 \times 10^{-3}$, the resulting estimated gain $1/\hat{K} = 0{,}1728$, and a closed loop stable system was achieved. However, from RUN 2 noted in Section 6.2, the estimated gain when the model had apparently converged was $1/\hat{K} = 1{,}8507$ which is representative c he actual plant gain.

The choice of Gamma must be made to prevent the model coefficients from becoming too large (causing controller switch off) or too small or negative such as to cause unstable control. Here the update equation must be born in mind:-

$$\hat{L}i(t) = \hat{L}i(t-1) + \Gamma \cdot \Delta v(t-1) \{\Delta y(t) - \Delta \hat{y}(t)\}$$

Since both simulation and implementation suggest that all model coefficients can be chosen equal to one another at start up, then choose

$$\hat{L}i(o) = \frac{R2 - 1}{\hat{K} \cdot N}$$

where N represents the number of coefficients used (this choice has been discussed before in Chapter 3). R2 is chosen by the process engineer, or the default value may be used. However, an estimate of the process, steady state gain needs to be specified. If this is known accurately, good and well. If not, an approximate guess is likely to suffice. As can be seen from Section 6.2 where $\hat{L}i(o)$ were all chosen = $1,2 \times 10^{-3}$, the resulting estimated gain $1/\hat{K} = 0,1728$, and a closed loop stable system was achieved. However, from RUN 2 noted in Section 6.2, the estimated gain when the model had apparently converged was $1/\hat{K} = 1,8507$ which is representative of the actual plant gain.

The choice of Gamma must be made to prevent the model coefficients from becoming too large (causing controller switch off) or too small or negative such as to cause unstable control. Here the update equation must be born in mind:-

$$\hat{L}i(t) = \hat{L}i(t-1) + \Gamma \cdot \Delta v(t-1) \{\Delta y(t) - \Delta \hat{y}(t)\}$$

The implementation suggests that over compensation of the coefficients occurs either at start up (where prediction error is large) or when a sizable disturbance occurs (again prediction error is large).

Using the initial model coefficients as a basis, choose Gamma assuming maximum variation of prediction error and manipulated variable,

i.e. $$\Gamma = \frac{a \cdot \hat{L}i(o)}{\Delta v(t-i)_m \cdot \{\Delta y(t) - \Delta \hat{y}(t)\}_m}$$

where a is a fraction of unity and m subscript represents the maximum expected value.

This should ensure that over compensation does not occur.

7.2 Convergence Properties

Here the implementation results were very similar to simulation. Given initial model coefficients that could not possibly represent the dynamics of the plant, the model converged to values that could well represent it. In particular the model coefficients $\hat{L}i(\infty)$ when plotted against i represent the familiar bell shaped curve so similar to the impulse response. The weighting constant Gamma was clearly shown to affect the speed of the model convergence. Figure 25 ($Li(\infty)$ versus i) seems to suggest that a smaller value of Gamma would have been more appropriate here.

This would have produced a smoother curve and ensured that all tuned coefficients were positive. However, the resulting decrease in speed of convergence of the model might have been untenable.

The calculation of Gamma in the previous section makes no mention of the affect of Gamma on the speed of the convergence of the model. In fact, this aspect is difficult to quantify other than to state that increased Gamma increases speed of convergence. The previous section's calculation of Gamma should therefore be taken as a first approximation and may be increased if desired, bearing in mind that instability may occur.

7.3 The Resulting Controller

What is of particular interest is the resulting controller, once the model has converged to a final form. Figure 26(b) clearly shows that the self tuner has converged to a slower closed loop response than that represented by the initial conditions in Figure 26(a) and is particularly gratifying and suggests that the algorithm fulfills the self tuning property, i.e. that if the coefficients converge, (to unique values or not) the resulting controller is such that the open and closed loop dynamics are equivalent. This is difficult to prove conclusively here, and suffice to say that (as with simulation), upon apparent convergence of the coefficients, the resulting controller is a good approximation to the required controller.

In addition, whether disturbed by a set point change or extraneous noise, the steady state error was zero at all times during any (stable) run. This concurred well with theory (see Chapter 3).

This would have produced a smoother curve and ensured that all tuned coefficients were positive. However, the resulting decrease in speed of convergence of the model might have been untenable.

The calculation of Gamma in the previous section makes no mention of the affect of Gamma on the speed of the convergence of the model. In fact, this aspect is difficult to quantify other than to state that increased Gamma increases speed of convergence. The previous section's calculation of Gamma should therefore be taken as a first approximation and may be increased if desired, bearing in mind that instability may occur.

7.3 The Resulting Controller

What is of particular interest is the resulting controller, once the model has converged to a final form. Figure 26(b) clearly shows that the self tuner has converged to a slower closed loop response than that represented by the initial conditions in Figure 26(a) and is particularly gratifying and suggests that the algorithm fulfills the self tuning property, i.e. that if the coefficients converge, (to unique values or not) the resulting controller is such that the open and closed loop dynamics are equivalent. This is difficult to prove conclusively here, and suffice to say that (as with simulation), upon apparent convergence of the coefficients, the resulting controller is a good approximation to the required controller.

In addition, whether disturbed by a set point change or extraneous noise, the steady state error was zero at all times during any (stable) run. This concurred well with theory (see Chapter 3).

All in all, the self tuner compares favourably with those self tuners mentioned in the literature. The performance criterion is such that open and closed loop responses are equated. The controller should be most useful in ensuring stability when time varying delays or strong non linearities are prevalent. The non parametric basis of the algorithm ensures that the equations are easy to implement in practice as only standard arithmetic functions are needed. The choice of certain variables to start the self tuner are not critical and only minimal prior knowledge of the plant is needed. The problem of estimating a steady state offset coefficient and the related problem of zero steady state error with respect to set point is handled automatically. (Certain other self tuners have only had marginal success in this regard). (See Chapter 2). A preplant filter has been included to allow closed loop variation, at the operator's discretion.

7.4 Suggestions for Future Work

First and foremost, the self tuner needs to be analysed from a theoretical viewpoint, especially with regard to the convergence properties and the resulting controller characteristics. In particular, it is felt that the algorithm proposed may well exhibit the self tuning property for all open loop stable plants, provided there is sufficient excitation. Analysis may also provide a better insight into the initial choices of the model coefficients and the weighting constant Gamma. Possibly the choice of Gamma itself could be automated.

All in all, the self tuner compares favourably with those self tuners mentioned in the literature. The performance criterion is such that open and closed loop responses are equated. The controller should be most useful in ensuring stability when time varying delays or strong non linearities are prevalent. The non parametric basis of the algorithm ensures that the equations are easy to implement in practice as only standard arithmetic functions are needed. The choice of certain variables to start the self tuner are not critical and only minimal prior knowledge of the plant is needed. The problem of estimating a steady state offset coefficient and the related problem of zero steady state error with respect to set point is handled automatically. (Certain other self tuners have only had marginal success in this regard). (See Chapter 2). A preplant filter has been included to allow closed loop variation, at the operator's discretion.

7.4 Suggestions for Future Work

First and foremost, the self tuner needs to be analysed from a theoretical viewpoint, especially with regard to the convergence properties and the resulting controller characteristics. In particular, it is felt that the algorithm proposed may well exhibit the self tuning property for all open loop stable plants, provided there is sufficient excitation. Analysis may also provide a better insight into the initial choices of the model coefficients and the weighting constant Gamma. Possibly the choice of Gamma itself could be automated.

Two major problems become apparent during the testing of the self tuner. Firstly, being constrained to real time, much time was wasted waiting for results. Hence the self tuner was not as thoroughly tested as might have been the case. Secondly, the transfer function of the plant was not known exactly nor would any identification scheme have been necessarily better than that used by the self tuner. The open loop step response tests were also not definitive. Hence it was difficult to determine whether in fact the closed and open loop response were identical (they were definitely similar) once the coefficients had apparently converged.

In order to solve both of these problems, it is suggested that the self tuner be thoroughly tested out on an analogue computer. Here the plant can be prespecified, results may be obtained faster and (in the absence of theoretical analysis) the self tuning property may be tested on plants with varying dead times and non linearities.

7.5 Conclusions

i) A non parametic self tuning controller algorithm has been proposed and implemented in practice. The explicit self tuner was able to satisfactorily control the level in the tank of a laboratory flow rig.

ii) The controller equates open and closed loop dynamics. This precludes the necessity of prespecifying a different desired response for different plants.

iii) The controller includes a facility to vary the closed loop response if required.

iv) The model used for tuning is non parametic. Hence the problems associated with parametic models do not arise.

v) The self tuner should provide adequate control for use on non minimum phase systems, in particular systems with uncertain dead times.

vi) The self tuner solves both the servo and regulator problems.

vii) Control action is not excessive.

viii) The self tuner exhibits zero steady state error characteristics even while 'untuned'.

ix) System stability seems to show a high degree of insensitivity to initial conditions, especially the starting values of the model coefficients.

x) Tests undertaken suggest that the self tuner exhibits

a) Reasonable convergence properties;

b) The self tuning property.

xi) The self tuner is particularly easy to install and commission.

xii) In general, open loop unstable plants are not controllable by this method.

xiii) The performance criterion may not be suitable for certain applications.

xiv) Further work needs to be done in the theoretical analysis of the controller, especially with regard to the convergence properties and the characteristics of the resulting controller.

xv) The self tuner should be particularly useful

* For those processes where a constant controller would need regular retuning;

* Where the requirements of the process are such that stability and zero steady state error at all times are of prime importance.

BIBLIOGRAPHY

1. WITTENMARK, B. Stochastic Adaptive Control Methods, A Survey. INT. J. CONTROL, 1975 Vol. 21, No. 5, pp 705-730.

2. ASTROM, K.J. Theory and Applications of Adaptive Control. PROC. IFAC, 8th World Congress, Kyoto, Japan 1981.

3. MORRIS, H.M. Operator Convenience is Key as Process Controllers Evolve CONTROL ENG., March 1981, pp 65-70.

4. FELDBAUM, A.A. D Control Theory I - IV, REMOTE CONTROL, 21 1960.

5. NISHIKAWA, Y SANNOMIYA, N A Method for Auto Tuning of PID Control Parame PROC. IFAC, 8th World Congress, Kyoto, Japan 198.

6. ANDREIEV, N. A New Dimension: A Self Tuning Controller that Continually Optimizes PID Constants, CONTROL ENG, August 1981.

7. ASTROM, K.J., EYKHOFF, P. System Identification, A Survey. AUTOMATICA Vol. 7, pp 123-162, Pergamon Press 1971.

8. DAVIS, W.D.T. System Identification for Self Adaptive Control, Wiley & Sons 1970.

9. WELLSTEAD, P.E., EDMUNDS, J.M., PRAGER, D., ZANKER, P. Self Tuning Pole/Zero Assignment Regulators, INT. J. CONTROL, Vol. 30, No. 1, 1-26 1979.

10. WELLSTEAD, P.E., ZANKER, P. Servo Self Tuners INT. J. CONTROL Vol. 30, No. 1, 27-36 1979.

BIBLIOGRAPHY

1. WITTENMARK, B. Stochastic Adaptive Control Methods, A Survey. INT. J. CONTROL, 1975 Vol. 21, No. 5, pp 705-730.

2. ASTROM, K.J. Theory and Applications of Adaptive Control. PROC. IFAC, 8th World Congress, Kyoto, Japan 1981.

3. MORRIS, H.M. Operator Convenience is Key as Process Controllers Evolve CONTROL ENG., March 1981, pp 65-70.

4. FELDBAUM, A.A. Dual Control Theory I - IV, AUTOMN REMOTE CONTROL, 21 1960.

5. NISHIKAWA, Y., SANNOMIYA, N. A Method for Auto Tuning of PID Control Parameters, PROC. IFAC, 8th World Congress, Kyoto, Japan 1981.

6. ANDREIEV, N. A New Dimension: A Self Tuning Controller that Continually Optimizes PID Constants, CONTROL ENG, August 1981.

7. ASTROM, K.J., EYKHOFF, P. System Identification, A Survey. AUTOMATICA Vol. 7, pp 123-162, Pergamon Press 1971.

8. DAVIS, W.D.T. System Identification for Self Adaptive Control, Wiley & Sons 1970.

9. WELLSTEAD, P.E., EDMUNDS, J.M., PRAGER, D., ZANKER, P. Self Tuning Pole/Zero Assignment Regulators, INT. J. CONTROL, Vol. 30, No. 1, 1-26 1979.

10. WELLSTEAD, P.E., ZANKER, P. Servo Self Tuners INT. J. CONTROL Vol. 30, No. 1, 27-36 1979.

11. WELLSTEAD, P.E.,
EDMUNDS, J.M. Least Squares Identification or Closed Loop Systems, INT. J. CONTROL, 1975, Vol. 21, No. 4, pp 689-699.

12. ASTROM, K.J.,
WITTENMARK, B. Self Tuning Controllers Based on Pole Zero Placement IEE PROC., Vol. 127, Pt. D, No. 3, May 1980.

13. WELLSTEAD, P.E.,
PRAGER, D.,
ZANKER, P. Pole Assignment Self Tuning Regulators, PROC IEE, Vol. 126, No. 8, August 1972.

14. CLARKE, D.W.,
GAWTHROP, P.J. Self Tuning Controller, PROC IEE, Vol. 122, No. 9, September 1975 pp 929-934.

15. CLARKE, D.W.,
GAWTHROP, P.J. Self Tuning Control, PROC IEE, Vol. 126, No. 6, June 1979, pp 633-639.

16. ASTROM, K.J.,
BORRISSON, U.,
LJUNG, J.,
WITTENMARK, B. Theory and Applications of Self Tuning Regulators, AUTOMATICA, Vol. 13, pp 457-476, 1977.

17. ASTROM, K.J.,
WITTENMARK, B. On Self Tuning Regulators AUTOMATICA, Vol. 9, pp 185-199, 1973.

18. EYKHOFF, P. System Identification - Parameter and State Estimation, Wiley & Sons, 1974.

19. GAWTHROP, P.J. Some Interpretations of the Self Tuning Controller, PROC IEE, Vol. 124, 889, 1977.

20. BORRISSON, U.,
WITTENMARK, B. An Industrial Application of a Self Tuning Regulator, PROC IFAC/IFIP Conference 1974.

21. GUSTAVSSON, I.,
LJUNG, L.,
SODERSTROM, T. Identification of Processes in Closed Loop - Identifiability and Accuracy Aspects,
AUTOMATICA, Vol. 13, pp 59-75, Pergamon Press 1977.

22. CLARKE, D.W.,
GAWTHROP, P.J. Implementation and Application of Microprocessor - Based Self Tuners,
AUTOMATICA, Vol. 17, No. 1, pp 233-244, 1981.

23. CLARKE, D.W.,
GAWTHROP, P.J.,
COPE, P.J. Feasibility Study of the Application of Microprocessors to Self Tuning Regulators,
Report No. 1137/75, 1975, Oxford Univ. E.L. Report.

24. TU, F.C.Y.,
TSING, J.Y.H. Synthesizing a Digital Algorithm for Optimized Control,
INTECH May 1979, pp 52-55.

25. FRANKLIN, F.G.,
POWELL, J.D. Digital Control of Dynamic Systems,
Addison Wesley 1980.

26. SHEIRAH, M.A.,
MALIK, O.P. Self Tuning Microprocessor Universal Controller,
IEEE TRANS. on IND. ELECTR. Vol. IE-29, No. 1, Feb 1982.

27. MORRIS, A.J.,
WRIGHT, A.R.,
NAZER, Y.,
CHRISHOLM, K.
WOOD, R.K.,
LIEUSON, H. Self Tuning Control of Some Pilot Plant Processes,
Microprocessors and Microsystems, Vol. 5, No. 1,
Jan/Feb 1981.

28. SODERSTROM, T., GUSTAVSSON, I., LJUNG, L. — Identifiability Conditions for Linear Systems Operating in Closed Loop
INT. J. CONTROL, 1975, Vol. 21, No. 2, pp 243-255.

29. In Circuit Emulation 80, Operator's Manual
INTEL CORP, No. 98-1[illegible]C, 1975.

30. PLM 80, Programming Manual
INTEL CORP, No. 9800268B, 1976.

31. 8080/8085, Assembly Language Programming Manual,
INTEL CORP, No. 9800301-04, 1977.

32. 8080/8085, Floating Point Arithmetic Library User's Manual
INTEL CORP, No. 9800452-03, 1977.

33. SBC-80/10, Hardware Reference Manual,
INTEL CORP, No. 98-230B, 1976.

34. Advanced Continuous Simulation Language (ACSL),
User Guide/Reference Manual,
Mitchell and Gauthier Assoc.

35. RTI 1200 Real Time Interface, User's Manual
ANALOG DEVICES, 1976.

36. HARRIS, C.J., BILLINGS S.A. (EDITORS) — Self Tuning and Adaptive Control,
IEE Control Engineering Series 15, 1981.

37. SMITH, C.L. — Digital Control of Industrial Processes,
Computing Surveys, Vol. 2, No. 3, 1970.

38. WILLIAMS, T.J. — Two Decades of Change - A Review of the 20 Year History of Computer Control,
CONTROL ENG., September 1977, pp 71-76.

39. GUPTA, S.C.,
HASDORF, L., <u>Fundamentals of Automatic Control</u>,
John Wiley & Sons, 1970.

40. KATZ, P. <u>Digital Control Using Microprocessors</u>,
Prentice Hall Inc. 1981.

41. MCLAREN, S.G.,
HATHORN, A.P. <u>Advances in Model Following Adaptive Control Theory</u>,
Proceedings-Symposium on Control Theory and Applications
CSIR, Pretoria, June 1982.

APPENDIX

APPENDIX SECTION A

Final Derivation of Algorithm

Given the following closed loop system:-

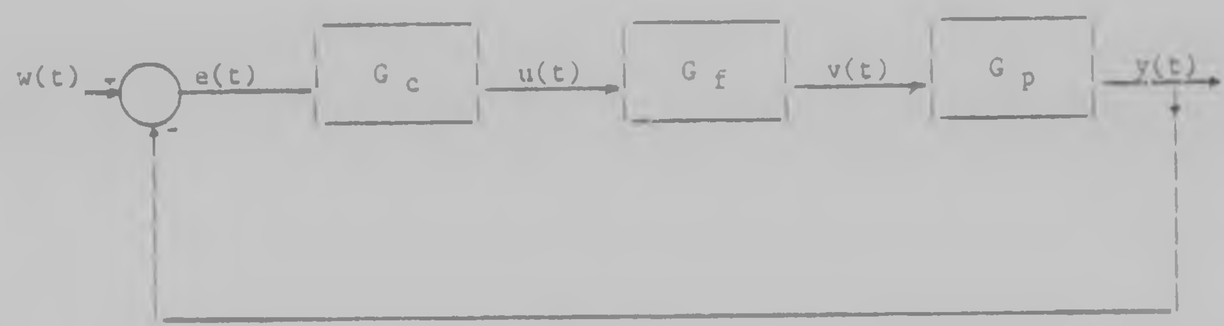


Figure 1(a)

G_c = Controller
G_f = Preplant Filter $= F_g\,(R1 - z^{-1})/(R2 - z^{-1})$
G_p = Plant

The following equations were developed in the text prior to the inclusion of the filter:-

$$G_p = \frac{Y(z)}{U(z)} = \left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots$$

$$+ (c_{N-1} - c_N)z^{-N}$$

and $$G_c = K/\{1 - K\{\left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots$$

$$+ (c_{N-1} - c_N)z^{-N}\}\}$$

See Chapter 3.

It is now worthwhile to revamp the above equations to include the filter G_f, i.e. the effective 'plant' now becomes:-

$$G_f \times G_p = \frac{Y(z)}{U(z)} = \left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots$$

$$+ (c_{N-1} - c_N)z^{-N}$$

APPENDIX SECTION A

Final Derivation of Algorithm

Given the following closed loop system:-

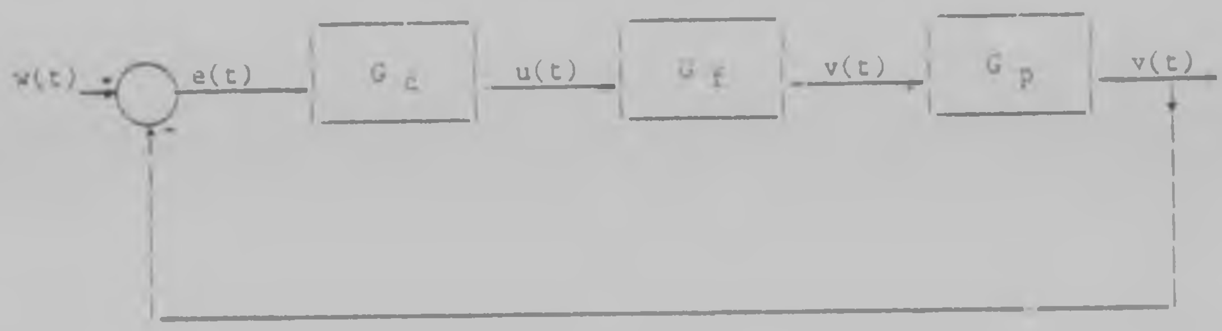


Figure 1(a)

G_c = Controller
G_f = Preplant Filter = $F_g\,(R1 - z^{-1})/(R2 - z^{-1})$
G_p = Plant

The following equations were developed in the text prior to the inclusion of the filter:-

$$G_p = \frac{Y(z)}{U(z)} = \left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots + (c_{N-1} - c_N)z^{-N}$$

and $$G_c = K/\left\{1 - K\left\{\left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots + (c_{N-1} - c_N)z^{-N}\right\}\right\}$$

See Chapter 3.

