IMPLEMENTATION AND EVALUATION OF AN INTELLIGENT TUNER FOR AN ILL-DEFINED SERVO-MOTOR SYSTEM

By

Shimshon Barlev

B.Sc., BEN-GURION UNIVERSITY, Israel.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 1992

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Mechanical Engineering

The University of British Columbia
Vancouver, Canada

Date May 4, 1992

DE-6 (2/88)

ABSTRACT

This research deals with automated, knowledge-based tuning of servo motors. Conventional adaptive techniques can perform unsatisfactorily when the controlled system is complex and incompeletly known. Furthermore, they cannot directly capture and utilize the knowledge of experienced human operators, in tuning a servo system. The tuning technique developed and implemented in this work can overcome these shortcomings.

To integrate the controller of a high speed servo-motor with the tuning knowledge of experienced system operators, a hierarchical control structure is developed in this research. Specifically, the programmable hard controller of a servo-motor is tuned automatically in the lowest level. In the highest level, tuning knowledge expressed as a set of linguistic rules is generated and mathematically formulated using fuzzy set theory and fuzzy logic. This leads to the development of an off-line decision table in which tuning actions are matched with the servo-motor performance. A computer implementation of a servo expert is used in the intermediate level to update the controller parameters so that the actual response would meet a set of predefined performance specifications expressed in terms of the performance of a reference model.

Learning and self-organization, as well as automated specification updating, if necessary, are used to improve the performance accuracy and system robustness.

The intelligent tuner is implemented on a commercially available servo-motor system, and experiments are carried out to demonstrate its performance when implemented on the physical system. Furthermore, simulation results are used to evaluate the performance of the intelligent tuner when implemented on an ill-defined process.

Table of Contents

ABSTRACT ii

List of Figures ix

List of Tables xii

Nomenclature xiii

ACKNOWLEDGEMENTS xvii

1 INTRODUCTION 1

1.1 Overview of the Thesis 1

1.2 Background 2

1.2.1 Servomechanism and Servo-Motor 2

1.2.2 Adaptive Control 3

1.2.3 Expert Systems 8

1.2.4 Expert Control 9

1.2.5 Generality, Uncertainty, Vagueness, Ambiguity and Imprecision of Knowledge 11

1.2.6 Fuzzy Logic 12

1.2.7 Fuzzy Control 12

1.2.8 Fuzzy Tuning 13

1.2.9 Learning Systems 13

1.3 Objective of the Research 14

1.4 Motivation 14
1.5 Relevance of this Research 15
1.6 Literature Review 16
1.6.1 Adaptive Control 16
1.6.2 Expert Systems and Expert Control 17
1.6.3 Fuzzy Sets and Fuzzy Logic 18
1.6.4 Control Application of Fuzzy Logic 18

2 FUZZY LOGIC **20**

2.1 Introduction 20
2.2 Fuzzy Sets 20
2.3 Membership Functions 21
2.4 Logical Operations 23
2.5 Fuzzy Relations 24
2.6 Composition and Inference 30

3 SYSTEM DEVELOPMETNT **35**

3.1 Introduction 35
3.2 General Hierarchical Structure 38
3.3 Servo-Motor Level 41
3.3.1 Physical Servo System 41
3.3.2 Simulated Servo-Motor 45
3.4 Servo Expert Level 49
3.4.1 Performance Specification 51
3.4.2 Response Preprocessor 55
3.4.3 Performance Evaluation 57
3.4.4 Performance Index Classification 59

3.4.5 Tuning of the Controller Attributes 61
3.4.6 Mapping the Attributes of the Controller to its Parameters 63
3.5 Fuzzy Tuner Level 67
3.5.1 Fuzzy Ruleset 70
3.5.2 Membership Functions for Performance and Tuning Variables . . 73
3.5.3 Fuzzy Relation between Condition and Action Variables 75
3.5.4 Decision Table 78

4 EXPERIMENTAL STRATEGY AND PROCEDURE 81
4.1 Introduction 81
4.2 Experimental Strategy 81
4.3 Performance Requirements 82
4.3.1 Model Parameters 82
4.3.2 Acceptable tolerance 83
4.3.3 Test Signal 83
4.4 Base-Line Servo-Motor System 84
4.5 Generation and Evaluation of Tuning Rules 86
4.6 Experimental Procedure 95
4.6.1 Tuning of the Simulated Servo-Motor System 95
4.6.2 Tuning of a Commercially Available Servo-Motor System 96

5 SIMULATION AND EXPERIMENTAL RESULTS 97
5.1 Simulation Experiments 98
5.1.1 Slow Servo-Motor System 98
5.1.2 Oscillatory System 100
5.1.3 Increased Motor Inertia 102
5.1.4 External Torque without Integral Compensation 104

5.1.5 External Torque with Integral Compensation 106
5.1.6 External Spring 108
5.1.7 Voltage Source Amplifier Model 111
5.2 Experiments using a Commercial Servo-Motor 115
5.2.1 Slow Physical Servo-Motor System 116
5.2.2 Physical Oscillatory Servo-Motor 118
5.2.3 Motor with Increased Inertia 120

6 RULE GENERATING BY SELF LEARNING 122

6.1 Introduction 122
6.2 Self Generated Rulset Development 123
6.2.1 Approach 123
6.2.2 Controller Presetting 124
6.2.3 System Perturbation 124
6.2.4 Response Sensitivity Index 126
6.2.5 Response Sensitivity Classification 127
6.2.6 Rule Pattern 128
6.3 Experimental Procedure and Results 129
6.3.1 Procedure 129
6.3.2 Results 129
6.4 Summary 134

7 CONCLUSIONS and RECOMMENDATIONS 137

7.1 Introduction 137
7.2 Summary of Accomplishments 137
7.2.1 The Development of the Experimental System 137
7.2.2 Rule Base of Tuning 138

7.2.3 Tuning Simulated System 139
7.2.4 Implementation on Physical System 139
7.2.5 Self Learning 139
7.3 Main Contributions of the Research 139
7.4 Advanatges and Limitations 140
7.5 Future Developments 141

Bibliography **142**

Appendices **151**

A The Physical Servo-Motor **151**
A.1 General Specification of the Servo-Motor System 151
A.2 Motor 152
A.2.1 Specifications of the 50/1000 DC Motor 152
A.2.2 Mechanical Drawing of the 50/1000 DC Motor 153
A.3 Specification of the DMC400 Controller 154
A.4 Specification of the Interconnection Board 155
A.5 Interface and Communication Software 156
A.5.1 Inerface to C 156

B Programs Listing **170**
B.1 Main Program coded in ACSL 170
B.2 Servo-Expert sub-programs listing 175
B.2.1 Sub-Program MODEL 175
B.2.2 Sub-Program PREPROCESS 176
B.2.3 Sub-Program EVALUATION 178
B.2.4 Sub-Program CLASSIFICATION 181

B.2.5 Sub-Program TUNING 184
B.2.6 Sub-Program DESIGN 188
B.2.7 Print Out 189
B.3 Fuzzy Tuner Program listing 192
B.3.1 Main Program DECISION 192
B.3.2 Sub-Program DATA 194
B.3.3 Sub-Program CNDMF 197
B.3.4 Sub-Program ACTMF 199
B.3.5 Sub-Program RULESET 201
B.3.6 Sub-Program RELATION 209
B.3.7 Sub-Program DECISION 212
B.3.8 Sub-Program DEFUZZY 214
B.4 Subprogram LEARN 215

List of Figures

1.1 Block Diagram of Gain Scheduling 5

1.2 Block Diagram of SOAS 6

1.3 Block Diagram of MRAS 6

1.4 Block Diagram of Self-Tuning Regulator 7

1.5 Block Diagram of a Knowledge-Based Expert Control System 11

2.1 A Fuzzy Set 22

2.2 Relations in Two Dimensional Space (plane) 25

2.3 Cartesian Product of $A_1 \times A_2$ (relation) 27

2.4 A Mapping from a Product Space to a Line 29

2.5 The Cylindrical Extension 32

3.1 The Experimental System 37

3.2 Hierarchical Structure of the System 40

3.3 The Hardware of the Physical Servo-Motor System 42

3.4 Block Diagram of the Simulated Servo-Motor 45

3.5 Block Diagram of the Servo Expert Level 50

3.6 Mapping from Performance to performance Index 60

3.7 Controller Attributes Definition 64

3.8 Fuzzy Tuner Block Diagram 69

3.9 Membership Functions for the Performance Variables 74

3.10 Membership Functions for the Tuning Variables 75

4.1 Time Domain Response of the Base-Line Servo-Motor System 85
4.2 Effect of Phase Lead Angle at Cross-over Gain Frequency 87
4.3 Effect of Cross-over Gain Frequency 87
4.4 Effect of Cross-over Gain 88
4.5 Effect of Cross-over Gain when External Load is Applied 88
4.6 Effect of Low-frequency at Cross-over Gain when External Load is Applied 89

5.1 Tuning of initially Slow Servo-Motor: Time Response 98
5.2 Tuning of an Initially Slow Servo-Motor: Controller Attributes 99
5.3 Tuning of an Initially Oscillatory Servo-Motor: Time Response 100
5.4 Tuning of an Initially Oscillatory Servo-Motor: Controller Attributes . . 101
5.5 Tuning a Motor with Increased Inertia: Time Response 102
5.6 Tuning a Motor with Increased Inertia: Controller Attributes 103
5.7 Tuning for External Torque without Integral Compensation: Time Response104
5.8 Tuning for External Torque without Integral Compensation: Controller Attributes 105
5.9 Tuning for External Torque, using Integral Compensation: Time Response 106
5.10 Tuning for External Torque, using Integral Comensation: Controller Attributes 107
5.11 Motor Block Diagram for External Spring 108
5.12 Tuning in the presence of an External Spring: Simulated System Response 109
5.13 Tuning in the Presence of an External Spring: Controller Attributes . . . 110
5.14 Tuning of a Slow Servo-Motor with Voltage Source Amplifier: Time Response111
5.8 Tuning of a Slow Servo-Motor with Voltage Source Amplifier: Controller Attributes 112

5.9 Tuning for Oscillaory Servo-Motor with Voltage Source Amplifier – Time Response 113
5.9 Tuning of an Oscillatory Servo-Motor with Voltage Source Amplifier: Controller Attributes 114
5.10 Tuning of a Slow Servo-Motor: Time Response of the Physical System . 116
5.10 Tuning of a Slow Servo-Motor: Controller Attributes of the Physical System 117
5.11 Tuning of an Oscillatory Servo-Motor: Response of the Physical System . 118
5.11 Tuning of an Oscillatory Servo-Motor: Controller Attributes of the Physical System 119
5.12 Tuning for Increased Motor Inertia: Physical System Response 120
5.12 Tuning for Increased Motor Inertia: Physical System Controller Attributes 121

6.1 The System in Learning Mode 125
6.2 The Response of Preset and Perturbed Systems 131
6.3 Tuning of a Slow Servo-Motor using a Self-Learning Ruleset 135
6.4 Tuning of an Oscillatory Servo using a Self-Learning Ruleset 136

List of Tables

3.1 Development of a Fuzzy Relation Table (for $RISTM \longrightarrow PHCOF$) . . . 77

3.2 Fuzzy Composite Relation Table for RISTM and PHCOV 78

3.3 Decision Table for the Servo-Motor 80

4.1 Condensed Form of the Ruleset 94

6.1 Self Generated Ruleset 133

Nomenclature

Most of the symbols used in the text are defined in the sections in which they occur.

Math italic type style is used for the servo-motor parameters and variables.

In the servo-expert level calligraphic letters are used for the performance parameters and Greek letters for the tuning actions.

Fuzzy performance parameters and tuning actions as well as performance and tuning quantities in the fuzzy tuner level are denoted by capital text letters and are arranged in fuzzy arrays indicated by the tilde math accent~.

T	Sampling time interval of controller$[s]$
ΔT_s	Communication time interval of servo-motor tuner $[s]$
t	time $[s]$
Po	motor position $[rad]$
K	torque constant $[N \cdot m/amp]$
V_i	supply voltage to the stator in voltage source amplifier $[volt]$
J	moment of inertia of the rotor $[kg \cdot m^2]$
L	inductance of the field winding $[mH]$
R	resistance of the field winding $[ohm]$
B	motor mechanical damping $[kg/s]$
T_l	external load $[N \cdot m]$
s	Laplace variable
I_i	supply current to the stator in current source amplifier $[amp]$
A	effective gain of the current source amplifier $[amp/volt]$

V_{max}	maximum voltage input to the amplifier $[volt]$
I_{max}	maximum current output of the amplifier $[amp]$
Y	measured position of the motor (encoder output) $[counts]$
N	number of pulses per revolution of encoder
G_e	transfer function of the encoder
G_c	transfer function of the controller
G_{cl}	transfer function of the lead-lag branch of controller
G_{ci}	transfer function of the integrator branch of controller
G_p	transfer function of the system, excluding the controller
C	controller output $[volt]$
Y_r	required motor position $[counts]$
K_l	lead-lag compensation net gain
K_i	integrator compensation net gain
Zr	frequency of the lead-lag zero $[rad/s]$
Pl	frequency of the lead-lag pole $[rad/s]$
z	Z'-transform
Gm	transfer function of the reference model
Ym	position response of the reference model $[counts]$
ω_n	undamped natural frequency of the reference model $[rad/s]$
ζ	damping ratio of reference model
θ	deterministic error (offset) of reference model
ϵ	nondeterministic error of reference model
γ	amplitude of response peaks
τ	timing of response peaks $[s]$
$\mathcal{P}$	number of response peaks

$\mathcal{RT}$	rise time of response $[s]$
$\overline{\mathcal{DF}}$	average damped natural frequency of response $[rad/s]$
$\overline{\mathcal{DR}}$	average damping ratio of response
$\mathcal{OS}$	overshoot of response
$\mathcal{OF}$	offset of response
$\mathcal{PR}$	accuracy of response
$\mathcal{ERR}$	normalized error vector between the performance parameters of servo-motor a
$\mathcal{TH}$	vector of performance classification thresholds
$\mathcal{K}$	vector of performance index
ω_{bw}	frequency bandwidth of servo-motor $[rad/s]$
ω_{cog}	cross-over gain frequency of servo-motor $[rad/s]$
ϕ_{cof}	phase lead angle of controller at ω_{cog} $[rad]$
ψ_{cof}	gain of controller at ω_{cog}
ωl_{cog}	low frequency at ψ_{cof} $[rad/s]$
$\odot_{(l)}$	vector containing the controller attributes
P	universe of fuzzy performance variables
I	cardinality of fuzzy performance space
i	index of fuzzy performance variables
$\widetilde{PV}_{(i)}$	fuzzy subspace in the universe of performance variables
j	index of fuzzy performance quantity
$\widetilde{PQ}_{(j)}$	element in $\mathrm{P}_{(i)}$
k	index of membership function of performance index
RISTM	fuzzy form of $\mathcal{RT}$
DMPFR	fuzzy form of $\mathcal{DF}$
DMPRT	fuzzy form of $\mathcal{DR}$

OVSHT	fuzzy form of $\mathcal{OS}$
OFFST	fuzzy form of $\mathcal{OF}$
ACCUR	fuzzy form of $\mathcal{AC}$
T	universe of fuzzy tuning actions
L	cardinality of fuzzy tuning action space
l	index of fuzzy action variables
$\widetilde{TV}_{(l)}$	fuzzy subset in the universe of tuning variables
m	index of fuzzy action quantity
$\widetilde{TQ}_{(m)}$	element in $\mathrm{T}_{(l)}$
n	index of membership function of tuning action
PHCOV	fuzzy form of ϕ_{cof}
FRCOG	fuzzy form of ω_{cog}
GNCOF	fuzzy form of ψ_{cof}
LFCOG	fuzzy form of ωl_{cog}
$\widetilde{RU}$	array of fuzzy rules
$\widetilde{PM}$	array of performance membership functions
$\widetilde{TM}$	array of tuning membership functions
$\widetilde{RL}$	array of fuzzy relations
$\widetilde{DT}$	decision table array

ACKNOWLEDGEMENTS

I would like to thank Dr. C.W. de Silva for his constant supervision and guidance throughout this research. Special thanks also to all my fellow students and friends from the of Industrial Automation Laboratory (IAL) for generous help and support, which made the whole experience more enjoyable.

I would like to thank RAFAEL for sponsoring my graduate studies and the Industrial Automation Chair of the Natural Sciences and Engineering Research Council (NSERC), held by Dr. C.W. de Silva for partial funding of this research.

Last and not least, I would like to thank my family for their constant support and encouragement.

Chapter 1

INTRODUCTION

1.1 Overview of the Thesis

This thesis presents a study of the analysis and practical implementation of an intellegent tuner for a servo-motor. A method to automate the tuning actions taken by an experienced process operator in a complicated (complex) system that is found to be beyond the capability of conventional adaptive techniques, is considered. While a number of studies have considered the use of fuzzy logic in control applications, the majority were employed in low level direct control rather than high level controller tuning. Also, fuzzy tuning has been implemented in simulated systems only.

An experimental system has been developed to explore, evaluate, implement and illustrate the tuner performance.

It has three hierarchal levels:

- Simulated servo-motor system consisting of a programmable controller, D.C motor and encoder has been developed in the lowest level of the hierarchical structure.
- In an intermediate level, called the servo expert level, an intelligent evaluator evaluates performance index using the responses of the actual and reference models. A design mechanism updates the controller parameters using tuning actions which were provided from a decision table, using a preformance index as a contex.
- In the highest level of the hierarchy, knowledge of the effect of each tuning action on the performance parameters is generated and mathematically formulated using

fuzzy sets theory and fuzzy logic operations to calculate a decision-table. Here the performance indices are matched with the tuning actions. Learning and self-organization is also used in this level to modify the decision table automatically.

The tuner has been implemented on a commercially available servo-motor and its performance has been examined. Further experiments have been conducted by gradually spoiling the simulated servo-motor far below the simple system model.

The thesis is divided into seven chapters. Chapter 1 presents the basic concepts, background, objectives, motivation, relevance of the work and literature review. Chapter 2 gives a brief review of fuzzy sets and fuzzy logic theory as this is necessary to understand the mathematical representation of human knowledge. Chapter 3 describes the system structure and development. Description of the experimental strategy and procedure is in Chapter 4. Experiments on simulated and physical systems are described in Chapter 5. Chapter 6 lays the foundation for future work by introducing a learning and self-organizing tuner system. The work is concluded in chapter 7.

1.2 Background

In this section the key terms that are used throughout this work are briefly explained. In particular servomechanism and servo-motor, various adaptive control methods, expert systems and expert control, knowledge base, fuzzy knowledge, fuzzy logic, fuzzy control and fuzzy tuning are explained.

1.2.1 Servomechanism and Servo-Motor

Servomechanisms are widely used to drive processes in industrial, transportation, aeronautical and space systems as well as domestic applications. The term "servo-mechanism"

has been used in the literature with a variety of meanings. We shall call a device a servomechanism if it satisfies the following description:

The device controls some physical quantity by comparing its actual response with its desired values and uses the difference (or error) to drive the actual response into correspondence with the desired value.

Electrical (stepper, DC, AC or synchronous) motors, pneumatic or hydraulic actuators [24]might be used to drive the process. Digital or analog controllers are used to control it. In both cases the parameters of the controller are chosen such as to meet some predefined performance specifications. The actuator of a digital D.C. servomotor is an electrical direct current motor and it has a digital controller. In this work the term "servo-motor" stands for DC brushless motor having a digital servo controller.

1.2.2 Adaptive Control

In everyday language, to "adapt" means to change a behaviour to conform to new circumstances. Intuitively, an adaptive controller is a controller that can modify its behaviour in response to changes in the dynamics of the process and the disturbances.

Adaptive control is a special type of nonlinear feedback control in which the states of the process can be separated into two categories, which change at different rates. The slowly changing states are viewed as parameters. This introduces the idea of two time scales: a fast time scale for the ordinary feedback and a slower one for updating the controller parameters.

A Brief History

In early 1950's there was extensive research on adaptive control, in connection with the design of autopilots for high performance aircraft, which operate over a wide range of speed and altitude. It was found that a constant parameter controller would work well

in one operating condition but a change in operating conditions led to difficulties, and a more sophisticated controller was therefore needed [4].

Theories for state space modeling, stability analysis and stochastic control, and techniques of system identification, parameter estimation and dynamic programming were developed in the 1960's and this also contributed to the understanding of adaptive processes. Many applications were reported in the 1970's when different estimation schemes were combined with various design methods.

In the early 1980's rapid and revolutionary progress in microprocessors has made it possible to implement adaptive methods simply and cheaply. Several commercial adaptive controllers based on different ideas are appearing on the market, and the industrial use of adaptive control is growing slowly but surely.

Adaptive Schemes

There are many schemes of adaptive control. Most of them are based on either prior knowledge of a system model or identification/estimation of a model of the process to be controlled. These methods can be classified as direct and indirect methods [4]:

- Direct methods:

  In direct methods the adjustment rules tell directly how the controller parameters should be updated.

- Indirect methods:

  An indirect scheme is obtained if a process model is identified, its parameters are estimated and the controller parameters are updated through the solution of a design problem.

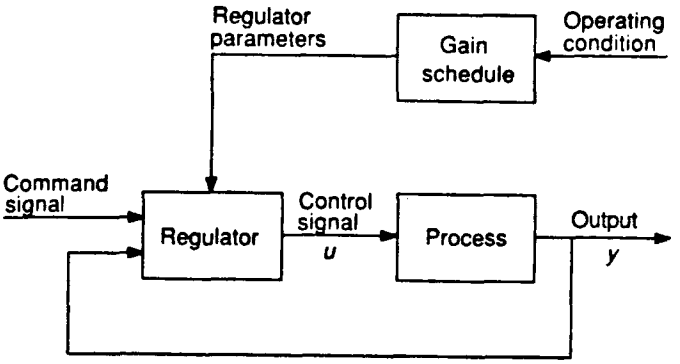


Figure 1.1: Block Diagram of Gain Scheduling

Gain Scheduling

In gain scheduling the parameters of a closed loop feedback controller are adjusted in a feed-forward manner, in an open loop with known operating conditions. A block diagram of system with gain scheduling is presented in Figure 1.1 [4]:

Self-Oscillating Adaptive System

A self-oscillating Adaptive system(SOAS) has high loop gain and its bandwidth is automatically adjusted to be as high as possible by introducing a relay in the feedback loop. This results in a robust system over a wide bandwidth. A limit cycle oscillation is presented and the system is always excited. A block diagram of a SOAS is shown in Figure 1.2[4]:

Model Reference Adaptive Systems

Model Reference Adaptive Systems (MRAS) consist of two loops: an inner loop, which is the ordinary feedback loop and an outer loop to adjust the controller parameters in such a way that the error between the output of the process and a reference model becomes small.A block diagram of MRAS is shown in Figure 1.3[4]

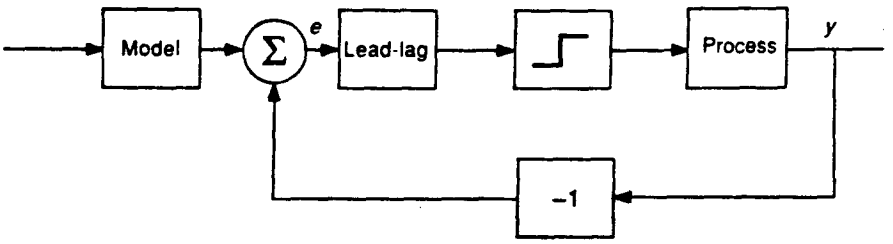


Figure 1.2: Block Diagram of SOAS

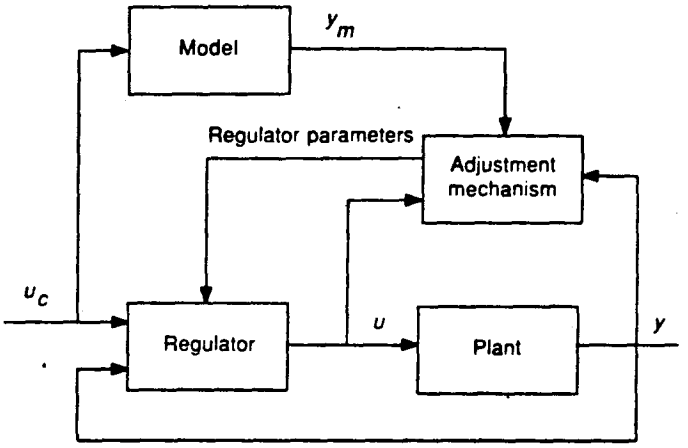


Figure 1.3: Block Diagram of MRAS

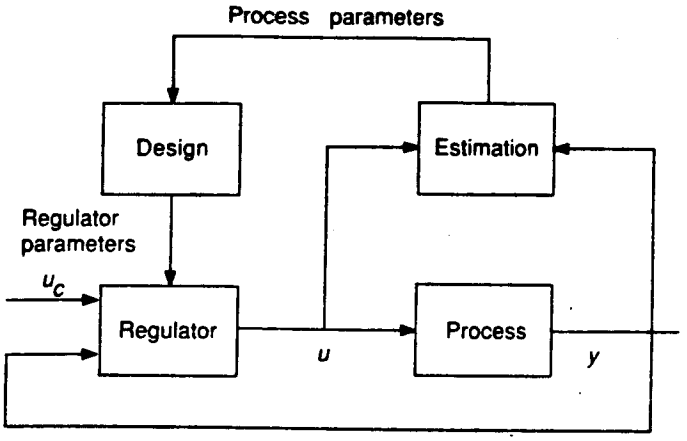


Figure 1.4: Block Diagram of Self-Tuning Regulator

Self-Tuning Regulator

The Self-Tuning Regulators (STR) are a family of methods based on the idea of separating the on-line estimation of unknown parameters from the design of the controller. These methods differ either by the design or estimation algorithm.

The following belong to this family :

- Linear Quadratic Self-Tuning Regulator (LQS).

  This make use of an optimal feedback law that minimizes a quadratic function (positive semidefine) error between the actual and model responses and also the control effort.

- Adaptive Predictive Control (APC).

  This algorithm is based on an assumed model of the process and on assumed scenario for the future control signals. This gives a sequence of control signals. Only the first one is applied to the process, and a new sequence of control signals is calculated when a new measurement is obtained.

A block diagram of a typical STR is shown in Figure 1.4[4]

Auto Tuning

Auto tuning is based on an experimental phase in which test signals are injected into the system. The controller parameters can be determined from the experiments using standard rules for tuning if these rules exist (for example Ziegler-Nichols rules for PID controllers). The main advantages of an auto tuner are that it requires little prior information on the process, is very robust and can generate good parameters for a simple control law.[79][18][54]

1.2.3 Expert Systems

An expert system attempts to model the knowledge and procedures used by a human expert in solving problems within a well-defined domain. An expert system consists of a knowledge base, inference engine and one or more data bases which may be linked to a user interface. [38]

Knowledge Base

The knowledge base consists of data and rules. The data are *facts* and *goals*. Data is introduced into the database by the user or via the real time knowledge acquisition system. A rule base contains production rules of the type:

" If premise *then* conclusion *do* action".

The premise represents fact or condition from database. The conclusion can result in a new fact being added to the database or modification of an existing facts. The action activates control, estimation or tuning algorithms. The rules are introduced by the knowledge engineer via the knowledge acquisition system, which assists in writing and testing rules. In control applications the rules represent knowledge about the control and estimation problem that is built into the system.

Inference Engine

The inference engine processes the rules to arrive at conclusions or satisfy goals. It scans the rules according to a strategy which is decided from the context (current database of facts and goals) and decides which production rules are to be selected next. This can be done according to different strategies:

In *forward chaining* it is attempted to find all conclusions from a given set of premises. In *backward chaining* the rules are traced backward from a given goal to see if it can be supported by the current premise.

User Interface

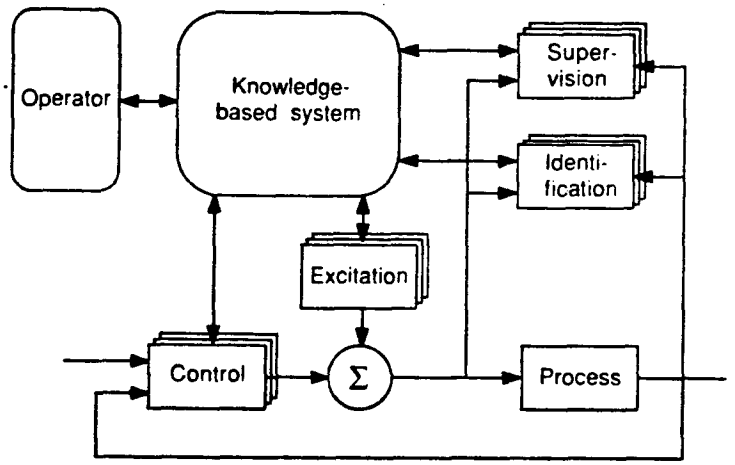
The user interface can be divided into two parts. The first part is the development support that the system gives, such as rules editor and rules browser for development of the system knowledge base. The other part is the run time user interface. This contains the explanation facilities that make it possible to question how a certain fact was concluded, why a certain estimation is executed, etc. The user interface can also contain facilities to deal with a natural language.

1.2.4 Expert Control

The idea of expert control is to have a collection of algorithms for control, supervision and adaptation that are orchestrated by an expert system. A block diagram of such system is shown in Figure 1.5[4] . A comparison with Figure 1.4 on page 7 shows that the system is a natural extension of a self-tuning controller. Instead of having one control algorithm and one estimation algorithm, the system has several algorithms. It also has algorithms for excitation and for diagnosis, as well as tables for storing data.

