

SELF-TUNING PID CONTROLLERS

A Thesis Submitted to the

Graduate School of Natural and Applied Sciences of

Dokuz Eylül University

In Partial Fulfillment of the Requirements for

**the Degree of Master of Science in Electronics Engineering, Electronics and
Communication Program**

by

Caner AYGÜN

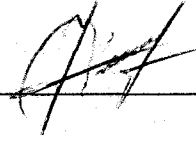
February, 1997

İZMİR

65352

M.Sc THESIS EXAMINATION RESULT FORM

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Dr. Yavuz Şenol

(Advisor)



Prof. Dr. Haldun Karaca

(Committee Member)



Dr. E. UNGÖR

(Committee Member)

Approved by the

Graduate School of Natural and Applied Sciences



Prof. Dr. Macit Toksoy

Director

ELAZIĞ İLİ MİLLÎ EĞİTİM BAKANLIĞI KURULU
DEĞERLENDİRME MERKEZİ

ACKNOWLEDGMENTS

The author wishes to express his sincere thanks and appreciation to Dr. Yavuz Şenol for his attention, guidance, insight, and support during this research and the preparation of this thesis. In addition, special thanks to Dr. Erginer Ugan for his constructive comments, suggestions and for providing valuable trace data to create many of the displays.

The author also acknowledges the generous technical support from Enko Limited Company and thanks to H. Sinan Kazazoğlu for his valuable recommendations.



ABSTRACT

PID controllers are widely used in many industrial applications. Although there are several conventional PID controller tuning techniques, these techniques are not appropriate for some systems such as time variant or remote located systems. Conventional tuning procedures consider systems with fixed and constant coefficients. Conventional control system design procedure is based on linear system approach. However, some applications whose design criteria can change in large ranges need nonlinear approach. Thus, if the system is time variant and changes are in large range, conventional PID controllers can not work properly.

A self-tuning system does not need to be tuned by the system engineer. Optimal tuning point can be obtained by the self-tuning mechanism for different conditions of time variant systems to work on the desired closed-loop performance.

In this thesis, development and performance of a self-tuning PID controller has been discussed. The realized system is based on recursive least squares (RLS) parameter estimation method and pole-placement technique for controller design process.

CONTENTS

	Page
Contents	III
List of Tables	VI
List of Figures	VII

Chapter One

INTRODUCTION

1. Introduction.....	1
1.1 Control Systems	1
1.2 Conventional Control System Approaches.....	2
1.3 Three-term (PID) Controllers	2
1.4 Self-Tuning Systems	3
1.5 Self-Tuning PID Controllers	4

Chapter Two

THREE TERM (PID) CONTROLLERS

2. Three-term (PID) Controllers	5
2.1 Conventional Feedback Control Structures	5
2.2 Proportional Control Structure	6
2.3 Root Locus Analysis	7
2.3.1 Variation of Closed-Loop System Poles.....	7
2.3.2 Real Axis Loci	8
2.3.3 Asymptotes	8
2.3.4 Breakaway Points	9
2.3.5 Departure and Arrival Angles	9
2.3.6 Construction of the Root-Locus	9
2.4 Integral Control Structure	10
2.4.1 Stability Analysis of First Order Integral Control Systems	11
2.4.2 Stability Analysis of Second Order Integral Control Systems	12
2.5 Derivative Control System Structure	12
2.6 Proportional + Integral (PI) Control Structure	13
2.7 Proportional + Integral + Derivative (PID) Control Structure	14

Chapter Three

SELF-TUNING SYSTEMS

3. Self-tuning Systems	17
3.1 Introduction	17
3.2 Self-Tuning and Adaptive Control Approaches	17
3.3 Self-Tuning System Structure	18
3.4 System Models	20
3.5 Least Squares (LS) Method	22
3.6 Recursive Least Squares (RLS)	24
3.6.1 Recursive Least Squares Algorithm	24
3.6.2 Initializing The Estimator	25
3.7 Pole Assignment Control	25
3.7.1 Controller Design by Pole Assignment for First Order Systems	25
3.7.2 Three-Term Controller Design by Pole Assignment for Second Order Systems.....	26
3.8 Self Tuning PID Controller Design Algorithm	28

Chapter Four

A SELF-TUNING PID CONTROLLER APPLICATION

4. A Self-tuning PID Controller Application	29
4.1 Introduction	29
4.2 Mathematical Analysis of a Serial RLC Circuit	29
4.3 Application	32

Chapter Five

HARDWARE AND SOFTWARE OF THE APPLICATION

5. Hardware and Software of the Application	34
5.1 Hardware Structure of the System.....	34
5.2 The Mathematical Model of the Plant	35
5.3 Software Structure of the System	37

Chapter Six

APPLICATION RESULTS

6. Application Results	40
6.1 Introduction	40
6.2 Recursive Estimation Results	40
6.3 Self-tuning PID Implementation Results	44

Chapter Seven

CONCLUSIONS

7. Conclusions.....	56
---------------------	----

REFERENCES

References	58
------------------	----

APPENDICES

1. Technical Product Information for the DAP 800	1
2. Recursive Estimation and Pole-assignment Control Software in C Language	6
3. Recursive Estimation Procedure for DAP800	13
4. PID Control Procedure for DAP800	14

LIST OF TABLES

		Page
Table 2.1	The effects of K_p , K_i and K_d on the closed-loop response.	16
Table 6.1	The plant and desired overall system specifications of the measured systems.	44



LIST OF FIGURES

		Page
Figure 2.1	Block diagram for output feedback control	5
Figure 2.2	Block diagram of proportional control	6
Figure 2.3	Simple closed-loop system	7
Figure 2.4	Root-locus plot of Eq.2.17	10
Figure 2.5	Root-locus for integral control of a first order system	11
Figure 2.6	Root-locus for integral control of a second order system	12
Figure 2.7	Block diagram of a PI control system.	13
Figure 2.8	Block diagram of a PID control system.	14
Figure 2.9	System response to a step input	15
Figure 3.1	Control system design steps	19
Figure 3.2	Self-tuning controller structure	20
Figure 3.3	Discrete signal and its continuous time equivalent	21
Figure 3.4	Discrete time system model	21
Figure 3.5	A system with all possible input and disturbance components	22
Figure 3.6	A closed-loop first order system	26
Figure 4.1	Serial RLC circuit	30
Figure 4.2	Parameter estimation step of the self-tuner	32
Figure 4.3	Implementation of PID control	33
Figure 4.4	Flow chart of the self-tuning application	33
Figure 5.1	Self-tuning system application	34

Figure 5.2	The plant which is controlled in the application	35
Figure 5.3a	Unit step response of the plant while $C=1.5\mu F$	36
Figure 5.3b	Unit step response of the plant while $C=15\mu F$	36
Figure 5.4	Flow chart of the recursive estimator software	37
Figure 5.5	Flow chart of PID implementation software	39
Figure 6.1a	Test signal(plant input) and plant output while $C=1.5\mu F$	41
Figure 6.1b	Test signal(plant input) and plant output while $C=15\mu F$	41
Figure 6.2a	Parameter estimation for $C=1.5\mu F$	42
Figure 6.2b	Parameter estimation for $C=15\mu F$	42
Figure 6.3a	Calculated step response of the estimated transfer function for $C=1.5\mu F$	43
Figure 6.3b	Calculated step response of the estimated transfer function for $C=15\mu F$	43
Figure 6.4	Step response of the RLC circuit while $C=15\mu F$, $\xi=0.1$, $\omega_n=10$	45
Figure 6.5	Step response of the RLC circuit while $C=15\mu F$, $\xi=0.8$, $\omega_n=10$	46
Figure 6.6	Step response of the RLC circuit while $C=15\mu F$, $\xi=0.1$, $\omega_n=3$	47
Figure 6.7	Step response of the RLC circuit while $C=15\mu F$, $\xi=0.8$, $\omega_n=3$	48
Figure 6.8	Step response of the RLC circuit while identified and controlled plants are different. ($C_{identified} = 15\mu F$, $C_{controlled} = 1.5\mu F$)	49
Figure 6.9	Step response of the RLC circuit while $C=1.5\mu F$, $\xi=0.1$, $\omega_n=10$	50
Figure 6.10	Step response of the RLC circuit while $C=1.5\mu F$, $\xi=0.8$, $\omega_n=10$	51
Figure 6.11	Step response of the RLC circuit while $C=1.5\mu F$, $\xi=0.1$, $\omega_n=3$	52
Figure 6.12	Step response of the RLC circuit while $C=1.5\mu F$, $\xi=0.8$, $\omega_n=3$	53
Figure 6.13	Step response of the RLC circuit while identified and controlled plants are different. ($C_{identified} = 1.5\mu F$, $C_{controlled} = 15\mu F$)	54

CHAPTER ONE

INTRODUCTION

1.1 Control Systems

Control system engineers are interested in understanding and controlling of their environment. The aim of control engineering is to provide useful and economic products and systems for the advantage of mankind. The systems to be controlled must be understood and modeled by system designers to obtain a more effective control structure and preferred performance. However, since most systems are extremely complex to understand and difficult to modeling, control engineers frequently must think about control of poorly defined systems (DORF, 1986, p.2).

Control engineering is established on the basis of feedback theory and linear system analysis. A control system is an interconnection of system parts that will provide a desired system response. Each parts of system can be represented by a block and input-output relations of blocks represent the cause and effect relationship of the process.

An open-loop control system utilizes a controller or control actuator in order to obtain the desired response without feedback. Therefore, the controller, the control actuator and other environmental conditions must be defined perfectly because, control system can not measure and compare the actual output of the system with respect to desired output. Nevertheless, most control systems are not suitable to obtain certain relations between inputs and outputs. Hence, open-loop systems are not appropriate for poorly defined systems.

In contrast to open-loop systems, closed-loop systems utilize feedback in order to compare actual output with the desired output. Feedback is an additional measure of actual output and used to calculate difference between actual and desired system outputs. "A feedback control system is a control system that tends to maintain a prescribed relationship of one system variable to another by comparing functions of these variables and using the difference as a mean of control (DORF, 1986, p.2)."

The aim of the closed-loop control system is to ensure that the actual output follows in some way the desired output and rejects the effect of the unpredictable inputs (disturbances) which corrupts the system output. A closed-loop control system consists of a precompensator that is used to shape the transfer function between the system input and

output, and a controller that is used to determine the closed loop stability, disturbance rejection and sensitivity characteristics of control system .

1.2 Conventional Control System Approaches

The conventional control system approach considers systems with fixed and constant coefficients. This assumption of time-invariance is essential to conventional design procedures. Therefore, control engineer must design a robust control system which can work properly in all of possible operating conditions. Since, the system which will be controlled must be known by designer and undetermined conditions must be restricted against to unrestrained system behaviors.

Conventional control system design procedure is constructed on linear system theory. Linear system approach is generally a tolerated approximation because a complete nonlinear theory does not exist and linear solutions are sufficient to most nonlinear control system design problems. However, some applications whose design criteria can change in large ranges need nonlinear approach. In this case, constant control coefficients are determined as capable to conform variable system conditions even though general performance loss (Wellstead & Zarrop, 1991, pp 1-4).

1.3 Three-term (PID) Controllers

One form of controller which is widely used in industrial process control is called three-term (PID) controller. Three-term controllers include a proportional, an integral and a derivative control terms. Therefore, the three-term controller is also called a PID controller. PID controllers are widely used in industrial applications because, they perform satisfyingly in spite of their functional simplicity. Conventional PID controllers are designed on the basis of local linearization about an operating point. These controllers are very effective if the load changes are small and the operating conditions do not force the system too far away from the linearizing balance point.

The terms of PID controller can be used one by one or any combination of two terms. Hence, P, I, D, PI, PD, ID and PID control concepts are available. However, some concepts which consist a term or a combination of two terms of PID, as D, ID and I are not suitable to apply to most applications. Furthermore, P, PI and PD control concepts are well-known controllers.

To implement a PID controller, proportional, integral and derivative gain parameters must be determined for the given process. The selection of the coefficients of PID controllers is a search problem in a three-dimensional space. Different coefficient selections cause different responses, for example, different step responses for a step input. The

coefficients of a PID controller can be determined by moving in the search space according to some experimental methods or trial and error basis. There are several rules and methods to solve this tuning problem. "Conventionally, a PID controller is tuned manually using one of the two procedures described by Zeigler and Nichols (Hang & Sin, 1991, p428)." However, system tuning issue is a main problem in control engineering (Dorf, pp573-615).

1.4 Self-Tuning Systems

Conventional control system design procedure needs to know about system behavior and system characteristics. Additionally, constant coefficients must be suitable for all determined operating conditions. However, most systems are not suitable to determine system characteristics for all conditions because, these systems include undetermined parts and these parts have some difficulties to find their characteristics. For that reason, tuning and calibration is necessary to most conventional control systems.

Second handicap of conventional control systems is that, most systems have time-variant parameters and constant coefficients are not sufficient to obtain desired system response while parameters change in large ranges.

"The basic idea of a self-tuning system is to construct an algorithm that will automatically change its parameters to meet a particular requirement or situation (Wellstead *et al*, 1991, p.4)." The self-tuning concept includes two different adjustment mechanism approaches: The first approach is based on initial auto-tuning of system parameters which are time-invariant but not desired constantly by the designer. Thus, after initial tuning, the adjustment mechanism is not required and can be disabled until another tuning process is requested by a control mechanism or system operator. The second auto-adjustment method is based on continuous adjustment. In practice, both ideas are based on nearly same basis except small algorithmic details. However, initial-adjusting systems are called *self-tuning* and continuous-adjusting systems are called *adaptive* in the terminology (Wellstead *et al*, 1991, pp 4-6).

The self-tuning controller concept for obtaining an automatic adjustment mechanism is to identify the system using measured input and output data and then, to form a suitable controller using the identified system. This concept can be divided into two main processes: To identify the system and to form an appropriate controller using the identified system.

Implementations of self-tuning systems in practical applications were started after low cost digital computer equipment had developed. However, the self-tuning system idea is older than low cost digital computers. For example:

Kalman described a self-tuning optimizing controller in 1958, but the algorithm was impractical at the time due to digital computer limitations of cost, speed and size. The current interest in self-tuning control was first stimulated by the Czech researcher Peterka who showed how system identification and controller synthesis could be combined into one iterative procedure for process control.(...) In the 1970s the results of this impetus became evident with the publication of several key papers presenting self-tuning controllers for various design criteria (WELLSTEAD & ZARROP, 1991, p.10).

Nowadays, self-tuning control systems are used in many applications especially industrial processes.

1.5 Self-Tuning PID Controllers

PID controllers are widely used components of industrial applications. These controllers have very simple structure and their performances are as pleasant as their simplicity. However, using of PID controllers are restricted by some applicational difficulties:

- PID controllers need initial tuning. Even though there are several effective theoretical based tuning techniques, in most applications, these techniques are not feasible because of the practical reasons. In most simple PID controller applications, trial and error methods can be used successfully. However, tuning of some complex systems which have difficulties about estimating the system characteristics is difficult and hazardous. Because, these systems may not appropriate for trial and error based methods. A wrong tuning may cause unwanted underdamped responses and high cost damages.
- PID controllers are designed and tuned to operate beside a constant operating point. If the load changes are large and the operating conditions force the system too far away from initial operating point, the performance of PID controller decreases.

Self-tuning approach is a reliable solution for these limitations of conventional PID controllers.

In this study, self-tuning PID controllers are analyzed and a self tuning PID controller which controls a second order RLC system is realized.

CHAPTER TWO

THREE-TERM (PID) CONTROLLERS

2.1 Conventional Feedback Control Structures

A very basic concept in control design is that of error feedback. Figure 2.1 illustrates a general output feedback control structure applied to an I/O system, with any disturbance inputs ignored. The transfer function for the original or primary system (often referred to as the plant) is designated by $G_p(s)$. As shown in Figure 2.1, the input to $G_p(s)$ is represented $U(s)$. The concept is to construct a controller $G_c(s)$ whose output $U(s)$ will drive the system $G_p(s)$ so that its output $Y(s)$ is desired. As shown in Figure 2.1, $E(s)$ is the difference between a scaled command input $K_s R(s)$ and some output feedback function $H(s)Y(s)$. When $K(s)=H(s)=1$, then $E(s)$ is just the difference between the input and the output which is called as *error*. The gain K_s on the input is a scaling that is useful in obtaining a desired steady-state relationship between the input and the output.

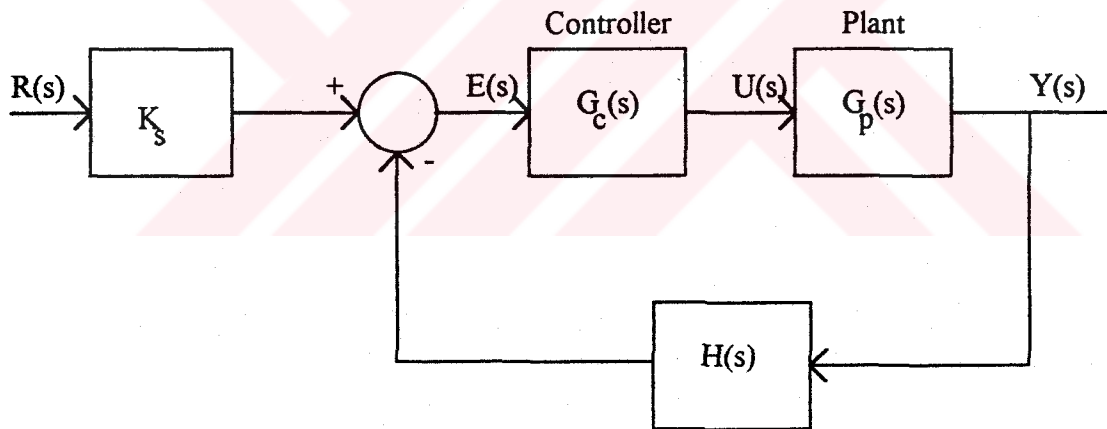


Figure 2.1 Block diagram for output feedback control

The overall transfer function of the controlled system shown in Figure 2.1 is determined as;

$$Y(s) = G_c(s)G_p(s)[K_s R(s) - H(s)Y(s)] \quad (\text{Eq2.1})$$

$$G(s) = \frac{Y(s)}{R(s)} = \frac{K_s G_c(s)G_p(s)}{1 + G_c(s)G_p(s)H(s)} = \frac{Q(s)}{P(s)} \quad (\text{Eq2.2})$$

The design problem is to pick K_s , $G_c(s)$ and $H(s)$ in order to obtain suitable response. If the system is to be a regulator, a typical command input is a constant input (i.e., a step input). For this case, to satisfy the required steady-state output and certain transient performance specification (such as percent overshoot, return time, settling time) is the main problem. If the system to be a tracking servo system, then a typical command input is a sinusoidal residual input. Therefore, the system must track low-frequency inputs and ignore high-frequency inputs associated with noise.

To accomplish these objectives, classical control theory focuses on the choice of the parameters in various possible structures for the controller and feedback transfer functions $G_c(s)$ and $H(s)$ (Grantham & Vincent, 1993, pp.175-180).

2.2 Proportional Control Structure

Proportional control builds a control which is proportional to the error signal. An ideal proportional controller may be represented in time and frequency domains as,

$$u(t) = K e(t) \quad (\text{Eq 2.3})$$

$$G_c(s) = K \quad (\text{Eq 2.4})$$

The gain K is typically provided by an electrical or a mechanical device, for example a power amplifier or a field-controlled DC motor or a gear box.

Proportional control concept is the simplest feedback control idea. However, this type of controllers can not be sufficient in many applications.

In any real application, to achieve a pure proportional control is nearly impossible. Because, in all real applications, there will be a finite inductance or a mass and so on. Therefore a real proportional controller may be represented as

$$G_c(s) = \frac{K}{1 + \tau_s s} \quad (\text{Eq 2.5})$$

However, generally for most proportional control systems, τ_s are sufficiently small and can be neglected.

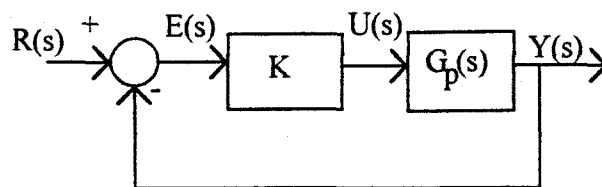


Figure 2.2 Block diagram of proportional control

The transfer function of a proportional control system which is illustrated in Figure 2.2 is,

$$\frac{Y(s)}{R(s)} = \frac{KG_p(s)}{1 + KG_p(s)} \quad (\text{Eq2.6})$$

The output of a proportional controller is directly related to error signal. If there is not any error, controller output provides zero output. Hence, proportional control systems generally operate with error. The measure of error is related to proportional gain K . However, stability of a proportional control system is also related to proportional gain K . Root-locus analysis method which is described in Section 2.3 is a graphic based stability analysis method (Grantham & Vincent, 1993, p.181).

2.3 Root Locus Analysis

An analytical method called *root-locus analysis* is presented in this section for displaying the location of the poles of the closed-loop transfer function

$$\frac{G}{1+GH} \quad (\text{Eq2.7})$$

as a function of the gain-factor K of the open-loop transfer function GH (Distefano & Stubberud & Williams, 1967, pp. 237-259).

2.3.1 Variation of Closed-Loop System Poles

Consider the closed-loop transfer function of the system illustrated in Figure 2.3 is

$$\frac{C}{R} = \frac{G}{1+GH} \quad (\text{Eq2.8})$$

Let the open-loop transfer function GH be represented by,

$$GH = \frac{KN(s)}{D(s)} \quad (\text{Eq2.9})$$

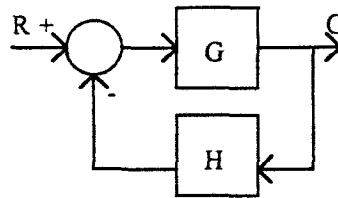


Figure 2.3 Simple closed loop system

where $N(s)$ and $D(s)$ are the finite polynomials in the complex variable s and K is the open-loop gain factor. The closed-loop transfer function then becomes

$$\frac{C}{R} = \frac{G}{1 + K N/D} = \frac{GD}{D + KN} \quad (\text{Eq.2.10})$$

The closed-loop poles are roots of the characteristic equation,

$$D(s) + KN(s) = 0 \quad (\text{Eq.2.11})$$

A locus of these roots plotted in the s -plane as a function of K is called a root-locus.

2.3.2 Real Axis Loci

The real axis section of the root-locus are determined by counting the total number of finite poles and zeros of GH to the right of the points. If $K > 0$, points of the root locus on the real axis lie to the left of an *odd* number of finite poles and zeros. In contrast $K < 0$, points of the root locus on the real axis lie to the left of an *even* number of finite poles and zeros.