It is now worthwhile to revamp the above equations to include the filter G_f, i.e. the effective 'plant' now becomes:-

$$G_f \times G_p = \frac{Y(z)}{U(z)} = \left(\frac{1}{K} - c_1\right)z^{-1} + (c_1 - c_2)z^{-2} \ldots + (c_{N-1} - c_N)z^{-N}$$

The controller, to set the closed loop response

$$\frac{G_c G_f G_p}{1 + G_c G_f G_p} = G_f G_p' \qquad (G_p' \text{ has unity steady state gain})$$

is $$G_c = \frac{K}{1 - K\,G_f G_p'} = \frac{U(z)}{E(z)}$$

$$= \frac{K}{1 - K\,\{(\frac{1}{K} - c_1)z^{-1} + (c_1 - c_2)z^{-2} \ldots + (c_{N-1} - c_N)z^{-N}\}}$$

Since both G_c and G_f are both software implementations of transfer functions, we can combine them into one equation, cancelling out the intermediate variable u(t) and including the actual manipulated variable v(t), i.e.

$$G_c = \frac{U(z)}{E(z)} \qquad G_f = \frac{V(z)}{U(z)}$$

Therefore $$G_c G_f = \frac{V(z)}{E(z)}$$

$$= \frac{(R1 - z^{-1})}{(R2 - z^{-1})} \cdot \frac{K\,F_g}{\{1 - K\{(1/K - c_1)z^{-1} \ldots (c_{N-1} - c_N)z^{-N}\}\}}$$

which when multiplied out, combining like values of z and extracting $(1 - z^{-1})$

$$V(z)\{R2 + (KR2c_1 - 1) + \ldots (KR2c_N - Kc_{N-1})z^{-N}\} \times (1 - z^{-1})$$
$$= E(z)\,K \cdot F_g \cdot (R1 - z^{-1})$$

which in the time domain is

$$\Delta v(t) = \frac{K}{R2} \cdot \{(\frac{1}{K} - R2c_1) \cdot \Delta v(t-1) + \ldots + (c_{N-1} - R2c_N) \cdot \Delta v(t-N)\}$$

$$+ K \cdot F_g \cdot \{R1 e(t) - e(t-1)\}$$

We would now like to manipulate the model equation to be in terms of $\Delta v(t-1)$ and then ensure that the correct coefficients are passed between model and controller, i.e.

$$G_f G_p = \frac{Y(z)}{U(z)} \qquad G_f = \frac{V(z)}{U(z)}$$

Therefore $G_p = \dfrac{G_f G_p}{G_f} = \dfrac{Y(z)}{U(z)} \times \dfrac{U(z)}{V(z)} = \dfrac{Y(z)}{V(z)}$

i.e. $G_p = \{(\frac{1}{K} - c_1)z^{-1} + \ldots(c_{N-1} - c_N)z^{-N}\}$

$$\times \frac{(R2 - z^{-1})}{(R1 - z^{-1})\,F_g} = \frac{Y(z)}{V(z)}$$

cross multiplying and solving for $Y(z)$

$$Y(z) = \frac{1}{R1} \cdot \{\frac{1}{F_g} \cdot \{R2(\frac{1}{K} - c_1)z^{-1}$$

$$+ \; R2(c_1 - c_2)z^{-2} + (c_1 - 1/K)z^{-2} \ldots$$

$$+ \; R2(c_{N-1} - c_N)z^{-N} + (c_{N-1} - c_{N-2})z^{-N}\}V(z)$$

$$+ \; Y(z)z^{-1}\}$$

By taking $(1 - z^{-1})$ out of both sides of the above equation we can have the model in terms of $Y(z)$ and $V(z)$ without altering the format.

The above equations are unwieldy in terms of the coefficients.

It makes sense to rewrite the model as:-

$$\Delta \hat{y}(t) = \frac{1}{R1} \cdot \{\frac{1}{F_g} \cdot \{\hat{L}_1 \cdot \Delta v(t-1) + \hat{L}_2 \; \Delta v(t-2) \ldots + \hat{L}_N \; \Delta v(t-N)\} + \Delta y(t-1)\} \quad \ldots \textbf{(a)}$$

where $\hat{c}_1 = \frac{1}{K} - \frac{\hat{L}_1}{R2}$ such that $\hat{L}_i$ = estimated coefficient

$$\hat{c}_N = \hat{c}_{N-1} + \frac{(\hat{c}_{N-1} - \hat{c}_{N-2} - \hat{L}_N)}{R2} \quad \ldots \textbf{(a1)}$$

and the controller

$$\Delta v(t) = \frac{K}{R2} \{Q_1 \cdot \Delta v(t-1) + Q_2 \cdot \Delta v(t-2) + \ldots + Q_N \cdot \Delta v(t-N)\}$$

$$+ K F_g \{R1 e(t) - e(t-1)\} \quad \ldots \textbf{(b)}$$

where $Q_N = \hat{c}_{N-1} - R2 c_N \quad \ldots$ **(b1)**

Now, in order to transfer from $\hat{L}_i$ to the controller coefficients Q_i, substitute equation **(a1)** into **(b1)**

$$Q_N = c_{N-1} - R2 \{c_{N-1} + \frac{(c_{N-1} - c_{N-2} - \hat{L}_N)}{R2}\}$$

$$= c_{N-2} + \hat{L}_N - R2 c_{N-1}$$

which from equation **(b1)** with N - 1 replacing N

$Q_N = \hat{L}_N + Q_{N-1}$ which is an easy transfer from model to controller.

Note also that $\sum_{i=1}^{N} \hat{L}_i$

$$= \sum_{i=1}^{N} R2(\hat{c}_{i-1} - \hat{c}_i) + (\hat{c}_{i-1} - \hat{c}_{i-2})$$

$$= R2(\frac{1}{\hat{K}} - \hat{c}_1)$$

$$+ R2(\hat{c}_1 - \hat{c}_2) + (\hat{c}_1 - \frac{1}{\hat{K}})$$

$$+ R2(\hat{c}_2 - \hat{c}_3) + (\hat{c}_2 - \hat{c}_1) \text{ etc.} \qquad \hat{c}_{N+1} \text{ negligable}$$

i.e. all cancel out except

$$\sum_{i=1}^{N} \hat{L}_i = \frac{R2 \times 1}{\hat{K}} - \frac{1}{\hat{K}}$$

or

$$\frac{1}{\hat{K}} = \frac{\hat{L}_i}{R2 - 1} \longrightarrow$$

which allows the estimated steady state gain to be calculated from the coefficients $\hat{L}_i$.

The resulting sequence of events to be carried out, using these equations, have been set out in Chapter 3.

APPENDIX **SECTION B**

Controller Software Listing

The following is a listing of the PLM and Assembler routines used to implement the self tuning controller. Each module is numbered and a brief description of the module is given.

A Memory Map and Symbol Table is also supplied. Many of the symbols used are self defining, e.g. SET POINT, CONTVARO (Controlled Variable), SETERRORO (Set Point Error).

Some of the symbols are used as in the text, e.g. the array L (Process Coefficients), GAMMA (Gain Factor), R1, R2 (Filter Parameters). All symbols with a suffix of 0, e.g. MANIPVARO (Manipulated Variable) denote the most Recent Value of the Variable. A suffix of 1 denotes the immediate Past Value of the Variable. a prefix of D, e.g. the array DV (Manipulated Variable) denotes an incremental variable. All flags are named as such with a qualifier. All symbols addressed between 0400H and 1FB8H are merely module names.

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SELFTUNECONTROLMODULE
OBJECT MODULE PLACED IN :F4:MOD1.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD1.SRC


               /*************************************************************/
               /*                                                           */
               /*                MODULE 1                                   */
               /*                                                           */
               /*      A SELF TUNING CONTROL PROGRAM                        */
               /*                                                           */
               /*      THIS MAIN MODULE INITIALIZES THE CONTROLLER          */
               /*      AND SETS UP A CONTINUAL LOOP WHICH CHECKS         */
               /*      THE FLAG WORD AND HENCE CALLS A MODE PROCEDURE       */
               /*      THESE ARE AUTOMATIC MODE, MANUAL MODE  OR            */
               /*      MONITOR ENTRY                                        */
               /*                                                           */
               /*************************************************************/
               /*

                                       */
  1             SELF$TUNE$CONTROL$MODULE:
                  DO;

  2    1          MANUAL$MODE:
                    PROCEDURE EXTERNAL;
  3    2          END MANUAL$MODE;

  4    1          AUTOMATIC$MODE:
                    PROCEDURE EXTERNAL;
  5    2          END AUTOMATIC$MODE;


  6    1          ASC$OUT:
                    PROCEDURE (THING) EXTERNAL;
  7    2            DECLARE THING BYTE;
  8    2          END ASC$OUT;

  9    1              MONINS:
                            PROCEDURE EXTERNAL;
 10    2              END MONINS;

 11    1              FSET:
                      PROCEDURE(FA,OP1,OP2) EXTERNAL;
 12    2              DECLARE(FA,OP1,OP2) ADDRESS;
 13    2              END FSET;

 14    1              FSUB:
                  PROCEDURE(FA,OA) EXTERNAL;
 15    2          DECLARE (FA,OA) ADDRESS;
 16    2          END FSUB;


 17    1           FADD:
```

```
                    PROCEDURE(FA,OA) EXTERNAL;
 18   2              DECLARE (FA,OA) ADDRESS;
 19   2              END FADD;

 20   1              FQFB2D:
                    PROCEDURE(FA,OA) EXTERNAL;
 21   2              DECLARE (FA,OA) ADDRESS;
 22   2              END FQFB2D;


 23   1              FDIV:
                    PROCEDURE(FA,OA) EXTERNAL;
 24   2              DECLARE (FA,OA) ADDRESS;
 25   2              END FDIV;

 26   1             FMUL:
                    PROCEDURE(FA,OA) EXTERNAL;
 27   2              DECLARE (FA,OA) ADDRESS;
 28   2             END FMUL;

 29   1             FLOAD:
                    PROCEDURE(FA,OA) EXTERNAL;
 30   2              DECLARE (FA,OA) ADDRESS;
 31   2             END FLOAD;

 32   1             FSTOR:
                    PROCEDURE(FA,OA) EXTERNAL;
 33   2              DECLARE (FA,OA) ADDRESS;
 34   2             END FSTOR;

 35   1              FCLR:
                    PROCEDURE (FA) EXTERNAL;
 36   2              DECLARE FA ADDRESS;
 37   2               END FCLR;

 38   1              FNEG:
                    PROCEDURE (FA) EXTERNAL;
 39   2              DECLARE FA ADDRESS;
 40   2              END FNEG;

 41   1              FLTDS:
                        PROCEDURE (FA,OA) EXTERNAL;
 42   2             DECLARE (FA,OA) ADDRESS;
 43   2              END FLTDS;

 44   1              FIXSD:
                    PROCEDURE (FA,OA) EXTERNAL;
 45   2             DECLARE (FA,OA) ADDRESS;
 46   2              END FIXSD;

 47   1              INTERRUPT$PROCESSOR:
                        PROCEDURE EXTERNAL;
 48   2              END INTERRUPT$PROCESSOR;

 49   1              DECLARE FPR(18D) BYTE PUBLIC;

 50   1                      DECLARE MIDDLEMAN(4D) BYTE PUBLIC;
```

```
 51   1          DECLARE FIRST$ERROR$FLAG BYTE PUBLIC;
 52   1          DECLARE FOREVER LITERALLY 'WHILE 1';
 53   1          DECLARE FLAG$WORD BYTE PUBLIC;
 54   1      DECLARE (MANUAL$FLAG,INTERROGATE$FLAG) BYTE
             PUBLIC;
 55   1          DECLARE SET$POINT(4D) BYTE PUBLIC;
 56   1          DECLARE FROM$PLANT(4D) BYTE PUBLIC;
 57   1      DECLARE TO$PLANT ADDRESS  PUBLIC;
 58   1          DECLARE CONTVAR0(4D) BYTE PUBLIC;
 59   1          DECLARE CONTVAR1(4D) BYTE PUBLIC;
 60   1          DECLARE MANIPVAR0(4D) BYTE PUBLIC;
 61   1          DECLARE MANIPVAR1(4D) BYTE PUBLIC;
 62   1          DECLARE DELTAV0(4D) BYTE PUBLIC;
 63   1          DECLARE DY0(4D) BYTE PUBLIC;
 64   1          DECLARE DY1(4D) BYTE PUBLIC;
 65   1          DECLARE (LOLEVEL,HILEVEL) ADDRESS PUBLIC;
 66   1          DECLARE (MAXLEVEL,MINLEVEL) ADDRESS PUBLIC;
 67   1          DECLARE (SAT,ERR) BYTE PUBLIC;
 68   1          DECLARE CONSTANT(4D) BYTE PUBLIC;
 69   1          DECLARE  C1(4D) BYTE;
 70   1          DECLARE SET$ERROR0(4D) BYTE PUBLIC;
 71   1          DECLARE SET$ERROR1(4D) BYTE PUBLIC;
 72   1          DECLARE INCREMENT(4D) BYTE PUBLIC;
 73   1          DECLARE L(200D) BYTE PUBLIC;
 74   1          DECLARE DV(200D) BYTE PUBLIC;
 75   1          DECLARE FILTGAIN(4D) BYTE PUBLIC;
 76   1          DECLARE R2(4D) BYTE PUBLIC;
 77   1          DECLARE R1(4D) BYTE PUBLIC;
 78   1          DECLARE ONE(4D) BYTE PUBLIC;
 79   1          DECLARE TLD(4D) BYTE PUBLIC;
 80   1      DECLARE TLG(4D) BYTE PUBLIC;
 81   1          DECLARE SAMPLING$PERIOD(4D) BYTE PUBLIC;
 82   1          DECLARE R2SUB1(4D) BYTE PUBLIC;
 83   1          DECLARE GAMMA(4D) BYTE PUBLIC;
 84   1          DECLARE HUNDRED(4D) BYTE PUBLIC;
 85   1          DECLARE PLANT$GAIN (4D) BYTE PUBLIC;
 86   1          DECLARE DYPRED(4D) BYTE PUBLIC;
 87   1          DECLARE ERRPRED(4D) BYTE PUBLIC;
 88   1      DECLARE SAMPLE$TIME BYTE PUBLIC;
 89   1          DECLARE MANUAL$COUNT BYTE PUBLIC;
 90   1          DECLARE SAMPLE$COUNT ADDRESS PUBLIC;
 91   1          DECLARE (RAMP$FLAG,ENABLE$MANUAL) BYTE PUBLIC;
 92   1          DECLARE END$COUNT ADDRESS PUBLIC;
 93   1      DECLARE SLM$L(4D) BYTE PUBLIC;
 94   1      DECLARE ARRAY$LENGTH BYTE PUBLIC;
 95   1          DECLARE PARMNO BYTE PUBLIC;
 96   1          DECLARE SPECIFY STRUCTURE(
              SIGN BYTE,
              SCALE ADDRESS,
              SLENGTH BYTE,
              STRING$PTR ADDRESS) PUBLIC;
 97   1      DECLARE DEC$STRING(10D) BYTE PUBLIC;
 98   1      DECLARE (TTY$FLAG,TTY$TIME,TTY$UP)
                  BYTE PUBLIC;
 99   1          DECLARE MESSAGE(*) BYTE DATA (
              'INITIALISATION COMPLETE');
100   1          DECLARE STATEMENT(*) BYTE DATA (
```

```
                                  'MONITOR ENTERED');
101   1              DECLARE I BYTE;
102   1                  DECLARE DAC1LO BYTE AT (7FF6H);
103   1              DECLARE DAC1HI BYTE AT (7FF7H);
104   1                  DECLARE DAC2LO BYTE AT (7FF4H);
105   1                  DECLARE DAC2HI BYTE AT (7FF5H);
106   1              DECLARE INITL BYTE AT (7FF0H);
                     /* CALLED SETUP IN RTI BOOK  */
107   1              DECLARE MUXADR BYTE AT (7FFAH);
108   1              DECLARE GAINSEL BYTE AT (7FF9H);
109   1                  DECLARE LOC$3C3D BYTE AT(3C3DH);
110   1                  DECLARE LOC$3C3E ADDRESS AT(3C3EH);
111   1                  DECLARE (A,B) ADDRESS PUBLIC;
112   1                  DECLARE CHANGE$TIME BYTE PUBLIC;

113   1           SELF$TUNE$CONTROL:
                   /********************************/

                          DO;

                  /* THE FOLLOWING SECTION OF PROGRAM INITIALIZES
                     THE CONTROLLER. ALL PORTS ARE DEFINED I.E.
                     AS INPUT OR OUTPUT.ALL VARIABLES NOT PRESET
                     ELSEWHERE ARE INITIALISED HERE.ALL SOFTWARE FLAGS
                     ARE INITIALISED HERE.

                         THIS SECTION OF PROGRAM IS RUN ONCE UPON A
                     HARDWARE RESET AND IS NOT CALLED AGAIN BY
                     ANY OTHER PROCEDURE.
                  */

114   2                      SETUP:;
                         /**********************************/

                  /* 1ST   TASK: INITIALISE PORTS
                  */

115   2                  DISABLE;  /* ALLOW NO INTERRUPTS */

116   2                  OUTPUT(0E7H)=8BH;

                  /* DEFINES  1)PORT E6H AS INPUT.
                                THESE ARE FRONT PANEL SWITCHES.

                             2)PORT E5H AS INPUT.
                                THESE ARE THUMBWHEEL SWITCHS.

                               3)PORT E4H AS OUTPUT.
                                THESE ARE LIGHT EMITTING DIODES
                                ON FRONT PANEL.
                       */

117   2           OUTPUT(0E4H) = 00H;
```

```
                  /* ALL LIGHTS OFF.    */

 118   2          SAT=00;
 119   2          ERR=00;

                  /* THESE TWO VARIABLES INDICATE:
                       1)SATURATION HAS OCCURED
                         (EITHER CONTROLLED OR
                         MANIPULATED VARIABLES)
                         AND VARIABLE 'SAT'
                         ENSURES CORRECT LED
                         LIGHTS UP.
                       2)PREDICTION ERROR IS
                         VISUALLY DISPLAYED
                         VIA 'ERR' DURING CONTROL
                                  TIME.
                         SAT AND ERR USE SAME
                         OUTPUT PORT.
                         BOTH ARE INITIALISED
                         TO ' LIGHTS OFF '
                         UPON INITIALISATION.
                  */

                      /* PLACE JMP INSTRUCTION AT LOCATION 3C3DH
                         TO INTERRUPT PROCESSOR MODULE             */

 120   2                 LOC$3C3D = 0C3H;
 121   2                 LOC$3C3E = .INTERRUPT$PROCESSOR;

                      /* NOW INITIALIZE RTI BOARD.
                         THIS ENTAILS INITIALIZING INTERRUPT
                         TIMING,CHANNEL SELECT,GAIN SELECT.
                         OUTPUTTING INITIAL VALUES.
                  */

 122   2              DAC1LO = 0FFH;
 123   2              DAC1HI = 07H;

                  /* INITIALIZE TO CONTROLLED VAR TO MIDRANGE
                  */

 124   2              DAC2LO = 00;
 125   2          DAC2HI = 00;

                      /* INITIALIZE CHART REC TO ZERO */

 126   2              MUXADR = 00; /* CHOOSE CHANNEL ZERO */

 127   2              GAINSEL = 00; /* AMPLIFIER AT UNITY GAIN *

 128   2              INITL = 02;

                      /* R-C PACER TRIGGERS INTRRUPT    */

                  /* NOW INITIALIZE TELETYPE .      */
```

```
                  /* ALL LIGHTS OFF.    */

 118    2         SAT=00;
 119    2         ERR=00;

                  /* THESE TWO VARIABLES INDICATE:
                      1)SATURATION HAS OCCURED
                        (EITHER CONTROLLED OR
                        MANIPULATED VARIABLES)
                        AND VARIABLE 'SAT'
                        ENSURES CORRECT LED
                        LIGHTS UP.
                      2)PREDICTION ERROR IS
                        VISUALLY DISPLAYED
                        VIA 'ERR' DURING CONTROL
                                TIME.
                        SAT AND ERR USE SAME
                        OUTPUT PORT.
                        BOTH ARE INITIALISED
                        TO ' LIGHTS OFF '
                        UPON INITIALISATION.
                  */

                      /* PLACE JMP INSTRUCTION AT LOCATION 3C3DH
                         TO INTERRUPT PROCESSOR MODULE              */

 120    2                LOC$3C3D = 0C3H;
 121    2                LOC$3C3E = .INTERRUPT$PROCESSOR;

                      /* NOW INITIALIZE RTI BOARD.
                         THIS ENTAILS INITIALIZING INTERRUPT
                         TIMING,CHANNEL SELECT,GAIN SELECT.
                         OUTPUTTING INITIAL VALUES.
                  */

 122    2             DAC1LO = 0FFH;
 123    2             DAC1HI = 07H;

                  /* INITIALIZE TO CONTROLLED VAR TO MIDRANGE
                  */

 124    2             DAC2LO = 00;
 125    2         DAC2HI = 00;

                      /* INITIALIZE CHART REC TO ZERO */

 126    2             MUXADR = 00; /* CHOOSE CHANNEL ZERO */

 127    2             GAINSEL = 00; /* AMPLIFIER AT UNITY GAIN */

 128    2             INITL = 02;

                       /* R-C PACER TRIGGERS INTRRUPT    */

                  /* NOW INITIALIZE TELETYPE .       */
```

```
129   2        OUTPUT(0EDH) = 0CFH;  /* MODE WORD */
130   2            OUTPUT(0EDH) = 25H;    /* COMMAND WORD */

                /* THE FOLLOWING VARIABLES ARE USED BY
               MODULE 7, I.E. EXTRA VARIABLE MODULE
               WHICH OUTPUTS A HISTORY OF A MODEL
               PARAMETER (DURING CONTROL TIME)
               TO THE TELETYPE.

