

3 4456 0364272 5

ORNL/TM-12040

# ornl

**OAK RIDGE  
NATIONAL  
LABORATORY**

**MARTIN MARIETTA**

## A Comparison of Three Self-Tuning Control Algorithms Developed for the Bristol-Babcock Controller

P. A. Tapp

OAK RIDGE NATIONAL LABORATORY

CENTRAL RESEARCH LIBRARY

CIRCULATION SECTION

4500N ROOM 175

**LIBRARY LOAN COPY**

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this  
report, send in name with report and  
the library will arrange a loan.

UCN-7969 (3 9-77)

MANAGED BY  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Instrumentation and Controls Division

**A COMPARISON OF THREE SELF-TUNING CONTROL ALGORITHMS  
DEVELOPED FOR THE BRISTOL-BABCOCK CONTROLLER**

P. A. Tapp

Date Published—April 1992

Prepared by the  
OAK RIDGE NATIONAL LABORATORY  
Oak Ridge, Tennessee 37831-6285  
managed by  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
for the  
U.S. DEPARTMENT OF ENERGY  
under contract DE-AC05-84OR21400





## CONTENTS

LIST OF FIGURES .....	v
LIST OF TABLES .....	vii
ABSTRACT .....	ix
1. INTRODUCTION .....	1
1.1 OBJECTIVES .....	1
1.2 BRIEF OVERVIEW OF RELATED ADAPTIVE CONTROL METHODS .....	1
1.3 ORGANIZATION .....	2
2. FURTHER SELF-TUNING CONTROL BACKGROUND .....	6
2.1 PROCESS IDENTIFICATION TECHNIQUES .....	6
2.1.1 Transient-Response Analysis .....	6
2.1.2 Frequency-Response Analysis .....	8
2.1.3 Parameter Estimation Techniques .....	10
2.2 PID PARAMETER ADJUSTMENT TECHNIQUES .....	10
2.2.1 Ziegler-Nichols Methods .....	12
2.2.2 Pole-Placement Method .....	13
2.3 TWO MOST COMMON INDUSTRIALLY IMPLEMENTED DESIGNS .....	18
2.3.1 Pattern Recognition Method .....	18
2.3.2 Process Identification Method .....	19
3. THEORY OF OPERATION OF THE BBI STPI ALGORITHMS .....	20
3.1 CLOSED-LOOP CYCLING ALGORITHM THEORY .....	20
3.2 PATTERN RECOGNITION ALGORITHM THEORY .....	23
3.3 MODEL-BASED ALGORITHM THEORY .....	25
4. PROCESS SIMULATIONS .....	30
4.1 INTEGRATING PROCESS .....	30
4.2 FIRST-ORDER SYSTEM .....	32
4.3 SECOND-ORDER SYSTEM .....	32
4.4 SYSTEM WITH INITIAL INVERSE RESPONSE .....	36
4.5 SYSTEM WITH VARIABLE TIME CONSTANT AND DELAY .....	36
5. TESTING AND COMPARISON OF THE BBI STPI ALGORITHMS .....	41
5.1 DESCRIPTION OF TESTS .....	41
5.2 PERFORMANCE EVALUATIONS .....	43
5.2.1 Process Incompatibilities .....	43
5.2.2 Tuned System Performance .....	47
5.2.3 Deadtime Effects .....	48
5.2.4 Noise Effects .....	48

**CONTENTS (continued)**

5.3 ROBUSTNESS COMPARISONS .....	51
5.4 PI PARAMETER ADJUSTMENT EFFICIENCY .....	53
6. CONCLUSIONS .....	55
7. REFERENCES .....	58
8. BIBLIOGRAPHY .....	59
Appendix A: DEVELOPMENT OF ACCOL SELF-TUNING PI (STPI) CONTROL MODULE .....	61
Appendix B: THE INTEGRATED STPI MODULE/PROCESS SIMULATION ACCOL PROGRAM .....	139
Appendix C: MATLAB PERFORMANCE ANALYSIS AND PLOTTING ROUTINES .....	171
Appendix D: MATHCAD ROBUSTNESS ANALYSIS ROUTINES .....	179

## LIST OF FIGURES

Figure	
1.1	Conventional feedback controller structure . . . . . 2
1.2	Gain-scheduling controller structure . . . . . 3
1.3	Self-tuning controller structure . . . . . 3
1.4	Model-reference adaptive controller structure . . . . . 4
2.1	Typical step response of a first-order process . . . . . 7
2.2	Typical step response of a second-order process . . . . . 8
2.3	Typical sinusoidal output response to a sinusoidal input . . . . . 9
2.4	Varied output response plots for the same process with different controller gains . . . . . 11
2.5	Relay feedback controller structure . . . . . 11
2.6	Sinusoidal output response generated by a relay feedback controller . . . . . 12
2.7	Typical open-loop step-response plot . . . . . 13
2.8	Effects of dominant pole location on system performance . . . . . 15
2.9	Changing system dynamics by moving one point on the Nyquist curve . . . . . 16
3.1	Bristol-Babcock, Inc., relay feedback controller structure . . . . . 20
3.2	Output response generated by the closed-loop cycling algorithm . . . . . 21
3.3	Percent overshoot vs damping ratio for the step response of a second-order system . . . . . 23
3.4	Phase margin vs damping ratio of a second-order system . . . . . 24
3.5	Annotated output response describing the operation of the pattern recognition algorithm . . . . . 25
3.6	Open-loop step response to determine a suitable task rate for the model-based algorithm . . . . . 27

## LIST OF FIGURES (continued)

### Figure

3.7	Output response generated by the model-based algorithm pseudorandom binary sequence .....	28
4.1	Block diagram of the connections between the self-tuning proportional-integral controller and the process simulations .....	31
4.2	Unbounded output response of a pure integrating process .....	31
4.3	Output response of a first-order process for a step input .....	33
4.4	Output response of an overdamped second-order system for a step input ....	34
4.5	Output response of a critically damped second-order system for a step input .....	35
4.6	Output response of an underdamped second-order system for a step input .....	35
4.7	Step response plots of a second-order system for various values of the damping factor .....	37
4.8	Initial inverse response resulting from two opposing first-order systems to a step input .....	38
4.9	Continuous concentration control of a chemical mixing process .....	39

## LIST OF TABLES

Table		
2.1	Controller parameters and dominant dynamics obtained by Ziegler-Nichols open-loop step-response method	14
2.2	Controller parameters and dominant dynamics obtained by Ziegler-Nichols closed-loop frequency-response method	14
3.1	Ziegler-Nichols ultimate frequency-response controller parameters	22
5.1	Transfer functions of the simulated processes	42
5.2	Calculated proportional-integral parameters for the integrating process	44
5.3	Calculated proportional-integral parameters for the first-order process	44
5.4	Calculated proportional-integral parameters for the second-order process ( $\omega_n = 0.2$ )	45
5.5	Calculated proportional-integral parameters for the second-order process ( $\omega_n = 0.04$ )	45
5.6	Calculated proportional-integral parameters for the system with initial inverse response	46
5.7	Calculated proportional-integral parameters for the system with variable time constant and delay	46
5.8	Tuned system integral of the absolute value of the error response to both setpoint and load changes	49
5.9	Calculated proportional-integral parameters for the first-order process with deadtime	50
5.10	Calculated proportional-integral parameters for the first-order process with noise	51
5.11	Tuned system robustness with respect to gain and deadtime increases	54



## ABSTRACT

A brief overview of adaptive control methods relating to the design of self-tuning proportional-integral-derivative (PID) controllers is given. The methods discussed include gain scheduling, self-tuning, auto-tuning, and model-reference adaptive control systems. Several process identification and parameter adjustment methods are discussed. Characteristics of the two most common types of self-tuning controllers implemented by industry (i.e., pattern recognition and process identification) are summarized. The substance of the work is a comparison of three self-tuning proportional-plus-integral (STPI) control algorithms developed to work in conjunction with the Bristol-Babcock PID control module. The STPI control algorithms are based on closed-loop cycling theory, pattern recognition theory, and model-based theory. A brief theory of operation of these three STPI control algorithms is given. Details of the process simulations developed to test the STPI algorithms are given, including an integrating process, a first-order system, a second-order system, a system with initial inverse response, and a system with variable time constant and delay. The STPI algorithms' performance with regard to both setpoint changes and load disturbances is evaluated, and their robustness is compared. The dynamic effects of process deadtime and noise are also considered. Finally, the limitations of each of the STPI algorithms is discussed, some conclusions are drawn from the performance comparisons, and a few recommendations are made.



## 1. INTRODUCTION

It used to be a difficult and time-consuming task to tune process controllers, but in the past few years several manufacturers have begun to incorporate self-tuning controller algorithms to automatically tune their proportional-integral-derivative (PID) controller parameters. This work describes the research, process simulation development, and tests for comparison of three self-tuning controller algorithms that were implemented by researchers at Sunderland Polytechnic, Sunderland, England, to work in conjunction with the Bristol-Babcock PID control module. These self-tuning PI control algorithms are based on closed-loop cycling theory, pattern recognition theory, and model-based theory. Bristol-Babcock, Inc., extended the opportunity to evaluate these self-tuning control algorithms prior to their commercial implementation.

### 1.1 OBJECTIVES

The objectives of this work are to

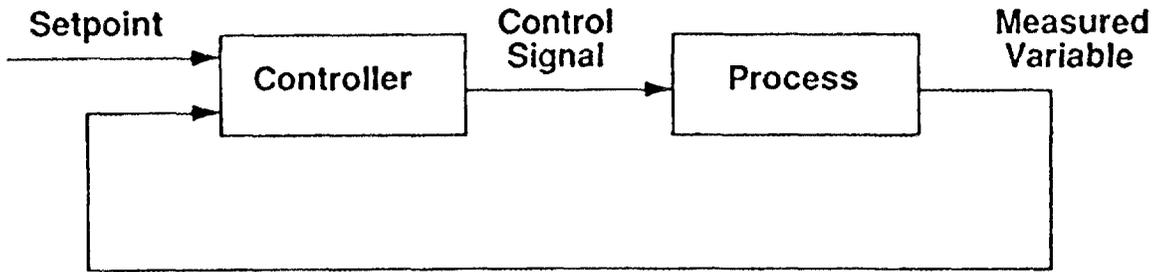
1. investigate the operation of the three self-tuning control algorithms developed for the Bristol-Babcock, Inc., controller;
2. develop process simulations needed to test these algorithms; and
3. test the performance and robustness of the three self-tuning algorithms prior to their commercial implementation.

### 1.2 BRIEF OVERVIEW OF RELATED ADAPTIVE CONTROL METHODS

Self-tuning control is just one of several related adaptive control methods. Most single-loop controllers in use today are designed to control a constant-gain linear feedback loop at a fixed operating point as shown in Fig. 1.1. However, it may be necessary or desirable to use adaptive controller tuning methods for one or more of the following reasons.

1. Most processes are really nonlinear.
2. Process parameters may change dynamically.
3. The process may have varying disturbance inputs.
4. Adaptive tuning techniques can improve performance.
5. Self-tuning improves engineering efficiency.

Several related adaptive tuning methods have developed from modern control theory, including gain scheduling, self-tuning, auto-tuning, and model-reference adaptive



1.1. Conventional feedback controller structure.

control (Aström and Wittenmark 1989). A brief introduction to each of these methods is given below.

A gain-scheduling system monitors a process variable and adjusts the controller parameters according to a predetermined gain schedule as shown in Fig. 1.2. There is some debate as to whether this technique should really be classified as *adaptive control*, because there is really no feedback path that interactively "fine tunes" the controller parameter values. This technique is used mainly to control processes for which the dynamics are well understood (e.g., aircraft control).

Self-tuning controllers (STCs) continuously adjust their controller parameters by using process identification and parameter by estimation techniques as shown in Fig. 1.3. Some manufacturers' implementations also add a small disturbance input to the control signal to assist with the process identification. Auto-tuning controllers (ATCs) are essentially the same as the STCs except that ATCs calculate new PID parameters only at start-up and on demand whereas STCs can continuously adjust their PID parameters.

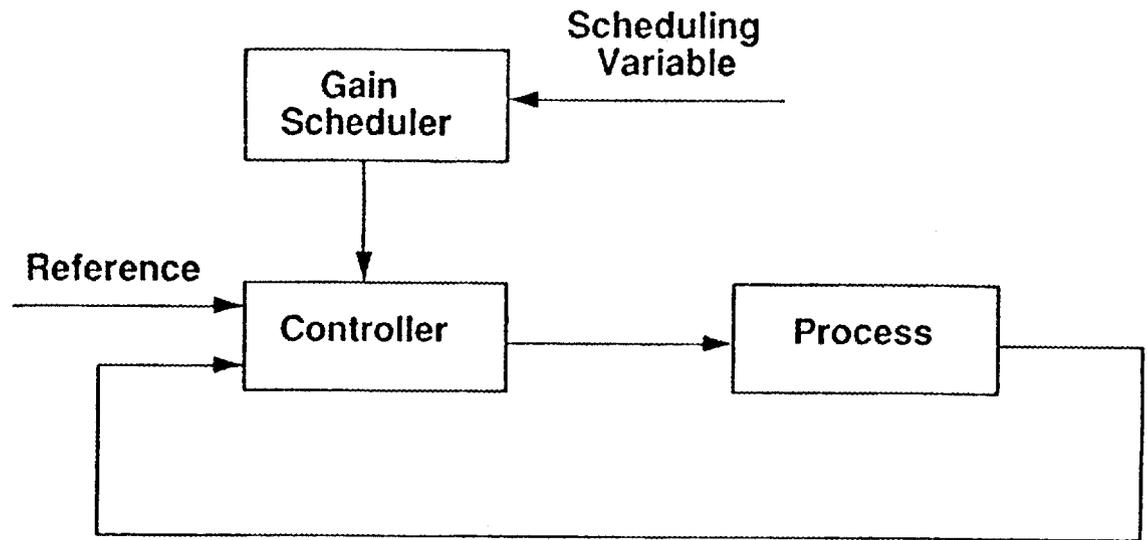
Model-reference adaptive systems use a reference model to adjust the controller parameters to obtain the desired performance as shown in Fig. 1.4. First, an ideal model is constructed to define the desired process behavior characteristics. Then, the measurement is compared to the model output and the controller parameters are adjusted as necessary to make the process behave like the model.

Self-tuning systems and model-reference systems are closely related. Both systems have two feedback loops; the inner loops are ordinary feedback loops and the inner loop parameters are set by the outer loop. Also, the controller adjustments for both types of systems are based on both input and output sampling. Although much research has been done for each of these adaptive control techniques, most of the industrial adaptive controllers that have been developed use the self-tuning control technique.

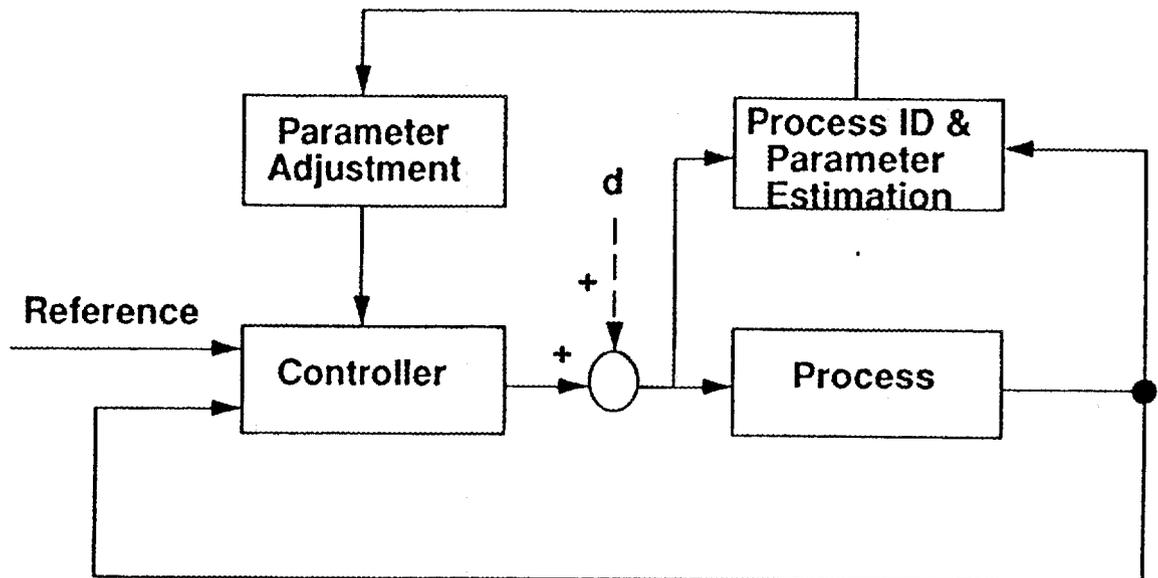
Other adaptive control techniques (e.g., linear quadratic gaussian, generalized minimum variance, and various predictive control techniques) have been industrially implemented, but these are considered to be beyond the scope of this work.

### 1.3 ORGANIZATION

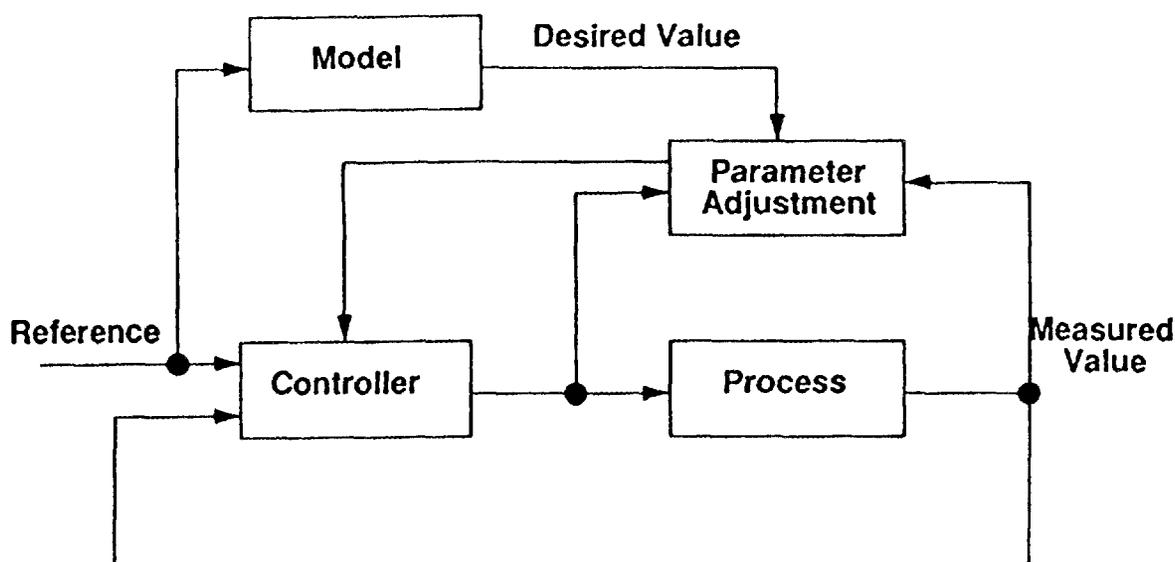
A literature survey was done to determine which adaptive control methods were most commonly being used. A brief introduction to the methods that relate to the design of currently available self-tuning PID controllers is given above. The methods



1.2. Gain-scheduling controller structure.



1.3. Self-tuning controller structure.



1.4. Model-reference adaptive controller structure.

discussed include gain scheduling, self-tuning, auto-tuning, and model-reference adaptive control systems.

Chapter 2 provides additional background information specifically relating to self-tuning controllers. The two most essential parts of the self-tuning controller are examined—the process identification technique and the parameter adjustment method. The process identification techniques discussed include transient-response analysis, frequency-response analysis, and parameter estimation methods. The PID controller parameter adjustment techniques presented are the Ziegler-Nichols and the pole-placement methods. Then, the characteristics of the two most common types of self-tuning controllers that have been implemented by industry (i.e., pattern recognition and process identification) are summarized.

A brief theory of operation for the three self-tuning proportional-plus-integral (PI) control algorithms developed by researchers at Sunderland Polytechnic, Sunderland, England, for use with the PID control module of Bristol-Babcock, Inc. (BBI) is given in Chapter 3 (full details are given in the original research report included in Appendix A). Bristol-Babcock graciously agreed to allow an independent evaluation of these algorithms prior to their commercial implementation. These algorithms are based on closed-loop cycling theory, pattern recognition theory, and model-based theory.

Various process simulations were developed to test each controller's performance and to determine the types of processes for which each of the controller algorithms might best be suited. The processes that were simulated include an integrating process, a first-order system, a second-order system, a system with initial inverse response, and a system with variable time constant and delay. The details of the process simulation design and the controller tests are given in Chapter 4.

In Chapter 5, the STPI algorithms' performance with regard to both setpoint changes and load disturbances is evaluated, and their robustness is compared. The effects of process deadtime and noise are also considered.

Finally, the limitations of each of the self-tuning controller algorithms is discussed in Chapter 6. Some conclusions are drawn from the performance comparisons, and several recommendations are made.

## 2. FURTHER SELF-TUNING CONTROL BACKGROUND

The two most essential parts of the self-tuning controller are the process (or system) identification technique and the parameter adjustment method (Fig. 1.3). These two important elements will be examined in greater detail in the following sections.

### 2.1 PROCESS IDENTIFICATION TECHNIQUES

Most commercially available self-tuning controllers use one of the following process identification techniques—transient-response analysis, frequency-response analysis, or parameter estimation methods.

#### 2.1.1 Transient-Response Analysis

Transient-analysis techniques can identify simple (first- or second-order systems with or without deadtime) processes from an open-loop step-input response plot when the following conditions are satisfied.

1. The system is initially in steady state when the test begins.
2. The system is approximately linear (in the test range).
3. Measurement errors are negligible (i.e., the system is relatively noise free).

Although most processes are nonlinear and complex, most can also be approximated as a first-order process with time delay as given by

$$G(s) = \frac{Ke^{-Ls}}{1 + Ts} \quad (2.1)$$

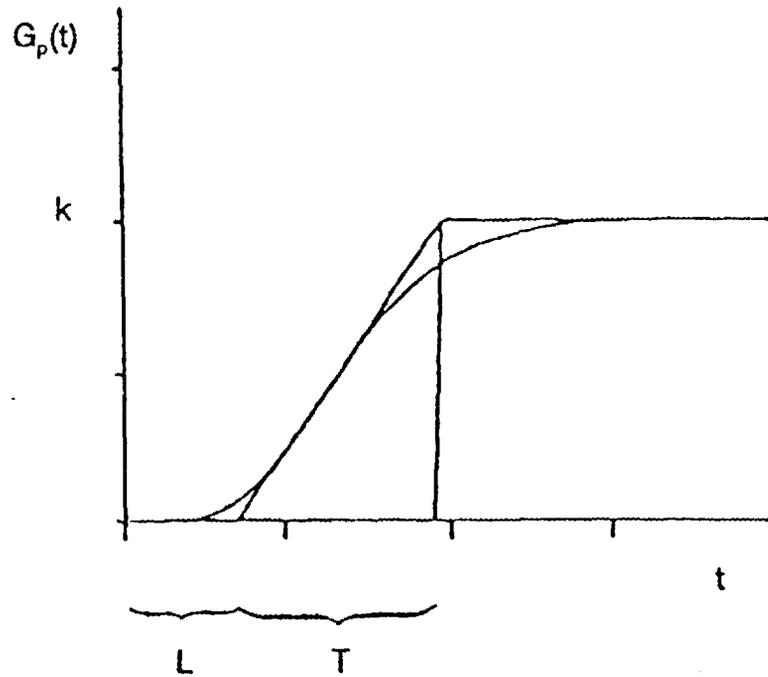
The process gain  $K$  time constant  $T$  and deadtime  $L$  can easily be determined from the step-response reaction curve of a first-order process (Fig. 2.1).

Oscillatory (i.e., second-order) systems can also be identified by using transient-response analysis techniques (Fig. 2.2). Once the period of oscillation  $T_p$  and damping  $d$  are obtained, they are used to calculate the natural frequency  $\omega_n$  and relative damping factor  $\zeta$  to identify a second-order system of the form

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (2.2)$$

where

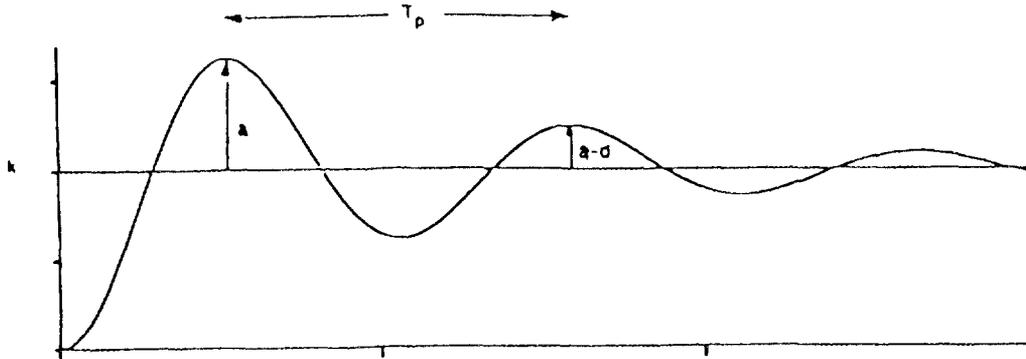
$$\zeta = \left[ \sqrt{1 + (2\pi/\log d)^2} \right]^{-1} \quad \text{and} \quad \omega_n = \frac{2\pi}{T_p \sqrt{1 - \zeta^2}} \quad .$$



L = Delay  
T = Time constraint

**2.1. Typical step response of a first-order process.**  
*Source: K. J. Aström and Tore Häggglund, Automatic Tuning of PID Controllers, Fig. 3.2B, p. 32, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.*

Transient-response process identification techniques are implemented in closed-loop self-tuning controllers in a variety of forms. Some STCs superimpose step (or pulse) disturbances on the reference signal. Some units only retune the controller parameters after setpoint changes or relatively large load disturbances. The desired system performance characteristics may also be requested in many different ways (e.g., desired damping, overshoot, time constant). Many units also include heuristics and additional logic to handle systems of increased complexity.



$k$  = Gain

$a$  = Magnitude of first peak

$T_p$  = Period of oscillation

$d$  = Damping

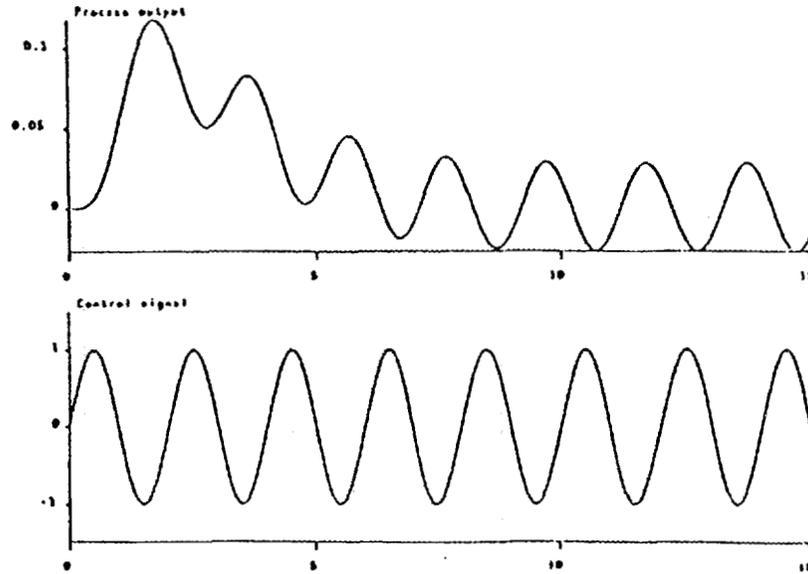
2.2. Typical step response of a second-order process. Source: K. J. Aström and Tore Hägglund, *Automatic Tuning of PID Controllers*, Fig. 3.4, p. 34, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.

### 2.1.2 Frequency-Response Analysis

Frequency-response analysis techniques can also be used to identify simple processes as well as some processes that have more complex forms. Many frequency-response analysis techniques exist. The Ziegler-Nichols frequency-response method is probably the most well known. However, the relay feedback method is really the most practical.

For a sinusoidal input, a stable linear system will produce a sinusoidal output after a brief transient response (Fig. 2.3). This means that the relationship between the input and output of a process can be described by two numbers:

1. the quotient of the output and input amplitudes  $\alpha$  and
2. the phase shift between the input and output signals  $\varphi$ .



**2.3. Typical sinusoidal output response to a sinusoidal input.** Source: K. J. Aström and Tore Hägglund, *Automatic Tuning of PID Controllers*, Fig. 3.5, p. 38, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.

However, the system response with this method can be determined at only one point from each sinusoidal input. To completely describe the transfer function of the process,  $\alpha$  and  $\varphi$  must be known at all frequencies

$$G(i\omega) = \alpha(\omega)e^{i\varphi(\omega)} \quad (2.3)$$

where

$$\alpha(\omega) = |G(i\omega)| ,$$

$$\varphi(\omega) = \arg[G(i\omega)] .$$

Fortunately, techniques have been developed that require the knowledge of the system response at only one frequency. The Ziegler-Nichols frequency-response technique is one experimental method of identifying the process. This can be done with the following steps.

1. Set the controller integral and derivative terms to zero.

2. Adjust the gain until uniform oscillations are obtained (Fig. 2.4). This gain is called the *ultimate gain*.
3. Calculate the critical system frequency at the ultimate gain.

Several design methods could then be used if this technique could be automated. However, implementation problems prevent the Ziegler-Nichols frequency-response method from being a practical design for implementation in an industrial self-tuning controller. The primary reason this technique is difficult to safely automate is that operating some processes at or near their point of instability may be harmful to the equipment or dangerous to personnel.

The relay feedback method (Aström and Hägglund 1988) is a practical design technique for identifying a process (Fig. 2.5). It uses a relay to automatically generate a sinusoidal output until the appropriate oscillations are obtained (Fig. 2.6). The ultimate period and ultimate gain are easily calculated from the critical frequency, and then the PID parameters can be determined.

This technique can be easily automated, and only one parameter must be specified—the initial relay amplitude. However, the most widely used process identification method is the parameter estimation technique.

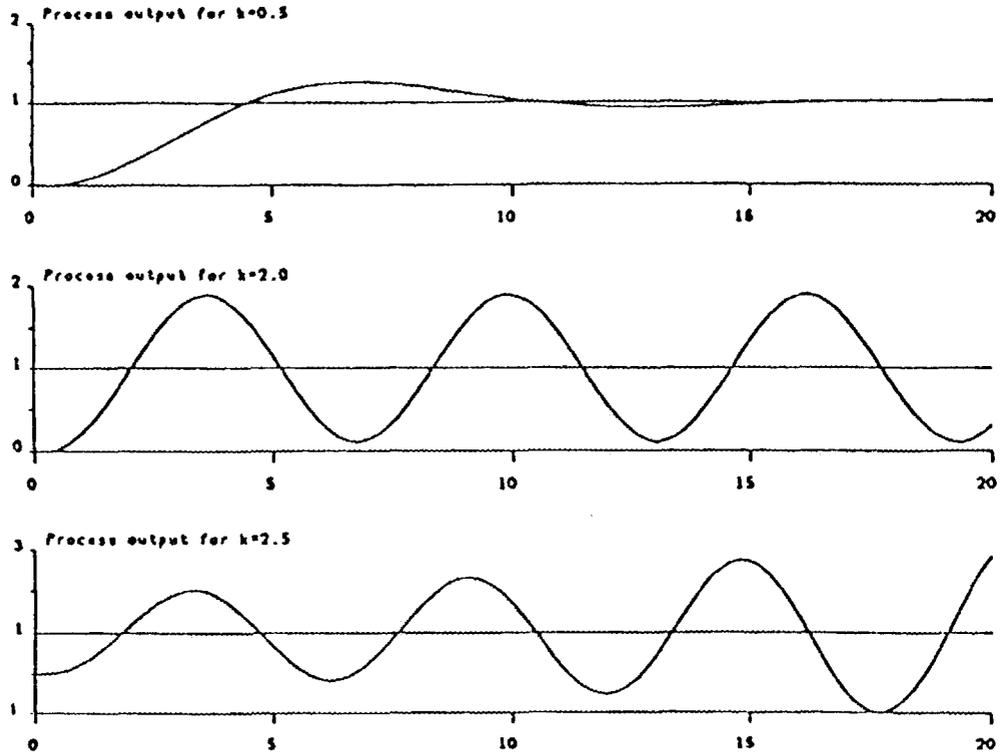
### 2.1.3 Parameter Estimation Techniques

Parameter estimation techniques involve sampling the controller's input and output and constructing a mathematical model of the process. The most common parameter estimation technique is recursive in nature. The controller input/output (I/O) is sampled, and process model parameters are computed recursively by using matrix manipulation techniques to fit a predetermined low-order process model.

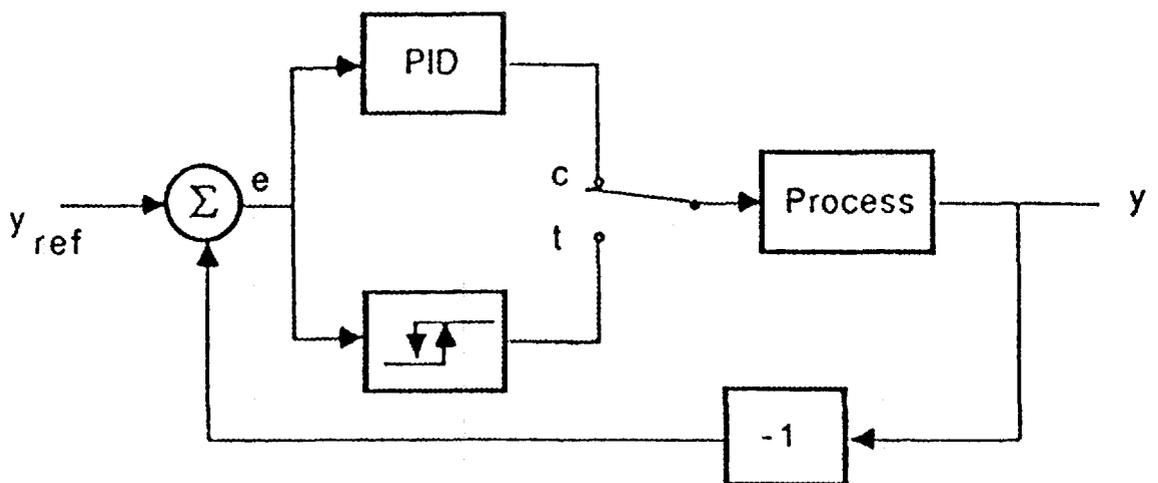
There are some distinct advantages to using the parameter estimation technique to identify the process. The process model output is continuously refined, and the controller can continuously update the PID parameters. However, the parameter estimation technique also has some disadvantages. The mathematics involved are more complex, and more prior information must be specified by the user (e.g., sampling period, initial model parameters). Thus, most products that use this technique have a pretuning phase (based on one of the transient or frequency analysis techniques) to obtain the additional required information.

## 2.2 PID PARAMETER ADJUSTMENT TECHNIQUES

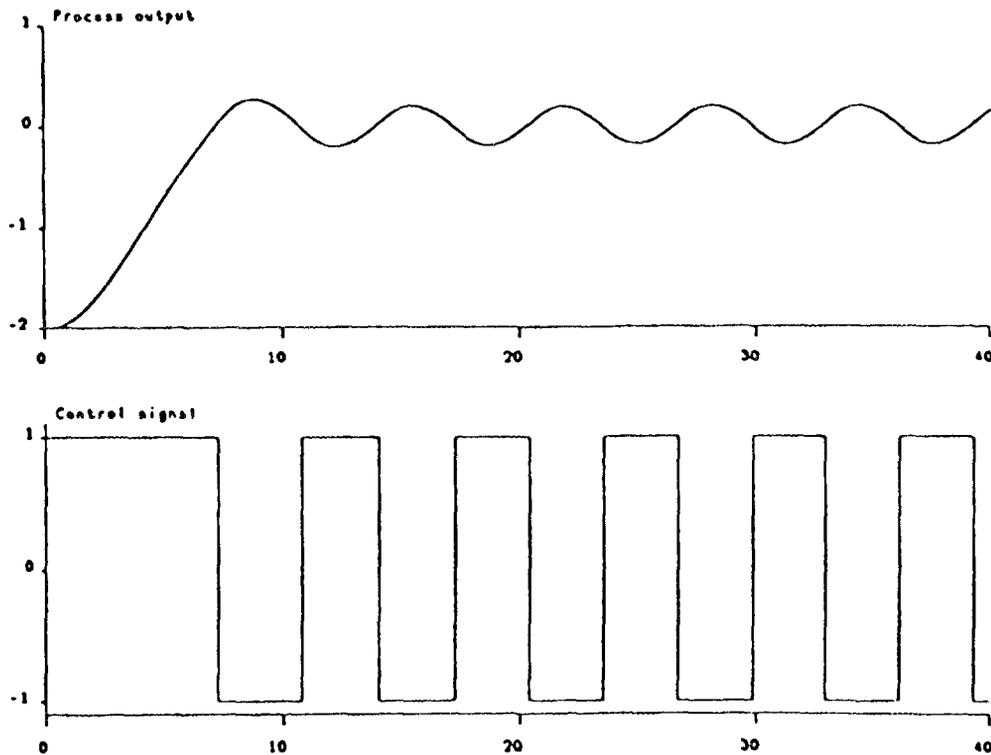
Once the process identification is complete, the self-tuning controller uses some technique to determine how to adjust the PID parameters. The most widely used PID parameter adjustment methods are the Ziegler-Nichols method and the pole-placement methods. Each of these methods will now be examined in further detail.



2.4. Varied output response plots for the same process with different controller gains. Source: K. J. Aström and Tore Hägglund, *Automatic Tuning of PID Controllers*, Fig. 3.7, p. 39, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.



2.5. Relay feedback controller structure. Source: K. J. Aström and Tore Hägglund, *Automatic Tuning of PID Controllers*, Fig. 5.2, p. 109, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.



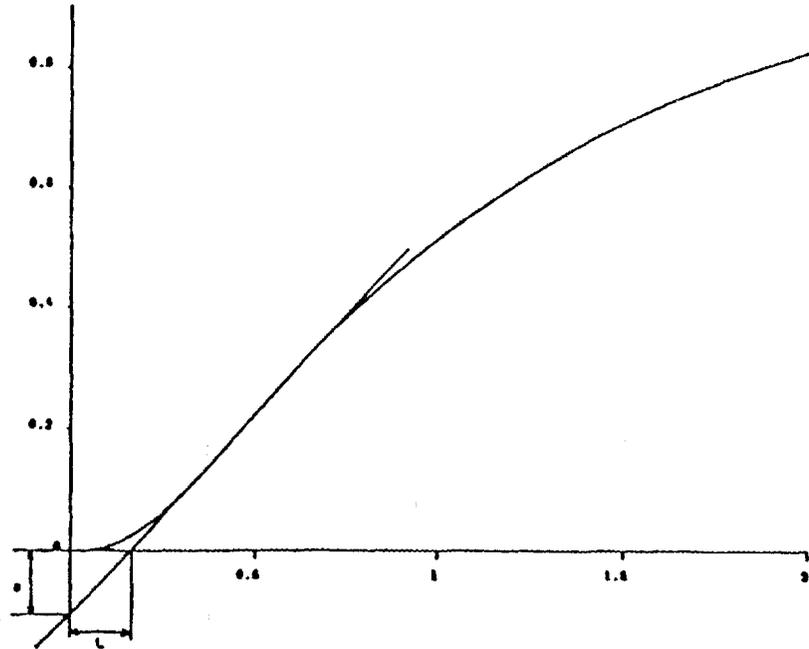
2.6. Sinusoidal output response generated by a relay feedback controller. Source: K. J. Aström and Tore Hägglund, *Automatic Tuning of PID Controllers*, Fig. 3.10, p. 41, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.

### 2.2.1 Ziegler-Nichols Methods

The two classical tuning methods that were presented by Ziegler and Nichols (1942) are still widely used—the Z-N step-response method and the Z-N frequency-response method. The Z-N step-response method is based on an analysis of the open-loop step response of the system (Fig. 2.7). Once the gain and apparent deadtime have been determined, the recommended PID parameters and an estimate of the dominant dynamics of the closed-loop system can be determined from Table 2.1.

The Z-N frequency response method uses the ultimate gain and ultimate period to calculate PID parameters and dominant system dynamics (Table 2.2). The location of the dominant system pole has a great effect on the system performance (Figure 2.8). The Z-N methods are based on the idea that the system dynamics can be changed by moving one point on the Nyquist curve (Fig. 2.9).

However, much uncertainty exists with the Z-N frequency design method. It is not possible to determine the location of all the dominant poles of the system from only one point on the Nyquist plot. Several other techniques could be used if two or more points on the Nyquist curve were known. However, most of these uncertainties vanish if the pole-placement design method can be used.



$a$  = Gain factor

$L$  = Apparent deadtime

2.7. Typical open-loop step-response plot. Source: K. J. Aström and Tore Häggglund, *Automatic Tuning of PID Controllers*, Fig. 4.1, p. 53, reprinted with permission from the Instrument Society of America, Raleigh, North Carolina, 1988.

## 2.2.2 Pole-Placement Method

For this technique, the process is approximated by a model of first or second order. Then, the PID parameters are calculated on the basis of the desired closed-loop pole-placement (Aström and Häggglund 1988). The effectiveness of the pole-placement method hinges on the ability to approximate the process accurately enough with a low- (i.e., first- or second-) order model.