2.3.3 Asymptotes

For large distances from the origin in the s -plane, the branches of a root-locus approach a set of straight-line asymptotes. These asymptotes emanate from a point in the s -plane on the real axis called the center of asymptotes σ_c given by,

$$\sigma_c = -\frac{\sum_{i=1}^n p_i - \sum_{i=1}^m z_i}{n - m} \quad (\text{Eq.2.12})$$

where $-p_i$ are poles, $-z_i$ are zeros, n is the number of poles, and m the number of zero of GH .

The angles between the asymptotes and the real axis are given by,

$$\begin{aligned} & \frac{(2l+1)180}{n-m} \text{ degrees for } K > 0 \\ & \frac{(2l)180}{n-m} \text{ degrees for } K < 0 \end{aligned} \quad (\text{Eq.2.13})$$

for $l=0,1,2,\dots, n-m-1$. This results in a number of asymptotes equal to $n-m$.

2.3.4 Breakaway Points

A breakaway point σ_b is a point on the real axis where two or more branches of the root-locus depart from or arrive at the real axis. The location of the breakaway point can be determined by solving the following equation for σ_b :

$$\sum_{i=1}^n \frac{1}{\sigma_b + p_i} = \sum_{i=1}^n \frac{1}{\sigma_b + z_i} \quad (\text{Eq.2.14})$$

2.3.5 Departure and Arrival Angles

The departure angle of a root-locus from a complex pole is given by

$$\theta_D = 180^\circ + \arg GH' \quad (\text{Eq.2.15})$$

where $\arg GH'$ is the phase angle of GH computed at the complex pole, but ignoring the contribution of that particular pole.

The angle of arrival of the root-locus at a complex zero is given by

$$\theta_A = 180^\circ - \arg GH'' \quad (\text{Eq.2.16})$$

where $\arg GH''$ is the phase angle of GH computed at the complex zero, ignoring the effect of that particular pole.

2.3.6 Construction of the Root-Locus

A root-locus plot may be sketched using the procedure as described below:

STEP1: The portions of the root-locus are determined on the real axis.

STEP2: The center and angles of the asymptotes are computed and the asymptotes on the plot are drawn.

STEP3: The departure and arrival angles at complex poles and zeros (if any) are determined and indicated on the plot.

STEP4: The branches of the root-locus are sketched so that each branch of the root-locus either terminates at a zero or approaches infinity along one of the asymptotes.

Example (Distefano *et al*, 1967, pp. 257-259):

The root-locus plot for the closed-loop whose open-loop transfer function is

$$GH = \frac{K}{s(s+2)(s+4)}, \quad K > 0 \quad (\text{Eq.2.17})$$

is sketched in Figure 2.4.

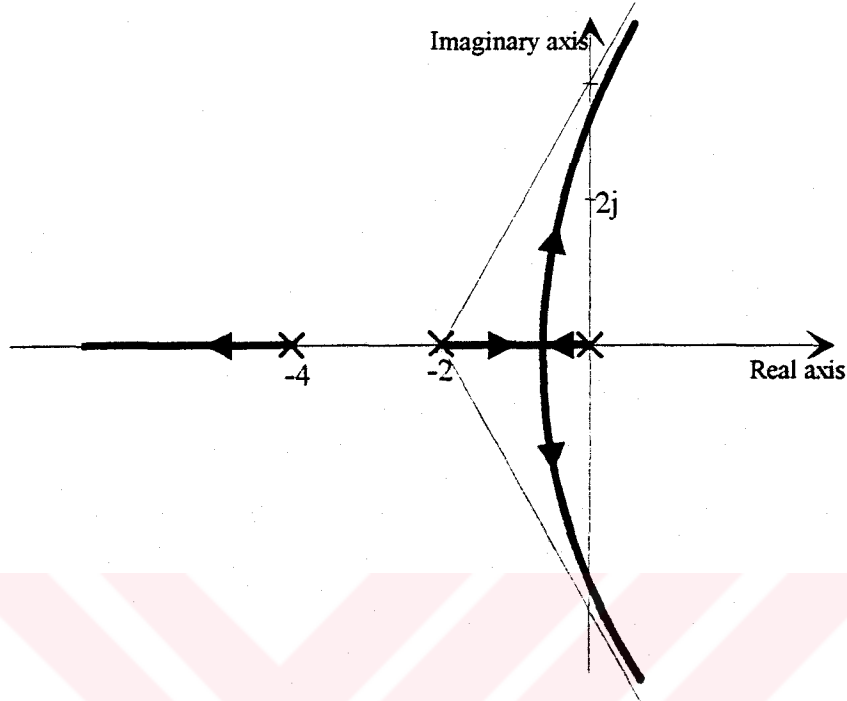


Figure 2.4 Root-locus plot of Eq.2.17

2.4 Integral Control Structure

Integral control is similar to proportional control, but proportional controllers provide constant output while constant error exists, on the other hand, the output of integral controller changes to minimize the error. This property is frequently needed to achieve a zero steady output error due to an uncertain constant system bias. However, if error $e(t)$ is not constant, for example changes from positive to negative, the value of the integral controller will change sign some time later because of the accumulated positive value of the integral. If changes are predictable, integrator may be reset to zero by a reset control system when conditions change. In contrast if changes are unpredictable, a pure integral controller generates an incorrect response.

An ideal integral controller may be represented in time and frequency domains as follows

$$u(t) = \frac{1}{\tau_i} \int_0^t e(\tau) d\tau \quad (\text{Eq.2.18})$$

$$G_C(s) = \frac{1}{\tau_i s} \quad (\text{Eq.2.19})$$

where τ_i is the integral gain constant.

Pure integral control cannot be achieved in a physical system. In physical systems, an integrator is approximated with the transfer function (Grantham *et al*, 1993, p.181),

$$G_C(s) = \frac{1}{\varepsilon + \tau_i s}, \quad 0 < \varepsilon \ll 1 \quad (\text{Eq.2.20})$$

2.4.1 Stability Analysis of First Order Integral Control Systems

An integral controller which has the transfer function as Eq.2.19. controls a first order system whose transfer function is

$$G_p(s) = \frac{q_0}{s + p_0} \quad (\text{Eq.2.21})$$

The closed-loop transfer function of the system is

$$G(s) = \frac{K_s q_0 / \tau_i}{s^2 + p_0 s + q_0 / \tau_i} \quad (\text{Eq.2.22})$$

and root-locus plot of the transfer function in Eq.2.22 is illustrated in Figure2.5.

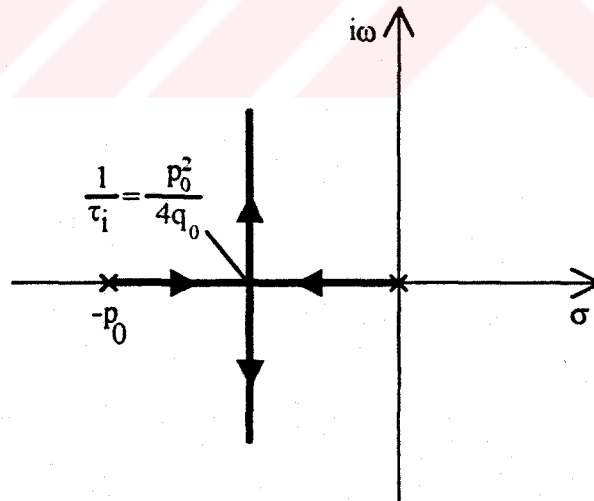


Figure 2.5 Root-locus for integral control of a first order system

According to root-locus plot, a first order integral control system is stable for all positive values of K and all values of q_0 and p_0 if open-loop system is stable. However, integral control cannot stabilize an unstable system (Grantham *et al*, 1993, pp.196-199).

2.4.2 Stability Analysis of Second Order Integral Control Systems

In this case, an integral controller which has the transfer function as Eq.2.19. controls a second order system whose transfer function is

$$G_p(s) = \frac{q_0}{s^2 + p_1s + p_0} \quad (\text{Eq.2.23})$$

The closed-loop transfer function of the system is

$$G(s) = \frac{K_s q_0 / \tau_i}{s^3 + p_1s^2 + p_0s + q_0 / \tau_i} \quad (\text{Eq.2.24})$$

and root-locus plot of the transfer function in Eq.2.22 is illustrated in Figure2.5.

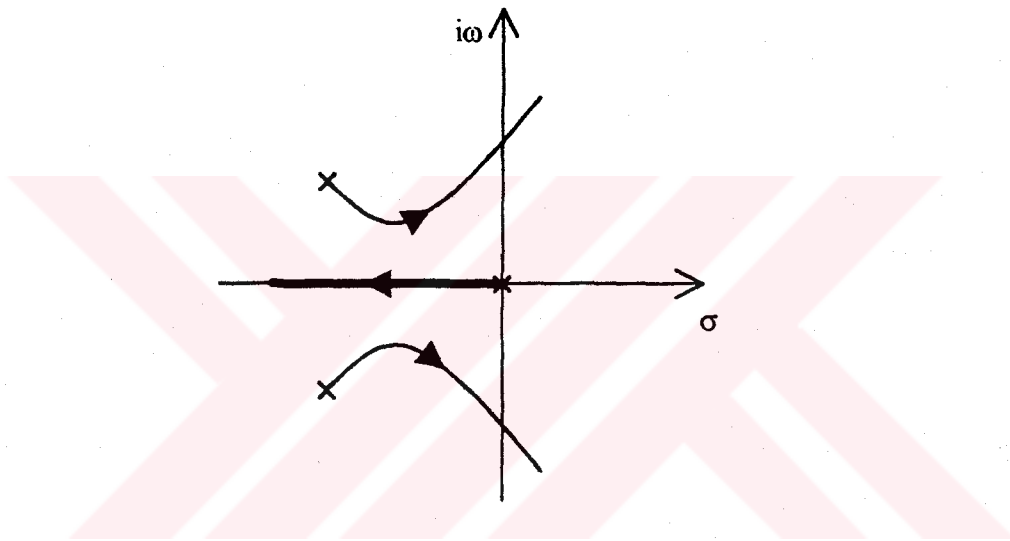


Figure2.6 Root-locus for integral control of a second order system

According to root-locus plot, integral control cannot stabilize an unstable system. Additionally, although open loop system is stable, if K or $1/\tau_i$ is set too high, the closed-loop system will be unstable (Grantham *et al*, 1993, pp.204-206).

2.5 Derivative Control System Structure

Derivative control produces a control that is proportional to the rate of change of the error. If the error is increasing, the control is positive and vice versa. If error is constant, the control is zero. Derivative control does not force the error to zero. Therefore, it is typically applied in conjunction with other control elements as proportional and integral control.

Differentiation is an inherently noisy process. Any noise on the input signal would be differentiated and the output would be disordered.

An ideal differential controller may be represented in time domain and frequency domain as follows (Grantham *et al*, 1993, p.182),

$$u(t) = \tau_d \frac{de(t)}{dt} \quad (\text{Eq.2.25})$$

$$G_C(s) = \tau_d s \quad (\text{Eq.2.26})$$

2.6 Proportional + Integral (PI) Control Structure

PI control is the sum of proportional and integral controls. A PI controller which is shown in Figure 2.7 may be represented in time domain as,

$$u = Ke + \frac{1}{\tau_i} \int_0^t e dt \quad (\text{Eq.2.27})$$

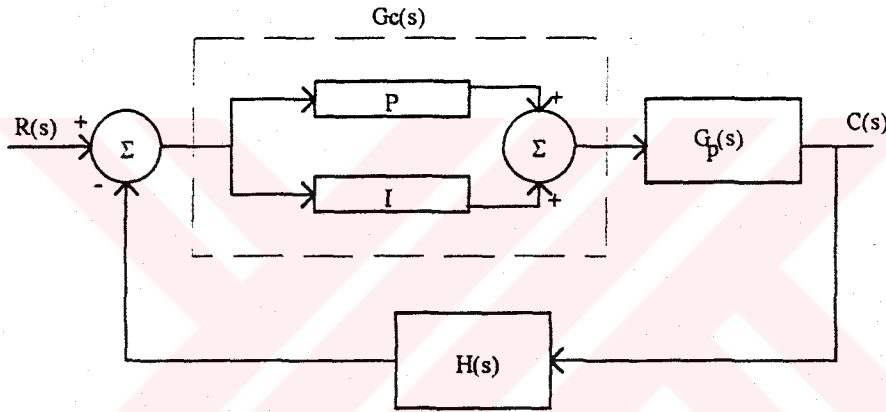


Figure 2.7 Block diagram of a PI control system.

The PI controller has the transfer function

$$G_C(s) = K + \frac{1}{\tau_i s} = \frac{K\tau_i s + 1}{\tau_i s} \quad (\text{Eq.2.28})$$

If the PI control is applied to a first order system described in Eq.2.21, a second order overall transfer function

$$G(s) = \frac{(K\tau_i s + 1)K_s q_0 / \tau_i}{s^2 + (p_0 + Kq_0)s + q_0 / \tau_i} \quad (\text{Eq.2.29})$$

is obtained.

According to Eq.2.29, the poles of overall transfer function can be located on any location by changing K and τ_i . Therefore, an unstable open-loop first order system can be stable by suitable choices for K and τ_i . Additionally, performance of the system can be

modified to obtain desired specifications such as settling time and percent overshoot. PI control is a good choice for controlling first-order systems.

If PI control is applied to a second order system described in Eq.2.23, the overall transfer function

$$G(s) = \frac{(K_s q_0 / \tau_i) + K_s K q_0 s}{s^3 + p_1 s^2 + (p_0 + K q_0) s + q_0 / \tau_i} \quad (\text{Eq.2.30})$$

is obtained.

In Eq.2.30, if $p_1 < 0$, the original system is unstable and PI control cannot stabilize it. Therefore, unlike first-order case, PI control cannot be used in all second-order applications (Grantham *et al*, 1993, p.206).

2.7 Proportional + Integral + Derivative (PID) Control Structure

A PID controller is composed by the sum of proportional, integral and derivative control concepts as shown in Figure 2.8. The time domain output equation of a PID controller is,

$$u(t) = K e(t) + \frac{1}{\tau_i} \int e(t) dt + \tau_d \frac{de(t)}{dt} \quad (\text{Eq.2.31})$$

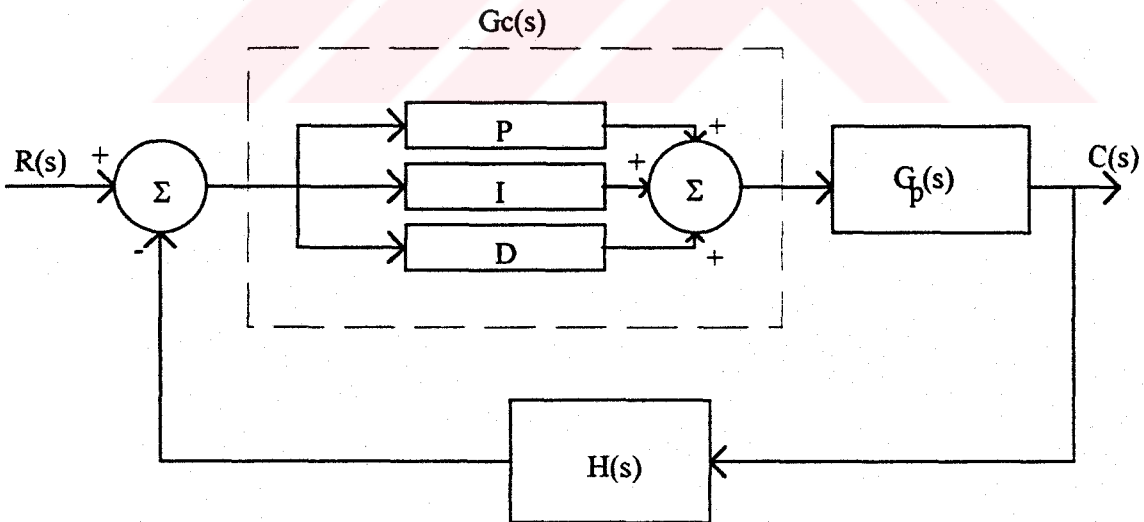


Figure 2.8 Block diagram of a PID control system.

This controller has a transfer function,

$$G_c(s) = K + \frac{1}{\tau_i s} + \tau_d s \quad (\text{Eq.2.32})$$

and this transfer function (Eq.2.32) may be rewritten as

$$G_c(s) = \frac{K(s^2 + as + b)}{s} \quad (\text{Eq.2.33})$$

Therefore, a PID controller introduces a transfer function with one pole at the origin and two zeros that can be located anywhere in the left-hand s-plane.

In a PID controller, proportional control avoids delay in the system. Integral control eliminates offset which can be invoked by using proportional control. The primary benefit from the integral term is the elimination of steady state error while differential term helps improve the responsiveness and stability. Proportional control has the effect of reducing the rise time and reduces (but never eliminate) the steady-state error. Integral control has the effect of eliminating the steady-state error, but it makes the transient response worse. If integral control to be used, a small K_i should always be tried first. Derivative control has the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response. Overshoot, rise time and settling time of a system response is shown in Figure 2.9. The effects on the closed-loop response of adding to the controller terms K_p , K_i and K_d are listed in Table 2.1.

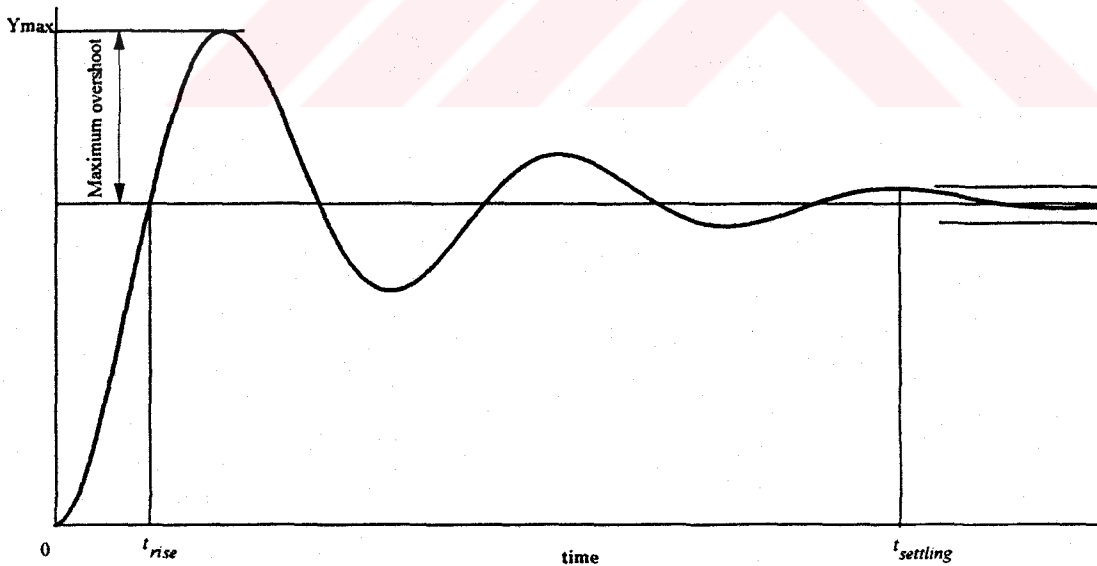


Figure 2.9 System response to a step input

Table 2.1 The effects of K_p , K_i and K_d on the closed-loop response.

Closed-loop response	Rise time	Overshoot	Settling time	Steady-state error
K_p	Decreases	Increases	No change	Decreases
K_i	Decreases	Increases	Increases	Eliminates
K_d	No change	Decreases	Decreases	No change

Note that these correlation are not exactly accurate, because K_p , K_i , K_d are related to each other. Changing one of these variables can change the effect of the other two. For this reason, the table should only be used as a reference while K_i , K_p , K_d values are being determined by trial & error (Web Site of The University of Michigan, 1997).

CHAPTER THREE

SELF-TUNING SYSTEMS

3.1 Introduction

The conventional feedback controllers which were analyzed in Chapter 2 have constant design parameters as K , τ_i , τ_d . These parameters must be determined for each system by the system engineers. The tuning of one or two term controllers as P, I, PI or PD is fairly easy, because these controllers have one or two parameters to adjust. However, a PID controller which has three or four parameters is not always easy to tune, particularly if the dynamic of the process is slow. The derivative action is, therefore, frequently switched off in industrial controllers although this deteriorates the operating performance. Additionally, some controllers, which may include feedforward or state feedback can often have more than 10 adjustable parameters. To adjust so many parameters without a systematic procedure is nearly impossible. One method to adjust the controller parameters is to develop a mathematical model for the process and to derive the parameters by using some control design procedures. However, another appropriate method to combine system identification and control design is self-tuning.

Another motivation for using adaptive or self-tuning control is that the characteristics of the process and its disturbances may change with time. Frequently, these changes are unpredictable and of a type which cannot be compensated for by robust design. The adjustment mechanism of a self-tuning or adaptive system can provide the means of adapting to system change. If the changes are not too rapid, a properly designed self-tuning controller may be used for continuous tuning to obtain a performance which is close to the optimal one (Narendra & Monopoli, 1979, pp.2-3).

3.2 Self-Tuning and Adaptive Control Approaches

The main conception of self-tuning and adaptive control is to construct a system that will automatically adjust its parameters to obtain required performance.

Self-tuning and adaptive control concepts convey nearly same idea. However, some systems are approximately time invariant. These systems have initial tuning problem and to change the system parameters is not necessary after initial tuning. Thus, adjustment mechanism can be disabled after initial tuning. In contrast, some systems change in time and need continuous tuning. In the terminology, initial adjustment is called self-tuning and continuous adjustment is called adaptive control (Wellstead *et al.* 1991, pp.4-6).

3.3 Self-Tuning System Structure

Tuning problem of a conventional controller may be defined as determining the parameters of the controller which controls an unknown system to obtain desired system response.

A self-tuning control system can adjust its parameters by using some additional mechanisms as a system identifier and a controller designer. The main concept of self-tuning is to identify the system (plant) dynamic and to change the controller parameters according to identified system and desired overall system response.

A self-tuning control system includes four main parts:

- i Controller
- ii. System (Plant)
- iii. System identifier
- iv. Controller designer.

A conventional control system includes a controller and a system as same as a self-tuning control system. However, since the system identifier and the design mechanisms are not included, conventional systems are had to tune by the design engineer. Moreover, a self-tuning system able to approximate the system transfer function to a desired structure as a second or higher order transfer function. Thus, the design engineer must determine a general system model, but not to determine the coefficients of the system transfer function.