Apart from this it also has an expert system which decides when a particular algorithm

should be used. The expert system contains knowledge about particular algorithms and conditions in which they can be used.



1.2.5 Generality, Uncertainty, Vagueness, Ambiguity and Imprecision of Knowledge

Generality, Uncertainty, Vagueness, Ambiguity and Imprecision are terms which are often used in the contents of fuzziness but strictly speaking these terms have meanings which differ from the formal definition of fuzziness [23][25]. Generality can be associated with the use of a single symbol to represent more than one element. Ambiguity is attributed deals with the presence of more than one interpretation for a particular situation or quantity. Uncertainty is associated with probability. Precision is defined in terms of tolerance.

In developing facts and rules for expert systems, it becomes clear that data and rules obtained from experienced specialists are somewhat uncertain. They may describe some rules in **linguistic** terms like *"maybe","sometimes"*, or *"often"*. Some method is needed to handle these types of possibilistic statements. Furthermore, expert systems, like human experts, may need to draw inference based on unavailable, unknown or uncertain data. These data are expressed in linguistic form like "tall", "fast", "beautiful", etc.

1.2.6 Fuzzy Logic

Fuzzy logic deals with fuzzy sets. A fuzzy set **has no sharp boundaries** and the **possibility** of an element to belong to the set is given by a membership function. Fuzzy logic uses logical operations like AND, OR, NOT which operate on the membership functions of fuzzy sets. Fuzzy sets weres found to be useful in uncertain data representation and fuzzy logic in uncertain rule representation.[77][78][29][52]

The relevant concepts of fuzzy logic, particulary those useful in fuzzy control are outlined in Chapter 2.

1.2.7 Fuzzy Control

In fuzzy control, linguistic descriptions of human expertise in controlling a process are represented as a fuzzy rules or relations, and this knowledge base is used, in conjunction with some knowledge of the state of the process (say, measurement response of variables), by an inference mechanism to determine control actions at a sufficiently fast rate.[12][19][70][46][53] [41][3]

A formal procedure in fuzzy control can be summarised in the following steps [23]:

I. Knowledge Base Development

1. Develop a set of linguistic control rules (protocols).
2. Develop a set of membership functions for process output variables and control input variables.
3. Using fuzzy implication on each rule in 1 and using 2, obtain the multi-dimensional array of membership values for that rule.

4. Combine the array using fuzzy operations to obtain the overall fuzzy rule base.

II. Real Time Control Action

1. Fuzzify the measured process variables, as fuzzy singletons.

2. Match the fuzzy measurements obtained in 1 with the membership array of the fuzzy rule base (obtained in the previous Step 4), using the appropriate fuzzy logic operations.

3. Defuzzify the control inference obtained in Step 2.

There are several variations. For example, a much faster approach would be to develop a crisp decision table by combining the four steps of fuzzy algorithm development and the first two steps of control, and using this table in a table look-up mode to determine a crisp control action during operation.

1.2.8 Fuzzy Tuning

To combine the advantages of low level control for nonlinear, high-order coupled dynamic with the knowledge of an experienced process operator, a high level fuzzy tuner is used to **tune** the parameters of a low level controller . This structure results in a conventional, high bandwidth inner control loop and a slow outer closed loop soft tuning algorithm.

1.2.9 Learning Systems

In their most ambitious form, learning systems attempt to describe or mimic human learning abilty. This goal is still far away. The learning systems that have actually been

implemented are simple systems that have strong relations to adaptive control. Such systems are neural nets, connectionist models, parallel distributed processing models, etc.[4]

1.3 Objective of the Research

The objective of this research is to use fuzzy logic to develop implement and evaluate a knowledge-based auto-tuning mechanism to automate the tuning actions taken by experienced operators in complex and partially known servo-motor systems.

1.4 Motivation

What is the motivation for trying "nonconventional" tuning approaches? Why use fuzzy logic? Why tune controller parameters rather than generate the control actions directly?

- **Motivation for Trying another Tuning Approach**

  To design a controller in a "conventional" adaptive technique we have to have either a prior known model or estimate/identify a suitable process model and parameters (see Figure 1.4 on page 7). Furthermore if the structure of the process is unknown it should be identified as well. Identification and estimation algorithms become unfeasible when applied to very complicated, manually tuned, real systems. In practical applications the knowledge of an experienced operator is concerning the effects of each tuning action on the performance rather than the knowledge of the model of the process. In this work the approach followed is to take advantage of this knowledge to automate these tuning actions directly without using estimation or identification algorithms.

- **Motivation for using Fuzzy Logic**

An existing knowledge of an expert process operator is expressed in terms of linguistic rules of the form "*if* the performance is **such and such** *then* the tuning action is **such and such**", where the condition and actions are inaccurate, vague or fuzzy quantities that can be formulated mathematically using fuzzy sets and processed using fuzzy logic.

- **Motivation for Tuning rather than Direct Control**

Difficulties in the implementation of direct fuzzy control on a low level of a high-bandwidth system motivate tuning rather then direct control. One drawback of direct fuzzy control is that human observation, interpretation, decision making and action are not fast enough for real time, closed-loop direct control in high bandwidth systems. Another drawback is that using fuzzy inference in closed control loop introduces errors directly into the control signals. Another argument against the conventional, low-level implementation of fuzzy control is that in high speed processes, human experience is gained not through manual, on line generation of control signal in response to process output, but typically through performing parameter adjustments and tuning (manual adaptive control) operations. [23]

1.5 Relevance of this Research

- **Conserving and Spreading Knowledge of Expert Process Operator**
  Since human experts are hard to come by, it is not economical to dedicate a human expert for every same process. Automating the tuning actions of an expert operator provides a means for the conservation and distribution of knowledge.

- **Evaluation and Comparison with other Technique**

  This work provides means for performance evaluation and comparison with other adaptive techniques.

- **Implementation on Real Systems**

  Implementation of the tuner on a commercially available servo-motor demonstrates the feasibility of the proposed tuner in the presence of physical factors and effects of a real environment like noise, friction, backlash and so on.

1.6 Literature Review

1.6.1 Adaptive Control

Many papers, books, and reports have been written on adaptive control. Some of the earlier developments are summarized in Gregory (1959) [35] and Mishkin and Braun (1961)[58] in flight control applications. Reprints of 44 fundamental papers in adaptive control are found in Gupta (1986)[37].

A good introduction to modern adaptive control is found in Astrom and Wittenmark [4]. Later developments in adaptive control are also treated by the same authors[8] in 1983 and [9] in 1987.

The following are some fundamental references on the different adaptive schemes:

Gain scheduling is described in NASA report(1977)[60], by Stein(1980) [68] and by Whatley and Pott (1984) [73] . Self-oscillating adaptive systems are discussed by Gregory(1959)[35] and Mishkin and Braun (1961)[58]. The model-reference adaptive control based on the MIT rule is given by Osburn, Whitaker and Kezer (1961) [61] Another book dedicated to MRAS is by Landau (1979)[49].

Stability problems in MRAS are treated by Parks(1966)[63]. Narendra and Lin (1980)[59] discussed different generic error model for MRAS.

Further details on MRAS can be found in Kornblugh(1984) [48]

The self-tuning idea and its asymptotic properties were first derived by Clark and Astrom(1973)[7]. Gawthrop(1986) [33] used a unified approach to continuous-time self-tuning regulators.

Linear Quadratic Self Tuning Regulators which combined least square estimation and minimum variance control was presented by Peterka(1970) [64], Wieslander and Wittenmark (1971)[74] and Grimble(1984)[36]. Application to robotic manipulators can be found in de Silva and Van Winssen (1987) [25]

Adaptive predictive control is discussed by:

Clarke and Gawthrop (1975)[15] Clarke, Mohtady and Tuffs (1987)[16] [17], Ydstie (1984)[76], De Keyser, and Van Cauwenberghe (1985) [28]

1.6.2 Expert Systems and Expert Control

A good source for knowledge about expert systems is the Handbook of Artificial Intelligence by Barr and Feigenbaum (1982)[11]. Another source is Hayes, Watermann and Lenat (1983)[38].

Expert systems research is done by many workers including Duda and Shortliffe (1983)[30] and by Rich (1983)[65].

Knowledge Representation in artificial intelligence is found in Brachman and Smith(1980)[13] and Davis and Lenat (1982)[21].

The notation "Expert Control" was introduced by Astrom, Anton and Arzen (1986) [6] and elaborate by dArzen (1987) [2] and Francisand Leitch (1986)[31].

Artificial intelligence in process control can be found in Goff (1985) [34].

1.6.3 Fuzzy Sets and Fuzzy Logic

Fuzzy sets were first discussed in 1965 by Zadeh [77]and in 1973 [78] for the purpose of defining a fuzzy relation between elements and their sets. Further discussion may be found in Dubois and Prade (1980) ,[29], Hirota (1979)[40] and Kaufmann (1975)[42], Braae and Rutherford (1979)[12].

1.6.4 Control Application of Fuzzy Logic

Fuzzy control as a new approach to the analysis and implementation of a control strategy of a human beings has been analysed by many researches. Kloeden (1982)[45], Kiszka Kochanska and Sliwinska (1985)[47] and Tong(1980)[70] are examples. Fuzzy process identification was investigated by Czogala and Pedrycz (1981)[19] and also in 1982[19]

A partial list of researches in the field of fuzzy control is given below:

Assiian and Mamdani(1974)[3], Holmblad and Ostergaard (1981)[41], Kickert and Nauta (1976)[43], Mamdani and Gaines (1981)[52], Mamdani (1974)[56] and (1983)[53], Tong (1976)[69].

Industrial application of fuzzy logic reported by Larsen (1980)[50] and Mamdani, Ostergaard and Lembessis (1984)[55] and King and Mamdani (1975)[44].

Application of fuzzy control in mining can be found in Carter and Rutherford (1976)[14] for a sinter plant.

Kickert and Nauta (1976)[43] report the use of fuzzy controller to a warm water plant and Ostergaard (1976)[62] to a Heat Exchanger Process.

Tong, Beck and Latter 1980[71] in activated sludge wastewater treatment process and Van Amerongen, Van Nauta Lemke and Van der Veen[72] in application on an autopilot for ships.

Many applications on robotic manipulators were reported: de Silva and MacFaralen (1989)[23], Mandic, Scharf and Mamdani (1985)[57] , Scharf and Mandic (1984[67] are examples.

Tuning of a PID controller in servo-motor systems has been done by de Silva (1989)[26] and de Silva and Barlev 1992[27].

Learning and self organizing systems can be found in Hiraim, Asai and Kitajima (1968)[39] , Procyk and Mamdani (1979)[66], Scharf and Mandic (1984)[67], Yamazaki and Mamdani (1982)[75] and many others.

Chapter 2

FUZZY LOGIC

2.1 Introduction

Formalisation of the concept of fuzzy sets and associated fuzzy reasoning (approximate reasoning) is due to Zadeh (1965,1973)[77][78].

In this chapter fuzzy logic is reviewed, as this is necessary to understand the mathematical representation of human knowledge , used in the next chapter. First the concept of fuzzy set and membership function, which represents the uncertainty in the human knowledge, is introduced as well as the fuzzy logic operations which operate on these membership functions. Next the relations of fuzzy sets are discussed and finally the ideas of composition and inference of fuzzy relations are explained.

What is presented here is the interpretation and geometrical illustration of de-Silva and MacFarlane (1989)[23] to the theoretical considerations described by Dubios and Prade (1980)[29].

2.2 Fuzzy Sets

Fuzzy logic deals with fuzzy sets. A fuzzy set does not have a sharp boundary. Consider a universe of discourse X whose elements are denoted by x and a subset A in X. A is a fuzzy subset in X if there is vagueness associated with the membership of x in A.

2.3 Membership Functions

In universe X, the *possibility* of each element x belonging to the fuzzy subset A is represented by a membership function μ_A. This assigns a number $\mu_A(x)$ in the interval [0,1]. If $\mu_A(x) = 0$ then the element x definitely does not belong to the subset A. If $\mu_A(x) = 1$ then the element x definitely belongs to the subset A (or: $x \in A$). A grade of membership greater than 0 and less than 1 corresponds to an element which falls on the fuzzy boundary of the set A. In other words, the membership function maps the elements of the universe X to numerical value [0,1].

$$X \xrightarrow{\mu_A(x)} [0,1]$$

A fuzzy set is demonstrated in Figure 2.1 [23]. A Venn Diagram of the subset A in X is shown in Figure 2.1 (a) and a typical membership function $\mu_A(x)$ is demonstrated in Figure 2.1 (b).

Fuzzy set may be specified using a convenient *form of notation* due to Zadeh, in which each element is paired with its grades of membership in the form of:

$$A = \Sigma_{x_i \forall X} \frac{\mu_A(x_i)}{x_i} \tag{2.1}$$

for discrete universe, or:

$$A = \int_{x \forall X} \frac{\mu_A(x)}{x} \tag{2.2}$$

for continuous universe.

Equations 2.1 and 2.2 are **symbolic shorthand forms of notion** only .

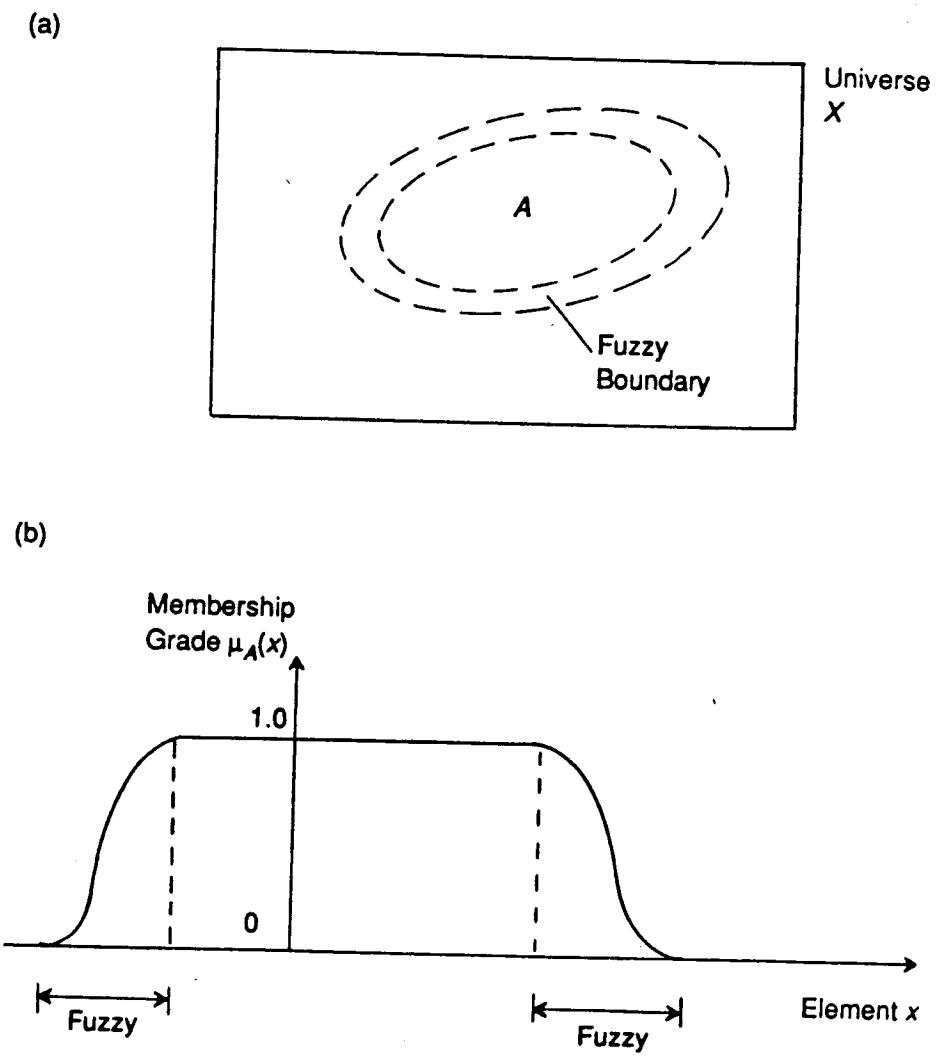


Figure 2.1: A Fuzzy Set
(a) Venn diagram
(b) Typical Membership Function

2.4 Logical Operations

It is well known that "complement","union","intersection" and "implication" of sets correspond to the logical operations NOT, OR, AND and IF - THEN respectively. These logical operations of fuzzy sets are used in fuzzy knowledge representation. In fuzzy logic these connectives have to be expressed in terms of membership functions of the sets which are operated on.

Complement (NOT) A'

Consider a fuzzy set A in a universe X. Its complement A' is a fuzzy set whose membership function is given by:

$$\mu_{A'}(x) = 1 - \mu_A(x) \qquad \forall x \in X \tag{2.3}$$

Union (OR) $A \cup B$

Consider two fuzzy sets A and B in the same universe X. Their union is a fuzzy set $A \cup B$. It membership function is given by:

$$\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)] \qquad \forall x \in X \tag{2.4}$$

The rationale for the use of "max" is that since the element x may be in one set **or** the other, the larger of the two membership function grades should apply.

Intersection (AND) $A \cap B$

Again consider two fuzzy sets A and B in the **same** universe X. Their intersection is a fuzzy set $A \cap B$. Its membership function is given by:

$$\mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)] \qquad \forall x \in X \tag{2.5}$$

The rationale for the use of "min" is that since the element x may belong to both sets **simultaneously**, the smaller of the two membership function grades should apply.

Implication (IF - THEN) $A \longrightarrow B$

Consider a fuzzy set A in a universe X and a second fuzzy set B in **another** universe Y. The fuzzy implication $A \longrightarrow B$ is a fuzzy relation in the cartesian product space $X \times Y$ and the membership function of the fuzzy implication is given by:

$$\mu_{A \rightarrow B}(x, y) = min[\mu_A(x), \mu_B(y)] \qquad \forall x \in X,\ \forall y \in Y \qquad (2.6)$$

2.5 Fuzzy Relations

Consider two universes $X_1 = x_1$ and $X_2 = x_2$. A crisp set R consisting of a subset of ordered pairs (x_1, x_2) is a crisp relation in the cartesian product space $X_1 \times X_2$. Analogously, a fuzzy set R which consists of a subset of ordered pairs (x_1, x_2) is a fuzzy relation in the cartesian product space $X_1 \times X_2$ and the relation R will be represented by the membership function $\mu_R(x_1, x_2)$.

This concept can be extended in a straightforward manner to fuzzy relations in the n-dimensional cartesian space. An example of a fuzzy relation is shown in Figure 2.2[23]

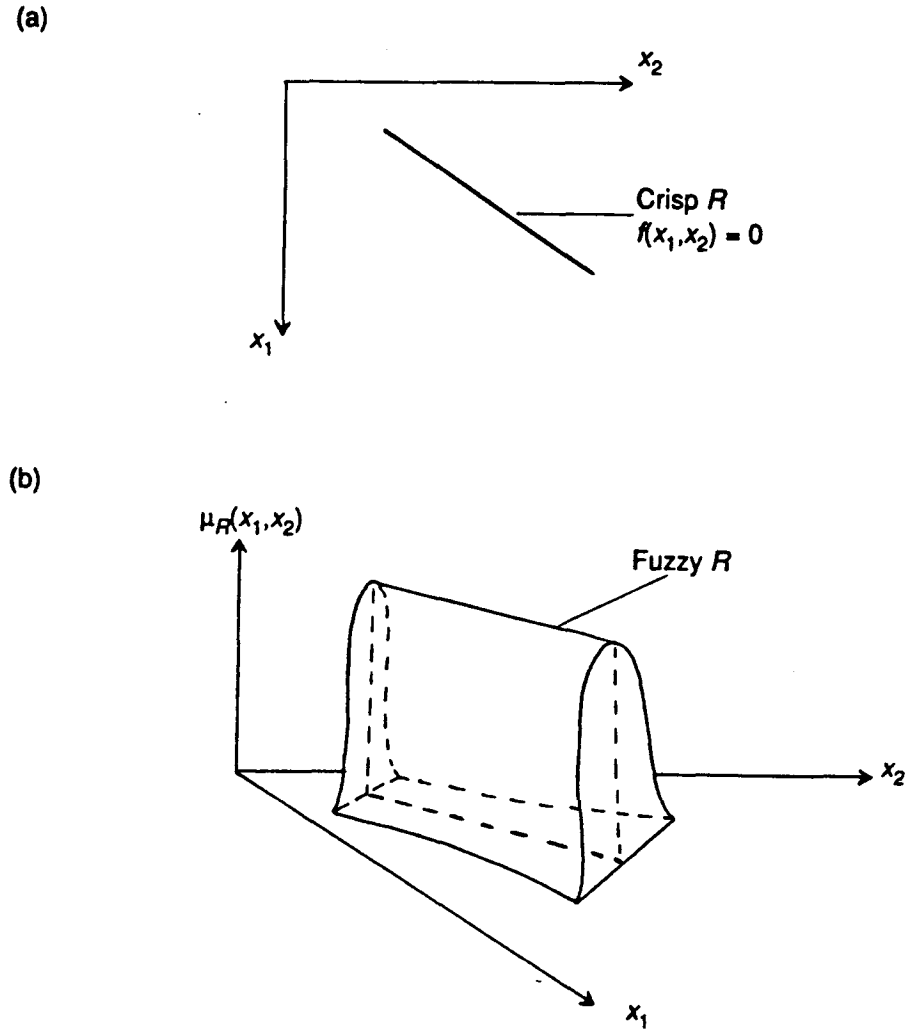


Figure 2.2: Relations in Two Dimensional Space (plane)
(a) A Crisp Relation
(b) A Fuzzy Relation

Cartesian Product of fuzzy Sets

Consider a fuzzy set A_1 in the universe X_1 and a second fuzzy set A_2 in the universe X_2. The cartesian product $A_1 \times A_2$ is then a fuzzy subset of the cartesian product space $X_1 \times X_2$ and its membership function is given by:

$$\mu_{A_1 \times A_2}(x_1, x_2) = min[\mu_{A_1}(x_1), \mu_{A_2}(x_2)] \qquad \forall x_1 \in X_1, \forall x_2 \in X_2 \qquad (2.7)$$

Note that the "min" combination applies here because each element (x_1, x_2) in a cartesian product is formed by taking **both** elements x_1 and x_2 and not just one or the other. This concept can be directly extended to more than two fuzzy sets. An example of a cartesian product of two fuzzy sets is shown in Figure 2.3[23]

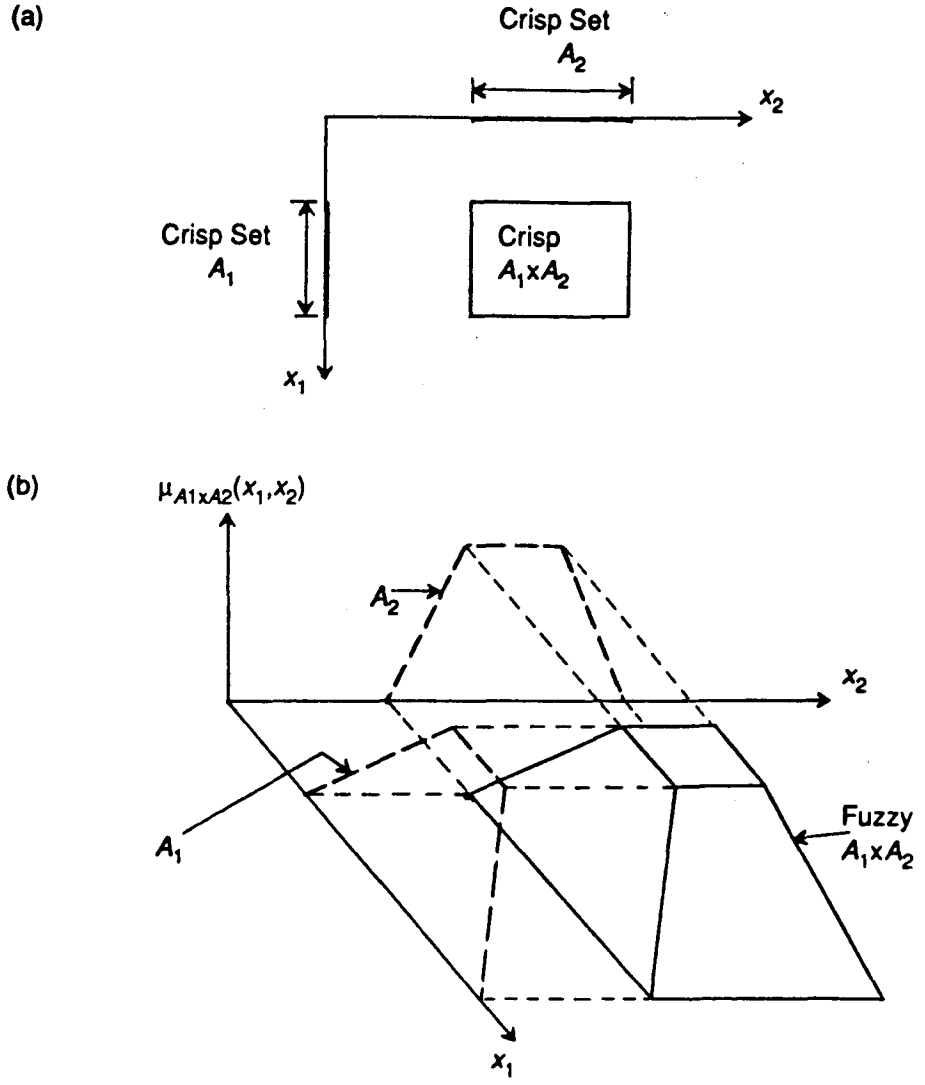


Figure 2.3: Cartesian Product of $A_1 \times A_2$ (relation)
(a) Of Two Crisp Sets
(b) Of Two Fuzzy Sets

Extension Principle

The extension principle was introduce by Zadeh to give a method for extending standard (non-fuzzy) mathematical concepts to their fuzzy counterparts. Consider the crisp relation:

$$y = \mathcal{F}(x_1, x_2, ..., x_r) \tag{2.8}$$

where y are elements in the fuzzy set B and x_i are elements in the universe X_i. Let A_i be fuzzy subsets in X_i. According to the extension principle, the fuzzy set B to which the element y belongs has a membership function given by:

$$\mu_B(y) = sup_{x_1, x_2, ..., x_r}\{min[\mu_{A_1}(x_1), \mu_{A_2}(x_2), ..., \mu_{A_r}(x_r)]\} \tag{2.9}$$

Note that the "min" operation applies first because the relation among A_i is the cartesian product. The "supremum" is applied over the mapping on to B, because more then one combination of $(x_1, x_2, ..., x_r)$ in the fuzzy space $A_1 \times A_2 \times ... \times A_r$ will be mapped to the same element y in the fuzzy subset B and the most possible mapping is the one with the highest membership grade. An example of a mapping from a two-dimensional crisp and fuzzy space $A_1 \times A_2$ on to a one dimensional fuzzy set B is shown in Figure 2.4 (a) and 2.4[23] (b) respectively.

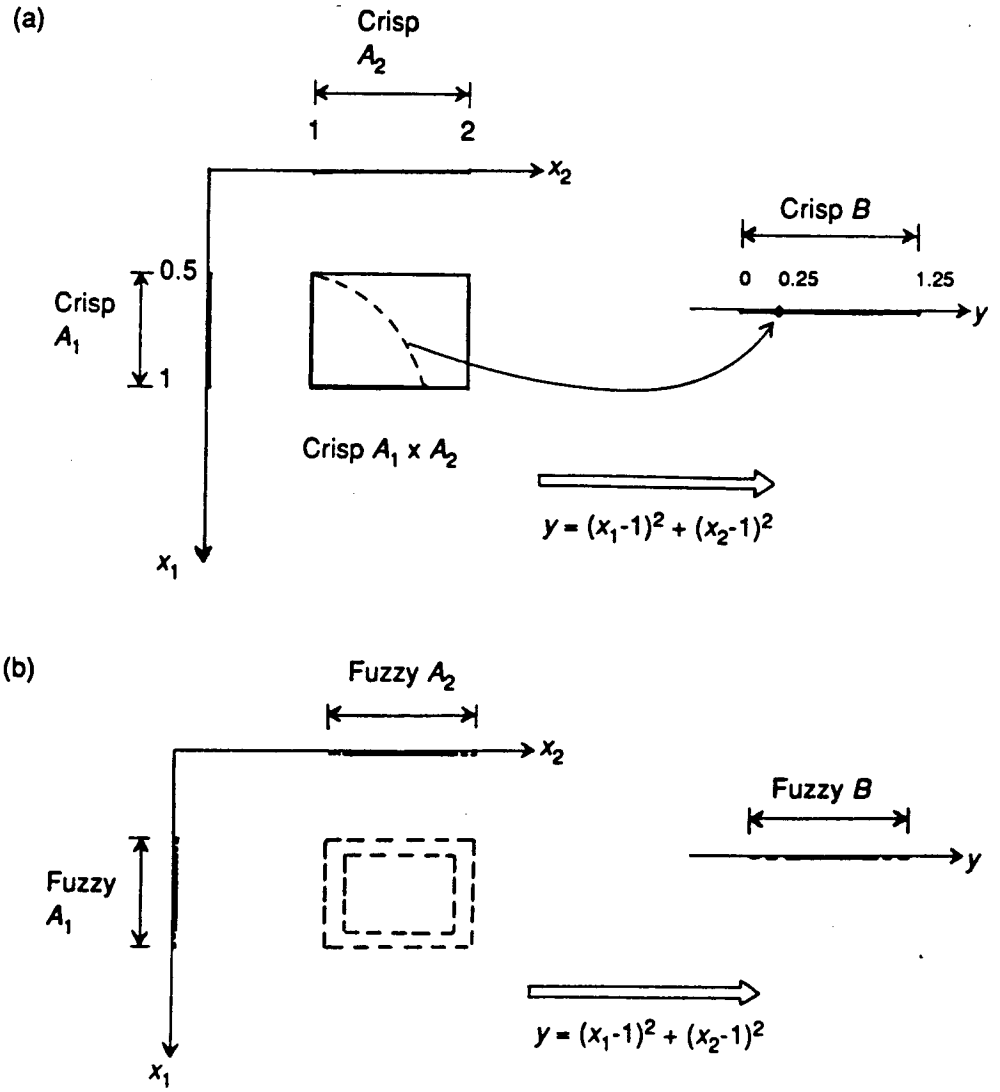


Figure 2.4: A Mapping from a Product Space to a Line
(a) An Example of Crisp Sets
(b) An example of Fuzzy sets
(Extension Principle)

2.6 Composition and Inference

Approximate reasoning is used in fuzzy inference and control. In particular, the *compositional rule of inference* is utilised. We have already introduced the concept of *fuzzy implication.* We shall start the present section by introducing the terms *projection, cylindrical extension,* and *join*, which will lead to the concept of *composition.* Finally the *compositional rule of inference* will be discussed, incorporating all these ideas.