### 2.2.2.1 First-order approximation

If the process can be described by a first-order model of the following form

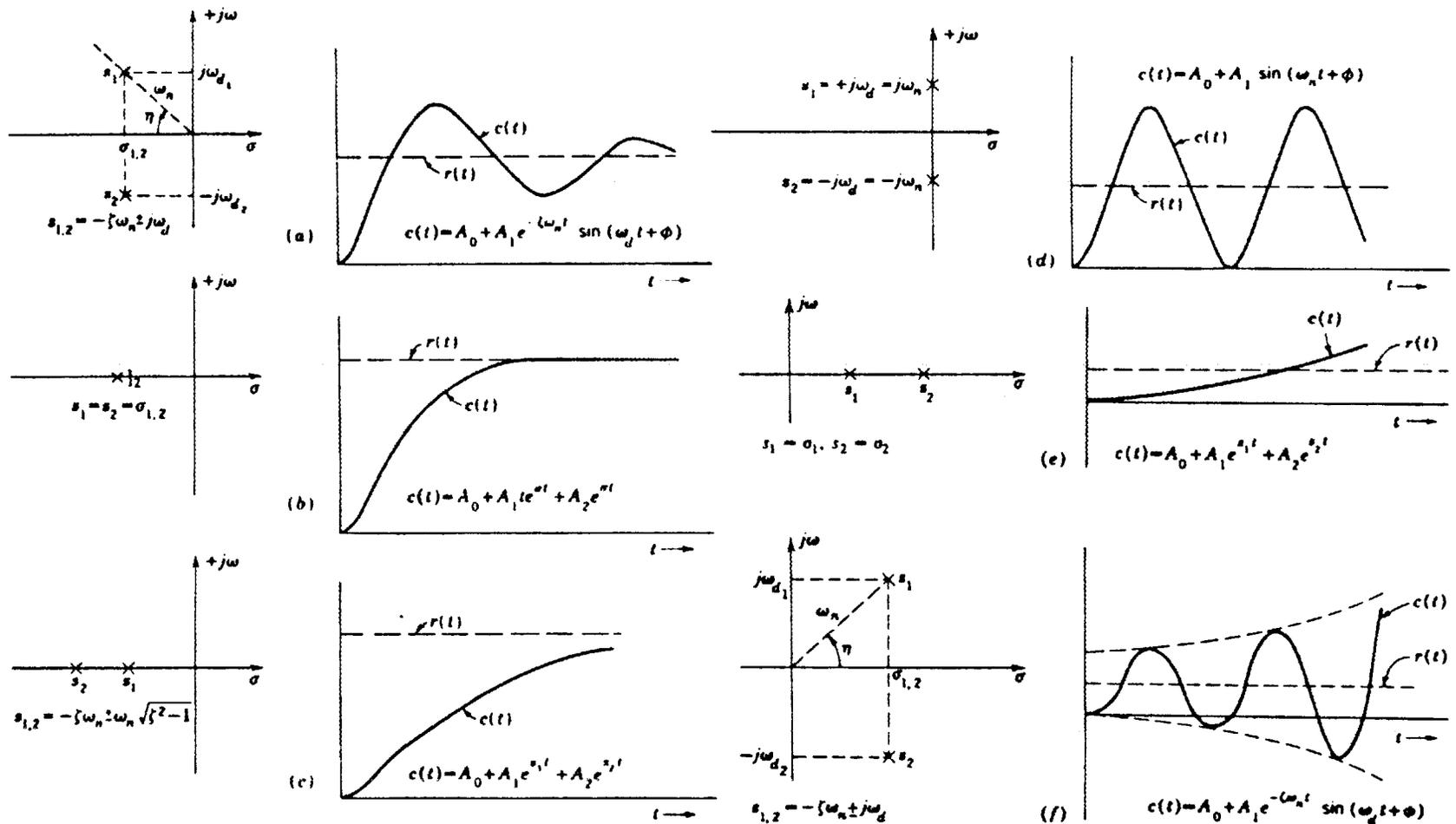
$$G_P = \frac{k_p}{1 + T_1 s}, \quad (2.4)$$

**Table 2.1. Controller parameters and dominant dynamics obtained by the Ziegler-Nichols open-loop step-response method**

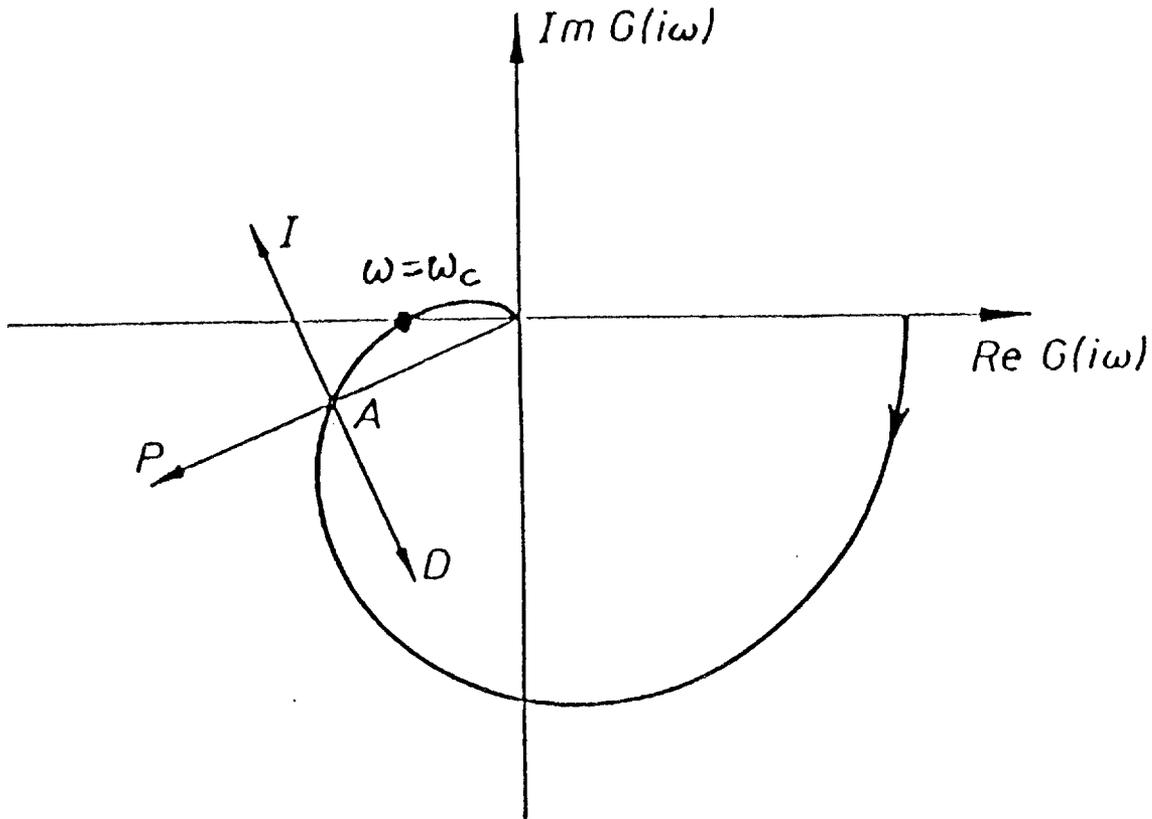
Controller type	Controller parameters			Dominant pole $T_p$
	K	$T_i$	$T_d$	
P	$1/a$	---	---	$4 L$
PI	$0.9/a$	$3L$	---	$5.7 L$
PID	$1.2/a$	$2L$	$L/2$	$3.4 L$

**Table 2.2 Controller parameters and dominant dynamics obtained by the Ziegler-Nichols closed-loop frequency-response method**

Controller type	Controller parameters			Dominant pole $T_p$
	K	$T_i$	$T_d$	
P	$0.5 k_c$	---	---	$t_c$
PI	$0.4 k_c$	$0.8 t_c$	---	$1.4 t_c$
PID	$0.6 k_c$	$0.5 t_c$	$0.12 t_c$	$0.85 t_c$



2.8. Effects of dominant pole location on system performance. Source: John J. D'Azzo and Constantine H. Houpsis, *Linear Control System Analysis and Design: Conventional and Modern*, Fig. 7.22, p. 243, reprinted with permission from McGraw-Hill, New York, 1988.



**2.9. Changing system dynamics by moving one point on the Nyquist curve.** *Source:* K. J. Aström and Tore Hägglund, *Automatic Tuning of PID Controllers*, Fig. 4.4, p. 57, reprinted with permission from Instrument Society of America, Raleigh, North Carolina, 1988.

then the process can be controlled by a controller of the form

$$G_R = K \left[ 1 + \frac{1}{T_i s} \right]. \quad (2.5)$$

The closed-loop system can then be described as

$$G_C = \frac{G_P G_R}{1 + G_P G_R}, \quad (2.6)$$

and the closed-loop pole can be obtained from the characteristic equation

$$1 + G_P G_R = 0 . \quad (2.7)$$

Substitution then shows that the characteristic equation is

$$s^2 + s \left( \frac{1}{T_1} + \frac{k_p K}{T_1} \right) + \frac{k_p K}{T_1 T_i} = 0 , \quad (2.8)$$

which can be compared to the characteristic equation described by the desired relative damping and frequency

$$s^2 + 2\zeta\omega s + \omega^2 = 0 . \quad (2.9)$$

Because the coefficients of Eqs. (2.8) and (2.9) should be equal, we have

$$\begin{aligned} \omega^2 &= \frac{k_p K}{T_1 T_i} \\ 2\zeta\omega &= \frac{1 + k_p K}{T_1} . \end{aligned} \quad (2.10)$$

Thus, the proportional-integral (PI) parameters can be determined as

$$\begin{aligned} K &= \frac{2\zeta\omega T_1 - 1}{k_p} \\ T_i &= \frac{2\zeta\omega T_1 - 1}{\omega^2 T_1} . \end{aligned} \quad (2.11)$$

### 2.2.2.2 Second-order approximation

If the process can be described by a second-order model of the form

$$G_P = \frac{k_p}{(1 + T_1 s)(1 + T_2 s)} , \quad (2.12)$$

then the process can be controlled by a PID controller of the form

$$G_R = \frac{K(1 + T_I s + T_I T_D s^2)}{T_I s} . \quad (2.13)$$

Then, if the desired response is described by the characteristic equation

$$(s + \alpha \omega)(s^2 + 2\zeta \omega s + \omega^2) = 0 , \quad (2.14)$$

similar techniques can be used to show that the PID parameters can be calculated as

$$\begin{aligned} K &= \frac{T_1 T_2 \omega^2 (1 + 2\zeta \alpha) - 1}{k_p} \\ T_i &= \frac{T_1 T_2 \omega^2 (1 + 2\zeta \alpha) - 1}{T_1 T_2 \alpha \omega^3} \\ T_d &= \frac{T_1 T_2 \omega (\alpha + 2\zeta) - T_1 - T_2}{\omega^2 T_1 T_2 (1 + 2\zeta \alpha) - 1} . \end{aligned} \quad (2.15)$$

## 2.3 TWO MOST COMMON INDUSTRIALLY IMPLEMENTED DESIGNS

In summary, the two most common industrially implemented self-tuning controllers are based on one of two basic techniques—pattern recognition or process identification. The characteristics for each type are listed in the following sections.

### 2.3.1 Pattern Recognition Method

Self-tuning controllers that use the pattern recognition method

1. monitor the controller's input and output;
2. identify the process by using transient- or frequency-response analysis;
3. compare the actual response to the desired response characteristics;
4. calculate new parameters by using Ziegler-Nichols methods;
5. automatically update PID values whenever possible; and
6. require only relatively simple mathematics techniques.

### 2.3.2 Process Identification Method

Self-tuning controllers that use the process identification method

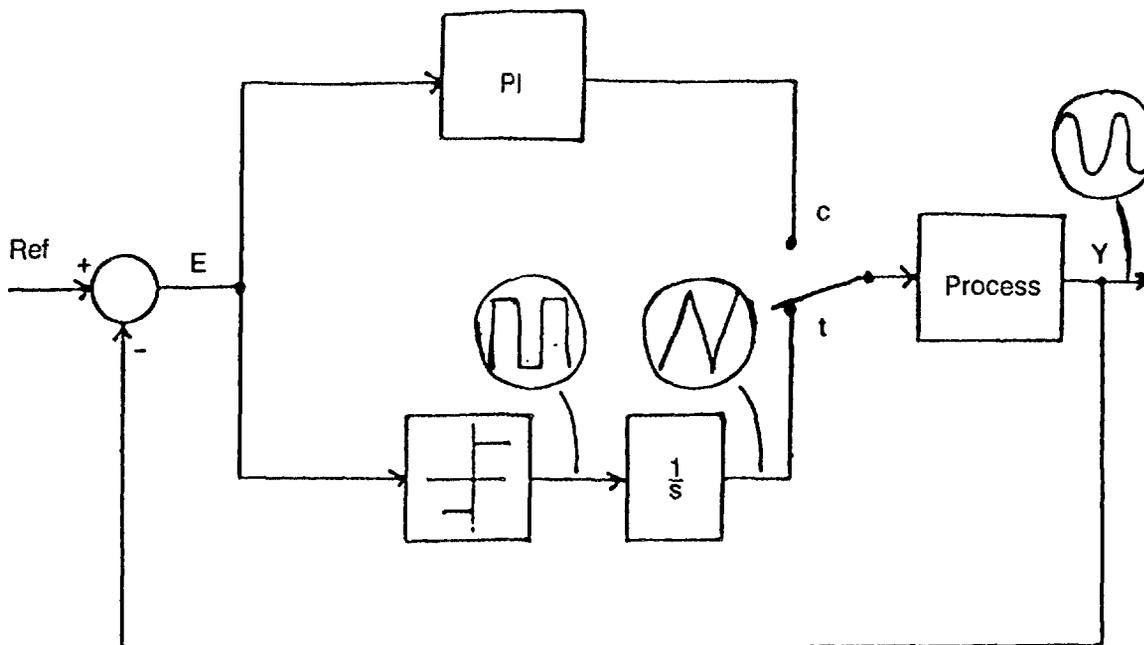
1. continuously monitor the controller's inputs and outputs;
2. identify the process by using parameter estimation techniques;
3. construct a mathematical model of the process;
4. calculate new PID parameters regularly by using the pole-placement methods;
5. automatically update PID parameters whenever necessary; and
6. require somewhat more complex mathematics techniques.

### 3. THEORY OF OPERATION OF THE BBI STPI ALGORITHMS

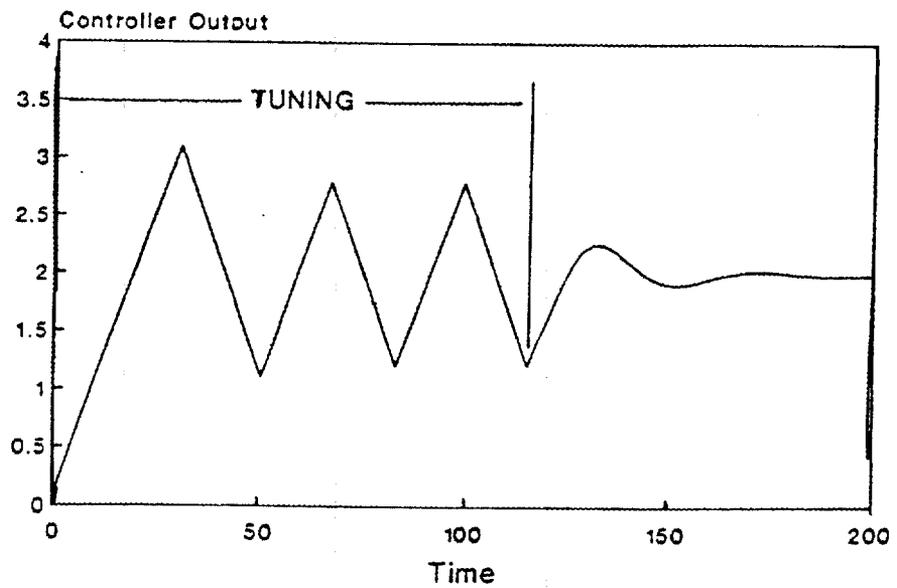
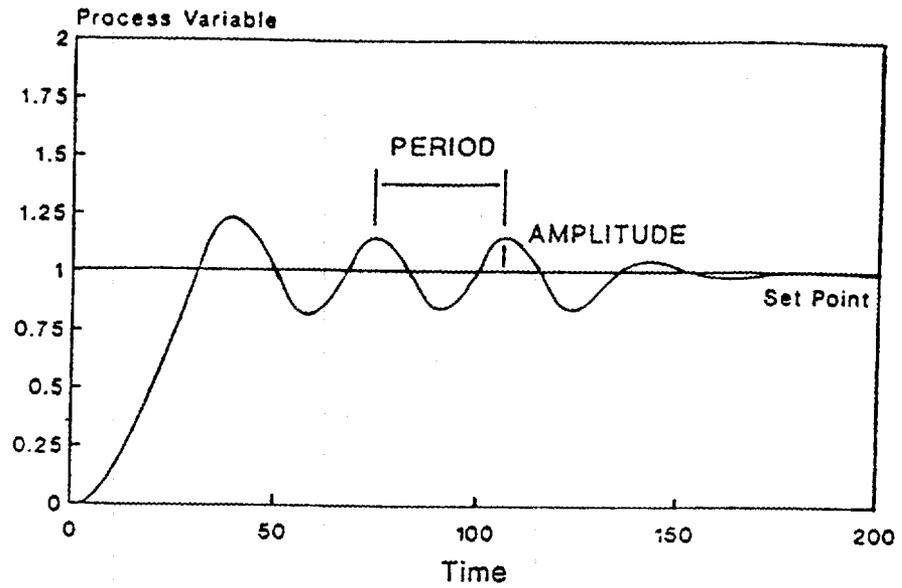
The remainder of this work focuses on the testing and comparison of the three self-tuning proportional-plus-integral (STPI) control algorithms. These STPI algorithms were implemented by researchers at Sunderland Polytechnic, Sunderland, England, for use with Bristol-Babcock's standard PID3TERM control module. The algorithms are based on closed-loop cycling theory, pattern recognition theory, and model-based theory (a copy of the original research report is included in Appendix A). An abbreviated theory of operation is given in the following sections.

#### 3.1 CLOSED-LOOP CYCLING ALGORITHM THEORY

This algorithm is a *one-shot* tuning method based on the Aström and Hägglund Relay Feedback Method (Aström and Wittenmark 1989). A relay controller and an integrator used as shown in Fig. 3.1 generates a periodic triangular perturbation output, and the process variable is forced to oscillate around its setpoint value as shown in Fig. 3.2. The period of the oscillations is determined by the dynamics of the process, but the user can constrain the amplitude of the oscillations by specifying the initial relay amplitude characteristic, maximum and minimum controller output limits, and the maximum allowable deviation of the process variable from setpoint. The tuning phase is automatically terminated when a number of *good* oscillations have been recorded.



3.1. Bristol-Babcock, Inc., relay feedback controller structure. Source: Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. III, Fig. 6, Sunderland Polytechnic, Sunderland, U.K., 1990.



**3.2. Output response generated by the closed-loop cycling algorithm.** *Source:* Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. 1, Fig. 2, Sunderland Polytechnic, Sunderland, U.K., 1990.

Upon termination, the period and amplitude of the oscillations are measured and used to calculate new PI controller settings. If the tuning phase does not obtain good results after the specified maximum number of cycles, then it will also terminate with no recalculation of the PI parameters. This technique is explained in further detail in the following paragraphs.

After activating the closed-loop cycling self-tuning procedure, the process should obtain constant-amplitude fixed-frequency oscillations within a few cycles. The algorithm is designed to automatically reduce the relay amplitude if the specified initial amplitude is too large. However, if the initial amplitude is obviously much too large, the user may want to manually adjust the amplitude during the tuning phase to keep the process variable near the setpoint.

Once constant oscillations have been obtained, the Ziegler-Nichols critical gain  $K_u$  for the process is easily calculated. The ultimate frequency  $P_u$  is also calculated by using the error signal and a *zero-crossing* routine. Once these parameters are evaluated, PI settings could easily be calculated (for quarter-amplitude damping) as shown in Table 3.1. However, Aström's proposed alternative approach, which allows calculation of PI settings of any desired phase margin, is implemented in this algorithm.

Table 3.1. Ziegler-Nichols ultimate frequency-response controller parameters

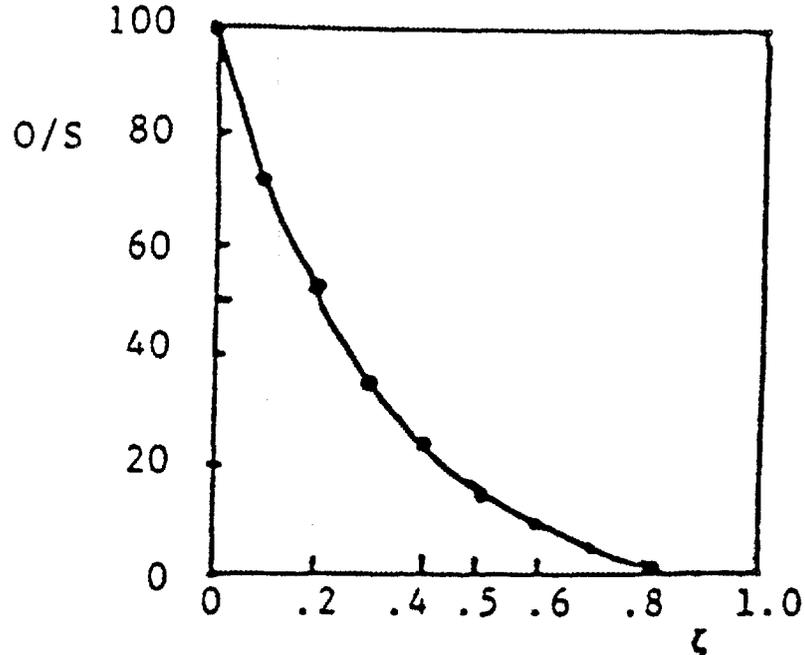
Controller Type	Controller Parameters		
	$K_c$	$T_i$	$T_d$
P	$0.5 K_u$	---	---
PI	$0.45 K_u$	$P_u/1.2$	---
PID	$0.6 K_u$	$P_u/2$	$P_u/8$

The developers recognized that every user may not understand the concept of phase margin. So, to make this concept more user friendly, they only require the user to specify the maximum desired percentage overshoot, which is then used to approximate the desired phase margin. Although this method does not allow the user to specify an overdamped response, from Figs. 3.3 and 3.4 it can be seen that this technique can be used over a wide range of overshoot values to approximate the desired phase margin. The resulting PI values can then be calculated by

$$K_c = \frac{4V_m P_u \sin(\phi_m)}{2\pi^2 A} \quad (3.1)$$

$$T_i = \frac{P_u \tan(\phi_m)}{2\pi}$$

Two optional enhancements may be needed if the process variable is somewhat noisy—relay hysteresis and digital filtering. The designers realized that noise



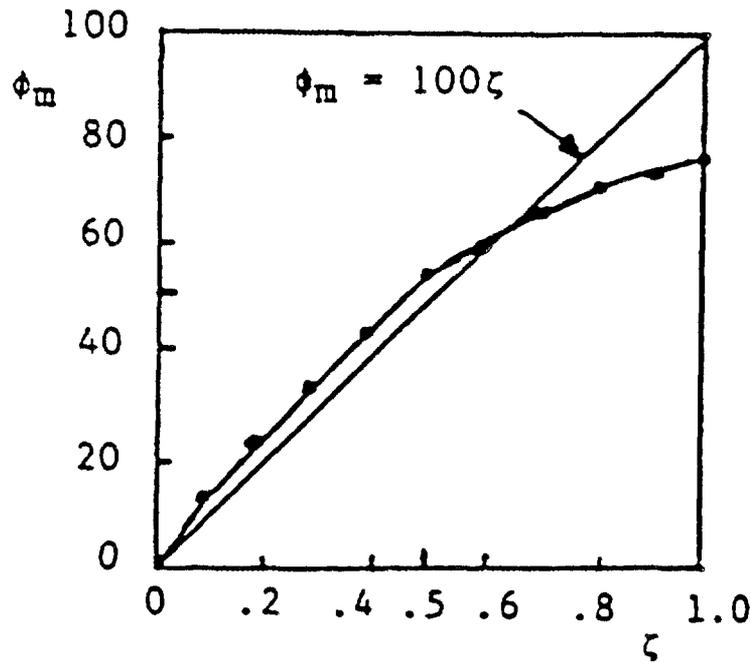
3.3. Percent overshoot vs damping ratio for the step response of a second-order system. *Source:* Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. III, Fig. 7(a), Sunderland Polytechnic, Sunderland, U.K., 1990.

superimposed on the process variable signal could result in *false* relay switching and invalidate the closed-loop cycling tuning procedure. Some hysteresis can easily be added to the relay to improve its noise rejection. Choosing the correct bandwidth for the digital filter is a more cumbersome problem. See the report in Appendix A, *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. III, pp. 148–49, for more details regarding these enhancements.

### 3.2 PATTERN RECOGNITION ALGORITHM THEORY

This algorithm provides continuous self-tuning of the PI controller parameters. When the pattern recognition self-tuning procedure is active, the PI controller parameters will be recalculated following any sufficiently large disturbance or setpoint change. New PI parameters are calculated in four distinct steps (Fig. 3.5).

1. The controller's error signal is continuously monitored for any disturbances that occur over a specified threshold value. When this threshold is exceeded, the



3.4. Phase margin vs damping ratio of a second-order system. Source: Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. III, Fig.7(b), Sunderland Polytechnic, Sunderland, U.K., 1990.

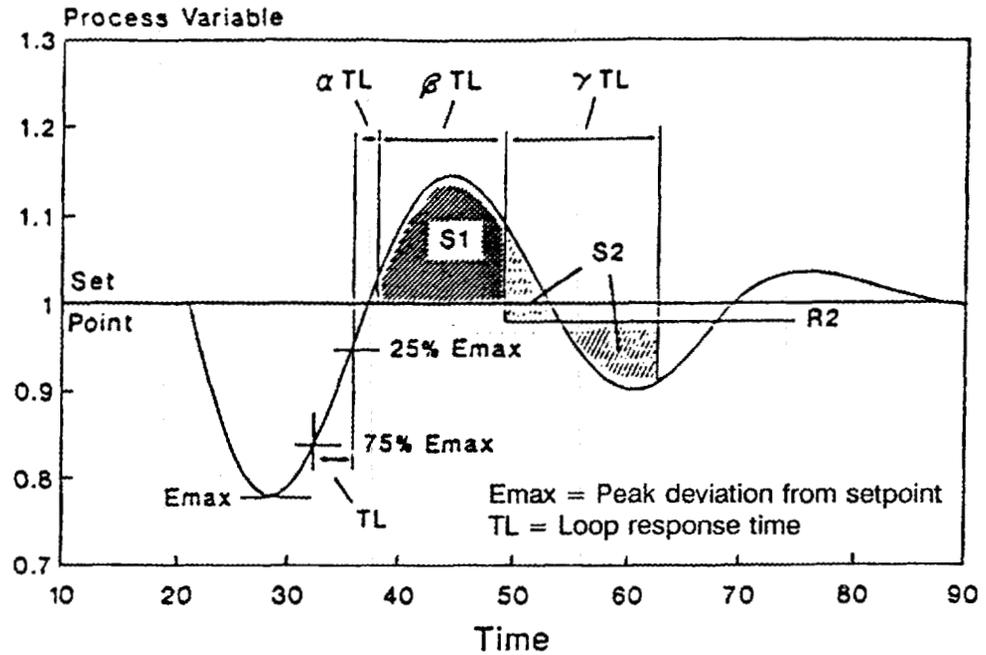
algorithm monitors the process variable to detect its peak deviation from setpoint,  $E_{\max}$

2. Then the recovery time of the loop response  $T_L$  is determined.  $T_L$  is calculated to be equal to the elapsed time it takes the system to go from 90% to 50% of the peak deviation from setpoint on return from the peak deviation.
3.  $T_L$  is then used in the evaluation of two integrals:  $S_1$  and  $S_2$ .  $T_1$  is the time when the system has reached 50% of the peak on return from the initial peak deviation (i.e., when  $T_L$  is just determined).

$S_1$  is the area under the curve from time  $T_1(1 + \alpha)$  to  $T_1(1 + \alpha + \beta)$ .

$S_2$  is the area under the curve from time  $T_1(1 + \alpha + \beta)$  to  $T_1(1 + \alpha + \beta + \gamma)$ .

4. Having obtained the value of these integrals, the new PI controller parameters can be calculated and updated as



3.5. Annotated output response describing the operation of the pattern recognition algorithm. Source: Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. I, Fig. 5, Sunderland Polytechnic, Sunderland, U.K., 1990.

$$K_c = K_c + (1 - DONE)[K_1(S_1 + R_1) + K_2 S_2] \quad (3.2)$$

$$K_i = K_i + (1 - DONE)[K_3(S_1 + R_1) + K_4 S_2]$$

where

$S_1$  = Area under the curve from time  $T_1(1 + \alpha)$  to  $T_1(1 + \alpha + \beta)$ .

$S_2$  = Area under the curve from time  $T_1(1 + \alpha + \beta)$  to  $T_1(1 + \alpha + \beta + \gamma)$ .

$R_2$  = Level related to desired overshoot ( $R_2 = \text{OVERSH}/900$ ).

$R_1$  = Area related to the actual overshoot ( $R_1 = \gamma R_2$ ).

$DONE$  = Confidence factor related to actual overshoot,

$K_1, K_2, K_3,$  and  $K_4$  = Constants.

### 3.3 MODEL-BASED ALGORITHM THEORY

The model-based algorithm is primarily intended for use as a *one-shot* tuner, although it may also be configured to operate in a continuous tuning mode (by the expert user). A very important difference between this algorithm and the previous two is

that the task rate must be carefully matched to the response time of the process. The developers suggest that a good rule for use with this model-based method is to select a task rate that is approximately one-tenth of the process rise time, which may be determined from a step test (Fig. 3.6).

During the tuning phase, a pseudorandom binary sequence (PRBS) is produced at the controller output, as shown in Fig. 3.7. The user must specify the initial mean level, *OPMEAN*, and the amplitude, *OPDEV*, of the PRBS: the mean level should be chosen to cause the process variable to deviate at or near its setpoint value, and the amplitude should be sufficiently large to cause significant deviations yet keep the process variable within acceptable limits. The mean level of the PRBS may need to be manually adjusted during the tuning phase to keep the process variable near the setpoint.

While the PRBS is applied, the process output and the controller output data are fed into a recursive least-squares-estimation algorithm that calculates the mathematical model parameters. The model is a first-order lag with time delay:

$$G_p(s) = \frac{Ke^{-\tau s}}{1 + Ts} \quad (3.3)$$

Although the digitized equivalent of this equation could theoretically have any number of terms in the numerator to accommodate any amount of delay time, the developers fixed the numerator terms to five. Thus, the digitized model equation is

$$G_p(z^{-1}) = \frac{b_1z^{-1} + b_2z^{-2} + b_3z^{-3} + b_4z^{-4} + b_5z^{-5}}{1 + a_1z^{-1}} \quad (3.4)$$

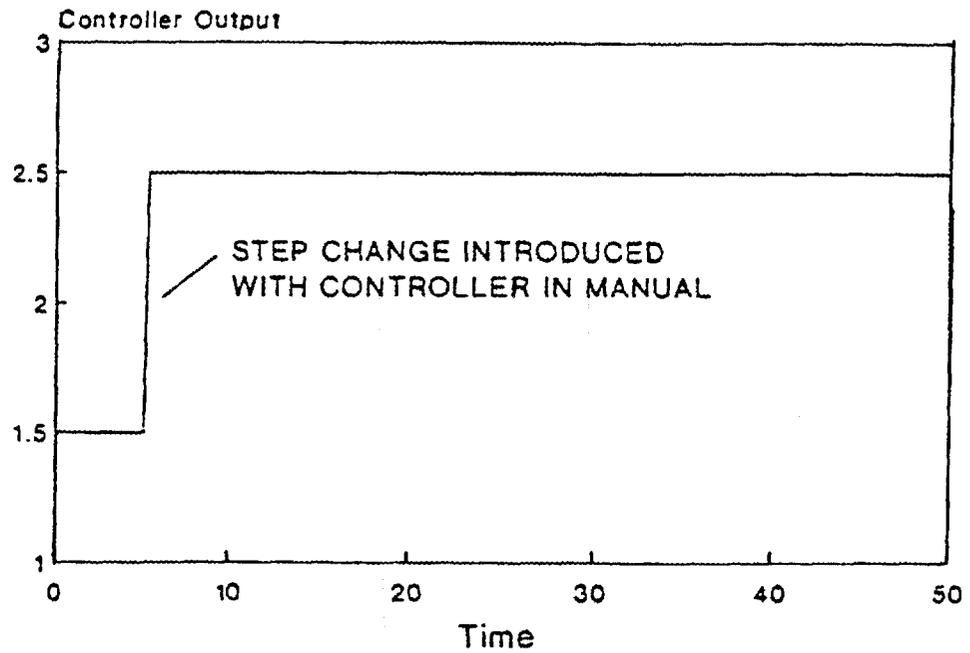
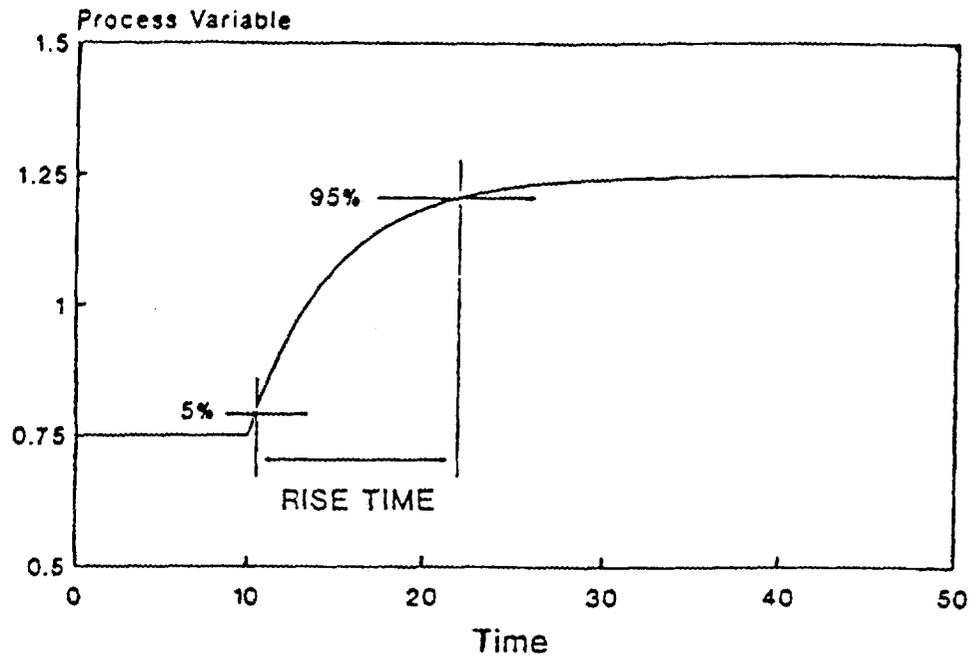
The process output and controller output are prefiltered by a digital band-pass filter to remove dc offsets and high-frequency noise and to make the estimation algorithm more robust.

At the end of the self-tuning phase, the identified model is used to calculate new PI controller settings. The discrete form of the ACCOL PI controller is given by

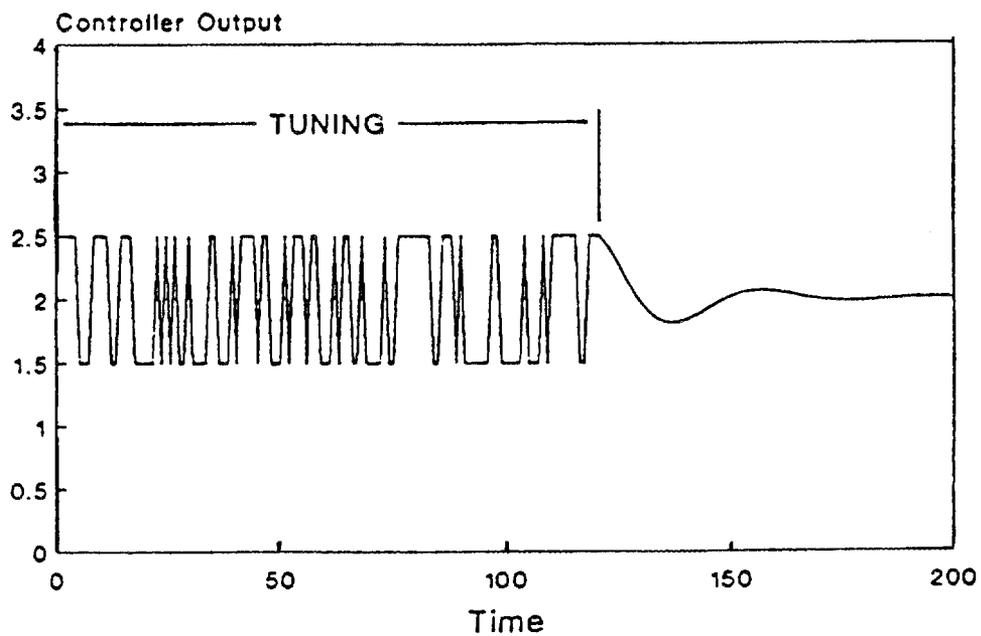
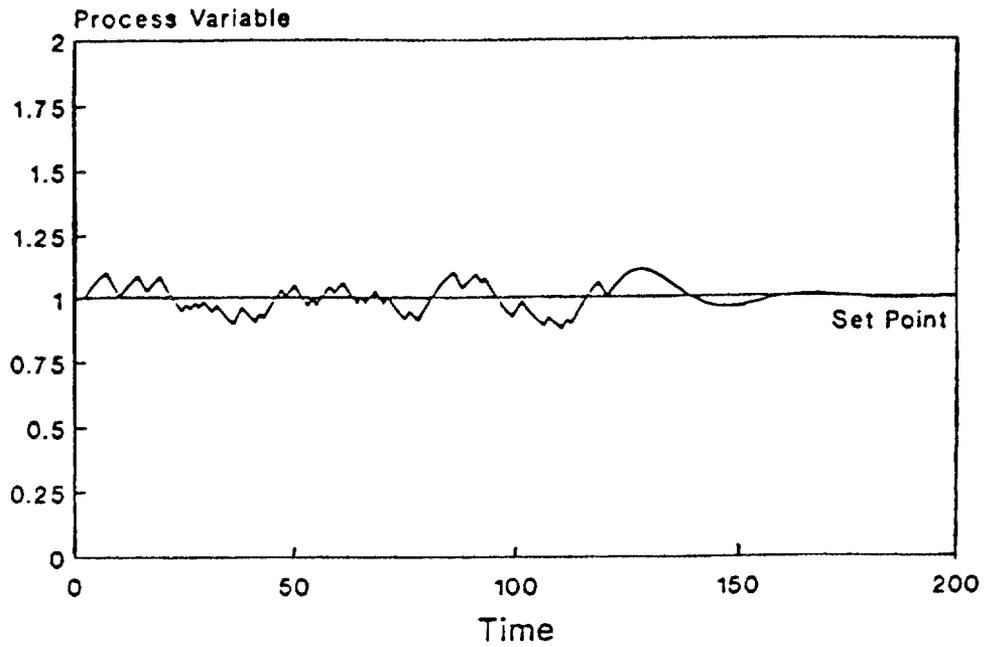
$$G_c(z^{-1}) = K_c \left[ \frac{1 + (K_i T - 1)z^{-1}}{1 - z^{-1}} \right] \quad (3.5)$$

$K_i$  is calculated such that the zero of the controller will cancel the pole of the system model in Eq. (3.4). Because the sample rate  $T$  is known,  $K_i$  is easily calculated as follows:

$$K_i = \frac{(1 + a_1)}{T} \quad (3.6)$$



3.6. Open-loop step response to determine a suitable task rate for the model-based algorithm. Source: Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. I, Fig. 6, Sunderland Polytechnic, Sunderland, U.K., 1990.



3.7. Output response generated by the model-based algorithm pseudorandom binary sequence. *Source:* Reprinted with permission from C. S. Cox et al., *Development of ACCOL Self-Tuning PI (STPI) Control Module*, Pt. I, Fig. 7, Sunderland Polytechnic, Sunderland, U.K., 1990.

Now the task is to calculate the  $K_c$  that will provide the required phase margin for the closed-loop compensated system. As with the closed-loop cycling algorithm, the user has simply defined the desired system performance by specifying the maximum desired percentage overshoot, which is then used to approximate the desired phase margin. However, the mathematics involved is slightly more complicated than before. With the pole-zero cancellation obtained by determining  $K_c$ , the remaining compensated open loop transfer function is given by

$$G_{OL}(z^{-1}) = K_c \left[ \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 - z^{-1}} \right]. \quad (3.7)$$

To determine the required  $K_c$ , the frequency response of the compensated system must be computed. This can be done using the discrete time to frequency domain mapping

$$z^{-1} = e^{-j\omega T}. \quad (3.8)$$

By using this substitution, the open-loop phase shift can be calculated at any frequency  $\omega$ , by using the relationship

$$\arg[G_{OL}(j\omega)] = -\tan^{-1} \left[ \frac{\sin \omega T}{1 - \cos \omega T} \right] - \tan^{-1} \left[ \frac{\sum_{i=1}^5 b_i \sin i \omega T}{\sum_{i=1}^5 b_i \cos i \omega T} \right]. \quad (3.9)$$

The angular frequency  $\omega_0$  at which the required phase margin occurs can be calculated as

$$\arg[G_{OL}(j\omega_0)] = -\pi + \phi_m. \quad (3.10)$$

The particular angular frequency  $\omega_0$  which yields the desired phase margin is then computed from Eqs. (3.9) and (3.10) by using a linear search algorithm in the range  $0 < \omega < \frac{\pi}{T}$ . Once  $\omega_0$  has been determined,  $K_c$  can be calculated from

$$|G_{OL}(j\omega)| = K_c \left[ \frac{\sqrt{\left[ \sum_{i=1}^5 b_i \cos i \omega_0 T \right]^2 + \left[ \sum_{i=1}^5 b_i \sin i \omega_0 T \right]^2}}{\sqrt{[1 - \cos \omega_0 T]^2 + [\sin \omega_0 T]^2}} \right] = 1. \quad (3.11)$$

## 4. PROCESS SIMULATIONS

To facilitate the self-tuning control algorithm tests, various process simulations were developed. Because the self-tuning algorithms were developed for Bristol-Babcock's distributed process controller model DPC 3330, it was the obvious process simulator of choice because the controller could execute both the self-tuning algorithms and the process models simultaneously. The self-tuning control algorithms and the simulation programs were written in ACCOL II, a language developed by Bristol-Babcock specifically for use with their distributed process controllers.