                1)TTY$FLAG (HARDWARE) :-
                  IF <> 01 THEN MODEL PARAMETER
                  IS NOT OUTED TO TTY.
                2)TTY$TIME:-
                  A COUNTER TO ENABLE PRINTING
                          ONLY ONCE EVERY 'TTY$UP' SAMPLING
                      PERIODS.
                3)TTY$UP:-
                  AS EXPLAINED ABOVE.
           */

131   2            TTY$TIME = 01H; /* INITIALISE COUNTER TO 1 */
132   2        TTY$UP    = 00D; /* PRINT EVERY  SAMPLING
                        PERIOD IF TTY$FLAG ALLOWS
                    */

                /* 'PARMNO' DEFINES WHICH PARAMETER
                IS OUTPUTTED TO TTY DURING RUN
                TIME.
           */

133   2        PARMNO = 10D; /* TENTH PARAMETER */

                /* THE ABOVE MENTIONED PARAMETER
               IS CONVERTED FROM BINARY TO
               DECIMAL FLOATING PT FORMAT(ASCII)
               PRECISION FOR CONVERSION IS DEFINED
               BY SPECIFY.SLENGTH.
           */

134   2        SPECIFY.SLENGTH = 5D; /* 5 DECIMAL DIGITS */

                /* NOW VALUES ACTUALLY RELATED TO IDENTIFICATION
               ARE INITIALISED.
           */

                /* [illegible] MODE CONSTANTS ARE NOW
                  INITIALIZED. A & B DEFINE RATE OF
                  [illegible]
                  CHANGE [illegible] WHICH ONE
                  (A OR B) IS ACTIVE AT ANY ONE TIME. */

[illegible]             [illegible] 10D;
[illegible]             [illegible] 3D;
[illegible]             [illegible]

                [illegible]
```

```
                          SECONDS.
                           8 S          4.            D     */
138   2            VARS:
                   /*********/

                    ARRAY$LENGTH = 4   ;
                    /* 40 MODEL PARAMETERS WOULD B    D */
139   2             LOLEVEL = 24D;
140   2             HILEVEL = 4076D;

                    /* CONTROLLED VARIABLE
                       SATURATED BEYOND THESE    S.
                    */

141   2             MINLEVEL = 24D;
142   2             MAXLEVEL = 4076D;

                    /* SAME FOR MANIPULATED V       */

                       /* INITIALIZE
                       UNCERTAINTY AND CONTROL
                          FIRST
                       UP TO CERTAIN PRESET VALUES */

143   2                MANIPVAR1(0) =    ;
144   2                MANIPVAR1(1) = 07  ;
145   2                MANIPVAR1(2) = 0   ;
146   2                MANIPVAR1(3) = 0   ;

147   2            DO I = 0 TO (ARRAY$LENGTH-1);
148   3               L(I) = 00;
149   3               DV(I) = 00;
150   3            END;

151   2                CALL FSET(.FPR,0,0);
152   2                CALL FCLR(.FPR);
153   2                CALL FSTOR(.FPR,.TLD);
154   2            CALL FSTOR(.FPR,.TLG);
155   2                CALL FSTOR(.FPR,.CONTVAR1);
156   2            CALL FSTOR(.FPR,.ONE);
157   2            CALL FSTOR(.FPR,.SAMPLING$PERIOD);
158   2                CALL FSTOR(.FPR,.GAMMA);
159   2            CALL FSTOR(.FPR,.HUNDRED);
160   2                CALL FSTOR(.FPR,.DELTAVO);
161   2                CALL FSTOR(.FPR,.DY1);
162   2                CALL FSTOR(.FPR,.SET$ERROR1);

163   2            DO I = 0 TO 4D*(ARRAY$LENGTH-1) BY 4D;
164   3               L(I) = 01H;
165   3                  DV(I) = 01H;
166   3               END;

                    /* INITIALIZE ARBITRARILY
```

```
                              L(I) =.01D
                              DV(I) = .01D
                              (DIVISION BY 1000 LATER ON)
                           */

 167   2                   TLD(0) = 108D;
 168   2              TLG(0) = 108D;

                           /* FILTER = (180S + 1)/(180S+1) */
                           /* IE CANCEL UNLESS CHANGED BY OPERATOR */

 169   2              SAMPLING$PERIOD(00) = 54D;

                           /* ASSUMING 2 MIN. TIME CONST. */

 170   2                   SAMPLE$TIME = 00;

                           /* FLAG OFF. DO NOT SAMPLE PLANT
                              UNTIL FLAG TURNED ON BY REAL
                              TIME CLOCK.
                           */

 171   2                   SAMPLE$COUNT = 00H;
 172   2                   MANUAL$COUNT = 00H;

                           /* SAMPLE$COUNT COUNTS INTERRUPTS
                              TO KEEP TRACK OF REAL TIME.
                              MANUAL$COUNT COUNTS INTERRUPTS
                              TO ENABLE CHANGE OF OUTPUT
                              SENSITIVITY DURING MANUAL MODE */

 173   2              ONE(0) = 1D;

                      /* THE CONSTANT ONE */

 174   2              GAMMA(0) = 0A0H;

                      /* UPDATE WEIGHTING FACTOR */
                           /* STILL TO BE DIVIDED BY 10**8D */

 175   2              HUNDRED(0) = 100D;

                           /* FACTOR = 100 */

 176   2                   C1(0) = 0FFH;
 177   2              C1(1) = 0FH;
 178   2                   C1(2) = 00H;
 179   2                   C1(3) = 00H;

                           /* USED TO GET CONSTANT 40,96 */

 180   2                   TO$PLANT = 07FFH;

                           /* PREVIOUS OUTPUT TO PLANT AS
                              USED BY ANALOGLE OUT MODULE  */
```

```
181        DO I=00 TO LAST(STATEMENT);
182           CALL ASC$OUT(STATEMENT(I));
183        END;
184           CALL ASC$OUT(0DH);
185           CALL ASC$OUT(0AH);

           /* INFORM OPERATOR OF MONITOR ENTRY */

            *::::::::::::::::::::::::*/
186        CALL MONINS;
            *::::::::::::::::::::::::*/

            * AT THIS POINT A JUMP TO MONITOR IS MADE
              TO ENABLE VARITIONS OF INITIAL
              CONDITIONS TO BE MADE.
            *

            * NOW DO ALL NECESSARY CALCS */

187        END$COUNT = (DOUBLE(SAMPLING$PERIOD(0))*1000D)/40D ;

            * NO. OF INTERRUPTS TO BE COUNTED BEFORE
              SAMPLING PLANT.
              40MS PER INTERRUPT ASSUMED HERE.
            *

            * 1) CONVERT ALL INTEGERS TO FLOAT PT */

188        CALL FLTDS(.FPR,.HUNDRED);
189        CALL FSTOR(.FPR,.HUNDRED);

190        DO I = 0 TO 4D*(ARRAY$LENGTH-1D) BY 4D:
191              CALL FLTDS(.FPR,.L(I));
192              CALL FDIV(.FPR,.HUNDRED);
193              CALL FSTOR(.FPR,.L(I));
194              CALL FLTDS(.FPR,.DV(I));
195              CALL FDIV(.FPR,.HUNDRED):
196         CALL FS OR(.FPR,.DV(I)):
197    END·


198    CALL FLTDS(.FPR,.C1);
199    CALL FSTOR(.FPR,.C1);
200        CALL FLTDS(.FPR,.DELTAVO);
201        CALL FSTOR(.FPR,.DELTAVO);
202    CALL FLTDS(.FPR,.GAMMA);
203        CALL FDIV(.FPR,.HUNDRED);
204        CALL FDIV(.FPR .HUNDRED);
205        CALL FDIV(.FPR,.HUNDRED);
206        CALL FDIV(.FPR,.HUNDRED);
207        CALL FSTOR(.FPR,.GAMMA);  /* DIVIDE BY 10**8 D */
208    CALL FLTDS(.FPR,.ONE);
209        CALL FSTOR(.FPR,.ONE);
210        CALL FLTDS(.FPR,.TLD);
211        CALL FSTOR(.FPR,.TLD);
212        CALL FLTDS(.FPR,.TLG);
```

```
213   2          CALL FSTOR(.FPR,.TLG);
214   2          CALL FLTDS(.FPR,.SAMPLING$PERIOD);
215   2              CALL FSTOR(.FPR,.SAMPLING$PERIOD);
216   2              CALL FDIV(.FPR,.TLG);
217   2              CALL FSTOR(.FPR,.R2SUB1);

                 /* CALCULATE FILTER CONSTANTS

                     R1=(SAMPLING$PERIOD/TLD) + 1

                     R2=(SAMPLING$PERIOD/TLG) + 1

                     FILTGAIN = TLD/TLG

                     R2SUB1 = R2 - 1

                 */

218   2          CALL FADD(.FPR,.ONE);
219   2          CALL FSTOR(.FPR,.R2);   /* R2 CALC */
220   2              CALL FLTDS(.FPR,.MANIPVAR1);
221   2              CALL FSTOR(.FPR,.MANIPVAR1);

222   2              CALL FLTDS(.FPR,.CONTVAR1);
223   2              CALL FSTOR(.FPR,.CONTVAR1);

224   2          CALL FLOAD(.FPR,.SAMPLING$PERIOD);
225   2          CALL FDIV(.FPR,.TLD);
226   2          CALL FADD(.FPR,.ONE);
227   2          CALL FSTOR(.FPR,.R1);   /* R1 CALC */

228   2          CALL FLOAD(.FPR,.TLD);
229   2          CALL FDIV(.FPR,.TLG);
230   2          CALL FSTOR(.FPR,.FILTGAIN); /* FILTGAIN */


231   2          CALL FLOAD(.FPR,.C1);
232   2          CALL FDIV(.FPR,.HUNDRED);
233   2          CALL FSTOR(.FPR,.CONSTANT); /* = 40.96 */
234   2              FIRST$ERROR$FLAG = 01H;

                     /* SIGNALS EXTRA VAR MODULE
                        TO TAKE FIRST ERROR AS REFERENCE */

                 /* NOW PRINT MESSAGE TO TTY */

235   2          DO I = 0 TO LAST(MESSAGE);
236   3          CALL ASC$OUT(MESSAGE(I));
237   3          END;

238   2              CALL ASC$OUT(0DH);
239   2              CALL ASC$OUT(0AH);

240   2              ENABLE; /* INTERRUPTS */
```

```
 241   2           BEGIN$LOOP:
                     /*****************************/

                             DO FOREVER;

                   /* SINCE THE INTERRUPT MODULE CONTROLS OUTPUT
                      DURING MANUAL OPERATION . A DANGER EXISTS
                      THAT WHILE IN THE MIDDLE OF AUTO MODULE
                      AN OPERATOR MIGHT REQUEST MANUAL OPERATION.
                      THE INTERRUPT MODULE SENSING CHANGE IN
                      HARDWARE FLAG WOULD OUTPUT TO PLANT EVERY
                      INTERRUPT.AT THE SAME TIME AUTO MODULE
                      WOULD ALSO OUTPUT TO PLANT.
                      THEREFORE TO ENSURE MANUAL OUTPUT ONLY
                      WHEN MANUAL MODULE IS ENTERED, A FLAG
                      CALLED ENABLE$MANUAL IS SET WHENEVER
                      MANUAL MODULE IS ENTERED. THE FLAG IS RESET
                      UPON A CALL TO AUTO OR TO INTERROGATE
                   */

                   /*   READ IN FLAG WORD AND ISOLATE REQUIRED
                           FLAG BITS              */

 242   3                 FLAG$WORD=INPUT(0E6H);
 243   3                 MANUAL$FLAG=SHR(FLAG$WORD,1) AND 01H;
 244   3                 INTERROGATE$FLAG=FLAG$WORD AND 01H;
 245   3                        TTY$FLAG = SHR(FLAG$WORD,2) AND 01H;
 246   3                DISABLE;

 247   3                 IF INTERROGATE$FLAG  =01H THEN
 248   3         DO;
 249   4           FIRST$ERROR$FLAG = 01H;
 250   4               ENABLE$MANUAL = 00;

 251   4               DO I=00H TO LAST(STATEMENT);
 252   5                  CALL ASC$OUT(STATEMENT(I));
 253   5               END;
 254   4                  CALL ASC$OUT(0DH);
 255   4                  CALL ASC$OUT(0AH);


 256   4              CALL MONINS  ;
                      /* CALL MONITOR
                      */

 257   4            END;

                   /* IF AN INTERROGATION IS REQUESTED  */
                      /* OR A HARDWARE RESTART OCCURS THE
                         FIRST ERROR FLAG SIGNALS TO THE
                         LEDS ROUTINE TO TAKE A NEW
                         REFERENCE FOR PREDICTION    */

 258   3           ENABLE;
```

```
 259   3              IF MANUAL$FLAG  =01H  THEN
 260   3                         DO;
 261   4                           ENABLE$MANUAL = 01H;
 262   4                     CALL MANUAL$MODE;
 263   4                         END;

                     ELSE
 264   3                         DO;
 265   4                             ENABLE$MANUAL = 00H;
 266   4                             CALL AUTOMATIC$MODE;
 267   4                         END;

 268   3         END$LOOP:
                         END; /* THE FOREVER */
                 /*********************************************/