The second necessity to determine the appropriate controller parameters is to decide the overall system response and result, the system transfer function.

Figure 3.1 illustrates the design steps of a control system. In step 1, the mathematical representation of the system is defined according to design objectives. In self-tuning design procedure, the system to be controlled must be modeled as a linear transfer function with undetermined parameters and required overall system must be modeled exactly according to the system response by the design engineer. Thus, automatic system identifier can determine the undetermined system parameters and complete the modeling task. Additionally, controller structure must be determined by the design engineer as a PID controller or another type of controller structure.

In step 2, design stage, the parameters of the controller are calculated individually by the controller designer mechanism according to determined overall system structure and

calculated system parameters. Then, the calculated parameters are downloaded to controller.

In the implementation stage, the controller operates with the parameters which were downloaded in the design stage.

The validation phase is essential to success and is one of the most important arguments in favor of self-tuning. Because, in conventional off-line design the validation phase often proves unsatisfactory and this leads to a time-consuming repetition of the whole sequence of modeling, design and implementation. The main advantage of self-tuning is that the sequence is performed on-line and much faster. However, validation is associated with a qualitative measurement of performance and it is not an algorithmic element of the self-tuning loop.

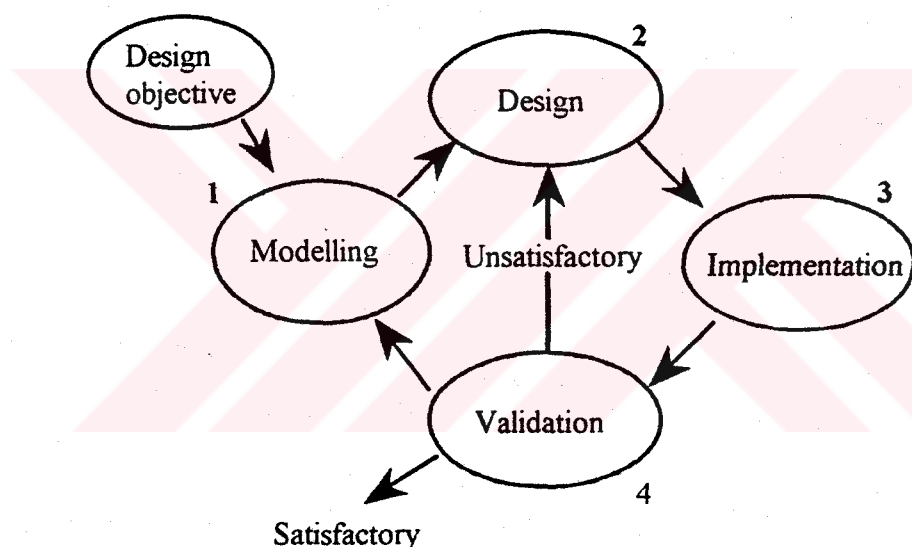


Figure 3.1 Control system design steps

A self-tuning structure and the three stages of system design, modeling, design and implementation are sketched in Figure 3.2. The sequence of the tasks is summarized in a simple algorithm below (Wellstead *et al*, 1991, pp. 11-12):

STEP1: Implement the parameters which are initialized or determined in the previous iteration.

STEP2: Estimate the system transfer function by using input and output relations of the system.

STEP3: Calculate appropriate controller parameters from estimated system transfer function and desired overall system transfer function.

STEP4: Change the controller parameters with calculated parameters in STEP3.

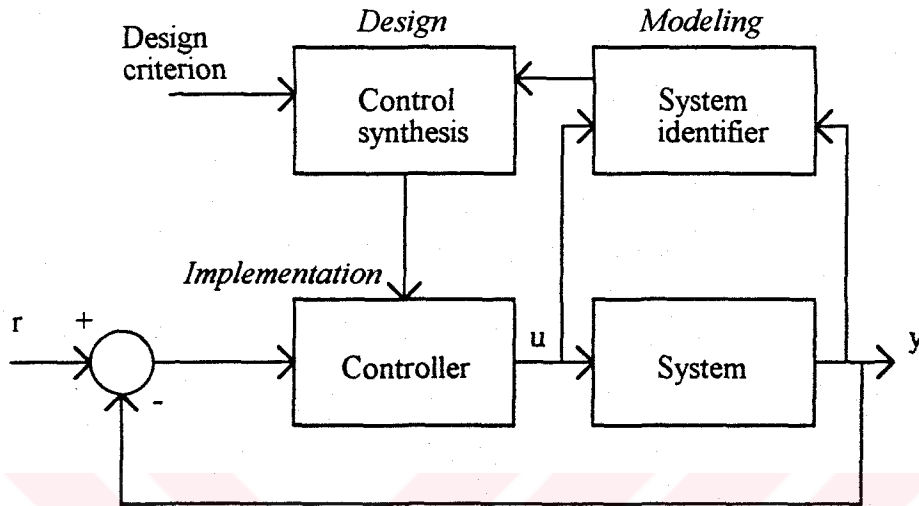


Figure 3.2 Self-tuning controller structure

3.4 System Models

"No mathematical model can ever display every nuance of behavior of the system that it is constructed to represent, but this is never required (Wellsteat *et al*, 1991, p.41)." Most real dynamic processes are nonlinear and continuous in time. In contrast, generally linear models for both the parameters and the data are used in self-tuning systems. Because, linear approach eases analysis, estimation and design.

Self-tuning systems are generally microprocessor based systems. As a result, self-tuning systems are inherently digital and discrete time modelling is suitable for these systems.

A discrete signal and its continuous time equivalent are shown in Figure 3.3. In this illustration, $u(t)$ is a discrete sequence that is obtained from a continuous signal $u_c(\tau)$. A discrete sequence may be obtained from

$$u(t) = u_c(t\tau_s) \quad (\text{Eq.3.1})$$

where τ_s is the sampling interval.

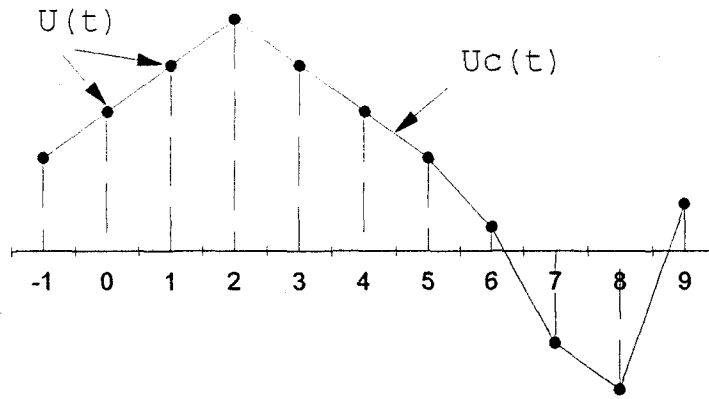


Figure 3.3 Discrete signal and its continuous time equivalent

The relation between input and output signals of a discrete time system can be represented as a linear difference equation.

$$x(t) + a_1 x(t-1) + \dots + a_{n_a} x(t-n_a) = b_0 u(t) + b_1 u(t-1) + \dots + b_{n_b} u(t-n_b) \quad (\text{Eq.3.2})$$

The unit backward shift operator z^{-1} defined by

$$z^{-1} x(t) = x(t-1) \quad (\text{Eq.3.4})$$

and Eq.3.2 can be rewritten as (Wellstead *et al*, 1991, pp.41-43),

$$x(t) = \left[\frac{B}{A} \right] u(t) \quad (\text{Eq.3.5})$$

where A and B are polynomials as,

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + \dots + a_{n_a} z^{-n_a} \\ B(z^{-1}) &= b_0 + b_1 z^{-1} + \dots + b_{n_b} z^{-n_b} \end{aligned} \quad (\text{Eq.3.6})$$

In Figure 3.4, a discrete time system model whose input and output are $u(t)$ and $x(t)$ is shown.

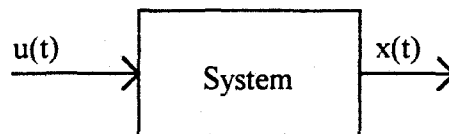


Figure 3.4 Discrete time system model.

3.5 Least Squares (LS) Method

A self-tuning control system must identify the physical system to be controlled according to construction defined by the design engineer. This construction is generally a discrete time transfer function with undetermined parameters. Control system must determine these parameters using input/output relation of the physical system. Least squares is the most familiar method to determine the parameters of a system(plant) transfer function.

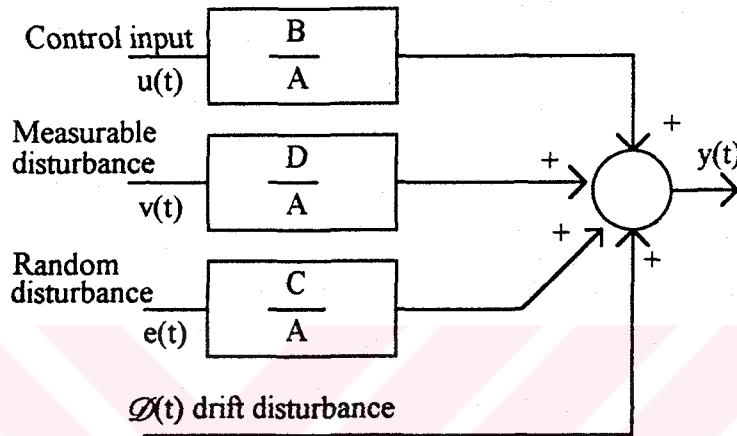


Figure 3.5 A system with all possible input and disturbance components.

In Figure 3.5, a system block diagram with control input, measurable, random and drift disturbance components is shown. This system has a transfer function (Wellstead *et al*, 1991, pp.71-72),

$$A y(t) = B u(t-1) + D v(t) + \varnothing(t) + C e(t) \quad (\text{Eq.3.7})$$

where,

$$A = 1 + a_1 z^{-1} + \dots + a_{n_a} z^{-n_a} \quad (\text{Eq.3.8})$$

$$B = b_0 + b_1 z^{-1} + \dots + b_{n_b} z^{-n_b}$$

$$D = d_0 + d_1 z^{-1} + \dots + d_{n_d} z^{-n_d}$$

$$\varnothing(t) = d_0 + d_1 t + \dots + d_{n_d} t^{n_d}$$

$$C = 1 + c_1 z^{-1} + \dots + c_{n_c} z^{-n_c}$$

Eq 3.7 can be rewritten as,

$$y(t) = x^T(t)\theta + e(t) \quad (\text{Eq.3.9})$$

where θ is the vector of unknown parameters, defined by

$$\theta^T = [-a_1, \dots, -a_{n_a}, b_0, \dots, b_{n_b}, d_0, \dots, d_{n_d}, d_0, \dots, d_{n_d}, c_1, \dots, c_{n_c}] \quad (\text{Eq.3.10})$$

and $x(t)$ is defined by,

$$x^T(t) = [y(t-1), \dots, y(t-n_a), u(t-1), \dots, u(t-n_b-1), v(t), \dots, v(t-n_d), 1, t, \dots, t^{n_d}, e(t-1), e(t-2), \dots, e(t-n_c)] \quad (\text{Eq.3.11})$$

To determine the unknown parameter vector θ , a model can be assumed

$$y(t) = x^T(t)\hat{\theta} + \hat{e}(t) \quad (\text{Eq.3.12})$$

where $\hat{\theta}$ is a vector of adjustable model parameters and $\hat{e}(t)$ is the corresponding modelling error at time t . A suitable selection of $\hat{\theta}$ must minimize $\hat{e}(t)$.

$$\hat{e}(t) = e(t) + x^T(t)(\theta - \hat{\theta}) \quad (\text{Eq.3.13})$$

Equation 3.12 can be rewritten,

$$y = X\hat{\theta} + \hat{e} \quad (\text{Eq.3.14})$$

and error vector \hat{e} can be obtained clearly,

$$\hat{e} = y - X\hat{\theta} \quad (\text{Eq.3.15})$$

To minimize \hat{e} , the sum of squares of errors, J , must be minimized.

$$J = \sum_{t=1}^N \hat{e}^2(t) = \hat{e}^T \hat{e} \quad (\text{Eq.3.16})$$

To find least squares estimate, J can be rewritten,

$$J = (y - X\hat{\theta})^T (y - X\hat{\theta}) = y^T y - \hat{\theta}^T X^T y - y^T X\hat{\theta} + \hat{\theta}^T X^T X\hat{\theta} \quad (\text{Eq.3.17})$$

Setting to zero the derivative of J with respect to $\hat{\theta}$ for a stationary point,

$$\frac{\partial J}{\partial \hat{\theta}} = -2X^T y + 2X^T X\hat{\theta} = 0 \quad (\text{Eq.3.18})$$

If the second derivative matrix

$$\frac{\partial^2 J^2}{\partial^2 \hat{\theta}} = X^T X \quad (\text{Eq.3.18})$$

is positive definite, the solution of the first derivative is a unique minimum.

Thus, the least squares estimator for the parameter vector is (Wellstead *et al*,1991,p.73),

$$\hat{\theta} = [X^T X]^{-1} [X^T y] \quad (\text{Eq.3.19})$$

3.6 Recursive Least Squares (RLS)

In many self-tuning systems, parameter estimation method is designed to be iterative because of two main reasons:

1. In self-tuning systems, estimated parameters are updated at each sample interval by using all previous data and a new data. However, in the previous estimation cycle, the old data are already calculated and previously estimated parameters contain the effect of the previous data. Least squares method can not use the previously estimated parameters and calculates all off the previous data in each sample interval. This means too many unnecessary computations and waste of time. In contrast, *RLS* is an iterative method that calculates the self-tuning system parameters by using the previously estimated parameters ($\hat{\theta}(t-1)$), and the last sampled data. So, the use of *RLS* method reduces parameter estimation time.
2. All of the previous data must be stored by using the least squares method since these data are needed for further calculations. However, *RLS* parameter estimation needs only previously estimated parameters and the last sampled data. As a result, *RLS* method reduces the necessary data memory capacity.

An *RLS* algorithm is described in the following section.

3.6.1 Recursive Least Squares Algorithm (Wellstead *et al*,1991,p.89)

At the step $t+1$,

Step1: Form $x(t+1)$ using the new data.

Step2: Form $\varepsilon(t+1)$ using

$$\varepsilon(t+1) = y(t+1) - x^T(t+1)\hat{\theta}(t) \quad (\text{Eq.3.20})$$

Step3: Form $P(t+1)$ using

$$P(t+1) = P(t) \left[I_m - \frac{x(t+1)x^T(t+1)P(t)}{1+x^T(t+1)P(t)x(t+1)} \right] \quad (\text{Eq.3.21})$$

STEP4: Update $\hat{\theta}(t)$

$$\hat{\theta}(t+1) = \hat{\theta}(t) + P(t+1)x(t+1)\varepsilon(t+1) \quad (\text{Eq.3.22})$$

STEP5: Wait for the next time step to elapse and loop back to STEP1.

3.6.2 Initializing The Estimator

In general, the choice of initial parameter estimations are not curical to convergence behavior. However, it is useful to select the initial parameters as

$$\begin{aligned} a_1 &= -1 & b_0 &= \tau_s \\ a_i &= 0 (i \neq 1) & b_i &= 0 (i \neq 0) \end{aligned}$$

The usual way to determine the data vector with initial values is to begin sampling the information for a few time steps before the recursive estimator is started.

A standard choice for $P(0)$ is the unit matrix scaled by a positive scalar r .

$$P(0) = rI_m$$

Typically, r is set in region 1-1000 according to application (Wellstead *et al*, 1991, pp. 119-120).

3.7 Pole Assignment Control

The main aims of feedback design are to modify the dynamic response of a system and to reduce the sensitivity of a system output to disturbances. The poles of a system determine the stability of a system and effect the nature of its transient response. Thus, it is possible to obtain desired dynamic response by changing the placement of the system poles.

3.7.1 Controller Design by Pole Assignment for First Order Systems

A first order system illustrated in Figure3.6 may be represented in discrete time domain by (Wellstead *et al*, 1991, pp.244-245),

$$y(t) = \frac{z^{-1}b}{1-az^{-1}}u(t) \quad (\text{Eq.3.23})$$

and this model has a continuous time transfer function as

$$\frac{Y(s)}{U(s)} = \frac{f}{1 + \alpha s} \quad (\text{Eq.3.24})$$

where α is the time constant and f is the system gain. α and f are related to the discrete time model parameters by

$$\begin{aligned} a &= \exp(-\tau_s / \alpha) \\ b &= (1 - a)f \end{aligned} \quad (\text{Eq.3.25})$$

where τ_s is the sampling time. As seen clearly, the time constant of open loop system is related to α .

However, the overall transfer function of the system shown in Figure 3.6 is represented by

$$y(t) = \frac{bh}{1 - (a - bg)z^{-1}} r(t-1) \quad (\text{Eq.3.26})$$

In this case, closed loop system response is related to $(a - bg)$ instead of a . The location of the pole may be assigned by changing the feedback loop gain g . Thus, the system time constant alters from α to β where

$$\begin{aligned} \alpha &= -\tau_s / \ln(a) \\ \beta &= -\tau_s / \ln(a - bg) \end{aligned} \quad (\text{Eq.3.27})$$

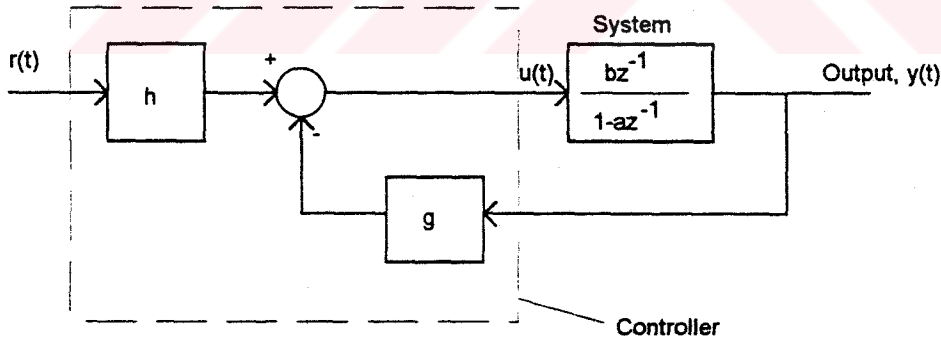


Figure 3.6 A closed loop first order system

3.7.2 Three-Term Controller Design by Pole Assignment for Second Order Systems

If the system shown in Figure 3.6 is a second order system then the desired pole set will consist of the two zeros of the second order polynomial

$$T = 1 + t_1 z^{-1} + t_2 z^{-2}$$

where,

$$\begin{aligned} t_1 &= -2 \exp(-\xi \omega_n \tau_s) \cos\{\tau_s \omega_n (1 - \xi^2)^{1/2}\} \\ t_2 &= \exp(-2\xi \omega_n \tau_s) \end{aligned} \quad (\text{Eq.3.28})$$

where ξ and ω_n are respectively the damping factor and natural frequency of the desired closed-loop second-order transient response.

Three-term controller (PID controller) is a second order controller. A PID controller can be represented in discrete time form as (Wellstead *et al*, 1991, p.249),

$$u(t) = \frac{r(t)(g_0 + g_1 + g_2) - (g_0 + g_1 z^{-1} + g_2 z^{-2})y(t)}{1 - z^{-1}} \quad (\text{Eq.3.29})$$

The coefficients g_0, g_1, g_2 are related to k_p, k_i, k_d which are the proportional, integral and derivative settings of the PID controller by

$$\begin{aligned} k_p &= -g_1 - 2g_2 \\ k_d &= g_2 \\ k_i &= g_0 + g_1 + g_2 \end{aligned} \quad (\text{Eq.3.30})$$

The transfer function of a second order system which is controlled by a PID controller is represented by

$$y(t) = \frac{b_0 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} u(t) \quad (\text{Eq.3.31})$$

The overall system transfer function is obtained by combining Eq.3.29 and Eq.3.31 as the following form ,

$$y(t) = \frac{b_0 z^{-1} (g_0 + g_1 + g_2)}{(1 - z^{-1})(1 + a_1 z^{-1} + a_2 z^{-2}) + b_0 z^{-1} (g_0 + g_1 z^{-1} + g_2 z^{-2})} r(t) \quad (\text{Eq.3.32})$$

The pole positions of the system is strongly related to system dynamic. The coefficients g_0, g_1, g_2 are the controller parameters and can be set to a value to provide desired closed loop performance. Thus, the pole placement of the system can be changed to appropriate locations by determining g_0, g_1, g_2 .

By equating the denominator of (Eq.3.32) to a general second order polynomial

$$(1 - z^{-1})(1 + a_1 z^{-1} + a_2 z^{-2}) + b_0 z^{-1} (g_0 + g_1 z^{-1} + g_2 z^{-2}) = 1 + t_1 z^{-1} + t_2 z^{-2} \quad (\text{Eq.3.33})$$

the following solution for the controller setting is obtained (Wellstead *et al*,1991,p.249),

$$\begin{aligned} g_0 &= \frac{t_1 + (1 - a_1)}{b_0} \\ g_1 &= \frac{t_2 + (a_1 - a_2)}{b_0} \\ g_2 &= \frac{a_2}{b_0} \end{aligned} \quad (\text{Eq.3.34})$$

3.8 Self Tuning PID Controller Design Algorithm

STEP1: Determine the required overall system dynamic response, damping factor and natural frequency.

STEP2: Calculate the denominator polynomial of the desired system from

$$T = 1 + t_1 z^{-1} + t_2 z^{-2}$$

$$\begin{aligned} t_1 &= -2 \exp(-\xi \omega_n \tau_s) \cos\{\tau_s \omega_n (1 - \xi^2)^{1/2}\} \\ t_2 &= \exp(-2\xi \omega_n \tau_s) \end{aligned}$$

STEP3: Find the transfer function of the system by using RLS method which is described in Section3.6.

STEP4: Calculate g_0, g_1, g_2 by using (Eq3.34)

STEP5: Calculate k_p, k_i, k_d by using (Eq3.30) and set the proportional, integral and derivative gains of PID controller.

CHAPTER FOUR

A SELF-TUNING PID CONTROLLER APPLICATION

4.1 Introduction

In this chapter, a self-tuning PID controller application is discussed. Most of physical systems may be represented by first or second order transfer functions. PI controllers are sufficient to control all first order systems as described in Section 2.6. However, second order systems may need PID controllers to obtain desired overall response. In this application, a simple second order system which is a serial RLC circuit is selected to control because of its applicational simplicity.