The processes that were simulated include an integrating process, a first-order system, a second-order system, a system with initial inverse response, and a system with variable time constant and delay. The process simulations are connected via software to the STPI module as shown in Fig. 4.1. More details of the test setup are given in Chapter 5. The simulation programs and the STPI module code have been integrated into a single ACCOL program (Appendix B).

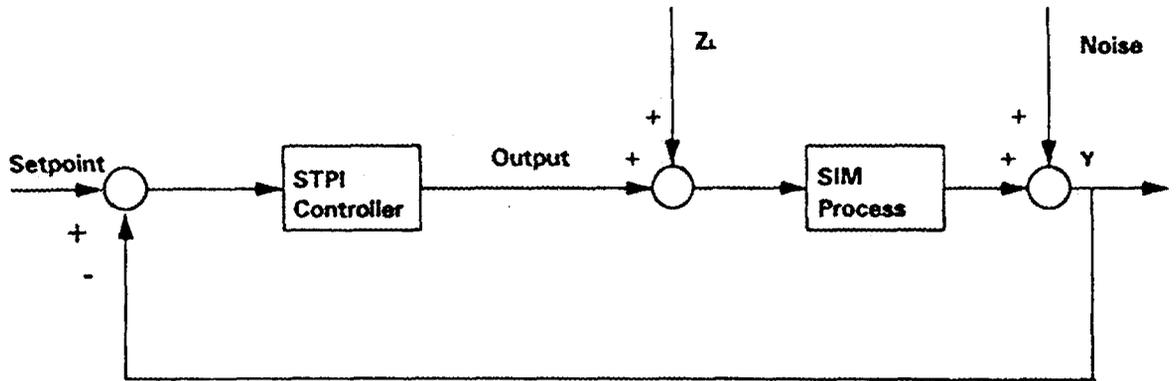
In physical processes, whenever the input to the system changes, there is frequently some time interval during which no effect can be measured or observed on the output. Thus, each of the simulations includes a delay, or deadtime, term to model the effect of this delay time.

There are usually also some known process dynamics that cannot be accounted for in a simple mathematical model (e.g., variance in properties of the inlet process materials, uncontrolled process environmental variables). These dynamics can be classified as disturbance inputs. In fact, any input that is not a result of an adjustment by the operator or the control system may be called a *disturbance input*. To account for some of these uncontrolled process dynamics and to measure their effects, each process includes load disturbances that can be added to the process inputs.

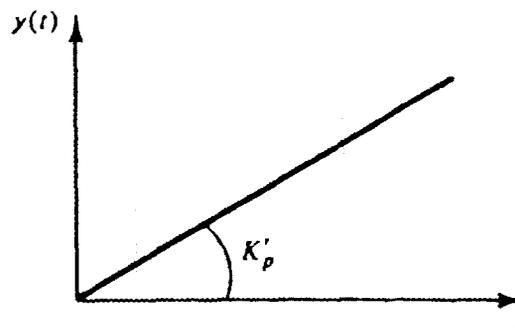
Although any unknown process dynamic could be classified as noise, one common source of noise is associated with measuring the process output. Thus, the process simulations include the capability of adding a noise signal to the process output to simulate measurement noise. This was done by using the noise generator in the GENESIS software package that is being used to monitor the process and controller output. Each process is described in more detail in the following sections.

### 4.1 INTEGRATING PROCESS

Processes with integrating action are common, especially in the chemical industry (e.g., tanks storing liquids, vessels storing gases, inventory systems storing raw materials). A purely capacitive, or integrating, process will behave as if there were an integrator between its input and output. Its output will grow (or shrink) linearly with time as shown in Fig. 4.2 (depending on whether material is being added or removed). The value of  $K_p$  (i.e., the process gain) is related to the rate of increase or decrease. The larger the value of  $K_p$ , the steeper the slope (i.e., the larger the increase) will be.



4.1. Block diagram of the connections between the self-tuning proportional-integral controller and the process simulations.



4.2. Unbounded output response of a pure integrating process. Source: George Stephanopoulos, *Chemical Process Control: An Introduction to Theory and Practice*, Fig. 10.3, p. 179, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

The simulated integrating process (with deadtime) is described by

$$G(s) = \frac{K_p e^{-\tau_d s}}{s} \quad (4.1)$$

The code to implement this process is in Task 10 of the ACCOL program in Appendix B. In practice, the process output will probably encounter some upper and lower limits (e.g., a tank has a finite capacity). So, the simulation of the integrating process has both upper and lower bounds.

## 4.2 FIRST-ORDER SYSTEM

A first-order system is so-named because the time-domain transfer function of the process can be described by a first-order differential equation. The first-order process simulation (with deadtime) is described by

$$G(s) = \frac{K_p e^{-\tau_d s}}{\tau_p s + 1} \quad (4.2)$$

Unlike the integrating process, when its input is changed, the first-order *lag* process automatically seeks a new equilibrium or steady state. The time constant  $\tau_p$  of a process is a measure of the time necessary for the process to adjust to a change in its input (Stephanopoulos 1984). The value of  $K_p$  corresponds to the ultimate or final value of the output. For a step change in input, the output response would be exponential as given by

$$y(t) = AK_p(1 - e^{-t/\tau_p}) \quad (4.3)$$

Figure 4.3 shows how the process output changes with respect to time in response to a step change in the input. The output will reach 63.2% of its final value when the elapsed time is equal to one time constant. After four time constants, the output will have essentially reached its final value. The code to implement this process is in Task 11 of the ACCOL program in Appendix B.

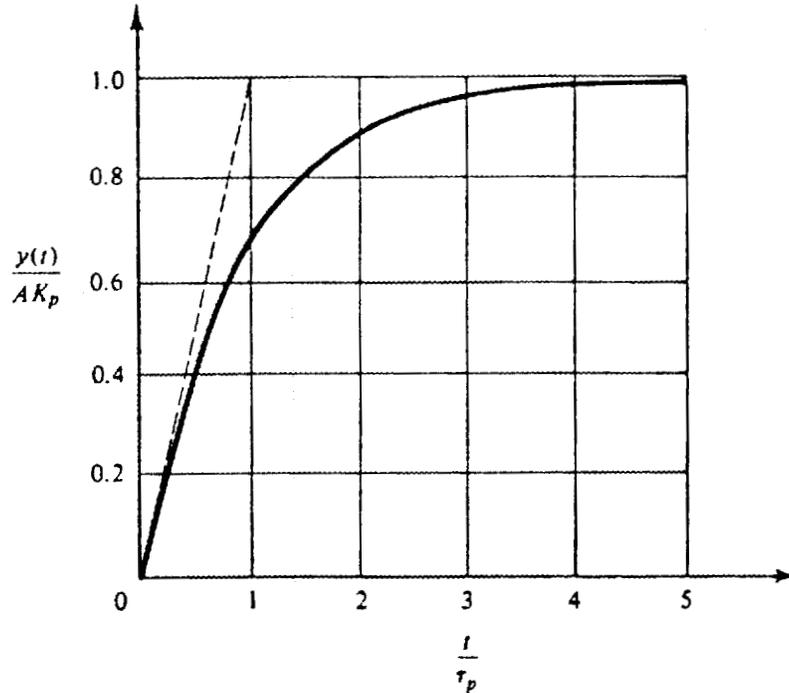
## 4.3 SECOND-ORDER SYSTEM

A second-order system is a process that can be described by a second-order differential equation. The familiar Laplace transformation for a second-order system is given by

$$G(s) = \frac{K_p \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2} \quad (4.4)$$

where

- $K_p$  = system gain,
- $\omega_n$  = undamped natural frequency,
- $\zeta$  = damping factor.



4.3. Output response of a first-order process for a step input. Source: George Stephanopoulos, *Chemical Process Control: An Introduction to Theory and Practice*, Fig. 10.4, p. 180, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

Thus, the characteristic equation is given by

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0, \quad (4.5)$$

and its roots are

$$s_1, s_2 = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}. \quad (4.6)$$

The form of the output response depends on the roots  $s_1$  and  $s_2$ , which describe the location of the two poles in the  $s$ -plane (D'Souza 1988). Three cases are easily distinguished:

Case 1: overdamped response,

Case 2: critically damped response, and

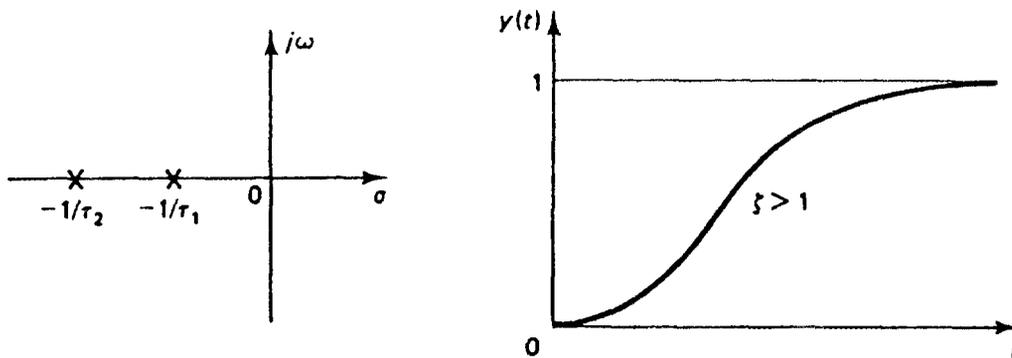
Case 3: underdamped response.

### Case 1: Overdamped response

When  $\zeta > 1$ , two distinct real poles exist (i.e., two system time constants can be defined) as shown in Fig. 4.4, and the roots can be expressed as

$$s_1 = -1/\tau_1 = -\zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1} \quad (4.7)$$

$$s_2 = -1/\tau_2 = -\zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1} .$$

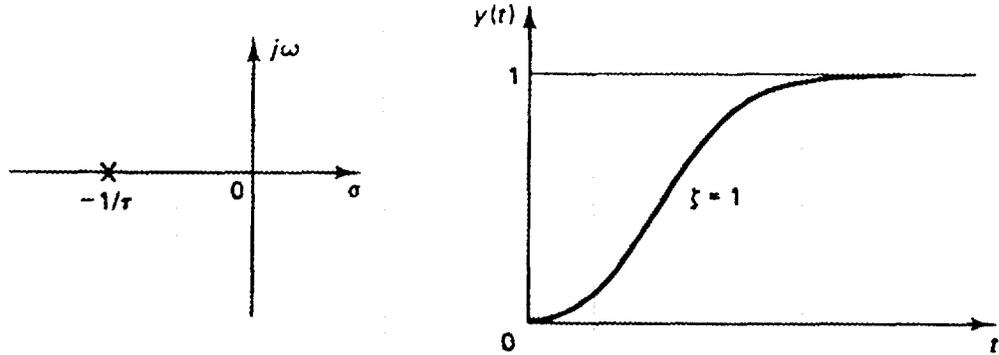


4.4. Output response of an overdamped second-order system for a step input. Source: A. Frank D'Souza, *Design of Control Systems*, Fig. 4.8, p. 139, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

### Case 2: Critically damped response

When  $\zeta = 1$ , two real, equal poles exist (i.e., a single repeated root) as shown in Fig. 4.5, and the multiple root can be expressed as

$$s_1 = s_2 = -1/\tau = -\omega_n . \quad (4.8)$$

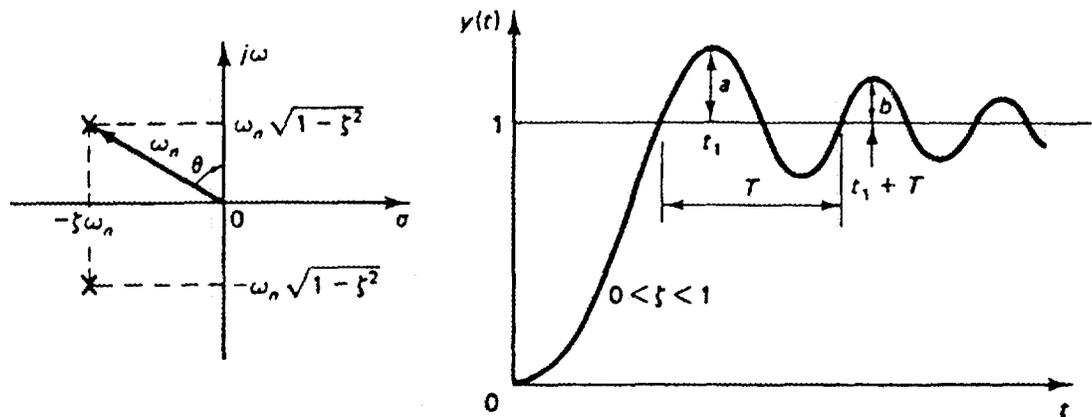


4.5. Output response of a critically damped second-order system for a step input. *Source:* A. Frank D'Souza, *Design of Control Systems*, Fig. 4.9, p. 140, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

### Case 3: Underdamped response

When  $\zeta < 1$ , two complex conjugate poles exist as shown in Fig. 4.6, and the roots can be expressed as

$$s_1, s_2 = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2}. \quad (4.9)$$



4.6. Output response of an underdamped second-order system for a step input. *Source:* A. Frank D'Souza, *Design of Control Systems*, Fig. 4.10, p. 141, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

Figure 4.7 shows the output response plots for various values of  $\zeta$ . It can be seen from the graph that for values of  $\zeta > 1$ , the response becomes more sluggish as the damping factor is increased. When  $\zeta = 1$ , the response is similar to the first-order response, except that its initial response is somewhat more sluggish. For values of  $\zeta < 1$ , the initial response is faster, but the system tends to oscillate around the final value. This oscillatory behavior becomes more pronounced as the damping factor is decreased. The code to implement this process is in Task 16 of the ACCOL program in Appendix B.

Most industrial processes can be adequately approximated by one of the three systems described above. However, to more thoroughly test the capabilities of the STPI controller, two more processes of interest were developed—a system with initial inverse response and a system with variable time constant and delay.

#### 4.4 SYSTEM WITH INITIAL INVERSE RESPONSE

The dynamic response of a boiler level-control system is quite different from those systems described thus far. If the flow rate of the cold feedwater to a boiler system is increased by a step amount, the total volume of the boiling water, and consequently the liquid level, will decrease for a short period of time before it starts to increase due to the initial cooling effect caused by adding the cold water. Thus, the system will initially have an inverse response to the desired behavior.

A system of this type can be mathematically described by the difference equation of two opposing first-order systems, yielding an overall response equal to

$$G(s) = \left[ \frac{K_1}{\tau_1 s + 1} - \frac{K_2}{\tau_2 s + 1} \right]. \quad (4.10)$$

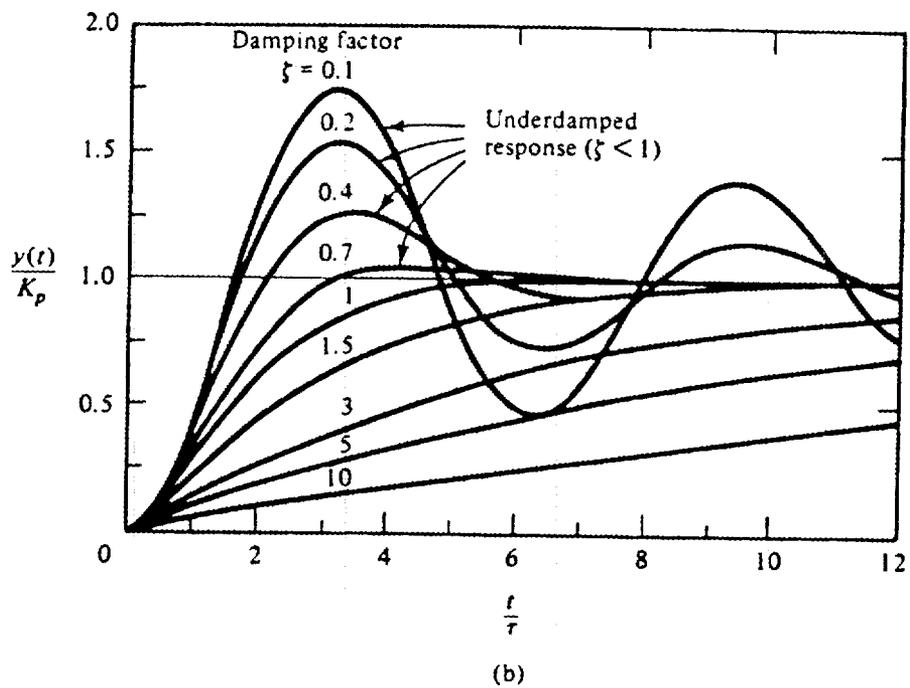
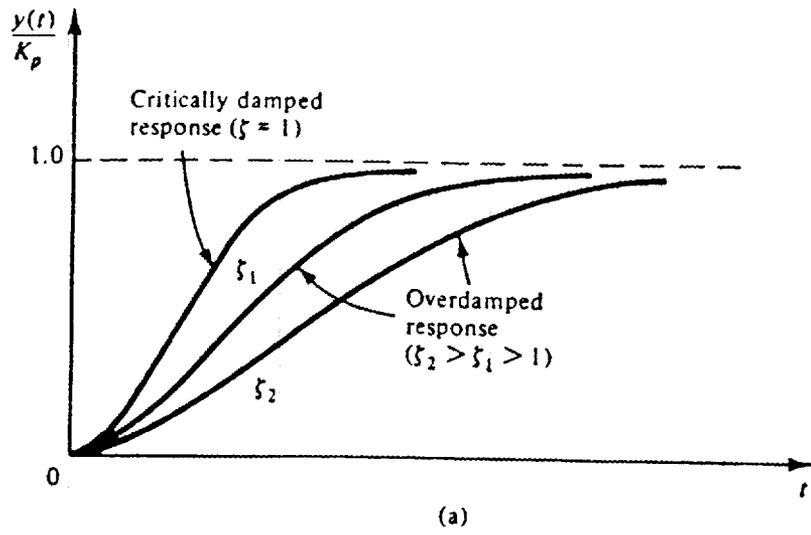
This system will have an initial inverse response when both of the following conditions are satisfied.

1. Process 1 is able to reach a higher steady-state value than Process 2 (i.e.,  $K_1 > K_2$ ) and
2. Process 2 is able to initially dominate the overall response of the system (i.e.,  $K_2/\tau_2 > K_1/\tau_1$ ).

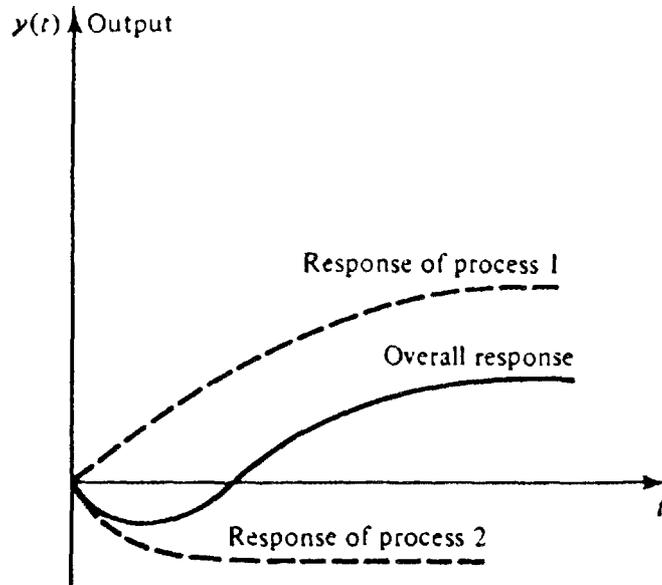
Figure 4.8 shows the overall response of the system. The code to implement this process is in Task 14 of the ACCOL program in Appendix B.

#### 4.5 SYSTEM WITH VARIABLE TIME CONSTANT AND DELAY

For the processes that have been described thus far, it has been assumed that the system parameters (e.g., gain, time constant) for physical processes always remain constant. However, this is not always the case, especially for chemical processes.



4.7. Step response plots of a second-order system for various values of the damping factor. Source: George Stephanopoulos, *Chemical Process Control: An Introduction to Theory and Practice*, Fig. 11.1, p. 189, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1984.



4.8. Initial inverse response resulting from two opposing first-order systems to a step input. *Source:* George Stephanopoulos, *Chemical Process Control: An Introduction to Theory and Practice*, Fig. 12.5, p. 219, reprinted with permission from Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

For example, consider the problem of controlling the chemical concentration of a continuously flowing output stream from a mixing tank (Fig. 4.9). The tank has two inlet streams, each of which has a distinctly different concentration of the desired chemical. A mass rate balance at the feed end of the pipe is given by

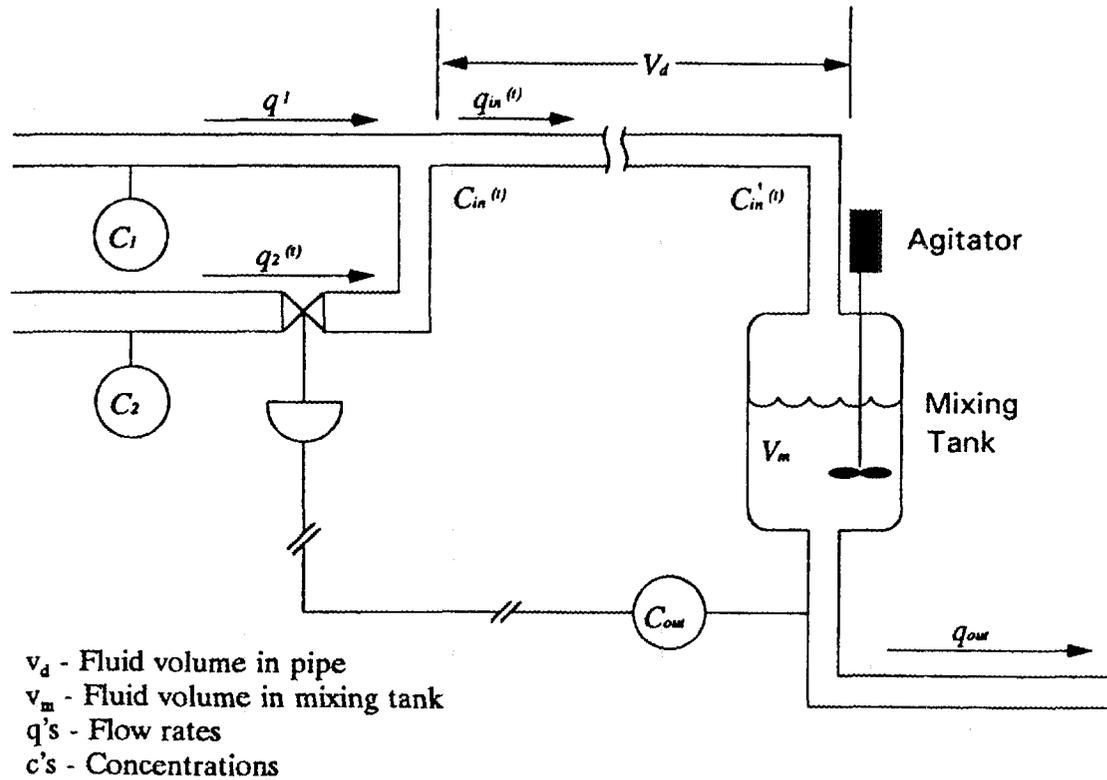
$$c_{in}(t)q_{in}(t) = c_1q_1(t) + c_2q_2(t) . \quad (4.11)$$

If only the flow rate of  $q_2$  can be controlled, then assuming the flow rate of  $q_1$  to be constant at a particular instant in time, then

$$c_{in}(t) = \frac{c_1q_1 + c_2q_2(t)}{q_{in}(t)} \quad (4.12)$$

where

$$q_{in}(t) = q_1 + q_2(t) .$$



#### 4.9. Continuous concentration control of a chemical mixing process.

To solve the problem, one can also assume that at a given instant in time, the volume in the mixing tank is constant. For a constant volume in the mixing tank, the process transfer function from the feed end of the pipe to the mixing tank outlet is given by

$$T \frac{dc(t)}{dt} = c_{in}(t - \tau) - c(t) \quad (4.13)$$

where

$$\tau(t) = \frac{V_d}{q_{in}(t)} \quad \text{and} \quad T(t) = \frac{V_m}{q_{in}(t)}$$

The Laplace transform of Eq. (4.13) is given by

$$\frac{c(s)}{c_{in}(s)} = \frac{e^{-\tau s}}{1 + Ts} . \quad (4.14)$$

Thus, the process transfer function is

$$G(s) = \frac{c_{in}(s)e^{-\tau s}}{1 + Ts} \quad (4.15)$$

where

$$c_{in}(t) = \frac{c_1 q_1 + c_2 q_2(t)}{q_1 + q_2(t)} . \quad (4.16)$$

The code to implement this process is in Task 15 of the ACCOL program in Appendix B.

## 5. TESTING AND COMPARISON OF THE BBI STPI ALGORITHMS

Each of the three STPI algorithms (i.e., closed-loop cycling, pattern recognition, and model based) was tested with each of the process simulations to determine the types of processes for which the STPI controllers might best be suited. A summary of the simulated processes is given in Table 5.1. By varying their parameters, these five processes represent a wide range of the industrial processes that would typically be encountered in industry. More details of the process simulations are provided in Chapter 3.

It is acknowledged that the practical implementation of most processes results in systems of higher order, especially when the dynamic effects of the sensors and control elements are considered. However, most industrial processes can be approximated by either a first- or second-order system with deadtime. Another reason for using low-order systems is that the fundamental concepts can be tested and understood more clearly without the additional mathematical complexity. For these reasons, most of the simulation testing concentrated on the first- and second-order systems (processes II and III).

### 5.1 DESCRIPTION OF TESTS

The process simulations are connected via software to the STPI module as shown in Fig. 4.1. The test procedure generally consisted of the following seven steps.

1. Select the desired process simulation and enter the appropriate process parameters (including percent noise and process deadtime, if desired).
2. Select the STPI algorithm to test (changing its defaults only if necessary).
3. Set initial P, I, and setpoint values and allow the process to stabilize.
4. Enable self-tuning on the STPI controller (PI values are automatically updated when self-tuning is complete).
5. Turn off self-tuning and allow the system to stabilize.
6. Test controller setpoint response with new PI values by changing setpoint from 40 to 50%.
7. Test load step response by adding 10% load disturbance (to the process input).

More details regarding these tests are given in the next section. A RESET feature was added to simplify the setup procedure (steps 1, 2, and 3), and the test procedures (steps 6 and 7) were automated to ensure repeatability.

Table 5.1. Transfer functions of the simulated processes

Simulated processes	
Process	Transfer function*
I	$\frac{K_p e^{-\tau s}}{s}$
II	$\frac{K_p e^{-\tau s}}{\tau_p s + 1}$
III	$\frac{K_p \omega_n^2 e^{-\tau s}}{s^2 + 2\zeta \omega_n s + \omega_n^2}$
IV	$\frac{K_1}{\tau_1 s + 1} - \frac{K_2}{\tau_2 s + 1}$
V*	$\frac{c_{in}(s) e^{-\tau s}}{1 + Ts}$

\*NOTE: A simplified linear approximation of a nonlinear process where

$K_p$  = Process gain,  
 $\tau_d, \tau$ , = Deadtime,  
 $T, \tau_p, \tau_1, \tau_2$  = Time constants,  
 $\omega_n$  = Undamped natural frequency,  
 $\zeta$  = Damping factor,  
 $c_1, c_2, c_{in}$  = Concentration of input streams,  
 $q_1, q_2$  = Flow rates of input streams.

## 5.2 PERFORMANCE EVALUATIONS

First, the desired process simulation was selected (process I, II, III, IV, or V), and the appropriate process parameters were entered (including percent noise and process deadtime, if desired). Then, the STPI algorithm to test was selected (changing its defaults only if necessary). Each of the three STPI algorithms is somewhat different in design, and each has several special features that may optionally be set by the user. Thus, it would be a difficult task to exhaustively compare the performance of the algorithms while varying all of their optional features. Therefore, the default settings were used for all parameters except where otherwise stated. For more details about the special features, see the report in Appendix A.

The Bristol-Babcock PID3TERM module uses a noninteracting PID control algorithm of the form

$$OUTPUT = K_p \left[ E(t) + K_I \int E(t) dt + K_D \frac{dM(t)}{dt} \right]. \quad (5.1)$$

For the STPI controller tests, the controller gain was initially set to unity (i.e.,  $P = 1.0$ ), the integral, or reset, was initially set to one repeat per minute (i.e.,  $I = 1.0$ ), and the derivative was not used (i.e.,  $D = 0$ ). All the process measurements and controller outputs were scaled from 0 to 100%, and the initial setpoint was generally set equal to 40%.

After allowing the system to stabilize with these initial tuning values, the STPI controller self-tuning was enabled. Self-tuning is performed without user intervention (as described in Chapter 4), and the PI values are automatically updated when self-tuning is complete. Tables 5.2 through 5.7 show the calculated PI tuning parameters for each of the five processes.

By comparing the calculated PI parameters in Tables 5.2 through 5.7, it seems that the results for both the closed-loop cycling method and the model-based method are generally comparable, although the model-based method generally seems to design slightly more conservative values. The values determined by the pattern recognition method are frequently widely different from the other two methods.

Upon further inspection, it was determined that the pattern recognition method was frequently unable to design useful controller parameters for the tests because it encountered  $\pm 20\%$  maximum percentage change limits which are imposed on it (i.e., it can change the PI parameters from those initially specified by the user up to a maximum of only 20% for each adaptation). The change limits are presumably imposed by the designers in an attempt to prevent it from designing erroneous results. However, these limits are a major hindrance to this algorithm when attempts are made to use it from a cold start.

### 5.2.1 Process Incompatibilities

If the process is naturally integrating, the user can set the INTEG flag in the ACCOL program before activating either the closed-loop cycling or model-based self-tuning algorithms to indicate that the process is naturally integrating (the INTEG flag

Table 5.2. Calculated proportional-integral (PI) parameters for the integrating process

Controller parameters for Process I				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 0.2$ $\tau_p = 0.0$	Gain ( $K_c$ )	1.740	1.200	0.5514
	Integ ( $K_I$ )	6.264	1.200	19.52
	CF	70.51%	0.0%	58.84%
$K_p = 1.0$ $\tau_p = 0.0$	Gain ( $K_c$ )	0.5583	0.8320	0.3396
	Integ ( $K_I$ )	9.019	0.8333	30.66
	CF	56.53%	21.27%	60.62%

CF = Confidence factor.  
 $K_c$  = Controller gain.  
 $K_I$  = Controller integral.  
 $K_p$  = Process gain.  
 $\tau_p$  = Process time constant.

Table 5.3. Calculated proportional-integral (PI) parameters for the first-order process

Controller parameters for Process II				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 0.0$ OVERSH = 10%	Gain ( $K_c$ )	1.556	1.200	2.506
	Integ ( $K_I$ )	11.27	1.200	11.96
	CF	94.72%	0.0%	99.95%
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 0.0$ OVERSH = 0%	Gain ( $K_c$ )	1.548	1.0	0.5672
	Integ ( $K_I$ )	2.038	1.0	10.90
	CF	51.35%	0.0%	99.85%

CF = Confidence factor.  
 $K_c$  = Controller gain.  
 $K_I$  = Controller integral.  
 $K_p$  = Process gain.  
 $\tau_d$  = Deadtime.  
 $\tau_p$  = Process time constant.  
OVERSH = Overshoot.

Table 5.4. Calculated proportional-integral parameters (PI) for the second-order process ( $\omega_n = 0.2$ )

Controller parameters for Process III ( $\omega_n = 0.2$ )				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 1.0$ $\tau_d = 0.0$ $\zeta = 0.2$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	0.2855 7.046 70.38%	0.9434 1.200 0.0%	0.1486 2.469 97.48%
$K_p = 1.0$ $\tau_d = 0.0$ $\zeta = 1.0$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.173 4.698 68.90%	1.200 1.200 0.0%	1.178 5.663 99.40%
$K_p = 1.0$ $\tau_d = 0.0$ $\zeta = 2.5$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	3.523 4.698 50.55%	1.200 1.200 0.0%	3.093 2.214 99.94%

CF = Confidence factor.  
 $K_c$  = Controller gain.  
 $K_I$  = Controller integral.

$K_p$  = Process gain.  
 $\tau_d$  = Deadtime.  
 $\zeta$  = Damping factor.

Table 5.5. Calculated proportional-integral parameters (PI) for the second-order process ( $\omega_n = 0.04$ )

Controller parameters for Process III ( $\omega_n = 0.04$ )				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 1.0$ $\tau_d = 0.0$ $\zeta = 0.2$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	0.3830 1.427 60.14%	0.8929 0.9356 71.43%	3.583 0.4287 98.14%*
$K_p = 1.0$ $\tau_d = 0.0$ $\zeta = 1.0$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.640 1.303 82.24%	1.200 1.200 0.0%	0.5741 2.143 87.87%
$K_p = 1.0$ $\tau_d = 0.0$ $\zeta = 2.5$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	3.939 1.181 64.11%	1.200 1.199 17.22%	20.98 1.106 25.63%*

CF = Confidence factor.  
 $K_c$  = Controller gain.  
 $K_I$  = Controller integral.  
 $K_p$  = Process gain.

$\tau_d$  = Deadtime.  
 $\zeta$  = Damping factor.  
 \* = Calculated value is unstable.

**Table 5.6. Calculated proportional-integral parameters (PI) for the system with initial inverse response**

Controller parameters for Process IV				
Process parameters	PI parameters	CLC	PR	MB
$K_1 = 2.0$	Gain ( $K_c$ )	0.8423	0.9450	0.4237
$\tau_1 = 5.0$	Integ ( $K_I$ )	5.500	0.9677	11.43
$K_2 = 1.0$	CF	78.32%	80.69%	99.95%
$\tau_2 = 0.5$				

CF = Confidence factor.

$K_1$  = Process 1 gain.

$K_2$  = Process 2 gain.

$K_c$  = Controller gain.

$K_I$  = Controller integral.

$\tau_1$  = Process 1 time constant.

$\tau_2$  = Process 2 time constant.

**Table 5.7. Calculated proportional-integral (PI) parameters for the system with variable time constant and delay**

Controller parameters for Process V				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 0.55$	Gain ( $K_c$ )	9.270	1.200	2.738
$c_1 = 10\%$	Integ ( $K_I$ )	3.089	1.200	1.356
$c_2 = 90\%$	CF	79.84%	0.0%	98.77%
$q_1 = 10.0$				
$V_d = 100$				
$V_m = 1000$				

$C_1$  = Input stream 1 concentration.

$C_2$  = Input stream 2 concentration.

CF = Confidence factor.

$K_c$  = Controller gain.

$K_I$  = Controller integral.

$K_p$  = Process gain.

$q_1$  = Flow rate of stream 1.

$V_d$  = Fluid volume in pipe.

$V_m$  = Fluid volume in mixing tank.

has no effect on the pattern recognition algorithm). In the case of the closed-loop cycling algorithm, the controller will use a square-wave perturbation output (instead of the sawtooth waveform). In either case, a proportional-only controller should be designed. However, some problems were experienced when attempting to use the STPI algorithms' INTEG feature to design a proportional-only controller (i.e., the feature did not seem to work reliably). Therefore, the INTEG feature was not used, and PI controllers were designed to control the integrating process as well as for all other processes.

Another limitation of the STPI controller algorithms, at least in their present implementation, is that they are not suitable for controlling fast processes. However, the one-second update will probably pose no problem for most industrial processes where the Bristol-Babcock controller is generally used. One would also expect the STPI algorithms to execute somewhat faster once they are commercially implemented (in microcode in PROMs) than they do when written in ACCOL.

### 5.2.2 Tuned System Performance

Several simple performance specifications can be used to evaluate characteristic features of the closed-loop system response (e.g., overshoot, rise time, settling time, decay ratio). However, several performance criterion also can be used to simultaneously minimize multiple requirements. The most popular criteria used to evaluate the overall quality of the tuned system response are

1. integral of the square error (ISE), where

$$ISE = \int_0^{\infty} e^2(t) dt ; \quad (5.2)$$

2. integral of the absolute value of the error (IAE), where

$$IAE = \int_0^{\infty} |e(t)| dt ; \quad (5.3)$$

3. integral of the time-weighted absolute error (ITAE), where

$$ITAE = \int_0^{\infty} t |e(t)| dt . \quad (5.4)$$

The determination of which criterion is best to use depends upon which characteristics of a particular process are the most important to control.

The ISE criterion strongly penalizes large errors because the errors are squared and thus contribute more to the value of the integral (however, small errors of less than one would actually be downplayed). The IAE criterion penalizes small errors the same as large errors. The ITAE criterion severely penalizes errors that persist for a long time. The IAE criterion seems to have the most practical significance because it gives a more

accurate indication of the actual error (e.g., the area under the curve that can be directly related to operating costs). For this reason, the IAE criterion was used to evaluate the STPI algorithms' performance.

A controller that is tuned to provide optimum setpoint response (to step changes in the setpoint) will not necessarily provide good load-step response. So, the controllers' responses to both setpoint changes and load-step response were tested. During these response tests, the controller's output and the process measurement variable were sampled simultaneously at one-second intervals. The STPI algorithms' performance was then evaluated by analyzing the tuned system IAE response data with MATLAB. The results of the response tests are in Table 5.8.

From Table 5.8, it can be seen that the pattern recognition outperformed the other two algorithms only once (for Process III, with  $\omega_n = 0.04$  and  $\zeta = 0.2$ ). Upon close observation of the data in Tables 5.2 through 5.7, it is obvious that this algorithm yields unreliable tuning results when used from a cold start. With this information, it is safe to say that this algorithm should probably not be used from a cold start, but only for continuous tuning refinements.

The rest of the test results show that the closed-loop cycling algorithm outperformed the model-based algorithm in 10 of the 15 other test cases. Although the model-based algorithm generally yielded results comparable to those designed by the closed-loop cycling method, it failed to design a controller with stable PI parameters for three of the tested processes. Thus, it appears from these tests that the closed-loop cycling algorithm will generally yield the best results.

### 5.2.3 Deadtime Effects

Various amounts of process deadtime were added to the first-order process to examine the effects of deadtime on the STPI algorithms. The results of these tests are shown in Table 5.9. The data in Tables 5.8 and 5.9 indicate that the closed-loop cycling algorithm is more likely to design better PI parameters for processes with deadtime. The pattern recognition method was unable to design useful controller parameters for these tests because it always encountered the  $\pm 20\%$  maximum change limits.

Note that the model-based algorithm actually computes unstable PI parameters for the test with 20 seconds of deadtime. This is an inherent limitation of the implementation of this algorithm. Specifically, because the developers fixed the number of numerator terms to five, only the deadtime information contained in the previous four time samples is available to model the deadtime. Thus, processes with a large amount of deadtime (relative to the process time constant) cannot be accurately modeled by the algorithm.

### 5.2.4 Noise Effects

Measurement noise was added to the first-order process output to observe the STPI algorithms' sensitivity to measurement noise. The results of these tests are shown in Table 5.10. Examination of these data along with the performance results obtained

Table 5.8. Tuned system integral of the absolute value of the error (IAE) response to both setpoint and load changes

Process and parameters*	IAE Response for setpoint/(load) changes		
	CLC	PR	MB
Process I $(\tau_d = 0)$ $K_p = 1.0$ $K_p = 0.2$	29.7 (122.3) 47.7 (57.4)	26.2 (707.8) 76.8 (388.6)	58.9 (108.9) 122.5 (131.5)
Process II $(K_p = 1, \tau_p = 5)$ OVERSH = 10% OVERSH = 0% $\tau_d = 5.0$ $\tau_d = 20.0$ NOISE = 2% NOISE = 5%	31.0 (36.5) 161.7 (178.3) 118.8 (113.3) 415.1 (416.1) 20.0 (32.7) 25.9 (24.6)	285.4 (235.8) 400.5 (324.4) 287.0 (252.9) UNSTABLE 285.4 (235.8) 285.4 (235.8)	9.9 (21.1) 86.8 (98.6) 277.0 (277.9) UNSTABLE 13.5 (25.7) 48.5 (49.3)
Process III $(K_p = 1, \tau_d = 0)$  $(\omega_n = 0.2)$ $\zeta = 0.2$ $\zeta = 1.0$ $\zeta = 2.5$  $(\omega_n = 0.04)$ $\zeta = 0.2$ $\zeta = 1.0$ $\zeta = 2.5$	285.1 (398.9) 92.0 (110.5) 70.2 (38.5)  857.3 (867.1) 288.5 (233.8) 337.5 (132.5)	391.7 (351.7) 295.4 (244.8) 315.6 (260.8)  687.1 (883.7) 351.9 (299.6) 660.5 (522.8)	754.9 (415.3) 68.8 (92.8) 62.1 (86.8)  UNSTABLE 482.3 (446.0) UNSTABLE
Process IV $K_1 = 2.0$ $\tau_1 = 5.0$ $K_2 = 1.0$ $\tau_2 = 0.5$	135.8 (172.3)	359.6 (328.3)	114.6 (150.9)
Process V $K_p = 0.55$ $c_1 = 10\%$ $c_2 = 90\%$ $q_1 = 10.0$ $V_d = 100$ $V_m = 1000$	78.0 (17.1)	550.1 (40.4)	340.9 (56.0)

\*See definitions on the next page.

Definitions for Table 5.8:

- $C_1$  = Input stream 1 concentration.  
 $C_2$  = Input stream 2 concentration.  
 $K_1$  = Process 1 gain.  
 $K_2$  = Process 2 gain.  
 $K_p$  = Process gain.  
 OVERSH = Overshoot.  
 $q_1$  = Flow rate of stream 1.  
 $V_d$  = Fluid volume in pipe.  
 $V_m$  = Fluid volume in mixing tank.  
 $\omega_n$  = Undamped natural frequency.  
 $\tau_1$  = Process 1 time constant.  
 $\tau_2$  = Process 2 time constant.  
 $\tau_d$  = Deadtime.  
 $\tau_p$  = Process time constant.  
 $\zeta$  = Damping factor.