 269   2       END SELF$TUNE$CONTROL;
 270   1      END SELF$TUNE$CONTROL$MODULE;
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 04A5H   1189D
     VARIABLE AREA SIZE = 0249H    585D
     MAXIMUM STACK SIZE = 0004H      4D
     633 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE AUTOMATICMODEMODULE
OBJECT MODULE PLACED IN :F4:MOD2.OBJ
COMPILER INVOKED BY: PLM80 :F4:MOD2.SRC

```
              /*****************************************************/
              /*                                                   */
              /*                 MODULE 2                          */
              /*                                                   */
              /*    THIS SECONDARY LEVEL MODULE DEFINES            */
              /*    THE SELF TUNING CONTROLLER AUTOMATIC           */
              /*    MODE.                                          */
              /*    IF SAMPLING PERIOD HAS PASSED, THE             */
              /*    ROUTINE CHECKS FOR SATURATED CONTROLLED        */
              /*    VARIABLE. IF NO SATURATION, PREDICTION         */
              /*    UPDATING AND CONTROLLER ACTION TAKES PLACE.    */
              /*    IF SATURATED, A WARNING LIGHT IS SET.          */
              /*    PROGRAM FLOW IS THEN RETURNED TO THE PRIMARY   */
              /*    ROUTINE IN MODULE 1.                           */
              /*                                                   */
              /*****************************************************/
              /*

              */
  1             AUTOMATIC$MODE$MODULE:
                     DO:

  2    1            DECLARE FROM$PLANT(4D)   BYTE    EXTERNAL;
                    /* RETAINED AS INTEGER THROUGHOUT */
  3    1            DECLARE (LOLEVEL,HILEVEL) ADDRESS EXTERNAL;
  4    1            DECLARE SAMPLE$TIME BYTE EXTERNAL;
  5    1            DECLARE CONTVAR0(4D) BYTE    EXTERNAL;
  6    1            DECLARE CONTVAR1(4D) BYTE    EXTERNAL;
  7    1                   DECLARE MANIPVAR0(4D) BYTE EXTERNAL;
  8    1                   DECLARE MANIPVAR1(4D) BYTE EXTERNAL;
  9    1                   DECLARE TO$PLANT ADDRESS EXTERNAL;
 10    1            DECLARE SET$ERROR0(4D) BYTE    EXTERNAL;
 11    1            DECLARE DV(200D) BYTE EXTERNAL;
 12    1            DECLARE DY0(4D) BYTE EXTERNAL;
 13    1            DECLARE CONSTANT(4D) BYTE EXTERNAL;
 14    1            DECLARE INCREMENT(4D) BYTE EXTERNAL;
 15    1            DECLARE (SAT,ERR) BYTE EXTERNAL;
 16    1                   DECLARE SET$POINT(4D) BYTE EXTERNAL;


 17    1            INPUT$CONTROLLED$VARIABLE:
                    PROCEDURE EXTERNAL;
 18    2            END INPUT$CONTROLLED$VARIABLE;

 19    1            PREDICT$AND$UPDATE:
                    PROCEDURE EXTERNAL;
 20    2            END PREDICT$AND$UPDATE;

 21    1            MANIP$VARIABLE$CALC:
```

```
                        PROCEDURE EXTERNAL;
 22   2                 END MANIP$VARIABLE$CALC;

 23   1                 ANALOGUE$OUT:
                        PROCEDURE EXTERNAL;
 24   2                        END  ANALOGUE$OUT;

 25   1                  EXTRA$VARIABLE:
                        PROCEDURE EXTERNAL;
 26   2                 END EXTRA$VARIABLE;

 27   1                 DATA$SHUFFLE:
                        PROCEDURE EXTERNAL;
 28   2                 END DATA$SHUFFLE;

            $INCLUDE(:F4:FLOAT.SRC)
          =
 29   1   =          FSET:
          =          PROCEDURE(FA,OP1,OP2) EXTERNAL;
 30   2   =          DECLARE(FA,OP1,OP2) ADDRESS;
 31   2   =          END FSET;
          =
 32   1   =          FSUB:
          =       PROCEDURE(FA,OA) EXTERNAL;
 33   2   =       DECLARE (FA,OA) ADDRESS;
 34   2   =       END FSUB;
          =
          =
 35   1   =        FADD:
          =       PROCEDURE(FA,OA) EXTERNAL;
 36   2   =        DECLARE (FA,OA) ADDRESS;
 37   2   =        END FADD;
          =
 38   1   =        FDIV:
          =       PROCEDURE(FA,OA) EXTERNAL;
 39   2   =        DECLARE (FA,OA) ADDRESS;
 40   2   =        END FDIV;
          =
 41   1   =       FMUL:
          =       PROCEDURE(FA,OA) EXTERNAL;
 42   2   =        DECLARE (FA,OA) ADDRESS;
 43   2   =       END FMUL;
          =
 44   1   =       FLOAD:
          =       PROCEDURE(FA,OA) EXTERNAL;
 45   2   =        DECLARE (FA,OA) ADDRESS;
 46   2   =       END FLOAD;
          =
 47   1   =         FCLR:
          =             PROCEDURE(FA) EXTERNAL;
 48   2   =        DECLARE FA ADDRESS;
 49   2   =         END FCLR;
          =
 50   1   =        FNEG:
          =       PROCEDURE (FA) EXTERNAL;
 51   2   =        DECLARE FA ADDRESS;
 52   2   =        END FNEG;
```

```
              =
  53   1      =      FLTDS:
              =         PROCEDURE (FA,OA) EXTERNAL;
  54   2      =     DECLARE (FA,OA) ADDRESS;
  55   2      =      END FLTDS;
              =
  56   1      =      FIXSD:
              =     PROCEDURE (FA,OA) EXTERNAL;
  57   2      =     DECLARE (FA,OA) ADDRESS;
  58   2      =      END FIXSD;
              =
  59   1      =      FSTOR:
              =         PROCEDURE(FA,OA) EXTERNAL;
  60   2      =         DECLARE (FA,OA) ADDRESS;
  61   2      =      END FSTOR;
              =
  62   1      =      DECLARE FPR(18D) BYTE EXTERNAL;


                    /*   MODULE 2 - MAIN  PROGRAM   */

  63   1            AUTOMATIC$MODE:
                      PROCEDURE PUBLIC;

  64   2            DECLARE (TENS,UNITS) BYTE;
  65   2                DECLARE FP ADDRESS;
  66   2                DECLARE I BYTE;

  67   2                IF SAMPLE$TIME = 01H THEN

  68   2                    DO;
  69   3                        SAMPLE$TIME = 00H;
  70   3                                CALL FCLR(.FPR);
  71   3                        CALL FSTOR(.FPR,.SET$POINT);
  72   3                        CALL FSTOR(.FPR,.FROM$PLANT);

                     /* CLEAR AREA FOR SETPOINT AND CONROLLED VARIABLE
                        STORAGE */
                     /* TWO FLOATING POINT VARIABLES OF CONTROLLED
                        VARIABLE ARE KEPT IE. CONTVAR0 (T=0),
                                      CONTVAR1 (T=-1 SAMPLE TIME)
                        DO NOT CONFUSE WITH DY0 AND DY1 WHICH ARE VELOCITY
                        VERSIONS OF ABOVE(FLOATING), OR VARIABLE CALLED
                        FROM$PLANT = INTEGER VALUE OF CONTVAR0
                     */
  73   3                      CALL INPUT$CONTROLLED$VARIABLE;

  74   3                          CALL FLTDS(.FPR,.FROM$PLANT);
  75   3                      CALL FSTOR(.FPR,.CONTVAR0);
                     /* CONVERT MEASURED VALUE TO FLOAT POINT */

  76   3                      CALL FSUB(.FPR,.CONTVAR1);
  77   3                      CALL FSTOR(.FPR,.DY0);
                     /* GENERATE VELOCITY VARIABLE DY0 =Y(T)-Y(T-1)
                     */

                     /*  CHECK FOR CONTROLLED VARIABLE SATURATION */
```

```
                /* FROM$PLANT IS 32 BIT INTEGER AND NOT
                OPERABLE BY PLM THEREFORE CONVERT TO
                16 BIT INTEGER. ( VALUE NEVER EXCEEDS
                4096 UNITS ANYWAY AND IS NEVER NEGATIVE
            */

 78   3         FP=DOUBLE(FROM$PLANT(0)) OR SHL(DOUBLE(FROM$PLANT(1)),8)

 79   3         IF (FP > LOLEVEL) AND (FP < HILEVEL)
                   THEN

 80   3              DO;
                   /* RESET WARNING LIGHT */
 81   4            CALL PREDICT$AND$UPDATE;

 82   4                SAT=00;

            /* 2 SETS OF  LIGHTS ARE AVAILABLE
                   1) SATURATION HAS OCCURRED.
                   2) INDICATION OF PREDICTION ERROR.
                  1) LIGHTS A SINGLE L.E.D.   (ALARM).
                   (RAISED ON FRONT PANEL)
               2) LIGHTS SIX IN LINE.

                  THEREFORE 2 BYTE VARIABLES KEEP TRACK OF
                THESE 2 FUNCTIONS. I.E SAT(URATION)
                AND ERR (PREDICTION ERROR)
                */


 83   4              END;

                 ELSE

            /* WARNING LIGHT THAT PREDICTION SKIPPED */

 84   3         DO;
 85   4          SAT=80H;
 86   4         END;

                  OUTPUT(0E4H)=SAT OR ERR;
 87   3

            /* READ IN SETPOINT FROM THUMBWHEEL SWITCH
            AND CONVERT TO FLOAT POINT
            */
 88   3          TENS = SHR((NOT( INPUT(0E5H))),4D);
 89   3          UNITS = NOT( INPUT(0E5H) ) AND 0FH;

            /* READ IN PERCENTAGE */

 90   3          SET$POINT(0)=(UNITS+10D*TENS);

 91   3          CALL FLTDS(.FPR,.SET$POINT);
 92   3          CALL FMUL(.FPR,.CONSTANT);
```

```
 93   3              CALL FSUB(.FPR,.CONTVARO);
 94   3              CALL FSTOR(.FPR,.SET$ERRORO);


          /* CONVERT FROM 100%  TO  4096 UNITS */


             CALL MANIP$VARIABLE$CALC;
 95   3   /* CALCULATES IN VELOCITY FORM                        */

                      CALL FLOAD(.FPR,.DV(0));
 96   3
          /* XFORM MANIPULATED VARIABLE TO INTEGER FORM */

             CALL FIXSD(.FPR,.INCREMENT);
 97   3
           IF (INCREMENT(3) AND 80H) = 80H THEN
 98   3        DO;
 99   3          INCREMENT(3) = 80H;
100   4           DO I = 0 TO 2;
101   4             INCREMENT(I)=NOT(INCREMENT(I,));
102   5           END;
103   5         INCREMENT(0) = INCREMENT(0)+1;
104   4      IF INCREMENT(0) = 00H THEN
105   4          DO;
106   4         INCREMENT(1) = INCREMENT(1) +1;
107   5      IF INCREMENT(1) = 00H THEN
108   5         INCREMENT(2) = INCREMENT(2)+1;
109   5          END;
110   5         END;
111   4

            /* CONVERT FROM 2S COMPLEMENT TO
               SIGNED BINARY.
            */

                    CALL ANALOGUE$OUT;
112   3    /* ADDS DELTA V(t) + V(t-1) AND OUTPUTS    */
           /* TO PLANT VIA D/A No. 1                 */

              MANIPVARO(0) = LOW(TO$PLANT);
113   3       MANIPVARO(1) = HIGH(TO$PLANT);
114   3       MANIPVARO(2) = 00;
115   3       MANIPVARO(3) = 00;
116   3
              /* REGENERATE DV(0) IN CASE OF SATURATION */


              CALL FLTDS(.FPR,.MANIPVARO);
117   3       CALL FSTOR(.FPR,.MANIPVARO);
118   3       CALL FSUB(.FPR,.MANIPVAR1);
119   3       CALL FSTOR(.FPR,.DV);
120   3
                CALL EXTRA$VARIABLE;
121   3    /* OUTPUTS TWO EXTRA VARIABLES                */
           /*      1) TO TELETYPE                         */
           /*      2) TO CHART RECORDER                   */
```

```
 122   3               CALL DATA$SHUFFLE;
                 /* THIS PREPARES DATA FOR NEXT ITERATION */

 123   3           END;

 124   2         END AUTOMATIC$MODE;

 125   1       END AUTOMATIC$MODE$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 017CH    380D
     VARIABLE AREA SIZE = 0005H      5D
     MAXIMUM STACK SIZE = 0004H      4D
     287 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MANUALMODEMODULE
OBJECT MODULE PLACED IN :F4:MOD3.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD3.SRC DEBUG


                 /*********************************************/
                 /*                                           */
                 /*      MODULE 3                             */
                 /*                                           */
                 /*      MANUAL MODE                          */
                 /*                                           */
                 /*   THIS ROUTINE IS ACCESSED WHENEVER       */
                 /*   THE OPERATOR REQUESTS MANUAL OPERATION. */
                 /*   HERE,THE NECESSARY TIME FLAGS ARE       */
                 /*   CHECKED FOR PREDICTION AND UPDATE       */
                 /*   PURPOSES. ACTUAL MANUAL OUTPUT IS       */
                 /*   CONTROLLED FROM THE INTERRUPPT          */
                 /*   MODULE FOR EASE OF TIMING.              */
                 /*                                           */
                 /*********************************************/
                 /*

                 */
  1             MANUAL$MODE$MODULE:
                    DO;

  2    1                 DECLARE CONTVARO(4D) BYTE EXTERNAL;
  3    1                 DECLARE CONTVAR1(4D) BYTE EXTERNAL;
  4    1                 DECLARE DELTAVO(4D) BYTE EXTERNAL;
  5    1                 DECLARE DYO(4D) BYTE EXTERNAL;
  6    1             DECLARE SAMPLE$TIME BYTE EXTERNAL;
  7    1             DECLARE FROM$PLANT(4D) BYTE EXTERNAL;
  8    1             DECLARE ENABLE$MANUAL BYTE EXTERNAL;
  9    1             DECLARE (LOLEVEL,HILEVEL) ADDRESS EXTERNAL;
 10    1             DECLARE (SAT,ERR) BYTE EXTERNAL;
 11    1                 DECLARE DV(200D) BYTE EXTERNAL;

 12    1             INPUT$CONTROLLED$VARIABLE:
                        PROCEDURE EXTERNAL;
 13    2             END INPUT$CONTROLLED$VARIABLE;

            $INCLUDE(:F1:FLOAT.SRC)
                =
 14    1        =        FSET:
                =        PROCEDURE(FA,OP1,OP2) EXTERNAL;
 15    2        =        DECLARE(FA,OP1,OP2) ADDRESS;
 16    2        =        END FSET;
                =
 17    1        =        FSUB:
                =    PROCEDURE(FA,OA) EXTERNAL;
 18    2        =    DECLARE (FA,OA) ADDRESS;
 19    2        =    END FSUB;
                =
                =
 20    1        =      FADD:
                =    PROCEDURE(FA,OA) EXTERNAL;
```

```
21   2  =      DECLARE (FA,OA) ADDRESS;
22   2  =      END FADD;
        =
23   1  =      FOFB2D:
        =     PROCEDURE(FA,OA) EXTERNAL;
24   2  =      DECLARE (FA,OA) ADDRESS;
25   2  =      END FOFB2D;
        =
        =
26   1  =      FDIV:
        =     PROCEDURE(FA,OA) EXTERNAL;
27   2  =      DECLARE (FA,OA) ADDRESS;
28   2  =      END FDIV;
        =
29   1  =     FMUL:
        =     PROCEDURE(FA,OA) EXTERNAL;
30   2  =      DECLARE (FA,OA) ADDRESS;
31   2  =     END FMUL;
        =
32   1  =     FLOAD:
        =     PROCEDURE(FA,OA) EXTERNAL;
33   2  =      DECLARE (FA,OA) ADDRESS;
34   2  =     END FLOAD;
        =
35   1  =       FCLR:
        =           PROCEDURE(FA) EXTERNAL;
36   2  =      DECLARE FA ADDRESS;
37   2  =       END FCLR;
        =
38   1  =      FNEG:
        =     PROCEDURE (FA) EXTERNAL,
39   2  =      DECLARE FA ADDRESS;
40   2  =      END FNEG;
        =
41   1  =      FLTDS:
        =         PROCEDURE (FA,OA) EXTERNAL;
42   2  =     DECLARE (FA,OA) ADDRESS;
43   2  =      END FLTDS;
        =
44   1  =      FIXSD:
        =     PROCEDURE (FA,OA) EXTERNAL;
45   2  =     DECLARE (FA,OA) ADDRESS;
46   2  =      END FIXSD;
        =
47   1  =      FSTOR:
        =         PROCEDURE'FA,OA) EXTERNAL;
48   2  =         DECLARE (FA,OA) ADDRESS;
49   2  =      END FSTOR;
        =
50   1  =      DECLARE FPR(18D) BYTE EXTERNAL;

51   1         PREDICT$AND$UPDATE:
                  PROCEDURE EXTERNAL;
52   2         END PREDICT$AND$UPDATE;

53   1         EXTRA$VARIABLE:
                  PROCEDURE EXTERNAL;
```

```
 54             END EXTRA$VARIABLE:

 55   1         DATA$SHUFFLE:
                   PROCEDURE EXTERNAL:
 56   1         END DATA$SHUFFLE:

 57   1         MANUAL$MODE:
                   PROCEDURE PUBLIC;

 58   2            DECLARE FP ADDRESS;


 59   2         IF SAMPLE$TIME = 01H THEN

 60   2           DO;
 61   3             SAMPLE$TIME = 00; /* RESET FLAG */
 62   3             CALL FCLR(.FPR);
 63   3             CALL FSTOR(.FPR,.FROM$PLANT);
                    /* CLEAR FROM $PLANT */

                       /* SINCE THE INTERRUPT HANDLER OUTPUTS
                          TO PLANT DURING MANUAL OP,ONCE
                          EVERY INTERRUPT,WE NEED TO ADD
                          UP ALL THESE OUTPUTS PER SAMPLING
                          PERIOD TC PASS TO PREDICT AND UPDATE
                          MODULE.DELTAVO IS THIS SUMMATION.IT
                          IS CONVERTED TO FLOATING PT ONCE EVERY
                          SAMPLING PERIOD AND THEN CLEARED FOR
                          NEXT PERIOD.                              */
 64   3           DISABLE:
 65   3               CALL FLOAD(.FPR,.DELTAVO);
 66   3               CALL FSTOR(.FPR,.DV(0));
 67   3               CALL FCLR(.FPR);
 68   3               CALL FSTOR(.FPR,.DELTAVO);
 69   3            ENABLE:

 70   3             CALL INPUT$CONTROLLED$VARIABLE;

 71   3                 CALL FLTDS(.FPR,.FROM$PLANT);
 72   3             CALL FSTOR(.FPR,.CONTVARO);
 73   3                 CALL FSUB(.FPR,.CONTVAR1);
 74   3                 CALL FSTOR(.FPR,.DYO);

                         /* GENERATE VELOCITY VARIABLE
                    DYO = CONTROL VAR AT TIME 'T'
                    DY1=    CONTROL VAR AT 'T-1'
                     */

 75   3                 FP = DOUBLE(FROM$PLANT(0)) OR
                      SHL(DOUBLE(FROM$PLANT(1)),8);

                         /* USE INTEGER VALUE OF CONTROLLED VAR
                            TO CHECK FOR SATURATION.
                         BOTTOM 2 BYTES ARE ADEQUATE AS
                         A/D CONVERTER HAS ONLY 12 BITS PRECISION
                       */
```

```
 76   3                 IF (FP > LOLEVEL ) AND
                       (FP < HILEVEL ) THEN

 77   3                  DO;
 78   4                     SAT = 00;
 79   4                     CALL PREDICT$AND$UPDATE;
 80   4                      END;

                        ELSE
 81   3                  DO;
 82   4                    SAT = 80;
                           /* WARN OPERATOR VIA L.E.D
                              THAT PREDICT$AND$UPDATE SKIPPED
                                    */
 83   4                         END;

 84   3                     OUTPUT(0E4H) = (SAT OR ERR);

 85   3             CALL EXTRA$VARIABLE;
 86   3             CALL DATA$SHUFFLE;
 87   3              END;

 88   2            END MANUAL$MODE;
 89   1          END MANUAL$MODE$MODULE;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 00B6H    182D
     VARIABLE AREA SIZE = 0002H      2D
     MAXIMUM STACK SIZE = 0004H      4D
     190 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INTERRUPTMODULE
OBJECT MODULE PLACED IN :F4:MOD4.OBJ
COMPILER INVOKED BY: PLM80 :F4:MOD4.SRC

```
              /*******************************************************/
              /*                                                     */
              /*                      MODULE 4                       */
              /*                                                     */
              /*  THIS IS THE INTERRUPT MODULE.                      */
              /*  THERE IS ONLY A SINGLE INTERRUPT IN THE ENTIRE     */
              /*  SYSTEM. THIS HAS TWO FUNCTIONS:                    */
              /*                                                     */
              /*          1) A REAL TIME CLOCK.                      */
              /*          2) OUTPUT OF MANIPULATED VARIABLE          */
              /*                 DURING MANUAL OPERATION.            */
              /*                                                     */
              /*  AN INTERRUPT OCCURS EVERY 40mS.(APPROX)            */
              /*  IF CONDITIONS ARE FAVOURABLE (i.e. THE OPERATOR    */
              /*  HAS REQUESTED MANUAL OPERATION AND WE ARE NOT      */
              /*  INTERRUPTING THE AUTOMATIC MODE.) THEN             */
              /*  MANIPULATED VARIABLE IS EVALUATED AND OUTPUTTED    */
              /*  TO THE PLANT.                                      */
              /*  EVERY INTERRUPT A COUNTER IS UPDATED.UPON          */
              /*  REACHING A TERMINAL VALUE, A FLAG IS SET,          */
              /*  SIGNIFYING THAT A SAMPLING PERIOD HAS PASSED.      */
              /*  THIS FLAG ( SAMPLE$TIME ) PERMITS TIMED ENTRY      */
              /*  INTO THE PREDICT$AND$UPDATE & MANIPULATED$         */
              /*  VARIABLE$CALC ROUTINES.                            */
              /*                                                     */
              /*******************************************************
              /*
              */
  1           INTERRUPT$MODULE:
                DO;
  2   1           DECLARE MANUAL$COUNT
                               BYTE EXTERNAL;
  3   1           DECLARE SAMPLE$COUNT ADDRESS EXTERNAL;
  4   1               DECLARE END$COUNT ADDRESS EXTERNAL;
  5   1               DECLARE (SAT,ERR) BYTE EXTERNAL;
  6   1               DECLARE STATUS BYTE [illegible]
                /* SAMPLING INITIALLY DEFINED A[illegible]              */
                /* 1.5 SECONDS = 45 * 40MS                              */
  7   1           DECLARE (SAMPLE$TIME) [illegible]
  8   1           DECLARE (FLAG$WORD, [illegible]) BY
                                  EXTERNAL;
  9   1           D[illegible]NABLE[illegible],
                           BYTE [illegible]
 10   1           [illegible]
 11   1           [illegible]
 12   1           [illegible]
 13   1               [illegible]
 14   1               [illegible]
 15   1               DECLARE [illegible](40) [illegible]
 16   1               [illegible]
```

```
                    FAN0:
                     [illegible]        EXTERNAL;
                     [illegible]
                     [illegible]

                    FLTDS1:
                     PROCEDURE (FA,DA) EXTERNAL;
                     DECLARE (FA,DA) ADDRESS;
                     END [illegible];

                    FLTDS2:
                     PROCEDURE (FA,DA) EXTERNAL;
                     DECLARE (FA,DA) ADDRESS;
                    END;

                 [illegible]:
                     [illegible] EXTERNAL;
                 END [illegible];

                 [illegible]:
                   [illegible] EXTERNAL;
                 END [illegible];

              INTERRUPT$PROCEDURE:
 /***************************** PROCEDURE INTERRUPT 7 PUBLIC;

                 /* SENSITIVITY OF MANUAL OUTPUT DECREASED  */
                   /* AFTER 8 SECONDS                              */
                 /* 8 SECONDS = 200 PERIODS * 40MS/PERIOD     */

 31   2          DECLARE NUMBER BYTE;
 32   2            DECLARE I BYTE;

                 /* TEST WHETHER MANUAL OPERATION PERMISSABLE */

 33   2            FLAG$WORD=INPUT(0E8H);
 34   2            MANUAL$FLAG=SHR(FLAG$WORD,1) AND 01H;

 35   2        IF MANUAL$FLAG AND ENABLE$MANUAL
                   THEN
 36   2            DO;
                        /* MANUAL OPERATION */

 37   3                MANUAL$COUNT=MANUAL$COUNT + 01;

 38   3                IF MANUAL$COUNT > CHANGE$TIME

                                 THEN

                                 /* SET MOD OF OUTPUT */

 39   3               INCREMENT(0) = A;
                              ELSE
 40   3               INCREMENT(0) = B;
```

```
                  /* INCREMENT IS THE AMOUNT BY WHICH THE         */
                  /* MANIPULATED VARIABLE IS INCREASED OR         */
                  /* DECREASED EVERY 40ms DURING MANUAL           */
                     /* OPERATION. TWO RATES ARE APPLICABLE.         */
                     /* NAMELY A UNITS PER 40,ms OR B UNITS          */
                  /* PER 40ms.                                    */
                  /* NOTE THAT A > B .                            */

41   3                 DO;
42   4              INCREMENT(1) = 00;
43   4                  INCREMENT(2) = 00H;
44   4                  INCREMENT(3D) = 00;

                        /* MSB OF INCREMENT(3) IS SIGN BIT &
                           IS SPECIFIED LATER DEPENDING ON
                           HARDWARE RAMP FLAG AS SET BY OPERATOR
                       I.E.   RAMP UP OR RAMP DOWN
                        */

45   4                 END;

                  /* NOW THE RAMP$FLAG IS INPUT AND TESTED        */
                  /* FOR DIRECTION OF MANUAL OUTPUT.              */

46   3              RAMP$FLAG=SHR(INPUT(0E2H),3) AND 03H;

                  /* CASE 1: OPERATOR REQUIRES NO OPERATION       */
                  /* NOTE: OUTPUT SENSITIVITY IS INCREASED        */
                  /* FROM HERE ONWARDS BY RESETTING               */
                  /* MANUAL$COUNT.                                */

47   3              IF RAMP$FLAG=03H THEN
48   3                  DO;
49   4                  MANUAL$COUNT=00;
50   4                              INCREMENT(0) = 00;

                                    /* I.E. OUTPUT NOTHING */

51   4                  GO TO ENOUGH;    /* SKIP */
52   4                  END;

                 /* CHECK FOR SATURATED MANIPULATED VARIABLE */
                 /* AND TAKE APPROPRIATE ACTION                  */

53   3          IF ((RAMP$FLAG = 01B) AND (TO$PLANT  > MAXLEVEL))
                                OR
                  ((RAMP$FLAG = 10B) AND (TO$PLANT  < MINLEVEL))

                                THEN
54   3                  DO;

                 /* ABNORMAL OPERATION:- EFFECTIVELY DO          */
                 /* NOTHING BUT WARN                             */
```

```
 55   4                     INCREMENT(0) = 00;

 56   4                SAT = 80H ; /* ALARM OPERATOR */
 57   4                END;

            ELSE
 58   3              DO;    /* NORMAL OPERATION */

 59   4                  SAT = 00; /* NORMAL OPERATION , NO ALARM */

          /* CASE 2&3:                                    */
          /* RAMP UP IF 01, DOWN IF 10 (BINARY)                     */

 60   4              IF RAMP$FLAG=03H THEN
 61   4                        DO;
 62   5                       INCREMENT(3) = 80H;
 63   5                        END;


          /* ADD NEGATIVE SIGN FOR RAMP DOWN      */

 64   4                END;

 65   3             OUTPUT(0E4H) = SAT OR ERR;

          /* RAMP$FLAG SHOULD NEVER  EQUAL 00
          /* IF THIS HAPPENS , IT IS CHECKED 4 TIMES.   */

 66   3           IF RAMP$FLAG = 00H THEN
 67   3              DO;
 68   4               NUMBER = 5;
 69   4               DO  WHILE (NUMBER > 1)
                          AND
                      ((SHR(INPUT(0E8H),3) AND 03H)=00H);
 70   5                   NUMBER = NUMBER-1;
 71   5               END;
 72   4               IF NUMBER < 2 THEN
 73   4               CALL FLASH;

                            /* ALARM TO OPERATOR VIA LEDS */

 74   4               INCREMENT(0) = 00H;

 75   4             END;

 76   3       ENOUGH:    CALL ANALOGUE$OUT;

          /* MANUAL OUTPUT AS DEFINED BY INCREMENT      */

 77   3            IF RAMP$FLAG = 02H
           THEN DO;
 79   4              INCREMENT(3D) = 00H;
 80   4              DO I = 0 TO 3;
 81   5                INCREMENT(I)=NOT(INCREMENT(I));
 82   5              END;
 83   4              INCREMENT(0)=INCREMENT(0)+1;
 84   4             END;
```

```
                         /* CONVERT NEGATIVE NUMBER TO 2S COMP *
 85   3                  CALL FLTDS(.FPR,.INCREMENT);
 86   3                  CALL FADD(.FPR,.DELTAVO);
 87   3                  CALL FSTOR(.FPR,.DELTAVO);

 88   3            END; /* MANUAL SECTION                         */

                      /* NOW PROCESS SAMPLING TIME FLAG            */

 89   2                SAMPLE$COUNT=SAMPLE$COUNT+1;
 90   2                IF SAMPLE$COUNT>END$COUNT THEN
 91   2                DO;
 92   3                   SAMPLE$COUNT=00H; /* RESET COUNT */
 93   3                   SAMPLE$TIME = 01H; /* SET FLAG   */
 94   3                END;

 95   2                    STATUS = STATUS;

                        /* CLEAR INTERRUPT BY THIS STATEMENT */

 96   2            END INTERRUPT$PROCESSOR;
 97   1          END INTERRUPT$MODULE;
```

```
MODULE INFORMATION:

     CODE AREA SIZE      = 0173H    371D
     VARIABLE AREA SIZE  = 0002H      2D
     MAXIMUM STACK SIZE  = 000EH     14D
     249 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE ANALOGUEOUTMODULE
OBJECT MODULE PLACED IN :F4:MOD5.OBJ
COMPILER INVOKED BY: PLM80 :F4:MOD5.SRC DEBUG

```
                /****************************************************/
                /*                                                  */
                   /*              MODULE   5                             */
                /*                                                  */
                /*              ANALOGUE OUT                        */
                /*                                                  */
                /*  THIS MODULE ACCEPTS A 16 BIT NUMBER, REPRESENTING */
                /*  THE INCREMENTAL CHANGE IN MANIPULATED VARIABLE.   */
                /*  THE 12 LSBS ARE SIGNIFICANT.                    */
                /*              . THIS IS ADDED/SUBTRACTED TO/FROM  */
                /*  THE PREVIOUS MANIPULATED VARIABLE AND OUTPUTTED */
                /*  TO THE PLANT VIA D/A NO. 1.(WHICH IS MEMORY     */
                /*  ADDRESSED)  THE VALUE 'TO$PLANT' IS MADE        */
                /*  AVAILABLE TO OTHER ROUTINES AS THE IMMEDIATE    */
                /*  PAST MANIPULATED VARIABLE.                      */
                /*                                                  */
                /****************************************************/
                /*


                */

  1             ANALOGUE$OUT$MODULE:
                 DO:
  2   1             DECLARE INCREMENT(4D) BYTE EXTERNAL;
  3   1             DECLARE TO$PLANT ADDRESS EXTERNAL:
  4   1                 DECLARE (SAT,ERR) BYTE EXTERNAL;
  5   1                 DECLARE (MINLEVEL,MAXLEVEL) ADDRESS EXTERNAL;

  6   1          ANALOGUE$OUT:
                 PROCEDURE PUBLIC;
  7   2             DECLARE TEMP ADDRESS;
  8   2             DECLARE DAC1LO BYTE AT (7FF6H);
  9   2             DECLARE DAC1HI BYTE AT (7FF7H);

 10   2              SAT = 00;

 11   2              TEMP=(DOUBLE(INCREMENT(0)) OR
                           SHL(DOUBLE((INCREMENT(1)),8));
 12   2              IF INCREMENT(2) OR (INCREMENT(3) AND 7FH)
                                 () 00H THEN
 13   3                DO;
 14   3                 SAT = 80H;
 15   3                 TO$PLANT=MAX$LEVEL;
 16   3                 IF ( INCREMENT(3) AND 80) THEN
 17   3                 TO$PLANT = MINLEVEL;
 18   3               END;
                                 ELSE
 19   2               DO:

 20   3           IF(INCREMENT(3D) AND 80H) = 80H
```

```
                              THEN
  21   3                    TO$PLANT = TO$PLANT - TEMP;
                                /* CHECK FOR NEGATIVE SIGN AND REMOVE IT */
                            /* IF NECESSARY.                           */

                                ELSE
  22   3                    TO$PLANT = TO$PLANT + TEMP;

  23   3                    IF (TO$PLANT)MAXLEVEL) THEN
  24   3                    DO;
  25   4                          SAT=80H;
  26   4                      TO$PLANT = MAX$LEVEL;
  27   4                    END;

  28   3                    IF (TO$PLANT(MINLEVEL) THEN
  29   3                    DO;
  30   4                          SAT=80H;
  31   4                          TO$PLANT=MINLEVEL;
  32   4                        END;
  33   3                    END;

  34   2                    OUTPUT(0E4H) = SAT OR ERR;
                                /* LIGHT LEDS */


  35   2                    DAC1LO = LOW(TO$PLANT);
  36   2                    DAC1HI = HIGH(TO$PLANT);

                            /* SEND TO D/A AND CONVERT.               */
  37   2                 END ANALOGUE$OUT;
  38   1              END ANALOGUE$OUT$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE      = 00B7H    183D
     VARIABLE AREA SIZE  = 0002H      2D
     MAXIMUM STACK SIZE  = 0004H .    4D
     183 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
/*************************************/
/*                                   */
/*          MODULE 12                */
/*                                   */
/* THIS MODULE IS CALLED TO ENABLE   */
/* A SMOOTH ENTRY TO & EXIT FROM     */
/* MONITOR WHEN REQUESTED BY OPERTR  */
/*                                   */
/*************************************/


$TITLE('JUMP TO MONITOR ROUTINE')
NAMETERROGATE

        CSEG

MONINS: RST 1
        BACK: RET;ENSURE THAT CONTROLLER
                                ;STACK IS USED ON RETURN
        END
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INCONTVARIABLEMODULE
OBJECT MODULE PLACED IN :F4:MOD6.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD6.SRC


                   /**********************************************/
                   /*                                            */
                   /*          MODULE 6                          */
                   /*                                            */
                   /*          INPUT CONTROLLED VARIABLE         */
                   /*                                            */
                   /*  THIS MODULE INITIATES A/D CONVERSION,     */
                   /*  WAITS, AND READS IN RESULT AS 12 BIT(LSB) */
                   /*  VALUE. THE VARIABLE FROM$PLANT IS         */
                   /*  AVAILABLE TO OTHER ROUTINES AS THE MOST   */
                   /*  RECENT VALUE OF THE CONTROLLED VARIABLE.  */
                   /*                                            */
                   /**********************************************/
                   /*

                   */
  1                IN$CONT$VARIABLE$MODULE:
                     DO;
  2   1                DECLARE FROM$PLANT(4D) BYTE  EXTERNAL;

  3   1            INPUT$CONTROLLED$VARIABLE:
                     PROCEDURE PUBLIC;

  4   2                DECLARE ADCHI BYTE AT (7FFEH);
  5   2                DECLARE ADCLO BYTE AT (7FFDH);
  6   2                DECLARE STATUS BYTE AT (7FFCH);
  7   2                DECLARE CNVCMD BYTE AT (7FFBH);

  8   2            DISABLE ;
  9   2            CNVCMD = 00;

                   /* START CONVERSION  */

 10   2            DO WHILE ((STATUS AND 80H) = 80H);
 11   3            END;  /* WHILE LOOP */

                   /* TEST FOR END OF A/D CONVERSION. */
                   /* READ WHEN READY.  */

 12   2            FROM$PLANT(0)=ADCLO;
 13   2            FROM$PLANT(1)=ADCHI;

                   /* READ IN RESULT FROM A/D       */

 14   2            ENABLE;

 15   2        END INPUT$CONTROLLED$VARIABLE;
 16   1        END IN$CONT$VARIABLE$MODULE;
```

PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INCONTVARIABLEMODULE
OBJECT MODULE PLACED IN :F4:MOD6.OBJ
COMPILER INVOKED BY: PLM80 :F4:MOD6.SRC

```
                /*************************************************/
                /*                                               */
                /*          MODULE 6                             */
                /*                                               */
                /*          INPUT CONTROLLED VARIABLE            */
                /*                                               */
                /*  THIS MODULE INITIATES A/D CONVERSION,        */
                /*  WAITS, AND READS IN RESULT AS 12 BIT(LSB)    */
                /*  VALUE. THE VARIABLE FROM$PLANT IS            */
                /*  AVAILABLE TO OTHER ROUTINES AS THE MOST      */
                /*  RECENT VALUE OF THE CONTROLLED VARIABLE.     */
                /*                                               */
                /*************************************************/
                /*

                */

  1             IN$CONT$VARIABLE$MODULE:
                  DO;
  2   1             DECLARE FROM$PLANT(40) BYTE  EXTERNAL;

  3   1         INPUT$CONTROLLED$VARIABLE:
                  PROCEDURE PUBLIC;

  4   2             DECLARE ADCHI BYTE AT (7FFEH);
  5   2             DECLARE ADCLO BYTE AT (7FFDH);
  6   2             DECLARE STATUS BYTE AT (7FFCH);
  7   2             DECLARE CNVCMD BYTE AT (7FFBH);

  8   2         DISABLE ;
  9   2         CNVCMD = 00;

                /* START CONVERSION  */

 10   2         DO WHILE ((STATUS AND 80H) = 80H);
 11   3         END;  /* WHILE LOOP */

                /* TEST FOR END OF A/D CONVERSION. */
                /* READ WHEN READY.  */

 12   2         FROM$PLANT(0)=ADCLO;
 13   2         FROM$PLANT(1)=ADCHI;

                /* READ IN RESULT FROM A/D        */

 14   2         ENABLE;

 15   2       END INPUT$CONTROLLED$VARIABLE;
 16   1       END IN$CONT$VARIABLE$MODULE;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 0021H     33D
     VARIABLE AREA SIZE = 0000H      0D
     MAXIMUM STACK SIZE = 0000H      0D
     49 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE EXTRAVARIABLEMODULE
OBJECT MODULE PLACED IN :F4:MOD7.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD7.SRC DEBUG


              /*********************************************/
              /*                                           */
              /*          MODULE 7                         */
              /*          EXTRA VARIABLE MODULE            */
                /*                                                  */
              /* THIS MODULE UNDERTAKES 3 TASKS:-          */
              /*   1) OUTPUT OF PREDICTION ERROR           */
              /*      IN ANALOGUE FORM TO A CHART          */
              /*      RECORDER .                           */
              /*   2) DISPLAY PREDICTION ERROR VIA         */
              /*      VIA LIGHT EMITTING DIODES ON         */
              /*      THE FRONT PANEL IN BAR GRAPH         */
              /*      FORMAT.                              */
              /*   3) OUTPUT OF A SINGLE MODEL             */
              /*      PARAMETER TO THE TELETYPE            */
              /*                                           */
              /*  ALL ABOVE FUNCT[illegible]CUR REGULAR-   */
              /*  LY DURING RUN TIME                       */
              /*  ALL ARE INTENDED TO KEEP THE OPER-       */
              /*  ATOR INFORMED OF PROGRESS.               */
              /*                                           */
              /*********************************************/
                /*


              */
              EXTRA$VARIABLE$MODULE:
                 DO;

          $INCLUDE (:F1:FLOAT.SRC)

                   FSET:
                   PROCEDURE(FA,OP1,OP2) EXTERNAL;
                   DECLARE(FA,OP1,OP2) ADDRESS;
                   END FSET;

                   FSUB:
              PROCEDURE(FA,OA) EXTERNAL;
              DECLARE (FA,OA) ADDRESS;
              END FSUB;


               FADD:
              PROCEDURE(FA,OA) EXTERNAL;
               DECLARE (FA,OA) ADDRESS;
               END FADD;

               FOFB2D:
               PROCEDURE(FA,OA) EXTERNAL;
               DECLARE (FA,OA) ADDRESS;
               END FOFB2D;
```

```
              =
              =
 14    1      =        FDIV:
              =       PROCEDURE(FA,OA) EXTERNAL;
 15    2      =        DECLARE (FA,OA) ADDRESS;
 16    2      =        END FDIV;
              =
 17    1      =       FMUL:
              =       PROCEDURE(FA,OA) EXTERNAL;
 18    2      =        DECLARE (FA,OA) ADDRESS;
 19    2      =       END FMUL;
              =
 20    1      =       FLOAD:
              =       PROCEDURE(FA,OA) EXTERNAL;
 21    2      =        DECLARE (FA,OA) ADDRESS;
 22    2      =       END FLOAD;
              =
 23    1      =         FCLR:
              =             PROCEDURE(FA) EXTERNAL;
 24    2      =        DECLARE FA ADDRESS;
 25    2      =         END FCLR;
              =
 26    1      =        FNEG:
              =       PROCEDURE (FA) EXTERNAL;
 27    2      =        DECLARE FA ADDRESS;
 28    2      =        END FNEG;
              =
 29    1      =        FLTDS:
              =           PROCEDURE (FA,OA) EXTERNAL;
 30    2      =       DECLARE (FA,OA) ADDRESS;
 31    2      =        END FLTDS;
              =
 32    1      =        FIXSD:
              =       PROCEDURE (FA,OA) EXTERNAL;
 33    2      =       DECLARE (FA,OA) ADDRESS;
 34    2      =        END FIXSD;
              =
 35    1      =        FSTOR:
              =           PROCEDURE(FA,OA) EXTERNAL;
 36    2      =           DECLARE (FA,OA) ADDRESS;
 37    2      =        END FSTOR;
              =
 38    1      =        DECLARE FPR(18D) BYTE EXTERNAL;


 39    1              DECLARE ERRPRED(4D) BYTE EXTERNAL;
 40    1                  DECLARE FIRST$ERROR$FLAG BYTE EXTERNAL
 41    1                  DECLARE SPECIFY STRUCTURE(
                                  SIGN BYTE,
                                  SCALE ADDRESS,
                                  SLENGTH BYTE,
                                  STRING$PTR ADDRESS) EXTERNAL;
 42    1                  DECLARE DEC$STRING(10) BYTE EXTERNAL;
 43    1                  DECLARE (TTY$FLAG,TTY$TIME,TTY$LP) BYT
                                  EXTERNAL;
 44    1                  DECLARE PARMNO BYTE EXTERNAL;
 45    1                  DECLARE [illegible] BYTE EXTERNAL;
```

```
 46   1              DECLARE L(200D) BYTE EXTERNAL;


 47   1      ASC$OUT:
                PROCEDURE(THING) PUBLIC:

            /***********************************/

             /* THIS PROCEDURE CHECKS STATUS OF */
             /* TTY AND OUTPUTS PARAMETER TO IT */
             /* WHEN READY.                     */

 48   2      DECLARE THING BYTE;


 49   2       CO: IF( INPUT(237D) AND 01H) = 00H

              THEN
 50   2         GO TO CO;

              ELSE
 51   2          OUTPUT(0ECH) = THING ;

 52   2      END ASC$OUT;

 53   1        NUMOUT:
                PROCEDURE (AMOUNT,WIDTH):
            /********************************/

                /* NUMOUT TAKES A BINARY NU[illegible]ER
                CONVERTS IT TO DECIMAL I[illegible]
                ASCII FORMPT. THIS IS OU[illegible] TO
                TTY.
                */

 54   2         DECLARE AMOUNT ADDRESS;
 55   2         DECLARE WIDTH BYTE;
 56   2         DECLARE I BYTE;
 57   2         DECLARE CHARS(1) BYTE;
 58   2         DECLARE DIGITS(*) BYTE DATA ('0123456789');

 59   2         DO I=1 TO WIDTH;
 60   3         CHARS(WIDTH-I)=DIGITS(AMOUNT MOD 10D);
 61   3         AMOUNT = AMOUNT/10D;
 62   3         END;

 63   2         I = 00;

 64   2         DO WHILE CHARS(I) = '0' AND I < WIDTH-1;
 65   3         CHARS(I)=00;
 66   3         I=I+1;
 67   3         END;

 68   2         DO I=0 TO WIDTH-1;
 69   3         CALL ASC$OUT(CHARS(I));
 70   3         END;
 71   2         END NUMOUT;
```

```
 72   1          EXTRA$VARIABLE:
                   PROCEDURE PUBLIC;
                 /****************************/

                   /* PREDICTION ERROR IS CONVERTED TO INTEGER */
                   /* AND OUTPUTTED TO D/A NO 2 AND CHART RECORDER */
                   /* SUCH THAT 0 TO 100% ERROR = 0 TO 10 VOLTS    */
                       /* NOTE THAT PRED ERR CAN RANGE 0 TO 200% BUT   */
                       /* IS NOT EXPECTED TO.                           */

 73   2            DECLARE DAC2LO BYTE AT (7FF4H);
 74   2            DECLARE DAC2HI BYTE AT (7FF5H);
 75   2                DECLARE LEDS$ERR ADDRESS;
 76   2                DECLARE LEDS BYTE;
 77   2                DECLARE REFERENCE ADDRESS;
 78   2                DECLARE DIGS(7) BYTE DATA (0,1,3,7,15D,31D,63D);
 79   2                DECLARE I BYTE;
 80   2                DECLARE P BYTE;

 81   2            CALL FLOAD(.FPR,.ERRPRED);
 82   2            CALL FIXSD(.FPR,.ERRPRED);

               /* CHECK FOR OVERFLOW */

 83   2            IF ((ERRPRED(2) <> 00H) OR
                      ((ERRPRED(3) AND 07FH) <> 00H) OR
                     ((ERRPRED(1) AND 0F0H)  <> 00H))    THEN

 84   2                   DO;
 85   3                DAC2LO=0FFH;
 86   3                DAC2HI=0FFH;

 87   3                LEDS$ERR = 0FFFH;
               /* PASSED TO FRONT PANEL DISPLAY SECTION */

 88   3              END;

               ELSE

 89   2              DO;
 90   3               DAC2LO=ERRPRED(0);
 91   3               DAC2HI=ERRPRED(1);

 92   3               LEDS$ERR = SHL(DOUBLE(ERRPRED(1)),8)
                               OR  DOUBLE(ERRPRED(0));

 93   3              END;

             /************************************/

             /* THIS ROUTINE DISPLAYS PREDICTION ERROR
                VIA 6 LEDS ON THE FRONT PANEL. THE FIRST
                PREDICTION ERROR AFTER A RESET OR AN INTER-
                ROGATE REQUEST IS USED AS A REFERENCE.
```

```
                SUBSEQUENT ERRORS ARE NORMALISED TO THIS */

 94   2        ERROR$LED:
              /************************/

                DO;

 95   3         IF FIRST$ERROR$FLAG = 01H THEN

 96   3         DO;
 97   4            REFERENCE = LEDS$ERR;
 98   4            FIRST$ERROR$FLAG = 00;
 99   4         END;

100   3         LEDS = DIGS(((SHL(LEDS$ERR,4)/REFERENCE)*6D)/15D);
101   3         IF (LEDS$ERR > REFERENCE) THEN
102   3             LEDS=063D;

               /* FOR A LINEAR DISPLAY ON LEDS , MAP
                  PREDICTION ERROR AS A NUMBER BETWEEN
                  0 & 6, THEN LOOK UP IN TABLE (DIGS)
                  AND OUTPUT THIS NUMBER TO LEDS PORT  */

103   3           ERR = SHL(LEDS, [illegible]);

104   3         OUTPUT(228D) = SAT OR ERR ;

105   3         END; /* ERROR$LED*/
106   2        PARM$OUT:
              /******************/

                  IF TTY$FLAG=01
                  THEN
107   2              DO;

108   3         IF TTY$TIME > TTY$UP THEN
109   3                 DO;
110   4                    TTY$TIME = 00;
111   4                    P = (PARMNO*4D);
112   4                    CALL FLOAD(.FPR,.L(P));
113   4                    SPECIFY.STRING$PTR=.DECSTRING;
114   4                    CALL FQFB2D(.FPR,.SPECIFY);

                    /* CONVERT TO BINARY THE PARAMETER POINTED
                       TO BY PARMNO.
                    */

                     /* NOW OUTPUT TO TTY */

115   4              CALL ASC$OUT(' ');
116   4              CALL ASC$OUT('L');
117   4          CALL ASC$OUT(' ');
118   4              CALL ASC$OUT(' ');
119   4          CALL ASC$OUT(SPECIFY.SIGN);
120   4          CALL ASC$OUT(' ');
121   4              CALL ASC$OUT('[illegible]');
122   4          CALL ASC$OUT('X');
```

```
 123   4            CALL ASC$OUT('P');

 124   4         IF ((SPECIFY.SCALE AND 8000H)=8000H) THEN
 125   4                 CALL ASC$OUT('-');
                  ELSE
 126   4                 CALL ASC$OUT('+');

 127   4          SPECIFY.SCALE =((NOT(SPECIFY.SCALE))+ 0001H);

                 /* CONVERT FROM 2S COMPLEMENT */

 128   4                 CALL NUMOUT(SPECIFY.SCALE,2);
 129   4                 CALL ASC$OUT(' ');

 130   4          DO I=0 TO SPECIFY.SLENGTH;

 131   5                 CALL ASC$OUT(DEC$STRING(I));
 132   5          END;

 133   4            CALL ASC$OUT(0DH);
 134   4            CALL ASC$OUT(0AH);

                  /* CARR RET & LINE FEED */

 135   4       END;
 136   3            TTY$TIME=TTY$TIME+01;
 137   3      END;

 138   2    END EXTRA$VARIABLE;
 139   1    END EXTRA$VARIABLE$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 026DH    621D
     VARIABLE AREA SIZE = 000DH     13D
     MAXIMUM STACK SIZE = 0006H      6D
     310 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PREDICTANDUPDATEMODULE
OBJECT MODULE PLACED IN :F4:MOD8.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD8.SRC

                  /*********************************************/
                  /*                                           */
                  /*              MODULE  NO. 8                */
                  /*          PREDICT AND UPDATE               */
                  /*                MODULE                     */
                  /*                                           */
                  /* THIS MODULE PREDICTS CONTROLLED VARIABLE  */
                  /* DYPRED. PREDICTION ERROR IS THEN USED TO  */
                  /* UPDATE MODEL PARAMETERS ( L(I) ).         */
                  /* GAMMA IS PURELY A WEIGHTING CONSTANT.     */
                  /* THIS ROUTINE IS CALLED IN BOTH MANUAL AND */
                  /* AUTO MODES. IMPORTANT VARIABLES ARE AVAIL-*/
                  /* ABLE TO OTHER ROUTINES ( DATA$SHUFFLE,    */
                  /* DISPLAY$VARIABLE ).                       */
                  /*                                           */
                  /*********************************************/
                  /*


                  */
   1              PREDICT$AND$UPDATE$MODULE:

                    DO;


   2    1          FADD:
                  PROCEDURE(FA,OA) EXTERNAL;
   3    2          DECLARE (FA,OA) ADDRESS;
   4    2          END FADD;

   5    1          FDIV:
                  PROCEDURE(FA,OA) EXTERNAL;
   6    2          DECLARE (FA,OA) ADDRESS;
   7    2          END FDIV;

   8    1         FMUL:
                  PROCEDURE(FA,OA) EXTERNAL;
   9    2          DECLARE (FA,OA) ADDRESS;
  10    2         END FMUL;

  11    1         FLOAD:
                  PROCEDURE(FA,OA) EXTERNAL;
  12    2          DECLARE (FA,OA) ADDRESS;
  13    2         END FLOAD;

  14    1         FSTOR:
                  PROCEDURE(FA,OA) EXTERNAL;
  15    2          DECLARE (FA,OA) ADDRESS;
  16    2         END FSTOR;

  17    1          FCLR:
```

```
                   PROCEDURE (FA) EXTERNAL;
 18   2             DECLARE FA ADDRESS;
 19   2              END FCLR;

 20   1             FNEG:
                   PROCEDURE (FA) EXTERNAL;
 21   2             DECLARE FA ADDRESS;
 22   2             END FNEG;

 23   1                   DECLARE SUM$L(4D) BYTE EXTERNAL;
 24   1                 DECLARE ARRAY$LENGTH BYTE EXTERNAL;

 25   1                 DECLARE FPR(18D) BYTE EXTERNAL;
 26   1                 DECLARE MIDDLEMAN (4D) BYTE EXTERNAL;
 27   1                     DECLARE L(200D) BYTE EXTERNAL;
 28   1                     DECLARE DV(200D) BYTE EXTERNAL;
 29   1                     DECLARE FILTGAIN(4D) BYTE EXTERNAL;
 30   1                     DECLARE R2(4D) BYTE EXTERNAL;
 31   1                     DECLARE R1(4D) BYTE EXTERNAL;
 32   1                     DECLARE DY1(4D) BYTE EXTERNAL;
 33   1                     DECLARE R2SUB1(4D) BYTE EXTERNAL;
 34   1                     DECLARE DY0(4D) BYTE EXTERNAL;
 35   1                     DECLARE GAMMA(4D) BYTE EXTERNAL;
 36   1                     DECLARE PLANT$GAIN(4D) BYTE EXTERNAL;
 37   1                     DECLARE DYPRED(4D) BYTE EXTERNAL;
 38   1                     DECLARE ERRPRED(4D) BYTE EXTERNAL;

 39   1            PREDICT$AND$UPDATE:
                          PROCEDURE PUBLIC;

 40   2            DECLARE I BYTE;
 41   2            DO;
 42   3               CALL FCLR(.FPR); /* CLEAR FLOATING PT ACC. */
 43   3               CALL FSTOR(.FPR,.DYPRED);
 44   3                   CALL FSTOR(.FPR,.ERRPRED);
                           /* CLEAR PREDICTION & ERROR FOR RECYCLE */

                   /* NOW PREDICT */

 45   3             DO I = 4D TO (ARRAY$LENGTH * 4D - 4D) BY 4D;

 46   4                CALL FLOAD(.FPR,.L(I));
                       /* LOAD MODEL PARAMETER */

 47   4                CALL FMUL(.FPR,.DV(I));
                       /* MULTIPLY BY MANIPULATED VARIABLE */

 48   4                CALL FADD(.FPR,.DYPRED);
                       /* ADD RESULT TO PREDICTION */

 49   4                CALL FSTOR(.FPR,.DYPRED);
                       /* AND STORE RESULT */

 50   4             END;
```

```
                        /* THE ABOVE DO LOOP CALCULATES SUM
                           OF MODEL PARAMETERS * HISTORY OF
                           MANIPULATED VARIABLE              */

  51   3                   CALL FDIV(.FPR,.FILTGAIN);
                           /* DIVIDE BY FILTER GAIN */

  52   3                   CALL FADD(.FPR,.DY1);
                           /* ADD PREVIOUS VALUE OF CONTROLLED
                           VARIABLE                          */

  53   3                   CALL FDIV(.FPR,.R1);

  54   3                   CALL FSTOR(.FPR,.DYPRED);
                           /* STORE FINAL PREDICTION RESULT */

                        /* NOW UPDATE */

  55   3                   CALL FNEG(.FPR);
                           /* NEGATES PREDICTION TO -YPRED*/

  56   3                   CALL FADD(.FPR,.DY0);
                           /* CALCULATE PREDICTION ERROR */

  57   3                   CALL FSTOR(.FPR,.ERRPRED);

  58   3                   CALL FMUL(.FPR,.GAMMA);
                           /* MULTIPLY BY WEIGHTING FACTOR */

  59   3                   CALL FSTOR(.FPR,.MIDDLEMAN);
                           /* KEEP FOR LATER USE     */

  60   3                DO I=4D TO (ARRAY$LENGTH*4D-4D) BY 4D;

  61   4                   CALL FLOAD(.FPR,.MIDDLEMAN);
  62   4                   CALL FMUL(.FPR,.DV(I));
                           /* DV(I)*GAMMA*PREDICTION ERROR */

  63   4                   CALL FADD(.FPR,.L(I));
                           /* FOR EACH MODEL PARAMETER L(I)
                              CALCULATE NEW PARAMETER
                              = OLD L(I) + GAMMA*DV(I)*ERRPRED */

  64   4                   CALL FSTOR(.FPR,.L(I));
                           /* STORE EACH NEW PARAMETER IN TURN */

  65   4                  END;
                              /* NOW CALCULATE ESTIMATED PLANT S.S. GAIN */
                              /* 1/\ = SSG = SUMMATION L(I)/R2-1          */

  66   3                      CALL FSTOR(.FPR,.L(4D));  /* LOAD FIRST L */

  67   3                    DO I= 4D TO (ARRAY$LENGTH*4D-8D) BY 4D;
  68   4                      CALL FADD(.FPR,.L(I+4D));
  69   4                    END;

  70   3                     CALL FSTOR(.FPR,.SUM$L);
```

```
  71   3                 CALL FDIV(.FPR,.R2SUB1);
                         /* R2SUB1 = CONSTANT = R2-1 INITIALIZED IN SETUP */
  72   3                 CALL FSTOR(.FPR,.PLANT$GAIN);

  73   3           END;

  74   2             END; /* PROCEDURE */
  75   1          END; /* PREDICT$AND$UPDATE$MODULE */
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0196H    406D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 0002H      2D
     174 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MANIPVARIABLECALCMODULE
OBJECT MODULE PLACED IN :F4:MOD9.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD9.SRC


              /*******************************************************/
              /*                                                     */
              /*                 MODULE NO. 9                        */
              /*            MANIPULATED VARIABLE                     */
              /*            CALCULATION MODULE.                      */
              /*                                                     */
              /* THIS MODULE TAKES THE UPDATED MODEL                 */
              /* PARAMETERS OBTAINED FROM PREDICT AND                */
              /* UPDATE MODULE [illegible]. THESE ARE USED TO CALC-  */
              /* ULATE THE REQUIRED MANIPULATED VARIABLE.            */
              /* THIS ROUTINE IS CALLED IN AUTO MODE ONLY.           */
              /*                IMPORTANT VARIABLES ARE AVAIL-       */
              /* ABLE TO OTHER ROUTINES (DATA$SAMPLE,                */
              /* DISPLAY$VARIABLE).                                  */
              /*
              /*******************************************************
              /*


              */
   1          MANIP$VARIABLE$CALC$MODULE:

                DO;

   2    1         FSUB:
              PROCEDURE(FA,OA) EXTERNAL;
   3    2     DECLARE (FA,OA) ADDRESS;
   4    2     END FSUB;


   5    1      FADD:
              PROCEDURE(FA,OA) EXTERNAL;
   6    2      DECLARE (FA,OA) ADDRESS;
   7    2      END FADD;

   8    1      FDIV:
              PROCEDURE(FA,OA) EXTERNAL;
   9    2      DECLARE (FA,OA) ADDRESS;
  10    2      END FDIV;

  11    1     FMUL:
              PROCEDURE(FA,OA) EXTERNAL;
  12    2      DECLARE (FA,OA) ADDRESS;
  13    2     END FMUL;

  14    1     FLOAD:
              PROCEDURE(FA,OA) EXTERNAL;
  15    2      DECLARE (FA,OA) ADDRESS;
  16    2     END FLOAD;

  17    1     FSTOR:
```

```
                 PROCEDURE(FA,OA) EXTERNAL;
 18   2           DECLARE (FA,OA) ADDRESS;
 19   2          END FSTOR;

 20   1           FCLR:
                 PROCEDURE (FA) EXTERNAL;
 21   2           DECLARE FA ADDRESS;
 22   2            END FCLR;

 23   1           FNEG:
                 PROCEDURE (FA) EXTERNAL;
 24   2           DECLARE FA ADDRESS;
 25   2           END FNEG;

 26   1               DECLARE ARRAY$LENGTH BYTE EXTERNAL;
 27   1               DECLARE SUM$L(4D) BYTE EXTERNAL;
 28   1               DECLARE SET$ERROR0 (4D) BYTE EXTERNAL;
 29   1               DECLARE SET$ERROR1 (4D) BYTE EXTERNAL;

 30   1               DECLARE FPR(18D) BYTE EXTERNAL;
 31   1                    DECLARE L(200D) BYTE EXTERNAL;
 32   1                    DECLARE DV(200D) BYTE EXTERNAL;
 33   1                    DECLARE FILTGAIN(4D) BYTE EXTERNAL;
 34   1                    DECLARE R2(4D) BYTE EXTERNAL;
 35   1                    DECLARE R1(4D) BYTE EXTERNAL;
 36   1                    DECLARE DY1(4D) BYTE EXTERNAL;
 37   1                    DECLARE R2SUB1(4D) BYTE EXTERNAL;
 38   1                    DECLARE DY0(4D) BYTE EXTERNAL;
 39   1                    DECLARE GAMMA(4D) BYTE EXTERNAL;
 40   1                    DECLARE PLANT$GAIN(4D) BYTE EXTERNAL;
 41   1                    DECLARE DYPRED(4D) BYTE EXTERNAL;
 42   1                    DECLARE ERRPRED(4D) BYTE EXTERNAL;


 43   1          MANIP$VARIABLE$CALC:
                           PROCEDURE PUBLIC;

 44   2          DECLARE I BYTE;
 45   2               DECLARE Q(4D) BYTE;

 46   2          DO;
 47   3             CALL FCLR(.FPR); /* CLEAR FLOATING PT ACC.*/
 48   3             CALL FSTOR(.FPR,.DV(0) ); /* DV(0) IS MANIP VAR
                                TO BE CALCULATED AND OUTED TO PLANT  */
                        /* INITIALIZED TO ZERO HERE                  */

 49   3                 CALL FLOAD(.FPR,.SUM$L  );
                         /* SUMMATION OF MODEL PARAMETERS CALCULATED
                            IN 'PREDICT AND UPDATE PROC '            */

 50   3                 CALL FNEG(.FPR);

                 /* -VE VALUE NEEDED */
                 /* Q1 = L1 - SUMMATION ( L(I) )
                  = L1 + ( 1-R2 )/N  CAN NOW BE CALCULATED. */

                 /* THE FOLLOWING DO LOOP CALCULATES
```

```
   [illegible] THEN CALCULATES
   [illegible] WHICH IS PARTIAL CONTRIBU[illegible]
   [illegible] MANIP VAR TO BE CALCULATED
   [illegible] LOADED FOR A RECYCLE [illegible]
   [illegible] (N+1) TO BE CALCULATED                */

   [illegible] ARRAY$LENGTH + 4D - 4D) BY -D:

       CALL FADD (.FPR,.L(I));
       /* [illegible] I) IN ACCUMULATOR */

       CALL FSTOR(.FPR,.[illegible]);
       /* STORE FOR USE IN [illegible] */

       CALL FMUL(.FPR,.DV([illegible]));
       /* DV(N) * Q(N)                     */

       CALL FADD (.FPR,.DV(0));
       CALL FSTOR(.FPR,.DV([illegible]));

       /* AND STORE PARTIAL RESULT TO MANIP VAR */
       CALL FLOAD(.FPR,.Q);
       /* LOAD Q FOR NEXT CYCLE Q(N+1)=Q(N)+[illegible] */

   END;

       CALL [illegible](.FPR,.DV(0));

   /* [illegible] CALCULATE PARTIAL RESULT
       DV[illegible] = (1/PLANTGAIN)*(1/[illegible]2)*[illegible] DV(I)*Q[illegible]
   */

       CALL FDIV(.FPR,.PLANT$GAIN);
       /* DIVIDE BY PLANT  GAIN */

       CALL FDIV(.FPR,.[illegible]);

       CALL [illegible](.FPR,.DV(0));

       /* [illegible] ERROR FROM SET POINT CONTRI[illegible]
              IS ADDED TO DV(0)
         */

       CALL FLOAD(.FPR,.SET$ERROR[illegible]);
          CALL FMUL(.FPR,.R1);
       /* CALCULATE R1*E(0)                  */

          CALL FSUB(.FPR,.SET$ERROR1);

       CALL [illegible] GAIN);

       CALL FDIV(.FPR,.PLANT$GAIN);

       CALL FADD(.FPR,.DV(0));
       CALL FSTOR(.FPR,.DV(0));
       /* [illegible] PLANT GAIN*[illegible] GA[illegible] */
```

```
PL/M-80 COMPILER
                                        PAGE   [illegible]

                               [illegible]


  70   [illegible]       END;

  71   [illegible]          END; /* PROCEDURE */
  72   1          END; /* [illegible] */


MODULE INFORMATION:

     CODE AREA SIZE       = 00F6H      [illegible]
     VARIABLE AREA SIZE = 0005H        5D
     MAXIMUM STACK SIZE = 000[illegible]H      [illegible]
     172 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER
                                        PAGE   1


ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE ALARMMODULE
OBJECT MODULE PLACED IN :F4:MOD10.OBJ
COMPILER INVOKED BY:  PLM80 :F4:MOD10.SRC


                /*****************************************
                   /*                                      */
                   /*        MODULE 10                     */
                   /*        ALARM MODULE                  */
                   /*                                      */
                   /* THIS MODULE IS CALLED ON FAULT DE-   */
                   /* TECTION. ALL LEDS FLASH.             */
                   /*                                      */
                   /****************************************/
                   /*

                   */
   1              ALARM$MODULE:
                       DO;

   2    1          DECLARE (SAT,ERR) BYTE EXTERNAL;
   3    1          DECLARE FIRST$ERROR$FLAG BYTE EXTERNAL;

   4    1          FLASH:
                        PROCEDURE PUBLIC;

   5    2              DECLARE (I,J) BYTE;

   6    2              SAT=00;
   7    2          ERR=00;

   8    2          FIRST$ERROR$FLAG = 01H;

   9    2          DO I=1 TO 60D;
  10    3          OUTPUT(0E4H)=NOT(SAT);
  11    3            DO J= 1 TO 25;
  12    4            CALL TIME(250D);
  13    4            END;
  14    3          END;
  15    2          SAT=00;
  16    2          END FLASH;
  17    1        END ALARM$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 004AH     74D
     VARIABLE AREA SIZE = 0002H      2D
     MAXIMUM STACK SIZE = 0002H      2D
     37 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
/*************************************************************/
/*                                                           */
/*              MODULE  NO. 11                               */
/*          DATA SHUFFLE MODULE                              */
/*                                                           */
/*                                                           */
/* THIS MODULE TAKES THE TIME DEPENDANT                      */
/* PARAMETERS USED DURING SELF TUNING AND                    */
/* CONTROLLING AND SHUFFLES THEM BACKWARDS                   */
/* IN TIME IN PREPERATION FOR FOR THE NEXT                   */
/* CYCLE . CONSEQUENTLY THIS MODULE IS ONLY                  */
/* CALLED AT THE END OF EACH CYCLE BY EITHER                 */
/* AUTO OR MANUAL MODE.                                      */
/*                                                           */
/*                                                           */
/*************************************************************/
/*


*/
DATA$SHUFFLE$MODULE:

  DO;


$INCLUDE(:F4:FLOAT.SRC)

    DECLARE ARRAY$LENGTH BYTE EXTERNAL;
    DECLARE SET$ERROR0 (40) BYTE EXTERNAL;
    DECLARE SET$ERROR1 (40) BYTE EXTERNAL;

            DECLARE L(2000) BYTE EXTERNAL;
            DECLARE DV(2000) BYTE EXTERNAL;
            DECLARE DY1(40) BYTE EXTERNAL;
            DECLARE DY0(40) BYTE EXTERNAL;
            DECLARE CONTVAR0(40) BYTE EXTERNAL;
            DECLARE CONTVAR1(40) BYTE EXTERNAL;
            DECLARE MANIPVAR0(40) BYTE EXTERNAL;
            DECLARE MANIPVAR1(40) BYTE EXTERNAL;
            DECLARE TO$PLANT ADDRESS EXTERNAL;


      DATA$SHUFFLE:
   PROCEDURE PUBLIC;

DECLARE I BYTE;
        DECLARE I1 BYTE;


      DO I=0 TO 40*ARRAY$LENGTH - 1D;

      I1 = 40*ARRAY$LENGTH - 1 - I;

      DV(I1) = DV(I1 - 40);

        /* SHUFFLE MANIPULATED VARIABLE BACKWARDS
           ONE SAMPLING PERIOD IN TIME.               */

      END;
```

```
      DO I = 00 TO 3D;

        DY1(I)=DY0(I);
        SET$ERROR1(I)=SET$ERROR0(I);
        CONTVAR1(I)=CONTVAR0(I);
        MANIPVAR1(I) = MANIPVAR0(I);

      END;

   END; /* PROCEDURE */
END; /* DA$SHUFFLE$MODULE */
```

```
ASM80 :F4:MOD12.SRC


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0                    PAGE   1

  LOC  OBJ          LINE        SOURCE STATEMENT

                      1 ;        /***********************************/
                      2 ;        /*                                 */
                      3 ;        /*          MODULE 12              */
                      4 ;        /*                                 */
                      5 ;        /* THIS MODULE IS CALLED TO ENABLE */
                      6 ;        /* A SMOOTH ENTRY TO & EXIT FROM   */
                      7 ;        /* MONITOR WHEN REQUESTED BY OPERTR*/
                      8 ;        /*                                 */
                      9 ;        /***********************************/
                     10
                     11
                     12 ;        TITLE('JUMP TO MONITOR ROUTINE')
                     13 ;        NAME    TERROGATE
                     14
                     15          CSEG
                     16
  0000 CF            17          MONINS: RST 1
  0001 C9            18          BACK: RET                 ;ENSURE THAT CONTROLLER
                     19                                    ;STACK IS USED ON RETUR
                     20          END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
BACK    C 0001    MONINS C 0000

ASSEMBLY COMPLETE,    NO ERRORS
```

```
ISIS-II OBJECT LOCATER V3.0 INVOKED BY:
-LOCATE :F1:BAD.MOD CODE(C400H) DATA(3D0FH) STACK(4120H) &
**ORDER(CODE DATA STACK) MAP SYMBOLS PRINT(:LP:)

SYMBOL TABLE OF MODULE BAD
READ FROM FILE :F1:BAD.MOD
WRITTEN TO FILE :F1:BAD

VALUE TYPE SYMBOL

      MOD  SELFTUNECONTROLMODULE
41A0H SYM  MEMORY
3D0FH SYM  ERR
3D21H SYM  MIDDLEMAN
3D25H SYM  FIRSTERRORFLAG
3D26H SYM  FLAGWORD
3D27H SYM  MANUALFLAG
3D28H SYM  INTERROGATEFLAG
3D29H SYM  SETPOINT
3D2DH SYM  FROMPLANT
3D31H SYM  TOPLANT
3D33H SYM  CONTVAR0
3D37H SYM  CONTVAR1
3D3BH SYM  MANIPVAR0
3D3FH SYM  MANIPVAR1
3D43H SYM  DELTAV0
3D47H SYM  DY0
3D4BH SYM  DY1
3D4FH SYM  LOLEVEL
3D51H SYM  HILEVEL
3D53H SYM  MAXLEVEL
3D55H SYM  MINLEVEL
3D57H SYM  SAT
3D58H SYM  ERR
3D59H SYM  CONSTANT
3D5DH SYM  C1
3D61H SYM  SETERROR0
3D65H SYM  SETERROR1
3D69H SYM  INCREMENT
3D6DH SYM  L
3E35H SYM  DV
3EFDH SYM  FILTGAIN
3F01H SYM  R2
3F05H SYM  R1
3F09H SYM  ONE
3F0DH SYM  TLD
3F11H SYM  TLG
3F15H SYM  SAMPLINGPERIOD
3F19H SYM  R2SUB1
3F1DH SYM  GAMMA
3F21H SYM  HUNDRED
3F25H SYM  PLANTGAIN
3F29H SYM  DYPRED
3F2DH SYM  ERRPRED
3F31H SYM  SAMPLETIME
3F32H SYM  MANUALCOUNT
3F33H SYM  SAMPLECOUNT
3F35H SYM  RAMPFLAG
3F36H SYM  ENABLEMANUAL
3F37H SYM  ENDCOUNT
3F39H SYM  SUML
3F3DH SYM  ARRAYLENGTH
3F3EH SYM  PARMNO
3F3FH SYM  SPECIFY
```

```
3F50H SYM  TTYTIME
3F51H SYM  TTYUP
0400H SYM  MESSAGE
0417H SYM  STATEMENT
3F52H SYM  I
7FF6H SYM  DAC1LO
7FF7H SYM  DAC1HI
7FF4H SYM  DAC2LO
7FF5H SYM  DAC2HI
7FF0H SYM  INITL
7FFAH SYM  MUXADR
7FF9H SYM  GAINSEL
3C3DH SYM  LOC3C3D
3C3EH SYM  LOC3C3E
3F53H SYM  A
3F55H SYM  B
3F57H SYM  CHANGETIME
0429H SYM  SELFTUNECONTROL
0429H SYM  SETUP
048CH SYM  MODELVARS
0822H SYM  BEGINLOOP
08A0H SYM  ENDLOOP
      MOD  INTERRUPTMODULE
7FE8H SYM  MEMORY
7FFCH SYM  STATUS
7C00H SYM  INTERRUPTPROCESSOR
3D00H SYM  NUMBER
3D01H SYM  I
7CFCH SYM  ENOUGH
      MOD  EXTRAVARIABLEMODULE
7FE8H SYM  MEMORY
7D84H SYM  ASCOUT
3D02H SYM  THING
7D88H SYM  CO
7D9AH SYM  NUMOUT
3D03H SYM  AMOUNT
3D05H SYM  WIDTH
3D06H SYM  I
3D07H SYM  CHARS
7D73H SYM  DIGITS
7E42H SYM  EXTRAVARIABLE
7FF4H SYM  DAC2LO
7FF5H SYM  DAC2HI
3D08H SYM  LEDSERR
3D0AH SYM  LEDS
3D0BH SYM  REFERENCE
7D7DH SYM  DIGS
3D0DH SYM  I
3D0EH SYM  P
7EB7H SYM  ERROPLED
7F0BH SYM  PARMOUT
      MOD  TERPOG
02B6H SYM  MONITO
08A6H SYM  BACK
08A5H SYM  MONINS
      MOD  MANUALMODEMODULE
41A0H SYM  MEMORY
08A7H SYM  MANUALMODE
3F58H SYM  FP
      MOD  ANALOGUEOUTMODULE
41A0H SYM  MEMORY
095DH SYM  ANALOGUEOUT
3F5AH SYM  TEMP
7FF6H SYM  DAC1LO
7FF7H SYM  DAC1HI
```

```
3F50H SYM  TTYTIME
3F51H SYM  TTYUP
0400H SYM  MESSAGE
0417H SYM  STATEMENT
3F52H SYM  I
7FF6H SYM  DAC1LO
7FF7H SYM  DAC1HI
7FF4H SYM  DAC2LO
7FF5H SYM  DAC2HI
7FF0H SYM  INITL
7FFAH SYM  MUXADR
7FF9H SYM  GAINSEL
3C3DH SYM  LOC3C3D
3C3EH SYM  LOC3C3E
3F53H SYM  A
3F55H SYM  B
3F57H SYM  CHANGETIME
0429H SYM  SELFTUNECONTROL
0429H SYM  SETUP
048CH SYM  MODELVARS
0822H SYM  BEGINLOOP
08A0H SYM  ENDLOOP
      MOD  INTERRUPTMODULE
7FE8H SYM  MEMORY
7FFCH SYM  STATUS
7C00H SYM  INTERRUPTPROCESSOR
3D00H SYM  NUMBER
3D01H SYM  I
7CFCH SYM  ENOUGH
      MOD  EXTRAVARIABLEMODULE
7FE8H SYM  MEMORY
7D84H SYM  ASCOUT
3D02H SYM  THING
7D88H SYM  CO
7D9AH SYM  NUMOUT
3D03H SYM  AMOUNT
3D05H SYM  WIDTH
3D06H SYM  I
3D07H SYM  CHARS
7D73H SYM  DIGITS
7E42H SYM  EXTRAVARIABLE
7FF4H SYM  DAC2LO
7FF5H SYM  DAC2HI
3D08H SYM  LEDSERR
3D0AH SYM  LEDS
3D0BH SYM  REFERENCE
7D7DH SYM  DIGS
3D0DH SYM  I
3D0EH SYM  P
7EB7H SYM  ERROFLED
7F0BH SYM  PARMOUT
      MOD  TERPOG
02B6H SYM  MONITO
08A6H SYM  BACK
08A5H SYM  MONINS
      MOD  MANUALMODEMODULE
41ADH SYM  MEMORY
08A7H SYM  MANUALMODE
3F58H SYM  FF
      MOD  ANALOGUEOUTMODULE
41ADH SYM  MEMORY
095DH SYM  ANALOGUEOUT
3F5AH SYM  TEMP
7FF6H SYM  DAC1LO
7FF7H SYM  DAC1HI
```

```
       MOD  AUTOMATICMODEMODULE
41ACH  SYM  MEMORY
0A0BH  SYM  AUTOMATICMODE
3F5CH  SYM  TENS
3F5DH  SYM  UNITS
3F5EH  SYM  FF
3F60H  SYM  I
       MOD  ALARMMODULE
41ADH  SYM  MEMORY
0B87H  SYM  FLASH
       MOD  INCONTVARIABLEMODULE
41ADH  SYM  MEMORY
0B8DH  SYM  INPUTCONTROLLEDVARIABLE
7FFEH  SYM  ADCHI
7FFDH  SYM  ADCLO
7FFCH  SYM  STATUS
7FFBH  SYM  CNVCMD
       MOD  DATASHUFFLEMODULE
41ADH  SYM  MEMORY
0BAEH  SYM  DATASHUFFLE
3F61H  SYM  I
3F62H  SYM  I1
       MOD  PREDICTANDUPDATEMODULE
41ADH  SYM  MEMORY
0C67H  SYM  PREDICTANDUPDATE
3F63H  SYM  I
       MOD  MANIPVARIABLECALCMODULE
41ADH  SYM  MEMORY
0DFDH  SYM  MANIPVARIABLECALC
3F64H  SYM  I
3F65H  SYM  Q

MEMORY MAP OF MODULE BAD
READ FROM FILE :F1:BAD.MOD
WRITTEN TO FILE :F1:BAD
MODULE START ADDRESS 0426H

START   STOP LENGTH REL NAME

0038H  003AH     3H  A  ABSOLUTE
0400H  1FE8H  1EE9H  B  CODE
3D0FH  3F68H   25AH  B  DATA
4120H  41ACH    8DH  B  STACK
41ADH  F6BFH  B513H  B  MEMORY
7C00H  7FE7H   3E8H  A  ABSOLUTE  (MEMORY OVERLAP FROM 7C00H THROUGH 7FE7H)
```