The selection of a RLC circuit as a plant provides some advantages. The first advantage of using RLC circuit is that system parameters can be changed in wide range easily by changing the value of a component for example the value of the capacitor. The second advantage is that a transducer which converts the system output to a voltage level in order to obtain feedback is not necessary since the system output is already a voltage level of a component as the capacitor. Most physical systems (for example a heater or an electric motor) must be driven by a power drive unit as a PWM modulator or a phase control unit. Therefore, calculation of these transfer functions may not be easy. Thus, comparing of the experimental results with calculated predictions may not be correct. The third advantage of using RLC circuit is that the system can be driven by digital to analog converter output without any non-linear drive unit.

However, a RLC circuit has also a disadvantage. Response time and damping factor of a RLC circuit is related with the values of R, L and C. Thus, inductors with large values are necessary to obtain slow and underdamped responses. Otherwise, system responses are faster than processor speed.

4.2 Mathematical Analysis of a Serial RLC Circuit

A serial RLC circuit which is shown in Figure 4.1, includes a resistor, an inductance and a capacitor in serial connection. The fundamental integrodifferential equation of this circuit is

$$V_{in} + Ri + L \frac{di}{dt} + \frac{1}{C} \int i dt - V_{out}(t_0) = 0 \quad (\text{Eq.4.1})$$

The second-order equation obtained by differentiating the equation with respect to time is,

$$L \frac{d^2 i}{dt^2} + R \frac{di}{dt} + \frac{i}{C} = 0 \quad (\text{Eq.4.2})$$

The overdamped response of Eq.4.2 is

$$i(t) = A_1 e^{s_1 t} + A_2 e^{s_2 t} \quad (\text{Eq.4.3})$$

where,

$$\begin{aligned} s_{1,2} &= -\frac{R}{2L} \pm \sqrt{\left(\frac{R}{2L}\right)^2 - \frac{1}{LC}} \\ &= -\alpha \pm \sqrt{\alpha^2 - \omega_0^2} \\ &= -\alpha \pm j\omega_d \end{aligned}$$

and thus,

$$\begin{aligned} \alpha &= \frac{R}{2L} \\ \omega_0 &= \frac{1}{\sqrt{LC}} \\ \omega_d &= \sqrt{\omega_0^2 - \alpha^2} \end{aligned}$$

The form of the critically damped response is

$$i(t) = e^{-\alpha t} (A_1 t + A_2) \quad (\text{Eq.4.4})$$

and the underdamped case may be written as

$$i(t) = e^{-\alpha t} (B_1 \cos \omega_d t + B_2 \sin \omega_d t) \quad (\text{Eq.4.5})$$

For three cases, the capacitor voltage, V_{out} , is (Hayt & Kemmerly, 1978, pp.244-246),

$$V_{out} = V_{out}(t_0) + \frac{1}{C} \int i(t) dt \quad (\text{Eq.4.6})$$

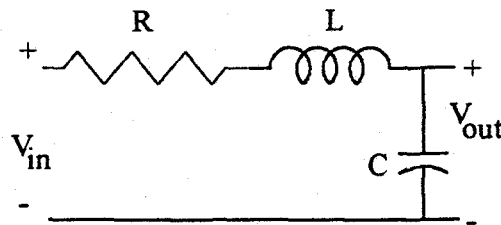


Figure 4.1 Serial RLC circuit

The frequency domain transfer function of the circuit shown in Figure 4.1 is,

$$\frac{V_{out}}{V_{in}} = \frac{1/LC}{s^2 + s\frac{R}{L} + \frac{1}{LC}} \quad (\text{Eq.4.7})$$

A general formulation for second order systems may be written as,

$$G(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \quad (\text{Eq.4.8})$$

and Eq.4.7 may be rewritten in discrete time domain as,

$$W(z^{-1}) = \frac{b_1 z^{-1} + b_2}{1 - a_1 z^{-1} - a_2 z^{-2}} \quad (\text{Eq.4.9})$$

where,

$$\xi > 1;$$

$$\sigma = \xi\omega_n, \quad \omega = \omega_n \sqrt{\xi^2 - 1}, \quad \gamma = \frac{\xi}{\sqrt{\xi^2 - 1}}$$

$$a_1 = 2 \exp(-\sigma T) \cosh(\omega T)$$

$$a_2 = -\exp(-2\sigma T)$$

$$b_1 = 1 - \exp(-\sigma T) [\cosh(\omega T) + \gamma \sinh(\omega T)]$$

$$b_2 = \exp(-\sigma T) [\exp(-\sigma T) - \cosh(\omega T) + \gamma \sinh(\omega T)]$$

$$\xi = 1;$$

$$a_1 = 2 \exp(-\omega_n T)$$

$$a_2 = -\exp(-2\omega_n T)$$

$$b_1 = 1 - \exp(-\omega_n T) [1 + \omega_n T]$$

$$b_2 = \exp(-\omega_n T) [\exp(-\omega_n T) + \omega_n T - 1]$$

$$\xi < 1;$$

$$\sigma = \xi\omega_n, \quad \omega = \omega_n \sqrt{1 - \xi^2}, \quad \gamma = \frac{\xi}{\sqrt{1 - \xi^2}}$$

$$a_1 = 2 \exp(-\sigma T) \cos(\omega T)$$

$$a_2 = -\exp(-2\sigma T)$$

$$b_1 = 1 - \exp(-\sigma T) [\cos(\omega T) + \gamma \sin(\omega T)]$$

$$b_2 = \exp(-\sigma T) [\exp(-\sigma T) - \cos(\omega T) + \gamma \sin(\omega T)]$$

(Neuman & Baradello, 1979, p.857).

4.3 Application

A self-tuning system must contain a controller, a plant, a system identifier and a controller designer parts as discussed in Section 3.3. In this application, a data acquisition board is used as the controller and the feedback input port of the system identifier. System identification and controller design steps are done by an 80486 based PC and the plant is constructed by a serial RLC circuit.

In this application, the process speed of the data acquisition board is insufficient to provide both PID control and send the measured values to the PC simultaneously. So, parameter estimation step of the self-tuning algorithm is realized by cancelling the PID control to apply a squarewave test signal to the plant directly as shown in Figure 4.2.

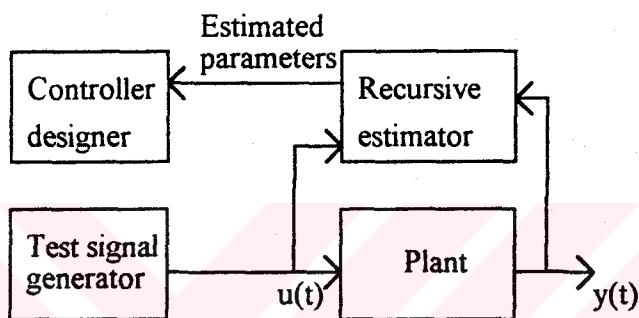


Figure 4.2 Parameter estimation step of the self-tuner

Recursive least squares (RLS) algorithm which was discussed in Section 3.6 is used to obtain the parameters of the plant. The recursive estimator determines the parameters of the plant as a second order discrete time transfer function

$$y(t) = \frac{b_0 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} u(t) \quad (\text{Eq.4.10})$$

After determining the parameters a_1 , a_2 , b_0 , the coefficients of the PID controller are calculated by using desired overall transfer function specifications (damping factor, natural frequency) and the estimated plant parameters. In this application, desired overall system is a second order system. The calculation method of the PID coefficients were discussed in Section 3.7.2.

After the calculated PID controller coefficients have been downloaded to the controller, the system performs PID control as shown in Figure 4.3.

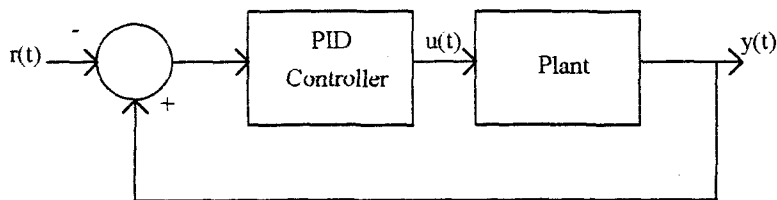


Figure 4.3 Implementation of PID control

A complete flow chart of the applied self-tuning algorithm is shown in Figure 4.4.

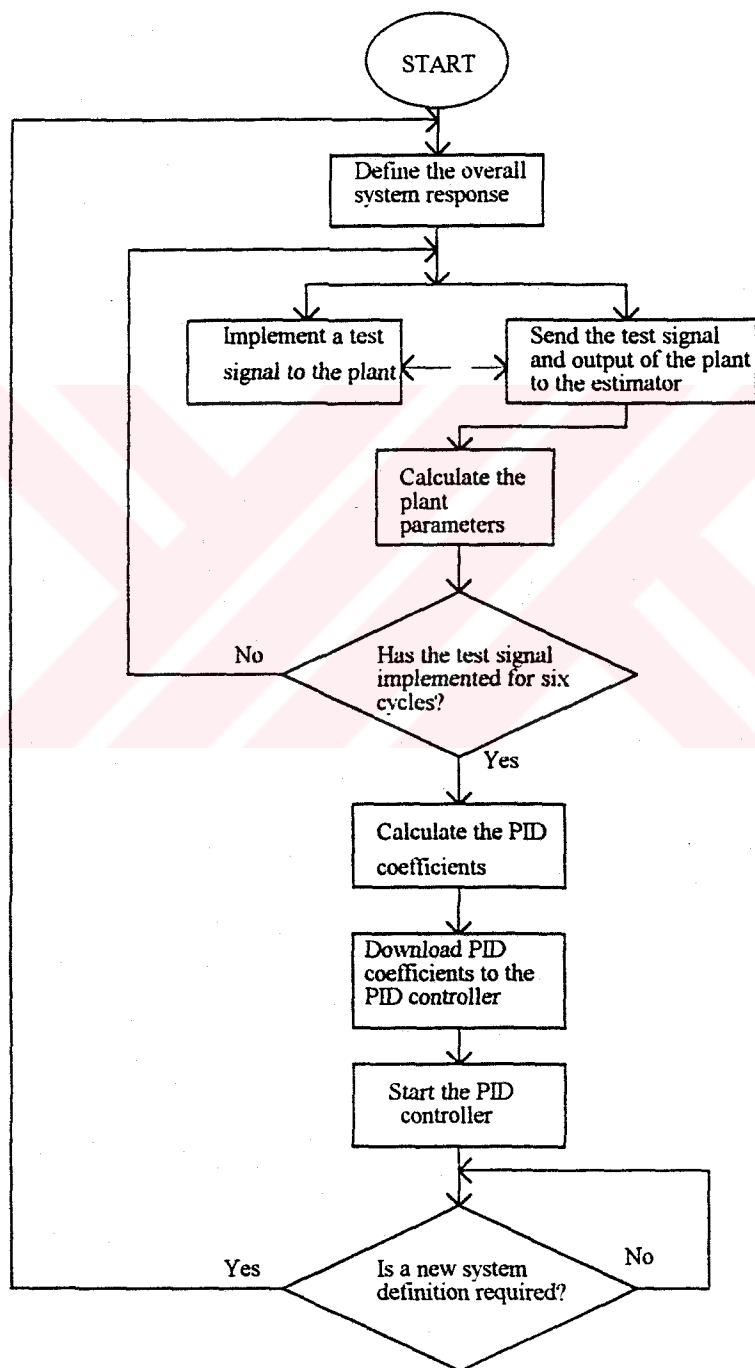


Figure 4.4 Flow chart of the self-tuning application.

CHAPTER FIVE

HARDWARE AND SOFTWARE OF THE APPLICATION

5.1 Hardware Structure of the System

In this application, analog input/output and PID implementation are provided by a data acquisition board called DAP800 (see Appendix). DAP800 has an on-board Intel 80C188 processor and is appropriate for intelligent data acquisition and control applications. It provides 8 analog inputs, 2 analog outputs, arithmetic operations and some mathematical functions as PID, FFT, digital filtering etc. DAP800 is located in an 80486 microprocessor based personal computer (PC). Since DAP800 has its own microprocessor, it can perform stand-alone while the PC performs another process. However, PC and DAP800 can communicate each other by using the data bus of the PC. Thus, the PC software can process the data from DAP800 while DAP800 performs another process such as signal generating or data sampling.

In the first stage of the self-tuning, DAP800 provides a squarewave test output to the plant. It sends the data -which are read from input and output of the plant- to the PC. The PC performs a recursive least square algorithm by using these data concurrently.

To obtain a better estimation result, the period of the test signal should be a few times longer than the damped period of the plant.

In the second stage of the self-tuning, the coefficients of the PID controller are calculated from the estimated parameters and desired overall system response by the PC software. After these calculations, calculated PID coefficients and another DAP800 software procedure which performs PID control are downloaded to DAP800. The setpoint of the PID controller is changed by PC software and the response of the overall system is recorded to compare with theoretical results. The illustration of the system is shown in Figure 5.1.

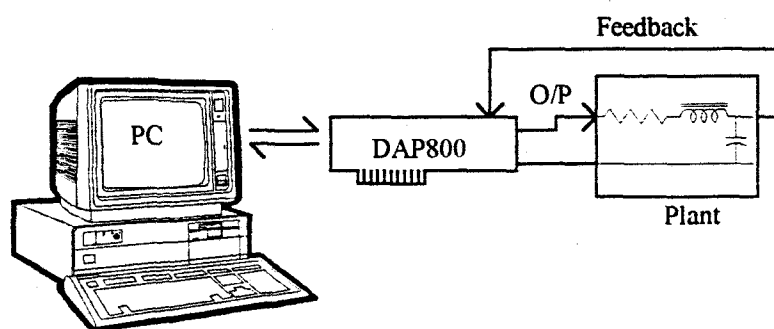


Figure 5.1 Self-tuning system application

5.2 The Mathematical Model of the Plant

The plant which is controlled by self-tuning PID controller in the application is shown in Figure 5.2.

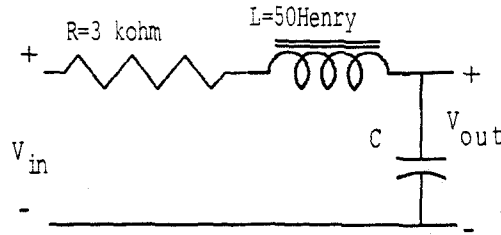


Figure 5.2 The plant which is controlled in the application

In this application, capacitor value, C , is changed to obtain different system responses. In this paper, only two values of C which are $1.5\mu F$ and $15\mu F$ are discussed:

For $C = 1.5\mu F$;

The damping factor, ξ , is obtained by combining Eq.4.7 and Eq.4.8 as,

$$\xi = \frac{R}{2} \sqrt{\frac{C}{L}} \quad (\text{Eq.5.1})$$

and the damping factor of the system can be calculated from Eq.5.1 as

$$\xi = \frac{3000}{2} \sqrt{\frac{1.5 \times 10^{-6}}{50}} = 0.26$$

The damped frequency, ω_d , can be calculated from Eq.4.3 as

$$\alpha = \frac{R}{2L} = 30$$

$$\omega_0 = \frac{1}{\sqrt{LC}} = 115.47$$

$$\omega_d = \sqrt{\omega_0^2 - \alpha^2} = 111.5 \text{ rad/s}$$

$$f_d = 17.74 \text{ Hz.},$$

$$T_d = 56.3 \text{ ms}$$

For $C = 15\mu F$;

The damping factor of the system can be calculated from Eq.5.1 as

$$\xi = \frac{3000}{2} \sqrt{\frac{1.5 \times 10^{-5}}{50}} = 0.82$$

The damped frequency, ω_d , can be calculated from Eq.4.3 as

$$\alpha = \frac{R}{2L} = 30$$

$$\omega_0 = \frac{1}{\sqrt{LC}} = 36.5$$

$$\omega_d = \sqrt{\omega_0^2 - \alpha^2} = 20.82 \text{ rad/s}$$

$$f_d = 3.31 \text{ Hz.},$$

$$T_d = 301.8 \text{ ms}$$

The computational step response of the system while $C=1.5\mu\text{F}$ and $C=15\mu\text{F}$ are shown in Figure 5.3a and Figure 5.3b.

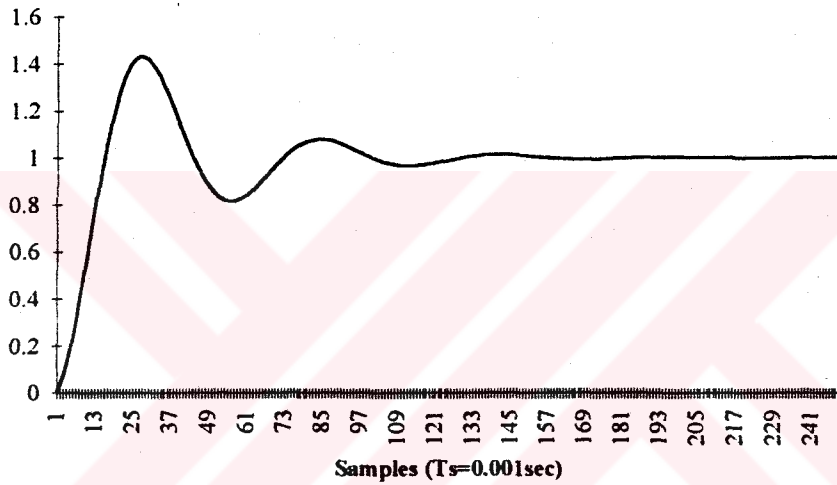


Figure 5.3a Unit step response of the plant while $C=1.5\mu\text{F}$

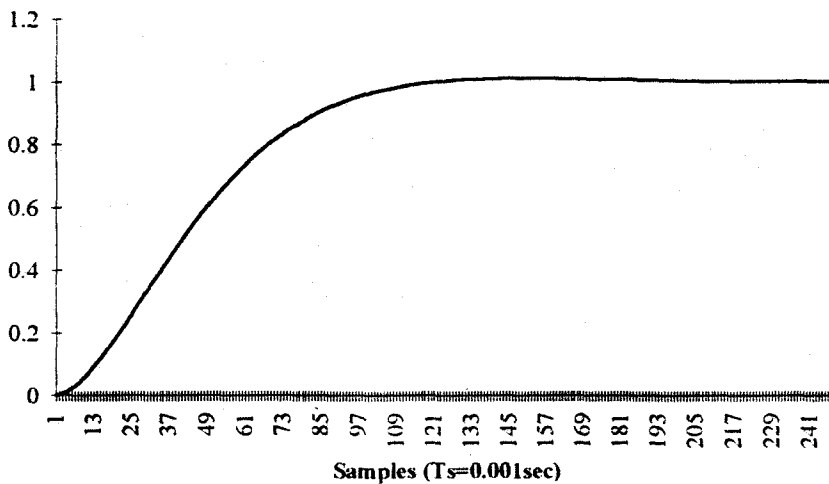


Figure 5.3b Unit step response of the plant while $C=15\mu\text{F}$

5.3 Software Structure of the System

The software of the self-tuning PID controller application is constructed by three main parts:

- Recursive estimator
- PID parameter calculator
- PID controller.

Recursive estimation process is performed by two different software which are run in the PC and the DAP800 concurrently. The flow chart of the recursive estimator subroutine of the software is shown in Figure 5.4.

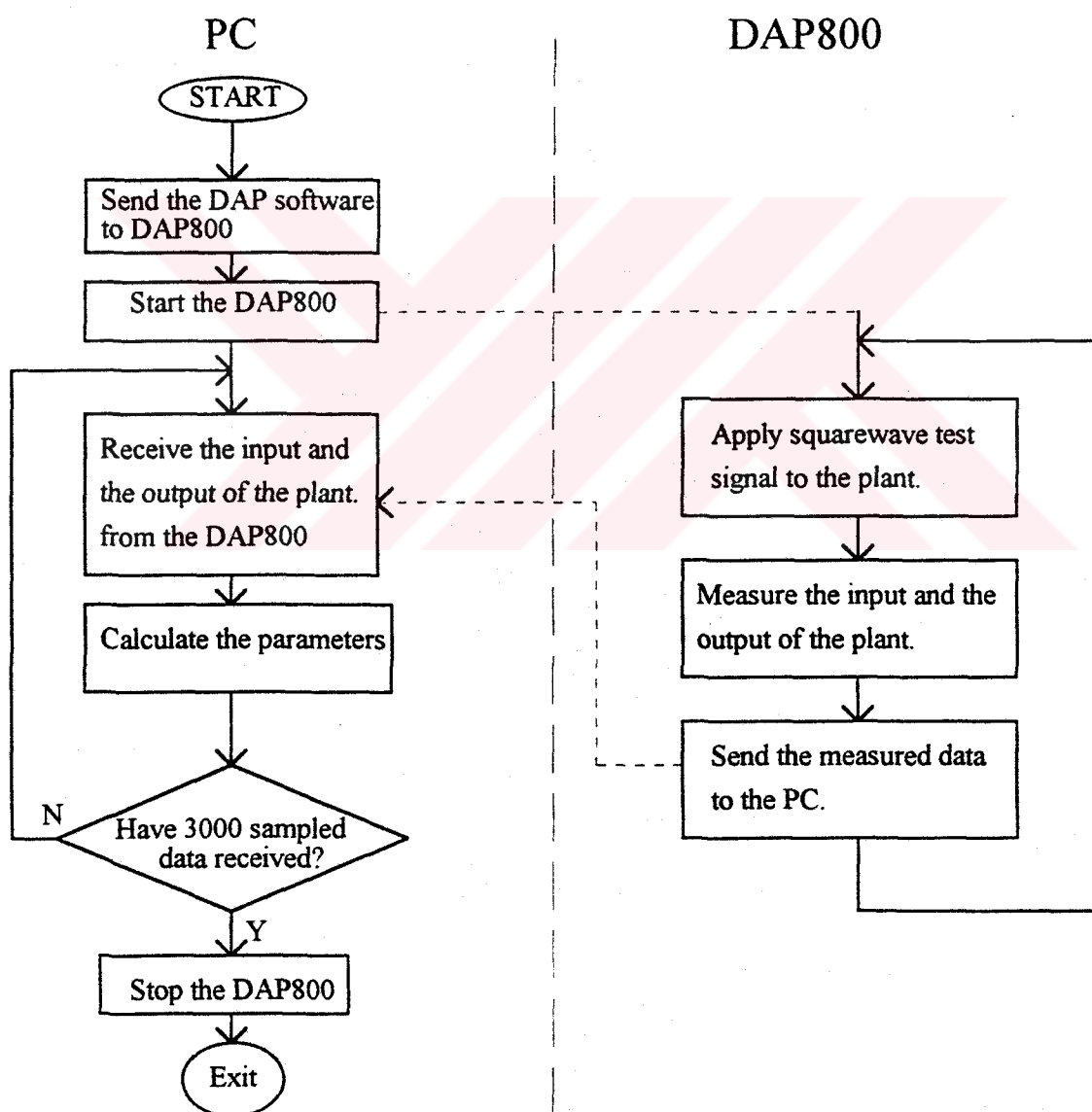


Figure 5.4 Flow chart of the recursive estimator software

RLS algorithm which was discussed in Section 3.6 is used to calculate the parameters of the plant transfer function. The initial parameters are obtained by presampling before estimation. The sampling period of the input and output is determined as 1 millisecond. The initial value of the P matrix is selected as

$$P = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

to obtain proper experimental estimation results.