Table 5.9. Calculated proportional-integral (PI) parameters for the first-order process with deadtime

Controller parameters for Process II (with deadtime)				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 0.0$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.556 11.27 94.72%	1.200 1.200 0.0%	2.506 11.96 99.95%
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 5.0$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.035 5.184 80.15%	1.200 1.200 0.0%	0.4445 4.583 92.14%
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 20.0$	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	0.7103 2.147 79.28%	1.200 1.200 18.22%*	3.126 27.23 13.56%*

- CF = Confidence factor.  
 $K_c$  = Controller gain.  
 $K_I$  = Controller integral.  
 $K_p$  = Process gain.  
 $\tau_d$  = Deadtime.  
 $\tau_p$  = Process time constant.  
 \* = Calculated value is unstable.

**Table 5.10. Calculated proportional-integral (PI) parameters for the first-order process with noise**

Controller parameters for Process II (with noise)				
Process parameters	PI parameters	CLC	PR	MB
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 0.0$ NOISE = 0%	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.556 11.27 94.72%	1.200 1.200 0.0%	2.506 11.96 99.95%
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 0.0$ NOISE = 2%	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.924 10.02 51.22%	1.200 1.200 0.0%	1.822 14.76 96.87%
$K_p = 1.0$ $\tau_p = 5.0$ $\tau_d = 0.0$ NOISE = 5%	Gain ( $K_c$ ) Integ ( $K_I$ ) CF	1.612 16.11 57.03%	1.200 1.200 0.0%	0.4356 31.286 31.06%

CF = Confidence factor.

$K_c$  = Controller gain.

$K_I$  = Controller integral.

$K_p$  = Process gain.

$\tau_d$  = Deadtime.

$\tau_p$  = Process time constant.

for these tests in Table 5.8, indicates that for small amounts of noise the performance of the model-based algorithm is best, but for larger amounts of noise the closed-loop cycling algorithm seems to yield better results. The pattern recognition method was once again unable to design useful controller parameters for these tests because it always encountered the  $\pm 20\%$  maximum change limits.

### 5.3 ROBUSTNESS COMPARISONS

The two primary parameters that greatly affect the value of the PI tuning constants are the process gain and deadtime. PI controllers can accommodate any decrease in process gain or deadtime without destabilizing the loop, although the output response will become more sluggish. However, the amount by which either of these two process parameters can be increased is much more important. The amount by which process (or controller) gain or deadtime can be increased before reaching the stability limit of the process is a measure of the robustness of the system (using the specified tuning constants).

In fact, system performance and robustness are inversely related (Shinsky 1991). Performance can generally be improved by increasing the gain ( $P$ ) and integral ( $I$ ) until the desired system performance is obtained. However, robustness can usually be improved by detuning the controller, although performance will be decreased. Thus, to determine which STPI controller algorithm is really best for a particular process, it is necessary to examine both the performance and robustness of the tuned systems simultaneously, considering which of these characteristics is most important to control for that particular process.

The robustness of the tuned system was determined by using MathCAD with the following method. Given a plant  $[G_p(s)]$ , and controller  $[G_c(s)]$ , find the maximum amount the gain could be increased (i.e., the gain limit  $K_l$ ), and the maximum amount of deadtime that could be added (i.e., the deadtime limit  $T_l$ ) before the closed-loop system reaches the stability limit.

The gain limit  $K_l$  can be determined by computing where the magnitude of the open-loop system is equal to 1.0:

$$|K_l G_p(j\omega_z) G_c(j\omega_z)| = 1 \quad (5.5)$$

To solve this equation for  $K_l$ , one must know the zero crossing frequency  $\omega_z$ . This is, of course, described as the point where the phase is equal to  $-\pi$  rad/s:

$$\arg[K_l G_p(j\omega_z) G_c(j\omega_z)] = -\pi \quad (5.6)$$

Because Eq. 5.6 is independent of  $K_l$  (i.e.,  $K_l$  does not affect the open-loop phase), this equation can be solved using a linear search algorithm to obtain  $\omega_z$ , then the gain limit can be calculated by solving Eq. 5.5 for  $K_l$ :

$$K_l = \frac{1}{|G_p(j\omega_z) G_c(j\omega_z)|} \quad (5.7)$$

Similarly, the deadtime limit  $T_l$  can be determined by

$$|K_l G_p(j\omega_z) G_c(j\omega_z) e^{-j\omega_z T_l}| = 1 \quad (5.8)$$

such that

$$\arg[G_p(j\omega_z) G_c(j\omega_z) e^{-j\omega_z T_l}] = -\pi \quad (5.9)$$

Because Eq. 5.8 is independent of  $T_l$ , it can be solved by using a linear search algorithm to obtain  $\omega_z$ . Then, noting that

$$\arg[G_p(j\omega_z) G_c(j\omega_z) e^{-j\omega_z T_l}] = \arg[G_p(j\omega_z) G_c(j\omega_z)] - \omega_z T_l \quad (5.10)$$

the deadtime limit can be calculated by solving Eq. 5.9 for  $T_1$ , as

$$T_1 = \frac{\pi + \arg[G_p(j\omega_z)G_c(j\omega_z)]}{\omega_z} \quad (5.11)$$

Although the above method calculates the gain limit and the deadtime limit independently, both the gain and deadtime could be increased simultaneously (by lesser amounts) to drive the closed-loop system to the stability limit. The MathCAD routines used to calculate these limits for both the first- and second-order systems are given in Appendix D. Table 5.11 shows the results of the robustness calculations. Because the pattern recognition method rarely yielded reliable results from a cold start, the robustness calculations compared the results from only the closed-loop cycling and the model-based tuning methods.

The tuned system is generally considered to be robust if the system remains stable when the gain (or deadtime) is increased by a factor of 2. Although no process deadtime was specified except for two of the second-order process tests, these calculations still give an overall indication of the robustness of the STPI algorithms. Comparison of the data in Table 5.11 indicates that the two algorithms are nearly equally robust, with one or the other having the edge for a particular process. Of the 11 tests, the closed-loop cycling values were more robust for six of the processes, while the model-based technique yielded better results for 5 processes. The gain limit is, of course, infinite for the integrating process and for first-order processes with no deadtime (because a  $-180^\circ$  phase shift would be approached for only extremely high values of gain). It was considered to be infinite for other processes if the gain could be increased by an extremely large amount before the system became unstable.

#### 5.4 PI PARAMETER ADJUSTMENT EFFICIENCY

The pattern recognition algorithm is the most time-efficient because it determines new PI values after each setpoint change or disturbance without any further process perturbation. However, this algorithm frequently failed to identify reliable PI values (when used from a cold start). In their present implementations, the closed-loop cycling algorithm is more efficient than the model-based algorithm. This is because the closed-loop cycling algorithm automatically terminates after the process oscillations have stabilized to update the PI values, whereas the model-based algorithm just calculates new PI values after a specified number of cycles.

**Table 5.11 Tuned system robustness with respect to gain and deadtime increases**

Process parameters	Gain/(deadtime) stability limits			
	CLC		MB	
	Gain limit	DT limit	Gain limit	DT limit
Process I ( $\tau_d = 0$ ) $K_p = 1.0$ $K_p = 0.2$	$\infty$ $\infty$	2.28 3.56	$\infty$ $\infty$	1.56 2.74
Process II ( $K_p = 1, \tau_p = 5$ ) OVERSH = 10% OVERSH = 0% $\tau_d = 5.0$ $\tau_d = 20.0$ NOISE = 2% NOISE = 5%	$\infty$ $\infty$ 1.97 1.56 $\infty$ $\infty$	5.23 8.84 4.21 42.0 4.42 4.07	$\infty$ $\infty$ 4.66 * $\infty$ $\infty$	3.14 15.3 39.6 * 3.86 6.55
Process III ( $K_p = 1, \tau_d = 0$ ) ( $\omega_n = 0.2$ ) $\zeta = 0.2$ $\zeta = 1.0$ $\zeta = 2.5$	7.49 $\infty$ $\infty$	49.6 11.8 7.7	$\infty$ $\infty$ $\infty$	275.9 10.1 11.6

$K_p$  = Process gain.

$\tau_p$  = Process time constant.

OVERSH = Overshoot.

$\omega_n$  = Undamped natural frequency.

$\tau_d$  = Deadtime.

$\zeta$  = Damping factor.

\* = Not calculated because tuned system was unstable.

## 6. CONCLUSIONS

Of the three STPI algorithms, the closed-loop cycling technique is the most reliable and the easiest to use because it only requires the user to specify the setpoint (when operated with the default parameters). It is also good from a cold start. Another good feature of this algorithm is that it terminates and updates the PI parameters as soon as the self-tuning is complete (i.e., when it has identified the period and amplitude of the constant process oscillations). This algorithm seems to have been implemented well, but the initial default relay amplitude frequently seemed to be too large for the processes that were tested.

Although the algorithm automatically reduces the relay amplitude when the specified initial amplitude is too large, when output limits are incurred it only reduces the amplitude by a predetermined factor. Thus, if the initial amplitude is much too large, several successive automatic amplitude reductions will be needed before the process variable begins to stay near the setpoint. Some form of intelligent amplitude reduction should be done analytically. One rather simple method would be to approximate the slope of the output response, compare it to the amount of time the output stays out of range, and then calculate the relay amplitude reduction factor.

Another minor disadvantage of the closed-loop cycling algorithm is that it is the only one of the three that simply cannot be used in continuous self-tuning mode. The self-tuning must be initiated by the user.

The pattern recognition algorithm is the only one of the three that does not cause any process disturbance during self-tuning. When this algorithm is activated by the user, it recalculates the PI controller parameters following any sufficiently large setpoint change or disturbance. Note that it is also the only one that stays on continuously until it is turned off by the user. This could be either an advantage or a disadvantage, depending on one's viewpoint (especially because it will retune following any sufficiently large process disturbance). It is also the most time-efficient because it determines new PI values after each setpoint change or disturbance without any further process perturbation.

The pattern recognition algorithm requires reasonably good initial values of the controller PI parameters and is therefore not suitable for use from a cold start. The algorithm also limits the adjustment of the PI values (the maximum change allowed after each adaptation) to  $\pm 20\%$ . It is the only one of the three algorithms that currently has any parameter change limits. A serious disadvantage of this algorithm is that it does not work when the process response is overdamped. The particular implementation of this algorithm is rather simplistic. However, its usefulness is greatly enhanced by using the closed-loop cycling method as a pretuning phase to obtain reasonably good estimates for the initial  $P$  and  $I$  values.

This algorithm could also be improved by adding some additional logic or heuristics (similar to those implemented in the Foxboro EXACT self-tuner) to enable it to work when the process is overdamped. Or, for overdamped processes, perhaps the pattern recognition algorithm could increase the controller gain until the required oscillatory response is obtained and then perform the self-tuning in the same fashion. The model-based algorithm can be used from a cold start, although it requires more values to be specified by the user than the other algorithms. Because the selection of an appropriate sampling rate (i.e., ACCOL task rate) is extremely important to the proper

operation of this algorithm, it should be modified to automatically approximate the response time of the process (with a step response) and adjust the ACCOL task rate accordingly. The STPI research report claims that this algorithm can also be used to provide continuous tuning refinements (by the expert user). However, no continuous parameter refinement tests were attempted with this algorithm, because of time constraints.

One serious disadvantage of this algorithm is that it cannot properly tune processes that have large amounts of deadtime (see additional explanation in Sect. 5.2.3). A potential problem with this algorithm is that it employs the pole-zero cancellation technique. This technique has the inherent disadvantage that if the process model is incorrectly identified, or if the process model dynamically changes over time, the pole-zero cancellation may not work and the tuned system may then be unstable.

The mean level of the PRBS could also be monitored and automatically adjusted during the tuning phase (as done in the closed-loop cycling algorithm) to keep the process variable near the setpoint. If intelligent PRBS amplitude adjustment is added, it should also be able to increase the amplitude if the initially specified value only causes very small deviations from the setpoint. Deviations of at least 3 to 5% would most likely result in better model estimation.

For slower processes, this algorithm's self-tuning takes an unacceptably long period of time (using the default values) even though it adequately approximated the model after the first few cycles. The algorithm should be modified to terminate model identification and update the PI parameters whenever the confidence factor (DONE) reaches some acceptable value (perhaps 85%) rather than just continuing to update these values for a specified number of cycles (the default number of cycles is 50).

Although not as critical, the coefficient  $\alpha$  in the digital bandpass filter could also be adjusted recursively as new estimates of the process model are obtained to obtain even better models (and thus more precise tuning).

As suggested in the original STPI research report, these tests confirmed that some additional logic should probably be added to check the confidence factor, DONE, before updating the controller parameters. In their present implementation, the PI values will be updated even if the algorithm practically fails. Note that although the confidence factor does give some indication as to the reliability of the tuning parameters for a particular STPI algorithm use, it should not be used to compare the performance or robustness of one STPI algorithm to another.

It might also be desirable to allow the user to limit the range for the PI parameters or specify the maximum percentage change allowed after each adaptation for both the closed-loop cycling and model-based algorithms (the pattern recognition algorithm already limits the change to  $\pm 20\%$ ).

In summary, these tests demonstrated that some good single-loop adaptive control techniques have been developed that can be used to adequately control many processes. Although it is certain that single-loop self-tuning controllers will not be enough to solve every process control problem, it may be possible to meet increased demands and achieve better process control results simply by using one of these single-loop advanced control techniques. Because most industrial processes are still being controlled with single-loop PID controllers, perhaps one of these techniques can be implemented to obtain the desired efficiency improvements without costly redesign of existing processes.

It should be noted that these tests actually evaluated the particular implementation of these STPI algorithms and their interactions with the Bristol-Babcock PID control algorithm. Different implementations of these same algorithms could provide somewhat different results. Likewise, if the same implementations of these STPI algorithms were used in conjunction with other control algorithms of a different form, widely differing results may be obtained.

## 7. REFERENCES

- Aström, K. J., and Hägglund, T. 1988. *Automatic Tuning of PID Controllers*, Instrument Society of America, Research Triangle Park, N.C.
- Aström, K. J., and Wittenmark, B. 1989. *Adaptive Control*, Addison-Wesley, Reading, Mass.
- D'Souza, A. F. 1988. *Design of Control Systems*, Prentice-Hall, Englewood Cliffs, N.J.
- Shinskey, F. G. 1991. "Evaluating Feedback Controllers Challenges Users and Vendors," *Control Eng.*, 75–78 (September).
- Stephanopoulos, G. 1984. *Chemical Process Control: An Introduction to Theory and Practice*, Prentice-Hall, Englewood Cliffs, N.J.
- Ziegler, J. G., and Nichols, N. B. 1942. "Optimum Settings for Automatic Controllers," *Trans. ASME*, 759–65 (November).

## 8. BIBLIOGRAPHY

- Aström, K. J., and Hägglund, T., "A New Auto-Tuning Design," *IFAC Adaptive Control of Chemical Processes* 141–46 (1988).
- Gerry, J. P., "Find Out How Good That Tuning Really Is," *Control Eng.* 69–71 (July 1987).
- Gupta, M. M., *Adaptive Methods for Control System Design*, IEEE Press, New York, 1986.
- Hang, C. C., and Aström, K. J., "Practical Aspects of PID Auto-Tuners Based on Relay Feedback," *IFAC Adaptive Control of Chemical Processes* 153–58 (1988).
- Hang, C. C., and Sin, K. K., "A Comparative Performance Study of PID Auto-Tuners," *IEEE Control Syst.* 41–47 (August 1991).
- Kaya, A., and Scheib, T. J., "Tuning of PID Controls of Different Structures," *Control Eng.* 62–65 (July 1987).
- Kraus, T. W., "Self-Tuning Control Using an Expert System Approach," *Meas. Control* 172–75 (June 1985).
- Kraus, T. W., and Myron, T. J., "Self-Tuning PID Controller Uses Pattern Recognition Approach," *Control Eng.* 106–11 (June 1984).
- McMillan, G. K., *Tuning and Control Loop Performance*, Instrument Society of America, Research Triangle Park, N.C., 1983.
- Miller, J. A., et al. "A Comparison of Controller Tuning Techniques," *Control Eng.* 72–75 (December 1967).
- Morris, H. M., "How Adaptive are Adaptive Process Controllers?" *Control Eng.* 96–100 (March 1987).
- Price, V. A., "Automatic Tuning Simplifies Process Control," *InTech* 9–16 (September 1988).
- Vermeer, P. J., Morris, A. J., and Shah, S. L., "Adaptive PID Control—A Pole Placement Algorithm with a Single Tuning Parameter," *IFAC Adaptive Control of Chemical Processes*, 159–64 (1988).
- Wade, H. L., "High-Capability Single-Station Controllers: A Survey," *InTech* 106–14 (September 1988).
- Warwick, K., *Simplified Self-Tuning Algorithms*, OUEL 1657/86, University of Oxford, Oxford, England, 1986.



**Appendix A**

**DEVELOPMENT OF ACCOL SELF-TUNING PI (STPI) CONTROL MODULE**



DEVELOPMENT OF ACCOL SELF-TUNING  
PI (STPI) CONTROL MODULE

Authors

Mr. C.S. Cox  
Mr. W.J.B. Arden  
Dr. I.G. French  
Dr. I. Fletcher  
Mr. A.R. Boucher

Control Systems Centre,  
School of Elect. Eng. & Applied Physics,  
Sunderland Polytechnic,  
Sunderland, SR1 3SD.

Tel: 091-515-2824

Fax: 091-515-2423

APPENDIX A.1

Contents of Report

Part I - Development and Operation of STPI Module

Part II - Summary of Field Trials

Part III - Closed Loop Cycling Algorithm Theory

Part IV - Pattern Recognition Algorithm Theory

Part V - Model Based Algorithm Theory

PART I

DEVELOPMENT AND OPERATION OF STPI MODULE

## Part I - Contents

1. Introduction .....	1
(a) Closed Loop Cycling Algorithm .....	4
(b) Pattern Recognition Algorithm .....	5
(c) Model Based Algorithm .....	6
2. Configuring a Self-Tuning PI Controller within ACCOL ...	8
3. Operation of STPI Module .....	10
4. Auxiliary Signal Lists .....	12
Figures .....	16

## 1. INTRODUCTION

The basis of the vast majority of today's commercial controllers and PLC's is the microprocessor. The new families of cheap powerful processors have produced environments suitable for the development of both fixed-parameter controllers, often with advanced features such as feedforward control and wind-up protection, or, those possessing 'self-tuning' capabilities. The idea behind self-tuning is to adjust the controller settings automatically, based on the measured input/output behaviour of the process under control. Fig. 1 presents the general self-tuning structure favoured by most academic researchers. The idea of a self-tuner has been with us for some time, the solution to the extra data-processing requirements has only been economically feasible in recent years.

The rapid advancement of microprocessor technology has re-stimulated the interest in digital control implementation. New control laws have been postulated but industry still appears to favour a digitisation of the well known continuous time PID three term controller. This dilemma has led to two contrasting approaches to the use of this new computational power. The first is to add tuning features to an otherwise standard PI(D) regulator. This approach recognises that the majority of regulators used in industry are still of the PID form and complex processes may have hundreds of regulators. However, even after careful instruction, instrument engineers and plant operators often still have difficulty in installing and operating such

regulators. A feedback control system is of little value if it is improperly tuned. Several different methods have been proposed for tuning PID regulators. The need in tuning a controller is to determine the 'optimum' values of the controller gain  $K_c$  (or the proportional band  $PB$ ), the reset time  $T_i$  (or the reset rate in repeats per minute) and the derivative time  $T_d$ . The adjustment of these tuning parameters on feedback controllers is one of the least understood yet extremely important aspects of automatic control theory. Several methods for manually tuning these algorithms are used in practice, ranging from 'trial-and-error' to the more systematic use of empirical formulae such as those proposed by Ziegler and Nichols (1943). However for some complex processes, where the plant dynamics vary significantly in the course of their operation, automatic retuning is the only real answer in order to maintain a consistent final product. The second philosophy is to provide a general purpose control law which is in some sense optimal. By careful 'tailoring' of these control laws, acceptable performance may be achievable in those situations where PI(D) may not function too well, e.g. processes with long time delays. These tuners might involve several design parameters which are used to prescribe the characteristics of the closed loop control system rather than direct entry of the controller gains, as is done with the standard PID law. Such general purpose techniques include: (i) Pole-Placement (PP), (ii) Linear Quadratic Gaussian (LQG), (iii) Generalised Minimum Variance (GMV), (iv) Long Range Predictive Control (LPRC) and (v) Generalised Predictive Control (GPC). Table 1 summarises the underlying control laws of some of the better known industrial adaptive

controllers; the majority are based on the PI(D) strategy.

Controller	Manufacturer	Law	GS	AT	CT	FF
Novatune	ASEA	GMV	■	■	■	■
Connoisseur	Predictive Control	LQG	■	■	■	■
STR	AccuRay Corp.	GMV			■	■
DMC	DMC Inc.	LPRC		●		■
IDCOM	Set Point Inc.	LPRC		■	■	■
Electromax V	Leeds & Northrup	PID		■	■	
Exact	Foxboro	PID		■	■	■
TCS 6355	Turnbull Control	PID		■	■	
2071 Microtuner	Goulton West	PID		■	■	
UDC 500	Honeywell	PID		●		
Micron P-200	Process Systems	PID		■		
CRL 452	Control & Readout	PID		■		
Eurotherm 810	Eurotherm	PID		■		
Microscan 1300	Taylor	PID	●			
SLC 3700	Bristol Babcock	PID	■			
PMS-100	Ferranti	PID	■	■		
Maxline	IRCON	PID		■		
5701	Fenwal	PID		■		
EST & ESKN	Omron Electronics	PID		■		
VeriTrim	Westinghouse	PID	■			●
SATT ECA40	Satt Controls	PID	■	■		
INTELLICON	Hungarian Sci. Acad.	PID			■	■
Firstloop	First Control	PP		■	■	■

GS = Gain Scheduling  
 AT = Auto-Tuning  
 CT = Continuous Tuning  
 FF = Feed-Forward

Table 1 - Characteristics of Some Adaptive Controllers

This report explains the implementation of an STPI module, within ACCOL, which provides an automatic facility for tuning proportional-plus-integral (PI) controllers, and has been designed for use with the standard PID3TERM module. The STPI module may be used in either a 'one-shot' or continuous tuning mode. In the 'one-shot' mode, when tuning is enabled, the module will perturbate the plant for a period of time, after which PI controller settings are determined. The module then returns control to the PID3TERM and effectively becomes transparent until

it is once more enabled. In the continuous tuning mode, the performance of the PID3TERM module is monitored, and the controller settings are adjusted accordingly. It should be noted that the STPI module will set the derivative gain of the PID3TERM to zero. The reason for developing a self-tuning PI module, as opposed to self-tuning PID, is that the final module is simpler to implement and use, and is more robust within industrial applications. In addition, because most processes exhibit non-oscillatory, stable, open-loop behaviour, the active damping provided by derivative action is not usually necessary for good control. This, along with the inherent disadvantage of noise amplification mean that derivative action is rarely employed in process control applications.

The STPI module incorporates three different algorithms for tuning PI controllers. These three algorithms have proved most popular with other controller manufacturers. This means that the ACCOL STPI module should be able to match the performance of most of its leading competitors. In addition to this reason, the algorithms have individual characteristics and in a particular application one may prove more suitable than the others.

(a) Closed Loop Cycling Algorithm (Alg. #0)

This 'one-shot' tuning algorithm forces the process variable to oscillate around its set point value, as shown in Fig. 2. The process variable is forced to oscillate through the use of a relay controller, as illustrated in Fig. 3. An integrator is also included in order to ensure that the process variable oscillates

around the set point value. The integrator gives rise to the characteristic triangular waveform produced by the controller output during the tuning phase. The period of the oscillations is determined by the dynamics of the process, but the user has the power to constrain the amplitude of the oscillations by specifying limits on the controller output and process variable. Thus the technique is inherently safer than the traditionally used Ziegler-Nichols ultimate method. The tuning phase is automatically terminated when a number of 'good' oscillations have been recorded. Upon termination, the period and amplitude of the oscillations are measured, and used to design the PI controller settings. When operated with default parameters, this technique only requires the user to specify the set point, and is therefore suitable for use from a 'cold start'.

(b) Pattern Recognition Algorithm (Alg. #1)

This algorithm provides continuous tuning of the controller gains. The key idea here is that processes respond to disturbances (or set point changes) with distinctive patterns whilst under PI control. By characterising these patterns, it is possible to formulate some rules for re-tuning the controller gains. Note that this algorithm is similar in many ways to how a skilled instrument engineer might re-tune a loop. Re-tuning takes place following the effect of disturbance, as shown in Fig. 4. During the disturbance, the performance of the controller is monitored, as shown in Fig. 5. Once the process variable has reached its peak deviation ( $E_{max}$ ) from the set point, the response time of the loop,  $T_L$ , is measured and subsequently used in the evaluation of the two

integrals: S1 and S2. Having obtained these values, the controller gains may be updated, as described in Fig. 5. Note that the pattern recognition algorithm requires initial values for proportional gain and integral gain, and is therefore not suitable for use from a 'cold start'.

(c) Model Based Algorithm (Alg. #2)

The model based algorithm is primarily intended as a 'one-shot' tuner, although it may also be configured the technique to operate in a continuous tuning mode. The important difference between this algorithm and the previous two is that the task rate of the control system must be carefully matched to the response time of the process. For example, the flow of a fluid through a pipe may respond within seconds to a change in valve position, whereas the pH within a large reaction vessel may take several minutes to respond to a change in acid dose. These two application examples would require the use of two different task rates. A good rule for use with the model based method, is to select a task rate that is approximately 1/10th of the process rise time, which may be determined from a step test, as shown in Fig. 6.

During the tuning phase, a pseudo random binary sequence (PRBS) is produced at the controller output, as shown in Fig. 7. The user must specify the mean level and the amplitude of the PRBS: the mean level should be chosen in order to cause the process variable to deviate at, or near, its set point value, and the amplitude should be sufficiently large to cause significant deviations, yet keep the process variable within acceptable

limits. While the PRBS is applied, the process output and the controller output data are fed into a recursive estimation algorithm, as illustrated in Fig. 1, which fits a mathematical model to the data. At the end of the tuning phase, the model is then used to design PI controller settings. The model based algorithm may be used from a 'cold start', although it requires more values to be specified by the user than the closed loop cycling algorithm.

In the STPI module, the user has four methods available for tuning. These are designated:

- Method #0 - Closed Loop Cycling followed by Pattern Recognition
- Method #1 - Closed Loop Cycling
- Method #2 - Pattern Recognition
- Method #3 - Model Based

Method #0 is the default method, as it is the most robust, requires the minimum amount of setting up, and will effectively provide continuous tuning from a 'cold start'. Method #1 is provided for applications where periodic re-tuning is more desirable than continuous tuning. Method #2 is provided to allow pattern recognition to be switched on and off following initial tuning. Method #3 may be used as an alternative to method #1, or else configured by the expert user to provide an alternative continuous tuning procedure.

## 2. CONFIGURING A SELF-TUNING PI CONTROLLER WITHIN ACCOL

Fig. 8 shows the basic structure of a self-tuning PI controller implemented within ACCOL. The detailed connections required to configure the self-tuning PI controller are presented in Fig. 9. The four major outputs of the STPI module are:

PROP2 - the designed value of proportional gain  
INT2 - the designed value of integral gain  
STATUS - a status word containing various flags  
DONE - a confidence factor relating to the tuning

STATUS is a seven bit word which contains information related to the tuning phase. Each bit represents a particular status condition, and their meanings are defined as follows:

STATUS bit 0 - tuning in progress  
(refers to all three algorithms)

STATUS bit 1 - pattern monitoring in progress  
(refers only to pattern recognition algorithm)

STATUS bit 2 - relay amplitude reduced during tuning  
(refers only to closed loop cycling algorithm)

STATUS bit 3 - relay amplitude very small  
(refers only to closed loop cycling algorithm)

STATUS bit 4 - termination of tuning after 21 cycles, due to limit cycle not converging  
(refers only to closed loop cycling algorithm)

- STATUS bit 5 - input or output limits incurred during tuning  
(refers to all three algorithms)
- STATUS bit 6 - model gain negative, possibly due to incorrect  
setting of the REVERSE flag  
(refers to the model based algorithm only)

The status word may be logically ANDed with the appropriate masks in order to determine the condition of individual bits. For example if the STATUS word has the value 37, then this corresponds to bits 0, 2 and 5 being set (i.e.  $1 + 4 + 32 = 37$ ), which means that the relay amplitude was reduced during tuning due to signal limits being incurred. Note that bits 0 and 1 are continually updated whereas the others are only set during tuning, and remain fixed until tuning is re-initialised.

DONE takes a value between 0 and 100% and provides an indication of the success of the tuning phase. Note that the three algorithms will produce different values for DONE because of the different ways that it is calculated. DONE should therefore not be used to compare the performance of the algorithms (the quality of control is a much better comparison). In general however, values of DONE which are less than 50% imply low confidence in the designed controller gains.

It is recommended that the results of the tuning are checked using a CALCULATOR block before feeding them into the PID3TERM module, as shown in Fig. 8. For example, the following calculator block could be used to limit the range of the proportional gain:

```

PROP1=PROP2
:IF(PROP2<1)
  PROP1=1
:ENDIF
:IF(PROP2>10)
  PROP1=10
:ENDIF

```

Alternatively, the following calculator would only update the gains if the confidence factor exceeded a specified value:

```

:IF(DONE>50)
  PROP1=PROP2
  INT1=INT2
:ENDIF

```

### 3. OPERATION OF STPI MODULE

This section presents a 'check-list' for operating the self-tuning PI module at its simplest level.

(1) Choose the self-tuning method using SELECT

SELECT = 0 => Closed Loop Cycling + Pattern Recognition

SELECT = 1 => Closed Loop Cycling

SELECT = 2 => Pattern Recognition

SELECT = 3 => Model Based

(2) Define whether the process is direct or reverse acting using the flag REVERSE. Note that ON implies reverse acting, i.e. an increase in the controller output produces a decrease in

the process variable (default is OFF).

- (3) Specify the desired SETPOINT.
- (4) Set the required performance of the closed loop system in terms of its percentage overshoot to a step change. Note default value = 10%.
- (5) Set the safety limits (if required) on the controller output: OPMAX and OPMIN.
- (6) (Optional) Set variables in auxiliary signal list, if required. Note that if method 3 is being used from a 'cold start', OPMEAN must be set in auxiliary signal list 'B'.
- (7) Initialise the tuning procedure by turning ENABLE on. Note that tuning is initialised by the OFF-ON transition of ENABLE.
- (8) On completion of a 'one-shot' tuning procedure, control is returned to the PID3TERM module, and the STPI module becomes transparent.

#### 4. AUXILIARY SIGNAL LISTS

Whilst the default values for the three algorithms have been chosen to work well in most applications, the expert user may want to tailor the parameters of each algorithm to match the needs of a particular process. This facility is provided, within the module, by allowing access to additional information which is contained in a series of auxiliary signal lists:

Auxiliary List 'A' (for use with Methods 0, 1 and 2)

1. PVDEV
2. RELAY
3. INTEG
4. ACCEPT
5. HYSTER
6. PVAMP
7. PERIOD
8. THRESH

- (1) PVDEV (RW) - maximum peak deviation of process variable from set point during closed loop cycling. This may be used as an additional safety feature. Default value = 100%.
  
- (2) RELAY (RW) - amplitude of relay characteristic during closed loop cycling. Note a good initial choice can reduce the tuning time. Default value = 2%.

- (3) INTEG (RW) - a flag which is set to indicate that the process has a natural integrating action, which is sometimes the case in level control problems. Setting this flag means that the closed loop cycling algorithm will use a square wave (as opposed to triangular) perturbation sequence and will design a proportional controller. Default value = OFF.
- (4) ACCEPT (RW) - tolerance between successive peaks which constitutes 'acceptable' oscillation during closed loop cycling. When successive peaks, P1 and P2, satisfy the condition  $100\% \times |P1 - P2| < P1 \times \text{ACCEPT}$ , controller settings are designed. Default value = 50%.
- (5) HYSTER (RW) - a noise protection facility which adds hysteresis to the relay characteristic. HYSTER should be set to 1/2 of the observed peak-to-peak noise. Default value = 0.2%.
- (6) PVAMP (RO) - current peak amplitude of the process variable oscillations.
- (7) PERIOD(RO) - current period of the oscillations.
- (8) THRESH (RW) - a noise protection feature for the pattern recognition algorithm. THRESH is a threshold value which the measured error must exceed before the controller settings are re-assessed. Default value = 5%.

(Note: RW = Read/Write, RO = Read Only )

Auxiliary Signal List 'B' (for use with Method 3)

1. OPMEAN	8a. A1	9a. DIAG1
2. OPDEV	8b. B1	9b. DIAG2
3. TOTAL	8c. B2	9c. DIAG3
4. ALPHA	8d. B3	9d. DIAG4
5. LAMBDA	8e. B4	9e. DIAG5
6. PERR	8f. B5	9f. DIAG6
7. ADAPTIVE	8g. C1	9g. DIAG7

- (1) OPMEAN (RW) - the mean value of the perturbation sequence during tuning. When tuning is enabled OPMEAN is automatically set to the last value of the PID3TERM output. OPMEAN may be manually adjusted during the tuning phase in order to keep the process variable at, or near, the set point.
- (2) OPDEV (RW) - the amplitude of the perturbation sequence. The default value is 5%.
- (3) TOTAL (RW) - the total number of samples in the tuning period. Default value = 100.
- (4) ALPHA (RW) - a first order digital filter coefficient ( $0 < \text{ALPHA} < 1$ ). Default value = 0.5.
- (5) LAMBDA (RW) - estimator forgetting factor ( $0.9 < \text{LAMBDA} < 1.0$ ). Default value = 0.99.

- (6) PERR (RO) - the current value of the prediction error.
- (7) ADAPTIVE (RW) - flag to set continuous closed loop tuning.
- (8) A1, B1, B2, B3, B4, B5, C1 (RW) - Estimated coefficients of the discrete time process model.
- (9) DIAG1, DIAG2, DIAG3, DIAG4, DIAG5, DIAG6, DIAG7 (RW) - Covariance matrix diagonal elements.

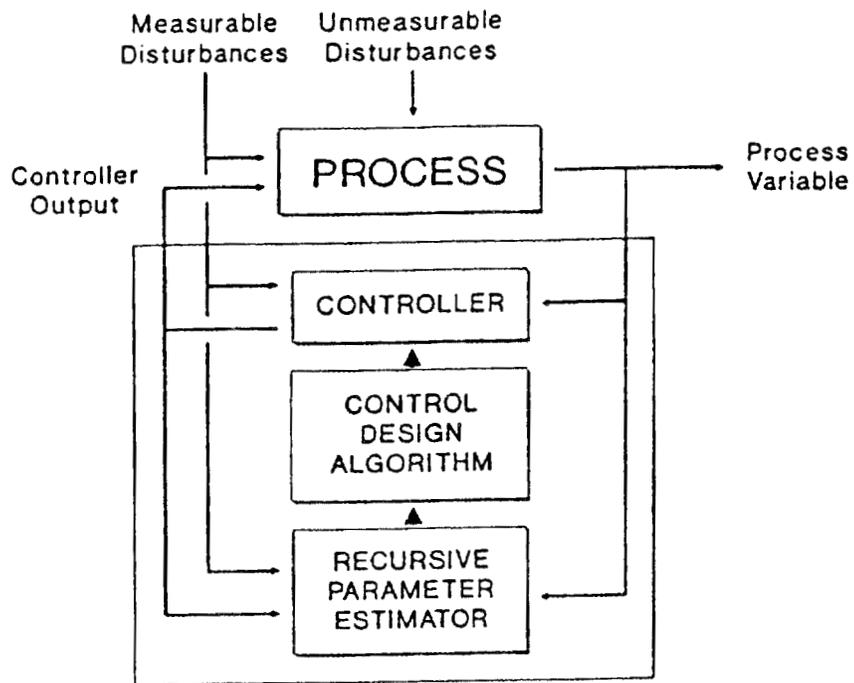


Figure 1 - General Self-Tuning Controller Structure

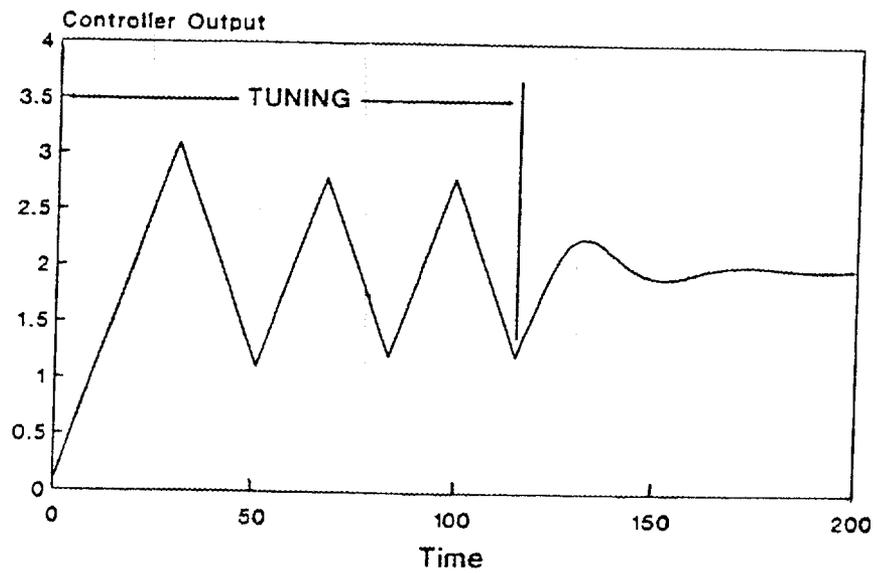
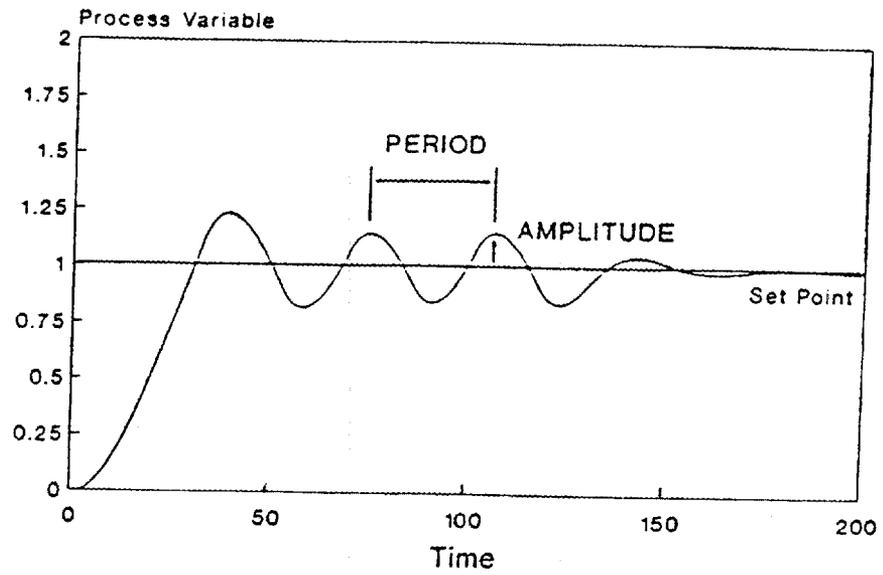


Figure 2 - Closed Loop Cycling Algorithm

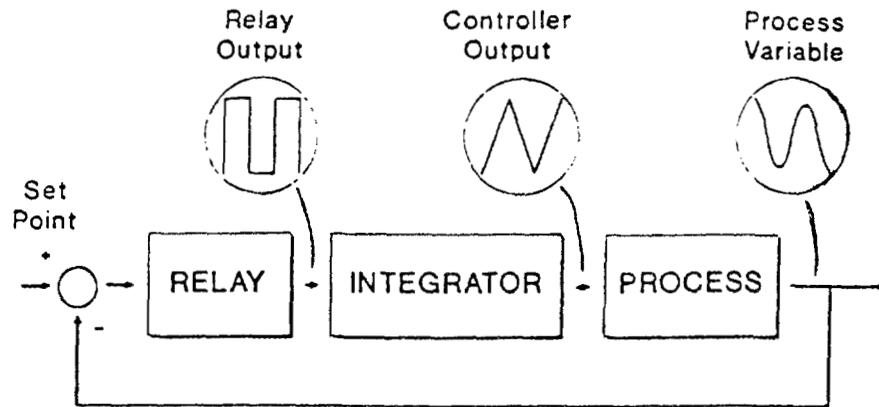


Figure 3 - Controller Structure for Closed Loop Cycling

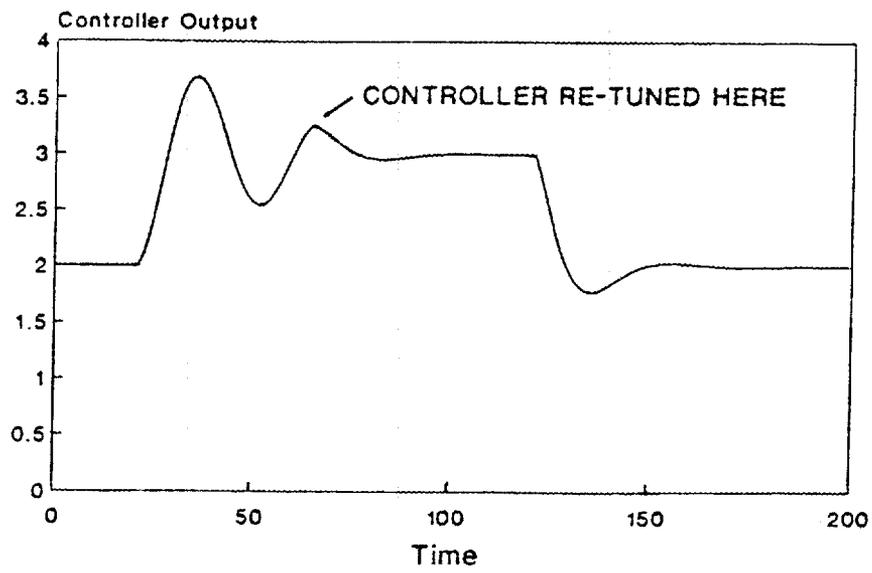
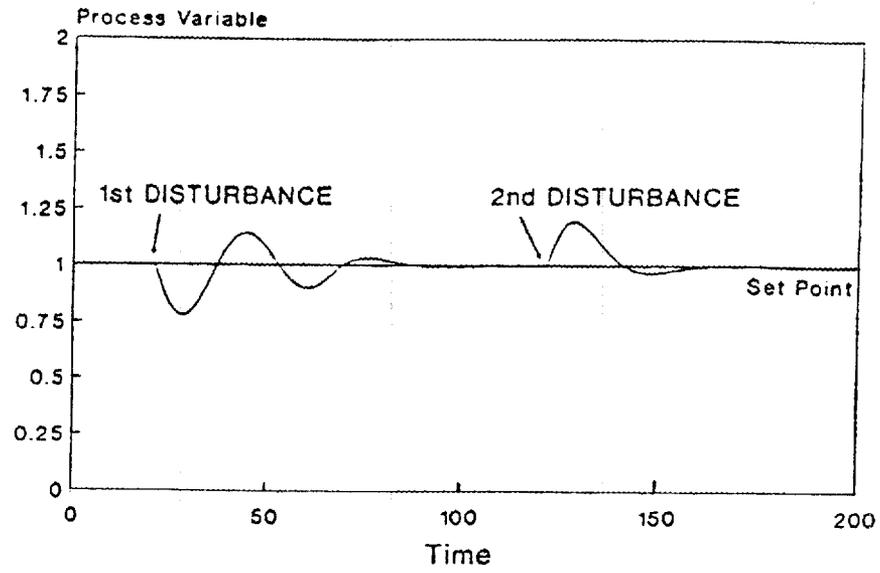


Figure 4 - Pattern Recognition Algorithm

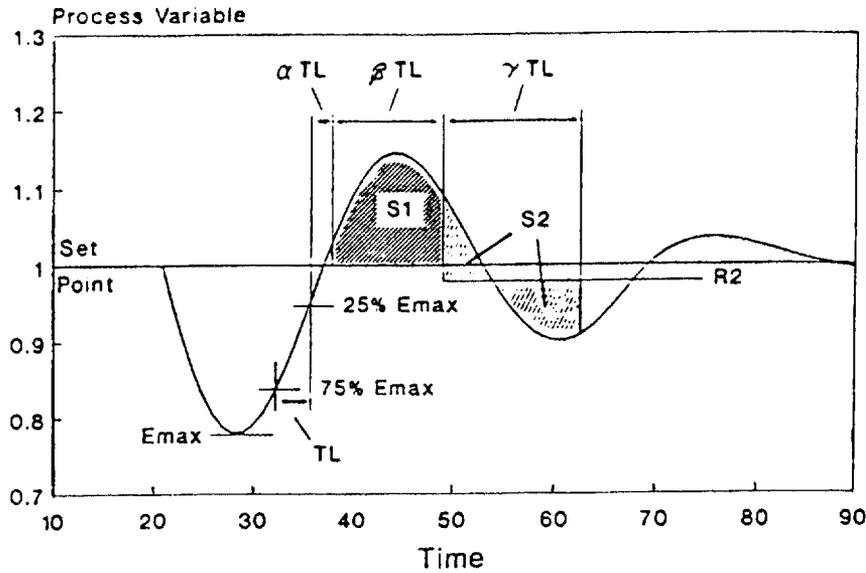


Figure 5 - Pattern Recognition Calculations

Re-tuning Algorithm:

$$\text{PROP2} = \text{PROP1} + (1-\text{DONE}) \cdot (K1 \cdot (S1+R1) + K2 \cdot S2)$$

$$\text{INT2} = \text{INT1} + (1-\text{DONE}) \cdot (K3 \cdot (S1+R1) + K4 \cdot S2)$$

where

R1 is a pre-defined area,

R2 is a pre-defined level,

DONE is a confidence factor related to overshoot,

and K1, K2, K3, K4,  $\alpha$ ,  $\beta$ ,  $\gamma$  are constants.