Projection

Consider a fuzzy relation R in the cartesian product space $X_1 \times X_2 \times ... \times X_n$. Suppose that the n indices are arranged as follows:

$$1, 2, ..., n \longrightarrow i_1, i_2, ...i_r, j_1, j_2, ..., j_m \tag{2.10}$$

Note that $n = r + m$ and that i and j denote the newly ordered set of n indices. The projection of R on the subspace $X_{i_1} \times X_{i_2} \times ... \times X_{i_r}$ is denoted by:

$$Proj[R : X_{i_1}, X_{i_2}, ..., X_{i_r}]$$

This is a fuzzy set P and its membership function is given by:

$$\mu_P(x_{i_1}, x_{i_2}, ..., x_{i_r}) = \sup_{x_{j_1}, x_{j_2}, ..., x_{j_m}} \mu_R(x_1, x_2, ..., x_n)$$

The rationale for using the "supermum" operation on the membership function of R should be clear in view of the fact that we have a many-to-one mapping from n dimensional to r dimensional space, with $r < n$.

Cylindrical Extension

Consider the cartesian product space $X_1 \times X_2 \times ... \times X_n$ and, suppose that n indices are arranged as follows:

$$1, 2, ..., n \longrightarrow i_1, i_2, ..., i_r, j_1, j_2, ..., j_m$$

Again note that $r + m = n$ and that i and j denote the newly ordered set of n indices. Now consider a fuzzy relation R in the subspace $X_{i_1} \times X_{i_2} \times ... \times X_{i_r}$.

Its cylindrical extension is denoted by:

$$C_R = \Sigma_{x_1 \times x_2 \times ... \times x_n} \frac{\mu_R(x_{i1}, x_{i2}, ..., x_{ir})}{x_1, x_2, ..., x_n}$$

(See definition in equation 2.1)

Note that a cylindrical extension is a fuzzy set in the n-dimensional space and is the converse of projection. An example is given in Figure 2.5 [23] Here a fuzzy set R in the universe X_1 has been cylindrically extended to a fuzzy set in the cartesian space $X_1 \times X_2$

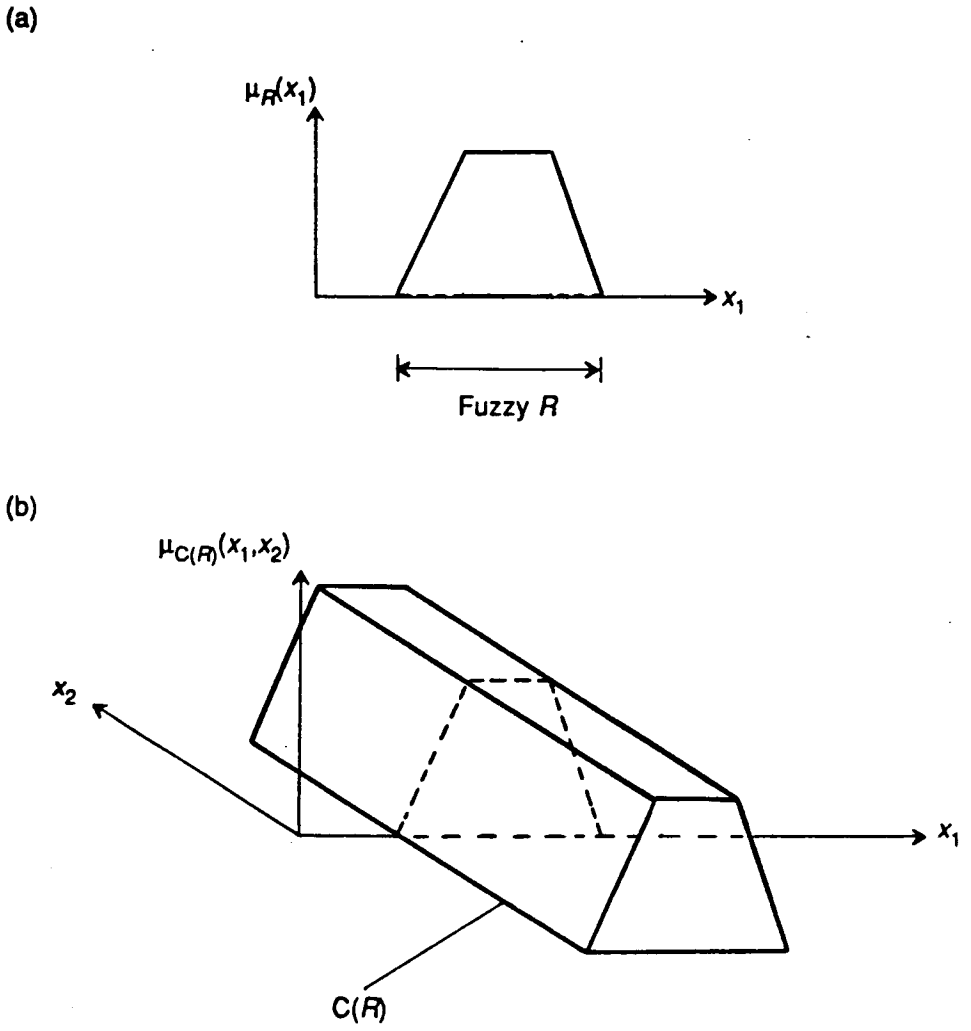


Figure 2.5: The Cylindrical Extension
(a) A Fuzzy Relation (Set)
(b) Its Cylindrical Extension

Join

Consider a fuzzy relation R in the subspace $X_1 \times X_2 \times ... \times X_r$ and a second fuzzy relation S in the subspace $X_m \times X_{m+1} \times ... \times X_n$ such that $m < r + 2$

Note that the union of these two subspaces gives the space $X_1 \times X_2 \times ... \times X_n$. The join of the fuzzy sets R and S is a fuzzy set in the $X_1 \times X_2 \times ... \times X_n$ and is given by the intersection of their cylindrical extensions; thus:

$$Join(R, S) = C(R) \bigcap C(S) \qquad \text{in} \qquad X_1 \times X_2 \times ... \times X_n \tag{2.11}$$

$$\mu_{Join}(X_1 \times X_2 \times ... \times X_n) = \min[\mu_{C_{(R)}}(X_1 \times X_2 \times ... \times X_n)\ ,\ \mu_{C_{(S)}}(X_1 \times X_2 \times ... \times X_n)] \tag{2.12}$$

Note that "min" applies here because the intersection of two fuzzy sets is considered.

Composition

consider a fuzzy relation (fuzzy set) R in the subspace $X_1 \times X_2 \times ... \times X_r$

and a second fuzzy relation (fuzzy set) S in the subspace $X_m \times X_{m+1} \times ... \times X_n$

such that $m < r + 1$. Note that unlike the previous case of *Join*, the two subspaces are never disjoint and hence their intersection is never null. But, as before, the union of the two subspaces gives $X_1 \times X_2 \times ... \times X_n$

The composition of R and S is denoted to by $R \circ S$ and is given by:

$$S \circ R = Proj[Join(R, S)\ ;\ X_1, ..., X_{m-1}, X_{r+1}, ..., Xn] \tag{2.13}$$

Here we take the join of the two sets, as given by equation 2.11 and then project the resulting fuzzy set on the subspace formed by the disjoint parts of the two subspace in which the fuzzy sets R and S are defined.

The membership function of the resulting fuzzy set is obtained from the membership functions of R and S, while noting that "min" applies for Join and "supermum" applies for projection. Specifically,

$$\mu_{S \circ R} = sup_{x_m,\ldots,x_r}[min(\mu_R, \mu_S)] \tag{2.14}$$

Composional Rule of Inference

Rules of the form:

"**If** output Y_1 is y_1 **then if** output Y_2 is y_2 **then** control C is c" are linguistic statements of expert process operators knowledge in which y_1 y_2 and c are fuzzy quantities. These rules are fuzzy relations that employ the fuzzy implication **If-then**.

If we denote the fuzzy relation form of such rules as a fuzzy set R, the output data by a fuzzy set D and the control action by the fuzzy set C, then the composition rule of inference states that:

$$C = D \circ R \tag{2.15}$$

Given the membership functions of the data and of the rule base we can determine the membership function of the control action:

$$\mu_C = sup_Y[min(\mu_D, \mu_R)] \tag{2.16}$$

This result follows directly from equation 2.14. Note that Y denotes the space in which the data D are defined. Furthermore, since R consists of fuzzy implications, its membership function can be formed by the constituent membership functions using the "min" operation.

Chapter 3

SYSTEM DEVELOPMETNT

3.1 Introduction

To implement and evaluate the tuner performance, an experimental system was developed for tuning a commercially available servo-motor system. A computer simulation was developed as well.

First, the general hierarchical structure of the system is described in Section 3.2 . Next, the servomotor level is described in Section 3.3 . The hardware of the servomotor is introduced, and the simulated servomotor is developed. Section 3.4 describes the performance evaluation and the controller parameter updating in the servo expert level. Finally, the fuzzy tuner level is described in section 3.5 .

Servo-motor simulation, input/output considerations, graphics and file management are programmed in ACSL (Advanced Continuous Simulation Language)[1]. Subprograms of the servo expert level are written in FORTRAN. A separate program for the fuzzy tuner was developed in FORTRAN and executed run off line to generate the decision table. Subprograms for the communication between the controller and the servo expert level are written in C.

The same program is used for tuning both the physical and the simulated servo systems. This enables us to switch the servo response and the tuning action from one system to the other in the same execution.

The system software was developed and executed on a PC 386 33[MHZ] equipped

with a mathcoprocessor. The PC 386 serves as an interface between the user and the physical servomotor system.

The description of the experimental system is based on Figure 3.1

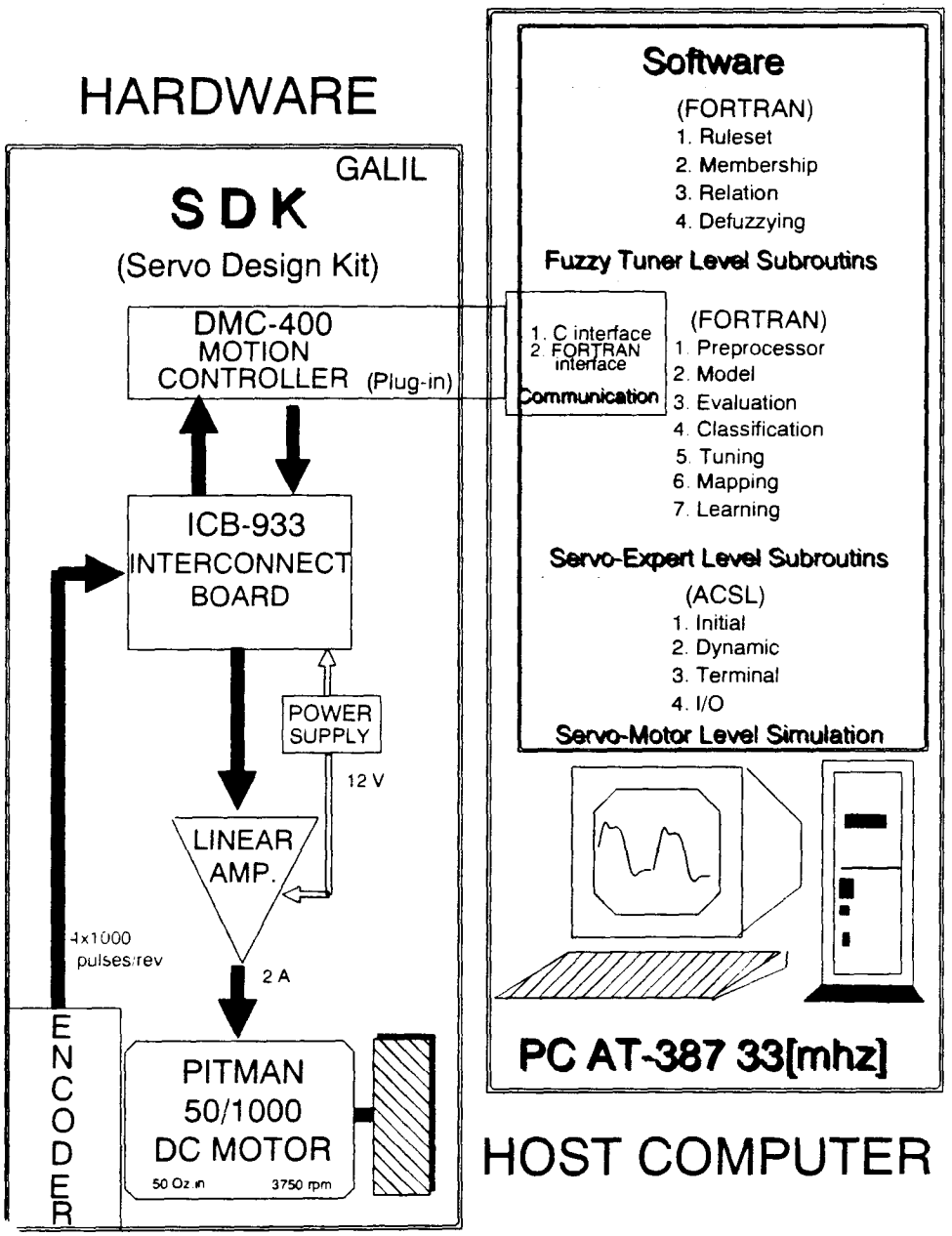


Figure 3.1: The Experimental System

The lefthand block in this figure represents the hardware of the system. Particularly the motor, encoder, amplifier, power supply, controller and the controller.

The righthand block represents the software of the system. It is arranged in three levels as will be described in the next section.

Communication programs interface these two blocks. Particularly, the measured position of the servo-motor is transfered to the servo-expert level subroutines, and the updated controller parameters are returned from that level to the hard controller.

3.2 General Hierarchical Structure

To combine the advantages of the high bandwidth, crisp characteristics of a hard controller in the servomotor system with the merits of a soft knowledge-based tuning a hierarchical structure was developed.

The lowest level is a closed-loop servo-motor system consisting of a D.C. motor to be controlled, and a digital controller which can be programmed to compensate for strong nonlinearities, dynamic decoupling, high-order dynamics and unknown disturbances, as in the case of a robot [23]. A test signal is injected into the controller and the motor response to this signal is entered into the higher level.

In the intermediate level, servo-expert algorithm evaluates time domain performance indices out of the difference between the servo-motor response attributes and a set of pre-defined specifications. These indices trigger tuning actions from a decision (look-up) table. In the same level well-known frequency-domain attributes of a controller are updated and the new parameters of the controller in the lowest level are designed using these updated attributes.

In the top level, knowledge expressed in terms of fuzzy rules is formulated mathematically using fuzzy theory and fuzzy logic operations to generate a decision table in

which the actual performance indices are matched with tuning actions in the intermediate level. A learning and self-organization algorithm may replace or modify the fuzzy logic operation in the generation of the decision table.

A block diagram of the hierarchical tuning structure is shown in Figure 3.2

Notice the conventional inner loop of the low level, hard controller and the outer loop which contain the high level, soft tuner algorithm.

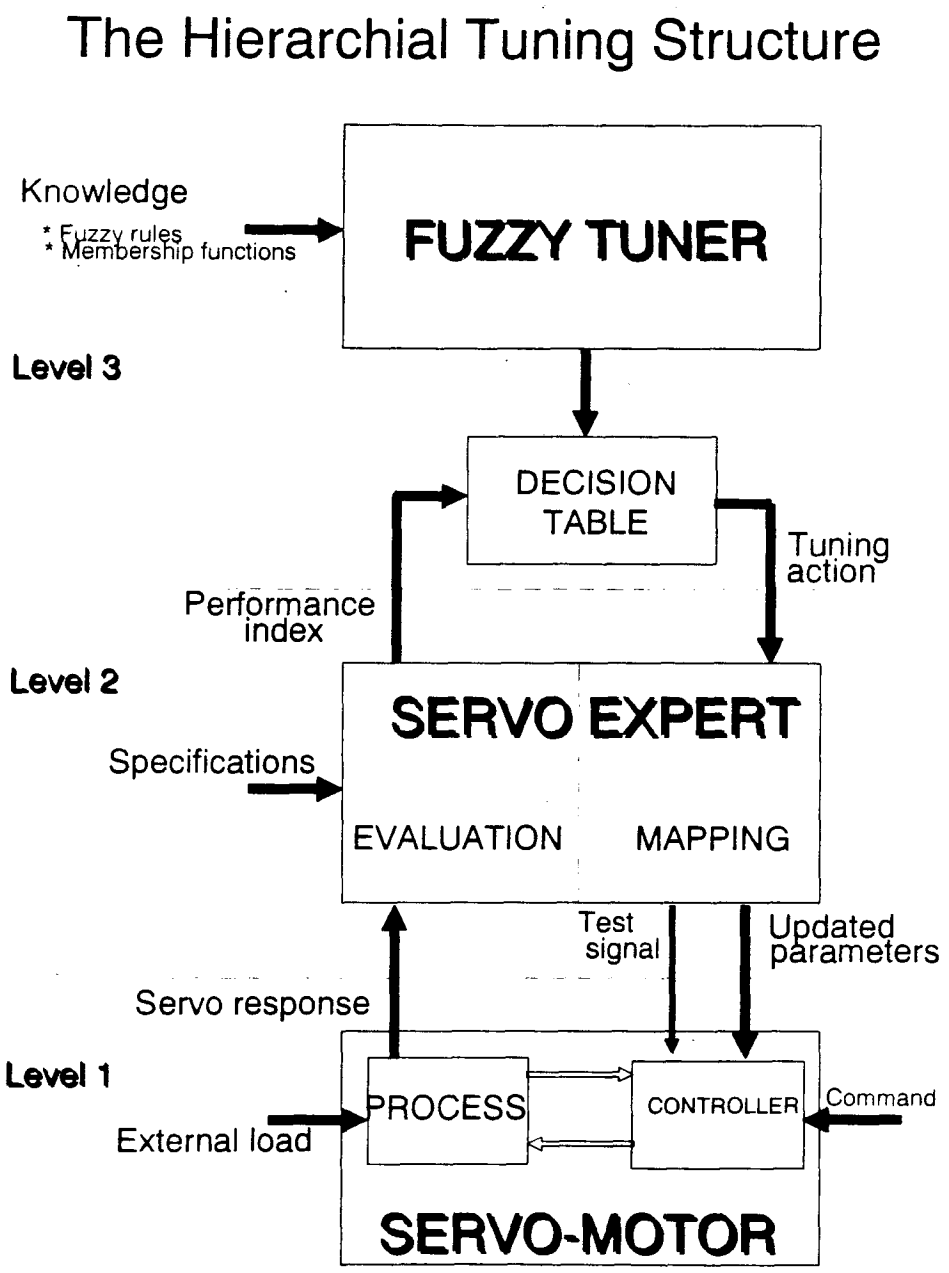


Figure 3.2: Hierarchical Structure of the System

3.3 Servo-Motor Level

In the following subsections physical (commercially available) and simulated servo systems are described and developed.

3.3.1 Physical Servo System

The servo-motor system supplied by "GALIL" consists of the following components:

- Motor
- Amplifier
- Encoder
- Controller
- Power supply
- Interconnected board
- Communication Software

A picture of the GALIL servo-motor hardware is shown in Figure 3.3

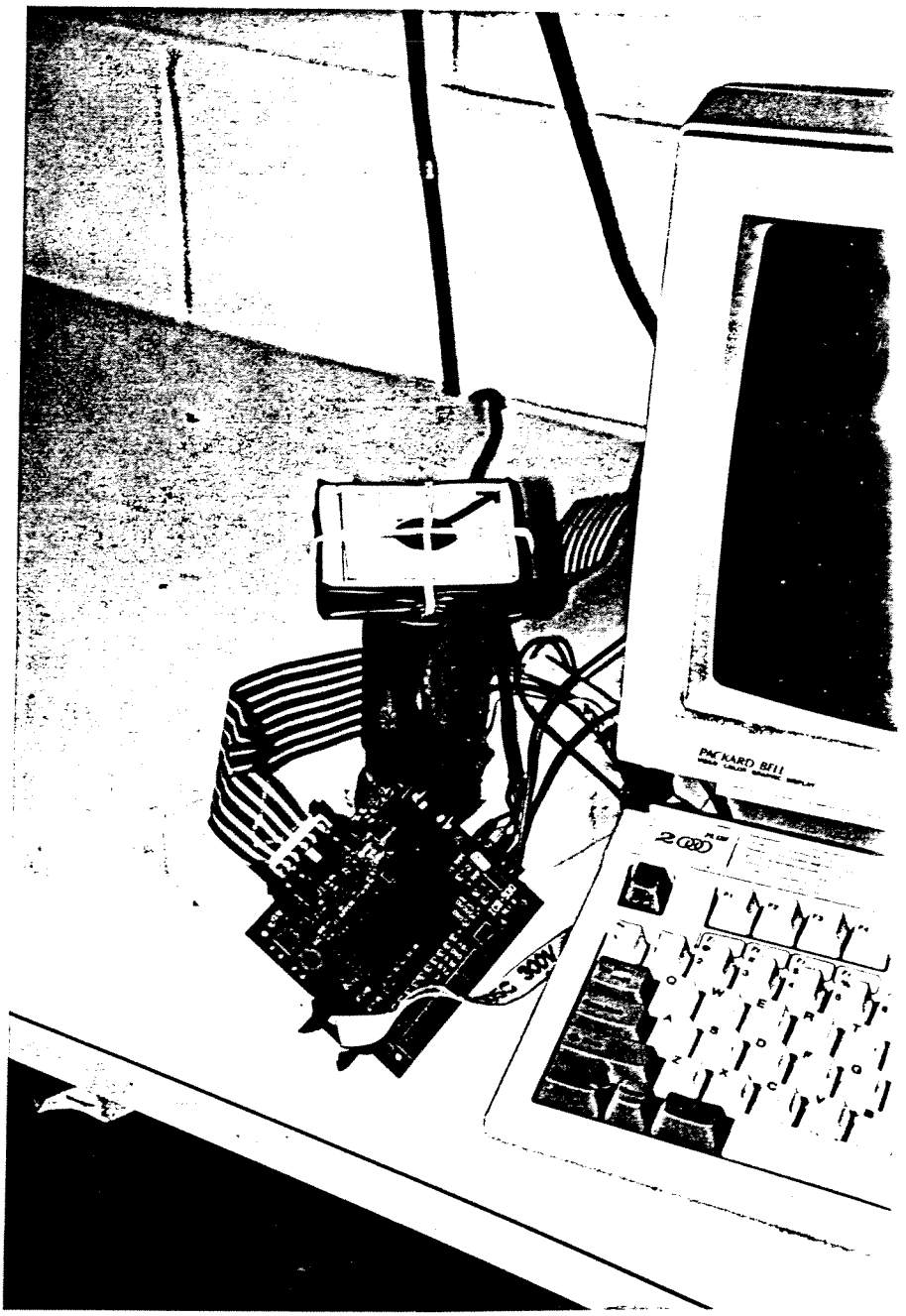


Figure 3.3: The Hardware of the Physical Servo-Motor System

The general specification of the servo-motor system is given in appendix A-1.

Motor

A DC motor, type PITMAN 50/1000 , of torque ranging 50[oz-in] is used. It has a permanent-magnet stator and 3-poles rotor. Its specifications and mechanicl drawing are listed in Appendix A-2.

Amplifier

A linear amplifier is used to drive the motor at constant gain of $0.2[A/V]$ with a dynamic range of $\pm 10[V]$.

Encoder

Incremental optical encoder is used. It generates $4000[pulses/rev]$ to measure the motor position.

Controller

The digital controller is a general-purpose, **programmable** controller of type GALIL DMC-400. It consists of parallel, forward route lead, and integral compensation nets. The controller operates in numerous modes, including point-to-point positioning and jogging. Several commands are provided, including instructions for specifying the motor position, velocity and acceleration.

The controller specifications are listed in Appendix A-3.

Interconnection board

A GALIL ICB-933 is used. It connects the DMC-400 controller to other system elements (motor, encoder, amplifier). The interconnection board specifications are listed in Appendix A-4.

Communication Software

The communication programs provide an interface between the servo expert software and the servo-motor hardware . They translate and send the controller parameters that are computed at the servo expert level, to the DMC-400 controller, and returns the actual actuator position to the main program. In addition they enable the user to interrogate and send commands using the computer keyboard. Two linked sub-programs were developed for communication:

- Sub-program written in FORTRAN, to interface with the FORTRAN coded tuner.
- Sub-program written in C to interface with the C coded GALIL system.

Both programs are listed in Appendix A-5.

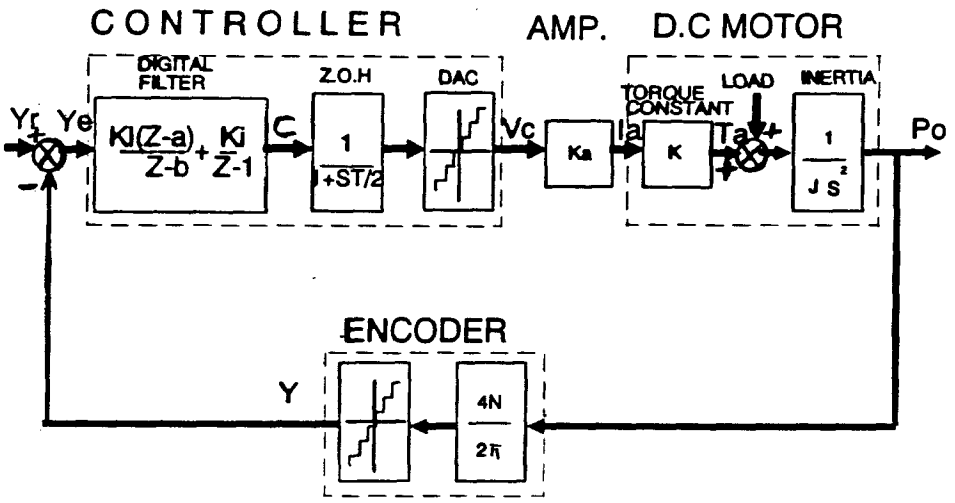


Figure 3.4: Block Diagram of the Simulated Servo-Motor

3.3.2 Simulated Servo-Motor

The simulated servo block diagram is shown in Figure 3.4

It consist of the following modules:

- Motor
- Amplifier
- Encoder
- Controller
- Digital to Analog Converter and Zero Order Hold.

Amplifier and Motor

The mathematical model of the motor-amplifier unit depends on the type of the amplifier. The two types are represented by different transfer functions. Both of them are exmined in the experimental work.

(1) Voltage source amplifier

For a voltage source amplifier the position Po is given by [24]:

$$Po = \frac{K \cdot V}{s \cdot [(L \cdot s + R) \cdot (J \cdot s + B) + K^2]} - \frac{T_l}{s \cdot (J \cdot s + B)} \tag{3.1}$$

where:

K - Motor torque constant$[N * m/amp]$

V - Supply voltage to the stator$[volt]$

J - The actuator moment of inertia$[Kg \cdot m^2]$

L - Armature inductance$[mH]$

R - Armature resistance$[ohm]$

B - Motor mechanical damping$[Kg/s]$

T_l - External Load$[N \cdot m]$

s - Laplace variable

(2) Current source, Current feedback amplifier

A current source, current feedback amplifier results in a different transfer function [32] for the actuator/amplifier, as given below:

$$Po = \frac{(A \cdot K \cdot Ii - T_l)}{J \cdot s^2} \tag{3.2}$$

here Ii is the input current to the motor. A, the effective gain of the current source amplifier is:

$$A = \frac{I_{max}}{V_{max}} \; [amp/volt] \tag{3.3}$$

in which V_{max} is the maximum available input voltage to the amplifier and I_{max} is the maximum output current to the actuator:

Encoder

The encoder generates N pulses per revolution, with two signals in quadrature giving $4N$ pulses per revolution. For $N = 1000$ the effective gain, given by the ratio of the encoder output in pulses to the actual motor rotation in radians (P) is:

$$Ge = \text{Integer}\left(\frac{Y}{P}\right) = \text{Integer}\left(\frac{4 \cdot 1000N}{2\pi}\right) \tag{3.4}$$

Controller

The digital controller consists of a lead compensator with a single zero, single pole and an integrator in parallel route.