PID parameter calculation is done by the PC according to the calculated plant parameters and desired overall system response criteria which are damping factor and natural frequency.

PID implementation is performed by DAP800. DAP800 is capable to provide PID function with respect to desired PID parameters. The DAP procedure which provides PID and PID parameters which are calculated in previous step are downloaded to DAP800 by the PC. The setpoint of the overall system can be sent to DAP800 by the PC at any time. The flow chart of the PID implementation software is shown in Figure 5.5

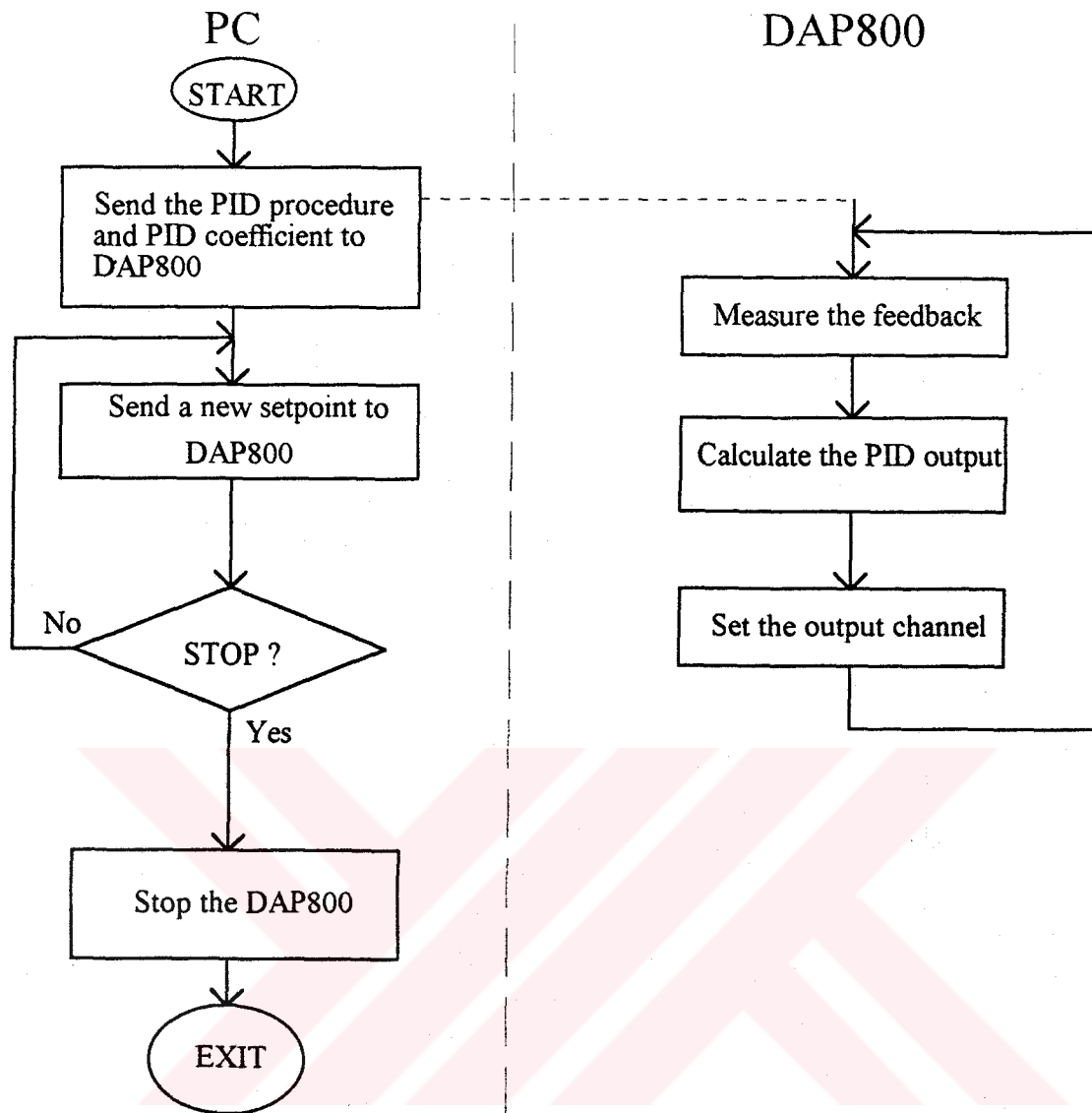


Figure 5.5 Flow chart of PID implementation software

CHAPTER SIX

APPLICATION RESULTS

6.1 Introduction

In this chapter, the experimental measurements and results of the self-tuning control system that controls the RLC circuit illustrated in Figure 5.2 are given.

6.2 Recursive Estimation Results

The squarewave test signal and the measured plant output are shown in Figure 6.1a and Figure 6.1b for $C=1.5\mu\text{F}$ and $C=15\mu\text{F}$.

The desired discrete time model of the plant is

$$y(t) = \frac{b_0 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} u(t)$$

and the estimated parameters are,

(For $C=1.5\mu\text{F}$)

$$a_1 = -1.922$$

$$a_2 = 0.934$$

$$b_0 = 0.013$$

(For $C=15\mu\text{F}$)

$$a_1 = -1.948$$

$$a_2 = 0.949$$

$$b_0 = 0.001$$

The estimation period of the parameters is shown in Figure 6.2a and Figure 6.2b.

As a result, the estimated discrete time transfer functions of these plants are,

$$y(t) = \frac{0.013 z^{-1}}{1 - 1.922 z^{-1} + 0.934 z^{-2}} u(t) \quad \text{for } C = 1.5\mu\text{F}$$

$$y(t) = \frac{0.001 z^{-1}}{1 - 1.948 z^{-1} + 0.949 z^{-2}} u(t) \quad \text{for } C = 15\mu\text{F}$$

These transfer functions can be represented in difference equation form as

$$y(t) = 1.922y(t-1) - 0.934y(t-2) + 0.013u(t-1) \quad \text{for } C = 1.5\mu\text{F}$$

$$y(t) = 1.948y(t-1) - 0.949y(t-2) + 0.001u(t-1) \quad \text{for } C = 15\mu\text{F}$$

The step responses of the estimated transfer functions are calculated to verify the estimation results by using the difference equations. These results are shown in Figure 6.3a and Figure 6.3b.

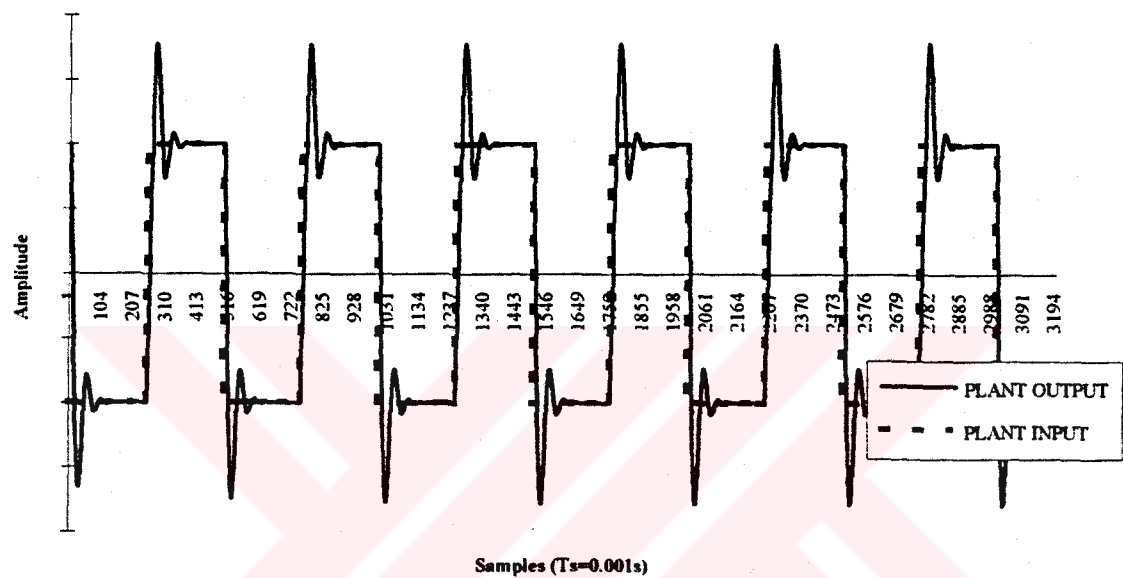


Figure 6.1a Test signal(plant input) and plant output while C=1.5μF

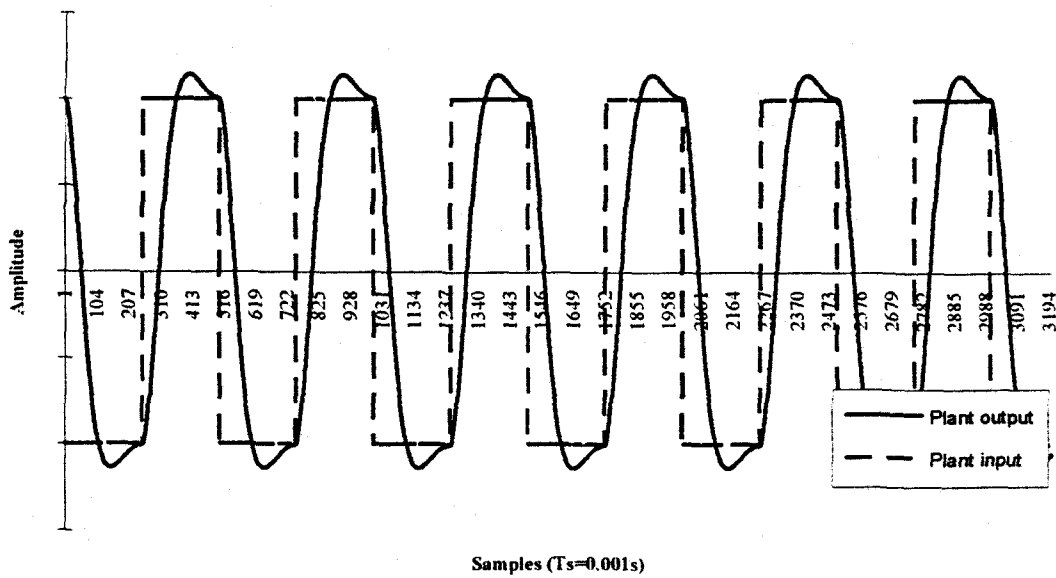


Figure 6.1b Test signal(plant input) and plant output while C=15μF

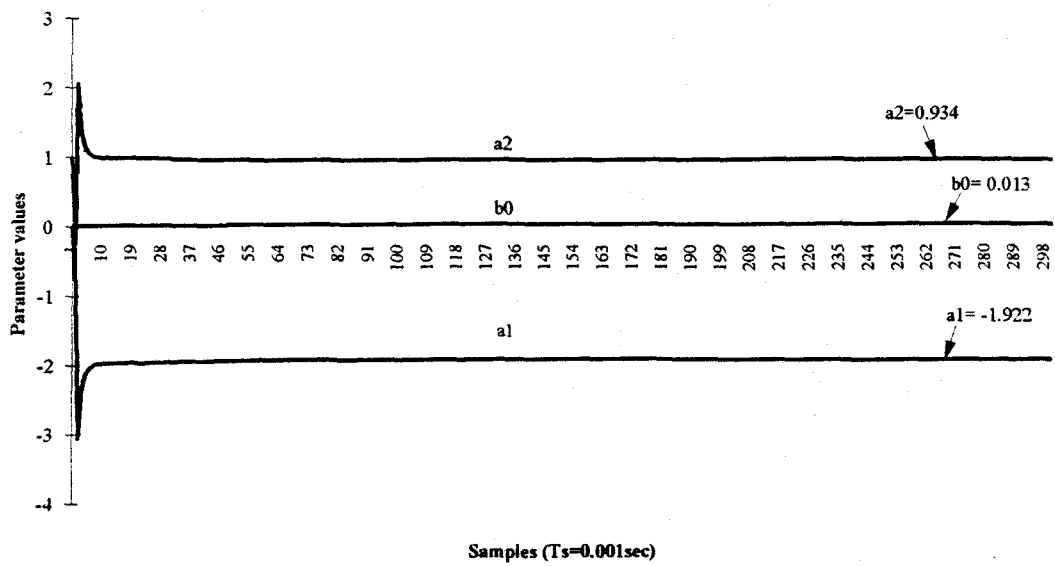


Figure 6.2a Parameter estimation for $C=1.5\mu F$.

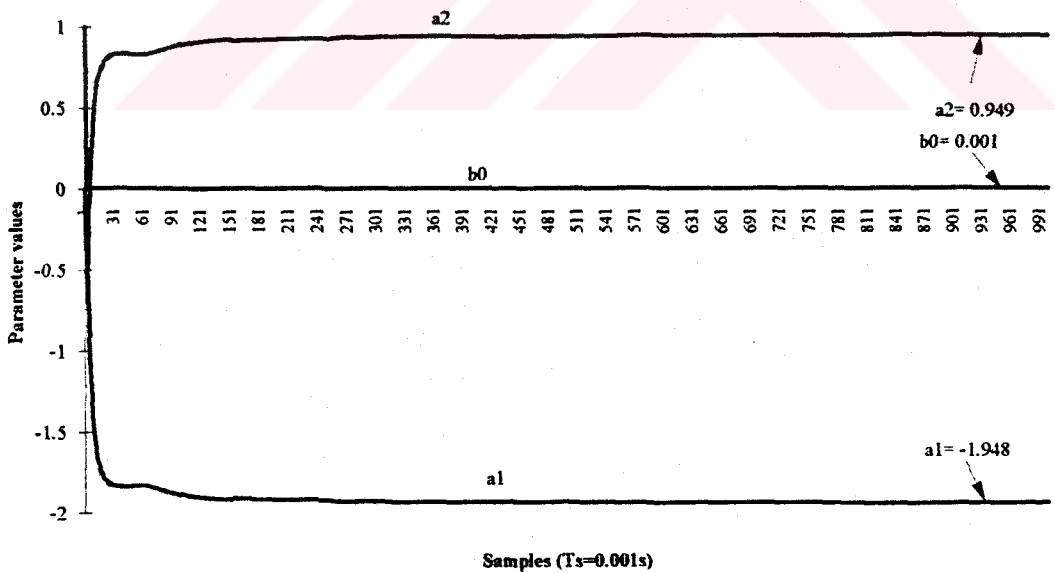


Figure 6.2b Parameter estimation for $C=15\mu F$.

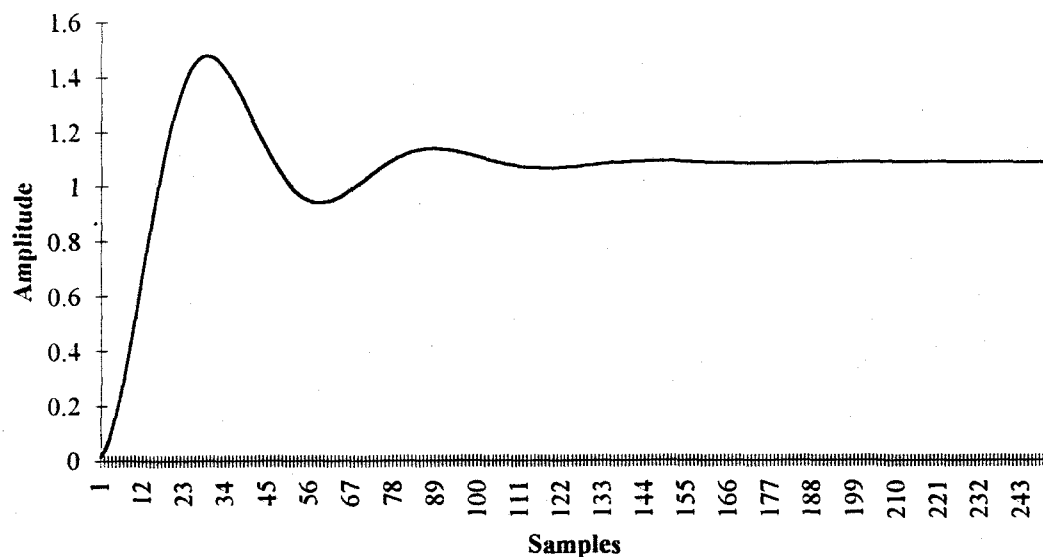


Figure 6.3a Calculated step response of the estimated transfer function for $C=1.5\mu\text{F}$

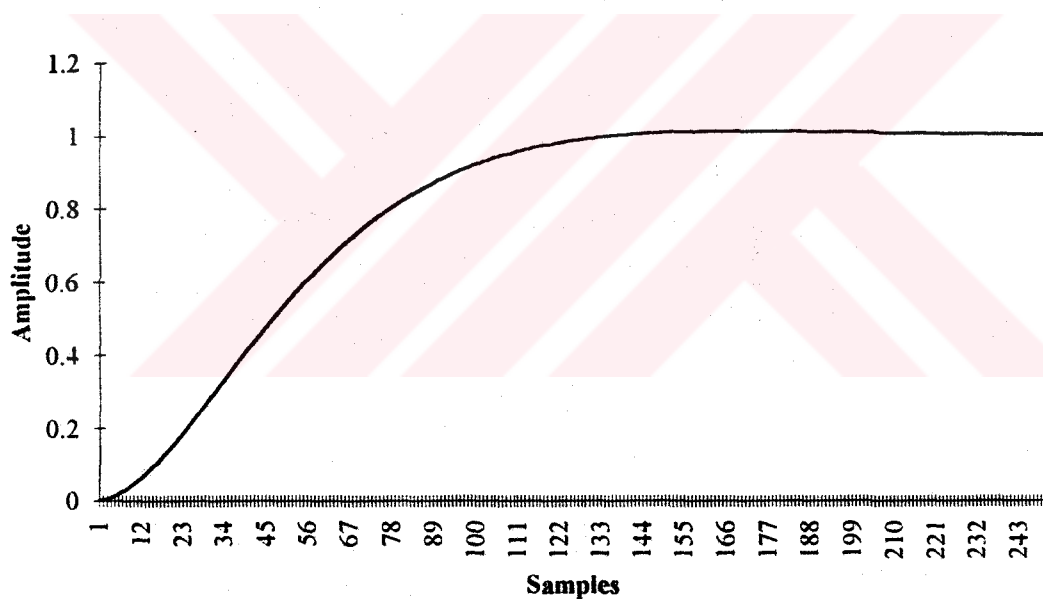


Figure 6.3b Calculated step response of the estimated transfer function for $C=15\mu\text{F}$

The step responses which are obtained from frequency domain transfer functions of the RLC circuits were shown in Figure 5.3a and Figure 5.3b. As seen clearly, step responses which are obtained from estimated parameters and frequency domain transfer functions are very similar. Therefore, estimation results are verified by the similarity between Figure 5.3 and Figure 6.3.

6.3 Self-tuning PID Implementation Results

In this section, step responses of two different RLC circuits which are controlled by the self-tuning PID controller are given for different damping factors and natural frequencies. The plant which is controlled by PID controller is shown in Figure 5.2. Different open loop transient responses of the plants are obtained by changing the value of the capacitor.

Table 6.1 shows the plant and desired overall system specifications of the measured systems. In this table, identification and PID control values of the capacitor is meaning the capacitor value of the RLC circuit (plant) while parameter estimation step and PID implementation step respectively. In normal conditions, these values are equal until the plant is forced to change by external effects. However, self-tuning system parameters should be identify again when the plant parameters are changed. Figure 6.8 and Figure 6.13 show step responses when identified system and controlled system are different.

The measurement results are illustrated in Figure 6.4 - Figure 6.13.

Table 6.1 The plant and desired overall system specifications of the measured systems.