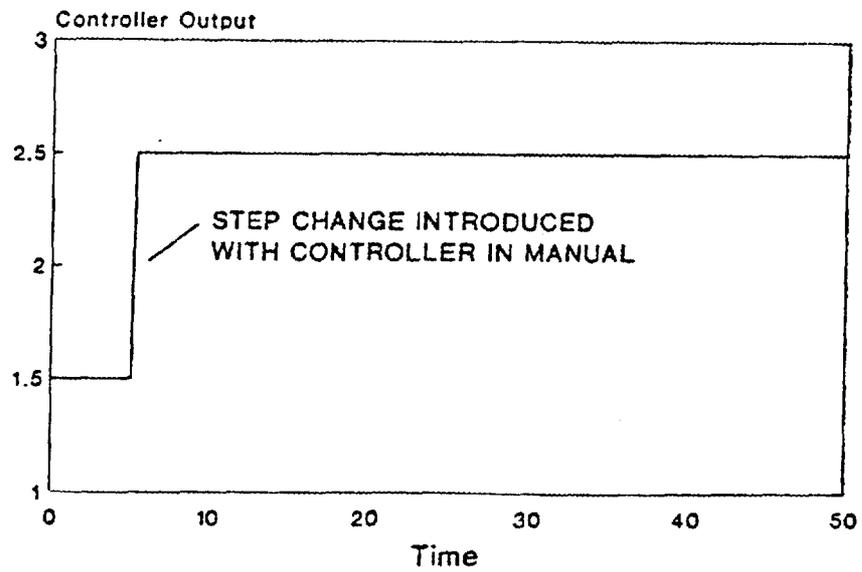
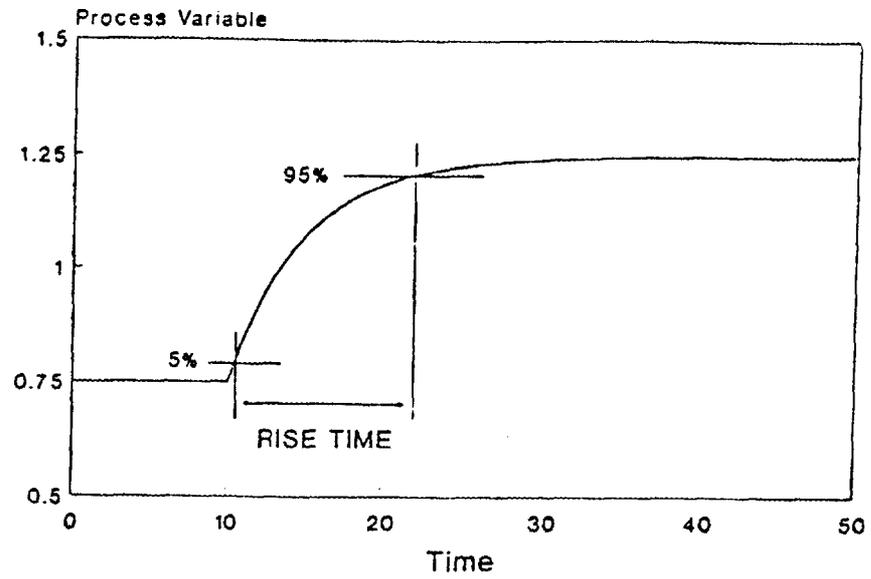


Figure 6 - Step Test to Determine Suitable Task Rate for Model Based Algorithm (Task Rate  $\approx$  Rise Time+10)

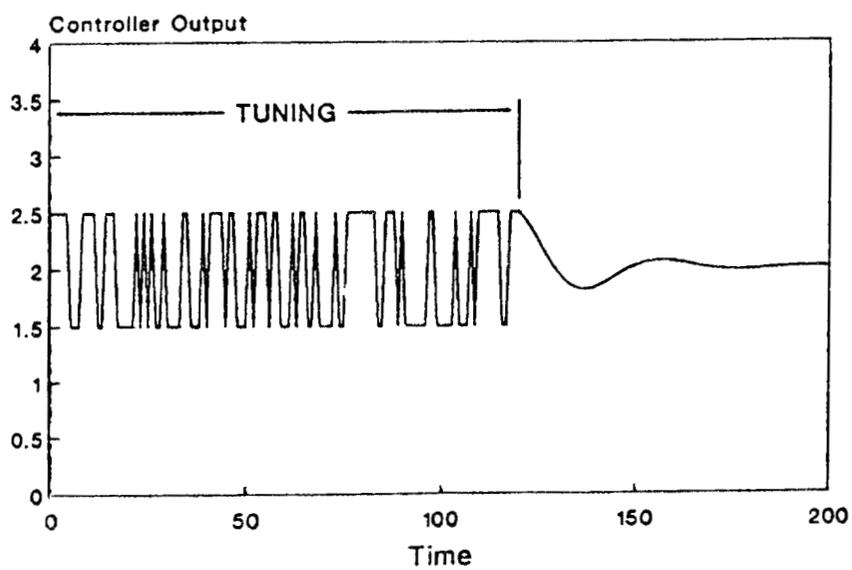
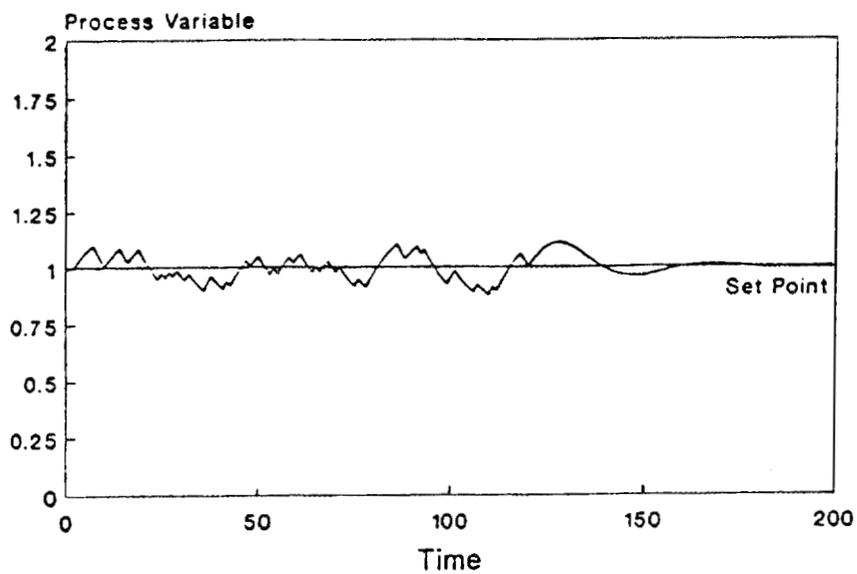


Figure 7 - Model Based Algorithm

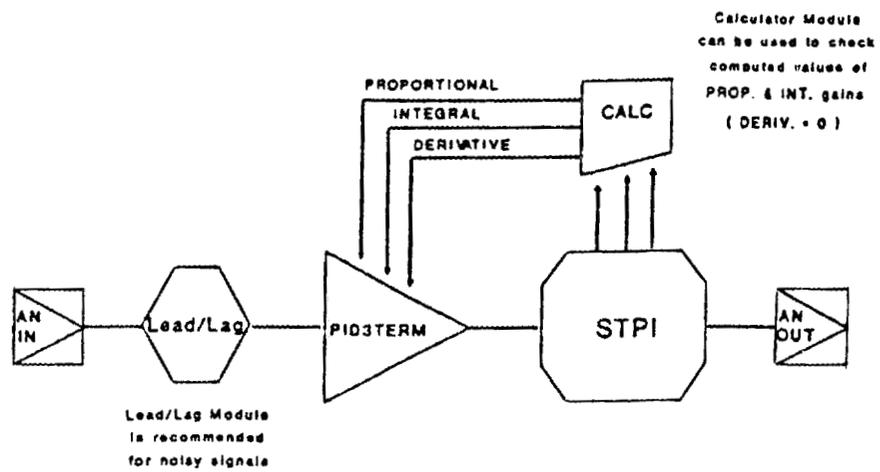


Figure 8 - Basic Structure of ACCOL Self-Tuning PI Controller

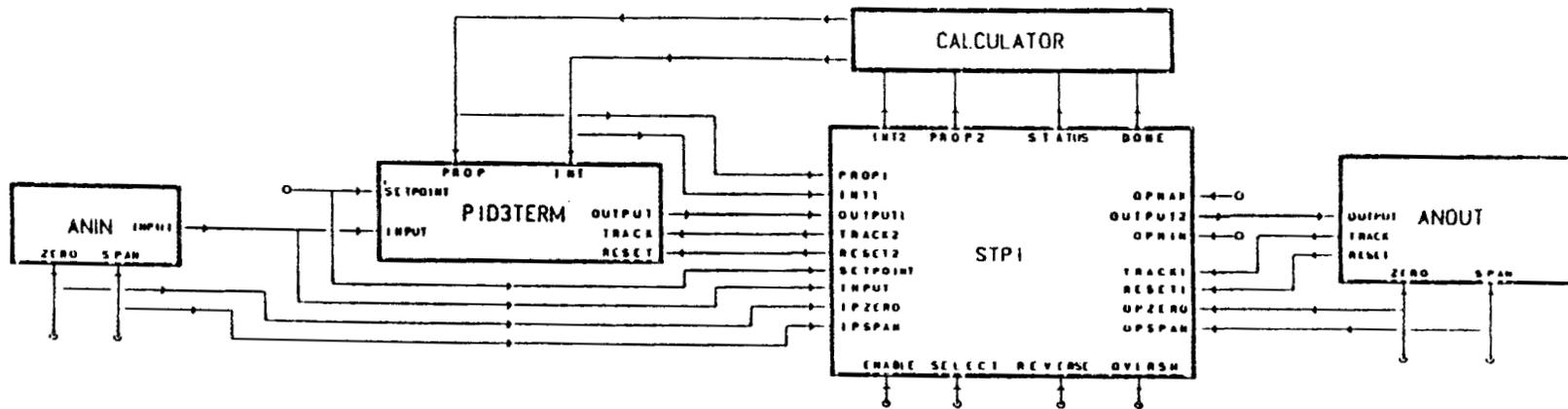


Figure 9 - Connection Details for STPI Module

PART II

SUMMARY OF FIELD TRIALS

## FIELD TRIALS SUMMARY

This section summarises the results of the field trials that were undertaken to evaluate the performance of the self-tuning algorithms on real industrial plant. Three sets of trials were carried out. The first of these was on a pH control loop at a water treatment plant belonging to the Sunderland and South Shields Water Company (S&SSWC). The second set of trials were carried out on various flow control loops at English China Clay (ECC). The third field trial involved a pressure loop at a British Gas Pressure Control Station. In all three trials, the algorithms performed well using their default settings despite the widely differing properties of the processes under test.

#### 1. Field Trial No.1 - pH Control Loop

Fig. 1 presents a schematic of the loop, which is used to adjust the pH of the treated water. The controller output is used to vary the speed of a lime pump which raises the pH to the correct value. Fig. 2 illustrates the operation of the closed loop cycling algorithm in action. It is apparent that the pH signal is rather noisy, and also that the process is subject to disturbances. Despite these adverse conditions, tuning proceeded in the normal way and produced good PI controller settings. Fig. 3 shows a longer time history of the tuning and it can be seen the resulting control is as good as, if not better than the existing enhanced PI controller. Also presented in Fig. 3 is the time history of the model based algorithm, which was obtained on the

following day. Once again good control is achieved. It should be noted that although the both tuning methods perturb the process, the amplitude of the perturbations are not much greater than those caused by the normal disturbances.

## 2. Field Trial No.2 - Flow Control

A range of tests were carried out during four days of testing at the English China Clay works, all of which involved flow control problems. Fig. 4 presents a general schematic of the type of loop under control. Fig. 5 shows the typical results obtained with closed loop cycling tuning and Fig. 6 illustrates those obtained using model based tuning. Both methods worked well in all tests. Note that a shorter tuning period could probably have been used with the model based algorithm, and also the amplitude of the perturbations could have been reduced, if required. Fig. 7 presents an interesting result obtained when applying closed loop cycling to a loop which had a faulty control valve. The fact that the valve was faulty was not known beforehand, but was soon diagnosed when the asymmetric oscillations were observed on the flow signal. Subsequent inspection revealed that the valve was sticking in one direction, and required maintenance.

## 3. Field Trial No.3 - Pressure Control Loop

Fig. 8 presents a schematic of the loop under control. The control objective is to maintain a desired pressure at a point which could be several miles downstream of the control valve.

Fig. 9 presents the time history of the tuning exercise. Closed loop cycling was initiated with a lower safety limit of 30% on the valve command signal in order to prevent excessively high pressures developing. It can be seen that during the first two cycles the valve command hits this lower limit and hence the relay amplitude is reduced until acceptable oscillations are obtained. In this test the tuning phase was deliberately forced to continue (for safety reasons) while the operators went off to lunch, and on return the cycling was disabled. A set point change was subsequently requested and it is apparent that the control is stable, although rather sluggish. This is most likely to have resulted from the severely nonlinear valve characteristic. The pattern recognition algorithm was then activated and another set point change introduced. Following this change, the controller gains were redesigned and the two set point changes at the end of the test show that good control has been achieved (N.B. lower safety limit has been moved to 25%). This example illustrates nicely how the "piggy back" arrangement of the tuning algorithms is used firstly to get reasonable settings which are subsequently refined.

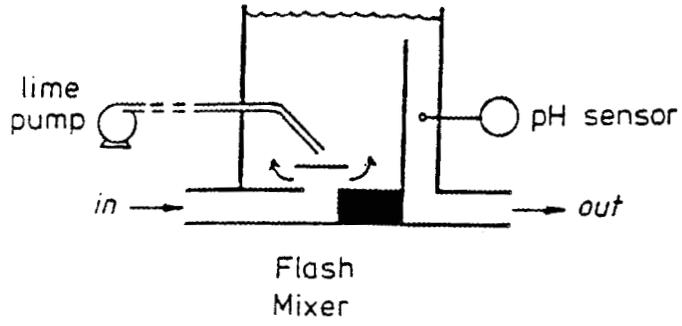


Figure 1 - pH Control Loop

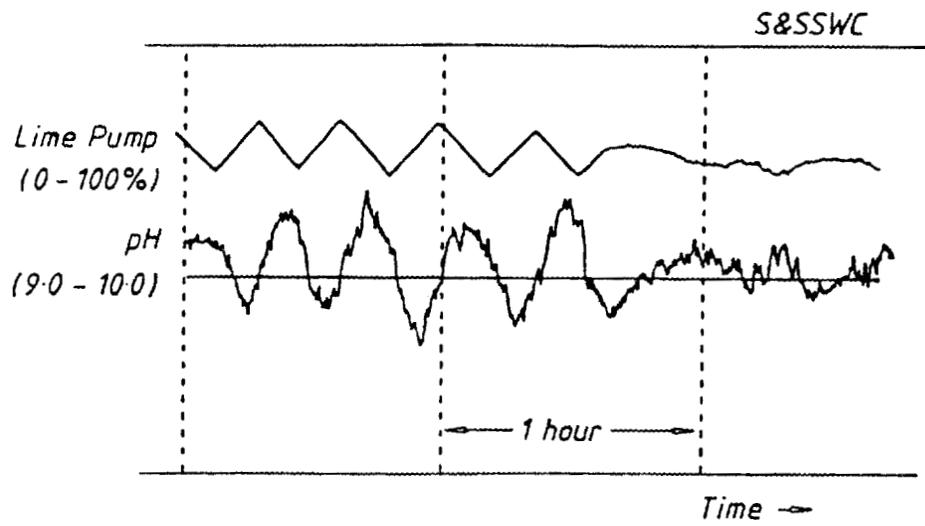


Figure 2 - Closed Loop Cycling Algorithm

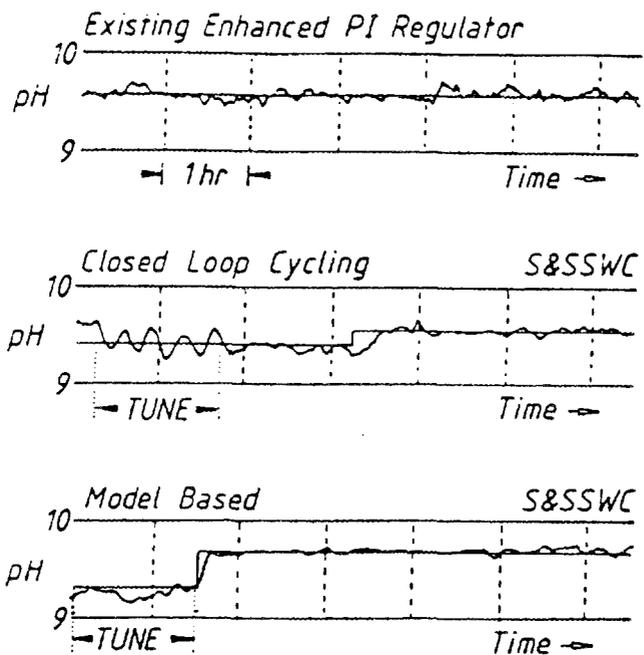


Figure 3 - Comparison of Tuning Algorithms

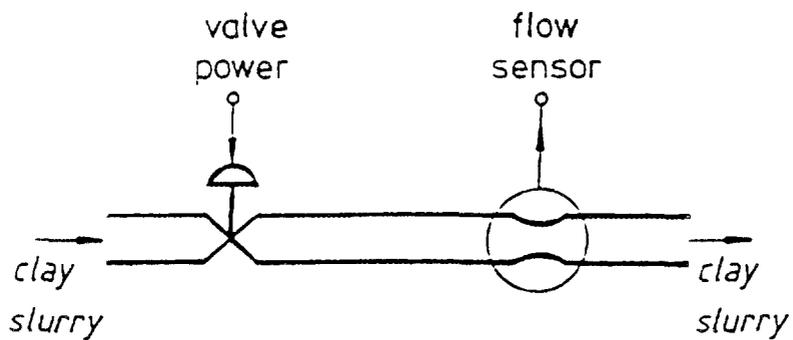


Figure 4 - Flow Control Loop at English China Clay

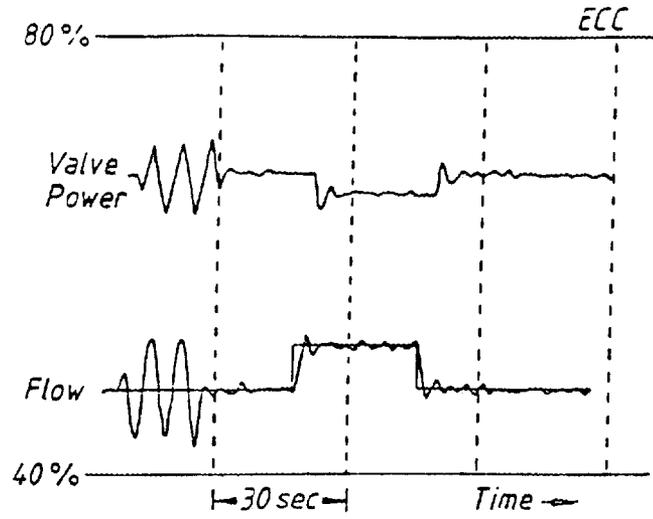


Figure 5 - Closed Loop Cycling Algorithm

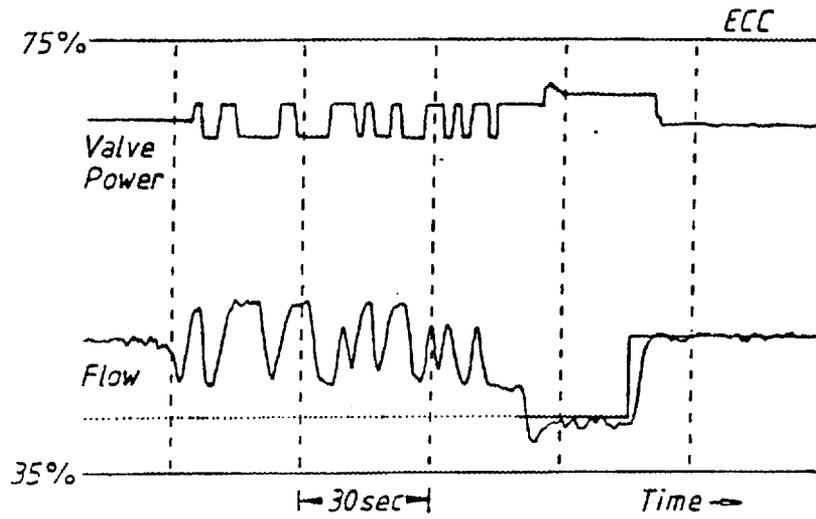


Figure 6 - Model Based Algorithm

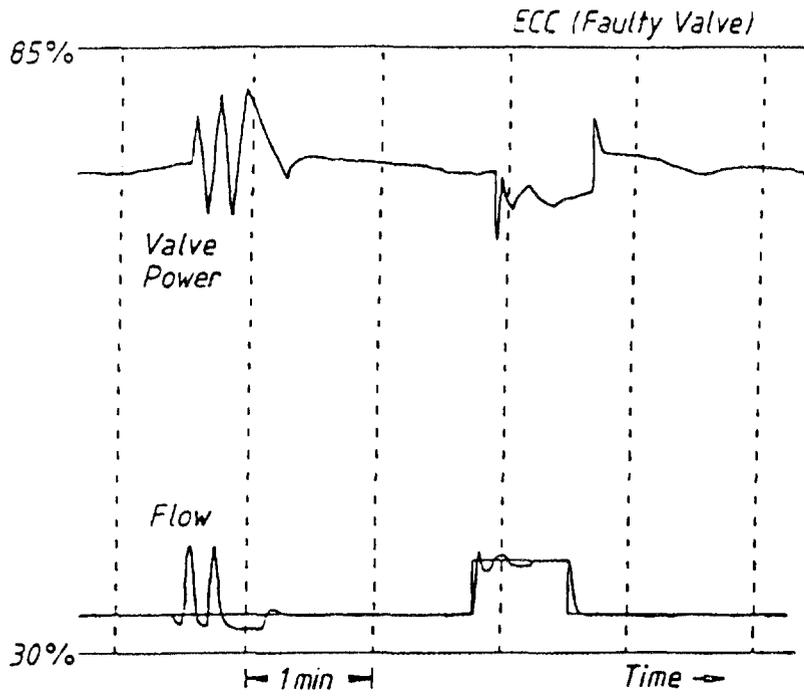


Figure 7 - Closed Loop Cycling on Loop with Faulty Valve

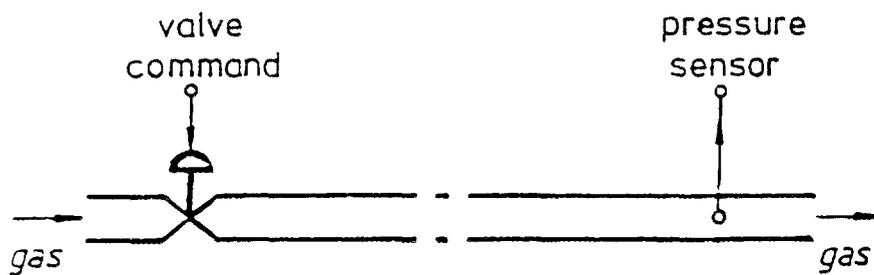


Figure 8 - British Gas Pressure Control Loop

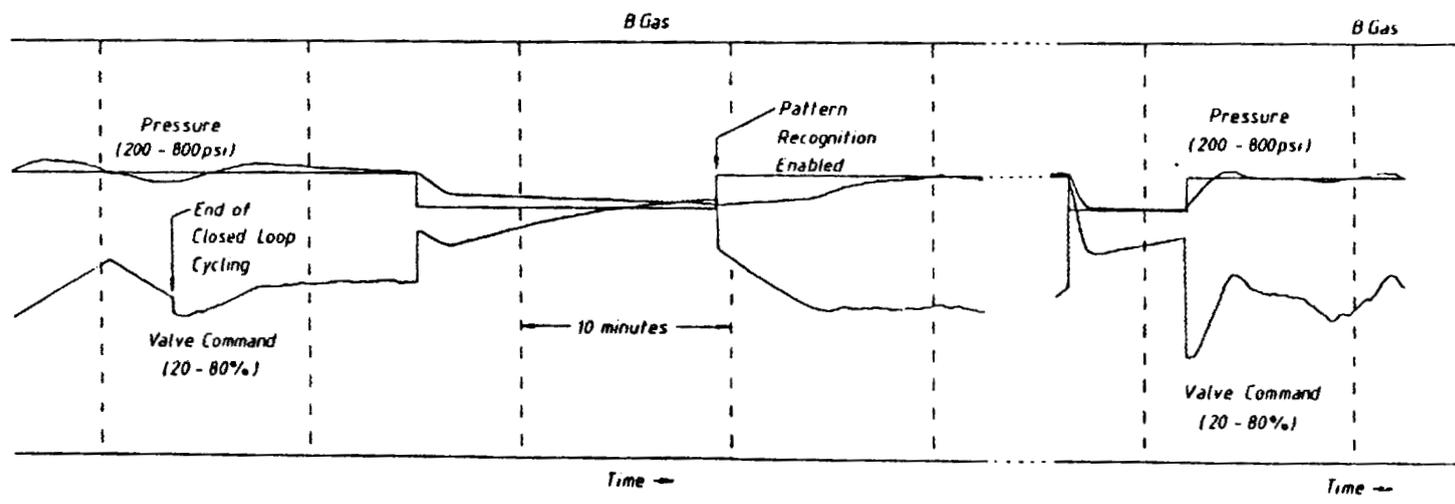
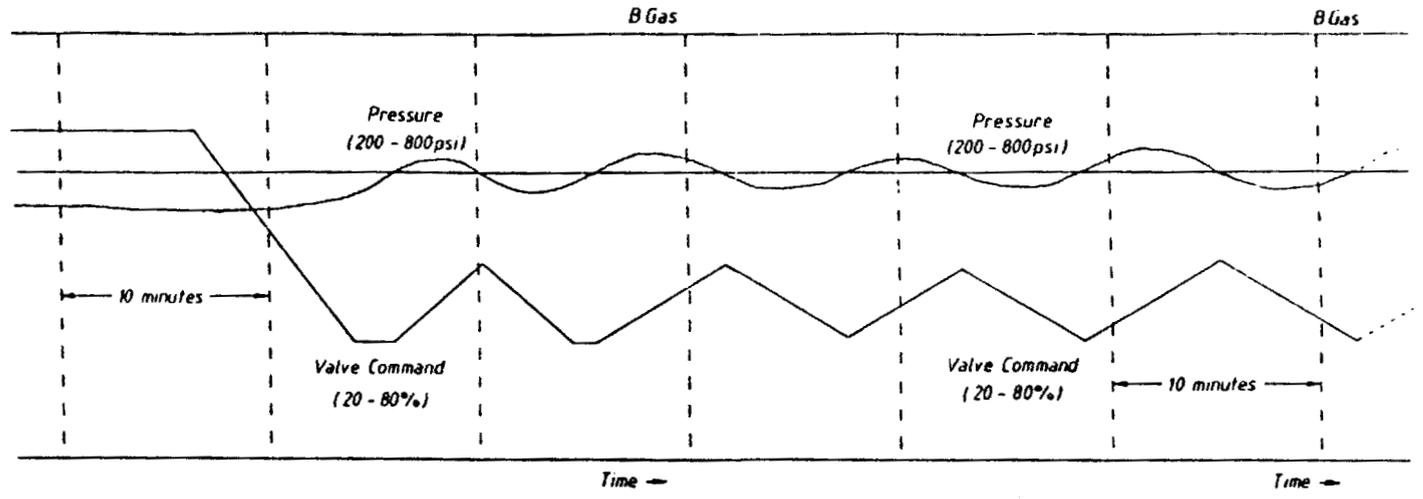


Figure 9 - British Gas Test Record



PART III

CLOSED LOOP CYCLING ALGORITHM THEORY

## CLOSED LOOP CYCLING AUTO-TUNER

## 1. Introduction

The majority of regulators used in industry are still of the PID form. Complex processes may have hundreds of regulators. Even after careful instruction instrument engineers and plant operators often still have difficulty in installing and operating such regulators. A feedback control system is of little value if it is improperly tuned. Several different methods have been proposed for tuning PID regulators. The need in tuning a controller is to determine the 'optimum' values of the controller gain  $K_c$  (or the proportional band  $PB$ ), the reset time  $T_i$  (or the reset rate in repeats per minute) and the derivative time  $T_d$ . The adjustment of these tuning parameters on feedback controllers is one of the least understood yet extremely important aspects of automatic control theory. Several methods for manually tuning these algorithms are used in practice, ranging from 'trial-and-error' to the more systematic use of empirical formulae such as those proposed by Ziegler and Nichols (1943). However for some complex processes, where the plant dynamics vary significantly in the course of their operation, automatic retuning is the only real answer in order to maintain a consistent final product. Astrom (1984) has proposed a simple robust estimation technique which provides the basis for a number of new methods for automatic tuning of PID regulators which easily can be incorporated into the new breed of microprocessor based controllers for single loop use.

This report develops the implementation (in ACCOL II) of a similar range of algorithms initially for use on the RDC 3350 unit. It should be emphasised that the approach will not work for problems where a more complicated regulator than the PID structure is required.

## 2. Theoretical Principles Supporting the Autotuner Design

In 1943 Ziegler-Nichols presented their seminal paper on controller tuning. In the interim period many other approaches have been suggested but rarely have they affected the popularity of this early simple strategy. The autotuner design to be developed here is based around an elementary approach to automate the Ziegler-Nichols rules as discussed by Astrom and Hagglund (1984).

Techniques for tuning controllers may be classified as either open-loop or closed-loop methods. The Ziegler-Nichols ultimate method is a closed-loop technique which has been applied successfully to both analogue and digital control situations. The basic method requires the determination of the ultimate gain,  $K_u$ . This is the value of gain (for a controller with only a proportional mode of operation) which causes the closed-loop controlled variable to cycle continuously with fixed amplitude. This 'marginally stable' situation implies that the Nyquist curve of the open-loop frequency response must pass through the critical  $-1+j0$  point on the Argand diagram (see Fig. 1). The period of the oscillation,  $P_u$ , is called the ultimate period. In the original

Ziegler-Nichols scheme,  $K_u$  and  $P_u$  were determined in the following way: tune out any reset or derivative action from the controller, leaving only the proportional mode. Maintain the controller on automatic, i.e. leave the loop closed. With the gain of the proportional mode set to some low arbitrary value impose an upset on the process (move the setpoint for a few seconds then return it to the original value) and observe the response. If the output response grows, reduce the controller gain; if the response damps out increase the controller gain. Continue in this way until sustained oscillations of constant amplitude are encountered. Finally, the controller parameters can then be obtained by using empirical formulae which rely on  $K_u$  and  $P_u$ . The ultimate method empirical results are presented as table 1.

Some of the shortcomings of the technique are listed below:-

- (i) the process transfer function must be at least third order.
- (ii) for a system with long time constants the technique is a very slow process.
- (iii) it is difficult to automate the experiment, and perform it in such a way that the amplitude of the oscillation is kept under control.
- (iv) the magnitude of the oscillation is dependent on the plant gain as well as the conditions supported by the plant when the test is initiated, i.e. the amplitude of oscillation is not known before the test.

As a consequence of the above features, another method which can provide automatic determination is proposed.

### 3. Astrom and Hagglund Relay Method

This method is based on the observation that a system with a phase lag of at least  $\pi$  radians at high frequencies must oscillate with period  $t_c$  under ideal relay control. One immediate advantage gained by including the relay is that the possibility of an unstable response is avoided. Secondly, the amplitude of the oscillation may be controlled simply by varying the limits of the relay.

The constant amplitude fixed frequency oscillation is called a limit cycle. Limit cycles arise in a wide variety of practical situations; consequently, considerable efforts have been expended to develop algorithms which can help the designer assess whether or not a system will exhibit such behaviour. Limit cycles can be stable or unstable; only stable oscillations exist in practice.

For systems of higher order than two, the basis for limit cycle studies is usually the frequency domain. Here, much of the published work assumes a separable system where the linear part is represented by its frequency response whilst the single non-linear element (in this case the ideal relay) is characterised by a quasilinear complex gain called a describing function. The describing function is evaluated on the assumption that the input to the non-linearity is a sinusoid of known amplitude.

The describing function is defined as

$$N(\lambda) = \frac{B + jC}{\lambda} \quad \dots 3.1$$

where B and C are the Fourier coefficients of the fundamental component present in the periodic non-linear output in response to the sinusoidal input  $\lambda \sin \theta$ . B and C are given by

$$B = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \sin \theta \, d\theta \quad \dots 3.2a$$

and

$$C = \frac{1}{\pi} \int_0^{2\pi} f(\theta) \cos \theta \, d\theta \quad \dots 3.2b$$

$f(\theta)$  is the true non-linear output in response to  $\lambda \sin \theta$ . For the ideal relay when

$$0 < \theta \leq \pi \quad f(\theta) = +V_m \quad \dots 3.3a$$

and  $\pi < \theta \leq 2\pi \quad f(\theta) = -V_m \quad \dots 3.3b$

assuming a symmetrical relay output.

From equations 3.1 to 3.3 it is easily shown that

$$N(\lambda) = \frac{4V_m}{\pi\lambda} \quad \dots 3.4$$

The resulting autotuner strategy using the ideal relay controller is presented as Fig. 2.

To illustrate how the circuit works consider the case when

$$G(j\omega) = \frac{\kappa\omega_0^2}{j\omega \{ (\omega_0^2 - \omega^2) + j2\zeta\omega\omega_0 \}} \quad \dots 3.5$$

Fig. 3 shows how the system responses for two different initial conditions; one a small (practically zero case) and the other a large value. In both cases we eventually converge onto the same limit cycles. To increase the amplitude of the limit cycle we simply increase  $V_m$ .

With reference to Figures 1 and 2, it is easily shown that Ziegler Nichols critical gain,  $Ku$ , is in fact the same numeric value as the describing function.

$$Ku = \frac{4V_m}{\pi A} \quad \dots 3.6$$

It follows, since  $V_m$  is fixed, to automatically determine  $Ku$  all that is required is to estimate  $A$ . In practice this is done using software programmed to implement a 'peak detection' strategy on the system error signal. The ultimate frequency is also calculated using the error signal and a 'zero-crossing' routine. Once these are evaluated, PI or PID settings can be determined using the look-up table 1. An alternative approach has been postulated by Astrom whereby systems with prescribed phase margin are obtained. The theory behind this approach is described in the next section.

#### 4. Control with Specified Phase Margin

Consider next the block diagram of Fig. 4(a) where  $G_{ol}(s)$  is the open-loop transfer function; the open-loop frequency response of this system is plotted as Fig. 4(b). The frequency  $\omega_g$  when the open-loop gain is 1, i.e.

$$| G_{OL}(j\omega_d) | = 1 \quad \dots 4.1$$

is called the gain cross-over frequency. The phase margin  $\phi_m$  is defined as

$$\phi_m = \angle G_{OL}(j\omega_d) + 180^\circ \quad \dots 4.2$$

where  $\angle G_{OL}(j\omega_d)$  is the angle corresponding to the magnitude condition of Equation 4.1.

Consider now Fig. 5(a) where the system  $G(s)$  is under PI control; the open-loop transfer function is given by

$$G_{OL}(s) = Kc \left\{ 1 + \frac{1}{sTi} \right\} G(s) \quad \dots 4.3$$

and the corresponding phase shift is

$$\angle G_{OL}(j\omega) = -90^\circ + \tan^{-1} \omega Ti + \angle G(j\omega) \quad \dots 4.4$$

If at frequency  $\omega_d$  rad/s, the argument of  $G(j\omega)$  is  $-90^\circ$ , then it is easily shown that

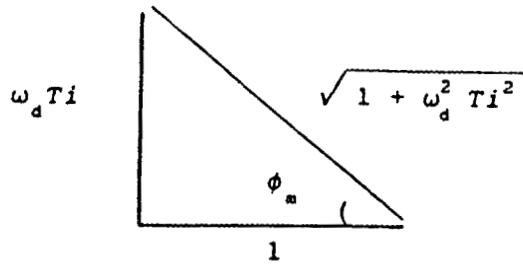
$$\tan^{-1} (\omega_d Ti) = \phi_m$$

or 
$$Ti = \frac{1}{\omega} \tan \phi_m \quad \dots 4.5$$

Further, from the definition of the phase margin, i.e. Equation 4.1

$$\frac{Kc \sqrt{1 + \omega_d^2 Ti^2}}{\omega_d Ti} | G(j\omega_d) | = 1 \quad \dots 4.6$$

Consider the triangle



then

$$Kc = \frac{\sin \phi_m}{|G(j\omega_d)|} \quad \dots 4.7$$

Equations 4.5 and 4.7 can be re-expressed in terms of the measured parameters  $Pu$  and  $\lambda$ . Firstly,

$$\omega_d = \frac{2\pi}{Pu} \quad \dots 4.8$$

and secondly from Fig. 5(b) under limit cycle conditions

$$\left[ \frac{4V_m}{\pi\lambda} \right] \frac{1}{j\omega_d} G(j\omega_d) = -1 \quad \dots 4.9$$

$$|G(j\omega_d)| = \frac{\pi\lambda\omega_d}{4V_m}$$

Hence from 4.6 to 4.9 the parameters  $Kc$  and  $Ti$  of the PI controller are given by

$$Ti = \frac{Pu \tan \phi_m}{2\pi} \quad \dots 4.10a$$

and

$$Kc = \frac{4V_m Pu \sin \phi_m}{2\pi^2 \lambda} \quad \dots 4.10b$$

The eventual scheme is presented in block diagram form as Fig. 6. The autotuner operates as a relay controller in the tuning mode (position 1) and then as an ordinary PI regulator in the control mode (position 2). Once  $P_u$  and  $A$  are determined as outlined in Section 2.  $K_c$  and  $T_i$  can be computed for any desired  $\phi_m$ ,  $\phi_m$  is an input parameter. A final point. It will have been noticed that the relay controller of Fig. 5b contains an integrator not present in the original structure (Fig. 2). An extra benefit of this arrangement is that it forces the limit cycle on the output to be sustained about the setpoint value. In control engineering terms the system, in the tuning mode, has Type 1 servomechanism tracking performance. This helps ensure 'bumpless' transfer between tuning and control modes.