APPENDIX SECTION C

Controller Simulation Software Listing

Three ASCL (Automatic Continuous Simulation Language) programs, used for simulating the self tuner are listed below. Program I was the program used to simulate self tuning and control. The following points should be noted:-

1) The actual process coefficients L(I) are listed as calculated from an open loop step test done prior to the self tuning run. These are used purely for ease of reference. The starting values for this run are all L(I) = 0,001 I = 1,N.

2) The form of the self tuner used here is the absolute form, i.e. not the incremental version used for the actual implementation for practical reasons. Here, the model equation is of the same format as the incremental version. However, the controller equation differs somewhat in appearance. Nevertheless, it is the same controller as derived previously in Appendix Section A in absolute form.

Program II is a closed loop unit change in set point test. Here the self tuning has been removed and only the controller is present. The coefficients used are those arrived at by self tunin using Program I until the coefficients appeared stable.

Program III is identical to Program II except that the coefficients used are the ideal coefficients calculated from an open loop step test. The objective here is a comparison between the 'ideal' case (Program III) and the self tuned case (Program II).

```
/P TROL
/FBHSLFB JCB 7,CLASS=(,MSGLEVEL=(2,0)
*ROUTE  PRINT HOLD
*JOBPARM F=9999
/ EXEC ACSLCLG,TIME.GL=5
/SYSIN DD *
ROGRAM SELF TUNING CONTROLLER
'THIS PROGRAM CONTROLS A                  *
* 3RD ORDER PLANT VIA:     -
      * THE TU TSING ALGORITHM. *
   * KP IS PLANT STEADY STATE GAIN *
   * KC IS CONTROLLER GAIN WHICH *
  * CONTROLS SPEED OF RESPONSE *
   * Q COEFS ARE DERIVED FROM *
   * STEP RESPONSE OF PLANT+FILTER *
* NOTE:                                   *
*       A)PREFILTER TLDS+1/TLGS+1          *
*           INCLUDED FOR STABILITY        *
 * IN Z DOMAIN THIS IS *
 *                      *
 *   GF= (R1-Z**-1)*(TLD/TLG)/(R2-Z**-1) *
 *                     *
 *   TLD/TLG ENSURES UNITY GAIN *
 *R1=1+ST/TLD : R2=1+ST/TLG        *
NITIAL
NTEGER N,COUNT,I,J
ONSTANT N=80,T1=15.0,WEIGHT=1.0
ONSTANT T2=3.0,K=1.7,COUNT=1
ONSTANT ST=1.0,TMAX=3000.0
ONSTANT TLD=100.0,TLG=15.0
RRAY L(80),V(80)
RRAY M1(80),M2(80),M3(80),M4(80)
RRAY G(80)
RRAY DUNCE(80)

 L(1)=0.00000
 L(2)=2.3475E-04
 L(3)=0.00122
 L(4)=0.00205
 L(5)=0.00256
 L(6)=0.00286
 L(7)=0.00300
 L(8)=0.00304
 L(9)=0.00301
 L(10)=0.00293
 L(11)=0.00282
 L(12)=0.00270
 L(13)=0.00256
 L(14)=0.00243
 L(15)=0.00229
 L(16)=0.00215
 L(17)=0.00202
 L(18)=0.00190
 L(19)=0.00178
 L(20)=0.00166
 L(21)=0.00155
```

```
      L(26)=0.00111
      L(27)=0.00103
      L(28)=9.6512E-04
      L(29)=9.0096E-04
      L(30)=8.4119E-04
      L(31)=7.8526E-04
      L(32)=7.3301E-04
      L(33)=6.8429E-04
      L(34)=6.3875E-04
      L(35)=5.9625E-04
      L(36)=5.5651E-04
      L(37)=5.1946E-04
      L(38)=4.8488E-04
      L(39)=4.5261E-04
      L(40)=4.2232E-04
      L(41)=3.9437E-04
      L(42)=3.6799E-04
      L(43)=3.4336E-04
      L(44)=3.2068E-04
      L(45)=2.9901E-04
      L(46)=2.7924E-04
      L(47)=2.6062E-04
      L(48)=2.4329E-04
      L(49)=2.2708E-04
      L(50)=2.1182E-04
      L(51)=1.9783E-04
      L(52)=1.8461E-04
      L(53)=1.7228E-04
      L(54)=1.6071E-04
      L(55)=1.5014E-04
      L(56)=1.3994E-04
      L(57)=1.3077E-04
      L(58)=1.2188E-04
      L(59)=1.1393E-04
      L(60)=1.0622E-04
      L(61)=9.9223E-05
      L(62)=9.2547E-05
      L(63)=8.6352E-05
      L(64)=8.0556E-05
      L(65)=7.5258E-05
      L(66)=7.0192E-05
      L(67)=6.5453E-05
      L(68)=6.67203E-05
      L(69)=5.7057E-05
      L(70)=5.3097E-05
      L(71)=4.9750E-05
      L(72)=4.6372E-05
      L(73)=4.3245E-05
      L(74)=4.0412E-05
      L(75)=3.7602E-05
      L(76)=3.5230E-05
      L(77)=3.2782E-05
      L(78)=3.0668E-05
      L(79)=2.8499E-05
      L(80)=2.6750E-05
        DO 989 I=1,N
        L(I)=0.001
  989..CONTINUE
        * INITIAL PLANT INPUT-FROM ESTIMATES *
        * ASSUMING INITIALLY SET PT = 1 *
        * THEREFORE PLANT INPUT = 1/SSG(E'T.) *
        MANIC=0.0
        DO 985 I=1,N
```

```
 2..CONTINUE
    YDEL=1.7
    PRERR=0.0
    M2(1)=0.0
    M4(1)=0.0
    GFILT=TLD/TLG
    R1=1.000+ST/TLD
    R2=1.000+ST/TLG
    MANVAR=0.1

    CINTERVAL CINT=1.0
ND$'OF INITIAL'
YNAMIC

ERIVATIVE

    NOS=OU(0.01,0.0,1.0)
    MANIP=MANVAR+NOS
   K=1.7
    YA=K*REALPL(20.0,MANIP,1.0)
    YB=REALPL(T1,YA,1.7)
   ' BAND LIMITED NOISE-1HZ'
    YINT=REALPL(T2,YB,1.7)
    ' ACTUAL PLANT AT 1.7 AT T=0 '
    ' LIMIT NOISE TO SYSTEM CAPABILITY '
    SP=1.0
    OUTERR=SP-YINT
ND$'OF DERIVATIVE'

   'INTEGRATES FORWARD TO NOW'
   'IE NEED TO PREDICT YINT'
  'CALCS ARE ONLY DONE AT SAMPLE INTERVALS'


    IF(COUNT.NE.1) GO TO 1000
    'SAMPLE ONCE EVERY ST'
    COUNT=0

    SUM=0.0
    DO 57 I=1,N
    SUM=SUM+L(I)*V(I)$' SUMMATION L(I)*V(I) '
57..CONTINUE


  ' NOW PREDICT PLANT OUTPUT '

    YPRED=(1.0/(GFILT*R1))*(YDEL*GFILT+SUM)
    YDEL=YINT
    PREDER=YINT-YPRED

    ' UPDATE PARMS ACCORDING TO ERROR '

    GAMMA=0.001
    DO 58 I=1,N
    L(I)=L(I)+(GAMMA)*(V(I)*PREDER)
 58..CONTINUE
    WEIGHT=1.0

    ' CALCULATE SSG **-1 '
    SUML=0.0
    DO 59 I=1,N
    SUML=SUML+L(I)
59..CONTINUE
```

```
    DO 61 I=1,N
    Q(I)=L(I)
  61..CONTINUE
    Q(1)=Q(1)+1.0/KP

    ' AND CONTROL '
    DO 60 I=2,N
      M3(I)=Q(I)*V(I)
      M4(I)=M3(I)+M4(I-1)
    60..CONTINUE
    M5=M4(N)+(Q(1)*V(1))$'SUMMATION Q(I)*V(I)'

    M6=R1*OUTERR-PRERR
    PRERR=OUTERR
    MANVAR=(KP/R2)*M5+(KC*GFILT/R2)*M6

    'DELAY ALL VALUES BY ST'
    DO 28 I=1,N
    DUNCE(I)=V(I)
 28..CONTINUE
    DO 30 I=2,N
    V(I)=DUNCE(I-1)

    30..CONTINUE

    V(1)=MANVAR


 1000..CONTINUE

    COUNT=COUNT+1
    TERMT(T.GE.TMAX)
NDS'OF DYNAMIC'
ERMINAL

NDS' OF TERMINAL '

NDS'OF PROGRAM'
/LKED.SYSLIB DD
/             DD
/             DD DSN=SYS2.PLOTLIB,DISP=SHR
/LKED.SYSIN DD *
 INCLUDE SYSLIB(CAPPLT)
/GO.PLOTAXYZ DD UNIT=PLOTOUT,SPACE=(CYL,(3,3))
/GO.SYSIN DD *
    SET NSTP=1
ET TITLE='SELF ID & CONTROL'
ET CALPLT=.TRUE.,PRNPLT=.FALSE.,XINCPL=9.0,YINCPL=6.0
 PREPAR T,L(10),OUTERR,YINT
    START
    PRINT T,L(10),YINT,'NCIPRN'=10
    DISPLY L,KP,YINT,MANVAR,NOS
    PLOT 'XAXIS'=T,'XLO'=0.0,'XHI'=3000.0,L(10)
    PLOT 'XAXIS'=T,'XLO'=0.0,'XHI'=2998.0,YINT
    STOP
/
```

PRG II

```
        **********ADVANCED  CONTINUCUS  SIMULATION  LANGUAGE**********          PAGE  1
  ACSL TRANSLATOR VERSION 4 LEVEL 50  80/337  14.24.42

PROGRAM  ESTIMATES TEST PROGRAM          *
 *THIS PROGRAM CONTROLS A
 * 3RD ORDER PLANT VIA:     *
     * THE TU TSING ALGORITHM. *
   * KP IS PLANT STEADY STATE GAIN *
   * KC IS CONTROLLER GAIN WHICH *
  * CONTROLS SPEED OF RESPONSE *
   * Q COEFS ARE DERIVED FROM *
   * STEP RESPONSE OF PLANT+FILTER *
                                   *
 * NCTE:                              *
 *       A)PREFILTER TLDS+1/TLGS+1
 *          INCLUDED FOR STABILITY    *
  * IN Z DCMAIN THIS IS *
  *                      *
  *  GF= (R1-Z**-1)*(TLD/TLG)/(R2-Z**-1) *
  *                      *
  *  TLD/TLG ENSURES UNITY GAIN *
  *R1=1+ST/TLD : R2=1+ST/TLG     *
INITIAL
INTEGER N.COUNT.I.J
CONSTANT N=50.T1=15.0.WEIGHT=1.0
CONSTANT T2=3.0.K=1.7.COUNT=1
CONSTANT ST=1.0.TMAX=30C.0
CONSTANT TLD=100.0.TLG=15.0
ARRAY L(50).V(50)
ARRAY M1(50).M2(50).M3(50).M4(50)
ARRAY Q(50)
ARRAY DUNCE(50)

   L(1)=0.0C000
   L(2)=3.99E-04
   L(3)=0.00208
   L(4)=0.00348
   L(5)=0.0C351
   L(6)=0.00382
   L(7)=0.00397
   L(8)=0.00402
   L(9)=0.00400
   L(10)=0.00393
   L(11)=0.00383
   L(12)=0.00372
   L(13)=0.00358
   L(14)=0.C0345
   L(15)=0.00332
   L(16)=0.C0318
   L(17)=0.C0305
   L(18)=0.C0292
   L(19)=0.C0280
   L(20)=0.C0268
   L(21)=0.00257
   L(22)=0.00246
   L(23)=0.C0237
   L(24)=0.C0228
   L(25)=0.C0219
```

PRG II


```
PROGRAM  ESTIMATES TEST PROGRAM
 'THIS PROGRAM CONTROLS A                   '
 ' 3RD ORDER PLANT VIA:   '
      ' THE TU TSING ALGORITHM. '
    ' KP IS PLANT STEADY STATE GAIN '
    ' KC IS CONTROLLER GAIN WHICH '
   ' CONTROLS SPEED OF RESPONSE '
    ' Q COEFS ARE DERIVED FROM '
    ' STEP RESPONSE OF PLANT+FILTER '
 ' NCTE:                                     '
 '        A)PREFILTER TLDS+1/TLGS+1             '
 '            INCLUDED FOR STABILITY         '
  ' IN Z DCMAIN THIS IS '
  '                     '
  '  GF= (R1-Z**-1)*(TLD/TLG)/(R2-Z**-1) '
  '                     '
  '  TLD/TLG ENSURES UNITY GAIN '
  'R1=1+ST/TLD : R2=1+ST/TLG        '
INITIAL
INTEGER N,COUNT,I,J
CONSTANT N=50,T1=15.0,WEIGHT=1.0
CONSTANT T2=3.0,K=1.7,COUNT=1
CONSTANT ST=1.0,TMAX=300.0
CONSTANT TLD=100.0,TLG=15.0
ARRAY L(50),V(50)
ARRAY M1(50),M2(50),M3(50),M4(50)
ARRAY Q(50)
ARRAY DUNCE(50)

   L(1)=0.00000
   L(2)=3.99E-04
   L(3)=0.00208
   L(4)=0.00348
   L(5)=0.00351
   L(6)=0.00382
   L(7)=0.00397
   L(8)=0.00402
   L(9)=0.00400
   L(10)=0.00393
   L(11)=0.00383
   L(12)=0.00372
   L(13)=0.00358
   L(14)=0.00345
   L(15)=0.00332
   L(16)=0.00318
   L(17)=0.00305
   L(18)=0.00292
   L(19)=0.00280
   L(20)=0.00268
   L(21)=0.00257
   L(22)=0.00246
   L(23)=0.00237
   L(24)=0.00228
   L(25)=0.00219
```

```
L(26)=0.00211
L(27)=0.00202
  L(28)=0.00195
  L(29)=0.00189
  L(30)=0.00182
  L(31)=0.00176
  L(32)=0.00171
  L(33)=0.00166
  L(34)=0.00161
  L(35)=0.00156
  L(36)=0.00152
  L(37)=0.00149
  L(38)=0.00145
  L(39)=0.00142
  L(40)=0.00139
  L(41)=0.00137
  L(42)=0.00134
  L(43)=0.00132
  L(44)=0.00130
  L(45)=0.00129
  L(46)=0.00127
  L(47)=0.00126
  L(48)=0.00125
  L(49)=0.00124
  L(50)=0.00123

 GO TO 996
L(51)=1.9783E-04
L(52)=1.8461E-04
L(53)=1.7228E-04
L(54)=1.6071E-04
L(55)=1.5014E-04
L(56)=1.3994E-04
L(57)=1.3077E-04
L(58)=1.2188E-04
L(59)=1.1393E-04
L(60)=1.0622E-04
L(61)=9.9223E-05
L(62)=9.2547E-05
L(63)=8.6352E-05
L(64)=8.0556E-05
L(65)=7.5258E-05
L(66)=7.0192E-05
L(67)=6.5453E-05
L(68)=6.67203E-05
L(69)=5.7057E-05
L(70)=5.3097E-05
L(71)=4.9750E-05
L(72)=4.6372E-05
L(73)=4.3245E-05
L(74)=4.0412E-05
L(75)=3.7602E-05
L(76)=3.5230E-05
L(77)=3.2782E-05
```

```
 L(78)=3.0668E-05
 L(79)=2.8499E-05
 L(80)=2.6750E-05
   996..CONTINUE
   DO 989 I=1,N
    L(I)=1.0*L(I)
989..CONTINUE
   DO 2 I=1,N
    V(I)=0.0
  2..CONTINUE
   YDEL=1.0
   PRERR=0.0
   M2(1)=0.0
   M4(1)=0.0
   GFILT=TLD/TLG
   R1=1.000+ST/TLD
   R2=1.000+ST/TLG
   MANVAR=0.0

   CINTERVAL CINT=1.0
ENDS'OF INITIAL'
DYNAMIC

   ' THE PLANT IS INITIALLY AT ZERO STATE '
   ' BUT UNIT SET POINT APPLIED AT T=0.0  '
   ' FOR COMPARISON WITH IDEAL CASE       '
DERIVATIVE

   MANIP=MANVAR
   K=1.7
   YA=K*REALPL(100.0,MANIP,0.0)
   YB=REALPL(T1,YA,0.0)
  ' BAND LIMITED NOISE-1HZ'
   YINT=REALPL(T2,YB,0.0)
   ' LIMIT NOISE TO SYSTEM CAPABILITY '
   SP=1.0
   OUTERR=SP-YINT
ENDS'OF DERIVATIVE'

   'INTEGRATES FORWARD TO NOW'
   'IE NEED TO PREDICT YINT'
  'CALCS ARE ONLY DONE AT SAMPLE INTERVALS'


    IF(COUNT.NE.1) GO TO 1000
    'SAMPLE ONCE EVERY ST'
    COUNT=0
    L(1)=0.0
    L(2)=3.990E-04
    L(3)=0.00208


    ' CALCULATE SSG **-1 '
    SUML=0.0
```

```
    DO 59 I=1,N
    SUML=SUML+L(I)
 59..CONTINUE

    KP=(R2-1.0)/SUML
    KC=KP

    ' TRANSFER PARMS TO CONTROLLER '
    DO 61 I=1,N
    Q(I)=L(I)
  61..CONTINUE
    Q(1)=Q(1)+1.0/KP

    ' AND CONTROL '
    DO 60 I=2,N
      M3(I)=Q(I)*V(I)
      M4(I)=M3(I)+M4(I-1)
    60..CONTINUE
    M5=M4(N)+(Q(1)*V(1))$'SUMMATION Q(I)*V(I)'

    M6=R1*CUTERR-PRERR
    PRERR=CUTERR
    MANVAR=(KP/R2)*M5+(KC*GFILT/R2)*M6

    'DELAY ALL VALUES BY ST'
    DO 28 I=1,N
    DUNCE(I)=V(I)
 28..CONTINUE
    DO 30 I=2,N
    V(I)=DUNCE(I-1)

    30..CONTINUE

    V(1)=MANVAR


 1000..CONTINUE

    COUNT=COUNT+1
    TERMT(T.GE.TMAX)
ENDS'OF DYNAMIC'
TERMINAL

ENDS' OF TERMINAL '

ENDS'OF PROGRAM'
```