Its transfer function $Gc_{(z)}$ in the discrete time domain is given by:

$$Gc_{(z)} = \frac{C}{(Yr - Y)} = \frac{Kl_z \cdot (z - Zr_z)}{(z - Pl_z)} + \frac{Ki_z}{(z - 1)} \tag{3.5}$$

where:

C - Position command to the amplifier[*counts*]

Yr - required position[*counts*]

Kl_z - Lead compensator gain.

Ki_z - Integrator gain.

Zr_z - Zero location in z domain.

Pl_z - Pole location in z domain.

z - Z transform variable.

Using bi-linear transformation $s = \frac{2}{Ts}\frac{(z-1)}{(z+1)}$ the equivalent continuous transfer function in s domain becomes:

$$Gc_{(s)} = Kl_s \cdot \frac{(s + Zr_s)}{(s + Pl_s)} + \frac{Ki_s}{s} \tag{3.6}$$

where:

$$\begin{aligned} Kl_s &= Kl_z \cdot \tfrac{(1+Zr_z)}{(1+Pl_z)} \\ Zr_s &= \tfrac{2}{Ts}\tfrac{(1-Zr_z)}{(1+Zr_z)} \\ Pl_s &= \tfrac{2}{Ts}\tfrac{(1-Pl_z)}{(1+Pl_z)} \\ Ki_s &= \tfrac{Ki}{Ts} \end{aligned} \tag{3.7}$$

and Ts is the sampling time.

Digital to Analog Converter (DAC)

This is 8-bit DAC quantizes the full range output ($20[V]$)of the controller to 256[bit]. The quantization is $\frac{20}{256}[volt/bit]$.

3.4 Servo Expert Level

The servo-expert level interfaces the hard controller of the servo-motor in the lowest level and the soft, knowledge-based tuner in the top level. An adjustment mechanism automatically tunes the parameters of the programable controller using measured response of the actual servo-motor and the decision table, so as to meet a set of desired performance parameters within a set of acceptable pre-defined tolerances.

This level consists of eight functions as shown in Figure 3.5:

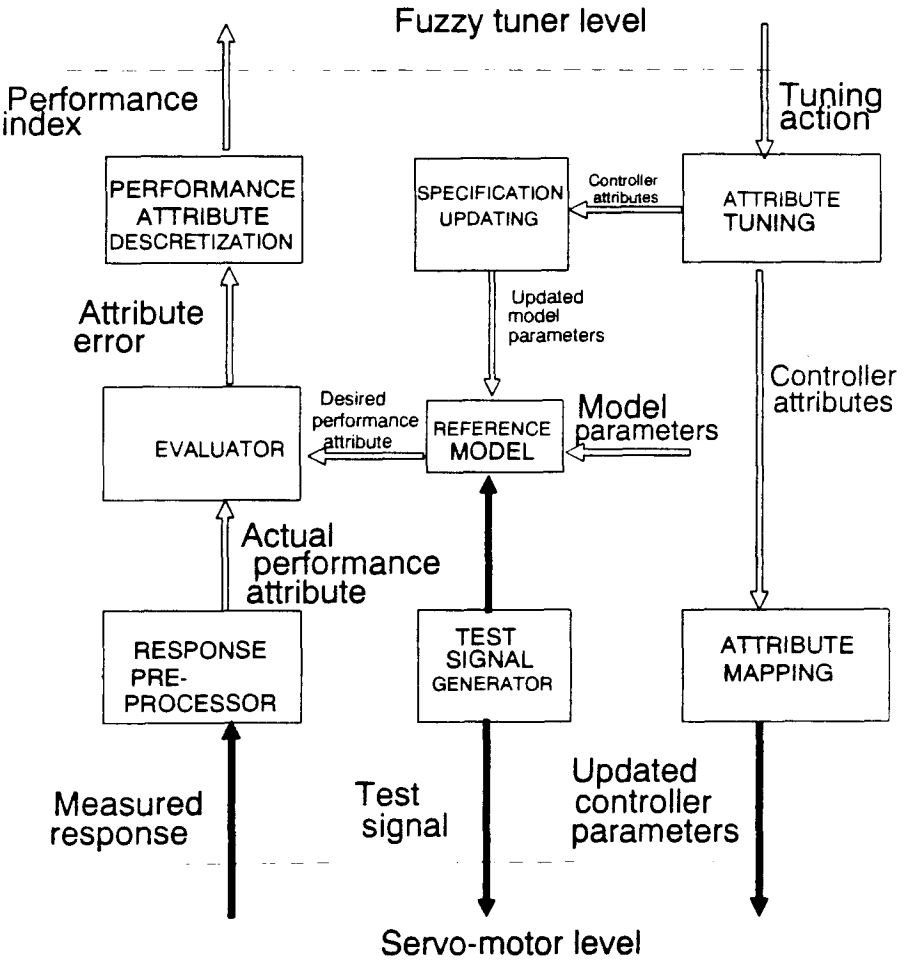


Figure 3.5: Block Diagram of the Servo Expert Level

- Test signal generator.
- Performance specification.
- Response preprocessing.
- Performance evaluation.
- Performance classification.
- Controller attribute tuning.
- Controller-parameter mapping.
- Specification updating.

Test signal is injected **simultaneously** to both the servo-motor and the reference model. The servo response to this signal is preprocssed to obtain time domain performance parameters. These parameters are compared with those of the reference model performance parameters and the error between them is classified to form performance index which triggers a tuning action by means of the decision table. The developement of the decision table is described in the next section.

The tuning actions update the frequency-domain controller attributes from which the updated controller parameters are calculated using crisp, conventional mapping. The reference model may be modified automatically, if necessary, when the limits of the controller attributes are reached and the tuning process terminated unsuccessfully.

3.4.1 Performance Specification

Several time domain parameters are useful for performance specification. These desired time domain performance parameters are represented in terms of a reference model with a set of acceptable tolerances.

Even though the user specifications can be expressed directly in terms of time domain performance parameters, a reference model is used to determine these parameters indirectly for the following reasons:

Since the tuning knowledge base is determined through response observation, it is found visually more convenient to evaluate the response of the actual system relative to that of a reference model rather than evaluate an "absolute" performance of the system. Furthermore the model provides some buffer against unreasonable requirements. Finally using reference model make the tuning process similar in a way to the Model Reference Adaptive System (MRAS) technique.

It is important to realize that this model represents the **desired** performance only, and **it is not a the model of the system**. Furthermore, unlike MRAS technique, the goal of the tuning process is to bring the **performance parameters** and not the **response** itself close to those of the model.

The order, structure and parameters of the model can be chosen and adjusted either analytically or by trial and error so as to meet the performance parameters specification in the time domain, and can be updated automatically, if necessary, during the tuning action.

To save real time computation either closed form analytical expressions or off-line numerical solution should be used to calculate the time domain parameters of the model.

Reference Model

The dynamics of the reference model in this research is represented by a second order transfer function, specified by the undamped natural frequency (ω_n) and the damping ratio (ζ). The static behavior of the model is represented by a deterministic steady-state error (θ) and can represent the undeterministic error as well.

The model transfer function is:

$$Gm_{(s)} = \frac{Ym}{Yr} = \frac{w_n^2}{s^2 + 2\zeta\omega_n \cdot s\omega_n^2} + \Theta \tag{3.8}$$

The solution to step input Yr is:

$$Ym = Yr \cdot [1 - a \cdot e^{-b \cdot t} \cdot \sin(c \cdot t + d) + \theta + \epsilon] \tag{3.9}$$

where:

$a = \frac{1}{\sqrt{1-\zeta^2}}$

$b = \zeta \cdot \omega_n$

$c = \omega_n \cdot \sqrt{1 - \zeta^2}$

$d = cos^{-1}(\zeta)$

The subroutine MODEL is listed in Appendix. B.2.1

Time Domain Performance Specifications

The following time domain parameters of the model are used to define the desired servo-motor performance.

- Rise time
- Damped natural frequency
- Average damping ratio
- Overshoot
- Offset

These parameters are calculated analytically from equation 3.9 and compared with those of the servo-motor to form performance parameter error.

1. **Rise-Time** - $\mathcal{RT}_m$

   The time taken to pass 0.95 of the steady-state response value for the first time.

2. **Model Damped Natural Frequency** - $\mathcal{DF}_m$

$$\mathcal{DF}_m = \omega_n \cdot \sqrt{1-\zeta^2} \tag{3.10}$$

3. **Model Average Damping Ratio** - $\overline{\mathcal{DR}_m}$

$$\overline{\mathcal{DR}_m} = \frac{1}{\mathcal{P}} \cdot \Sigma_{p=1}^{\mathcal{P}} \frac{\gamma_{m_{(p+1)}} - \gamma_{m_{(p)}}}{\gamma_{m_{(p)}} - \gamma_{m_{(p-1)}}} \tag{3.11}$$

where the p'th peak level $\gamma_{m_{(p)}}$ of under-dumped model ($\zeta < 1$) is calculated analytically using:

$$\gamma_{m_{(p)}} = Yr \cdot [(1 - a \cdot e^{-b\tau_{m_{(p)}}} \cdot \sin(c \cdot \tau_{m_{(p)}} + d)] \tag{3.12}$$

in which the time $\tau_{m_{(p)}}$ of the p'th peak is:

$$\tau_{m_{(p)}} = \frac{p \cdot \pi}{\omega_n \cdot \sqrt{(1-\zeta^2)}} \qquad \text{for} \qquad p = 1, 2, ..., \mathcal{P} \tag{3.13}$$

and a, b, c and d are as defined in equation 3.9

Peaks smaller than a predefined value are neglected.

4. **Model Overshoot** - $\mathcal{OS}_m$

   The level of the first peak

$$\mathcal{OS}_m = \gamma_{m_{(1)}} = \exp \frac{-\pi \cdot \zeta}{\sqrt{1-\zeta^2}} \tag{3.14}$$

5. **Model Steady State Error (Offset) - $\mathcal{OF}_m$**

The difference between the desired and the actual response at steady state.

$$\mathcal{OF}_m = \Theta \tag{3.15}$$

These desired performance are arranged in vector $Pm_{(i)}$:

$$\begin{aligned} Pm_{(i)} &= \mathcal{RT}_m \\ Pm_{(i)} &= \mathcal{DF}_m \\ Pm_{(i)} &= \overline{\mathcal{DR}}_m \\ Pm_{(i)} &= \mathcal{OS}_m \\ Pm_{(i)} &= \mathcal{OF}_m \end{aligned} \tag{3.16}$$

3.4.2 Response Preprocessor

The preprocessor calculates the time domain performance parameters of the servo-motor measured response.

Peak detector observes the response to find $\mathcal{P}$ peak level $\gamma_{s_{(p)}}$ and timing $\tau_{s_{(p)}}$ using the sign change between any two consecutive pairs $Y_{(t+1)} - Y_{(t)}$ and $Y_{(t)} - Y_{(t-1)}$ where $Y_{(t)}$ is the servo-motor response at the communication time interval.

Peak levels which are found to be less than a predefined value are ignored, and the sign changing is checked for continuity.

Time domain performance parameters are evaluated out of the servo-motor response, the peak levels and peak times. These parameters are compared later with those of the reference model to form the error of the performance parameters. Furthermore, these parameters may be compared with those used in other tuning techniques.

Time domain performance parameters of the Servo-motor are calculated as follows:

1. **Rise-Time** - $\mathcal{RT}_s$

   The time taken to pass the 95 % of the steady-state response value for the first time. A high $\mathcal{RT}_s$ indicates slow response.

2. **Average Damped Natural Frequency** - $\overline{\mathcal{DF}}_s$

   The average damped natural frequency is calculated using consecutive peak times:

$$\overline{\mathcal{DF}}_s = \frac{1}{\mathcal{P}} \Sigma_{p=1}^{\mathcal{P}} \frac{\pi}{\tau_{s_{(p)}} - \tau_{s_{(p-1)}}} \tag{3.17}$$

   for all:

   $\gamma_{s_{(\mathcal{P})}} > 0.02 \cdot Yr$

   where $\mathcal{P}$ is the number of peaks having magnitude greater than a predefined value. A high average damped natural frequency indicates a fast response.

3. **Average Damping Ratio** - $\overline{\mathcal{DR}}_s$

   The average ratio of each consecutive pair of response peaks gives:

$$\overline{\mathcal{DR}}_s = \frac{1}{\mathcal{P}} \Sigma_{p=1}^{\mathcal{P}} \frac{\gamma_{s_{(p+1)}} - \gamma_{s_{(p)}}}{\gamma_{s_{(p)}} - \gamma_{s_{(p-1)}}} \tag{3.18}$$

   Average damping ratio less then 1 indicates oscillatory convergence. Average damped ratio over 1 indicates unstable, oscillatory response.

4. **Overshoot** - $\mathcal{OS}_s$

   The level of the first peak:

$$\mathcal{OS}_s = \gamma_{s_{(1)}} \tag{3.19}$$

   Small overshoot indicates less oscillatory response.

5. **Steady state error** - $\mathcal{OF}_s$

   The normalized, **systematic** (deterministic) deviation of the steady-state value from the desired value:

$$\mathcal{OF}_s = \frac{1}{t_f - \tau_{(\mathcal{P})}} \Sigma_{\tau_{(\mathcal{P})}}^{t_f} \frac{(Ys_{(t)} - Yr) \cdot \Delta t}{Yr} \tag{3.20}$$

   were t_f is the duration of the input test signal and Δt is the communication time interval.

These actual performance are arranged in vector $Ps_{(i)}$:

$$\begin{aligned} Ps_{(1)} &= \mathcal{RT}_s \\ Ps_{(2)} &= \mathcal{DF}_s \\ Ps_{(3)} &= \overline{\mathcal{DR}}_s \\ Ps_{(4)} &= \mathcal{OS}_s \\ Ps_{(5)} &= \mathcal{OF}_s \end{aligned} \tag{3.21}$$

3.4.3 Performance Evaluation

The time domain performance parameters of the servo-motor are now compared with those of the model and normalized to form a set of nondimensional performance parameter errors which takes values in the interval $[-\infty, 1]$:

Negative error indicates a servo performance better than that of the model.

Zero error indicates a servo performance equal to that of the model.

Positive error indicates a servo performance less than that of the model.

The following normalized, nondimensional errors are calculated using the actual servo-motor and model performance parameters:

1. **Rise Time** - $\mathcal{RT}_e$

$$\mathcal{RT}_e = 1 - \frac{\mathcal{RT}_m}{\mathcal{RT}_s} \tag{3.22}$$

Error approching 1 indicates non responding system ($\mathcal{RT}_s \Rightarrow \infty$)

2. **Damped Natural Frequency Error** - $\mathcal{DF}_e$

$$\mathcal{DF}_e = 1 - \frac{\overline{\mathcal{DF}}_s}{\mathcal{DF}_m} \tag{3.23}$$

Error approaching 1 ($\overline{\mathcal{DF}}_s \Rightarrow 0$) indicates critically, over-damped, non-oscillating system.

3. **Damping Ratio Error** - $\mathcal{DR}_e$

$$\mathcal{DR}_e = 1 - \frac{\overline{\mathcal{DR}}_m}{\overline{\mathcal{DR}}_s} \tag{3.24}$$

Error approaching 1 ($\mathcal{DR}_s \Rightarrow \infty$) indicates unstable, oscillatory system.

4. **Overshoot** - $\mathcal{OS}_e$

$$\mathcal{OS}_e = 1 - \frac{\mathcal{OS}_m}{\mathcal{OS}_s} \tag{3.25}$$

5. **Steady State Error (offset)** - $\mathcal{OF}_e$

$$\mathcal{OF}_e = 1 - \frac{\mathcal{OF}_m}{\mathcal{OF}_s} \tag{3.26}$$

These normalized, nondimensional errors of the performance parameter are arranged in a vector $\mathcal{ERR}_{(i)}$:

$$
\begin{aligned}
\mathcal{ERR}_{(1)} &= \mathcal{RT}e && \text{Rise Time Error} \\
\mathcal{ERR}_{(2)} &= \mathcal{DF}e && \text{Damped Natural Frequency Error} \\
\mathcal{ERR}_{(3)} &= \mathcal{DR}e && \text{Average Damping Ratio Error} \\
\mathcal{ERR}_{(4)} &= \mathcal{OS}e && \text{Overshoot Error} \\
\mathcal{ERR}_{(5)} &= \mathcal{OF}e && \text{Offset Error}
\end{aligned}
\tag{3.27}
$$

3.4.4 Performance Index Classification

Since in the present application the knowledge on the response behaviour is expressed in terms of discrete and finite number of performance quantities, the input to the decision table should be discrete with a finite cardinality. Therefore the performance parameter errors are now assigned discrete performance indices. Let us define an I-dimensional universe of performance, in which each dimension i represents a performance index stored as vector $\mathcal{K}_{(i)}$.

We obtain these indices by classifying each of the i performance parameter errors into j subsets, separated by $j-1$ predefined thresholds stored in vector $\mathcal{TH}_{(j)}$, and assigning an integer number (index) to each subset.

Given the performance parameter error at every communication time, as calculated in the previous subsection, a performance index is computed for every condition variable. Notice that since the performance parameter errors are computed accurately the resulting performance indices are crisp and not fuzzy variables (or fuzzy variables having unity membership grade).

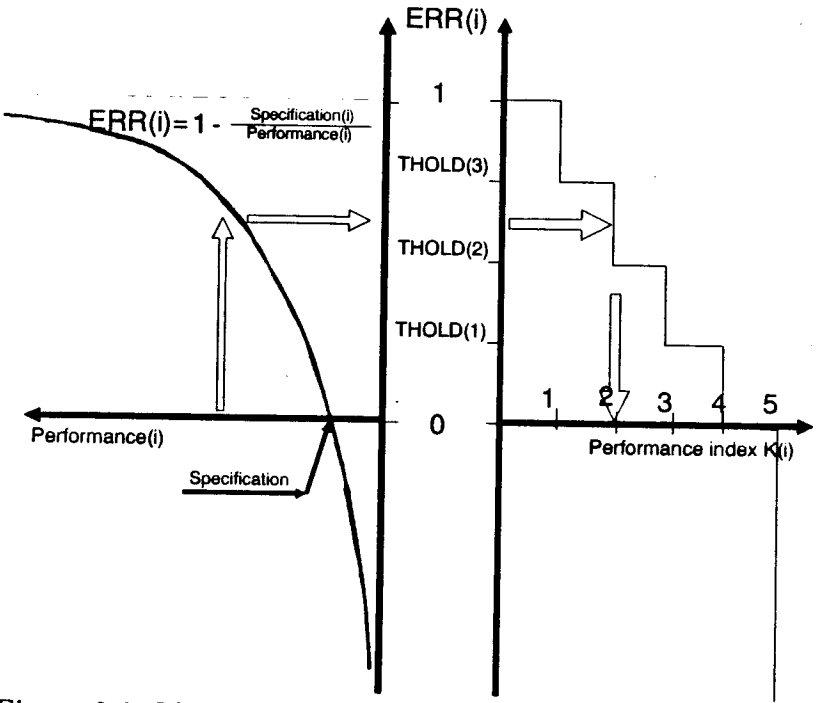


Figure 3.6: Mapping from Performance to performance Index

$$\mathcal{K}_{(i)} = \begin{cases} 5 & \mathcal{ERR}_{(i)} < 0. \\ 4 & 0 \leq \mathcal{ERR}_{(i)} < \mathcal{TH}_{(1)} \\ 3 & \mathcal{TH}_{(1)} \leq \mathcal{ERR}_{(i)} < \mathcal{TH}_{(2)} \\ 2 & \mathcal{TH}_{(2)} \leq \mathcal{ERR}_{(i)} < \mathcal{TH}_{(3)} \\ 1 & \mathcal{TH}_{(3)} \leq \mathcal{ERR}_{(i)} < 1. \end{cases} \tag{3.28}$$

In this way, as a part of the performance specification, $\mathcal{TH}_{(1)}$ can be defined as the **acceptable tolerance** of a performance index. Notice that the first set are semi-bounded since over-specification performance are unlimited.

The process of evaluatation and classification of the response is demonstrated schematically in Figure 3.6

These performance indices are the entries to the pre-calculated decision table. The output of this look-up table is a tuning action used to update the controller attributes as will be explained in the next subsection.

Following the outer tuning loop we proceed with the description on the updating of the controller parameters, leaving the development of the decision table to the next section.

3.4.5 Tuning of the Controller Attributes

Keeping in mind that the off-line computed decsion table is already avaliable, the frequency domain attributes of the controller are now updated using tuning actions that are fired from the decision table, triggered by the performance indices.

We begin with the meaning of the frequency domain attributes controller. Usually controller attributes are chosen so as to meet a predefined set of speed, stability and accuracy specifications. Let $Gc_{(j\omega)}$ and $Gp_{(j\omega)}$ be the transfer functions of the controller and the plant (motor, amplifier and encoder) respectively in the frequency domain. Define Cross-Over Gain Frequency (ω_{cog}) as the frequency at which the magnitude of the controlled, open-loop gain equals 1. (0[db]):

$$|Gc_{(j\omega_{cog})} \cdot Gp_{(j\omega_{cog})}| = 1$$

As a rule of thumb, in many control systems the frequency bandwidth ω_{bw} which is a measure for the speed of the system is related to the cross-over gain frequency by:

$$\omega_{bw} \approx 1.5 - 2.0 \quad \text{of} \quad \omega_{cog}$$

Controller design procedure based on this rule is as follows:

First, as a first guess, choose the cross-over frequency ω_{cog} to be 0.7 of the desired bandwidth ω_{bw}. Next, from the frequency response of the plant, design a controller such

that its phase lead angle $arg(Gc_{(j\omega)})$ would reach its maximum value at ω_{cog} to meet the stability requirement *Phase Margin.*

Finally select the controller gain such that

$$|Gc_{(j\omega_{cog})}| \cdot |Gp_{(j\omega_{cog}}| = 1$$

or:

$$|Gc_{(j\omega_{cog})}| = \frac{1}{|Gp_{(j\omega_{cog}}|}$$

If the closed-loop gain is too low in the very low frequency (results in steady state errors), an integrator may be added to the controller. In this case we define a low frequency cross over gain ωl_{cog} at which the controller gain is the same as the cross-over gain:

$$|Gi_{(j\omega l_{cog})}| = |Gc_{(j\omega_{cog})}|$$

where Gi is the transfer function of the integrator branch. The lower is ωl_{cog} the slower is the decaying of the steady state error to zero.

We now use this design procedure to define four controller attributes to be tuned, arranged as vector $\odot_{(l)}$:

$$\begin{aligned}
&\odot_{(1)} = \phi_{cof} && \text{Phase lead angle at cross-over gain freqency}\\
&\odot_{(2)} = \omega_{cog} && \text{Frequency at cross-over gain}\\
&\odot_{(3)} = \psi_{cof} && \text{Gain at cross-over frequency}\\
&\odot_{(4)} = \omega l_{cog} && \text{Low frequency at cross-over gain}
\end{aligned} \tag{3.29}$$

These attributes that shape the frequency response of the system demonstrated in Figure 3.7

When a tuning action is taken the controller attributes are updated and translated to updated parameters of the controller. The relation used for updating the controller attribute is:

$$\odot_{(l)_{new}} = \odot_{(l)_{old}} + \Delta\odot \cdot (\odot_{(l)_{max}} - \odot_{(l)_{min}}) \cdot \odot_{(l)_{sen}} \tag{3.30}$$

where the subscript "*new*" denotes updated value and "*old*" denotes previous value. The incremental tuning action $\Delta\odot$ is the output of the decision table and the parameter $\odot_{sen}$ is a sensitivity parameter. The subscripts "*max*" and "*min*" denote the upper and lower bounds of the controller attributes.

Since more than one incremental tuning action value might be triggered by different performance parameters through the decision table for each controller attribute, the strategy of **firing the performance parameter with the lowest performance index** is adopted, even though another strategy would suggest some other priority.

The mapping of this updated attributes to the parameters of the controller is described in the next subsection.

3.4.6 Mapping the Attributes of the Controller to its Parameters

Using the updated controller attributes, its parameters are calculated now in Laplace domain, transformed to the discrete Z domain using bi-linear transformation and, finally, translated to the DMC400 controller format. This mapping process is described now.

Transfer function of a simple lead compensation network can be written in the frequency domain as:

$$Gc_{(j\omega)} = K_l \cdot \frac{1 + A \cdot j\omega}{1 + A \cdot \alpha \cdot j\omega} \qquad 0 < \alpha < 1 \tag{3.31}$$

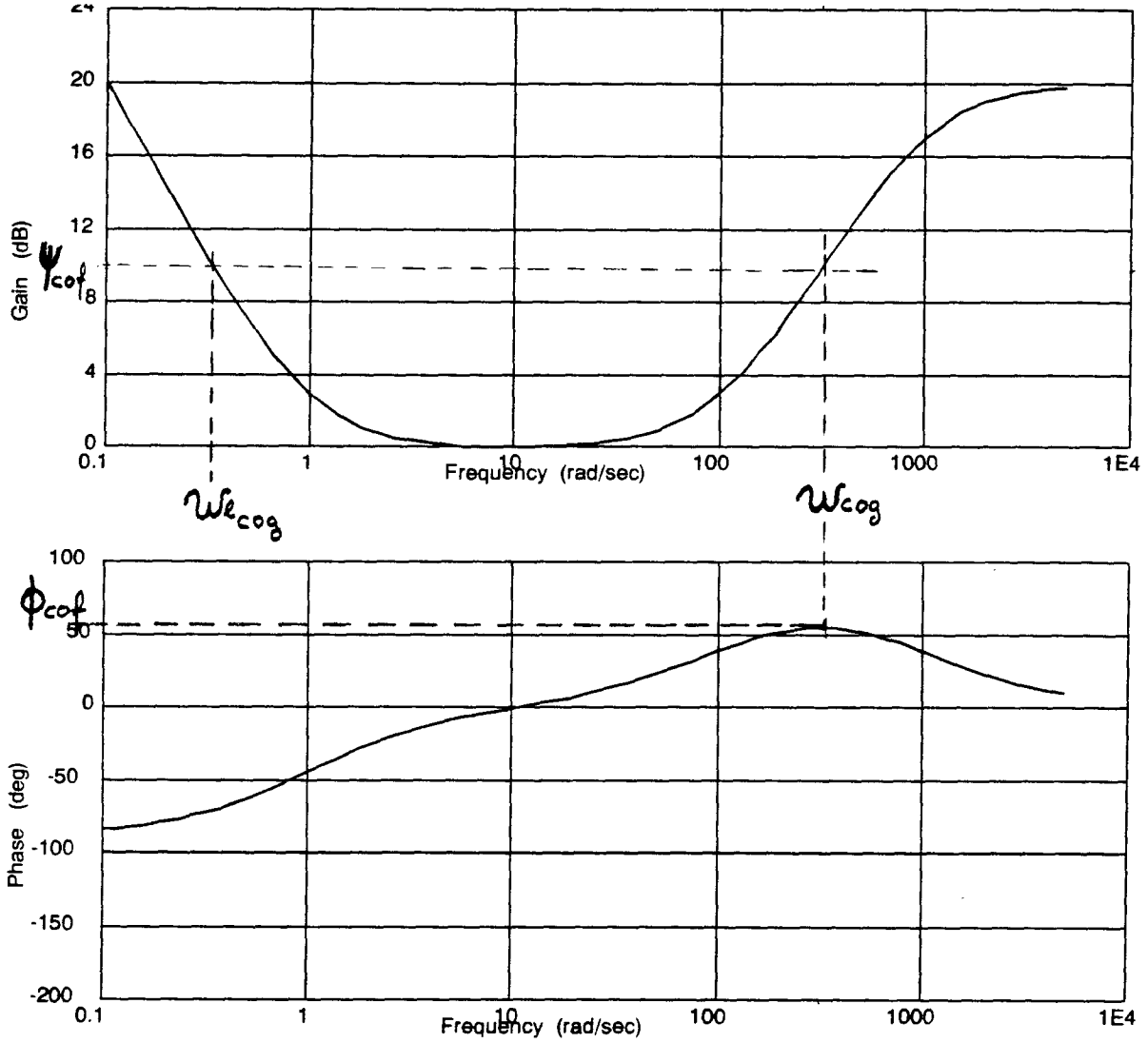


Figure 3.7: Controller Attributes Definition

where the transfer function "zero" is located at $\frac{1}{A}$ and the transfer function "pole" at $\frac{1}{A\alpha}$ on the frequency axis.