## 5. Autotuner Refinements

This section describes two refinements to the basic method described earlier; one is intended to make the algorithm more user friendly while the second is included to improve its noise rejection properties.

### The Phase Margin-Overshoot(O/S) Concept

The phase margin is a frequency response design parameter introduced to describe the relative stability situation, i.e. just how stable is a stable system? Closed-loop systems with large phase margins have well damped step responses. Many control

system design criteria assume that the system can in effect be adequately described by a second-order process. The behaviour of second-order systems step and sine wave inputs is well understood and profusely documented. The following results have been abstracted from the technical literature.

- (i) the maximum percentage O/S of an ideal second-order process to a step function input is given by

$$\text{Maximum percentage O/S} = 100 \exp \left[ \frac{-\zeta\pi}{\sqrt{1 - \zeta^2}} \right] \quad \dots 5.1$$

- (ii) the phase margin  $\phi_m$  of an ideal second-order process is given by

$$\phi_m = \tan^{-1} \left[ \frac{2\pi}{\sqrt{[4\zeta^4 + 1]^{1/2} - 2\zeta^2}} \right] \quad \dots 5.2$$

Note both equations 5.1 and 5.2 depend only on the damping ratio  $\zeta$ . It is appreciated that many process operators may not have heard of a phase margin. However, most should understand the concept of peak overshoot related to step input behaviour. It follows that by specifying the maximum percentage overshoot one can evaluate  $\zeta$ . Once  $\zeta$  is known using Eqn. 5.2 one can determine the phase margin. Fig. 7 displays plots of both Eqns. 5.1 and 5.2. From Fig. 7(b) it can be seen that over a wide range

$$\phi_m \approx 100 \zeta \quad \dots 5.3$$

## Improvement of Noise Rejection Performance

A problem with the simple ideal relay strategy is that any noise superimposed on the useful signal can result in 'false' relay switching which in turn invalidates the tuning procedure. The noise rejection properties can be improved by simply adding some hysteresis into the relay characteristic as shown in Fig. 8. If too much hysteresis is added a degradation of the ultimate process will result. The general effect is an increase in the amplitude of the signal appearing at the input to the relay with a consequent lowering of  $K_u$  as compared with the ideal case. Section 7 presents a number of illustrative examples to clarify what may occur. Another feature is that  $P_u$  also tends to increase further consolidating a slower more heavily damped response than may have been anticipated. The ACCOL implemented algorithm has a default hysteresis band of  $\pm 2$  quantum levels; however, the operator can set the hysteresis to any desired level.

Tests to date have indicated that for signal to noise ratios greater than 5:1 additional 'analogue' filtering may be required. In practice this is supplied via an optimal 'digital' filter which is also addressable by the operator. The filter chosen is a discrete equivalent of the simple analogue filter

$$\frac{V_o}{V_i} = \frac{1}{1 + s/\omega_o} \quad \dots 5.4$$

The algorithm requests a value for  $\omega_o$  expressed in Hz, i.e.  $f_o$

where

$$\omega_0 = 2\pi f_0$$

The choice of hysteresis width plus filter bandwidth (if required) is left to the user, however, the following simple rules should help the selection process. With the filter inactive vary the relay characteristics (always ensuring initially that  $D > E$ ) until an 'oscillatory trend' is obtained. Measure the frequency of the oscillation then set  $f_0$  equal to this value or exceptionally equal to twice this value.

Before leaving this section it should be emphasised that a large number of problems WILL NOT require the above refinements. We estimate perhaps less than 10% of the problems we have looked at over the years. Nevertheless, with commercial software we must try and consider all possible contingencies.

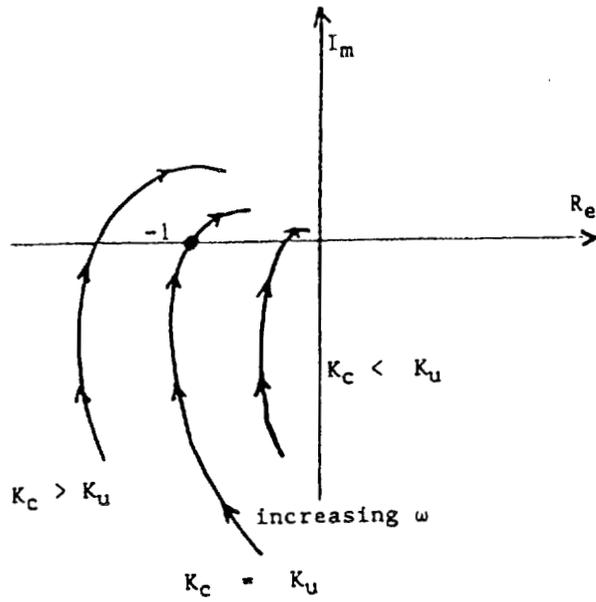
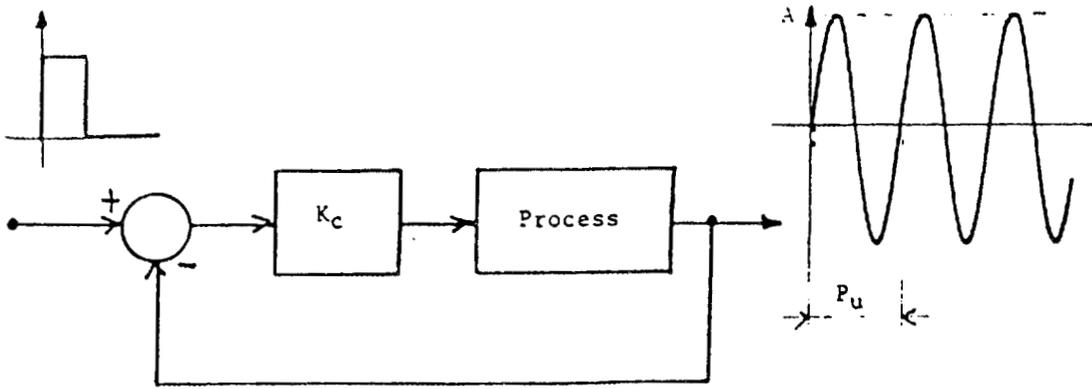


Fig. 1

Controller	$K_c$	$T_i$	$T_d$
P	$0.5 K_u$	0	0
PI	$0.45 K_u$	$P_u/1.2$	0
PID	$0.6 K_u$	$P_u/2$	$P_u/8$

Z-N Look-up Table

Table I

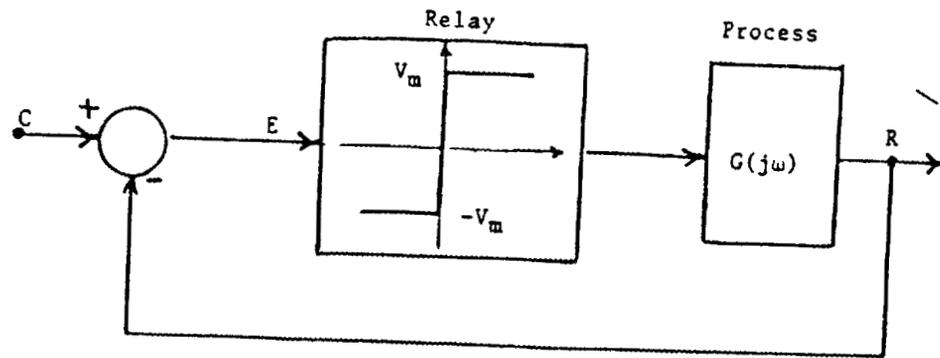


Fig. 2

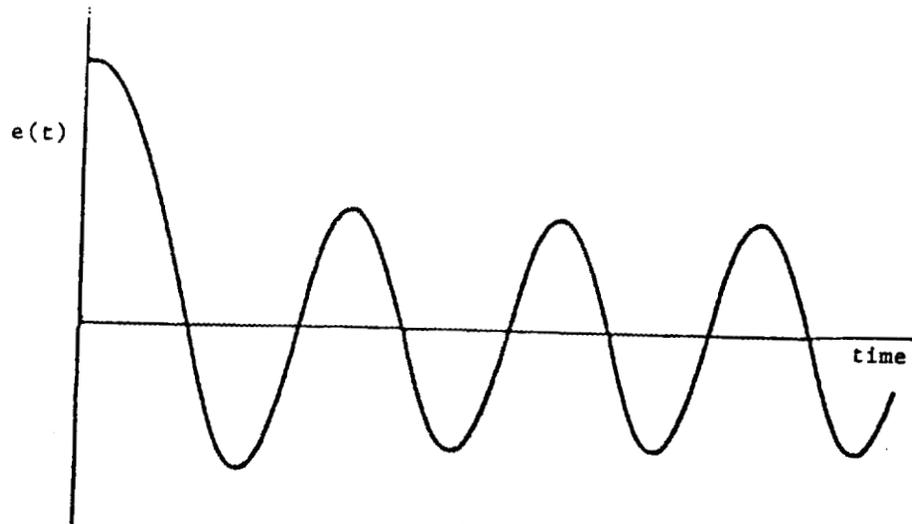


Fig. 3a

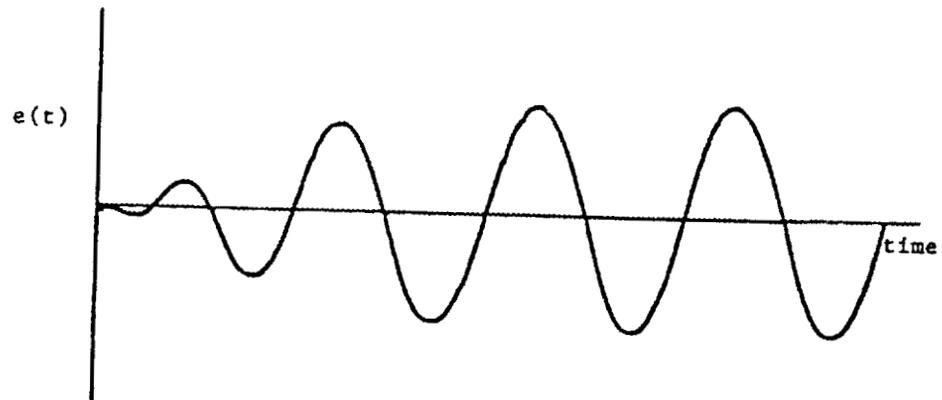


Fig. 3b

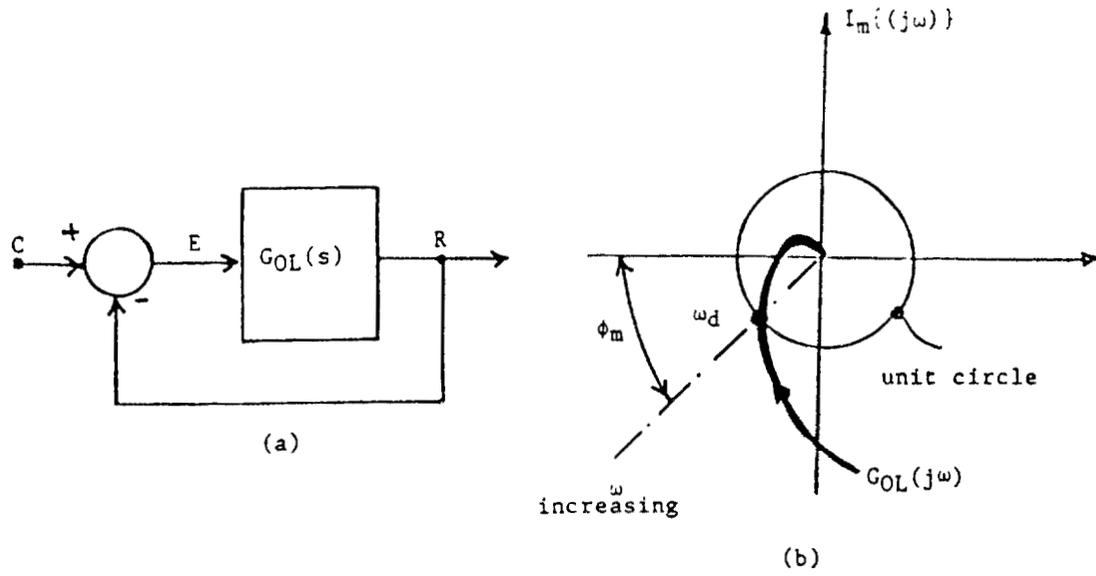


Fig. 4

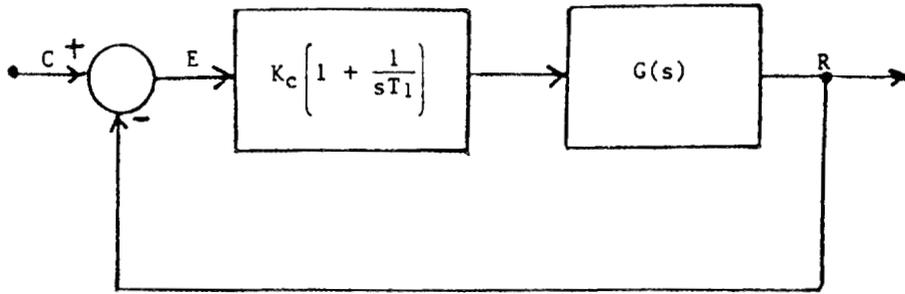


Fig. 5a

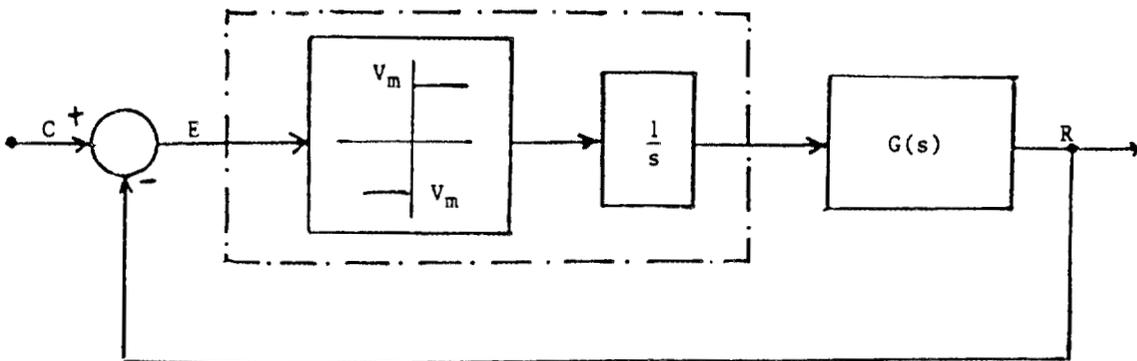


Fig. 5b

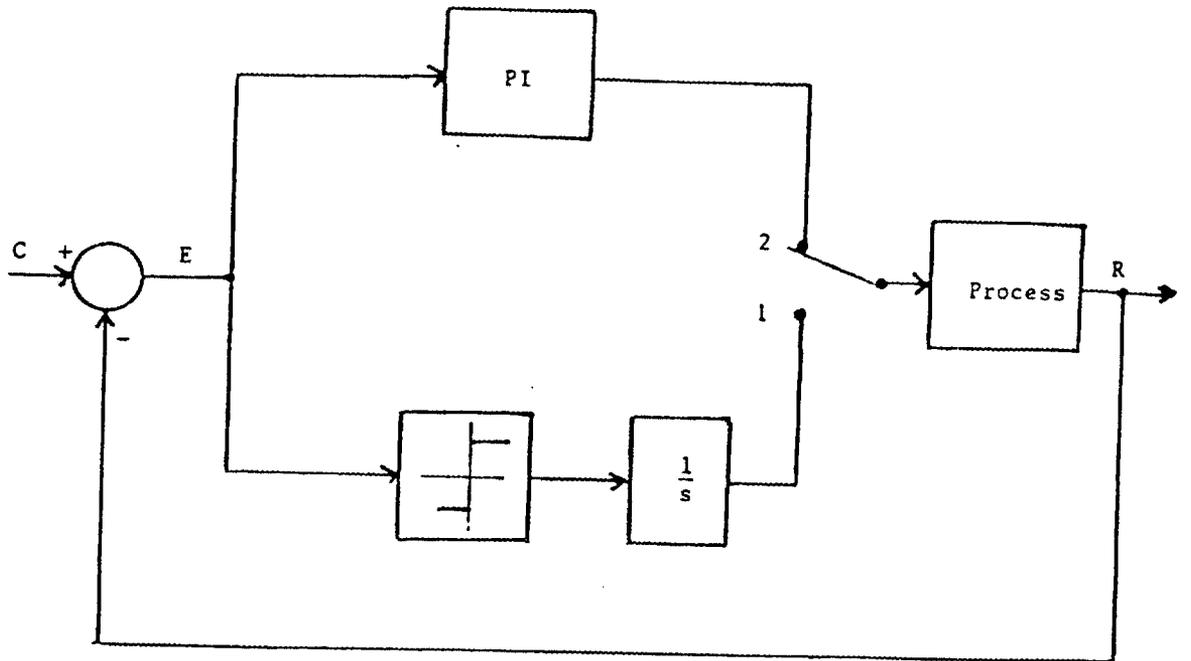


Fig. 6

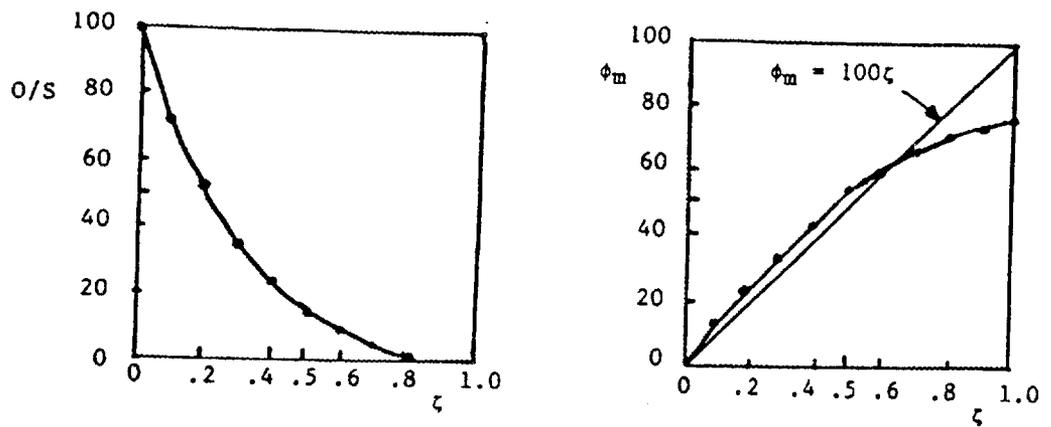
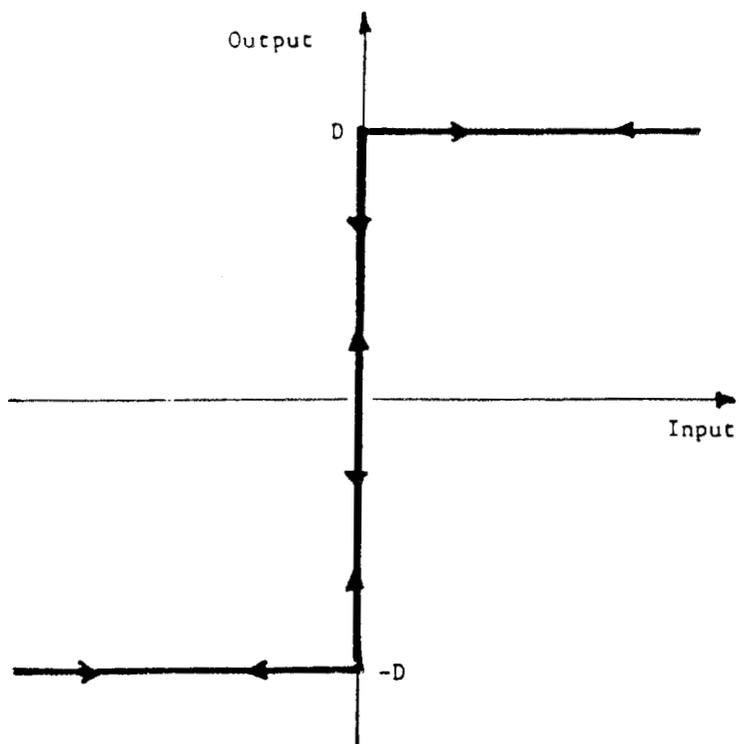
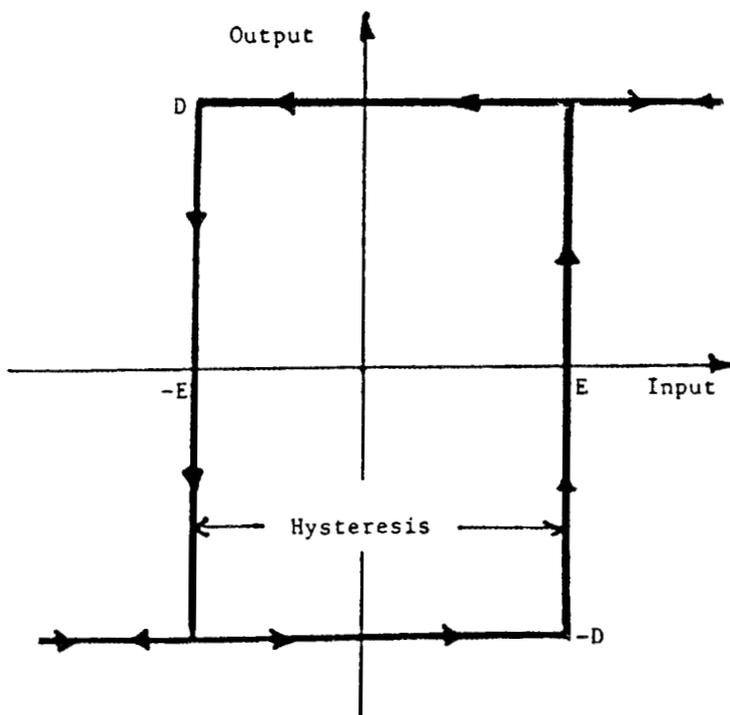


Fig. 7



(a) Simple Relay



(b) 2-Position Relay with Hysteresis

Fig. 8

PART IV

PATTERN RECOGNITION ALGORITHM THEORY

## PATTERN RECOGNITION CONTROL

1. Introduction

The need for self tuning controllers arises as instrument engineers and plant operators often have great difficulty in installing and operating control systems. The ability of the self tuner to model processes using some predefined testing sequence and establish suitable controller parameters to meet some pre-defined performance criteria can produce considerable savings in both time and expense during plant commissioning.

However, because processes are often time variant or nonlinear in operation then no guarantee exists that the system will perform to the required levels without the need for frequent retuning. This, in itself can lead to several problems. Firstly, when is re-tuning deemed necessary and secondly, will the application of the input disturbance sequence cause the process output to exceed plant limits and introduce further expense through down time.

One solution to this particular problem is to re-assess the present control scheme performance when the plant is subjected to some form of disturbance. The exact method for changing the controller parameters is normally based upon the experience and expertise of the control engineer. However, because these mechanisms, for PI regulators, are well understood several methods of automating this adaption procedure have been suggested [1-3].

Generally these types of technique can be termed 'Pattern Recognition Controllers'.

## 2. Pattern Recognition Philosophy

The basic procedure followed by the pattern recognition controllers is as follows:

- (i) monitor the error signal for any disturbances that occur over a specific amplitude, typically two times the process noise threshold. When recognized, record the maximum amplitude of the disturbance,
- (ii) identify the necessary information regarding the response of the present control scheme with respect to some predefined performance criteria,
- (iii) update the present control parameters, if necessary, using some empirical formulae.

The major advantage of this type of adaptive control scheme over others is that it does not require a model of the system in order to re-evaluate the controller parameters. Therefore any problems that may arise with systems whose model dimensions vary with time are avoided. Moreover, the implementation of the scheme in software is relatively straightforward, its lack of complexity leading to much faster sampling rates than might otherwise be

possible from other self tuning strategies.

Its one real disadvantage is its reliance upon some other technique to provide the controllers starting parameters. Although the coupling of the technique with one of the previously encoded self tuners, acting as an initialization stage, will provide a simple solution to this problem.

### 3. Pattern Recognition PI Adaptive Controller

The operation of the proposed adaptive control scheme occurs in four distinct stages, represented graphically in Figure 1, based on a setpoint disturbance.

- (i) Recognition of a new disturbance with a peak error ( $ER_{MAX}$ ) larger than a predefined noise threshold ( $NOISE$ ).
- (ii) Identification of the recovery time of the response ( $T_1$ ), the time taken by the present system to go from 75% of the peak error ( $T_0$ ) to 25% of the peak error ( $T_1$ ).
- (iii) Definition of the pattern features for adaption, Figure 1. Firstly, the area  $S1$ , representing the first peak of overshoot of the response with respect to the area  $R1$  and secondly, the area  $S2$ . This represents the decay rate of the response with respect to the level  $R2$ . Both  $R1$  and  $R2$  are evaluated from the defined performance specifications, where

for ideal operation  $S1 = R1$ ,  $S2 = 0$ .

Although these areas can readily be measured when the process response is oscillatory, this information is not recoverable when the response is overdamped. The answer, in each case, is to compute the areas when they lie within the time slots defined below:

$$S1 = \int_{T_1 + \alpha T_1}^{T_1 + (\alpha + \beta)T_1} (ERR) dt$$

$$S2 = \int_{T_1 + (\alpha + \beta)T_1}^{T_1 + (\alpha + \beta + \gamma)T_1} (ERR - R2) dt$$

where  $\alpha$ ,  $\beta$  and  $\gamma$  are all constants evaluated from a study of the response of a third order system.

- (iv) Updating of the controller parameters using the defined pattern features and the empirical relationships:

$$Kc = Kc + (1-DONE) \left[ K_1(S1-R1) + K_2S2 \right]$$

$$Ki = Ki + (1-DONE) \left[ K_3(S1-R1) + K_4S2 \right]$$

where the variables  $K_1$ ,  $K_2$ ,  $K_3$  and  $K_4$  are weighting constants and  $DONE$  is a confidence factor related to overshoot.

If required the updating procedure can be constrained using a re-tuning factor. Thus limiting the maximum percentage change that can occur at each adaption step.

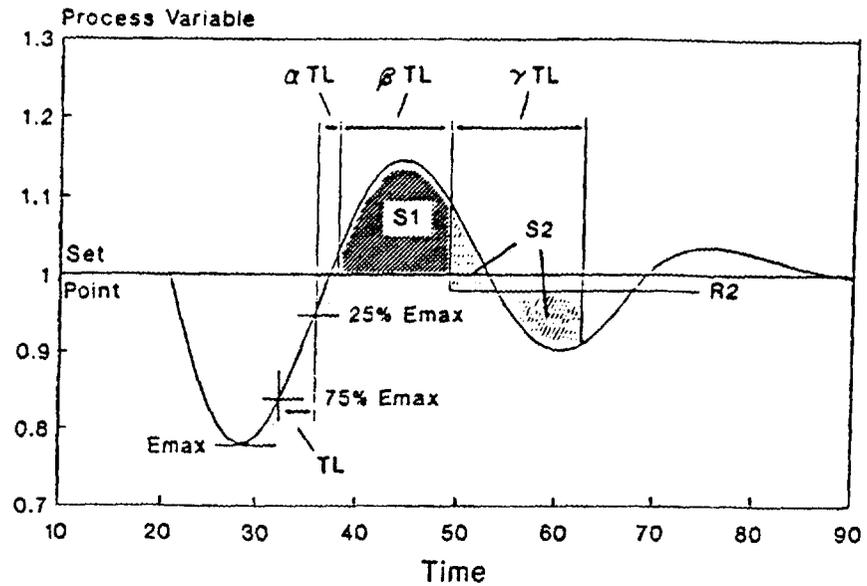


Figure 1 - Pattern Recognition Calculations

Re-tuning Algorithm:

$$\text{PROP2} = \text{PROP1} + (1-\text{DONE}) \cdot (K1 \cdot (S1+R1) + K2 \cdot S2)$$

$$\text{INT2} = \text{INT1} + (1-\text{DONE}) \cdot (K3 \cdot (S1+R1) + K4 \cdot S2)$$

where

R1 is a pre-defined area,

R2 is a pre-defined level,

DONE is a confidence factor related to overshoot,

and K1, K2, K3, K4,  $\alpha$ ,  $\beta$ ,  $\gamma$  are constants.



PART V

MODEL BASED ALGORITHM THEORY

## MODEL BASED CONTROL

1. Introduction

Proportional-Integral-Derivative (PID) controllers are employed extensively within the process industries. In many application however, only proportional and integral action are utilised because derivative action causes the controller to respond too energetically to any noise that is present on the measured process variable. In addition, many processes exhibit non-oscillatory open loop behaviour and therefore the active damping provided by derivative action is rarely required. Finally, the specifications for controller responses are often loose and PI controllers are capable of providing acceptable performance in a number of process applications.

Acceptable performance can only be obtained however if the PI controller is properly tuned, which means that the amounts of proportional and integral action provided by the controller are correctly set. Before these two values can be selected, information about the plant must be known, therefore a mathematical description of the process is required. Once this description, or 'model' has been obtained, values of proportional and integral gain can be evaluated such that some pre-specified design objective is achieved. When these two operations are automatically performed, the resulting scheme is popularly known as a self-tuning controller.

For the purposes of this report, a self tuning controller is defined as one which uses an on-line estimator/design procedure for an initial tuning period, after which the procedure is turned off and the controller effectively operates in a fixed gain mode.

## 2. Model Estimation

### 2.1 Model Structure

The structure of the model is developed from the commonly encountered process reaction curve that is a standard first order lag with time delay [1], whose step response is displayed in Fig. 2 and transfer function by Equ. 2.1:

$$Gp(s) = \frac{K \cdot e^{-s\theta}}{1 + s\tau} \quad (2.1)$$

where  $s$  is the Laplace operator,  
 $K$  is the process gain,  
 $\tau$  is the process time constant,  
and  $\theta$  is the process time delay.

However, the self-tuning PI controller estimates a discrete time model of the process dynamics. The main reasons for adopting the discrete time approach, as opposed to a continuous time scheme, are that it removes difficulties involved in digitising systems, and that it handles time delays naturally. Thus good control will be provided even when the sampling time (task rate) is 'coarse' with respect to the process time constant:

Using the structure described in Equ. 2.1, its digitised equivalent is of the form:

$$Gp(z^{-1}) = \frac{b_{d+1} z^{-(d+1)} + b_{d+2} z^{-(d+2)}}{1 + a_1 z^{-1}} \quad (2.2)$$

where  $z^{-1}$  is the backward shift operator (the value one sample previously) and  $d$  the integer number of sampling times in the process time delay. Therefore

the numerator can be extended to accommodate any value of time delay. A direct comparison between the continuous and discrete time systems is possible if the parameters within Equ. 2.2 are defined by:

$$\begin{aligned} a_1 &= -e^{-T/\tau} \\ b_{d+1} &= K \{ 1 - e^{-(1-m)T/\tau} \} \\ b_{d+2} &= K \{ e^{-(1-m)T/\tau} - e^{-T/\tau} \} \end{aligned} \quad (2.3)$$

where  $T$  is the sampling time,  
 $d$  is the integer part of  $\theta/T$   
and  $m$  is the fractional part of  $\theta/T$

### 2.1.1 Effects of Sampling Time Selection

A good choice of sampling time will improve the efficiency of the on-line model estimation algorithm, and will therefore result in a better controller being designed. Ultimately, the choice of sampling time must reflect the response time of the system. As a rule of thumb, approximately 10 sampling intervals should span the rise time of the process. When this rule is followed, the value of  $d$  in Equ. 2.2 typically lies in the range 0 to 5. Therefore, the fixed structure of Equ. 2.4 can be used to represent the majority of cases for which the self-tuning PI controller is designed.

$$Gp(z^{-1}) = \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 + a_1 z^{-1}} \quad (2.4)$$

## 2.2 Recursive Least Squares Estimation

Having developed the necessary model structure for the PI self-tuner (Equ. 2.4), a technique is required to estimate the model parameters. The Recursive Least Squares (RLS) algorithm provides a general purpose statistical tool [2] for estimating the parameters of any system that can be represented

by the summation:

$$y = \sum_{i=1}^n \theta_i x_i \quad (2.4a)$$

or, alternatively, in vector notation as:

$$y = \underline{\theta}^T \cdot \underline{x} \quad (2.5b)$$

where  $\underline{\theta}^T$  is the parameter vector,

and  $\underline{x}$  is the data vector.

It is recursive in the sense that the algorithm updates its parameter estimates as each new observation, or sample, is recorded. The discrete time transfer of Equ. 2.4 is a subset of Equ. 2.5, with:

$$\begin{aligned} \underline{\theta}^T &= [ a_1 \quad b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 ] \\ \underline{x} &= [ -y_{k-1} \quad u_{k-1} \quad u_{k-2} \quad u_{k-3} \quad u_{k-4} \quad u_{k-5} ]^T \end{aligned} \quad (2.6)$$

where  $k$  refers to the present sample value and  $k-n$  to the value  $n$  samples in the past. Therefore, provision of the input and output data required to complete the data vector defined above will allow the RLS algorithm, detailed in Appendix A, to estimate the discrete time model parameters.

In practice, the input/output data are pre-filtered by a digital band-pass filter, given in Equ. 2.7, in order to remove d.c. offsets and high frequency noise, thus making the estimator more robust.

$$G_{pf}(z^{-1}) = \frac{(1 - \alpha) [ 1 - z^{-1} ]}{1 - \alpha z^{-1}} \quad (2.7)$$

Ideally the coefficient  $\alpha$  should be chosen to be equal to the system bandwidth ( $-a_1$  in Equ. 2.4), but the choice is not critical and a default value of  $\alpha = 0.5$  is employed.

### 3. Controller Design

#### 3.1 Performance Specification

The specification for system performance is in terms of the maximum percentage overshoot of the closed loop system's step response. This specification is translated, using the well documented theory of the behaviour of second order systems to the frequency domain concept of phase margin ( $\phi_m$ ) using the relationships:

(i) Maximum % overshoot of an ideal second order process to a step function input:

$$\text{maximum \% overshoot} = 100 \cdot e^{-(\zeta\pi / \sqrt{1 - \zeta^2})} \quad (3.1)$$

(ii) Phase margin of an ideal second order system [3]:

$$\phi_m = \tan^{-1} \left[ \frac{2\pi}{\sqrt{\sqrt{4\zeta^4 + 1} - 2\zeta^2}} \right] \quad (3.2)$$

When translated, the phase margin specification for a stable system will lie in the range:

$$0^\circ < \phi_m < 90^\circ \quad (3.3)$$

In general, the smaller the phase margin, the faster and more oscillatory the closed loop system's behaviour. Larger phase margins result in less oscillatory, more sluggish responses. A good default value for phase margin is

$60^\circ$ , which produces a cautious response with little overshoot. Fig. 4 illustrates the responses of PI controllers designed for different phase margins.

### 3.2 Controller Design Algorithm

In order to establish the controller design algorithm for processes described by the discrete time model of Equ. 2.4, we must first consider the discrete time structure of the ACCOL PI controller:

$$\begin{aligned} G_c(z^{-1}) &= K_c \left[ 1 + \frac{K_i T z^{-1}}{1 - z^{-1}} \right] \\ &= K_c \left[ \frac{1 + (K_i T - 1)z^{-1}}{1 - z^{-1}} \right] \end{aligned} \quad (3.4)$$

Selection of the controller's numerator to cancel the denominator of the discrete time transfer function of Equ. 2.4 fixes the value of  $K_i$ :

$$K_i = (1 + a_1)/T \quad (3.5)$$

This results in the compensated open loop transfer function:

$$G_{OL}(z^{-1}) = K_c \left[ \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 - z^{-1}} \right] \quad (3.6)$$

To establish the value of  $K_c$  which will provide the required phase margin, the frequency response of the compensated system must be computed. This is achieved using the discrete time to frequency domain mapping:

$$z^{-1} = e^{-j\omega T} \quad (3.7)$$

Under this transformation, the open loop phase shift can be evaluated at any frequency  $\omega$ , using the relationship:

$$\angle G_{OL}(j\omega) = -\tan^{-1} \left[ \frac{\sin \omega T}{1 - \cos \omega T} \right] - \tan^{-1} \left[ \frac{\sum_{i=1}^5 b_i \sin i\omega T}{\sum_{i=1}^5 b_i \cos i\omega T} \right] \quad (3.8)$$

The angular frequency,  $\omega_0$ , at which the required phase margin occurs can be simply evaluated, since at this frequency:

$$\angle G_{OL}(j\omega_0) = -180^\circ + \phi_m \quad (3.9)$$

The combinations of Equ's. 3.8 and 3.9 allows  $\omega_0$  to be computed using a linear search.