C value while identification (μF)	C value while PID control (μF)	ξ (Damping fac.)	ω_n (rad/sec)	Figure number
15	15	0.1	10	Figure 6.4
15	15	0.8	10	Figure 6.5
15	15	0.1	3	Figure 6.6
15	15	0.8	3	Figure 6.7
15	1.5	0.1	10	Figure 6.8
1.5	1.5	0.1	10	Figure 6.9
1.5	1.5	0.8	10	Figure 6.10
1.5	1.5	0.1	3	Figure 6.11
1.5	1.5	0.8	3	Figure 6.12
1.5	15	0.1	10	Figure 6.13

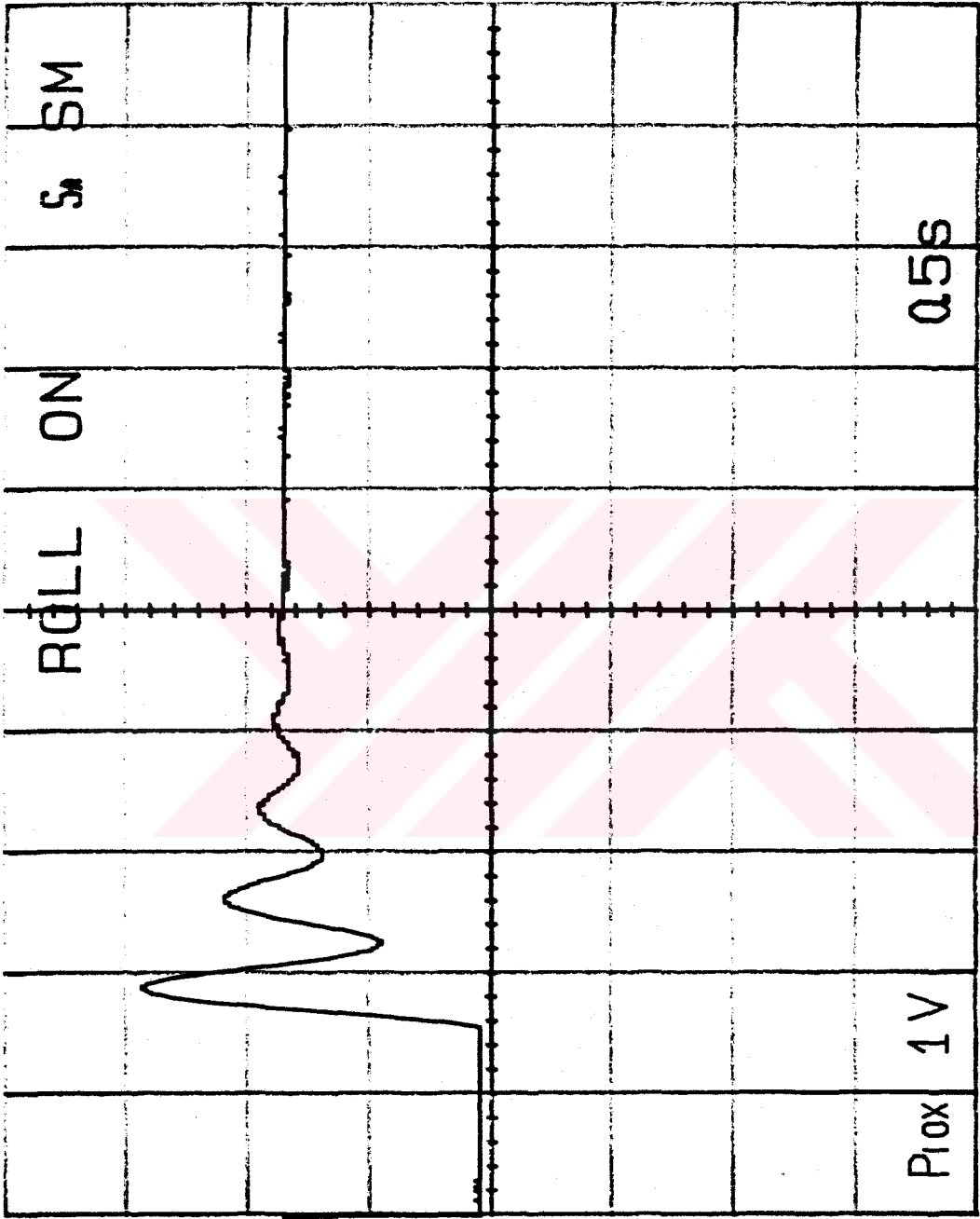


Figure 6.4 Step response of the RLC circuit while $C=15\mu\text{F}$, $\xi=0.1$, $\omega_n=10$

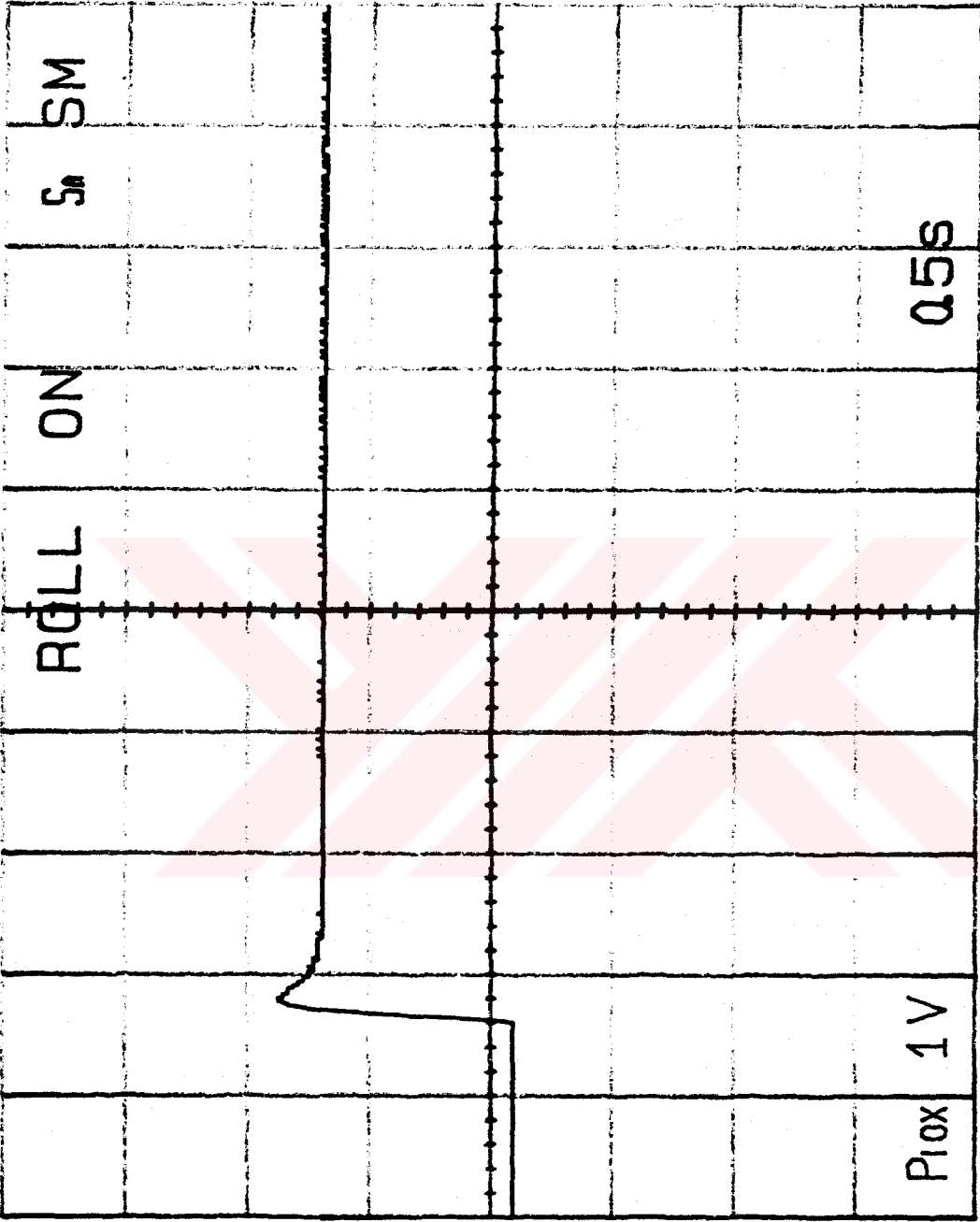


Figure 6.5 Step response of the RLC circuit while $C=15\mu\text{F}$, $\xi=0.8$, $\omega_n=10$

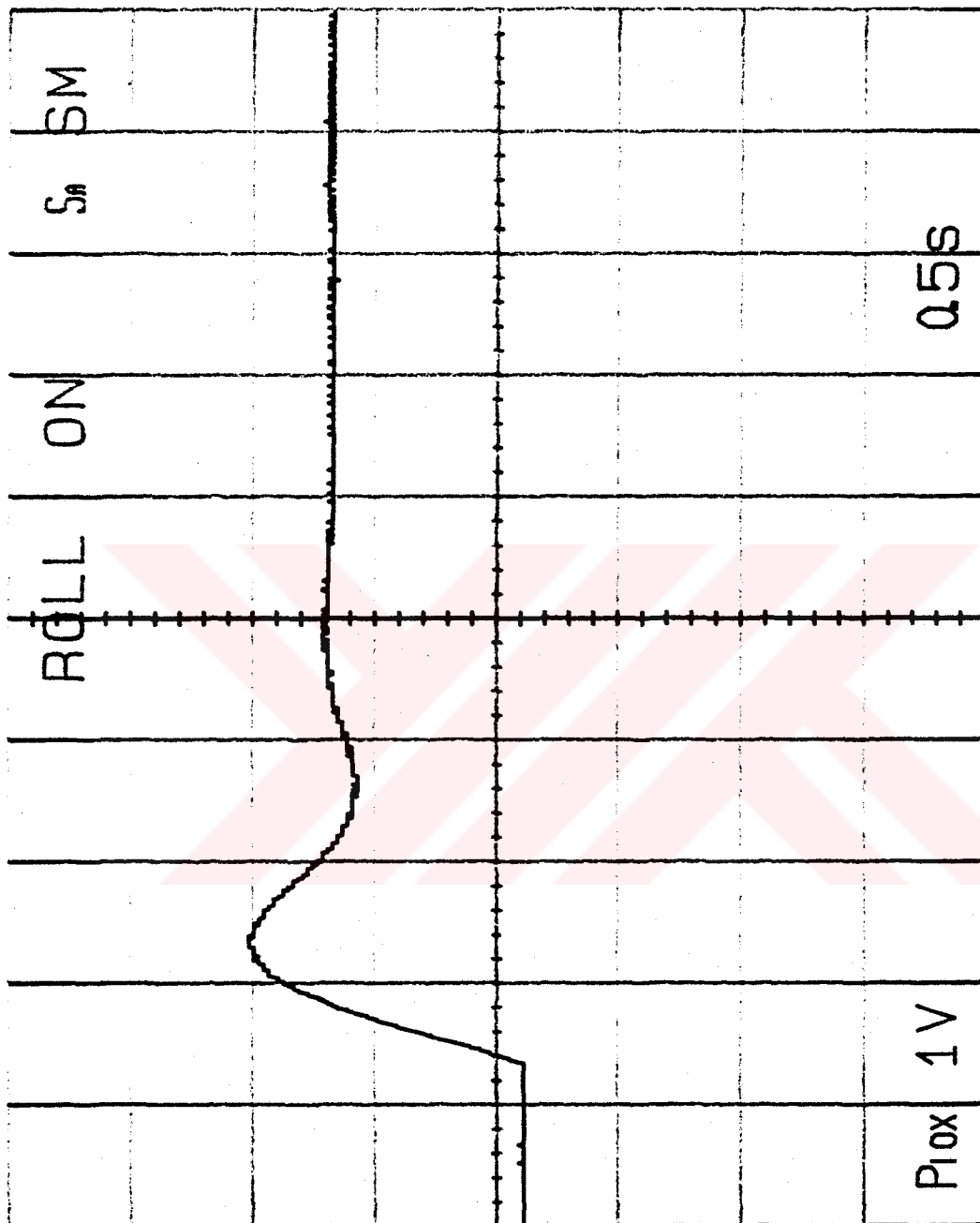


Figure 6.6 Step response of the RLC circuit while $C=15\mu\text{F}$, $\xi=0.1$, $\omega_n=3$

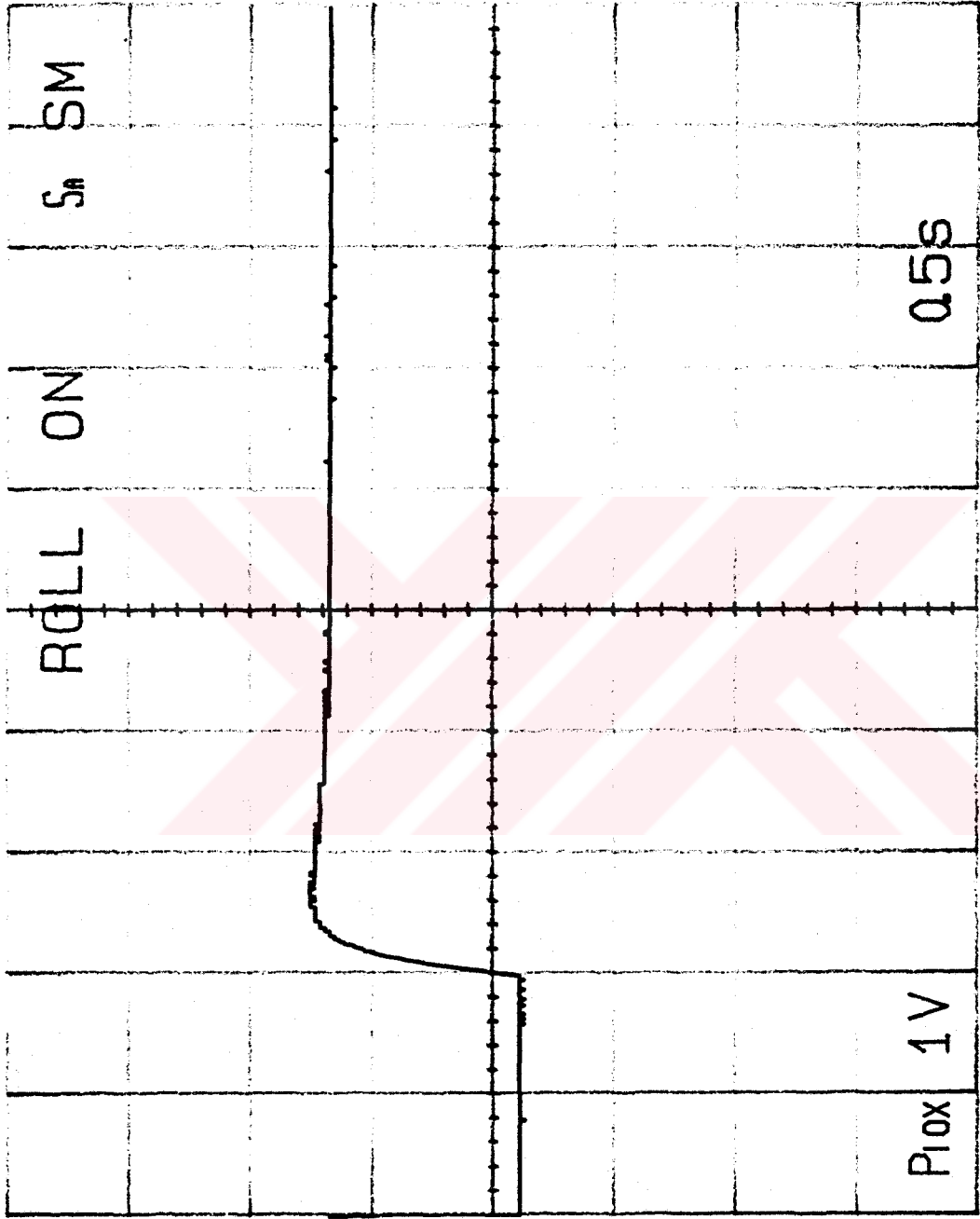


Figure 6.7 Step response of the RLC circuit while $C=15\mu\text{F}$, $\xi=0.8$, $\omega_n=3$

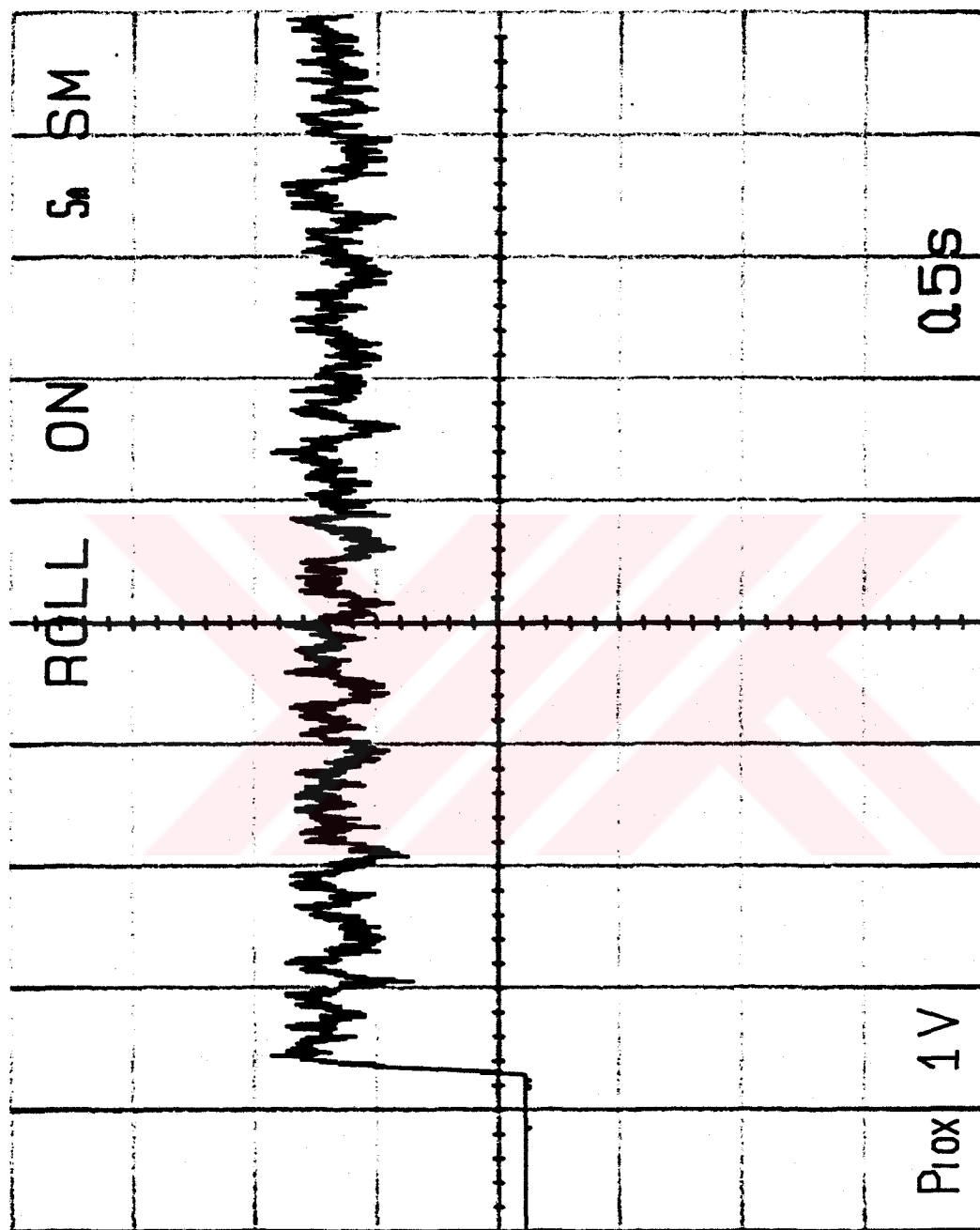


Figure 6.8 Step response of the RLC circuit while identified and controlled plants are different. ($C_{identified} = 15\mu F$, $C_{controlled} = 1.5\mu F$)

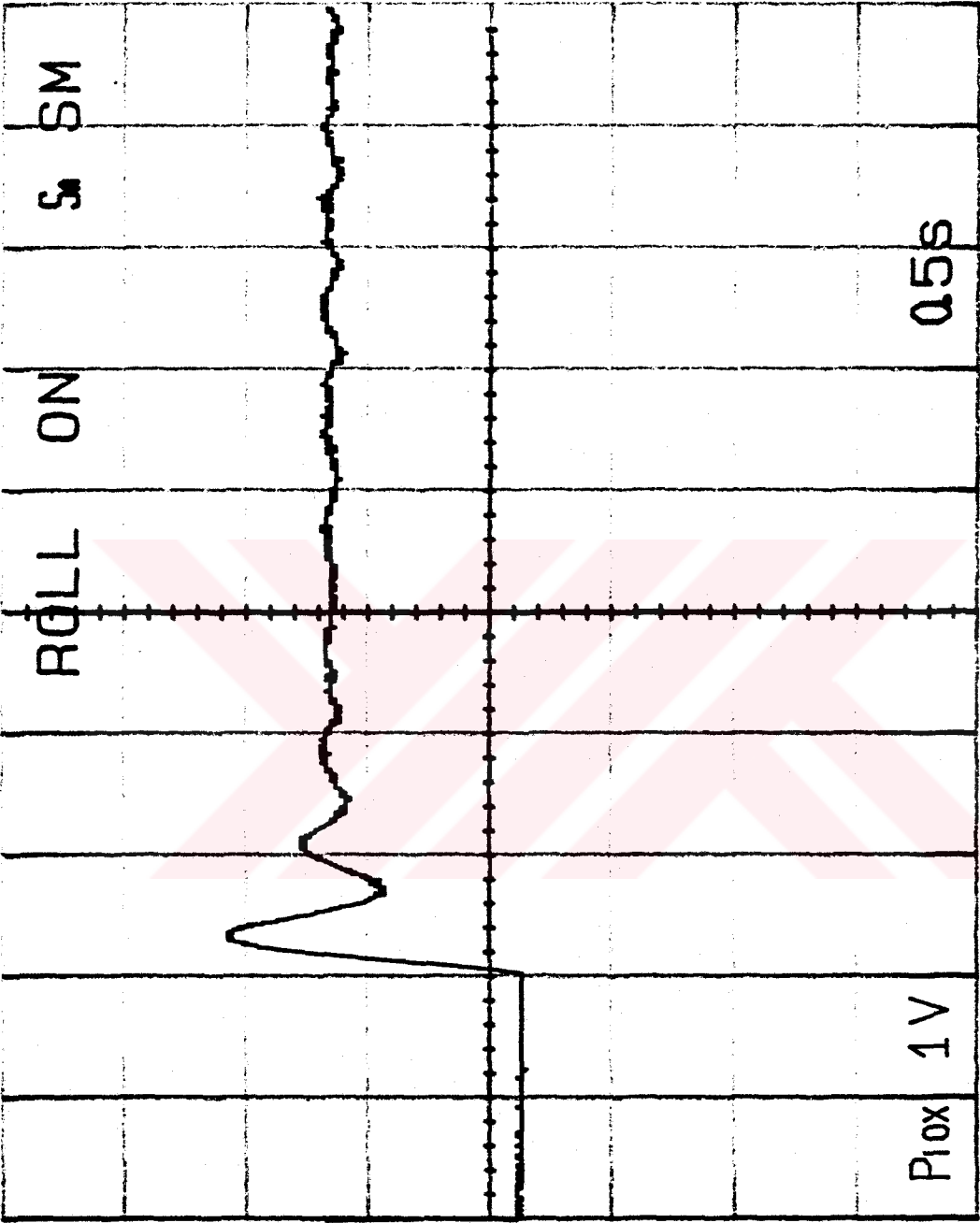


Figure 6.9 Step response of the RLC circuit while $C=1.5\mu\text{F}$, $\xi=0.1$, $\omega_n=10$.