The phase lead angle $\phi_{(j\omega)}$ of this circuit is:

$$arg[Gc_{(\omega)}] = \Phi_{(\omega)} = tan^{-1}(A \cdot \omega) - tan^{-1}(A \cdot \alpha \cdot \omega) \tag{3.32}$$

To find the maximum phase lead angle ϕ_{max} and the frequency ω_{max} in which this angle occurs as a function of α and$A$, set the derivative of $\phi_{(\omega)}$ in equation 3.32 to 0.

$$\frac{d\phi_{(\omega)}}{d\omega} = \frac{A}{1+\omega^2A^2} - \frac{\alpha A}{1+\omega^2\alpha^2A^2} = 0 \tag{3.33}$$

Then ω_{max} is obtained as:

$$\omega_{max} = \frac{1}{A\sqrt{\alpha}} \tag{3.34}$$

and the corresponding ϕ_{max} is:

$$\phi_{max} = \tan^{-1}\left(\frac{1}{\sqrt{\alpha}}\right) - \tan^{-1}\left(\sqrt{\alpha}\right) = \sin^{-1}\left(\frac{1-\alpha}{1+\alpha}\right) \tag{3.35}$$

Given ϕ_{max} and ω_{max}, α and A can be calculated from equations 3.34 and 3.35:

$$\alpha = \frac{1 - \sin\left(\phi_{max}\right)}{1 + \sin\left(\phi_{max}\right)} \tag{3.36}$$

$$A = \frac{1}{\sqrt{\alpha} \cdot \omega_{max}} \tag{3.37}$$

and the "Zero" and the "Pole" location from equations 3.31

The lead network gain:

$$K_l = \sqrt{\alpha} \cdot |Gc_{(j\omega_{max})}| \tag{3.38}$$

We see that the parameters of the lead net K_l, α and A depend on the controller attributes ϕ_{max}, ω_{max} and $|Gc_{(j\omega_{max})}|$

Designing the controller such that the maximum phase lead angle occurs at the cross-over gain frequency we obtain:

$$\omega_{cog} = \omega_{max}\phi_{cof} = \phi_{max}\psi_{cof} = |Gc_{(j\omega_{max})}| \tag{3.39}$$

Given the updated controller attributes, the lead branch of the controller can be computed using equations 3.36 and 3.37 .

Let the transfer function of the integrator branch be:

$$Gci_{(s)} = \frac{Ki}{s} \tag{3.40}$$

We can compute K_i, given the controller attribute: low frequency at cross-over gain, ωl_{cog}, and the controller gain at cross-over frequency $|C_{(j\omega_{cog})}|$

$$K_i = |C_{(j\omega_{cog})}| \cdot \omega l_{cog} \tag{3.41}$$

This completes the mapping of the attributes of the controller to its parameters.

3.5 Fuzzy Tuner Level

In the top level of the heirarchical structure an expert tuning knowledge is mathematically formulated using fuzzy set theory and fuzzy logic operations and a decision table is calculated off-line to match the performance indices with tuning actions in the servo-expert level. First, an expert knowledge on the tuning procedure is expressed as a set of linguistic fuzzy statements. Next, membership functions are assigned for each performance and tuning variable and, by applying the compositional rule of inference, fuzzy composite relation tables are established to express the relations between each performance and tuning variable. Finally, this fuzzy composite relation is defuzzified to obtain a **crisp** value for the tuning actions to be used in the servo expert level.

To save real time computation this program is computed off-line.

There are five subprograms in this level:

- SUB-RULSET - Generates array of condensed form for the linguistic rules which relate the performance variables and quantities to tuning variables and quantities.
- SUB-MEMBERSHIP - Generates the membership function array for the performance variables, to indicate the possibility of each performance index to belong to performance fuzzy subset, and membership function array for the tuning variables to indicate the possibility of each tuning action to belong to the tuning fuzzy subset.
- SUB-RELATION - Establish fuzzy relation array between fuzzy performance and tuning variables, according to the ruleset array, by applying the cartesian product space of the two-variable membership functions.
- SUB-DECISION - Generates a decision array by applying the compositional rules of inference.

- SUB-DEFUZIFIED - Compute the crisp value of the tuning action using the centre of gravity method.

Block diagram of the fuzzy tuner is shown in Figure 3.8

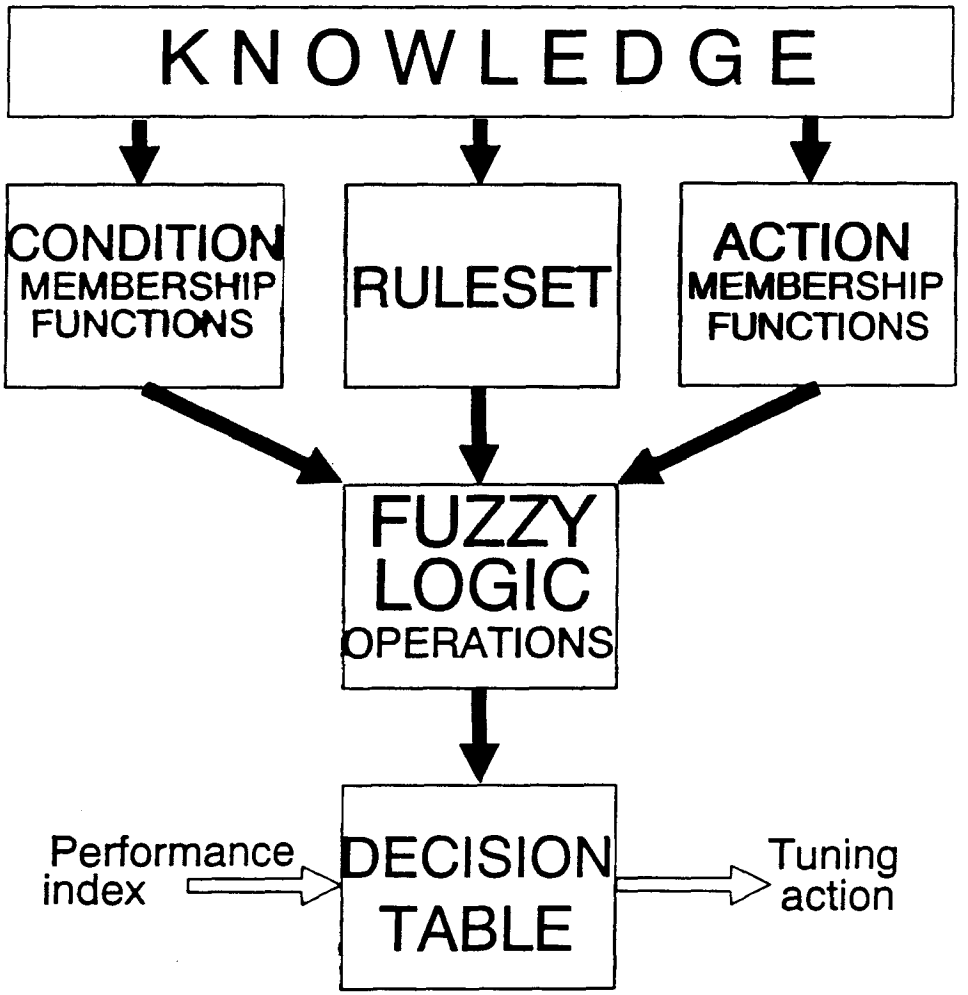


Figure 3.8: Fuzzy Tuner Block Diagram

3.5.1 Fuzzy Ruleset

Experts in system tuning usually learn and gain tuning knowledge by performing tuning actions and observing the response of the system to these actions. Often the tuning actions are expressed in linguistic fuzzy terms like "Turn the knob a little bit to the left" or "Raise the lever slightly".

In a similar way, the system performance is expressed in linguistic fuzzy terms as "The speed is too slow" or "High offset is existing in the system".

Suppose that actions like "Turn the knob" and "Raise the lever" are *fuzzy tuning variables* of the system, and "A little bit to the left" or "Slightly" are their *fuzzy tuning quantities.* Furthermore, let performance variables like "Speed" and "Offset" be the *fuzzy performance variables* and "slow" and "High" be their *fuzzy performance quantities.*

To formulate this knowledge mathematically we define an I-dimensional universe of fuzzy performance variables denoted by P, in which the i'th dimension represents performance parameter variable $\widetilde{PV}_{(i)}$. Let the elements of this universe be the actual performance indices, as calculated in the servo-expert level. Divide each performance parameter $\widetilde{PV}_{(i)}$ to j fuzzy subsets in which the j'th subset represents fuzzy performance quantity $\widetilde{PQ}_{(j)}$.

In the same manner we define L dimensional universe of tuning actions T, in which the l'th dimension represents a tuning variable $\widetilde{TV}_{(l)}$, each of which is divided into m fuzzy subsets $\widetilde{TQ}_{(m)}$, representing m fuzzy tuning quantities.

Now define the performance variables and quntities as well as the tuning variables and quantities in the **fuzzy form** of crisp variables and quantites, as has been defined for the servo-expert level (see equation 3.27 on page 3.27.

Fuzzy performance variables

$$
\begin{aligned}
\widetilde{PV}_{(1)} &= \text{RISTM} \quad \text{for the fuzzy form of Rise Time error } (\mathcal{RT}_e) \\
\widetilde{PV}_{(2)} &= \text{DMPFR} \quad \text{for the fuzzy form of Damped Natural Frequency error } (\mathcal{DF}_e) \\
\widetilde{PV}_{(3)} &= \text{DMPRT} \quad \text{for the fuzzy form of Damping Ratio error } (\mathcal{DR}_e) \\
\widetilde{PV}_{(4)} &= \text{OVSHT} \quad \text{for the fuzzy form of Overshoot error } (\mathcal{OV}_e) \\
\widetilde{PV}_{(5)} &= \text{OFFST} \quad \text{for the fuzzy form of Offset } (\mathcal{OF}_e)
\end{aligned}
\tag{3.42}
$$

Each condition variable is assigned one of the following fuzzy quantities and stored in the vector $\widetilde{PQ}_{(j)}$:

$$
\begin{aligned}
\widetilde{PQ}_{(1)} &= \text{OVRSP} \quad \text{(Over specification)} \\
\widetilde{PQ}_{(2)} &= \text{IN-SP} \quad \text{(In-specification)} \\
\widetilde{PQ}_{(3)} &= \text{MODRT} \quad \text{(Moderate)} \\
\widetilde{PQ}_{(4)} &= \text{POOR} \quad \text{(Poor)} \\
\widetilde{PQ}_{(5)} &= \text{UNSTF} \quad \text{(Unsatisfactory)}
\end{aligned}
\tag{3.43}
$$

For analytical convenience assign a numerical integer to each quantity:

$$
\begin{aligned}
\widetilde{PQ}_{(1)} &= 1 \\
\widetilde{PQ}_{(2)} &= 2 \\
\widetilde{PQ}_{(3)} &= 3 \\
\widetilde{PQ}_{(4)} &= 4 \\
\widetilde{PQ}_{(5)} &= 5
\end{aligned}
\tag{3.44}
$$

Similarly define vector $\widetilde{TV}_{(l)}$ of the fuzzy tuning variables as the fuzzy form of the incremental changes of the controller attributes (see equation 3.39:

$$\begin{aligned}
\widetilde{TV}_{(1)} &= \text{PHCOF} \quad \text{the fuzzy form of } \phi_{cof} \\
\widetilde{TV}_{(2)} &= \text{FRCOG} \quad \text{the fuzzy form of } \omega_{cog} \\
\widetilde{TV}_{(3)} &= \text{GNCOG} \quad \text{the fuzzy form of } \Psi_{cof} \\
\widetilde{TV}_{(4)} &= \text{LFCOG} \quad \text{the fuzzy form of } \Omega l_{cog}
\end{aligned} \tag{3.45}$$

Each tuning variable is assigned one of the following fuzzy quantities which are stored in the vector $\widetilde{TQ}_{(m)}$:

$$\begin{aligned}
\widetilde{TQ}_{(1)} &= \text{NEGHI} \quad \text{for high, negative increment} \\
\widetilde{TQ}_{(2)} &= \text{NEGLO} \quad \text{for low, negative increment} \\
\widetilde{TQ}_{(3)} &= \text{NOCNG} \quad \text{for no change} \\
\widetilde{TQ}_{(4)} &= \text{POSLO} \quad \text{for low, positive increment} \\
\widetilde{TQ}_{(5)} &= \text{POSHI} \quad \text{for high positive increment}
\end{aligned} \tag{3.46}$$

For analytical convenience each subset is assigned an integer value:

$$\begin{aligned}
\widetilde{TQ}_{(1)} &= -2 \\
\widetilde{TQ}_{(2)} &= -1 \\
\widetilde{TQ}_{(3)} &= 0 \\
\widetilde{TQ}_{(4)} &= +1 \\
\widetilde{TQ}_{(5)} &= +2
\end{aligned} \tag{3.47}$$

Knowledge is gained by observing the effect of each tuning action on the performance and expressing these effects as a set of rules of the form:

If $\widetilde{PV}_{(i)}$ **is** $\widetilde{PQ}_{(j)}$ **then** $\widetilde{TV}_{(l)}$ **is** $\widetilde{TQ}_{(m)}$

Using these definitions the condensed form of the rules are stored in a three dimensional array as:

$$\widetilde{RU}_{(i,j,l)} = m \tag{3.48}$$

For example: $\widetilde{RU}_{(4,3,2)} = 1$ is the condensed form of the rule:

If OVSHT **is** MODRT **then** FRCOG **is** NEGHI

The default value is $\widetilde{RU}_{(i,j,l)} = 3$ (Tuning quantity is unchanged for all combinations of performance variable, performance quantity and tuning variable). Modification of the table is done by typing the three indices i, j, l and the value of the desired tuning quantity (m index). The effect of each tuning variable on each performance variable, and the rule generation based on these effects are described in the next chapter.

3.5.2 Membership Functions for Performance and Tuning Variables

Subroutine SUB-PRFMSF

Subprogram SUB-PRFMSF generates the three dimensional array $\widetilde{PM}_{(i,j,k)}$ membership function for the performance quantity in which i stands for the performance fuzzy variable, j stands for the performance fuzzy quantity and k is the performance index. For example, $\widetilde{PM}_{(1,2,3)} = 1$ means that $\widetilde{PQ}_{(1)}$ (Rise time) with performance index equal 3 is **definitely in** the fuzzy quantity subset IN-SPC (see definitions in equations 3.42 , 3.44 , 3.45 , and 3.47 .

Currently subprogram SUB-PRFMSF assigns 1 to the representative value and uniformly decreasing membership grade to the other performance indices. The membership functions of the performance variables are shown schematically in Figure 3.9

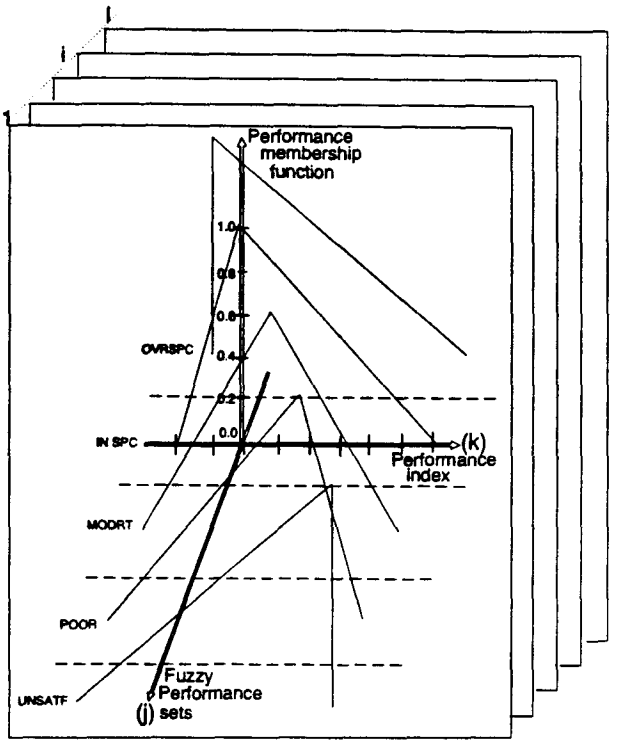


Figure 3.9: Membership Functions for the Performance Variables

Modification of the table is done by assigning values to the i, j, k indices and a membership grade to the desired element.

Subprogram SUB-PRFMSF is listed in appendix B-3-3.

Subroutine SUB-TUNMSF

Subprogram SUB-TUNMSF generates three dimensional array $\widetilde{TM}_{(l,m,n)}$ of membership functions for the tuning quantity in which l stands for the fuzzy tuning variable, m stands for the tuning fuzzy quantity and n is the tuning action index. For example, $\widetilde{TM}_{(1,2,3)} = 0$ means that PHCOF (fuzzy phase lead angle at cross-over frequency) with quantity index 3 is **definitely not** in the fuzzy quantity subset NOCNG (no change) (See equations 3.45 ,3.46 and 3.47).

Currently subprogram SUB-TUNMSF assigns 1 to the representative value and uniformly decreasing membership grade to the other performance indices. The membership

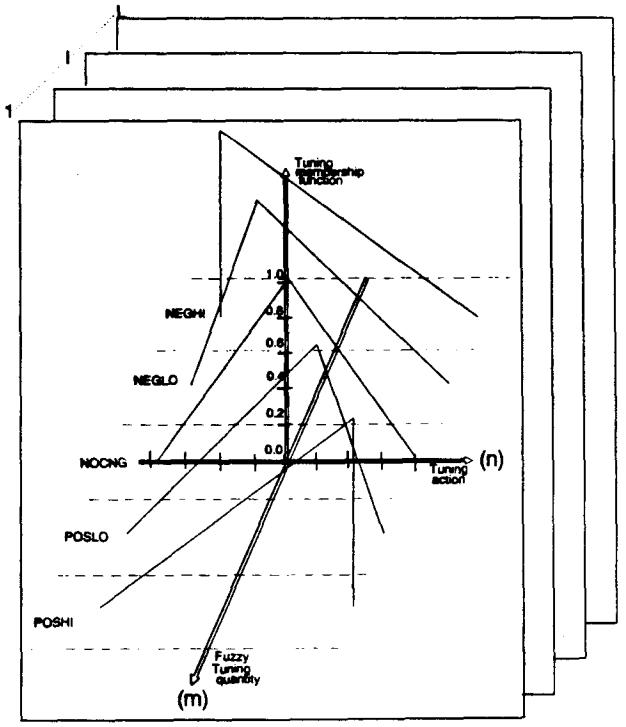


Figure 3.10: Membership Functions for the Tuning Variables

functions of the tuning variables are shown schematically in Figure 3.10

Modification of the table is done by assigning values to the l, m, n indices and a membership grade to the desired element.

Subprogram SUB-TUNMSF is listed in appendix B-3-4.

3.5.3 Fuzzy Relation between Condition and Action Variables

Subprogram SUB-RELATION computes a four dimensioned array $\widetilde{RL}_{(i,l,k,n)}$. This matrix is a condensed form of the fuzzy relation between each fuzzy performance variable ($\widetilde{PV}_{(i)}$) and each fuzzy tuning variable ($\widetilde{TV}_{(l)}$). The i, l, k and n indices are as defined previously.

First, using the rules in $\widetilde{RU}_{(i,j,l)}$ (equation 3.48 on page 73) an *INTERSECTION operation* (see equation 2.5 on page 23) is applied by taking the minimum of the membership function grade $\widetilde{PM}_{(i,j,k)}$ of each element k in the performance quantity subspace

$\widetilde{PQ}_{(j)}$ belonging to the fuzzy performance variable $\widetilde{PV}_{(i)}$ and the membership function $\widetilde{TM}_{(l,m,n)}$ of each element n in the fuzzy tuning subspace $\widetilde{TQ}_{(m)}$, that belongs to the fuzzy tuning variable $\widetilde{TV}_{(l)}$, for every i, j and l in $\widetilde{RU}_{(i,j,l)}$.

This results in a temporary five dimensional matrix $BARLEV_{(i,l,j,k,n)}$, which stands for the fuzzy relation:

If $\widetilde{PV}_{(i)} = \widetilde{PQ}_{(j)}$ **then** $\widetilde{TV}_{(l)} = \widetilde{TQ}_{(m)}$

For example a typical fuzzy relation table relating the fuzzy performance variable RISTM and the fuzzy tuning variable $PHCOF$ is demonstrated in Table 3.1

Table 3.1: Development of a Fuzzy Relation Table (for $RISTM \longrightarrow PHCOF$)

IF RISTM = UNSATF Then PHCOF = NEGHI

	-2	-1	0	1	2
1	1.0	.8	.6	.4	.2
2	.8	.8	.6	.4	.2
3	.6	.6	.6	.4	.2
4	.4	.4	.4	.4	.2
5	.2	.2	.2	.2	.2

IF RISTM = POOR Then PHCOF = NEGLO

	-2	-1	0	1	2
1	.8	.8	.8	.6	.4
2	.8	1.0	.8	.6	.4
3	.8	.8	.8	.6	.4
4	.6	.6	.6	.6	.4
5	.4	.4	.4	.4	.4

IF RISTM = MODRAT Then PHCOF = NEGLO

	-2	-1	0	1	2
1	.6	.6	.6	.6	.4
2	.8	.8	.8	.6	.4
3	.8	1.0	.8	.6	.4
4	.8	.8	.8	.6	.4
5	.6	.6	.6	.6	.4

IF RISTM = IN-SPC Then PHCOF = NOCHG

	-2	-1	0	1	2
1	.4	.4	.4	.4	.4
2	.6	.6	.6	.6	.6
3	.6	.8	.8	.8	.6
4	.6	.8	1.0	.8	.6
5	.6	.8	.8	.8	.6

IF RISTM = OVRSPC Then PHCOF = POSLO

	-2	-1	0	1	2
1	.2	.2	.2	.2	.2
2	.4	.4	.4	.4	.4
3	.4	.6	.6	.6	.6
4	.4	.6	.8	.8	.8
5	.4	.6	.8	1.0	.8

Table 3.2: Fuzzy Composite Relation Table for RISTM and PHCOV

	-2	-1	0	1	2
1	1.0	.8	.8	.6	.4
2	.8	1.0	.8	.6	.6
3	.8	1.0	.8	.8	.6
4	.8	.8	1.0	.8	.8
5	.6	.8	.8	1.0	.8

Next, *UNION operation* (see equation 3.4 on page 47) is applied by taking the maximum membership grade of $BARLEV_{(i,l,j,k,n)}$ over index j for every i, l, m, k and n. This operation results in a four dimensional Matrix $\widetilde{RL}_{(i,l,k,n)}$ which is the *Composite Fuzzy Relation* between the fuzzy performance variable $\widetilde{PV}_{(i)}$ and the fuzzy tuning variable $\widetilde{TV}_{(l)}$.

For example a typical composite fuzzy relation table relating the fuzzy performance variable $RISTM$ and the fuzzy tuning variable PHCOF is demonstrated in Table 3.2

Subprogram SUB-RELATION is listed in appendix B-3-5.

3.5.4 Decision Table

The subroutine SUB-DECISION generates a three dimensional matrix $\widetilde{DT}_{(i,l,j)}$ by matching the composite relation matrix $\widetilde{RL}_{(i,l,k,n)}$ with the membership function matrix $\widetilde{PM}_{(i,j,k)}$ of the performance quantity matrix $\widetilde{PQ}_{(i,j,k)}$ that belongs to the performance variable subset $\widetilde{PV}_{(i,j,k)}$.

First, INTERSECTION operation (see equation 2.5 on page 23) is applied to $\widetilde{PM}_{(i,j,k)}$ and $\widetilde{RL}_{(i,l,k,n)}$, to form the temporary matrix $SHANY_{(i,l,j,k,n)}$.

Next, UNION operation (see equation 3.4 on page 47) on $SHANY$ is applied to form

the temporary matrix $YAEL_{(i,l,j,n)}$, the membership function matrix of the action l due to the performance variable i with fuzzy quantity j.

Finally, subprogram SUB-DEFUZZY is called to compute the crisp value of the tuning quantity, using the Centre of Gravity Method, In order to obtain a crisp value for the tuning action. Specifically, we weight the elements in the universe of the tuning action using the membership grades of the action, and then take the average. This value is the entry for the tuning action in the decision table, and stored now in the decision table array $\widetilde{DT}_{(i,j,l)}$, to be used on-line in the servo expert level.

For example, in:

$$\widetilde{DT}_{(1,2,3)} = 0.6$$

the Rise Rime ($\mathcal{RT}_e$) having a performance index equal to 2 is the entry point to the decision table and it triggers the gain increment at cross-over frequency $\odot_{(3)}$ to be 0.6 as can be seen in Table 3.3

A typical **crisp** decision table is shown in Table 3.3

Table 3.3: Decision Table for the Servo-Motor

Table for Performance Parameter: $\mathcal{RT}_e$

	ϕ_{cof}	ω_{cog}	ψ_{cof}	ωl_{cog}
1	-1.0	1.0	1.0	.0
2	-.6	.6	.6	.0
3	-.2	.2	.2	.0
4	.0	.0	.0	.0
5	.2	-.2	-.2	.0

Table for Performance Parameter: $\mathcal{DF}_e$

	ϕ_{cof}	ω_{cog}	ψ_{cof}	ωl_{cog}
1	.0	.0	1.0	.0
2	.0	.0	.6	.0
3	.0	.0	.2	.0
4	.0	.0	.0	.0
5	.0	.0	-.2	.0

Table for Performance Parameter: $\mathcal{DR}_e$

	ϕ_{cof}	ω_{cog}	ψ_{cof}	ωl_{cog}
1	-1.0	1.0	1.0	.0
2	-.6	.6	.6	.0
3	-.2	.2	.2	.0
4	.0	.0	.0	.0
5	.2	-.2	-.2	.0

Table for Performance Parameter: $\mathcal{OS}_e$

	ϕ_{cof}	ω_{cog}	ψ_{cof}	ωl_{cog}
1	1.0	-1.0	1.0	.0
2	.6	-.6	.6	.0
3	.2	-.2	.2	.0
4	.0	.0	.0	.0
5	-.2	.2	-.2	.0

Table for Performance Parameter: $\mathcal{OF}_e$

	ϕ_{cof}	ω_{cog}	ψ_{cof}	ωl_{cog}
1	.0	.0	1.0	1.0
2	.0	.0	1.0	.6
3	.0	.0	.6	.2
4	.0	.0	.6	.0
5	.0	.0	.4	-.2

Chapter 4

EXPERIMENTAL STRATEGY AND PROCEDURE

4.1 Introduction

This chapter describes the experimental testing strategy and procedure both in the simulated and the physical servo-motor system.

The experiments were designed for three purposes. Firstly, it is desired to study the performance of the tuning mechanism with regard to its parameters. Secondly, it is desirable to evaluate the performance in comparison with those of other conventional techniques. Finally, and most importantly, it is important to determine if a fuzzy tuner algorithm is a practical means of automating the tuning the actions taken by an expert operator, by implementing it on a commercially available servo-motor system.

The results of these tests are shown graphically in the next chapter.

4.2 Experimental Strategy

Given a commercially available servo-motor system the starting point of the experimental work is the designing of a controller for a simulated, well-defined and simplified linear servo-motor system at the lowest level, using well known "classical" design tools to meet a set of frequency and time domain specifications. This step results in a well-tuned base-line system.

Next, the algorithms of the servo-expert and the fuzzy tuner in the higher levels are integrated with the simulated servo-motor system, and the knowledge-based ruleset is

established by observing the effect of each tuning action on each performance parameter, thereby simulating the knowledge of an expert operator. Then the system is tested when the simulated base-line servo-motor is gradually spoilt toward an ill-defined servo-motor system, **leaving the two upper levels unchanged**. Finally, the tuning algorithm is implemented and tested on the commercially available servo-motor system.

4.3 Performance Requirements

The required time domain performance parameters are derived from the actual application of the servo-motor system, considering several limitation factors such as as minimum available sampling time of the controller, maximum execution speed, or maximum available acceleration of the servo-motor system.

Often a compromise among these requirements is needed. Such an /settle/ example, between the speed (typically represented by the rise time) requirement and the oscillation (typically represented by the overshoot) requirement.

4.3.1 Model Parameters

Considering the given system limitations (of the motor, amplifier, controller and computer) the dynamic performance parameters are chosen (see definitions in Section 3.3.1):

1. Rise time $(\mathcal{RT}_m) = 33[msec]$
2. Average damping ratio $(\overline{\mathcal{DR}}_m) = 0.095$
3. Average damped natural frequency $(\overline{\mathcal{DF}}_m) = 75[hz]$
4. Overshoot $(\mathcal{OV}_m) = 10[\%]$

and the static performance parameters:

5. Offset ($\mathcal{OF}_m$) = 2[%]

A reference model that satisfies these requirements (see equation 3.8 on page 53) is:

- $\zeta_m = 0.55$ (Model damping ratio)
- $\omega_m = 75$ (Model undamped natural frequency)
- $\theta = 0.02$ (Model offset)

4.3.2 Acceptable tolerance

The following acceptable tolerances are used for the dynamic performance parameters.

1. Rise time tolerance = 0.1
2. Overshoot tolerance = 0.1
3. Average Damping Ratio tolerance = 0.1
4. Average Damped natural response = 0.1

   and for the static performance parameters.:

5. Offset tolerance = 0[%]

4.3.3 Test Signal

The magnitude of the square wave test signal should be far above the noise and the quantization levels of the servo-motor system. The frequency of the square wave should by small enough to let the system response settle down. Amplitude of 200[counts] and frequency 2[hz] is used.

4.4 Base-Line Servo-Motor System

Denote Base-Line servo-motor system as a simulated, *well-defined, well-tuned* system. Well-defined system reflects accurate and complete knowledge of the mathematical model of the system. Well-tuned system reflects a controller design procedure based on well-defined system, and results in "in specifications" performance.

The values of specific motor parameters (see equation 3.1 on page 46 are:

$K = 0.076[N \cdot [M \cdot m/A]$ - torque constant

$R = 1.59[ohm]$ - resistance of the field winding

$L = 2.5[mH]$ - inductance of the field winding

$J = 2.6 * 10^{-5}[Kgm^2]$ - moment of inerta of the rotor

$B = 0.0037[Nm \cdot sec]$ - motor mechanical dampling

and for current supplay amplifier (see equation 3.2 on page 46:

$A = 0.2[amp/volt]$ - effective gain of the current source amplifier

$V_{max} = 10[volt]$ - maximum voltage input to the amplifier

$I_{max} = 2[amp]$ - maximum current output of the amplifier

The well-tuned controller attributes (see equation 3.39 and equation 3.40 on page 66) of the base-line system are designed to be:

1. Phase lead Angle at cross-over frequency: $\phi_{(cof)} = 1[rad]$
2. Cross-over gain frequency: $\omega_{(cog)} = 200[rad/sec]$
3. Cross-over gain: $\psi_{(cof)} = 12[db]$
4. Low frequency gain :$\omega_{l(cog)} = 0.$

Time domain response of the reference model as well as that of the base-line servo-motor system to a squared wave input is shown in Figure 4.1 . Notice that all the actual resopnse parameters are in or over specification.