## APPENDIX A.2

## Recursive Least Squares (RLS) Parameter Estimation

The algorithm uses the following vectors:

Data vector:  $x_k = [ -y_{k-1}^*, u_{k-1}^*, u_{k-2}^*, u_{k-3}^*, u_{k-4}^*, u_{k-5}^* ]^T$

Parameter vector:  $\hat{\theta} = [ -\hat{a}_1, \hat{b}_1, \hat{b}_2, \hat{b}_3, \hat{b}_4, \hat{b}_5 ]^T$  (init. values = 0 )

"Diagonal" vector: diag ( 6 elements, initial values =  $10^6$  )

"Upper triangular" vector: upper ( 15 elements, initial values = 0 )

"Kalman gain" vector: K ( 6 elements )

The calculations performed at every sampling interval are:

(i) Form the data vector

(ii) Calculate the prediction error,  $e_k$

$$e_k = x_k^T \cdot \hat{\theta}_{k-1} - y_k$$

(iii) Update covariance matrix (UD method)

$$f_j = x(1)$$

$$v_j = \text{diag}(1) \cdot f_j$$

$$\alpha_j = 1 + v_j \cdot f_j$$

$$\text{diag}(1) = \text{diag}(1) / \alpha_j$$

$$K(1) = v_j$$

$$K_f = 0$$

$$K_u = 0$$

FOR j=2 TO 6 STEP 1

$$f_j = x(j)$$

$$j1 = j-1$$

FOR i=1 TO j1 STEP 1

$$K_f = K_f + 1$$

$$f_j = f_j + x(i) \cdot \text{upper}(K_f)$$

ENDFOR

```

vj = fj . diag(j)
αlast = αj
αj = αlast + vj . fj
diag(j) = diag(j) . αlast / αj
K(j) = vj
pj = -fj / αlast
FOR i=1 TO j1 STEP 1
    Ku = Ku + 1
    temp = upper(Ku) + K(i) . pj
    K(i) = K(i) + upper(Ku) . vj
    upper(Ku) = temp
ENDFOR
ENDFOR
(iv) Update the parameter vector

$$\hat{\theta}_k = \hat{\theta}_{k-1} - \underline{K}_k \cdot e_k / \alpha_j$$


```

## APPENDIX A.3

## Controller Design Algorithm

Discrete time plant model (sampling time = T seconds):

$$G_P(z^{-1}) = \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 + a_1 z^{-1}}$$

Discrete time proportional-plus-integral (PI) controller:

$$\begin{aligned} G_C(z^{-1}) &= K_C \left[ 1 + \frac{K_I T z^{-1}}{1 - z^{-1}} \right] \\ &= K_C \left[ \frac{1 - z^{-1} + K_I T z^{-1}}{1 - z^{-1}} \right] \\ &= K_C \left[ \frac{1 + (K_I T - 1) z^{-1}}{1 - z^{-1}} \right] \end{aligned}$$

By choosing  $K_I$  such that  $(K_I T - 1) = a_1 \Rightarrow K_I = (1 + a_1)/T$

the open loop transfer function becomes:

$$G_{OL}(z^{-1}) = K_C \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 - z^{-1}}$$

By replacing  $z^{-1}$  by  $e^{-j\omega T}$ , the phase of this plant can be computed as:

$$\angle G_{OL}(j\omega) = -\tan^{-1} \left[ \frac{\sin \omega T}{1 - \cos \omega T} \right] - \tan^{-1} \left[ \frac{\sum_{i=1}^5 b_i \sin i\omega T}{\sum_{i=1}^5 b_i \cos i\omega T} \right]$$

For a certain phase margin,  $\phi_m$ , the angular frequency  $\omega_0$  is found from:

$$\angle G_{OL}(j\omega) = -\pi + \phi_m$$

$\omega_0$  is computed by using a linear search in the range  $0 < \omega < \frac{\pi}{T}$

The fact that the  $\tan^{-1}$  function is not available can be resolved by re-writing the expression as:

$$\tan(\angle G_{OL}(j\omega)) = \tan(-\pi + \phi_m)$$

$$\text{and recalling that } \tan(\alpha + \beta) = \frac{\tan \alpha + \tan \beta}{1 - \tan \alpha \tan \beta}$$

Therefore the search equation becomes:

$$-\frac{(A+B)}{1-A \cdot B} = \tan \phi_m$$

$$\text{where } A = \left[ \frac{\sin wT}{1 - \cos wT} \right] \quad \text{and} \quad B = \left[ \frac{\sum_{i=1}^5 b_i \sin iwT}{\sum_{i=1}^5 b_i \cos iwT} \right]$$

The search algorithm is:

```

wT_high = π, wT_low = π/100, wT_inc = (wT_high - wT_low)/10
FOR pass = 1 TO 2 STEP 1
  w_0T = wT_low, flag = 0
  FOR wT = wT_low TO wT_high STEP wT_inc
    IF f(A,B,wT) < tan φ_m THEN flag = 1
    IF flag = 0 THEN w_0T = wT
  ENDFOR
  wT_high = w_0T + wT_inc, wT_low = w_0T, wT_inc = (wT_high - wT_low)/5
ENDFOR

```

Once  $w_0T$  has been determined,  $K_C$  may be evaluated using  $|G_{OL}(jw)| = 1$ .

$$\Rightarrow K_C \left[ \frac{\sqrt{\left[ \sum_{i=1}^5 b_i \cos iw_0T \right]^2 + \left[ \sum_{i=1}^5 b_i \sin iw_0T \right]^2}}{\sqrt{[1 - \cos w_0T]^2 + [\sin w_0T]^2}} \right] = 1$$

**Appendix B**

**THE INTEGRATED STPI MODULE/PROCESS SIMULATION ACCOL PROGRAM**



## APPENDIX B

```

*TARGET 3330      VERS: 0084
*SECURITY-CODES  6  555555  444444  333333  222222  111111
*MEMORY
  EXPANDED_MEM    OK
  RO_ARRAY_LOC    BASE
  EQUATION_LOC    BASE
  RW_ARRAY_LOC    BASE
  AGAB_LOC        BASE
  STORAGE_ROWS    0
  EVENTS          0
  TEMPLATES       0
*COMMUNICATIONS
  AUX_1  UNUSED
  AUX_2  UNUSED
  PORT_A  SLAVE      9600
  PORT_B  PSLAVE     9600
  PORT_C  UNUSED
  PORT_D  UNUSED
  BUFFERS 15
*PROCESS-I/O
  1  4AI
  2  4AI
  3  2AO
  4  2AO
  5  2AO
*TASK 1  RATE: 1.0  PRI: 31
*TASK 9  RATE: 0.0  PRI: 1
*TASK 10 RATE: 0.3  PRI: 1
*TASK 11 RATE: 0.3  PRI: 1
*TASK 12 RATE: 0.3  PRI: 1
*TASK 13 RATE: 0.3  PRI: 1
*TASK 14 RATE: 0.3  PRI: 1
*TASK 15 RATE: 0.3  PRI: 1
*TASK 16 RATE: 0.3  PRI: 1
*BASENAMES
*SIGNALS
  #ALARM.FORMAT.      L R1 W4 MI CI      O ON      OFF
  #DIAG.001.          LA R1 W4 MI CI  AE O ON      OFF  TRUE  C
  #DIAG.002.          A R1 W4 MI CI                      0.0000000
  #DIAG.003.          A R1 W4 MI CI                      60.0000000 SECS
  #DIAL.000.          A R1 W4 MI CI                      0.0000000
  #DIAL.001.          A R1 W4 MI CI                      0.0000000
  #DIAL.002.          A R1 W4 MI CI                      0.0000000
  #DIAL.003.          A R1 W4 MI CI                      0.0000000
  #E..                A R1 W4 MI CI                      2.7182817
  #ERARRAY..         A R1 W4 MI CI                      10.0000000
  #ERRCT.000.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C
  #ERRCT.001.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C
  #ERRCT.009.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C
  #ERRCT.010.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C
  #ERRCT.011.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C
  #ERRCT.012.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C
  #ERRCT.013.        AA R1 W4 MI CI  AE                      0.0000000 ERRORS
  HALM: #ERRCT.LIM.      A  C

```

#ERRCT.014.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #ERRCT.LIM.	A C	
#ERRCT.015.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #ERRCT.LIM.	A C	
#ERRCT.016.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #ERRCT.LIM.	A C	
#ERRCT.LIM.	A R1 W4 MI CI	0.0000000	ERRORS
#LINE.000.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#LINE.001.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#LINE.002.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#LINE.003.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#LINE.004.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#LINE.005.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#LINKE.001.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #LINKE.LIM.	A C	
#LINKE.002.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #LINKE.LIM.	A C	
#LINKE.LIM.	A R1 W4 MI CI	20.0000000	ERRORS
#LINKF.001.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #LINKF.LIM.	A C	
#LINKF.002.	AA R1 W4 MI CI AE	0.0000000	ERRORS
	HALM: #LINKF.LIM.	A C	
#LINKF.LIM.	A R1 W4 MI CI	20.0000000	ERRORS
#NDARRAY..	A R1 W4 MI CI	0.0000000	
#NODEADR..	A R1 W4 MI CI	0.0000000	
#OCTIME..	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#OCTIME.ERROR.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#OFF..	L R1 W4 MI CI 0 ON	OFF	
#ON..	L R1 W4 MI CI 1 ON	OFF	
#PDM.000.	A R1 W4 ME CI	0.0000000	
#PDM.001.	A R1 W4 ME CI	0.0000000	
#PDM.002.	L R1 W4 ME CI 0 ON	OFF	
#PDM.003.	L R1 W4 MI CE 0 ON	OFF	
#PDM.004.	A R1 W4 MI CE	0.0000000	
#PDM.005.	A R1 W4 MI CE	0.0000000	
#PDM.006.	A R1 W4 MI CE	0.0000000	
#PDM.007.	A R1 W4 MI CE	0.0000000	
#PDM.008.	A R1 W4 MI CE	0.0000000	
#PI..	A R1 W4 MI CI	3.1415927	
#POLLPER.000.	A R1 W4 MI CI	20.0000000	SECS
#POLLPER.001.	A R1 W4 MI CI	20.0000000	SECS
#POLLPER.002.	A R1 W4 MI CI	20.0000000	SECS
#POLLPER.003.	A R1 W4 MI CI	20.0000000	SECS
#POLLPER.004.	A R1 W4 MI CI	20.0000000	SECS
#POLLPER.005.	A R1 W4 MI CI	20.0000000	SECS
#PRI.001.	A R1 W4 MI CI	31.0000000	
#PRI.009.	A R1 W4 MI CI	1.0000000	
#PRI.010.	A R1 W4 MI CI	1.0000000	
#PRI.011.	A R1 W4 MI CI	1.0000000	
#PRI.012.	A R1 W4 MI CI	1.0000000	
#PRI.013.	A R1 W4 MI CI	1.0000000	
#PRI.014.	A R1 W4 MI CI	1.0000000	
#PRI.015.	A R1 W4 MI CI	1.0000000	
#PRI.016.	A R1 W4 MI CI	1.0000000	
#PWRUP.000.	LA R1 W4 MI CI AE 0 ON	OFF	TRUE C
#RATE.001.	A R1 W4 MI CI	1.0000000	
#RATE.009.	A R1 W4 MI CI	0.0000000	
#RATE.010.	A R1 W4 MI CI	0.2500000	
#RATE.011.	A R1 W4 MI CI	0.2500000	
#RATE.012.	A R1 W4 MI CI	0.2500000	

#RATE.013.	A R1 W4 MI CI			0.2500000
#RATE.014.	A R1 W4 MI CI			0.2500000
#RATE.015.	A R1 W4 MI CI			0.2500000
#RATE.016.	A R1 W4 MI CI			0.2500000
#RCNT.001.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.009.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.010.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.011.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.012.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.013.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.014.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.015.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.016.	AA R1 W4 MI CI	AE		0.0000000 COUNTS
	HALM: #RCNT.LIM.		A C	
#RCNT.LIM.	A R1 W4 MI CI			20.0000000 COUNTS
#RTTIME.000.	A R1 W4 ME CI			0.0000000
#RTTIME.001.	A R1 W4 MI CI			0.0000000
#SPARE.000.	L R1 W4 ME CE	0 ON	OFF	
#SPARE.001.	L R1 W4 ME CE	0 ON	OFF	
#SPARE.002.	L R1 W4 ME CE	0 ON	OFF	
#SPARE.003.	L R1 W4 ME CE	0 ON	OFF	
#SPARE.004.	L R1 W4 ME CE	0 ON	OFF	
#SPARE.005.	L R1 W4 ME CE	0 ON	OFF	
#TIME.000.	A R1 W4 MI CI			0.0000000
#TIME.001.	A R1 W4 MI CI			0.0000000 SECS
#TIME.002.	A R1 W4 MI CI			0.0000000
#TIME.003.	A R1 W4 MI CI			0.0000000
#TIME.004.	A R1 W4 MI CI			0.0000000
#TIME.005.	A R1 W4 MI CI			0.0000000 HOURS
#TIME.006.	A R1 W4 MI CI			0.0000000 MINS
#TIME.007.	A R1 W4 MI CI			0.0000000 SECS
A..	A			0.0000000
A1..	A			0.0000000
ACCEPT..	A			50.0000000 %
ADAPT.FLAG.	L	0 ON	OFF	
ADAPTIVE..	L	0 ON	OFF	
AJ..	A			0.0000000
ALAST..	A			0.0000000
ALGO..	L	0 ON	OFF	
ALG1..	L	0 ON	OFF	
ALG2..	L	0 ON	OFF	
ALPHA..	A			0.5000000
B..	A			0.0000000
B1..	A			0.0000000
B2..	A			0.0000000
B3..	A			0.0000000
B4..	A			0.0000000
B5..	A			0.0000000
C1..	A			0.0000000
CDOT..	A			0.0000000
CIN..	A			0.0000000
CIN.PRIME.	A			0.0000000

COLD..	L	1 ON	OFF
COUNT..	A		1.0000000
DATA.TIMER.TIME	A		0.0000000
DESIGN..	L	0 ON	OFF
DESIGN.FACTOE.KC	A		0.0000000
DESIGN.FACTOR.KC	A		0.0000000
DESIGN.FACTOR.KI	A		0.0000000
DESIGN.K1.	A		1.0000000
DESIGN.K2.	A		-0.6000000
DESIGN.K3.	A		1.0000000
DESIGN.K4.	A		-0.3000000
DIAG1..	A		0.0000000
DIAG2..	A		0.0000000
DIAG3..	A		0.0000000
DIAG4..	A		0.0000000
DIAG5..	A		0.0000000
DIAG6..	A		0.0000000
DIAG7..	A		0.0000000
DISTURB..	L	0 ON	OFF
DONE..	A		0.0000000
DONE.FACTOR.	A		0.0000000
EO..	A		0.0000000
E1..	A		0.0000000
ENABLE..	L	0 ON	OFF
ENABLE.NEW.	L	0 ON	OFF
ENABLE.OLD.	L	0 ON	OFF
EVAR..	A		0.0000000
FJ..	A		0.0000000
FLAG.ERMAX.	L	0 ON	OFF
FLAG.INIT.	L	0 ON	OFF
HYSTER..	A		0.2000000
I..	A		0.0000000
IMAG..	A		0.0000000
INPUT..	A	GLB	0.0000000
INPUT.OLD.	A		0.0000000
INT1..	A	GLB	1.0000000
INT2..	A		0.0000000
INTEG..	L	1 ON	OFF
IPMAX..	A		100.0000000
IPMIN..	A		0.0000000
IPSPAN..	A		100.0000000
IPZERO..	A		0.0000000
J..	A		0.0000000
J1..	A		0.0000000
K..	A		0.0000000
KF..	A		0.0000000
KP..	A		1.0000000
KP1..	A		2.0000000
KP2..	A		1.0000000
KU..	A		0.0000000
LAMBDA..	A		0.9900000
LHS..	A		0.0000000
LOAD.INIT.	L	0 ON	OFF
MAGN..	A		0.0000000
MAGN.DEN.	A		0.0000000
MAGN.NUM.	A		0.0000000
NOISE..	A	GLB	0.0000000
NOISE.AMP.	A	GLB	0.0000000
NOISE.REQ.	L	GLB 0 ON	OFF
OPDEV..	A		5.0000000

OPMAX..	A		100.0000000 %
OPMEAN..	A		0.0000000 %
OPMIN..	A		0.0000000 %
OPSPAN..	A		100.0000000 %
OPZERO..	A		0.0000000 %
OUTPUT.SAMPLE.	L	0 ON	OFF
OUTPUT1..	A		0.0000000 %
OUTPUT2..	A	GLB	0.0000000 %
OUTPUT2.LIMIT.	L	0 ON	OFF
OUTPUT2.OLD.	A		0.0000000 %
OVERSH..	A		10.0000000 %
OVERSH.CHECK.	A		0.0000000 %
OVERSH.MAX.	A		75.0000000 %
OVERSH.MIN.	A		1.0000000 %
PASS..	A		0.0000000
PERIOD..	A		0.0000000 SECS
PERR..	A		0.0000000
PHI..	A		0.0000000 DEG
PJ..	A		0.0000000
PRBS.BIT0.	L	1 ON	OFF
PRBS.BIT1.	L	1 ON	OFF
PRBS.BIT2.	L	1 ON	OFF
PRBS.BIT3.	L	1 ON	OFF
PRBS.BIT4.	L	1 ON	OFF
PRBS.BIT5.	L	1 ON	OFF
PRBS.BIT6.	L	1 ON	OFF
PRBS.BIT7.	L	1 ON	OFF
PRBS.BIT8.	L	1 ON	OFF
PRBS.BIT8.NEW	L	0 ON	OFF
PROC.INIT.	L	0 ON	OFF
PROC1.INIT.	L	0 ON	OFF
PROC2.INIT.	L	0 ON	OFF
PROP1..	A	GLB	1.0000000
PROP2..	A		0.0000000
PVAMP..	A		0.0000000 %
PVDEV..	A		100.0000000 %
PVERROR..	A		0.0000000 %
PVERROR.MAX.	A		0.0000000 %
PVERROR.MAX.OLD	A		0.0000000 %
PVERROR.MIN.	A		0.0000000 %
PVERROR.MIN.OLD	A		0.0000000 %
Q1..	A		10.0000000
Q2..	A		0.0000000
REAL..	A		0.0000000
REDUCE.ENABLE.	L	0 ON	OFF
REDUCE.FACTOR.	A		0.6666667
RELAY..	A		2.0000000 %
RELAY.REDUCE.	L	0 ON	OFF
RELAY.SIGN.	A		0.0000000
RESET..	L	0 ON	OFF
RESET.TIMER.DATA	L	0 ON	OFF
RESET1..	L	0 ON	OFF
RESET2..	L	0 ON	OFF
REVERSE..	L	0 ON	OFF
RHS..	A		0.0000000
SELECT..	A		0.0000000
SETPOINT..	A	GLB	40.0000000 %
SIM..	A		1.0000000
SIM.C.	A		0.0000000
SIM.C1.	A		10.0000000

SIM.C2.	A		90.0000000
STATUS..	A		0.0000000
STATUS.BIT0.	A		0.0000000
STATUS.BIT1.	A		0.0000000
STATUS.BIT2.	A		0.0000000
STATUS.BIT3.	A		0.0000000
STATUS.BIT4.	A		0.0000000
STATUS.BIT5.	A		0.0000000
STATUS.BIT6.	A		0.0000000
STPI.Y0.	A		0.0000000
STPI.Y1.	A		0.0000000
SYSINIT..	L	0 ON	OFF
TANPHI..	A		0.0000000
TAU.DEL.	A		0.0000000
TAU.L.	A		0.0000000
TAU.P.	A		5.0000000
TAU.P1.	A		5.0000000
TAU.P2.	A		0.5000000
TEMP..	A		0.0000000
TESTPI..	L	0 ON	OFF
TESTPI.TIME.	A		120.0000000
THI..	A		0.0000000
THRESH..	A		5.0000000 %
TIMER.IN.	L	0 ON	OFF
TIMER.OUTPUT.1	L	0 ON	OFF
TIMER.PULSE.	L	0 ON	OFF
TOTAL..	A		100.0000000 COUNTS
TRACK1..	L	0 ON	OFF
TRACK2..	L	0 ON	OFF
TUNE.ALPHA.	A		1.0000000
TUNE.BETA.	A		2.0000000
TUNE.COUNT.	A		0.0000000 COUNTS
TUNE.COUNT.OLD	A		0.0000000 COUNTS
TUNE.DB.	A		2.0000000 %
TUNE.GAMMA.	A		6.0000000
TUNE.LOCK.	L	0 ON	OFF
TUNE.R1.	A		0.0000000 %
TUNE.R2.	A		0.0000000 %
TUNE.S1.	A		0.0000000
TUNE.S2.	A		0.0000000
TUNE.STAGE.1	L	0 ON	OFF
TUNE.STAGE.2	L	0 ON	OFF
TUNE.STAGE.3	L	0 ON	OFF
TUNE.STAGE.4	L	0 ON	OFF
TUNE.T0.	A		0.0000000 COUNTS
TUNE.T2.	A		0.0000000 COUNTS
TUNE.T3.	A		0.0000000 COUNTS
TUNE.T4.	A		0.0000000 COUNTS
TUNE.T5.	A		500.0000000 COUNTS
TUNE.TL.	A		0.0000000 COUNTS
TUNING..	L	0 ON	OFF
U0..	A		0.0000000
U1..	A		0.0000000
U2..	A		0.0000000
U3..	A		0.0000000
U4..	A		0.0000000
U5..	A		0.0000000
UPDATE..	L	1 ON	OFF
VD..	A		100.0000000
VJ..	A		0.0000000

VM..	A		1000.0000000
WN..	A		2.0000000
WT..	A		0.0000000
WT.O.	A		0.0000000
WT.FLAG.	L	O ON	OFF
WT.HIGH.	A		0.0000000
WT.INC.	A		0.0000000
WT.LOW.	A		0.0000000
X1..	A		0.0000000
X1DOT..	A		0.0000000
X2..	A		0.0000000
X2DOT..	A		0.0000000
Y..	A	GLB	0.0000000
Y.RES.	A		0.0000000
Y.SPAN.	A		100.0000000 %
Y.TRK.	L	O ON	OFF
Y.ZERO.	A		0.0000000 %
Y1..	A		0.0000000
Y1DOT..	A		0.0000000
Y2..	A		0.0000000
Y2.MAX.	A		100.0000000
Y2.MIN.	A		0.0000000
Y2DOT..	A		0.0000000
YD..	A		0.0000000
YP..	A		0.0000000
YP.MAX.	A		100.0000000
YP.MIN.	A		0.0000000
YPDOT..	A		0.0000000
YVAR..	A		0.0000000
Z..	A		0.0000000
Z.SPAN.	A		100.0000000 %
Z.ZERO.	A		0.0000000 %
ZEROX.COUNT.	A		0.0000000 COUNTS
ZEROX.TOTAL.	A		0.0000000 COUNTS
ZETA..	A		0.0000000
ZETA.SIM.	A		0.5000000
ZL..	A	GLB	0.0000000
ZL.IN.	A		0.0000000
ZLDOT..	A		0.0000000

\*TASK 0

\*TASK 1

```

10 * C *****
20 * C ***
30 * C *** SELF TUNING PI CONTROLLER ***
40 * C ***
50 * C *** 22ND MARCH 1990 ***
60 * C ***
70 * C *****
80 * C
90 * C *** RESET DEFAULT VALUES BETWEEN STPISIM TESTS
100 * CALCULATOR
10 :IF(RESET)
12 TESTPI=#OFF
15 ENABLE=#OFF
20 STATUS.BIT0=0.
30 STATUS.BIT1=0.
40 STATUS.BIT2=0.
50 STATUS.BIT3=0.
60 STATUS.BIT4=0.
70 STATUS.BIT5=0.

```

```

80  STATUS.BIT6=0.
90  PROP2=0.
100 INT2=0.
110 DONE=0.
115  :CI:ZL=#OFF
120  ZL=0.
125  :CI:ZL=#ON
130  REVERSE=#OFF
140  OVERSH=10.
150  SETPOINT=40.
160  PROP1=1.
170  INT1=1.
180  PVDEV=100.
190  RELAY=2.
200  INTEG=#ON
210  ACCEPT=50.
220  HYSTER=0.2
230  THRESH=5.
240  OPMEAN=1.
250  OPDEV=5.
260  TOTAL=100.
270  ALPHA=0.5
280  LAMBDA=0.99
290  A1=0.
300  B1=0.
310  B2=0.
320  B3=0.
330  B4=0.
340  B5=0.
350  C1=0.
360  DIAG1=0.
370  DIAG2=0.
380  DIAG3=0.
390  DIAG4=0.
400  DIAG5=0.
410  DIAG6=0.
420  DIAG7=0.
430  RESET=#OFF
440 :ENDIF
110 * C
112 * C *** TEST PI VALUES (STEP RESPONSE & DISTURBANCE RESPONSE)
115 * C
120 * TIMER
      INPUT          TESTPI..
      SETPOINT       TESTPI.TIME.
      RESET          TESTPI..
      OUTPUT_2       DISTURB..
130 * CALCULATOR
      10 :IF(TESTPI)
      20   SETPOINT=50.
      30 :ENDIF
      40 :IF(DISTURB)
      50   :CI:ZL=#OFF
      60   ZL=10.
      70   :CI:ZL=#ON
      80 :ENDIF
140 * C
150 * C *** SELECT ALGORITHM *****
160 * C
170 * C

```

```

180 * CALCULATOR
  10 :IF(SELECT==1)
  20   ALGO=#ON
  30   ALG1=#OFF
  40   ALG2=#OFF
  50 :ENDIF
  60 :IF(SELECT==2)
  70   ALGO=#OFF
  80   ALG1=#ON
  90   ALG2=#OFF
 100 :ENDIF
 110 :IF(SELECT==3)
 120   ALGO=#OFF
 130   ALG1=#OFF
 140   ALG2=#ON
 150 :ENDIF
 160 ENABLE.NEW=ENABLE
 170 :IF(ENABLE.OLDE-ENABLE.NEW)
 180   TUNING=#OFF
 190   TUNE.LOCK=#OFF
 200 :ENDIF
190 * C
200 * C
210 * C *** MEASURE THE PROCESS VARIABLE *****
220 * C
230 * C
240 * ANIN
      DEVICE                1
      INITIAL                2
      INPUT          1      INPUT..
      ZERO           1      IPZERO..
      SPAN           1      IPSPAN..
250 * CALCULATOR STPI.YO=ALPHA*STPI.Y1+(1-ALPHA)*(INPUT-INPUT.OLD)
260 * C
270 * C
280 * C *** UPDATE THE OUTPUT OF THE PID3TERM MODULE *****
290 * C
300 * C
310 * PID3TERM
      INPUT          INPUT..
      SETPOINT       SETPOINT..
      PROPORTION     PROP1..
      INTEGRAL       INT1..
      RESET          OUTPUT2..
      TRACK          TRACK2..
      OUTPUT         OUTPUT1..
320 * C
330 * C
340 * C *** UPDATE THE OUTPUT OF THE STPI MODULE *****
350 * C
360 * C --- UPDATE PROCEDURE WHEN NOT TUNING -----
370 * C
380 * IF (~TUNING)
390 *   CALCULATOR
      10 OUTPUT2=OUTPUT1
      20 TRACK2=TRACK1
400 * ENDIF
410 * C
420 * IF (TUNING)
430 * C

```

```

440 * C --- UPDATE PROCEDURE WHEN TUNING WITH ALGORITHM #0 -----
450 * C
460 *   IF (ALGO)
470 *     CALCULATOR
    10 :IF(-REVERSE)
    20   :IF(INTEG)
    30     OUTPUT2=OUTPUT2+RELAY.SIGN*RELAY*#RATE.001
    40   :ENDIF
    50   :IF(-INTEG)
    60     OUTPUT2=OPMEAN+RELAY.SIGN*RELAY
    70   :ENDIF
    80 :ENDIF
    90 :IF(REVERSE)
   100   :IF(INTEG)
   110     OUTPUT2=OUTPUT2-RELAY.SIGN*RELAY*#RATE.001
   120   :ENDIF
   130   :IF(~INTEG)
   140     OUTPUT2=OPMEAN-RELAY.SIGN*RELAY
   150   :ENDIF
   160 :ENDIF
   170 TRACK2=#ON
480 *   ENDIF
490 * C
500 * C --- UPDATE PROCEDURE WHEN TUNING WITH ALGORITHM #1 -----
510 * C
520 *   IF (ALG1)
530 *     CALCULATOR
    10 OUTPUT2=OUTPUT1
    20 TRACK2=TRACK1
540 *   ENDIF
550 * C
560 * C --- UPDATE PROCEDURE WHEN TUNING WITH ALGORITHM #2 -----
570 * C
580 *   IF (ALG2)
590 *     CALCULATOR
    10 :IF(~ADAPT.FLAG)
    20   :IF(PRBS.BIT0)
    30     OUTPUT2=OPMEAN+OPDEV
    40   :ENDIF
    50   :IF(~PRBS.BIT0)
    60     OUTPUT2=OPMEAN-OPDEV
    70   :ENDIF
    80   TRACK2=#ON
    90 :ENDIF
   100 :IF(ADAPT.FLAG)
   110   OUTPUT2=OUTPUT1
   120   TRACK2=TRACK1
   130 :ENDIF
600 *   ENDIF
610 * ENDIF
620 * C
630 * C --- LIMIT OUTPUT AND CHECK FOR INPUT LIMITING -----
640 * C
650 * CALCULATOR
    10 OUTPUT2.LIMIT=#OFF
    20 :IF(OUTPUT2<OPMIN)
    30   OUTPUT2=OPMIN
    40   OUTPUT2.LIMIT=#ON
    50   TRACK2=#ON
    60 :ENDIF

```

```

70 :IF(OUTPUT2>OPMAX)
80  OUTPUT2=OPMAX
90  OUTPUT2.LIMIT=#ON
100 TRACK2=#ON
110 :ENDIF
120 :IF(TUNING)
130  :IF(OUTPUT2.LIMIT|(INPUT<=IPMIN)|(INPUT>=IPMAX))
140    TUNE.STAGE.1=#OFF
150    TUNE.STAGE.2=#OFF
160    TUNE.STAGE.3=#OFF
170    TUNE.STAGE.4=#OFF
180    ZEROX.COUNT=0
190    STATUS.BIT5=32
200  :ENDIF
210 :ENDIF
660 * C
670 * C
680 * C *** OUTPUT THE NEW ACTUATOR COMMAND SIGNAL *****
690 * C
700 * C
710 * ANOUT
      DEVICE                3
      INITIAL                2
      OUTPUT      1      OUTPUT2..
      ZERO        1      OPZERO..
      SPAN        1      OPSPAN..
720 * C
730 * C
740 * C *** TUNING INITIALISATION PROCEDURES *****
750 * C
760 * C
770 * IF (~ENABLE.OLD&ENABLE.NEW)
780 *   CALCULATOR
      10 TUNING=#ON
      20 STATUS.BIT0=0
      30 STATUS.BIT1=0
      40 STATUS.BIT2=0
      50 STATUS.BIT3=0
      60 STATUS.BIT4=0
      70 STATUS.BIT5=0
      80 STATUS.BIT6=0
790 *   IF (SELECT==0)
800 *     CALCULATOR
      10 ALGO=#ON
      20 ALG1=#OFF
      30 ALG2=#OFF
810 *   ENDIF
820 * C
830 * C --- TUNING INITIALISATION PROCEDURE FOR ALGORITHM #0 -----
840 * C
850 *   IF (ALGO)
860 *     CALCULATOR
      10 REDUCE.ENABLE=#OFF
      20 TUNE.COUNT=0
      30 ZEROX.COUNT=0
      40 ZEROX.TOTAL=0
      50 :IF((SETPOINT-INPUT)>=0.0)
      60   RELAY.SIGN=1
      70 :ENDIF
      80 :IF(SETPOINT-INPUT<0.0)

```

```

90 RELAY.SIGN=-1
100 :ENDIF
110 :IF(-INTEG)
120 OPMEAN=OUTPUT2
130 :ENDIF
870 * ENDF
880 * C
890 * C --- NO INITIALISATION REQUIRED FOR ALGORITHM #1 -----
900 * C
910 * C --- TUNING INITIALISATION PROCEDURE FOR ALGORITHM #2 -----
920 * C
930 * IF (ALG2)
940 * CALCULATOR
10 OPMEAN=OUTPUT2
20 TUNE.COUNT=0
30 EO=0
40 EVAR=0
50 YVAR=0
60 DONE=0
70 ADAPT.FLAG=#OFF
80 DIAG1=1000
90 DIAG2=1000
100 DIAG3=1000
110 DIAG4=1000
120 DIAG5=1000
130 DIAG6=1000
140 DIAG7=1000
950 * ENDF
960 * ENDF
970 * C
980 * C
990 * C *** TUNING PROCEDURES *****
1000 * C
1010 * C
1020 * IF (TUNING)
1030 * CALCULATOR PVELOCITY=SETPOINT-INPUT
1040 * C
1050 * C --- TUNING PROCEDURE FOR ALGORITHM #0 -----
1060 * C
1070 * IF (ALGO)
1080 * CALCULATOR
10 TUNE.COUNT=TUNE.COUNT+1
20 :IF(REDUCE.ENABLE)
30 :IF((:ABS(PVELOCITY)>PVDEV|OUTPUT2.LIMIT)&~RELAY.REDUCE)
40 RELAY.REDUCE=#ON
50 ZEROX.COUNT=0
60 :ENDIF
70 :IF(RELAY.REDUCE&ZEROX.COUNT>0)
80 RELAY.REDUCE=#OFF
90 STATUS.BIT2=4
100 RELAY=REDUCE.FACTOR*RELAY
110 :IF(RELAY<HYSTER)
120 STATUS.BIT3=8
130 :ENDIF
140 :ENDIF
150 :ENDIF
160 :IF(RELAY.SIGN<0&PVELOCITY>HYSTER)
170 REDUCE.ENABLE=#ON
180 ZEROX.COUNT=ZEROX.COUNT+1
190 ZEROX.TOTAL=ZEROX.TOTAL+1

```

```

200 TUNE.DB=:ABS(PVEEROR.MAX.OLD*ACCEPT/100.0)
210 DONE=50+50*(1-:ABS(PVEEROR.MAX-PVEEROR.MAX.OLD)/TUNE.DB)
220 :IF(DONE<0)
230     DONE=0
240 :ENDIF
250 :IF(:ABS(PVEEROR.MAX-PVEEROR.MAX.OLD)<TUNE.DB&ZEROX.COUNT>3)
260     PVAMP=(PVEEROR.MAX-PVEEROR.MIN.OLD)/2.0
270     PERIOD=(TUNE.COUNT+TUNE.COUNT.OLD)*#RATE.001
280     DESIGN=#ON
290     :IF(~INTEG)
300         OUTPUT2=OPMEAN
310     :ENDIF
320 :ENDIF
330 PVEEROR.MAX.OLD=PVEEROR.MAX
340 PVEEROR.MAX=0.0
350 TUNE.COUNT.OLD=TUNE.COUNT
360 TUNE.COUNT=0
370 RELAY.SIGN=1
380 :ENDIF
390 :IF(RELAY.SIGN>0&PVEEROR<-HYSTER)
400     REDUCE.ENABLE=#ON
410     ZEROX.COUNT=ZEROX.COUNT+1
420     ZEROX.TOTAL=ZEROX.TOTAL+1
430     TUNE.DB=:ABS(PVEEROR.MIN.OLD*ACCEPT/100)
440     DONE=50+50*(1-:ABS(PVEEROR.MIN-PVEEROR.MIN.OLD)/TUNE.DB)
450     :IF(DONE<0)
460         DONE=0
470     :ENDIF
480     :IF(:ABS(PVEEROR.MIN-PVEEROR.MIN.OLD)<TUNE.DB&ZEROX.COUNT>3)
490         PVAMP=(PVEEROR.MAX.OLD-PVEEROR.MIN)/2.0
500         PERIOD=(TUNE.COUNT+TUNE.COUNT.OLD)*#RATE.001
510         DESIGN=#ON
515         :IF(~INTEG)
516             OUTPUT2=OPMEAN
517         :ENDIF
520     :ENDIF
530 PVEEROR.MIN.OLD=PVEEROR.MIN
540 PVEEROR.MIN=0.0
550 TUNE.COUNT.OLD=TUNE.COUNT
560 TUNE.COUNT=0
570 RELAY.SIGN=-1.0
580 :ENDIF
590 :IF(ZEROX.TOTAL>21)
600     DESIGN=#ON
610     STATUS.BIT4=16
620 :ENDIF
630 :IF(PVEEROR>PVEEROR.MAX)
640     PVEEROR.MAX=PVEEROR
650 :ENDIF
660 :IF(PVEEROR<PVEEROR.MIN)
670     PVEEROR.MIN=PVEEROR
680 :ENDIF
1090 *   ENDIF
1100 * C
1110 * C --- TUNING PROCEDURE FOR ALGORITHM #1 -----
1120 * C
1130 *   IF (ALG1)
1140 *       CALCULATOR
10     :IF(:ABS(PVEEROR)>THRESH&~TUNE.LOCK)
20     TUNE.LOCK=#ON

```

```

30  TUNE.STAGE.1=#ON
40  PVEERROR.MAX=0.0
50  :ENDIF
60  :IF(TUNE.STAGE.1)
70    :IF(:ABS(PVEERROR)>:ABS(PVEERROR.MAX))
80      FLAG.ERMAX=#OFF
90      PVEERROR.MAX=PVEERROR
100   :ENDIF
110  :IF(:ABS(PVEERROR)<:ABS(PVEERROR.MAX)&~FLAG.ERMAX)
120    FLAG.ERMAX=#ON
130    FLAG.INIT=#ON
140  :ENDIF
150  :IF(FLAG.INIT)
160    FLAG.INIT=#OFF
170    TUNE.STAGE.2=#ON
180    TUNE.STAGE.3=#OFF
190    TUNE.STAGE.4=#OFF
200    STATUS.BIT5=0
210    TUNE.COUNT=0
220    TUNE.R1=OVERSH/150.0
230    TUNE.R2=TUNE.R1/6
240    TUNE.S1=0.0
250    TUNE.S2=0.0
260  :ENDIF
270 :ENDIF
280 :IF(TUNE.STAGE.2&:ABS(PVEERROR)<0.9*ABS(PVEERROR.MAX))
290  TUNE.STAGE.2=#OFF
300  TUNE.STAGE.3=#ON
310  TUNE.T0=TUNE.COUNT
320 :ENDIF
330 :IF(TUNE.STAGE.3&:ABS(PVEERROR)<0.5*ABS(PVEERROR.MAX))
340  TUNE.STAGE.3=#OFF
350  TUNE.STAGE.4=#ON
360  TUNE.TL=TUNE.COUNT-TUNE.T0
370  :IF(TUNE.TL<1)
380    TUNE.TL=1
390  :ENDIF
400  TUNE.T2=TUNE.COUNT+TUNE.ALPHA*TUNE.TL
410  TUNE.T3=TUNE.T2+TUNE.BETA*TUNE.TL
420  TUNE.T4=TUNE.T3+TUNE.GAMMA*TUNE.TL
430  TUNE.T5=TUNE.T4+TUNE.T4
440 :ENDIF
450 :IF(TUNE.STAGE.4)
460  :IF(TUNE.COUNT>=TUNE.T2&TUNE.COUNT<TUNE.T3)
470    TUNE.S1=TUNE.S1+PVEERROR/PVEERROR.MAX/TUNE.BETA/TUNE.TL
480  :ENDIF
490  :IF(TUNE.COUNT>=TUNE.T3&TUNE.COUNT<TUNE.T4)
500    TUNE.S2=TUNE.S2+(PVEERROR/PVEERROR.MAX+TUNE.R2)/TUNE.GAMMA/TUNE.TL
510  :ENDIF
520  :IF(TUNE.COUNT>=TUNE.T4)
530    TUNE.STAGE.1=#OFF
540    TUNE.STAGE.4=#OFF
550    DESIGN=#ON
560    DONE.FACTOR=(ABS(TUNE.R1)-ABS(TUNE.S1+TUNE.R1))/ABS(TUNE.R1)
570    :IF(TUNE.R1==0)
580      DONE.FACTOR=0.5
590    :ENDIF
600    DONE=50+50*DONE.FACTOR
610    :IF(DONE<0)
620      DONE=0

```

```

630      :ENDIF
640      :ENDIF
650 :ENDIF
660 :IF(TUNE.LOCK)
670   TUNE.COUNT=TUNE.COUNT+1
680   :IF(TUNE.COUNT>TUNE.T5&:ABS(PVERROR)<THRESH)
690     TUNE.LOCK=#OFF
700   :ENDIF
710 :ENDIF
1150 *   ENDIF
1160 * C
1170 * C ---- TUNING PROCEDURE FOR ALGORITHM #2 -----
1180 * C
1190 *   IF (ALG2)
1200 *     CALCULATOR
1210 *       10 :IF(-ADAPT.FLAG)
1220 *         20 TUNE.COUNT=TUNE.COUNT+1
1230 *         30 :ENDIF
1240 *       IF (TUNE.COUNT>5)
1250 *         CALCULATOR
1260 *           10 TEMP=1
1270 *           20 :IF(REVERSE)
1280 *             30 TEMP=-1
1290 *           40 :ENDIF
1300 *           50 #ADATA 1{1,1}=-STPI.Y1
1310 *           60 #ADATA 2{1,1}=A1
1320 *           70 #ADATA 3{1,1}=DIAG1
1330 *           80 #ADATA 1{1,2}=U1*TEMP
1340 *           90 #ADATA 2{1,2}=B1
1350 *           100 #ADATA 3{1,2}=DIAG2
1360 *           110 #ADATA 1{1,3}=U2*TEMP
1370 *           120 #ADATA 2{1,3}=B2
1380 *           130 #ADATA 3{1,3}=DIAG3
1390 *           140 #ADATA 1{1,4}=U3*TEMP
1400 *           150 #ADATA 2{1,4}=B3
1410 *           160 #ADATA 3{1,4}=DIAG4
1420 *           170 #ADATA 1{1,5}=U4*TEMP
1430 *           180 #ADATA 2{1,5}=B4
1440 *           190 #ADATA 3{1,5}=DIAG5
1450 *           200 #ADATA 1{1,6}=U5*TEMP
1460 *           210 #ADATA 2{1,6}=B5
1470 *           220 #ADATA 3{1,6}=DIAG6
1480 *           230 #ADATA 1{1,7}=E1
1490 *           240 #ADATA 2{1,7}=C1
1500 *           250 #ADATA 3{1,7}=DIAG7
1510 *         CALCULATOR EO=-STPI.Y0
1520 *         FOR 1., 7., 1., J..
1530 *           CALCULATOR EO=EO+#ADATA 1{1,J}+#ADATA 2{1,J}
1540 *         ENDFOR
1550 *         CALCULATOR
1560 *           10 PERR=EO
1570 *           20 :IF(TUNE.COUNT>12)
1580 *             30 TEMP=TUNE.COUNT-12
1590 *             40 EVAR=(EVAR*(TEMP-1)/TEMP)+(EO*EO/TEMP)
1600 *             50 YVAR=(YVAR*(TEMP-1)/TEMP)+(STPI.Y0*STPI.Y0/TEMP)
1610 *             60 DONE=100*(1-EVAR/YVAR)
1620 *             70 :IF(DONE<0)
1630 *               80 DONE=0
1640 *             90 :ENDIF
1650 *           100 :ENDIF

```

```

1280 *      CALCULATOR
10 FJ=#ADATA 1[1,1]
20 VJ=#ADATA 3[1,1]*FJ
30 AJ=1+(VJ*FJ)
40 #ADATA 3[1,1]=#ADATA 3[1,1]/AJ/LAMBDA
50 #ADATA 5[1,1]=VJ
60 KF=0
70 KU=0
1290 *      FOR 2., 7., 1., J..
1300 *      CALCULATOR
10 FJ=#ADATA 1[1,J]
20 J1=J-1
1310 *      FOR 1., J1.., 1., I..
1320 *      CALCULATOR
10 KF=KF+1
20 FJ=FJ+ (#ADATA 1[1,I]*#ADATA 4[1,KF])
1330 *      ENDFOR
1340 *      CALCULATOR
10 VJ=FJ*#ADATA 3[1,J]
20 ALAST=AJ
30 AJ=ALAST+(VJ*FJ)
40 #ADATA 3[1,J]=#ADATA 3[1,J]*ALAST/AJ/LAMBDA
50 #ADATA 5[1,J]=VJ
60 PJ=-FJ/ALAST
1350 *      FOR 1., J1.., 1., I..
1360 *      CALCULATOR
10 KU=KU+1
20 TEMP=#ADATA 4[1,KU]+(#ADATA 5[1,I]*PJ)
30 #ADATA 5[1,I]=#ADATA 5[1,I]+(#ADATA 4[1,KU]*VJ)
40 #ADATA 4[1,KU]=TEMP
1370 *      ENDFOR
1380 *      ENDFOR
1390 *      FOR 1., 7., 1., J..
1400 *      CALCULATOR
10 #ADATA 2[1,J]=#ADATA 2[1,J]-(#ADATA 5[1,J]*EO/AJ)
1410 *      ENDFOR
1420 *      CALCULATOR
10 A1=#ADATA 2[1,1]
20 B1=#ADATA 2[1,2]
30 B2=#ADATA 2[1,3]
40 B3=#ADATA 2[1,4]
50 B4=#ADATA 2[1,5]
60 B5=#ADATA 2[1,6]
70 C1=#ADATA 2[1,7]
80 DIAG1=#ADATA 3[1,1]
90 DIAG2=#ADATA 3[1,2]
100 DIAG3=#ADATA 3[1,3]
110 DIAG4=#ADATA 3[1,4]
120 DIAG5=#ADATA 3[1,5]
130 DIAG6=#ADATA 3[1,6]
140 DIAG7=#ADATA 3[1,7]
1430 *      CALCULATOR
10 :IF(-ADAPT.FLAG)
20 :IF(TUNE.COUNT>=TOTAL)
30 OUTPUT2=OPMEAN
40 DESIGN=#ON
50 :ENDIF
60 :ENDIF
70 :IF(ADAPT.FLAG)
80 DESIGN=#ON