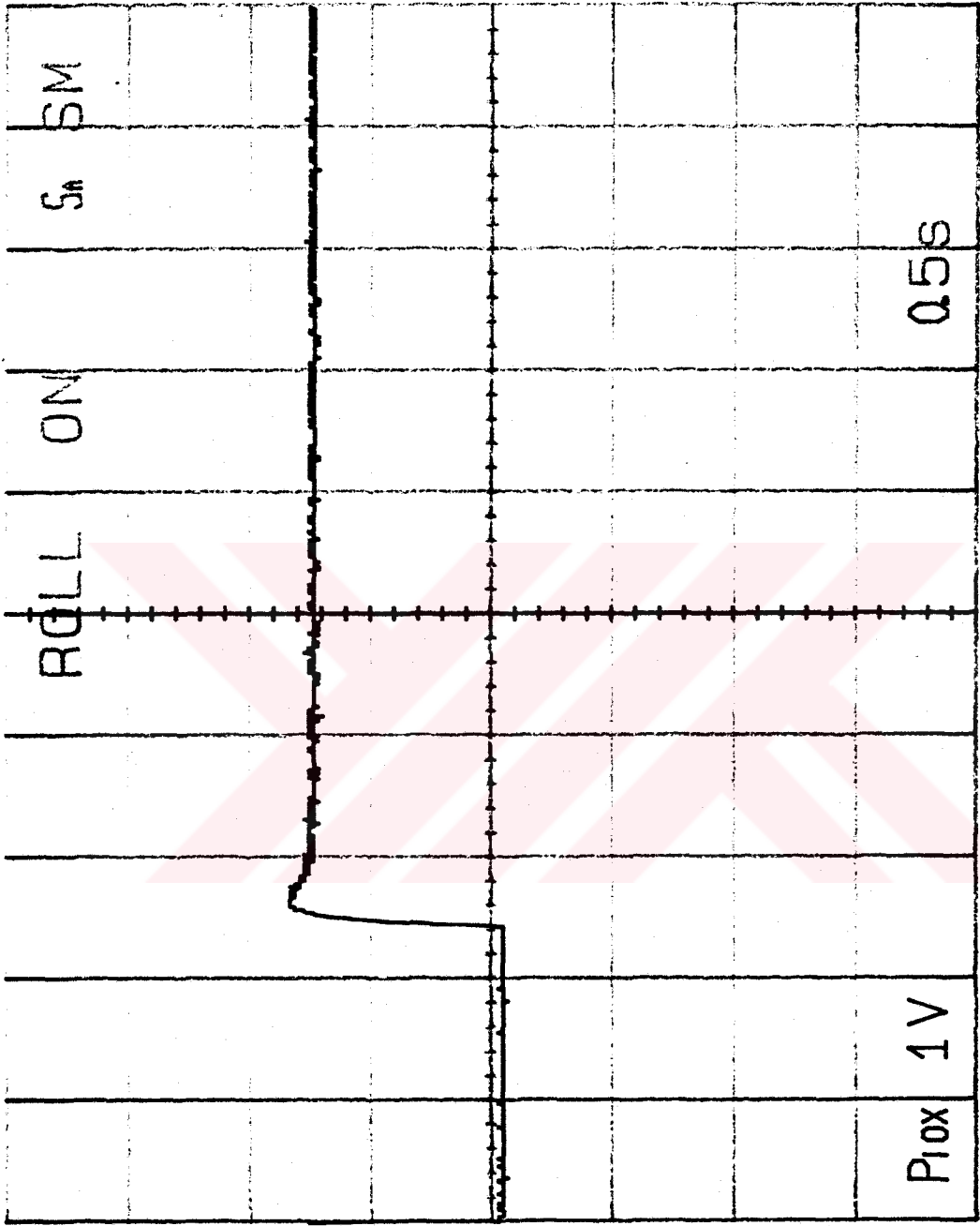


Figure 6.10 Step response of the RLC circuit while $C=1.5\mu\text{F}$, $\xi=0.8$, $\omega_n=10$.

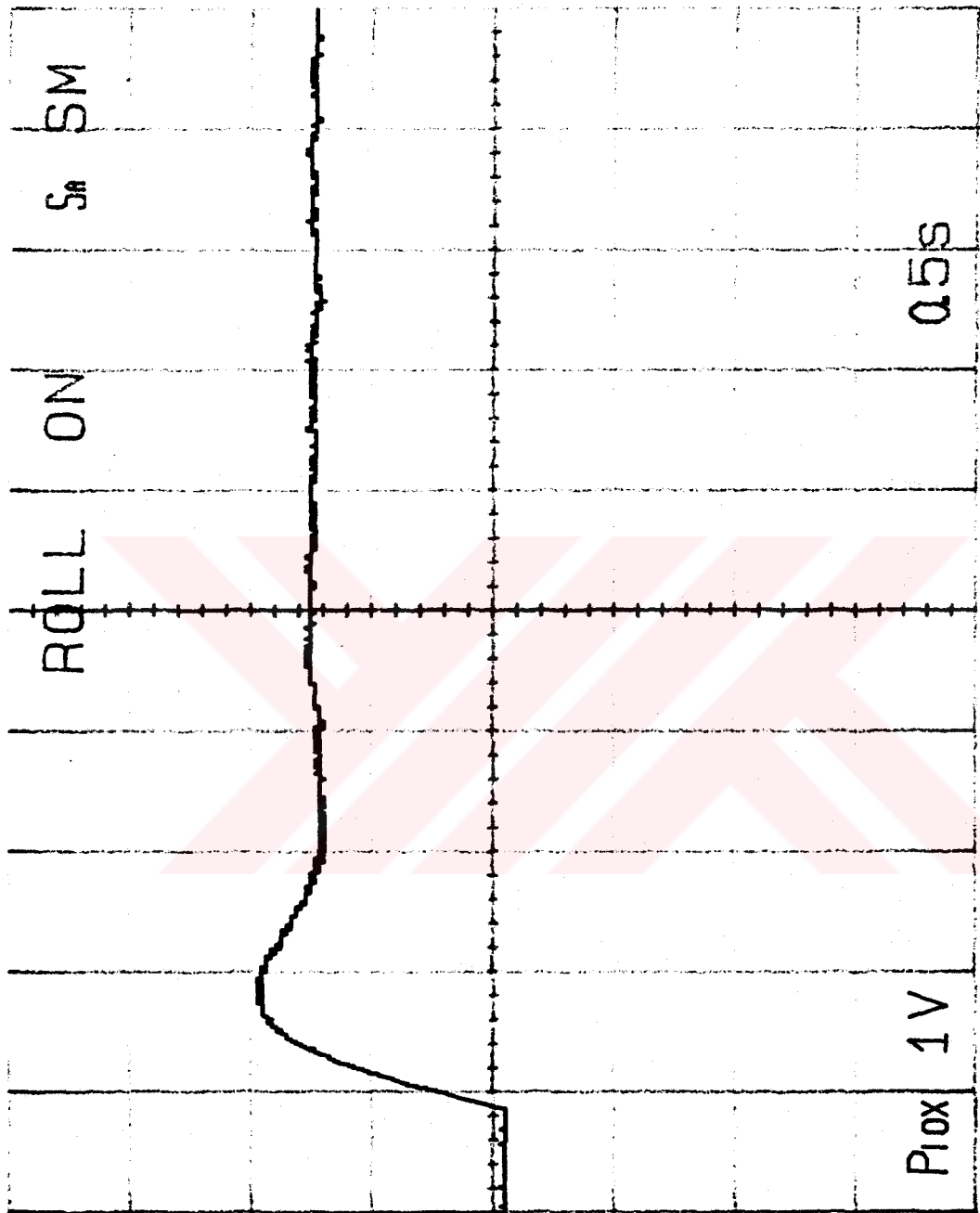


Figure 6.11 Step response of the RLC circuit while $C=1.5\mu\text{F}$, $\xi=0.1$, $\omega_n=3$.

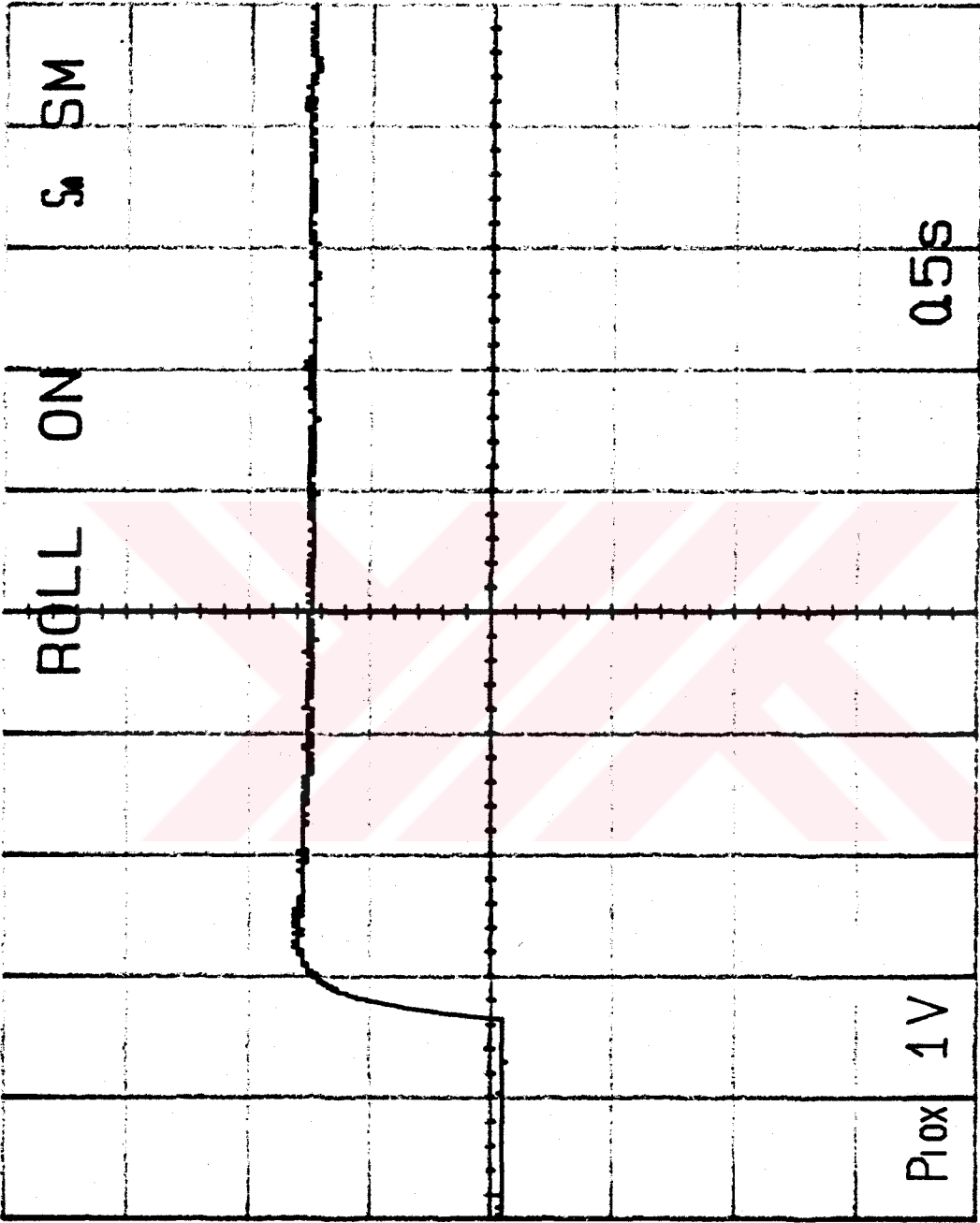


Figure 6.12 Step response of the RLC circuit while $C=1.5\mu\text{F}$, $\xi=0.8$, $\omega_n=3$.

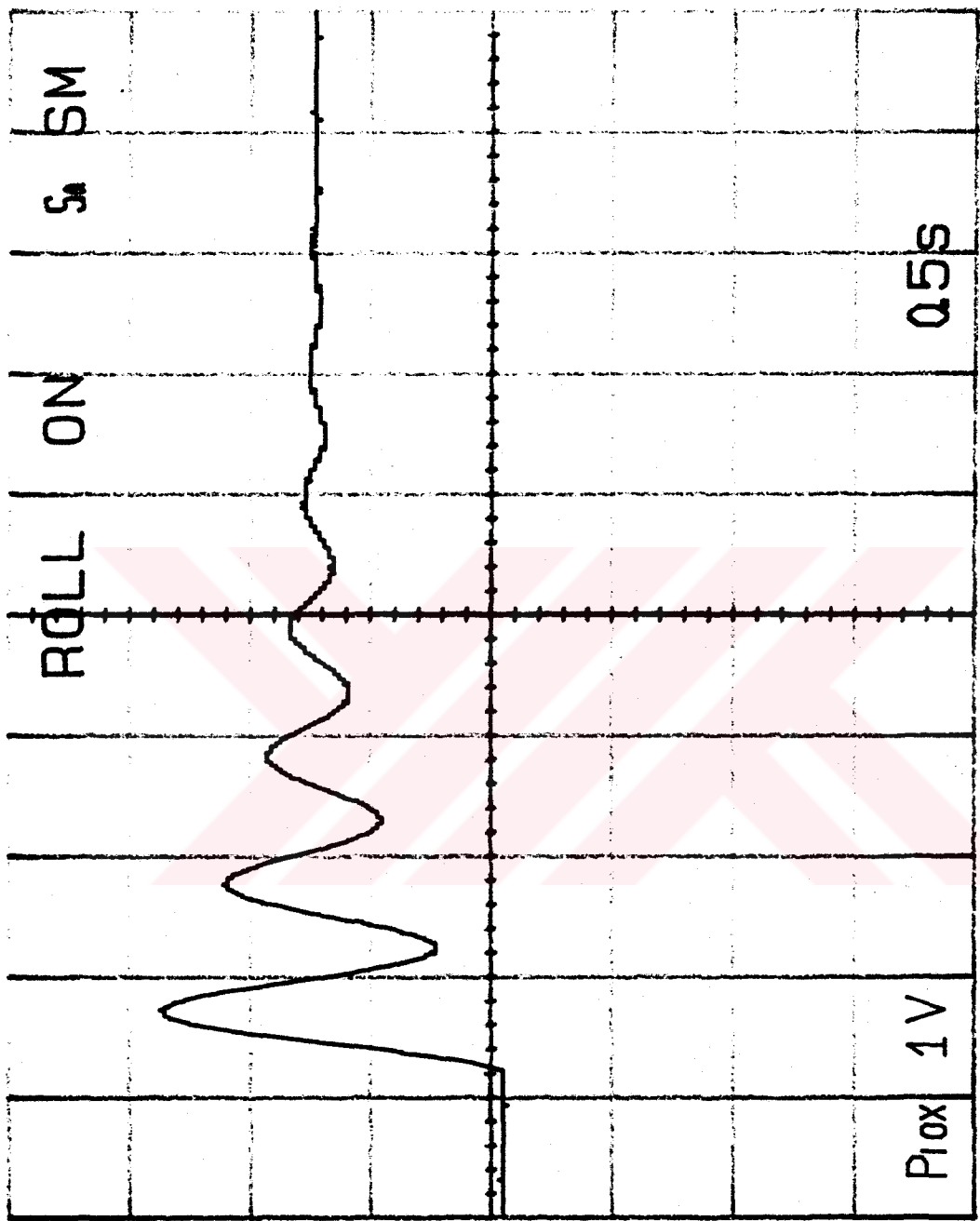


Figure 6.13 Step response of the RLC circuit while identified and controlled plants are different ($C_{identified} = 1.5\mu F$, $C_{controlled} = 15\mu F$).

As seen clearly, the measured systems which are desired to obtain same overall step responses (for example Figure 6.4 and Figure 6.9) have similar damping frequency and similar damping factors. However, if the system is controlled by an invalid parameter set as shown in Figure 6.8 and Figure 6.13, the system gives undesired response. This means that the self-tuner can estimate the plant parameters and calculate the proper PID coefficients successfully.

Weights of the PID function coefficients of DAP800 are unknown because of the PID calculation method of DAP800 is not transparent. In other words, the PID function coefficients of DAP800 are multiplied by unknown coefficients. Thus, calculated PID parameters must be scaled by other coefficients to use with DAP800. In this application, calculated PID coefficients, K_p , K_i , K_d are scaled as,

$$\begin{aligned} K_{P-DAP800} &= K_p * 0.9 \\ K_{I-DAP800} &= K_i * 1.2 \\ K_{D-DAP800} &= K_d * 0.36 \end{aligned}$$

However, these scale factors are found experimentally and they are not actual values of these coefficients. Therefore, there are small differences between the systems which are desired to obtain same overall step responses.

CHAPTER SEVEN

CONCLUSIONS

In this thesis, a self-tuning PID controller has been realized. In many industrial automatic control applications, the controlled systems contain different physical magnitudes such as temperature, speed, humidity etc. However, in this study, the controlled object was the voltage of a capacitor which was used in a serial RLC circuit. The reasons of using a RLC circuit were;

1. Calculating the transfer function and transient response of a RLC circuit is simpler than another physical system contains physical magnitudes except voltage or current. Thus comparing of the experimental results with the theoretical calculations can be done properly.
2. The response of a RLC circuit can be changed in large ranges by altering the values of R, L or C components easily.
3. There is no need to use any electrical drive unit to control the plant and any transducer to obtain feedback signal.

The realized self-tuning PID controller hardware were constructed by a data acquisition board called the DAP800 and an 80486 microprocessor based PC. In parameter estimation step of self-tuning process, the input and output of the plant were measured by the DAP800 and the parameters were calculated by the PC. All of the calculations consist of simple arithmetical operations. Therefore, the applied self tuning algorithm can be realized by a simpler microcontroller instead of 80486 based PC. Additionally, most of physical systems which are used in industrial applications have slower transient responses than the used RLC circuit. Therefore, a microcontroller which is slower than the PC can perform self-tuning PID properly. Thus, the use of self-tuning PID controllers become more practical and these controllers can be employed in wide range of industrial applications.

Recursive least squares parameter estimation method was used to obtain open-loop discrete time transfer function parameters of the RLC circuit. The estimated transfer functions are compared with the theoretical transfer functions and verified.

PID implementation stage of self-tuning controller is performed by the DAP800. In this application, the open-loop damping frequency of the RLC circuit is larger than 10Hz. The sampling frequency of the PID controller is selected as 1 Ksample/second to obtain proper

closed-loop transient and steady-state responses. However, the DAP800 can not provide PID control and another task such as communicating with PC simultaneously, because of this sampling frequency is very fast for process speed of the DAP800. So, undesired transient responses are obtained when the setpoint of the PID controller is changed by PC using the communication pipe of the DAP800. These undesired responses continue for a while and after communication process is completed, PID process continues properly.

The performance of the self-tuner was verified by comparing the desired closed-loop step responses with measured step responses of the RLC systems. Additionally, when the PID controller was tuned to obtain same natural frequency and damping factor from different RLC circuits, very similar closed-loop step responses were obtained. The difference between closed-loop step responses obtained from different RLC circuits results from the unknown scale factors - which are different from each other and estimated by using trial and error method - of PID function of the DAP800.

The designed system provides two main advantages: The first advantage is that desired second order system responses can be obtained easily without any calculation and tuning operations. It is sufficient to determine the damping factor and natural frequency of the overall system to obtain desired system response. The second benefit is that if the characteristic of the controlled system alters in time, controller can identify the new system and perform properly without any manual tuning.

REFERENCES

- Distefano, J.J. & Stubberud, A.R. & Williams, I.J. (1967). Theory and problems of feedback and control systems. Shaum Publishing.
- Dorf, R.C. (1986). Modern control systems. Addison-Wesley
- Grantham, W.J. & Vincent, T.L. (1993). Modern control systems analysis and design. Singapore: John Wiley & Sons.
- Hang, C.C & Sin, K.K (1991). On-line auto tuning of PID controllers based on the cross-correlation technique. IEEE transactions on industrial electronics, 38, 428-437
- Hayt, W.H. & Kemmerly, J.E. (1978). Engineering circuit analysis. Tokyo: McGraw-Hill Kogakusha.
- Narendra, K.S & Monopoli, R.V (1980). Applications of adaptive control. New York: Academic Press
- Neuman, C.P & Baradello, C.S. (1979). Digital transfer functions for microcomputer control. IEEE transactions on system, man, and cybernetics, 9, 856-857
- Web Site of The University of Michigan (1997). Control Tutorials for Matlab. PID.html at www.engin.umich.edu.
- Wellstead, P.E. & Zarrop, M.B (1991). Self-tuning systems control and signal processing. New York: John Wiley & Sons.

APPENDICES



Technical Product Information for the DAP 800™

The DAP 800 models

- each have an on-board Intel 80C188XL 10-MHz or 16-MHz processor.
- work with the XT/PC/AT/ISA bus for 8086/286/386/486 and Pentium PC platforms.
- transfer data at high rates: up to 105K samples per second from a DAP 800 to the PC.
- allow fast real-time processing.
- offer low latency — 1 ms per task — for fast response.
- sample analog or digital inputs at rates up to 105K samples per second.
- update two analog outputs at rates up to 105K samples per second each.
- update digital outputs at rates up to 105K samples per second.

This technical note describes all of the DAP 800 models in terms of software speed and functionality, special hardware characteristics, and similarities with other Data Acquisition Processor™ boards.

There are three DAP 800 models: the DAP 801/101, the DAP 800/102, and the DAP 800/103. Their hardware differs mostly in three areas: speed of the on-board CPU, DRAM size, and sampling rate. These specifications are compared in Table 2 — “DAP 800 Typical Hardware Specifications.”

The DAP 801/101 has the added feature of a serial connector which allows it to communicate with the PC in stand-alone mode. Ask for Microstar Laboratories Technical Note TN-158 for more information on configuring the DAP 801/101 in stand-alone mode.

The DAP 800 is the lowest-priced Data Acquisition Processor board available from Microstar Laboratories, and is appropriate for intelligent data acquisition and control applications where cost is important. With the DAP 800, Microstar Laboratories provides intelligent data acquisition and processing at the cost of a non-intelligent board. The DAP 800 provides all the standard DAPL™ commands available on other boards, performing them at rates appropriate for lower speed applications.

The on-board multi-tasking operating system, DAPL, is a complete software environment for real-time data acquisition. DAPL is common to all Data Acquisition Processors and ensures that board-level hardware differences are transparent. To aid application development, DAPL comes complete with many system diagnostics, in addition to automatic memory and system checks that are done at initialization. Tasks that perform averaging, triggering, FFTs, filtering, arithmetic operations, and many other functions are pre-coded in DAPL. These tasks, or DAPL commands, are chained together to form a complete data acquisition application. Custom commands also can be written with the Advanced Development Toolkit if multiple commands need to be combined or if a specific application cannot be implemented with standard DAPL commands.

Another common element shared by the DAP 800 series is the bus interface. The DAP 800 models work with both XT and AT ISA busses for 286/386/486 and Pentium PC platforms. 256-byte first-in-

first-out (FIFO) buffers allow fast data transfer to the host PC. For example, the DAP 800/102 and DAP 800/103 can transfer information to the PC at rates as high as 105K samples per second.