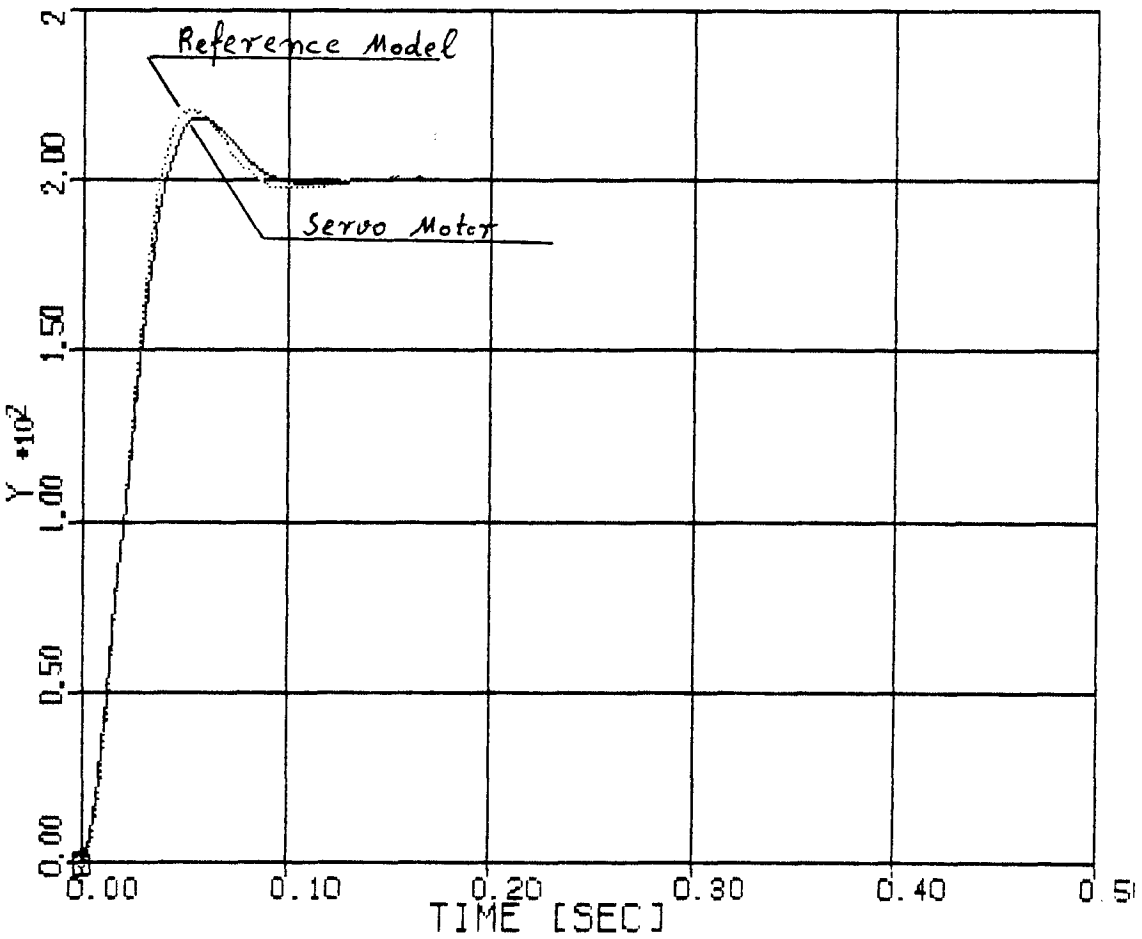


Figure 4.1: Time Domain Response of the Base-Line Servo-Motor System

4.5 Generation and Evaluation of Tuning Rules

To simulate an expert knowledge in tuning a servo-motor system, the effect of each controller attribute on each performance parameter is examined by applying the servo-expert level algorithm on the simulated servo-motor system. These effects are demonstrated in the following figures:

The effect of the cross-over phase lead angle $\phi_{(cof)}$ is demonstrated in Figure 4.2.

The effect of the cross-over gain frequency $\omega_{(cog)}$ is demonstrated in Figure 4.3.

The effect of the cross-over gain $\psi_{(cof)}$ is shown in Figure 4.4.

The effect of the cross-over gain $(\psi_{(cof)})$ when an external load is applied is shown in Figure 4.5.

The effect of the low-frequency gain $(\omega_{l_{(cog)}})$ when an external load is applied is demonstrated in Figure 4.6.

Observations of these effects are now expressed as linguistic fuzzy rules to form the ruleset of the high level fuzzy tuner. Notice that since the knowledge on tuning actions are gained by **looking** at these graphs rather than actually measuring (crisp) performance parameters they become fuzzy variables. Furthermore, we can "pretend" that the tuning actions that have been taken are not known accurately, to simulate tuning actions taken by an experienced operator, and therefore they become fuzzy variables as well.

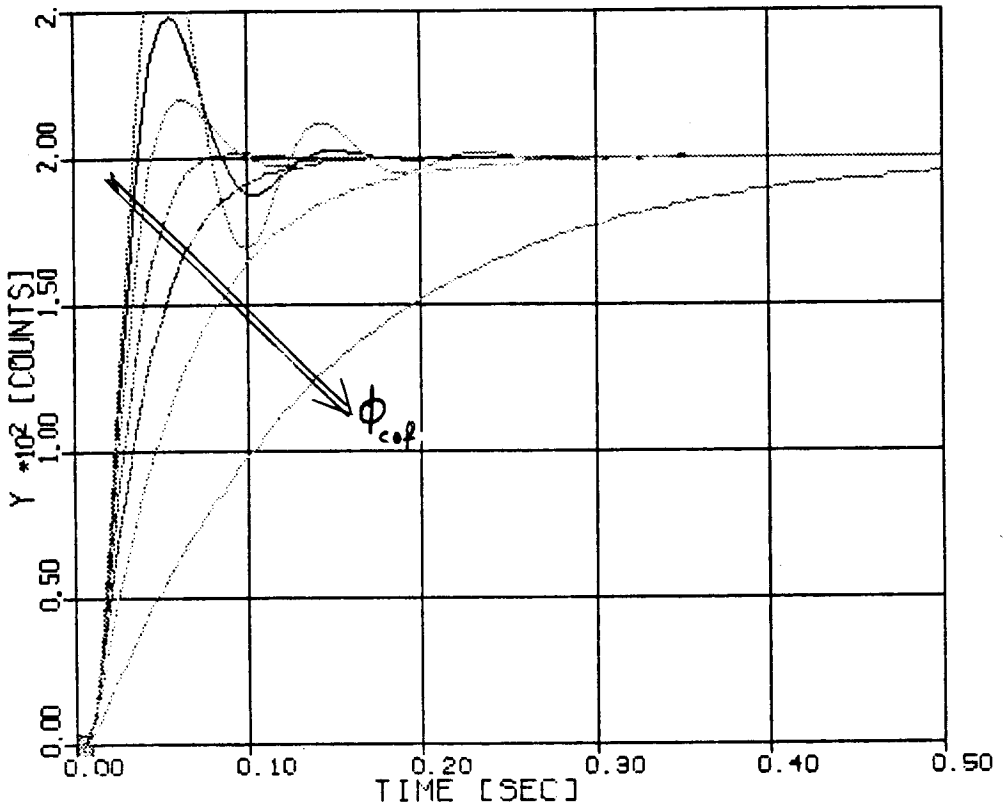


Figure 4.2: Effect of Phase Lead Angle at Cross-over Gain Frequency

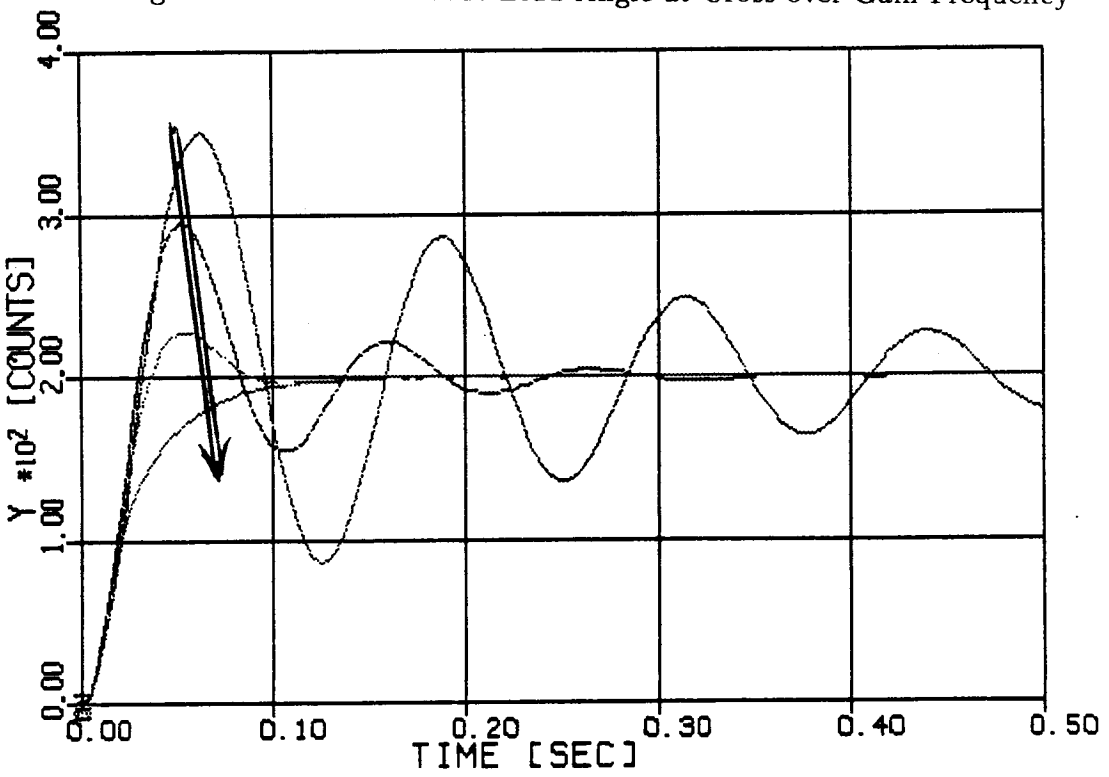


Figure 4.3: Effect of Cross-over Gain Frequency

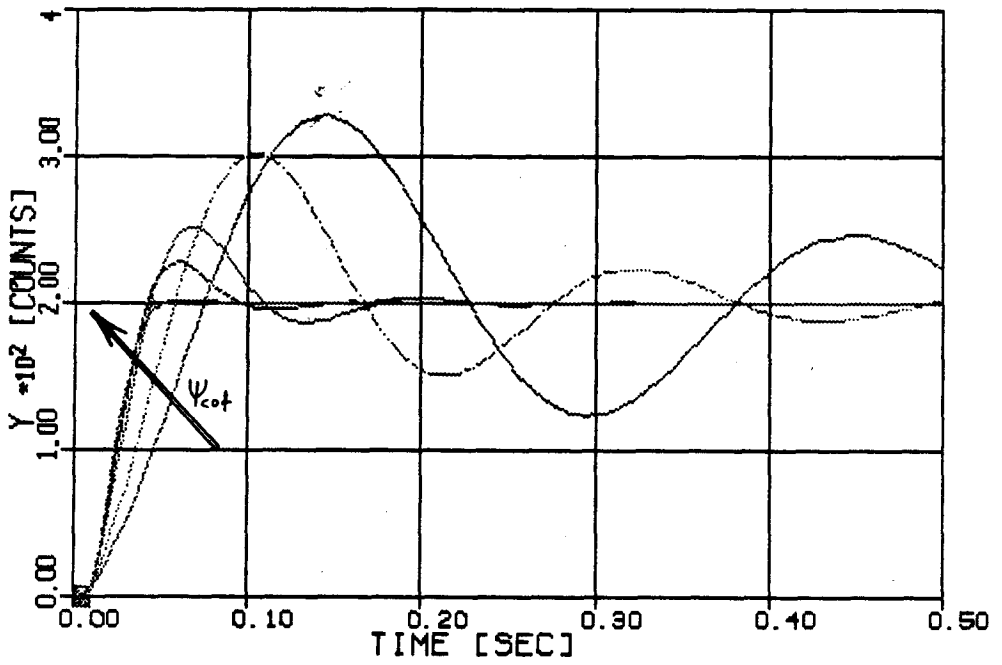


Figure 4.4: Effect of Cross-over Gain

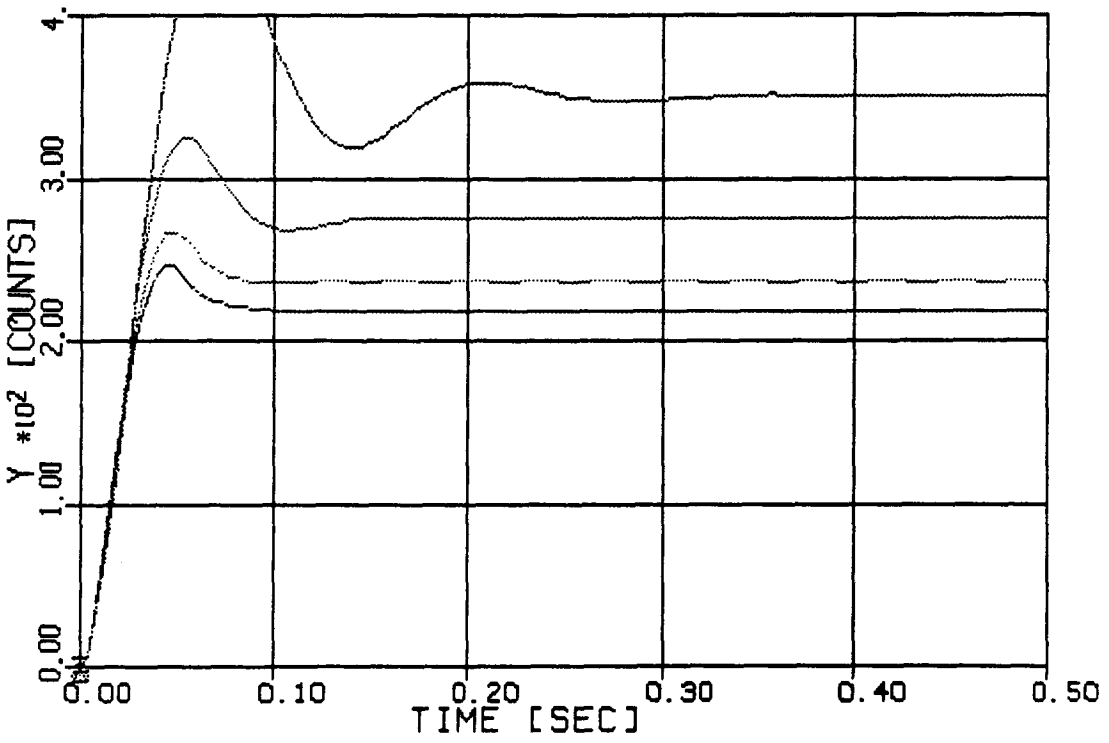


Figure 4.5: Effect of Cross-over Gain when External Load is Applied

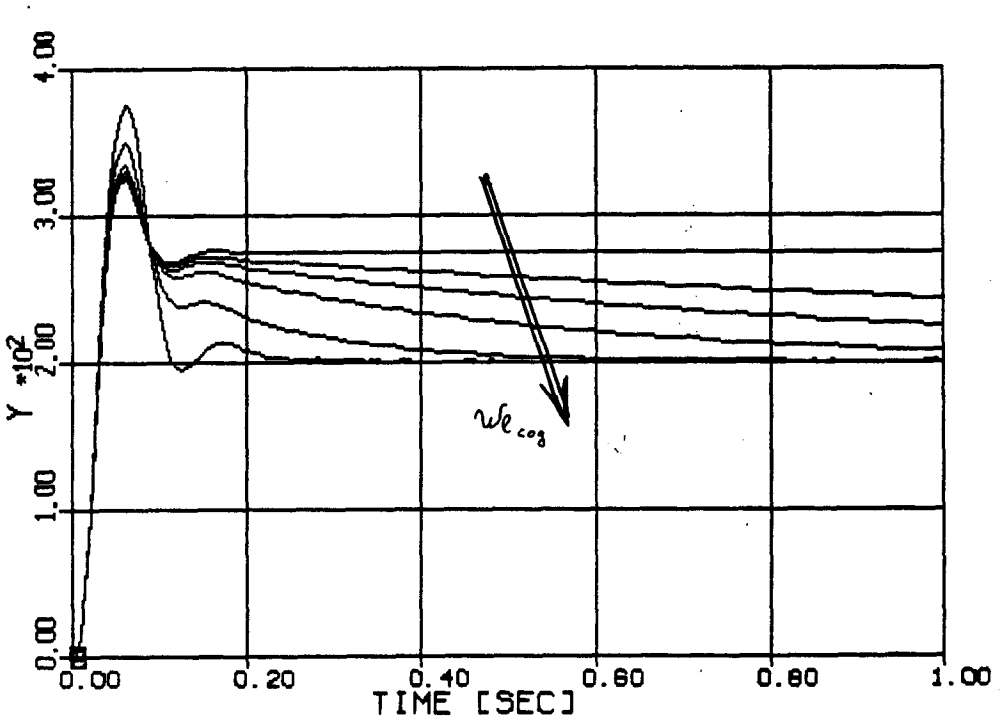


Figure 4.6: Effect of Low-frequency at Cross-over Gain when External Load is Applied

Rules for Tuning the Phase at Cross-over Frequency

Based on Figure 4.2 the following rules are established for tuning PHCOF.

RISTM dependence on PHCOF

```
      If RISTM is UNSTF    then  PHCOF be NEGHI
else  if RISTM is POOR     then  PHCOF be NEGLO
else  if RISTM is MODRT    then  PHCOF be NEGLO
else  If RISTM is INSPC    then  PHCOF be NOCNG
else  If RISTM is OVRPC    then  PHCOF be POSLO
end   if
```

DMPRT dependence on PHCOF

```
      If DMPRT is UNSTF    then  PHCOF be POSHI
else  if DMPRT is POOR     then  PHCOF be POSLO
else  if DMPRT is MODRT    then  PHCOF be POSLO
else  If DMPRT is INSPC    then  PHCOF be NOCNG
else  If DMPRT is OVRPC    then  PHCOF be NEGLO
end   if
```

OVSHT dependence on PHCOF

```
      If OVSHT is UNSTF    then  PHCOF be POSHI
else  if OVSHT is POOR     then  PHCOF be POSLO
else  if OVSHT is MODRT    then  PHCOF be POSLO
else  If OVSHT is INSPC    then  PHCOF be NOCNG
else  If OVSHT is OVRPC    then  PHCOF be NEGLO
end   if
```

Rules for Tuning the Frequency at Cross-over Gain

Based on Figure 4.3 the following rules are established for FRCOG.

DMPRT dependence on FRCOG

```
      If DMPRT is UNSTF   then  FRCOG be NEGHI
else  if DMPRT is POOR    then  FRCOG be NEGLO
else  if DMPRT is MODRT   then  FRCOG be NEGLO
else  If DMPRT is INSPC   then  FRCOG be NOCNG
else  If DMPRT is OVRPC   then  FRCOG be POSLO
end   if
```

DMPRT dependence on FRCOG

```
      If OVSHT is UNSTF   then  FRCOG be NEGHI
else  if OVSHT is POOR    then  FRCOG be NEGLO
else  if OVSHT is MODRT   then  FRCOG be NEGLO
else  If OVSHT is INSPC   then  FRCOG be NOCNG
else  If OVSHT is OVRPC   then  FRCOG be POSLO
end   if
```

DMPFR dependence on FRCOG

```
      If DMPFR is UNSTF   then  FRCOG be NEGHI
else  if DMPFR is POOR    then  FRCOG be NEGLO
else  if DMPFR is MODRT   then  FRCOG be NEGLO
else  If DMPFR is INSPC   then  FRCOG be NOCNG
else  If DMPFR is OVRPC   then  FRCOG be POSLO
end   if
```

Rules for Tuning the Gain at Cross-over Frequency

Based on Figure 4.4 the following rules are established for GNCOF.

RISTM dependence on GNCOF

	If RISTM is UNSTF	then	GNCOF be POSHI
else	if RISTM is POOR	then	GNCOF be POSLO
else	if RISTM is MODRT	then	GNCOF be POSLO
else	If RISTM is INSPC	then	GNCOF be NOCNG
else	If RISTM is OVRPC	then	GNCOF be NEGLO
end	if		

DMPRT dependence on GNCOF

	If DMPRT is UNSTF	then	GNCOF be POSHI
else	if DMPRT is POOR	then	GNCOF be POSLO
else	if DMPRT is MODRT	then	GNCOF be POSLO
else	If DMPRT is INSPC	then	GNCOF be NOCNG
else	If DMPRT is OVRPC	then	GNCOF be NEGLO
end	if		

DMPFR dependence on GNCOF

	If DMPFR is UNSTF	then	GNCOF be POSHI
else	if DMPFR is POOR	then	GNCOF be POSLO
else	if DMPFR is MODRT	then	GNCOF be POSLO
else	If DMPFR is INSPC	then	GNCOF be NOCNG
else	If DMPFR is OVRPC	then	GNCOF be NEGLO
end	if		

Rules for Tuning the Gain at Cross-over Frequency when External Load is Applied

Based on Figure 4.5 the following rules are established for tuning the GNCOF.

OFFST dependence on GNCOF

	If OFFST is UNSTF	then	GNCOF be NOCNG
else	if OFFST is POOR	then	GNCOF be NOCNG
else	if OFFST is MODRT	then	GNCOF be POSLO
else	If OFFST is INSPC	then	GNCOF be NOCNG
else	If OFFST is OVRPC	then	GNCOF be NOCNG
end	if		

Rules for Low-Frequency at Gross-over Gain

Based on Figure 4.6 the following rules are established for LFCOG.

OFFST dependence on GNCOF

	If OFFST is UNSTF	then	LFCOG be POSHI
else	if OFFST is POOR	then	LFCOG be POSLO
else	if OFFST is MODRT	then	LFCOG be POSLO
else	If OFFST is INSPC	then	LFCOG be NOCNG
else	If OFFST is OVRPC	then	LFCOG be NEGLO
end	if		

These tables may be rewritten in condensed form as in table 4.1:

Table 4.1: Condensed Form of the Ruleset

Rules for Condition Variable:OFFST

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	NOCHG	NOCHG	POSHI	POSHI
POOR	NOCHG	NOCHG	POSHI	POSLO
MODRAT	NOCHG	NOCHG	POSLO	POSLO
IN_SPC	NOCHG	NOCHG	POSLO	NOCHG
OVRSPC	NOCHG	NOCHG	NOCHG	NEGLO

Rules for Condition Variable:DMPRT

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	POSHI	NEGHI	POSHI	NOCHG
POOR	POSLO	NEGLO	POSLO	NOCHG
MODRAT	POSLO	NEGLO	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	NEGLO	POSLO	NEGLO	NOCHG

Rules for Condition Variable:RISTM

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	NEGHI	NEGHI	POSHI	NOCHG
POOR	NEGLO	NEGLO	POSLO	NOCHG
MODRAT	NEGLO	NEGLO	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	POSLO	POSLO	NEGLO	NOCHG

Rules for Condition Variable:OVSHT

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	POSHI	NEGHI	POSHI	NOCHG
POOR	POSLO	NEGLO	POSLO	NOCHG
MODRAT	POSLO	NEGLO	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	NEGLO	POSLO	NEGLO	NOCHG

Rules for Condition Variable:DMPFR

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	NOCHG	NOCHG	POSHI	NOCHG
POOR	NOCHG	NOCHG	POSLO	NOCHG
MODRAT	NOCHG	NOCHG	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	NOCHG	NOCHG	NEGLO	NOCHG

4.6 Experimental Procedure

The off-line tuner level program is executed to compute the decision table to be used by the servo-expert level algorithm. To examine the tuner performance degradation when applied on a more realistic system, the simulated base-line servo-motor is now changed, leaving the servo expert level algoritms and the decision table **unchanged**.

Two types of changes are made:

- Initially ill-tuned controller controls a well-defined system.
- Initially "base-line" controller controls an ill-defined system.

In the first type of changes tuning of an initially ill-tuned controller is demonstrated on slow and on oscillatory servo-motor system. In both cases the controller initial attributes were chosen **intentionally** to bring the well-tuned system response to an under specification performance.

In the second type of changes the performance of the tuner is examined when the servo-motor differs from the well defined system which was used to generate the knowledge base, leaving the initial controller attributes as those of the "base-line" servo-motor. In other words the system is tested when the base-line system is spoilt gradually toward an ill-defined system. The tuner was tested when applied on both simulated and commercially available servo-motor systems.

4.6.1 Tuning of the Simulated Servo-Motor System

The following cases were tested in the simulated servo-motor system.

Tuning for ill-tuned controller

- Tuning for initially ill-tuned controller that has a slow response.

- Tuning for initially ill-tuned controller that has a oscillatory response.

Tuning for system parameter changing

- Tuning for increased motor inertia.
- Tuning for external torque - without the integrator branch.
- Tuning for external torque - with the integrator branch.

Tuning for additional dynamices

- Tuning for external spring.
- Tuning an initially slow servo-motor with voltage source amplifier.
- Tuning an initially oscillatory servo-motor with voltage source amplifier.

4.6.2 Tuning of a Commercially Available Servo-Motor System

In a similar way to the experimentation with the simulated servo-motor system, experiments were carried out on a commercially available servo-motor system to demonstrate the implementation of the tuner on a physical system. The commercially available servo-motors are generally ill-defined systems, since accurate mathematical models are not available when the system is developed.

As in the simulated system, the system is tested when the controller is initially ill-tuned, and when the process is changed.

The following cases have been tested with the phyisical servo-motor system.

- Tuning for an initially slow system.
- Tuning for an initially oscillatory system.
- Tuning for increased motor inertia.

The results of these tests are shown graphically in the next chapter.

Chapter 5

SIMULATION AND EXPERIMENTAL RESULTS

In this chapter the experimental results are shown graphically, to demonstrate the performance of the fuzzy tuner when applied to simulated and physical systems. Each experiment is described briefly followed by two pages showing the results:

The responses of the servo-motor as well as the reference model to a square wave are shown in the first page using three frames. The uppermost frame shows the response before tuning, the middle frame shows the response while tuning, and the response after the tuning process is shown in the bottom frame.

Different time scales have been used in the three frames.

The second page shows how the controller attributes are changed while tuning.