```

```

90 :ENDIF
1440 *   ENDIF
1450 *   ENDIF
1460 * ENDIF
1470 * C
1480 * C
1490 * C *** SHIFT DATA *****
1500 * C
1510 * C
1520 * CALCULATOR
      10 U0=ALPHA*U1+(1-ALPHA)*(OUTPUT2-OUTPUT2.OLD)
      20 ENABLE.OLD=ENABLE.NEW
      30 INPUT.OLD=INPUT
      40 OUTPUT2.OLD=OUTPUT2
      50 STPI.Y1=STPI.Y0
      60 U5=U4
      70 U4=U3
      80 U3=U2
      90 U2=U1
     100 U1=U0
     110 E1=E0
     120 PRBS.BIT8.NEW=PRBS.BIT1^PRBS.BIT6
     130 PRBS.BIT0=PRBS.BIT1
     140 PRBS.BIT1=PRBS.BIT2
     150 PRBS.BIT2=PRBS.BIT3
     160 PRBS.BIT3=PRBS.BIT4
     170 PRBS.BIT4=PRBS.BIT5
     180 PRBS.BIT5=PRBS.BIT6
     190 PRBS.BIT6=PRBS.BIT7
     200 PRBS.BIT7=PRBS.BIT8
     210 PRBS.BIT8=PRBS.BIT8.NEW
1530 * C
1540 * C
1550 * C *** CONSTRUCT STATUS WORD *****
1560 * C
1570 * C
1580 * CALCULATOR
      10 STATUS.BIT0=0
      20 :IF(TUNING)
      30   STATUS.BIT0=1
      40 :ENDIF
      50 STATUS.BIT1=0
      60 :IF(TUNE.LOCK)
      70   STATUS.BIT1=2
      80 :ENDIF
     100 STATUS=STATUS.BIT0+STATUS.BIT1+STATUS.BIT2+STATUS.BIT3+STATUS.BIT4+@
          STATUS.BIT5+STATUS.BIT6
1590 * C
1600 * C
1610 * C *** DESIGN PROCEDURES *****
1620 * C
1630 * C
1640 * IF (DESIGN)
1650 * C
1660 * C --- CHECK OVERSHOOT -----
1670 * C
1680 *   CALCULATOR
      10 OVERSH.CHECK=OVERSH
      20 :IF(OVERSH>OVERSH.MAX)
      30   OVERSH.CHECK=OVERSH.MAX

```

```

40 :ENDIF
50 :IF(OVERSH<OVERSH.MIN)
60   OVERSH.CHECK=OVERSH.MIN
70 :ENDIF
80 ZETA=:LOG(OVERSH.CHECK/100.0)*:LOG(OVERSH.CHECK/100.0)
90 ZETA=:SQR(ZETA/(ZETA+#PI*#PI))
100 PHI=ZETA*100.0
110 THI=PHI*#PI/180.0
1690 * C
1700 * C --- DESIGN PROCEDURE FOR ALGORITHM #0 -----
1710 * C
1720 *   IF (ALGO)
1730 *     CALCULATOR
10   :IF(~REVERSE)
20   PROP2=2.0*RELAY*PERIOD*:SIN(THI)/(#PI*#PI*PVAMP)
25   :IF(~INTEG)
26   PROP2=OVERSH.CHECK/100.0*4.0*RELAY/#PI/PVAMP
27   :ENDIF
30 :ENDIF
40 :IF(REVERSE)
50   PROP2=-2.0*RELAY*PERIOD*:SIN(THI)/(#PI*#PI*PVAMP)
55   :IF(~INTEG)
56   PROP2=-OVERSH.CHECK/100.0*4.0*RELAY/#PI/PVAMP
57   :ENDIF
60 :ENDIF
70 INT2=120.0*#PI/(PERIOD*:TAN(THI))
75 :IF(~INTEG)
77   INT2=0
78 :ENDIF
80 TUNE.T5=4*INT2/#RATE.001/60
1740 *   ENDIF
1750 * C
1760 * C --- DESIGN PROCEDURE FOR ALGORITHM #1 -----
1770 * C
1780 *   IF (ALG1)
1790 *     CALCULATOR
10   DESIGN.FACTOR.KC=(1-DONE.FACTOR)*(DESIGN.K1*(TUNE.S1+TUNE.R1)+DESIGN.K2@
   *TUNE.S2)
20   :IF(:ABS(DESIGN.FACTOR.KC)>0.2)
30   DESIGN.FACTOR.KC=0.2*DESIGN.FACTOR.KC/:ABS(DESIGN.FACTOR.KC)
40   :ENDIF
50   PROP2=PROP1*(1+DESIGN.FACTOR.KC)
60   DESIGN.FACTOR.KI=(1-DONE.FACTOR)*(DESIGN.K3*(TUNE.S1+TUNE.R1)+DESIGN.K4@
   *TUNE.S2)
70   :IF(:ABS(DESIGN.FACTOR.KI)>0.2)
80   DESIGN.FACTOR.KI=0.2*DESIGN.FACTOR.KI/:ABS(DESIGN.FACTOR.KI)
90   :ENDIF
100  INT2=INT1*(1+DESIGN.FACTOR.KI)
110  :IF(~INTEG)
120  INT2=0
130  :ENDIF
1800 *   ENDIF
1810 * C
1820 * C --- DESIGN PROCEDURE FOR ALGORITHM #2 -----
1830 * C
1840 *   IF (ALG2)
1850 *     CALCULATOR TEMP=0
1860 *     FOR 1., 5., 1., J..
1870 *       CALCULATOR TEMP=TEMP+#ADATA 2[1,1+J]
1880 *     ENDFOR

```

```

1890 *      IF (TEMP<0)
1900 *          CALCULATOR STATUS.BIT6=64
1910 *      ENDIF
1920 *      CALCULATOR
10 INT2=60*(1+#ADATA 2[1,1])/#RATE.001
20 :IF(INT2<0)
30 INT2=0
40 :ENDIF
50 TANPHI=:TAN(PHI*#PI/180)
60 WT.HIGH=#PI
70 WT.LOW=#PI/100
80 WT.INC=(WT.HIGH-WT.LOW)/10
1930 *      FOR 1., 2., 1., PASS..
1940 *      CALCULATOR
10 WT.O=WT.LOW
20 WT.FLAG=#OFF
1950 *      FOR WT.LOW., WT.HIGH., WT.INC., WT..
1960 *      CALCULATOR
10 A=:SIN(WT)/(1:-COS(WT))
20 IMAG=#ADATA 2[1,2]*:SIN(WT)
30 REAL=#ADATA 2[1,2]*:COS(WT)
1970 *      FOR 2., 5., 1., J..
1980 *      CALCULATOR
10 IMAG=IMAG+#ADATA 2[1,J+1]*:SIN(J*WT)
20 REAL=REAL+#ADATA 2[1,J+1]*:COS(J*WT)
1990 *      ENDFOR
2000 *      CALCULATOR
10 B=IMAG/REAL
20 LHS=-:(A+B)/(1-A*B)
30 RHS=TANPHI
40 :IF(LHS<RHS)
50 WT.FLAG=#ON
70 :ENDIF
80 :IF(-WT.FLAG)
90 WT.O=WT
100 :ENDIF
2010 *      ENDFOR
2020 *      CALCULATOR
10 WT.HIGH=WT.O+WT.INC
20 WT.LOW=WT.O
30 WT.INC=(WT.HIGH-WT.LOW)/10
2030 *      ENDFOR
2040 *      CALCULATOR
10 IMAG=:SIN(WT.O)
20 REAL=1:-COS(WT.O)
30 MAGN.DEN=:SQR(REAL*REAL+IMAG*IMAG)
40 IMAG=#ADATA 2[1,2]*:SIN(WT.O)
50 REAL=#ADATA 2[1,2]*:COS(WT.O)
2050 *      FOR 2., 5., 1., J..
2060 *      CALCULATOR
10 IMAG=IMAG+#ADATA 2[1,J+1]*:SIN(J*WT.O)
20 REAL=REAL+#ADATA 2[1,J+1]*:COS(J*WT.O)
2070 *      ENDFOR
2080 *      CALCULATOR
10 MAGN.NUM=:SQR(REAL*REAL+IMAG*IMAG)
20 MAGN=MAGN.NUM/MAGN.DEN
30 PROP2=1/MAGN
40 :IF(REVERSE)
50 PROP2=-:PROP2
60 :ENDIF

```

```

2090 *   ENDIF
2100 * C
2110 * C --- UPDATE CONTROLLER GAINS -----
2120 * C
2130 *   CALCULATOR
      10 DESIGN=#OFF
      20 :IF(UPDATE)
      30   PROP1=PROP2
      40   INT1=INT2
      50 :ENDIF
      60 :IF(SELECT==0)
      70   :IF(ALGO)
      80     ALGO=#OFF
      90     ALG1=#ON
     100   :ENDIF
     110 :ENDIF
     120 :IF(SELECT==1)
     130   TUNING=#OFF
     140 :ENDIF
     150 :IF(SELECT==3)
     160   :IF(ADAPTIVE)
     170     ADAPT.FLAG=#ON
     180   :ENDIF
     190   :IF(~ADAPT.FLAG)
     200     TUNING=#OFF
     210   :ENDIF
     220 :ENDIF
2140 * ENDIF
*TASK 9
  10 * C OUTPUT VALUES TO PC USING LOGGER MODULE
  20 * TIMER
      INPUT          TIMER.PULSE.
      SETPOINT          5.0000000
      RESET            RESET.TIMER.DATA
      TIME              DATA.TIMER.TIME
      OUTPUT_1          TIMER.OUTPUT.1
      OUTPUT_2          OUTPUT.SAMPLE.
  30 * IF (TIMER.OUTPUT.1)
  40 *   CALCULATOR
      10 TIMER.PULSE=#OFF
  50 * ENDIF
  60 * IF (OUTPUT.SAMPLE)
  70 *   LOGGER
      PORT              2.0000000
      MODE              #OFF..
      FORMAT            9.0000000
      LIST              9.0000000
  80 *   CALCULATOR
      10 TIMER.PULSE=#ON
  90 * ENDIF
*TASK 10
  1 * IF (SIM==0.0)
  2 * C PURE INTEGRATING PROCESS SIMULATION (SIMO)
  5 * C
 10 * C GET OUTPUT FROM THE SELF TUNING CONTROLLER
 20 * ANIN
      DEVICE              1
      INITIAL            1
      INPUT              1      Z..
      ZERO               1      Z.ZERO.

```

```

SPAN          1      Z.SPAN.
25 * C
30 * C *** BEGINNING OF PROCESS SIMULATION CODE
40 * CALCULATOR
  10 YPDOT=KP*(Z+ZL)
50 * INTEGRATOR
  INPUT          YPDOT..
  RESET          SYSINIT..
  ZERO           0.000000
  SPAN           1.000000
  OUTPUT         YP..
52 * C BOUND THE PROCESS BETWEEN ITS UPPER AND LOWER LIMITS
55 * CALCULATOR
  10 :IF(YP>YP.MAX)
  20   YP=YP.MAX
  30 :ENDIF
  40 :IF(YP<YP.MIN)
  50   YP=YP.MIN
  60 :ENDIF
60 * C UPDATE ARRAY USED TO SIMULATE DEADTIME
80 * CALCULATOR
  10 :IF(COUNT<=200)
  20   #ADATA 6(COUNT)=YP
  30   COUNT=COUNT+1
  40 :ENDIF
  50 :IF(COUNT>200)
  60   COUNT=1
  70 :ENDIF
95 * C CALCULATE DELAYED INPUT
110 * CALCULATOR
  10 K=COUNT-(1/#RATE.010*TAU.DEL)-1
  20 :IF(K<1)
  30   K=K+200
  40 :ENDIF
  50 YD=#ADATA 6(K)
  60 Y=YD+NOISE
112 * C *** END OF PROCESS SIMULATION CODE
113 * C
115 * C OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
120 * ANOUT
  DEVICE          3
  INITIAL         1
  OUTPUT          1      Y..
  ZERO           1      Y.ZERO.
  SPAN            1      Y.SPAN.
  TRACK           1      Y.TRK.
  RESET           1      Y.RES.
130 * ENDIF
*TASK 11
  5 * IF (SIM==1.0)
  10 * C FIRST ORDER PROCESS SIMULATION (SIM1)
  20 * C
  25 * C CODE TO RESET SYSTEM
  30 * CALCULATOR
  10 :IF(SYSINIT)
  20   PROC.INIT=#ON
  30   LOAD.INIT=#ON
  40 :ENDIF
40 * C
50 * C GET OUTPUT FROM THE SELF TUNING CONTROLLER

```

```

60 * ANIN
  DEVICE                1
  INITIAL               1
  INPUT      1      Z..
  ZERO       1      Z.ZERO.
  SPAN       1      Z.SPAN.
70 * C
80 * C *** BEGINNING OF PROCESS SIMULATION CODE
82 * C
83 * C CALCULATE LOAD DISTURBANCE
85 * CALCULATOR
  10 :IF(TAU.L<0.5)
  20  TAU.L=0.5
  30 :ENDIF
  40 ZLDOT=(ZL.IN-ZL)/TAU.L
87 * INTEGRATOR
  INPUT      ZLDOT..
  RESET      LOAD.INIT.
  OUTPUT     ZL..
88 * C CALCULATE NEW PROCESS VALUE
90 * CALCULATOR
  10 :IF(TAU.P<.5)
  20  TAU.P=.5
  30 :ENDIF
  40 YPDOT=(KP*(Z+ZL)-YP)/TAU.P
100 * INTEGRATOR
  INPUT      YPDOT..
  RESET      PROC.INIT.
  ZERO      0.0000000
  SPAN      1.0000000
  OUTPUT     YP..
110 * C UPDATE ARRAY USED TO SIMULATE DEADTIME
120 * CALCULATOR
  10 :IF(COUNT<=200)
  20  #ADATA 6[COUNT]=YP
  30  COUNT=COUNT+1
  40 :ENDIF
  50 :IF(COUNT>200)
  60  COUNT=1
  70 :ENDIF
130 * C CALCULATE DELAYED INPUT
140 * CALCULATOR
  10 K=COUNT-(1/#RATE.011*TAU.DEL)-1
  20 :IF(K<1)
  30  K=K+200
  40 :ENDIF
  50 YD=#ADATA 6[K]
  60 Y=YD+NOISE
150 * C *** END OF PROCESS SIMULATION CODE
160 * C
170 * C OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
180 * ANOUT
  DEVICE                3
  INITIAL               1
  OUTPUT      1      Y..
  ZERO       1      Y.ZERO.
  SPAN       1      Y.SPAN.
  TRACK      1      Y.TRK.
  RESET      1      Y.RES.
190 * ENDIF

```

```

*TASK 12
 5 * IF (SIM==2.0)
10 * C SECOND ORDER PROCESS SIMULATION (SIM2)
20 * C
25 * C CODE TO RESET SYSTEM
30 * CALCULATOR
 10 :IF(SYSINIT)
 20 PROC1.INIT=#ON
 25 PROC2.INIT=#ON
 30 LOAD.INIT=#ON
 40 :ENDIF
40 * C
50 * C GET OUTPUT FROM THE SELF TUNING CONTROLLER
60 * ANIN
  DEVICE                1
  INITIAL                1
  INPUT      1      Z..
  ZERO      1      Z.ZERO.
  SPAN      1      Z.SPAN.
70 * C
80 * C *** BEGINNING OF PROCESS SIMULATION CODE
82 * C
83 * C CALCULATE LOAD DISTURBANCE
85 * CALCULATOR
 10 :IF(TAU.L<0.5)
 20 TAU.L=0.5
 30 :ENDIF
 40 ZLDOT=(ZL.IN-ZL)/TAU.L
87 * INTEGRATOR
  INPUT      ZLDOT..
  RESET      LOAD.INIT.
  OUTPUT     ZL..
88 * C CALCULATE NEW PROCESS VALUE
90 * CALCULATOR
 10 :IF(TAU.P1<.5)
 20 TAU.P1=.5
 30 :ENDIF
 40 :IF(TAU.P2<.5)
 50 TAU.P2=.5
 60 :ENDIF
 70 X1DOT=X2
 80 X2DOT=-X1/(TAU.P1*TAU.P2)-((TAU.P1+TAU.P2)*X2/(TAU.P1*TAU.P2))+(KP*(Z@
  +ZL)/(TAU.P1*TAU.P2))
100 * INTEGRATOR
  INPUT      X1DOT..
  RESET      PROC1.INIT.
  ZERO      0.0000000
  SPAN      1.0000000
  OUTPUT     X1..
105 * INTEGRATOR
  INPUT      X2DOT..
  RESET      PROC2.INIT.
  ZERO      0.0000000
  SPAN      1.0000000
  OUTPUT     X2..
110 * C UPDATE ARRAY USED TO SIMULATE DEADTIME
120 * CALCULATOR
 10 :IF(COUNT<=200)
 20 #ADATA 6(COUNT)=X1
 30 COUNT=COUNT+1

```

```

40 :ENDIF
50 :IF(COUNT>200)
60  COUNT=1
70 :ENDIF
130 * C  CALCULATE DELAYED INPUT
140 *  CALCULATOR
110 K=COUNT-(1/#RATE.012*TAU.DEL)-1
120 :IF(K<1)
130  K=K+200
140 :ENDIF
150 YD=#ADATA 6[K]
160 Y=YD+NOISE
150 * C  ***  END OF PROCESS SIMULATION CODE
160 * C
170 * C  OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
180 *  ANOUT
      DEVICE                3
      INITIAL                1
      OUTPUT      1      Y..
      ZERO        1      Y.ZERO.
      SPAN        1      Y.SPAN.
      TRACK       1      Y.TRK.
      RESET       1      Y.RES.
190 * ENDIF
*TASK 13
5 * IF (SIM==3.0)
10 * C  BOILER PROCESS SIMULATION WITH INVERSE RESPONSE (SIM3)
20 * C
30 * C  CODE TO RESET SYSTEM
40 *  CALCULATOR
110 :IF(SYSINIT)
120  PROC1.INIT=#ON
130  PROC2.INIT=#ON
140 :ENDIF
50 * C
60 * C  GET OUTPUT FROM THE SELF TUNING CONTROLLER
70 *  ANIN
      DEVICE                1
      INITIAL                1
      INPUT      1      Z..
      ZERO        1      Z.ZERO.
      SPAN        1      Z.SPAN.
80 * C
90 * C  ***  BEGINNING OF PROCESS SIMULATION CODE
95 * C  CALCULATE LIQUID LEVEL WRT FEEDWATER
100 *  CALCULATOR
110 Y2DOT=KP2*(Z+ZL)
110 *  INTEGRATOR
      INPUT      Y2DOT..
      RESET      PROC2.INIT.
      ZERO        0.0000000
      SPAN        1.0000000
      OUTPUT      Y2..
120 * C  BOUND THE FEEDWATER PROCESS BETWEEN ITS UPPER AND LOWER LIMITS
130 *  CALCULATOR
140 :IF(Y2>Y2.MAX)
150  Y2=Y2.MAX
160 :ENDIF
170 :IF(Y2<Y2.MIN)
180  Y2=Y2.MIN

```

```

60 :ENDIF
135 * C CALCULATE LIQUID LEVEL WRT HEAT SUPPLY
140 * CALCULATOR
10 :IF(TAU.P1<.5)
20 TAU.P1=.5
30 :ENDIF
40 Y1DOT=((KPI*(Z+ZL))-Y1)/TAU.P1
150 * INTEGRATOR
INPUT Y1DOT..
RESET PROC1.INIT.
ZERO 0.000000
SPAN 1.000000
OUTPUT Y1..
160 * C CALCULATE OVERALL RESPONSE & UPDATE ARRAY USED TO SIMULATE DEADTIME
170 * CALCULATOR
5 YP=Y2-Y1
10 :IF(COUNT<=200)
20 #ADATA 6(COUNT)=YP
30 COUNT=COUNT+1
40 :ENDIF
50 :IF(COUNT>200)
60 COUNT=1
70 :ENDIF
180 * C CALCULATE DELAYED INPUT
190 * CALCULATOR
10 K=COUNT-(1/#RATE.013*TAU.DEL)-1
20 :IF(K<1)
30 K=K+200
40 :ENDIF
50 YD=#ADATA 6(K)
60 Y=YD+NOISE
200 * C *** END OF PROCESS SIMULATION CODE
210 * C
220 * C OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
230 * ANOUT
DEVICE 3
INITIAL 1
OUTPUT 1 Y..
ZERO 1 Y.ZERO.
SPAN 1 Y.SPAN.
TRACK 1 Y.TRK.
RESET 1 Y.RES.
240 * ENDIF
*TASK 14
5 * IF (SIM==4.0)
10 * C PROCESS SIMULATION WITH INVERSE RESPONSE (SIM4)
20 * C
30 * C CODE TO RESET SYSTEM
40 * CALCULATOR
10 :IF(SYSINIT)
20 PROC1.INIT=#ON
30 PROC2.INIT=#ON
40 :ENDIF
50 * C
60 * C GET OUTPUT FROM THE SELF TUNING CONTROLLER
70 * ANIN
DEVICE 1
INITIAL 1
INPUT 1 Z..
ZERO 1 Z.ZERO.

```

```

SPAN      1      Z.SPAN.
80 * C
90 * C *** BEGINNING OF PROCESS SIMULATION CODE
100 * C CALCULATE SLOWER PROCESS WITH DIRECT-ACTING RESPONSE
110 * CALCULATOR
    10 :IF(TAU.P1<.5)
    20   TAU.P1=.5
    30 :ENDIF
    40 Y1DOT=((KP1*(Z+ZL))-Y1)/TAU.P1
120 * INTEGRATOR
    INPUT      Y1DOT..
    RESET      PROC1.INIT.
    ZERO              0.0000000
    SPAN            1.0000000
    OUTPUT      Y1..
130 * C CALCULATE FASTER PROCESS WITH INVERSE RESPONSE
140 * CALCULATOR
    10 :IF(TAU.P2<.5)
    20   TAU.P2=.5
    30 :ENDIF
    40 Y2DOT=((KP2*(Z+ZL))-Y2)/TAU.P2
150 * INTEGRATOR
    INPUT      Y2DOT..
    RESET      PROC2.INIT.
    ZERO              0.0000000
    SPAN            1.0000000
    OUTPUT      Y2..
160 * C CALCULATE OVERALL RESPONSE & UPDATE ARRAY USED TO SIMULATE DEADTIME
170 * CALCULATOR
    5 YP=Y1-Y2
    10 :IF(COUNT<=200)
    20   #ADATA 6(COUNT)=YP
    30   COUNT=COUNT+1
    40 :ENDIF
    50 :IF(COUNT>200)
    60   COUNT=1
    70 :ENDIF
180 * C CALCULATE DELAYED INPUT
190 * CALCULATOR
    10 K=COUNT-(1/#RATE.014*TAU.DEL)-1
    20 :IF(K<1)
    30   K=K+200
    40 :ENDIF
    50 YD=#ADATA 6(K)
    60 Y=YD+NOISE
200 * C *** END OF PROCESS SIMULATION CODE
210 * C
220 * C OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
230 * ANOUT
    DEVICE      3
    INITIAL     1
    OUTPUT      1      Y..
    ZERO        1      Y.ZERO.
    SPAN        1      Y.SPAN.
    TRACK       1      Y.TRK.
    RESET       1      Y.RES.
240 * ENDIF
*TASK 15
  5 * IF (SIM==5.0)
  10 * C SYSTEM WITH VARIABLE TIME CONSTANT AND DELAY (SIM5)

```

```

20 * C
30 * C CODE TO RESET SYSTEM
40 * CALCULATOR
  10 :IF(SYSINIT)
  20 PROC.INIT=#ON
  30 LOAD.INIT=#ON
  40 :ENDIF
50 * C
60 * C GET OUTPUT FROM THE SELF TUNING CONTROLLER
70 * ANIN
  DEVICE                               1
  INITIAL                               1
  INPUT      1      Z..
  ZERO       1      Z.ZERO.
  SPAN       1      Z.SPAN.
80 * C
90 * C *** BEGINNING OF PROCESS SIMULATION CODE
100 * CALCULATOR
  10 Q2=(Z+ZL)
  20 CIN=((SIM.C1*Q1)+(SIM.C2*Q2))/(Q1+Q2)
110 * C UPDATE ARRAY USED TO SIMULATE DEADTIME
120 * CALCULATOR
  10 :IF(COUNT<=200)
  20 #ADATA 6[COUNT]=CIN
  30 COUNT=COUNT+1
  40 :ENDIF
  50 :IF(COUNT>200)
  60 COUNT=1
  70 :ENDIF
130 * C CALCULATE DELAYED INPUT CONCENTRATION
140 * CALCULATOR
  5 TAU.DEL=VD/(Q1+Q2)
  10 K=COUNT-(1/#RATE.015*TAU.DEL)-1
  20 :IF(K<1)
  30 K=K+200
  40 :ENDIF
  50 CIN.PRIME=#ADATA 6[K]
150 * C CALCULATE NEW CONCENTRATION
160 * CALCULATOR
  10 TAU.P=VM/(Q1+Q2)
  20 CDOT=((KP*CIN.PRIME)-SIM.C)/TAU.P
170 * INTEGRATOR
  INPUT      CDOT..
  RESET      PROC.INIT.
  ZERO              0.000000
  SPAN              1.000000
  OUTPUT          SIM.C.
180 * CALCULATOR Y=SIM.C+NOISE
190 * C *** END OF PROCESS SIMULATION CODE
200 * C
210 * C OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
220 * ANOUT
  DEVICE                               3
  INITIAL                               1
  OUTPUT      1      Y..
  ZERO       1      Y.ZERO.
  SPAN       1      Y.SPAN.
  TRACK      1      Y.TRK.
  RESET      1      Y.RES.
240 * ENDIF

```

```

*TASK 16
 5 * IF (SIM==6.0)
10 * C SECOND ORDER PROCESS SIMULATION ALLOWING COMPLEX POLES (SIM6)
20 * C
25 * C CODE TO RESET SYSTEM
30 * CALCULATOR
 10 :IF(SYSINIT)
 20 PROC1.INIT=#ON
 25 PROC2.INIT=#ON
 30 LOAD.INIT=#ON
 40 :ENDIF
40 * C
50 * C GET OUTPUT FROM THE SELF TUNING CONTROLLER
60 * ANIN
  DEVICE                1
  INITIAL                1
  INPUT      1      Z..
  ZERO       1      Z.ZERO.
  SPAN       1      Z.SPAN.
70 * C
80 * C *** BEGINNING OF PROCESS SIMULATION CODE
82 * C
83 * C CALCULATE LOAD DISTURBANCE
85 * CALCULATOR
 10 :IF(TAU.L<0.5)
 20 TAU.L=0.5
 30 :ENDIF
 40 ZLDOT=(ZL.IN-ZL)/TAU.L
87 * INTEGRATOR
  INPUT      ZLDOT..
  RESET      LOAD.INIT.
  OUTPUT     ZL..
88 * C CALCULATE NEW PROCESS VALUE
90 * CALCULATOR
 10 X1DOT=X2
 20 X2DOT=-((WN**2)*X1)-(2.*ZETA.SIM*WN*X2)+(KP*(WN**2)*(Z+ZL))
100 * INTEGRATOR
  INPUT      X1DOT..
  RESET      PROC1.INIT.
  ZERO              0.0000000
  SPAN              1.0000000
  OUTPUT     X1..
105 * INTEGRATOR
  INPUT      X2DOT..
  RESET      PROC2.INIT.
  ZERO              0.0000000
  SPAN              1.0000000
  OUTPUT     X2..
110 * C UPDATE ARRAY USED TO SIMULATE DEADTIME
120 * CALCULATOR
 10 :IF(COUNT<=200)
 20 #ADATA 6{COUNT}=X1
 30 COUNT=COUNT+1
 40 :ENDIF
 50 :IF(COUNT>200)
 60 COUNT=1
 70 :ENDIF
130 * C CALCULATE DELAYED INPUT
140 * CALCULATOR
 10 K=COUNT-(1/#RATE.016*TAU.DEL)-1

```

```

20 :IF(K<1)
30 K=K+200
40 :ENDIF
50 YD=#ADATA 6[K]
60 Y=YD+NOISE
150 * C *** END OF PROCESS SIMULATION CODE
160 * C
170 * C OUTPUT PROCESS VALUE TO SELF TUNING CONTROLLER
180 * ANOUT
      DEVICE                3
      INITIAL                1
      OUTPUT      1      Y..
      ZERO        1      Y.ZERO.
      SPAN        1      Y.SPAN.
      TRACK       1      Y.TRK.
      RESET       1      Y.RES.
190 * ENDIF
*LIST 1
  10 ENABLE..
  20 SELECT..
  30 REVERSE..
  40 OVERSH..
  50 SETPOINT..
  60 INPUT..
  70 OPMAX..
  80 OUTPUT2..
  90 OPMIN..
 100 PROP2..
 110 INT2..
 120 STATUS..
 130 DONE..
*LIST 2
  10 PVDEV..
  20 RELAY..
  30 INTEG..
  40 ACCEPT..
  50 HYSTER..
  60 PVAMP..
  70 PERIOD..
  80 THRESH..
*LIST 3
  10 OPMEAN..
  20 OPDEV..
  30 TOTAL..
  40 ALPHA..
  50 LAMBDA..
  60 PERR..
  70 ADAPTIVE..
*LIST 4
  10 A1..
  20 DIAG1..
  30 B1..
  40 DIAG2..
  50 B2..
  60 DIAG3..
  70 B3..
  80 DIAG4..
  90 B4..
 100 DIAG5..
 110 B5..

```

```
120 DIAG6..
130 C1..
140 DIAG7..
*LIST 9
 10 Z..
 15 ZL..
 17 NOISE..
 20 Y..
 30 #TIME.007.
*A-ARRAY 1 RW ( 7, 1 )
*A-ARRAY 2 RW ( 7, 1 )
*A-ARRAY 3 RW ( 7, 1 )
*A-ARRAY 4 RW ( 21, 1 )
*A-ARRAY 5 RW ( 7, 1 )
*A-ARRAY 6 RW (200, 1 )
*A-ARRAY 10 RW ( 15, 4 )
*FORMAT 9
 10 F5.2,1X,F5.2,1X,F5.2,1X,F5.2,1X,I2, /
```

**Appendix C**

**MATLAB PERFORMANCE ANALYSIS AND PLOTTING ROUTINES**



## APPENDIX C

```
% MODIFY.M

% -----
% Before using this program, run an external QB program to strip header
% and timestamp info from the GENESIS testXXX.prn output file, using the
% !striphdr command.

% Then execute the following commands in MATLAB.
% -----

% Ask user which test file data to use load the file to matrix testXXX

num=input('Enter Test No. to modify : ','s');
testnum=['test',num];
filename=[testnum,'.mod'];
eval(['load ',filename])

% Transpose the testXXX matrix and convert it to a long vector

testx=eval(['test',num]);
testv=testx(:);

% Find out how many complete data records there are

k=fix(length(testv)/5);

% Put data into the appropriate vectors

for j=0:(k-1)
% th(j+1)=testv(j*8+1);
% tm(j+1)=testv(j*8+2);
% ts(j+1)=testv(j*8+3);
  sp(j+1)=testv(j*5+1);
  ip(j+1)=testv(j*5+2);
  op(j+1)=testv(j*5+3);
  p(j+1)=testv(j*5+4);
  i(j+1)=testv(j*5+5);
end

% Create 'proc' and 'cont' arrays needed for plotting

procp=[sp;ip;op];
proc=procp';
contp=[p;i];
cont=contp';
```

```
% Display the data
```

```
datap=[ip;sp;op;p;i];  
data=datap';
```

```
% DEFDATA.M
```

```
% -----  
% MATLAB routine to define what data will be used for  
% plotting (or as input to the evaluation criteria  
% -----
```

```
start=input('Enter First Sample Number for Plotting: ','s');
```

```
last=input(' Enter Last Sample Number for Plotting: ','s');
```

```
datax=data(eval(start):eval(last),:);
```

```
% EVALCRIT.M

% -----
% MATLAB routine to evaluate the performance of the tuned
% control systems using the IAE, ISE, and ITAE performance
% criterion
% -----

start1=input('Enter Criteria Starting Sample Number: ','s');

last1=input(' Enter Criteria Ending Sample Number: ','s');

datay=data(eval(start1):eval(last1),:);

n=fix(length(datay));

iae=0;
ise=0;
itae=0;

for j=1:n
    iae = iae + abs(datay(j,1) - datay(j,2));
    ise = ise + (datay(j,1) - datay(j,2))^2;
    itae= itae+ j * abs(datay(j,1) - datay(j,2));
end
```

```

% LABEL.M

% -----
% MATLAB routine to plot and label measurement vs.
% setpoint, controller output, and controller PI
% parameter plots (without performance criterion)
% -----

eval(['delete ',testnum,'.met'])

clg
plot(datax(:,1:2))
title(['Measurement vs Setpoint (',testnum,')'])
xlabel('Sample Number')
ylabel('Percent')
% text(100,90,[' IAE = ',num2str(iae)])
% text(100,85,[' ISE = ',num2str(ise)])
% text(100,80,['ITAE = ',num2str(itae)])

eval(['meta ',testnum])
pause
clg

plot(datax(:,3))
title(['Controller Output'])
xlabel('Sample Number')
ylabel('Percent')

meta
pause
clg

axis

plot(datax(:,4:5))
title(['Controller PI Parameters'])
xlabel('Sample Number')
ylabel('Gain and Repeats/Minute')

meta

```

```

% LABELCRIM

% -----
% MATLAB routine to plot and label measurement vs.
% setpoint, controller output, and controller PI
% parameter plots (with performance values)
% -----

eval(['delete ',testnum,'.met'])

clg
plot(datax(:,1:2))
title(['Measurement vs Setpoint (',testnum,')'])
xlabel('Sample Number')
ylabel('Percent')
text(.72,.85,[' IAE'], 'sc')
text(.82,.85,[num2str(iae)], 'sc')
text(.79,.85,['='], 'sc')
text(.72,.8,[' ISE'], 'sc')
text(.82,.8,[num2str(ise)], 'sc')
text(.79,.8,['='], 'sc')
text(.72,.75,['ITAE'], 'sc')
text(.82,.75,[num2str(itae)], 'sc')
text(.79,.75,['='], 'sc')

eval(['meta ',testnum])
pause
clg

plot(datax(:,3))
title(['Controller Output'])
xlabel('Sample Number')
ylabel('Percent')

meta
pause
clg

plot(datax(:,4:5))
title(['Controller PI Parameters'])
xlabel('Sample Number')
ylabel('Gain and Repeats/Minute')

meta

```

```

' STRIPHDR.BAS
' -----
' Microsoft QuickBASIC ver 4.0 program designed to:
' (1) input a GENESIS testXXX.prn data file,
' (2) strip the header and timestamp information, and
' (3) output the data to a testXXX.mod file
' -----

INPUT "Test File to modify: ", FileName1$
INPUT "Name of Output file: ", FileName2$

OPEN FileName1$ FOR INPUT AS #1
OPEN FileName2$ FOR OUTPUT AS #2
IF FileName1$ = "" THEN END

CONST QUOTE = 34, COLON = 58

' Skip the header info
FOR I = 1 TO 7
    LINE INPUT #1, LineBuffer$
NEXT

' Keep modifying as long as there are enough bytes left in
' the file.

DO UNTIL EOF(1)

    Character$ = INPUT$(1, #1)
    CharVal = ASC(Character$)

    SELECT CASE CharVal
        CASE QUOTE
            Character$ = INPUT$(1, #1)
            CharVal = ASC(Character$)
            DO UNTIL CharVal = QUOTE
                Character$ = INPUT$(1, #1)
                CharVal = ASC(Character$)
            LOOP
        CASE ELSE
            PRINT #2, Character$;
    END SELECT

LOOP

CLOSE #1

CLOSE #2

```

**Appendix D**

**MATHCAD ROBUSTNESS ANALYSIS ROUTINES**



## APPENDIX D

This file is used to determine the robustness of a control system by indicating the amount the plant gain and deadtime may be increased before the onset of instability. The algorithm simply finds the gain factor K and deadtime that yields zero phase margin. File Robust1.mcd

First, describe the known plant and controller in the s domain

$$K_p := 1 \quad \tau := 5 \quad T := 20 \quad K_c := .7103 \quad K_i := \frac{2.147}{60}$$

Plant WO Deadtime

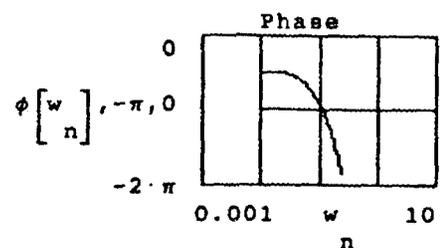
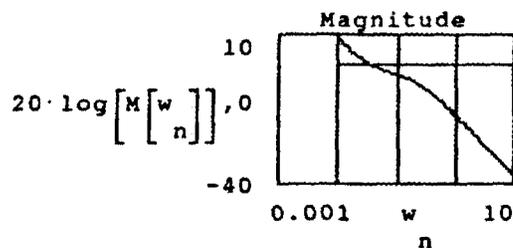
$$G_p(s) := \frac{K_p}{\tau \cdot s + 1}$$

Controller

$$G_c(s) := K_c \cdot \left[ 1 + \frac{K_i}{s} \right]$$

Phase  $\phi(\omega) := \arg[G_p(j \cdot \omega)] + \arg[G_c(j \cdot \omega)] - \omega \cdot T$

Magnitude  $M(\omega) := \left| G_p(j \cdot \omega) \cdot G_c(j \cdot \omega) \right|$



A linear search is used to find the frequency where phase=-180 degrees

Supply an initial guess  $\omega := .5$   
Then find the frequency where phase margin is zero

Given

$$\phi(\omega) = -\pi$$

$$\omega := \text{Find}(\omega)$$

$$\omega = 0.116$$

Now compute the gain limit that causes zero phase margin

$$K_1 := \frac{1}{M\left[\begin{array}{c} \omega \\ z \end{array}\right]}$$

$$K_1 = 1.555$$

The deadtime limit can also be found. We first find the zero db crossing frequency

Supply a guess of zero db crossing frequency  $\omega := .001$

Given

$$M(\omega) = 1$$

$$\omega := \text{Find}(\omega)$$

$$\omega = 0.035$$

The deadtime limit can then be computed

$$T_1 := \frac{\pi + \phi\left[\begin{array}{c} \omega \\ z \end{array}\right]}{\omega}$$

$$T_1 = 41.994$$

This file is used to determine the robustness of a control system by indicating the amount the plant gain and deadtime may be increased before the onset of instability. The algorithm simply finds the gain factor  $K$  and deadtime that yields zero phase margin. File Robust.mcd

First, describe the known plant and controller in the  $s$  domain

$$w_n := .2 \quad \lambda := 2.5 \quad K_c := 3.093 \quad K_i := \frac{2.214}{60}$$

Plant WO Deadtime

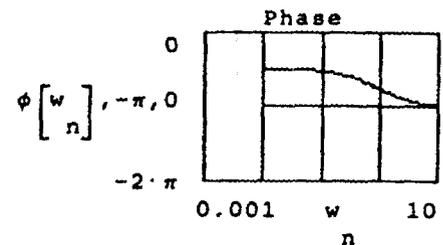
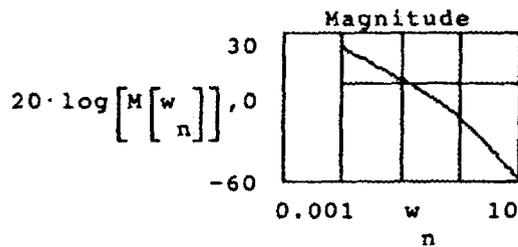
$$G_p(s) := \frac{w_n^2}{s^2 + 2 \cdot \lambda \cdot w_n \cdot s + w_n^2}$$

Controller

$$G_c(s) := K_c \cdot \left[ 1 + \frac{K_i}{s} \right]$$

$$\text{Phase} \quad \phi(\omega) := \arg[G_p(j \cdot \omega)] + \arg[G_c(j \cdot \omega)]$$

$$\text{Magnitude} \quad M(\omega) := \left| G_p(j \cdot \omega) \cdot G_c(j \cdot \omega) \right|$$



Supply an initial guess  $\omega := .5$   
Then find the frequency where phase margin is zero

Given

$$\phi(\omega) = -\pi$$

$$\omega := \text{Find}(\omega)$$

$$\omega = 2.742 \cdot 10^8$$

Now compute the gain limit that causes zero phase margin

$$K_1 := \frac{1}{M\left[\begin{array}{c} \omega \\ z \end{array}\right]}$$

$$K_1 = 6.078 \cdot 10^{17}$$

The deadtime limit can also be found. We first find the zero db crossing frequency

Supply a guess of zero db crossing frequency  $\omega := .001$

Given

$$M(\omega) = 1$$

$$\omega := \text{Find}(\omega)$$

$$\omega = 0.127$$

The deadtime limit can then be computed

$$T_1 := \frac{\pi + \phi\left[\begin{array}{c} \omega \\ z \end{array}\right]}{\omega}$$

$$T_1 = 11.644$$

## INTERNAL DISTRIBUTION

- |                    |                                      |
|--------------------|--------------------------------------|
| 1. H. R. Brashear  | 14. J. O. Stiegler                   |
| 2. C. L. Carnal    | 15. P. C. Turner                     |
| 3. B. C. Duggins   | 16-20. P. A. Tapp                    |
| 4. B. G. Eads      | 21. R. E. Uhrig                      |
| 5. D. N. Fry       | 22. B. Chexal, Advisor               |
| 6. J. M. Googe     | 23. V. Radeka, Advisor               |
| 7. S. S. Gould     | 24. R. M. Taylor, Advisor            |
| 8. S. E. Groothuis | 25-26. Central Research Library      |
| 9. D. W. McDonald  | 27. Y-12 Technical Reference Section |
| 10. D. K. Mee      | 28-29. Laboratory Records Dept.      |
| 11. D. R. Miller   | 30. Laboratory Records ORNL-RC       |
| 12. R. E. Neal     | 31. ORNL Patent Section              |
| 13. T. E. Rowe     | 32. I&C Publications Office          |

## EXTERNAL DISTRIBUTION

33. Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Field Office, P.O. Box 2001, Oak Ridge, TN 37831-8600.
- 34-35. Office of Scientific and Technical Information, U.S. Department of Energy, P.O. Box 62, Oak Ridge, TN 37831
36. C. S. Cox, Control Systems Centre, School of Electrical Engineering and Applied Physics, Sunderland Polytechnic, Sunderland, England, SR1 3SD
37. G. J. Hill, Bristol-Babcock Ltd., Vale Industrial Estate, Stourport Road, Kidderminster, DY11 7QP, Worcestershire, England
38. Jim Pettit, Bristol-Babcock, Inc., 1100 Buckingham St., Watertown, CT 06795