The main feature of the DAP 800 is its ability to solve applications at a low-cost. The DAP 800 is an excellent choice for applications where there is a need for moderate real-time triggering, averaging, control, interpolation, or many other functions, but no need for high-speed FFTs or other computationally intensive operations. Table 1 gives information about the execution speed of DAPL commands on the DAP 800. For higher power applications, any of the DAP 1200e™, DAP 2400e™, DAP 1216e™, DAP 2416e™, or DAP 3200e™ series may be appropriate. Contact Microstar Laboratories for more information on these products.

Table 1: DAPL command execution speed for the DAP 800 series

DAPL Command	Description	Time of Execution ¹ on DAP 801/101	Time of Execution on DAP 800/102 or DAP 800/103
AVERAGE	Averages groups of 16 data points ²	345.6 μs	152 μs
FFT	FFT of blocksize of 512 points	463 ms	247 ms
RFILTER	Filters input data with 20 tap filter	580.4 μs	316 μs
LIMIT	Generates level based triggers on 1% of data	12.4 μs	7 μs
WAIT	Processes data based upon triggers at a retention rate of 5 out of 100 samples	14 μs	6 μs
DAPL Expression: P3 = P1 + P2	Adds two word-length pipe values together	174.4 μs	64 μs

In addition to its processing capabilities, the DAP 800 provides a complete arrangement of analog and digital input and output sections. The analog input section is expandable—up to 32 single-ended or 16 differential inputs. See Table 2 for more information.

Data is sent or received by the DMA controller of the 80C188XL at a rate of up to 105K samples per second. This data is clocked at a sampling rate or output rate controlled in software, but the actual rate is accurately set by on-board crystal-controlled timers. The sample period is specified in steps as small as a quarter of a microsecond. The length of every sample period is accurate to 50 parts per million.

In addition to on-board timing, the DAP 800 also has provisions for an external input trigger and an external clock input for input and output.

¹ The speed given is an actual application speed for the DAPL task, including sampling, DAPL task-switching and activation, and simulated transfer time. Kernel speeds for the tasks are actually faster.

² The speed given is for the complete block operation, if applicable. For a per value speed, the time of execution must be divided by the block size.

To clarify the operation for the various hardware sections, Figure 1 displays the architecture of the internal processing in the DAP 800.

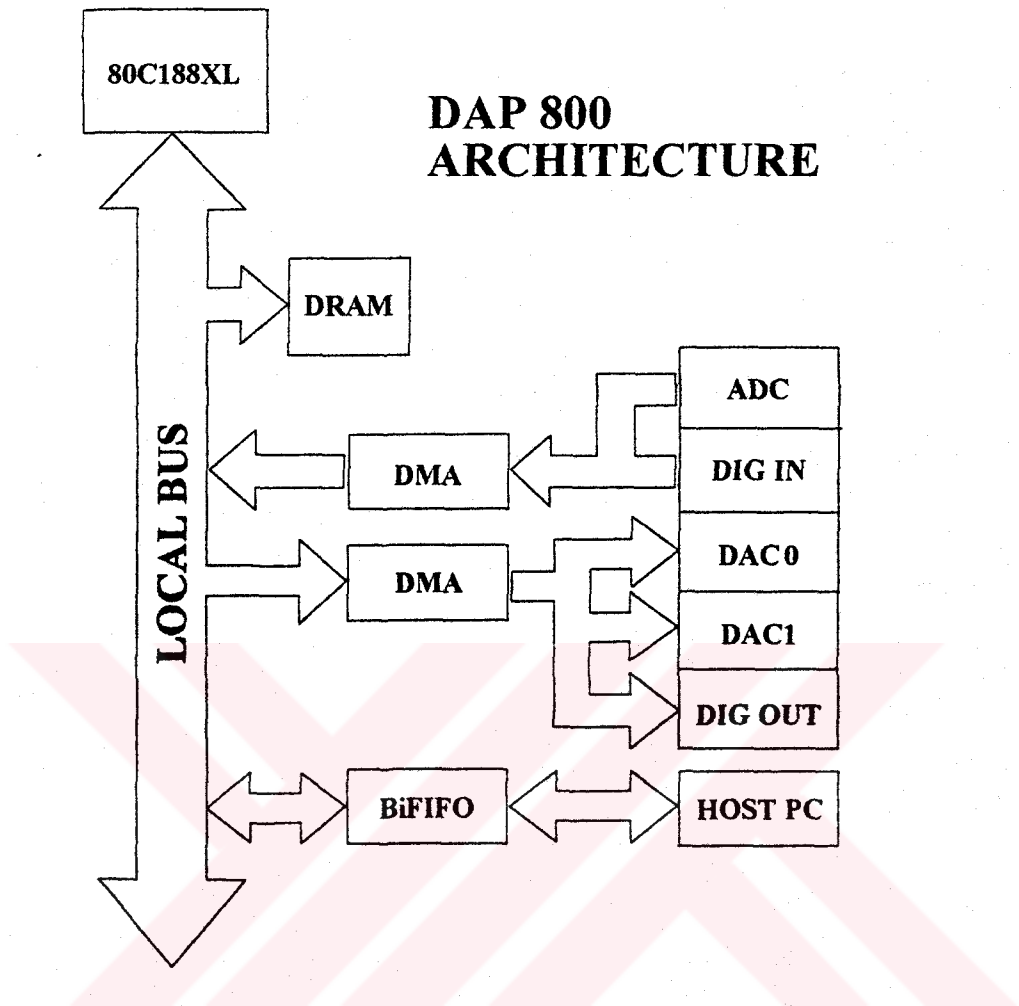


Figure 1: DAP 800 Data Acquisition Hardware

The 80C188XL processor, shown in Figure 1, performs the operations necessary for data acquisition and control. The CPU resides on the local DAP 800 bus and directs all data transfers. For instance, data from the analog and digital inputs are sent via DMA transfers to the on-board DRAM memory. From there it can be processed by the CPU, transferred to the PC, and/or directed via DMA to the output section.

Transfer of data and other communication to the PC is handled by a FIFO buffer. Information can be exchanged with the PC in both directions simultaneously and can be either DAPL programs, binary or text data, error messages, or DAPL system commands. This communication method is not only faster than DMA, but allows multiple Data Acquisition Processors to share one interrupt line. In this way, up to 14 Data Acquisition Processors can control and acquire data in one PC.

In addition to the processor and data transfer hardware, some important hardware specifications of the DAP 800 are given in Table 2 on the following pages.

Table 2: DAP 800 Typical Hardware Specifications

Specification	DAP 801/101	DAP 800/102	DAP 800/103
Dimensions	13.37" x 4.2"	13.37" x 4.2"	13.37" x 4.2"
Weight	9.1 oz	9.1 oz	9.1 oz
PU Type	Intel 80C188XL	Intel 80C188XL	Intel 80C188XL
PU Clock Speed	10 MHz	16 MHz	16 MHz
PU DRAM	256 Kbytes	256 Kbytes	1 Mbyte
Data Acquisition Mode	DMA	DMA	DMA
Bus Support ³	XT, AT	XT, AT	XT, AT
PC Interface Hardware	256 byte FIFO	256 byte FIFO	256 byte FIFO
PC Transfer Mode	I/O Interrupt	I/O Interrupt	I/O Interrupt
Maximum Transfer Rate ⁴	75K samples/sec	105K samples/sec	105K samples/sec
Power Requirements	+5V, 2.0 Amps	+5V, 2.0 Amps	+5V, 2.0 Amps
Operating Temperature	0-50 °C	0-50 °C	0-50 °C
Accuracy of Crystal Clocks	50 parts per million	50 parts per million	50 parts per million
Type of A⇒D Converter	Successive Approximation	Successive Approximation	Successive Approximation
Model of A⇒D Converter	Maxim MAX163	Maxim MAX163	Maxim MAX163
Max. Analog Sampling at			
Gain = 1	75 K samples/sec	105 K samples/sec	105 K samples/sec
Gain = 10	75 K samples/sec	100 K samples/sec	100 K samples/sec
Gain = 100	25 K samples/sec	25 K samples/sec	25 K samples/sec
Gain = 500	2 K samples/sec	2 K samples/sec	2 K samples/sec
Number of Channels	8	8	8
Expandable To	32	32	32
Analog Input Voltage Ranges	-2.5 to +2.5 V 0 to 5 V -5 to 5 V -10 to 10 V	-2.5 to +2.5 V 0 to 5 V -5 to 5 V -10 to 10 V	-2.5 to +2.5 V 0 to 5 V -5 to 5 V -10 to 10 V
Resolution	12 bits	12 bits	12 bits
of Range is -5 to 5 Volts	2.4 mV	2.4 mV	2.4 mV
Accuracy	±1 LSB	±1 LSB	±1 LSB
of Range is -5 to 5 Volts	±2.4 mV	±2.4 mV	±2.4 mV
Analog Input Bias Current	12 nA	12 nA	12 nA
Analog Input Impedance	>> 10 MΩ	>> 10 MΩ	>> 10 MΩ

³ The DAP 801/101 also can communicate in stand-alone mode via an RS-232 connection.

⁴ When used in stand-alone mode, the DAP 801/101 can transfer data at a maximum rate of 100 samples per second.

Table 2: DAP 800 Typical Hardware Specifications cont.

Specification	DAP 800/1 or DAP 801/1	DAP 800/2	DAP 800/3
Common Mode Rejection	90 dB	90 dB	90 dB
Max. Input Voltage	±25 V	±25 V	±25 V
Type of D⇒A Converter	Voltage Output	Voltage Output	Voltage Output
Model of D⇒A Converter	Burr-Brown DAC811	Burr-Brown DAC811	Burr-Brown DAC811
Maximum Update Rate	75K updates/sec	105K updates/sec	105K updates/sec
Number of Channels	2	2	2
Output Ranges	0 to 10 V -5 to 5 V -10 to 10 V	0 to 10 V -5 to 5 V -10 to 10 V	0 to 10 V -5 to 5 V -10 to 10 V
Resolution	12 bits	12 bits	12 bits
If Range is -5 to 5 volts	2.4 mV	2.4 mV	2.4 mV
Accuracy	±1 LSB,	±1 LSB,	±1 LSB,
If Range is -5 to 5 volts	±2.4 mV	±2.4 mV	±2.4 mV
Output Impedance	0.2 Ω	0.2 Ω	0.2 Ω
Current Source Maximum	±1 mA	±1 mA	±1 mA
Digital Input/Output Logic	ALS TTL	ALS TTL	ALS TTL
Max. Digital Update Rate	75K words/sec	105K words/sec	105K words/sec
Number of Input Bits	8	8	8
Number of Output Bits	8	8	8
Digital Input			
Min. Logical High	2 V	2 V	2 V
Max. Logical Low	0.8 V	0.8 V	0.8 V
Max. Current Sink	20 µA	20 µA	20 µA
Max. Current Source	20 µA	20 µA	20 µA
Digital Output			
Min. Logical High	2.6 V	2.6 V	2.6 V
Max. Logical Low	0.5 V	0.5 V	0.5 V
Max. Current Sink	24 mA	24 mA	24 mA
Max. Current Source	2.6 mA	2.6 mA	2.6 mA
Hardware Clock	25 ns	25 ns	25 ns
Min. Pulse Width			
Hardware Trigger	60 ns	60 ns	60 ns
Min. Pulse Width			
Trigger Modes	GATED ONE-SHOT	GATED ONE-SHOT	GATED ONE-SHOT

// APPENDIX 2

```

/*****
RECURSIVE ESTIMATION AND POLE-ASSIGNMENT CONTROL SOFTWARE
*****/

```

```

#include<stdio.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
#include<fcntl.h>
#include<dos.h>

```

```

#include<dapiocd.c>
#include<clock.c>
#include<c_lib.c>
#include<dapio.c>
#include<cfgdap.c>

```

```
FILE *DapBinIn, *DapTextOut, *DapTextIn, *DapBinOut;
```

```
// X[i][j]=X[3][1]
```

```
void XxXT(float x[4],float result[4][4])
```

```
{
    result[1][1]= x[1] * x[1] ;
    result[1][2]= x[1] * x[2] ;
    result[1][3]= x[1] * x[3] ;

```

```

    result[2][1]= x[2] * x[1] ;
    result[2][2]= x[2] * x[2] ;
    result[2][3]= x[2] * x[3] ;

```

```

    result[3][1]= x[3] * x[1] ;
    result[3][2]= x[3] * x[2] ;
    result[3][3]= x[3] * x[3] ;

```

```
}
```

```
void mux3x3(float A[4][4],float B[4][4],float result[4][4])
```

```
{
    result[1][1]= A[1][1]*B[1][1] + A[1][2]*B[2][1] + A[1][3]*B[3][1];
    result[1][2]= A[1][1]*B[1][2] + A[1][2]*B[2][2] + A[1][3]*B[3][2];
    result[1][3]= A[1][1]*B[1][3] + A[1][2]*B[2][3] + A[1][3]*B[3][3];

```

```

    result[2][1]= A[2][1]*B[1][1] + A[2][2]*B[2][1] + A[2][3]*B[3][1];
    result[2][2]= A[2][1]*B[1][2] + A[2][2]*B[2][2] + A[2][3]*B[3][2];
    result[2][3]= A[2][1]*B[1][3] + A[2][2]*B[2][3] + A[2][3]*B[3][3];

```

```

result[3][1]= A[3][1]*B[1][1] + A[3][2]*B[2][1] + A[3][3]*B[3][1];
result[3][2]= A[3][1]*B[1][2] + A[3][2]*B[2][2] + A[3][3]*B[3][2];
result[3][3]= A[3][1]*B[1][3] + A[3][2]*B[2][3] + A[3][3]*B[3][3];

```

```

}

```

```

void mux13_33(float A[4],float B[4][4],float result[4])

```

```

{
    result[1]= A[1]*B[1][1] + A[2]*B[2][1] + A[3]*B[3][1];
    result[2]= A[1]*B[1][2] + A[2]*B[2][2] + A[3]*B[3][2];
    result[3]= A[1]*B[1][3] + A[2]*B[2][3] + A[3]*B[3][3];

```

```

}

```

```

void mux33_31(float A[4][4],float B[4],float result[4])

```

```

{
    result[1]= A[1][1]*B[1] + A[1][2]*B[2] + A[1][3]*B[3];
    result[2]= A[2][1]*B[1] + A[2][2]*B[2] + A[2][3]*B[3];
    result[3]= A[3][1]*B[1] + A[3][2]*B[2] + A[3][3]*B[3];

```

```

}

```

```

float mux13_31(float A[4],float B[4])

```

```

{
    return A[1]*B[1] + A[2]*B[2] + A[3]*B[3];
}

```

```

void equal3x3(float source[4][4],float destination[4][4])

```

```

{
    int i,j;

    for(i=1;i<4;++i) for(j=1;j<4;++j) destination[i][j]=source[i][j];

```

```

}

```

```

void DapRun1()

```

```

{
    int i, y1, y2;

    if(((DapBinIn = fopen("ACCEL1","rb")) == NULL) ||
        ((DapTextOut = fopen("ACCELO","wt")) == NULL))
    {
        printf("Error opening DAP device driver\n");
        exit(1);
    }

```

```

    fWriteIoCtlStr(DapBinIn,"S,M00");

```

```

    fprintf(DapTextOut, "RESET\n"); /* Send a command to reset the DAP */

// fFlushDap(DapTextIn);
fFlushDap(DapBinIn);

if (fConfigDap(DapTextOut, "TEZ_DENE.DAP") >= 200)
{
    printf("Error while configuring DAP\n");
    exit(2);
}

sleep(1);

fprintf(DapTextOut, "LET TEST_PER = 500\n");
fprintf(DapTextOut, "EMPTY OPIPE0\n");
fprintf(DapTextOut, "EMPTY IP0\n");
fprintf(DapTextOut, "EMPTY IP1\n");
fprintf(DapTextOut, "START OA,IA,PA\n");
}

void DapRun2()
{
    int i, y1, y2;

    if (((DapBinIn = fopen("ACCEL1", "rb")) == NULL) ||
        ((DapTextOut = fopen("ACCEL0", "wt")) == NULL))
    {
        printf("Error opening DAP device driver\n");
        exit(1);
    }

    fWriteIoCtlStr(DapBinIn, "S,M00");

    fprintf(DapTextOut, "RESET\n"); /* Send a command to reset the DAP */

    fFlushDap(DapTextIn);
    fFlushDap(DapBinIn);

    if (fConfigDap(DapTextOut, "PID.DAP") >= 200)
    {
        printf("Error while configuring DAP\n");
        exit(2);
    }

    sleep(1);
}

```

```

void DapStop()
{
    fprintf(DapTextOut, "STOP\n"); /* Send a command to stop the DAP */
    fWriteIoCtlStr(DapBinIn, "R"); /* Restore ACCEL device driver mode */

    fclose(DapBinIn); /* Close files */
    fclose(DapTextOut);
    fclose(DapTextIn);
}

```

```

void main()
{
    FILE *fp;

    float x[4],t[4],P[4][4],R1[4][4],R2[4][4],R3[4];
    float y_1,y_2,u_1,y,u;
    float E,denominator;
    int xx,yy,y_factor;
    int i,j,k=0;
    int gdriver = DETECT, gmode, errorcode;

    int KP,KI,KD,ch;

    float zeta,Wn,Ts;
    float t1,t2,scale;

    float g0,g1,g2;

    //initial conditions

    clrscr();

    if((fp=fopen("estimate.xls","wt"))==NULL) exit(-1);

    for(i=1;i<4;++i) for(j=1;j<4;++j)
    {
        if(i==j) P[i][j]=0; else P[i][j]=5;
    }

    t[1]=0;t[2]=0;t[3]=1; //initial values of tetha vector

    printf("zeta=");scanf("%f",&zeta);
    printf("Wn="); scanf("%f",&Wn);

    Ts=0.001;
    y_factor=1;

```

```

k=0;

initgraph(&gdriver, &gmode, "C:\\COMPS\\TCPP\\BGI\\");

errorcode = graphresult();

if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

setbkcolor(GREEN);
cleardevice();
setcolor(BLUE);

line(0,getmaxy()/2,getmaxx(),getmaxy()/2);

DapRun1();

y=y_1=y_2=1.0;
u_1=0;

for(i=0;i<100;++i)
{
    y=(float)getw(DapBinIn)/100.0;
    u=(float)getw(DapBinIn)/100.0;

    y_2=y_1;
    y_1=y;
    u_1=u;
}

while(1)
{
    ++k;

    // STEP i

    y=(float)getw(DapBinIn)/100.0;
    u=(float)getw(DapBinIn)/100.0;

    putpixel(k/5,y*y_factor+240,RED);

```

```
x[1]=-y_1;x[2]=-y_2;x[3]=u_1;
```

```
// STEP ii
```

```
E=y-mux13_31(x,t);
```

```
// STEP iii
```

```
XxXT(x,R1); //x(t+1)*xT(t+1)
mux3x3(R1,P,R2); /* numerator */
```

```
mux13_33(x,P,R3);
denominator=1.0+mux13_31(R3,x); //denominator.
```

```
for(i=1;i<4;++i) for(j=1;j<4;++j) R2[i][j]/=denominator; //Divide
```

```
for(i=1;i<4;++i) for(j=1;j<4;++j) //Subtract from I
{
  if(i!=j) R2[i][j]*=-1;
  else R2[i][j]=1-R2[i][j];
}
```

```
mux3x3(P,R2,R1); // mux by P,result is in R1
```

```
equal3x3(R1,P);
```

```
// STEP iv
```

```
mux33_31(P,x,R3);
R3[1]*=E;R3[2]*=E;R3[3]*=E;
t[1]=t[1]+R3[1];
t[2]=t[2]+R3[2];
t[3]=t[3]+R3[3];
```

```
//STEP v
```

```
y_2=y_1;
y_1=y;
u_1=u;
```

```
if(k>3100 || kbhit()) {sound(500);delay(100);nosound();break;}
```

```
putpixel(k/5,t[1]*y_factor+240,RED);
putpixel(k/5,t[2]*y_factor+240,YELLOW);
putpixel(k/5,-t[3]*y_factor+240,LIGHTCYAN);
```

```
}
```



```

t1=-2*exp(-zeta*Wn*Ts)*cos(Ts*Wn*pow((1-zeta*zeta),0.5));
t2=exp(-2*zeta*Wn*Ts);
g0=(t1+(1-t[1]))/t[3];
g1=(t2+(t[1]-t[2]))/t[3];
g2=t[2]/t[3];

```

```

DapStop();
closegraph();

```

```

clrscr();
printf("\ny(t):%f y(t-1):%f y(t-2):%f\nHesaplanan deşerler: a1:%f a2:%f
b0:%f\n",y,y_1,y_2,t[1],t[2],t[3]);

```

```

DapRun2();

```

```

loop;;

```

```

scale=0.55;

```

```

KP=-(int)((g1+2*g2)*1000/scale);
KI=-(int)((g0+g1+g2)*10000/scale);
KD=-(int)(g2*10/scale);

```

```

printf("\nKp:%d Ki:%d Kd:%d",-KP,-KI,-KD);

```

```

fprintf(DapTextOut,"LET SETP=0\n");
fprintf(DapTextOut,"LET P=%d\n",KP);
fprintf(DapTextOut,"LET I=%d\n",KI);
fprintf(DapTextOut,"LET D=%d\n",KD);

```

```

fprintf(DapTextOut, "START IA,PA\n");

```

```

while(!kbhit());

```

```

ch=getch();

```

```

if(ch==' ')
{
fprintf(DapTextOut,"LET SETP=-10000\n");
sleep(2);
goto loop;
}
fclose(fp);
DapStop();

```

```

}

```

; APPENDIX 3
;RECURSIVE ESTIMATION PROCEDURE FOR DAP800

RESET

VARIABLES TEST_PER=500,SETP=5000
PIPES P0,P1

ODEFINE OA 1
SET OPIPE0 A0
TIME 1000
END

IDEFINE IA 2
SET IP0 S0
SET IP1 S1
TIME 500
END

PDEFINE PA
SQUAREWAVE(10000,TEST_PER,OPIPE0)
MERGE(IP0,IP1,\$BINOUT)
END

START IA,OA,PA

; APPENDIX 4
; PID CONTROL PROCEDURE FOR DAP800

RESET

OPTIONS LLATENCY=ON
VARIABLES SETP,P,I,D
PIPES P0,P1

IDEFINE IA 1
SET IP0 S0
TIME 1000
END

PDEFINE PA
PID(IP0,SETP,P,2000,I,15000,D,50,P1)
DACOUT(P1,0)
END

TAJEM KURUMU
DOKUMANTASYON MERKEZİ