5.1 Simulation Experiments

5.1.1 Slow Servo-Motor System

The use of excessive phase lead ϕ_{cof} as well as low cross-over gain frequency at ω_{cog} results in slow, over-damped response.
The response before, while and after tuning are shown in Figure 5.1 and the controller attributes while tuning are shown in Figure 5.2

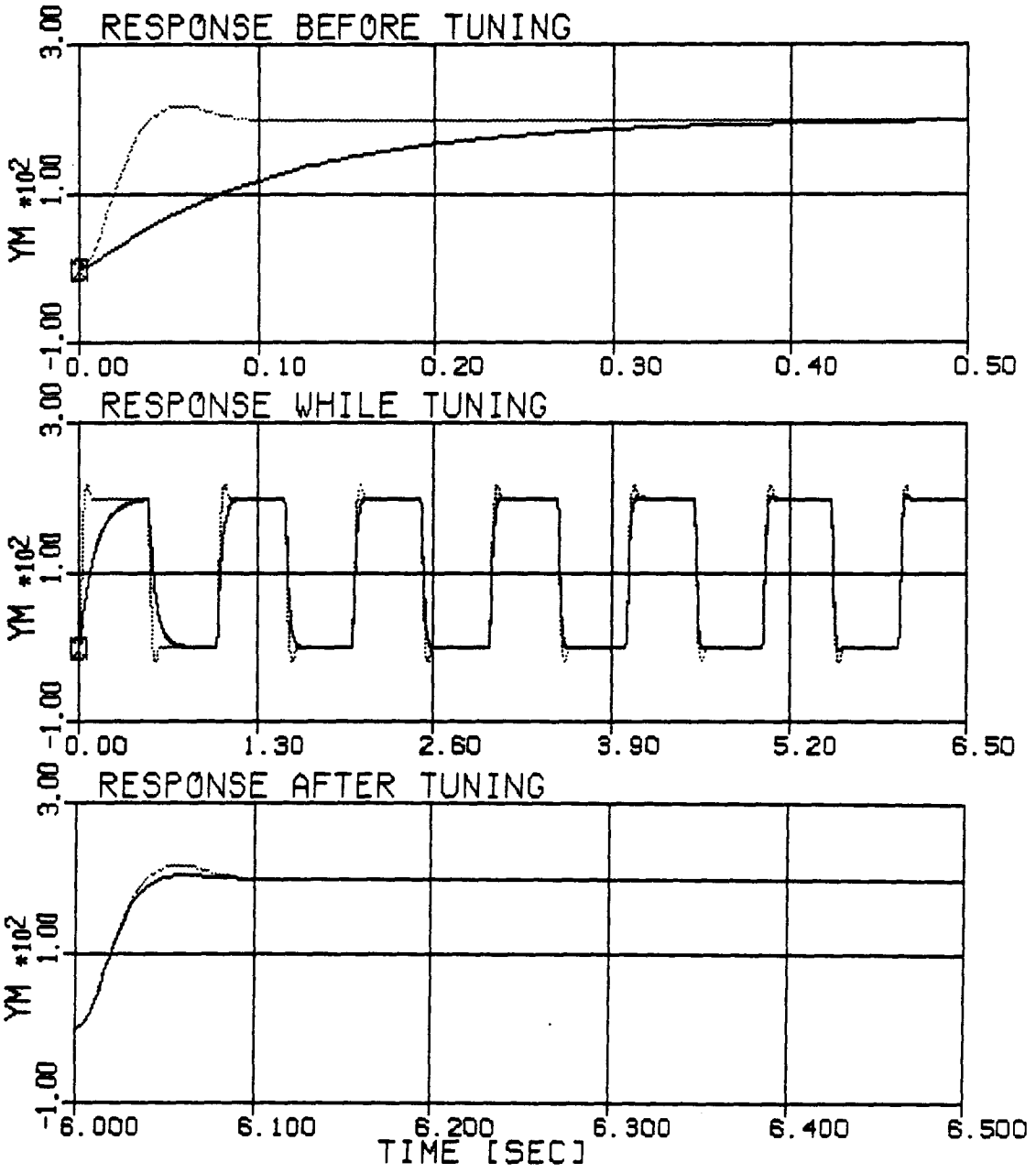


Figure 5.1: Tuning of initially Slow Servo-Motor: Time Response

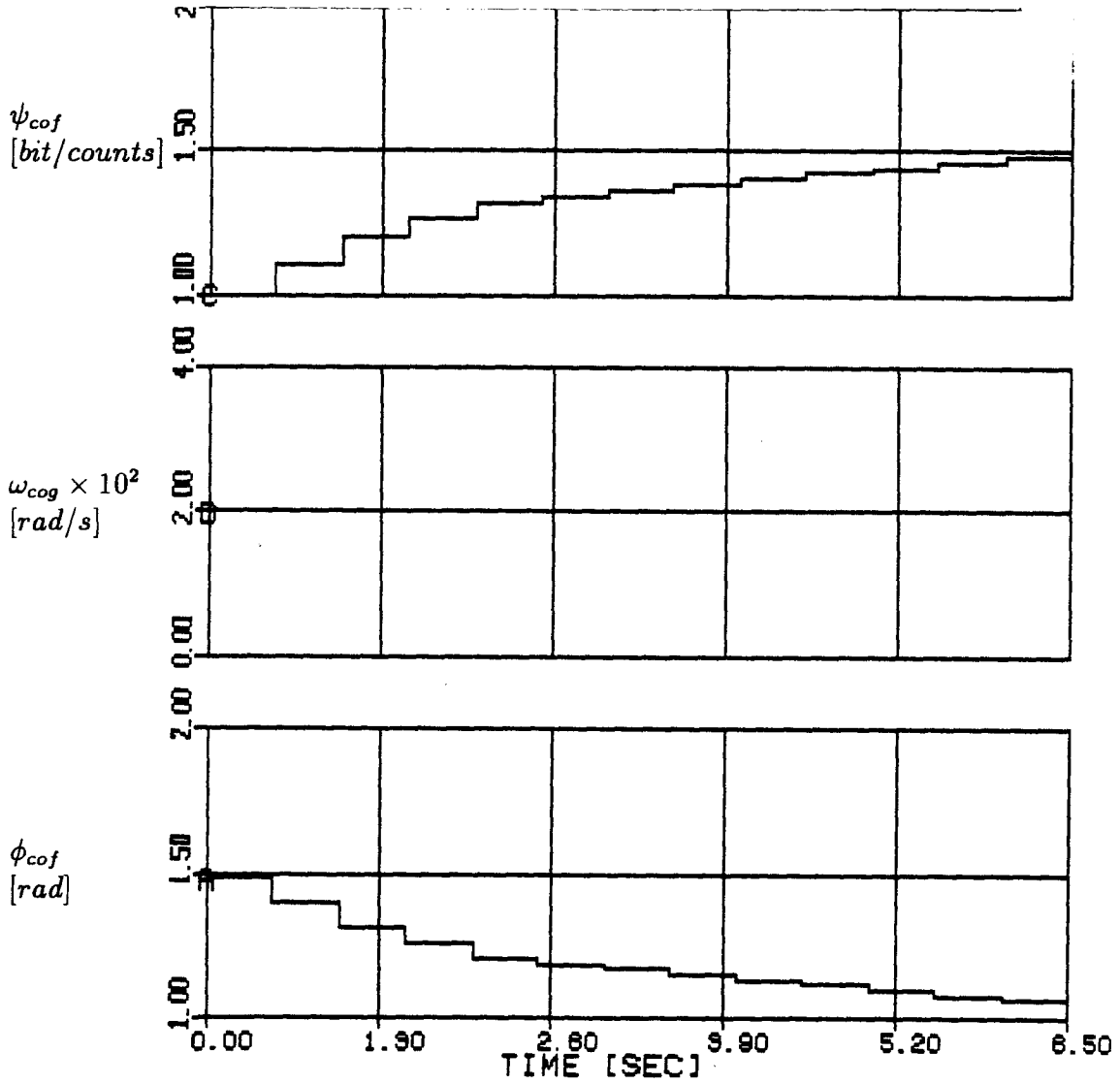


Figure 5.2: Tuning of an Initially Slow Servo-Motor: Controller Attributes

5.1.2 Oscillatory System

Insufficient phase lead ϕ_{cof} as well as too high cross over gain frequency ω_{cog} results in an oscillatory response. The time response is shown in Figure 5.3 and the controller attributes in Figure 5.4

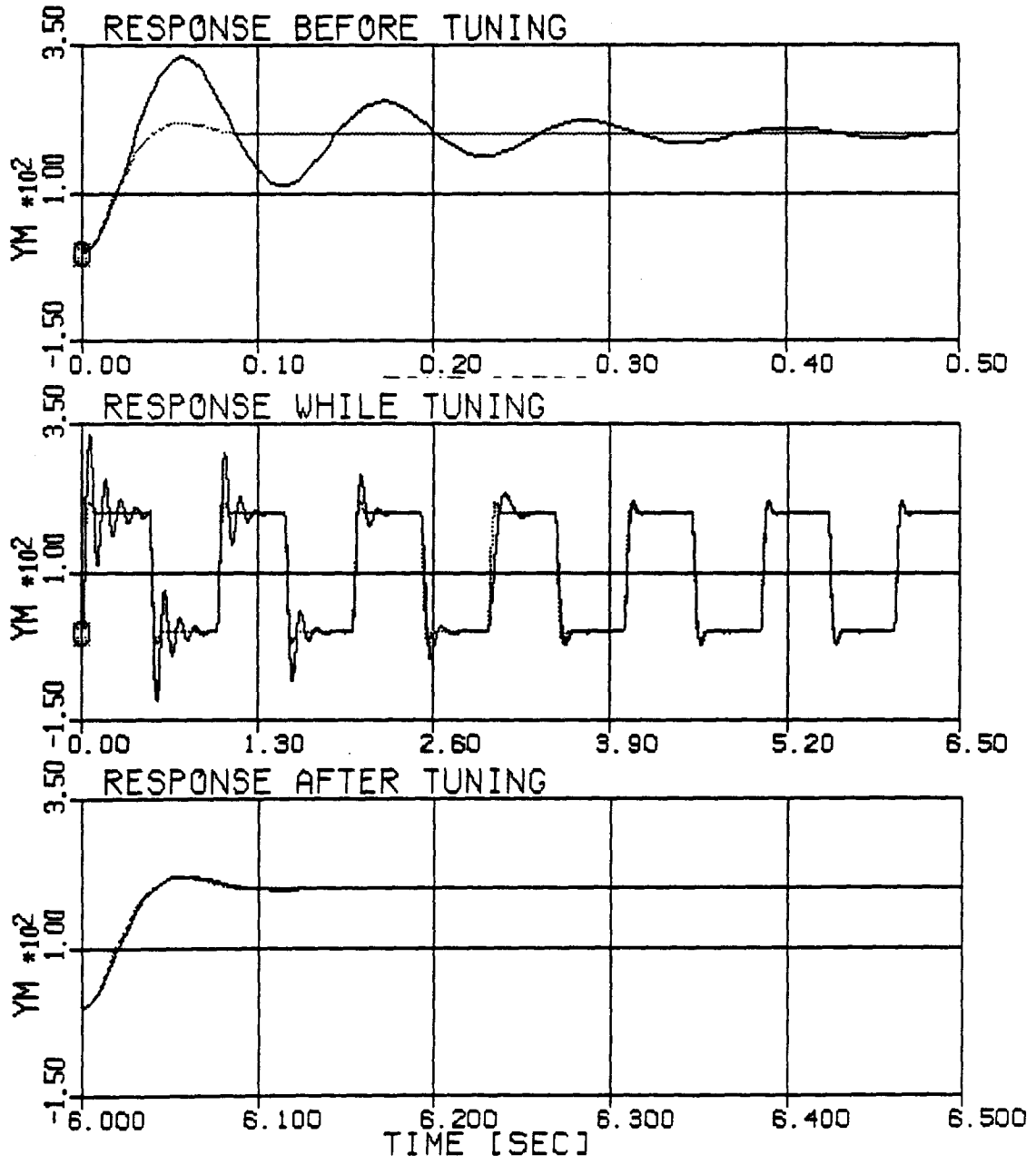


Figure 5.3: Tuning of an Initially Oscillatory Servo-Motor: Time Response

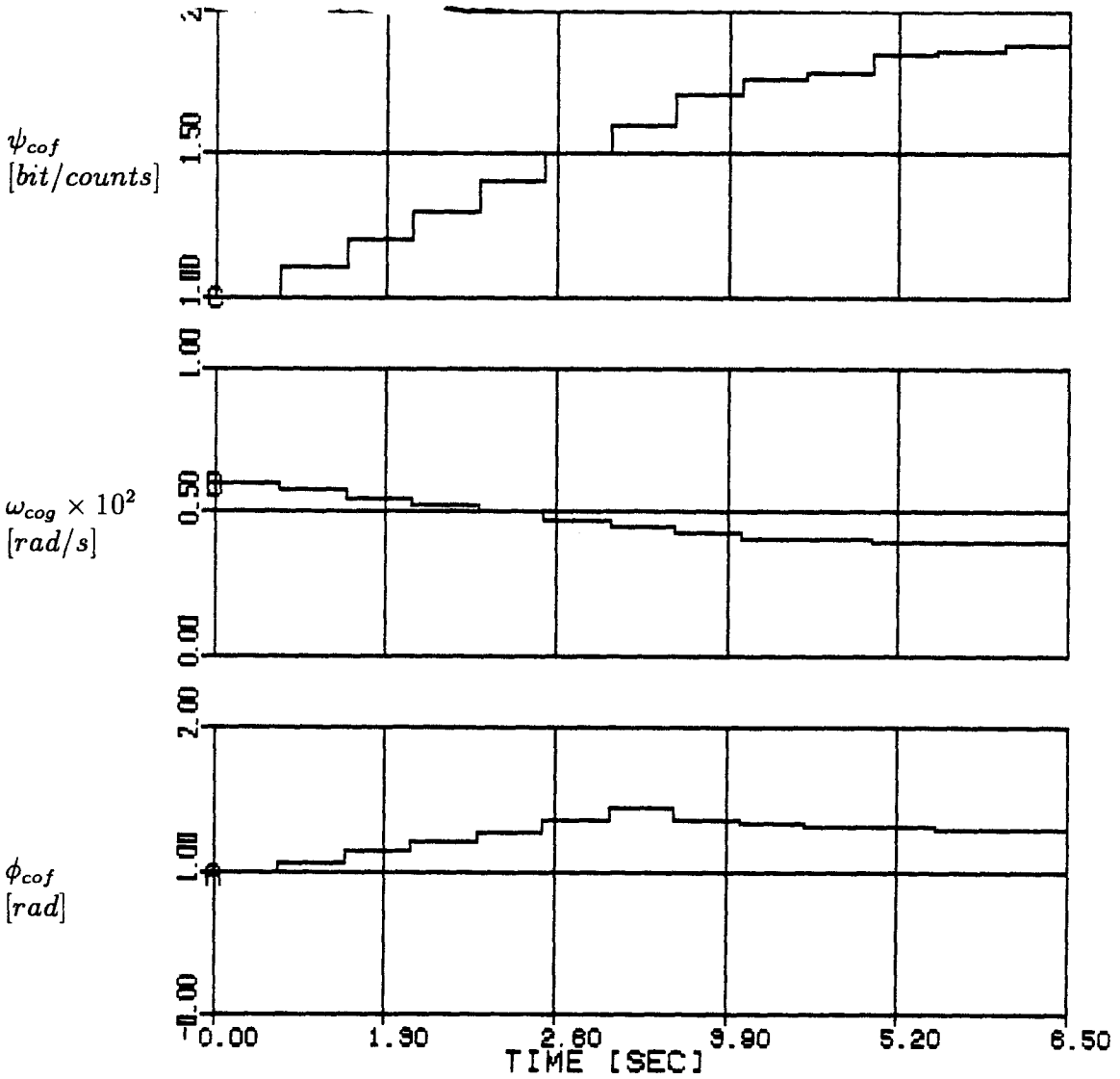


Figure 5.4: Tuning of an Initially Oscillatory Servo-Motor: Controller Attributes

5.1.3 Increased Motor Inertia

Adding inertia to the motor leads to sluggish and oscillatory response. The response before, while and after tuning is shown in Figure 5.5. The controller attributes are shown in Figure 5.6

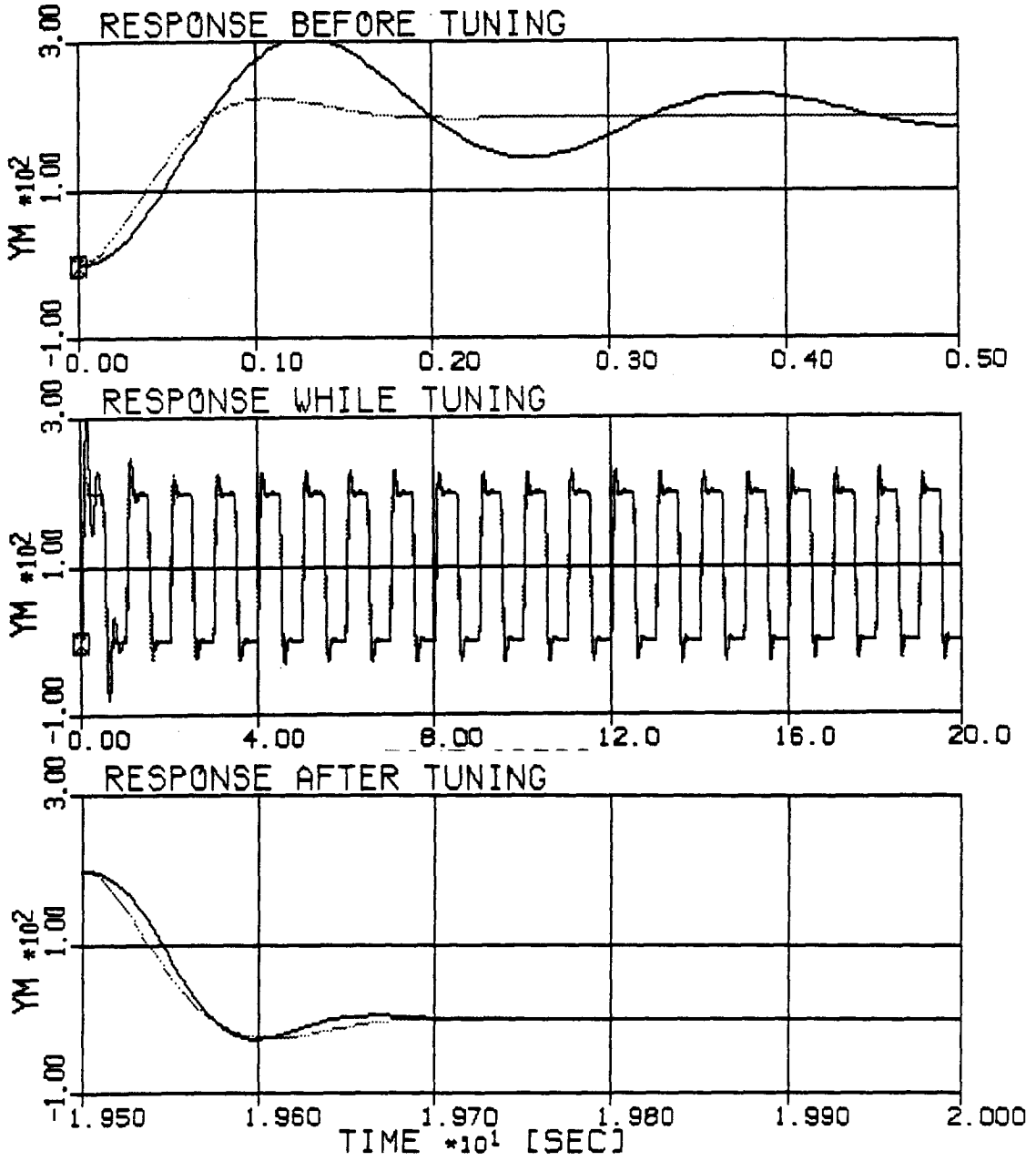


Figure 5.5: Tuning a Motor with Increased Inertia: Time Response

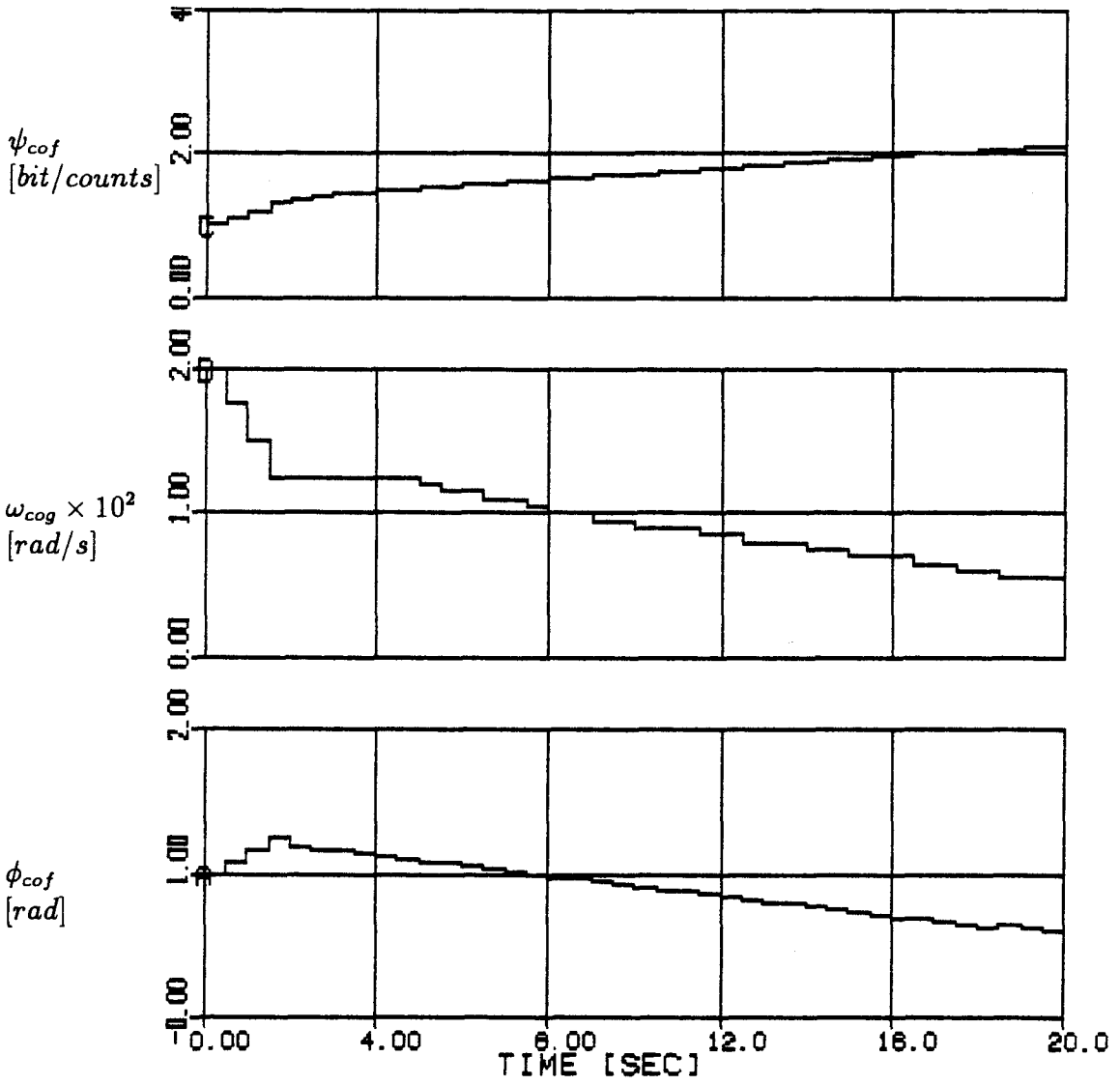


Figure 5.6: Tuning a Motor with Increased Inertia: Controller Attributes

5.1.4 External Torque without Integral Compensation

Without using the integrator compensation of the controller, a constant external torque in the system results in a steady, deterministic error (offset). The time response is shown in Figure 5.7 and the controller attributes in Figure 5.8

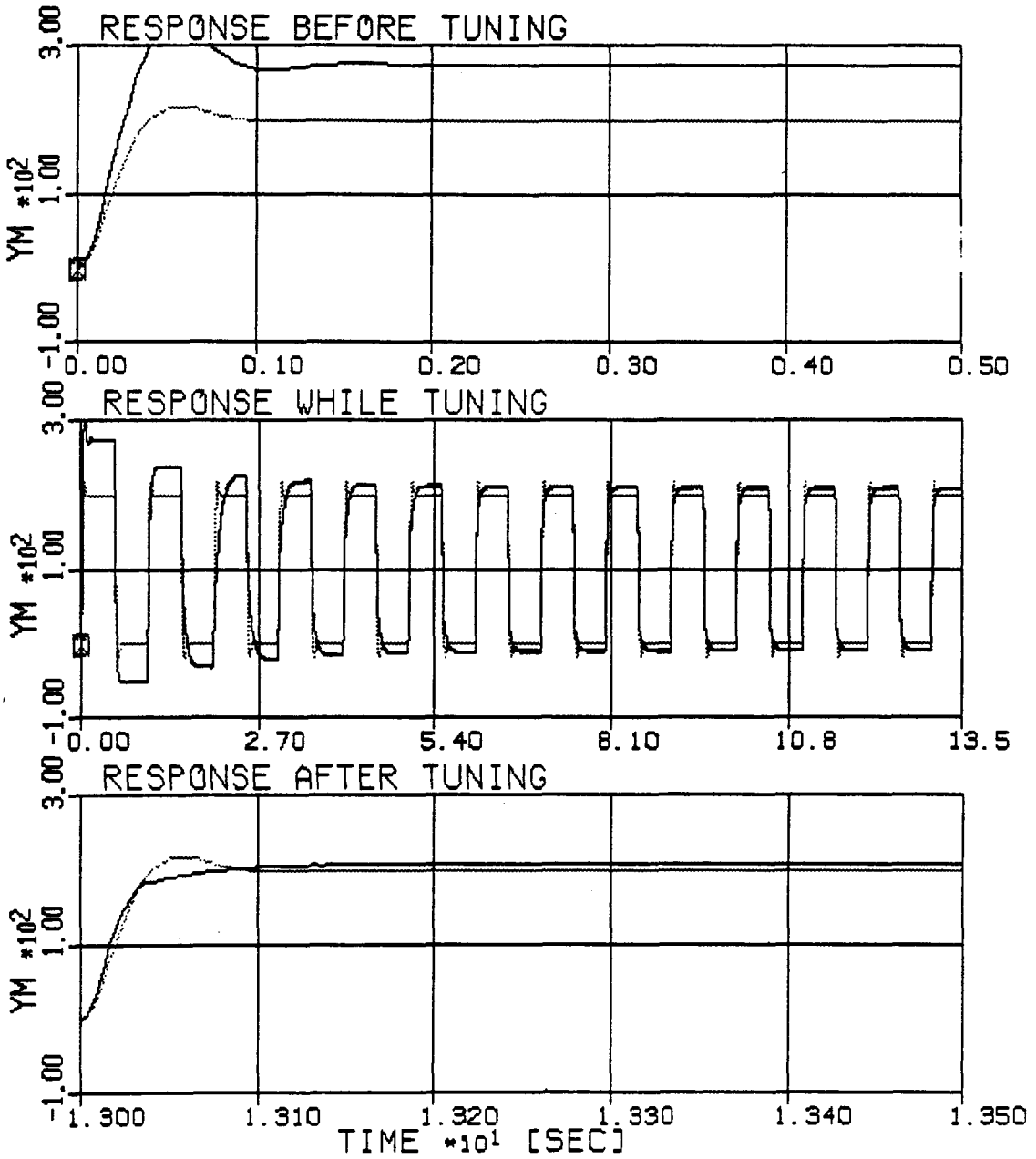


Figure 5.7: Tuning for External Torque without Integral Compensation: Time Response

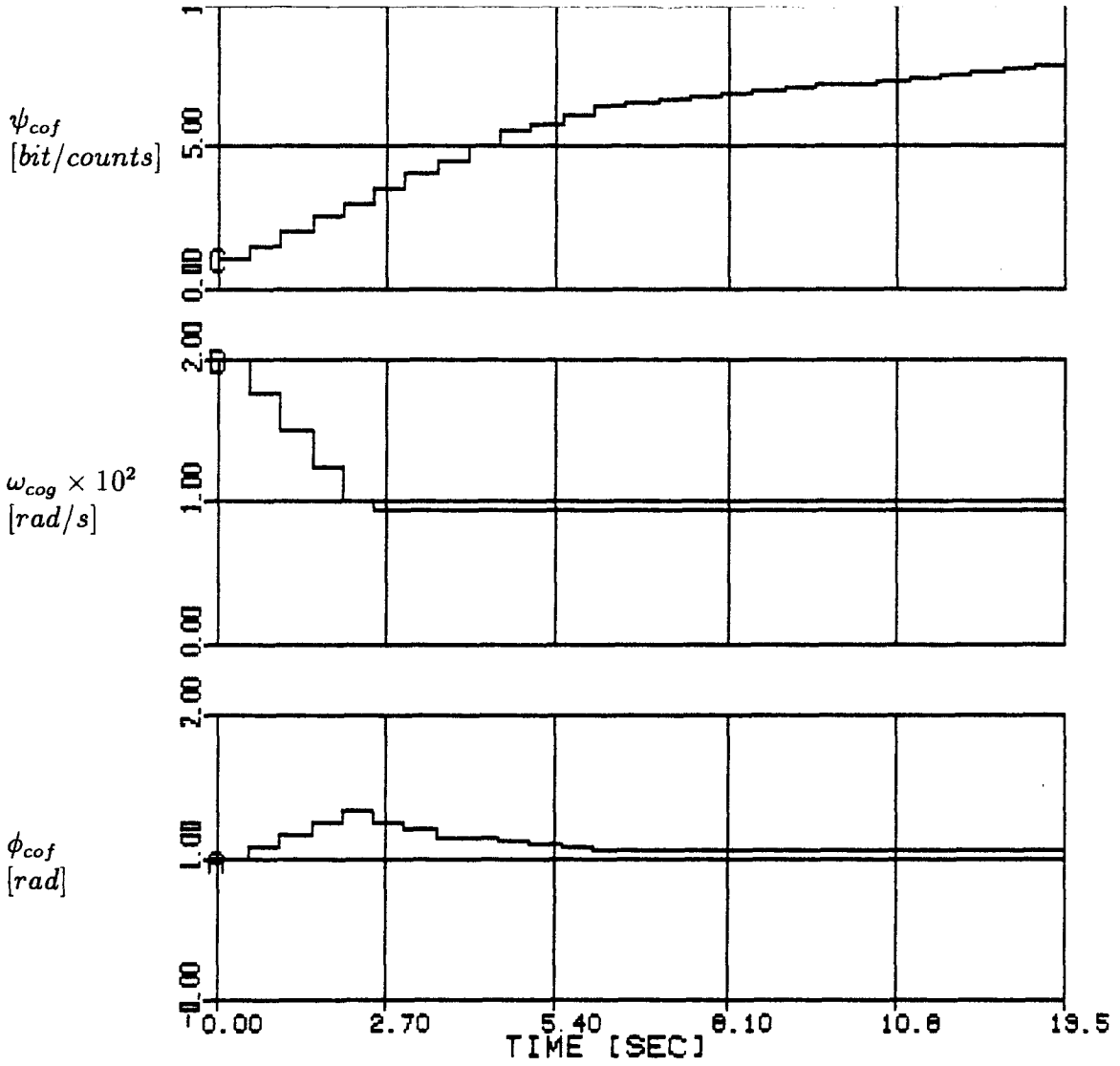


Figure 5.8: Tuning for External Torque without Integral Compensation: Controller Attributes

5.1.5 External Torque with Integral Compensation

In the presence of integral compensation, the steady state error in the response to a constant load will approach zero. How fast the error decays will depend on the D.C gain. The error decay toward an acceptable error within the test signal duration is demonstrated in Figure 5.9 and the controller attributes in Figure 5.10