PRGM III

```
          **********ADVANCED  CONTINUOUS  SIMULATION  LANGUAGE*********
     ACSL TRANSLATOR VERSION 4 LEVEL 5D  80/337  15.52.01              PAGE  1

PROGRAM  ESTIMATES TEST PROGRAM
 'THIS PROGRAM CONTROLS A
 ' 3RD ORDER PLANT VIA:    '
      ' THE TU TSING ALGORITHM. '
    ' KP IS PLANT STEADY STATE GAIN '
    ' KC IS CONTROLLER GAIN WHICH '
   ' CONTRCLS SPEED OF RESPONSE '
    ' Q COEFS ARE CERIVED FROM '
    ' STEP RESPONSE OF PLANT+FILTER '
 ' NCTE:                                '
 '        A)PREFILTER TLDS+1/TLGS+1        '
 '           INCLUDED FOR STABILITY     '
  ' IN Z DCMAIN THIS IS '
  '                     '
  '  GF= (R1-Z**-1)*(TLD/TLG)/(R2-Z**-1) '
  '                     '
  '  TLD/TLG ENSURES UNITY GAIN '
  'R1=1+ST/TLD : R2=1+ST/TLG      '
INITIAL
INTEGER N.COUNT.I.J
CONSTANT N=50.T1=15.0.WEIGHT=1.0
CONSTANT T2=3.0.K=1.7.COUNT=1
CONSTANT ST=1.0.TMAX=300.0
CONSTANT TLD=100.0.TLG=15.0
ARRAY L(50).V(50)
ARRAY M1(50).M2(50).M3(50).M4(50)
ARRAY Q(50)
ARRAY DUNCE(50)

  L(1)=0.0C000
  L(2)=2.3475E-04
  L(3)=0.00122
  L(4)=0.0C205
  L(5)=0.00256
  L(6)=0.0C286
  L(7)=0.0C300
  L(8)=0.00304
  L(9)=0.0C301
  L(10)=0.C0293
  L(11)=0.C0282
  L(12)=0.00270
  L(13)=0.00256
  L(14)=0.00243
  L(15)=0.C0229
  L(16)=0.C0215
  L(17)=0.C0202
  L(18)=0.C0190
  L(19)=0.00178
  L(20)=0.C0166
  L(21)=0.C0155
  L(22)=0.00145
  L(23)=0.C0136
  L(24)=0.00127
  L(25)=0.C0119
```

```
L(26)=0.00111
L(27)=0.00103
L(28)=9.6512E-04
L(29)=9.0096E-04
L(30)=8.4119E-04
L(31)=7.8526E-04
L(32)=7.3301E-04
L(33)=6.8429E-04
L(34)=6.3875E-04
L(35)=5.9625E-04
L(36)=5.5651E-04
L(37)=5.1946E-04
L(38)=4.8488E-04
L(39)=4.5261E-04
L(40)=4.2232E-04
L(41)=3.9437E-04
L(42)=3.6799E-04
L(43)=3.4336E-04
L(44)=3.2068E-04
L(45)=2.9901E-04
L(46)=2.7924E-04
L(47)=2.6062E-04
L(48)=2.4329E-04
L(49)=2.2708E-04
L(50)=2.1182E-04

 GO TO 996
L(51)=1.9783E-04
L(52)=1.8461E-04
L(53)=1.7228E-04
L(54)=1.6071E-04
L(55)=1.5014E-04
L(56)=1.3994E-04
L(57)=1.3077E-04
L(58)=1.2188E-04
L(59)=1.1393E-04
L(60)=1.0622E-04
L(61)=9.9223E-05
L(62)=9.2547E-05
L(63)=8.6352E-05
L(64)=8.0556E-05
L(65)=7.5258E-05
L(66)=7.0192E-05
L( 7)=6.5453E-05
L(68)=6.67203E-05
L(69)=5.7057E-05
L(70)=5.3097E-05
L(71)=4.9750E-05
L(72)=4.6372E-05
L(73)=4.3245E-05
L(74)=4.0412E-05
L(75)=3.7602E-05
L(76)=3.5230E-05
L(77)=3.2782E-05
```

```
  L(78)=3.0668E-05
  L(79)=2.8499E-05
  L(80)=2.6750E-05
     996..CONTINUE
     DO 989 I=1,N
      L(I)=1.7*L(I)
 989..CONTINUE
     DO 2 I=1,N
      V(I)=0.0
  2..CONTINUE
     YDEL=1.0
     PRERR=0.0
     M2(1)=0.0
     M4(1)=0.0
     GFILT=TLD/TLG
     R1=1.000+ST/TLD
     R2=1.000+ST/TLG
     MANVAR=0.0

     CINTERVAL CINT=1.0
ENDS'OF INITIAL'
DYNAMIC

     ' THE PLANT IS INITIALLY AT ZERO STATE '
     ' BUT UNIT SET POINT APPLIED AT T=0.0  '
                                            '
DERIVATIVE

     MANIP=MANVAR
     K=1.7
     YA=K*REALPL(100.0,MANIP,0.0)
     YB=REALPL(T1,YA,0.0)
    ' BAND LIMITED NOISE-1HZ'
     YINT=REALPL(T2,YB,0.0)
     ' LIMIT NOISE TO SYSTEM CAPABILITY '
     SP=1.0
     OUTERR=SP-YINT
ENDS'OF DERIVATIVE'

    'INTEGRATES FORWARD TO NOW'
    'IE NEED TO PREDICT YINT'
   'CALCS ARE ONLY DONE AT SAMPLE INTERVALS'


     IF(COUNT.NE.1) GO TO 1000
     'SAMPLE ONCE EVERY ST'
     COUNT=0




     ' CALCULATE SSG **-1 '
     SUML=0.0
```

```
   CO 59 I=1,N
   SUML=SUML+L(I)
59..CONTINUE

   KP=(R2-1.0)/SUML
   KC=KP

   ' TRANSFER PARMS TO CONTROLLER '
   CO 61 I=1,N
   Q(I)=L(I)
 61..CONTINUE
   Q(1)=Q(1)+1.0/KP

   ' AND CONTROL '
   CO 60 I=2,N
     M3(I)=Q(I)*V(I)
     M4(I)=M3(I)+M4(I-1)
   60..CONTINUE
   M5=M4(N)+(Q     *V(1))$'SUMMATION Q(I)*V(I)'

   M6=R1*OUTERR-PRERR
   PRERR=OUTERR
   MANVAR=(KP/R2)*M5+(KC*GFILT/R2)*M6

   'DELAY ALL VALUES BY ST'
   CO 28 I=1,N
   DUNCE(I)=V(I)
 28..CONTINUE
    CO 30 I=2,N
    V(I)=DUNCE(I-1)

    30..CONTINUE

    V(1)=MANVAR

 1000..CONTINUE

    COUNT=COUNT+1
    TERMT(T.GE.TMAX)
ENDS'OF DYNAMIC'
TERMINAL

ENDS' OF TERMINAL '

ENDS'OF PROGRAM'
```

APPENDIX SECTION D

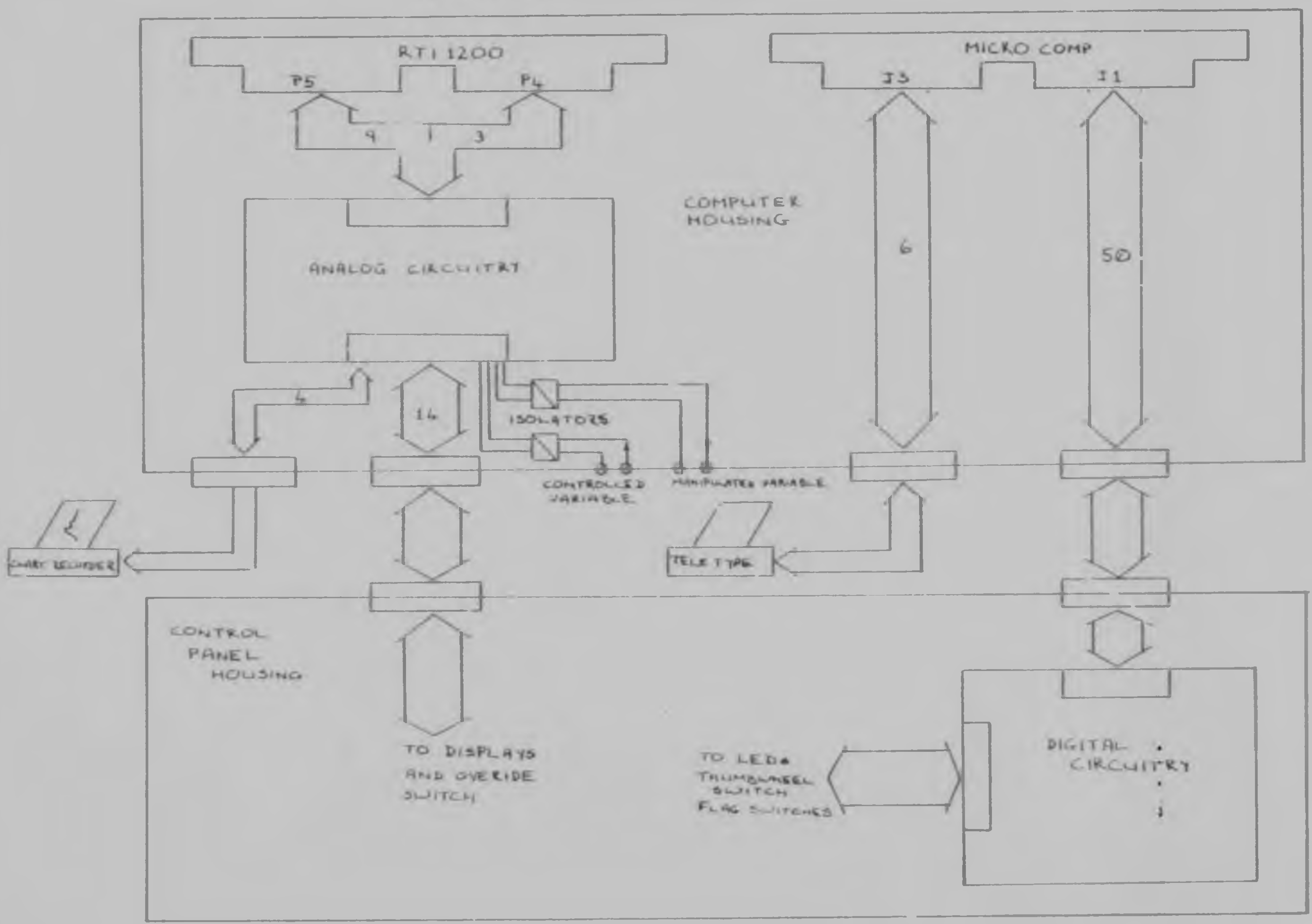
Circuit Diagrams

An overall connection diagram is shown in Figure A-1. This shows the essential connection between the four boards containing analog and digital electronic circuitry and various peripherals. This figure must be read in conjunction with Figures A-2 and A-3 which respectively depict the analog and digital circuits built specifically for this project.

Figure A-4 is the system interconnection diagram. Here 4-20 mA controlled variable as measured was connected to the input stage of the existing PI controller. This enabled the existing display and recorder to be used for this variable.

INTERNAL CONFIGURATION

Fig A1



PIN CONNECTIONS AND RELATED INFORMATION

EDGE CONNECTOR J1 | SBC 80/10 BOARD | 50 PIN

Pin No.	Function
1	Least significant digit
3	Thumb Wheel Switch
5	
7	
9	Most significant Digit
11	Thumb Wheel Switch
13	
15	
25	Interrogate Flag Switch
29	MAN/AUT Flag Switch
19	Display Flag Switch
17	Ramp Flag Switch (LSB)
21	Ramp Flag Switch (MSB)
41	Display Diodes (LSB)
45	(LSB+1)
47	
39	
37	
35	(MSB)
33	Controller Skipped Switch

EDGE CONNECTOR J1 (Continued) | SBC 80/10 BOARD | 50 PIN

Pin No.	Function
27	
28	
23	+ 5V Supply
24	
31	
32	
43	Shield Connection
44	To Earth
49	
50	

Note:

1) All others are tied to ground.

2) 3M mating connector 3415-0001.

EDGE CONNECTOR J3	SBC 80/10 BOARD	26 PIN TO TELETYPE
RS232C Connect	Pin No.	Function
12	23	TTY RX DATA
24	22	TTY RX DATA RETURN
25	24	TTY TX DATA RETURN
13	25	TTY TX DATA

Notes:

1) 3M Mating Connector No. 3462-0001.

EDGE CONNECTOR P-5	RTI 1200 BOARD	20 PIN - ALL ANALOG SIGNALS
	Pin No.	Function
	5	Extra Channel, Voltage Out to Chart Recorder
	6	Analog Common to Chart Recorder
	8	Tied to Pin 19
	9	Analog Common
	10	Analog Common
	15	Analog Common
	16	Analog Common
	17	Manipulated Variable Voltage Out to Plant and Chart Recorder
	18	Anlog Common to Chart Recorder
	19	Tied to Pin 8 (+15V)
	20	Manipulated Variable, Current Output to Plant and Chart Recorder

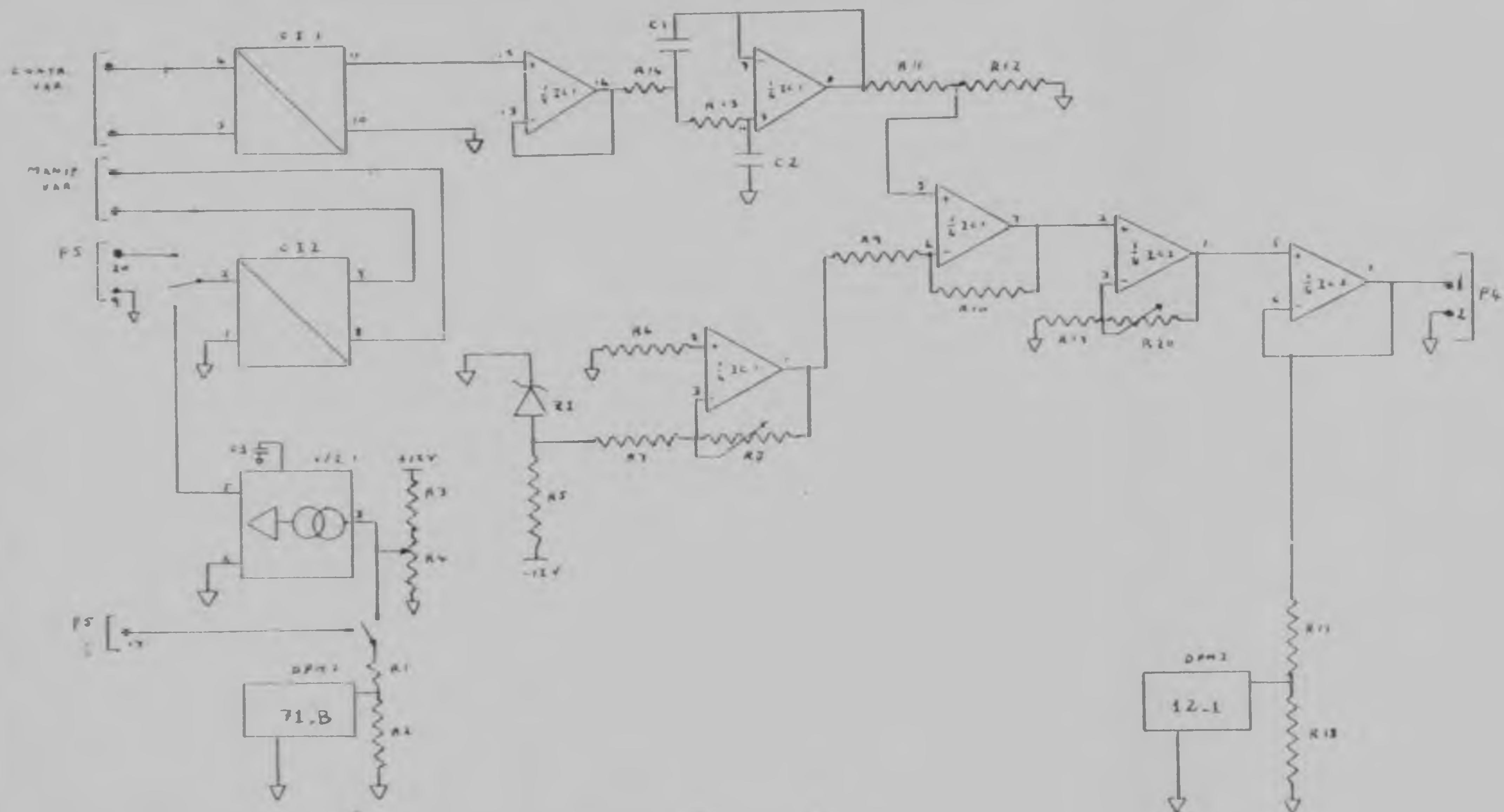
EDGE CONNECTOR P4 RTI 1200

Controlled Variable form Analog Board

Pin No.	Function
1	Signal HI
2	Signal LO
3	GND (Shield)

Notes:

1) 3M Connector 3433-1002.



ANALOG CIRCUITRY

Fig A2

Component List

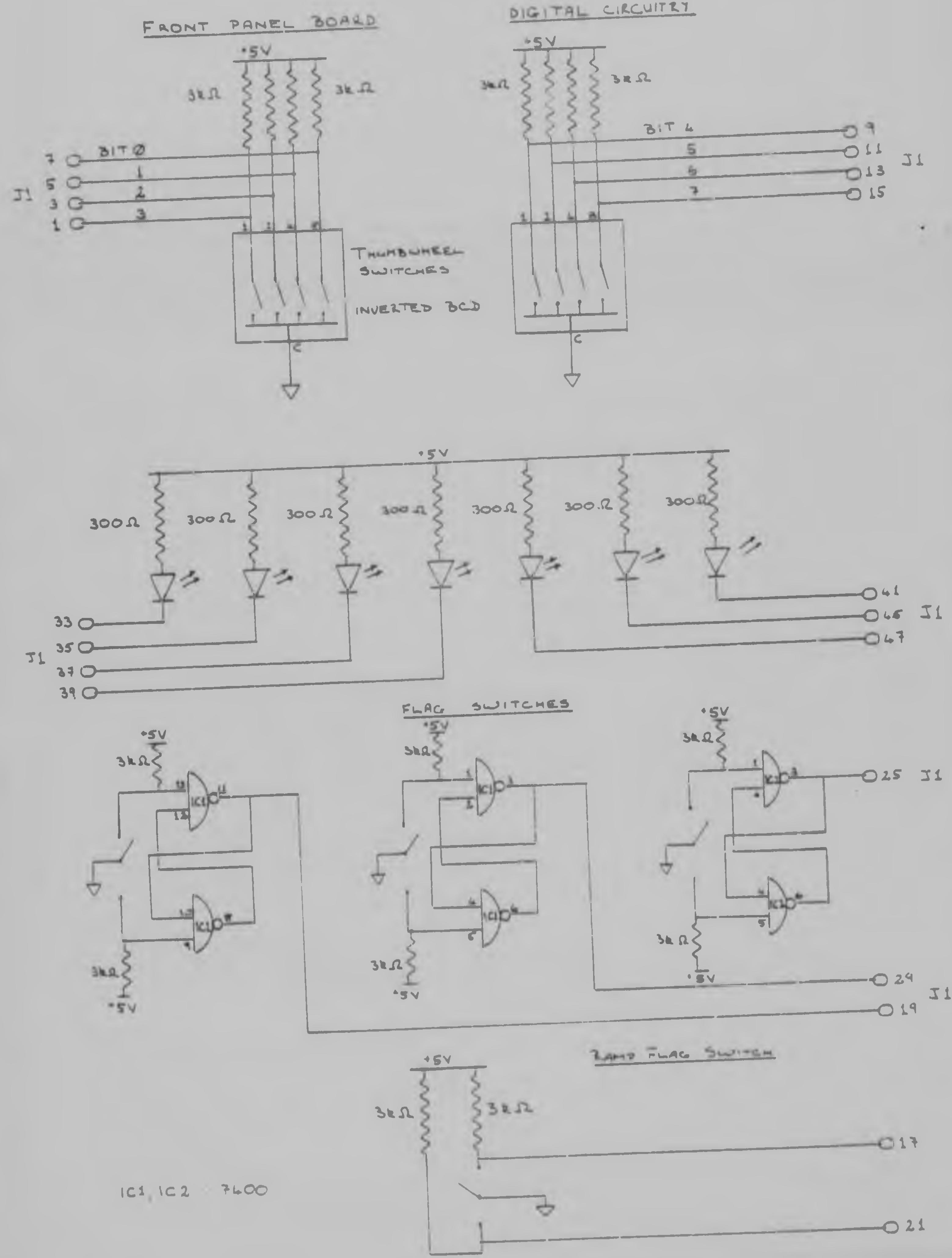
	Resistors in Kilo-ohms
R1, R17	400
R2, R18	100
R3	200
R4, R6, R9, R10, R11, R12, R13, R14, R16, R19, R21	1
R5, R15	0,25
R7	6,20
R8	10
R20	2,50

	Capacitances in Micro-Farads
C1	45
C2	22
C3	0,1

Z1	Zener Diode 5W BZX85C

IC1, IC2	TL 084C
DPM	NLS PM 3,5
ISOL XFORMER (C I)	80-111
V/I CONVERTER	AD 2B20A

FIG A3



IC1, IC2 7400

FIG A4

SCHEMATIC SYSTEM CONFIGURATION