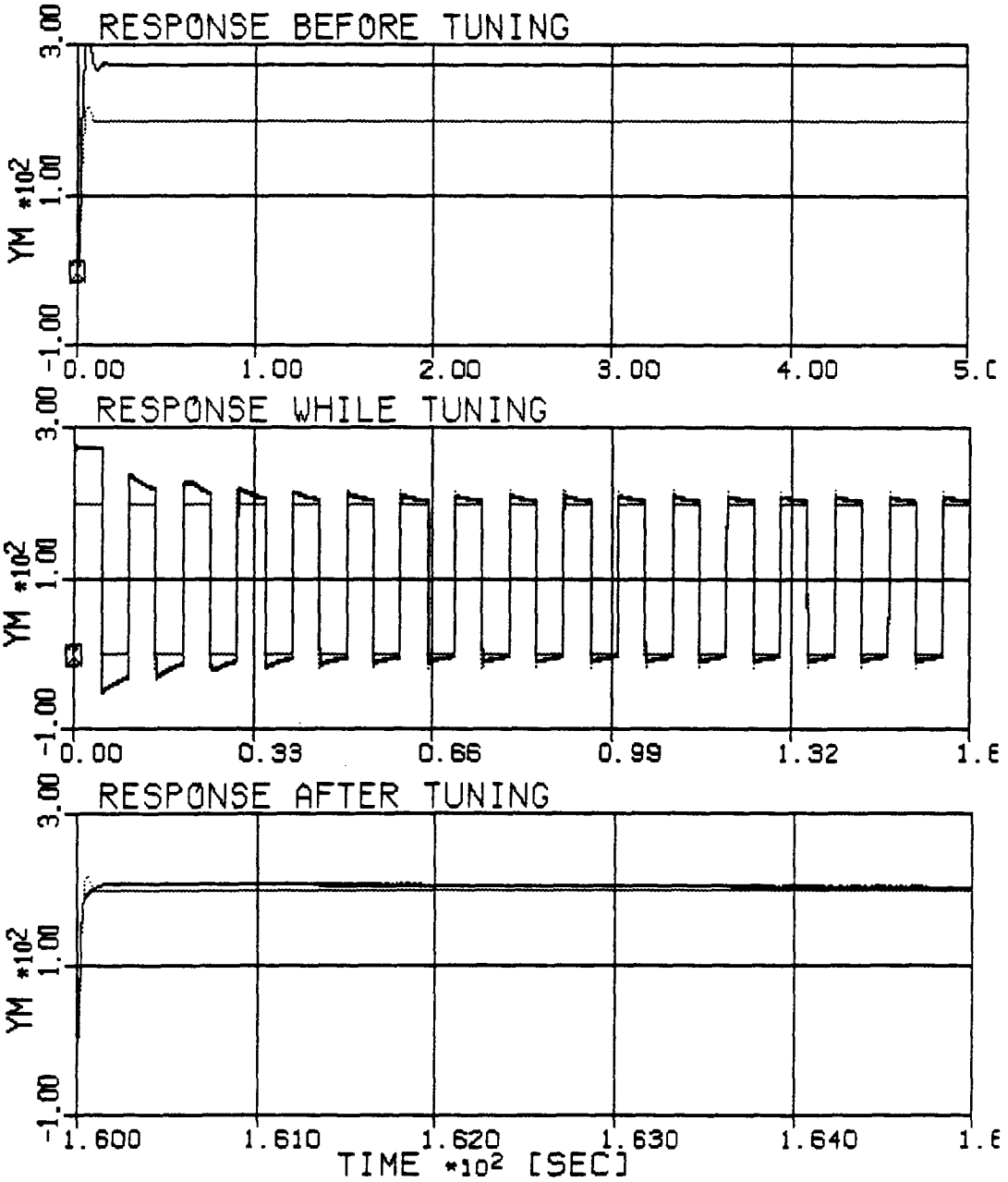


Figure 5.9: Tuning for External Torque, using Integral Compensation: Time Response

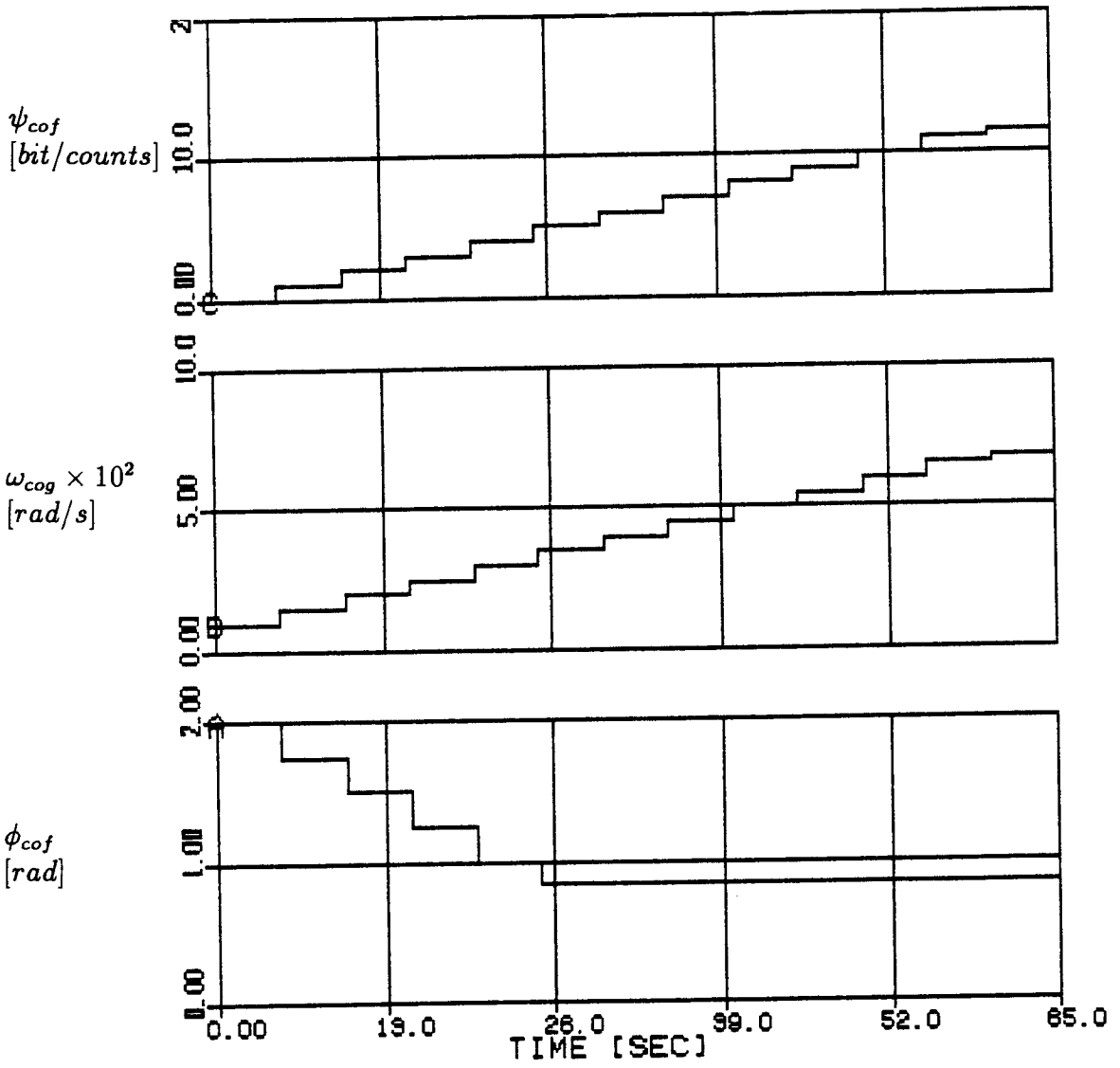


Figure 5.10: Tuning for External Torque, using Integral Comensation: Controller Attributes

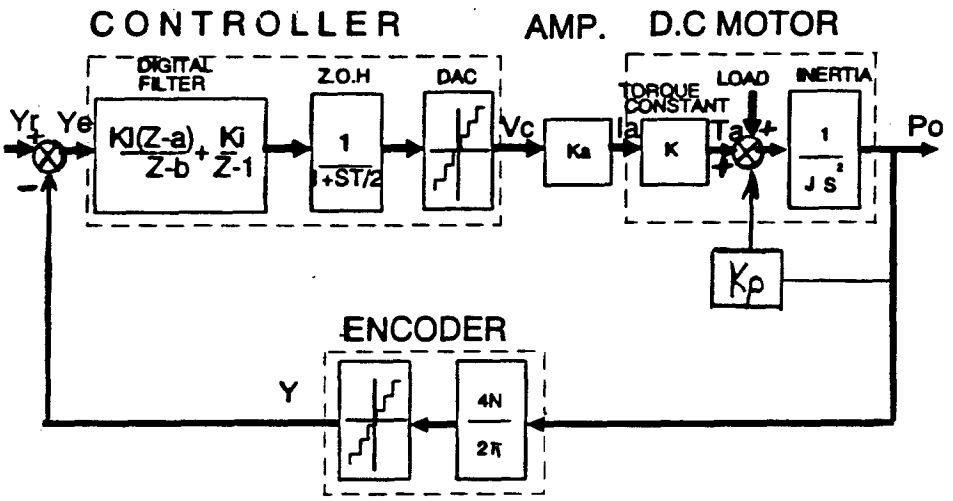


Figure 5.11: Motor Block Diagram for External Spring

5.1.6 External Spring

We can further complicate the servo-motor system by relating the external load to the actual response. For example let the external torque be a function of the motor position:

$$Tl_{(t)} = -Kp * Po_{(t)} \tag{5.1}$$

where Kp is a "spring" constant and Po is the actual motor position. A simple block diagram of this configuration is shown in Figure 5.7 . It can be seen that the plant dynamics now become a simple oscillator:

$$Po = \frac{A \cdot K/J}{s^2 + K \cdot K_p}$$

which should be compared with equation 3.3 on 47.

The response before, while and after tuning is shown in Figure 5.12 and the controller attributes are shown in Figure 5.12

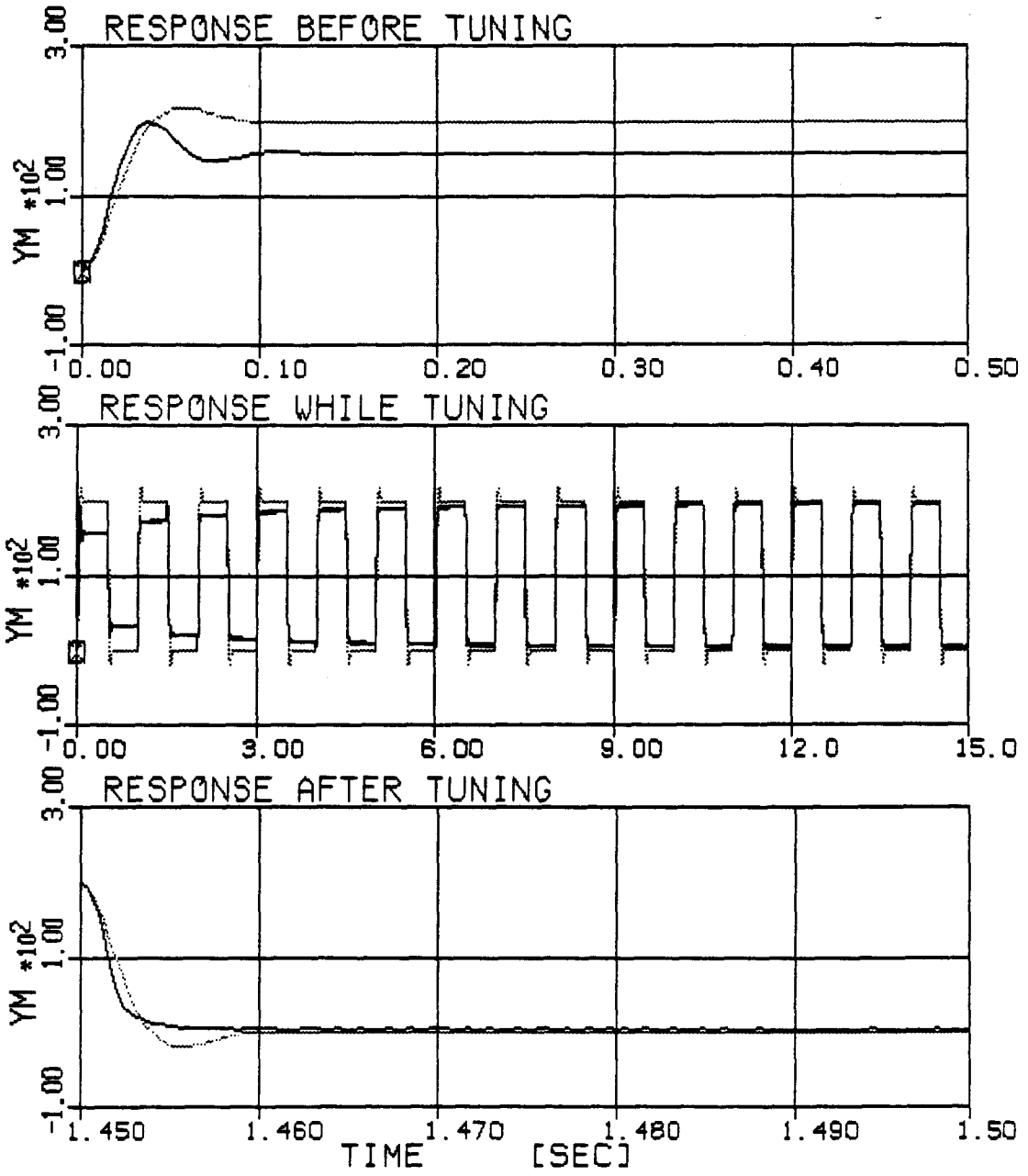


Figure 5.12: Tuning in the presence of an External Spring: Simulated System Response

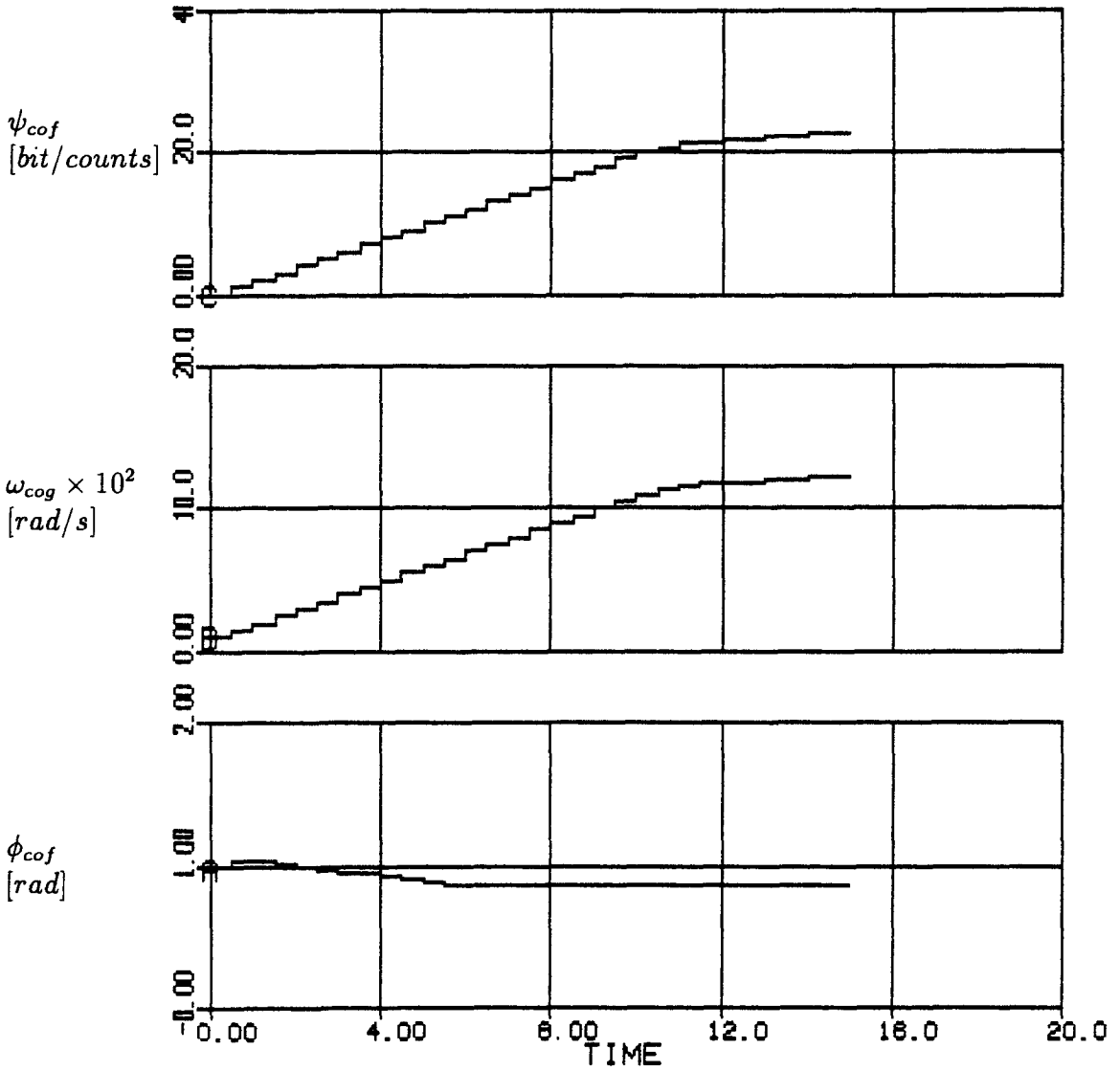


Figure 5.13: Tuning in the Presence of an External Spring: Controller Attributes

5.1.7 Voltage Source Amplifier Model

As described in Chapter 3, the motor-amplifier transfer function depends on the type of the amplifier. Figures 5.14 to 5.9 demonstrate the tuning process when applied to plant with higher dynamics and initially ill-tuned controller.

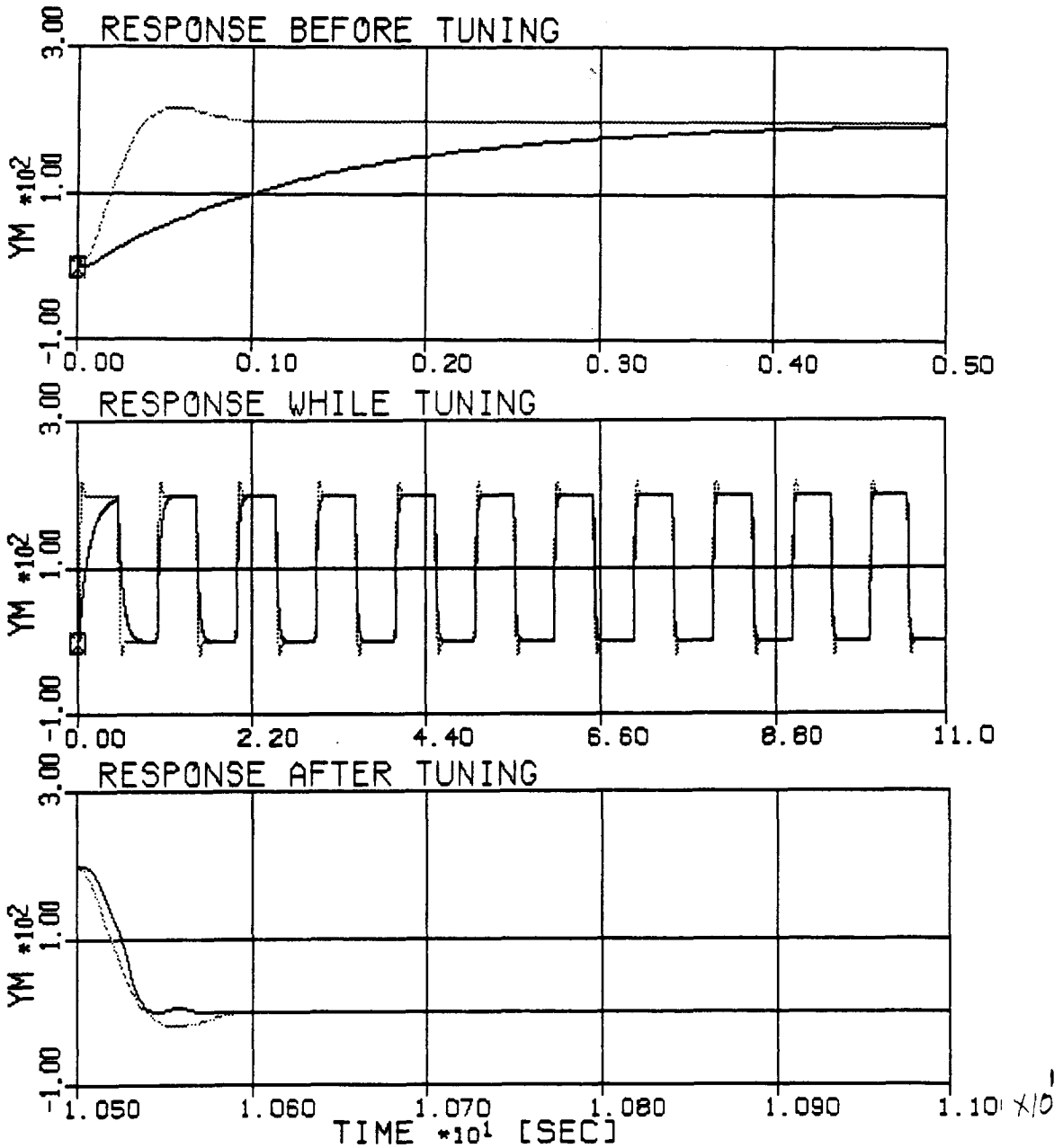


Figure 5.14: Tuning of a Slow Servo-Motor with Voltage Source Amplifier: Time Response

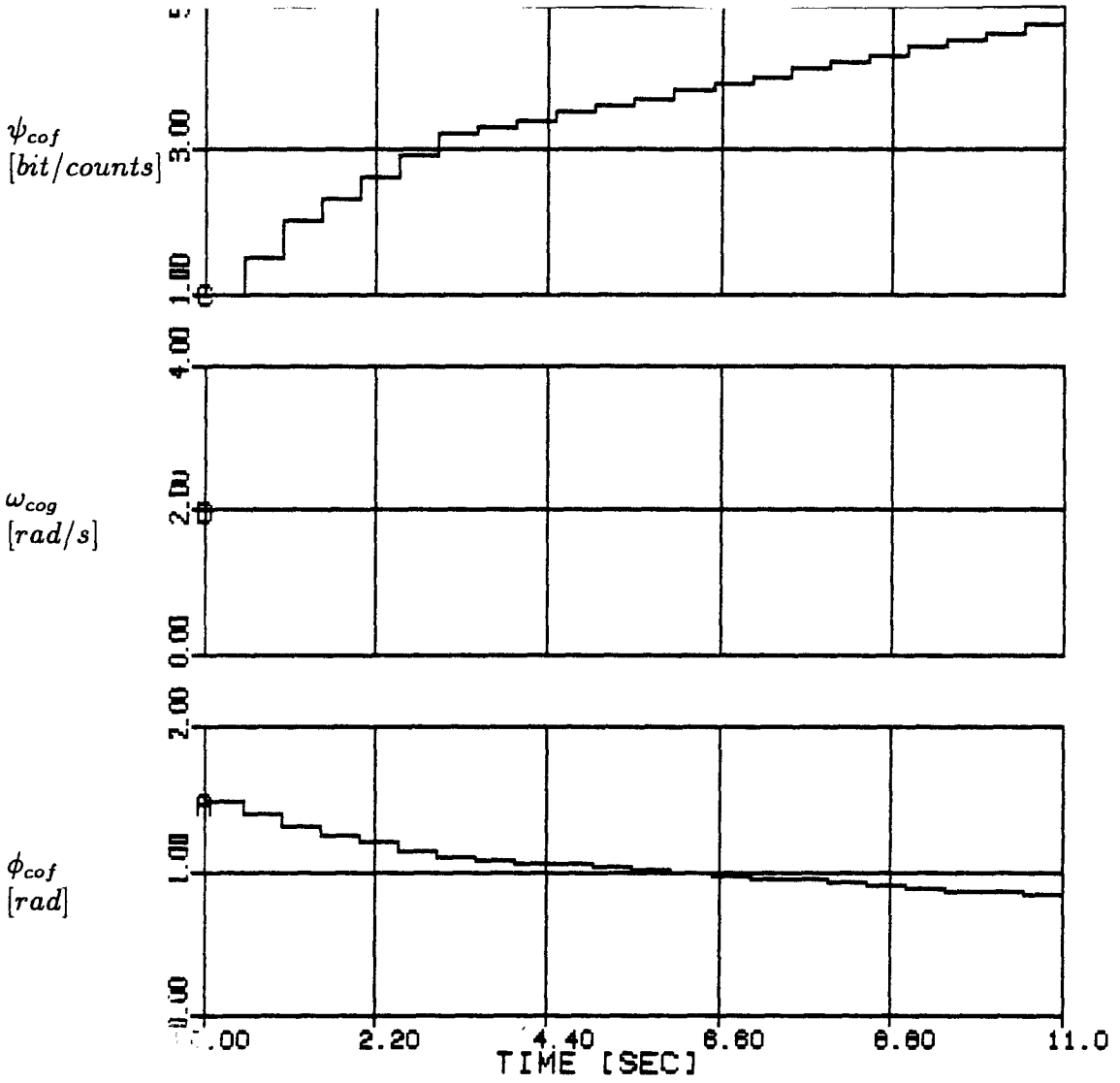


Figure 5.15: Tuning of a Slow Servo-Motor with Voltage Source Amplifier: Controller Attributes

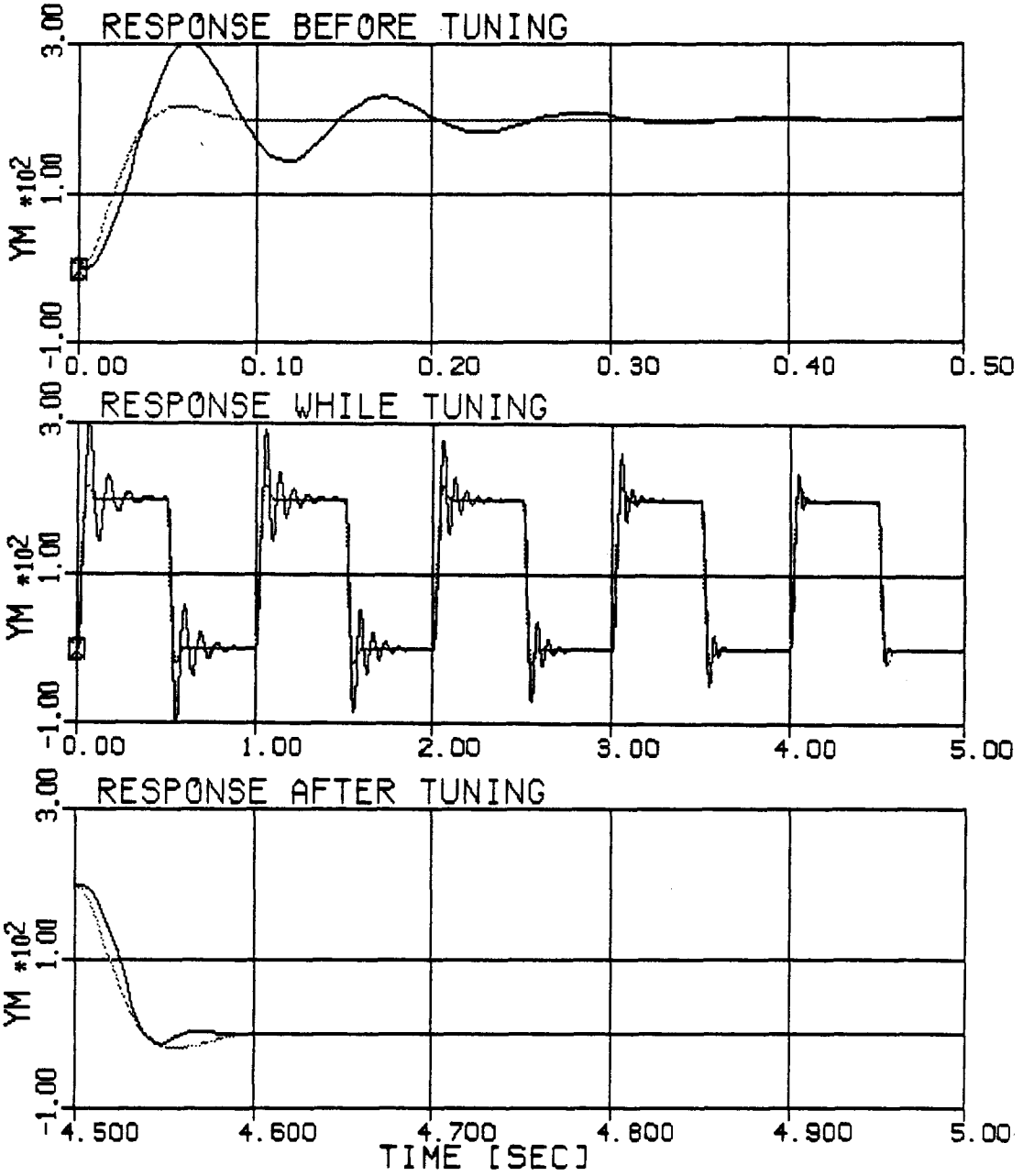


Figure 5.16: Tuning for Oscillaory Servo-Motor with Voltage Source Amplifier – Time Response

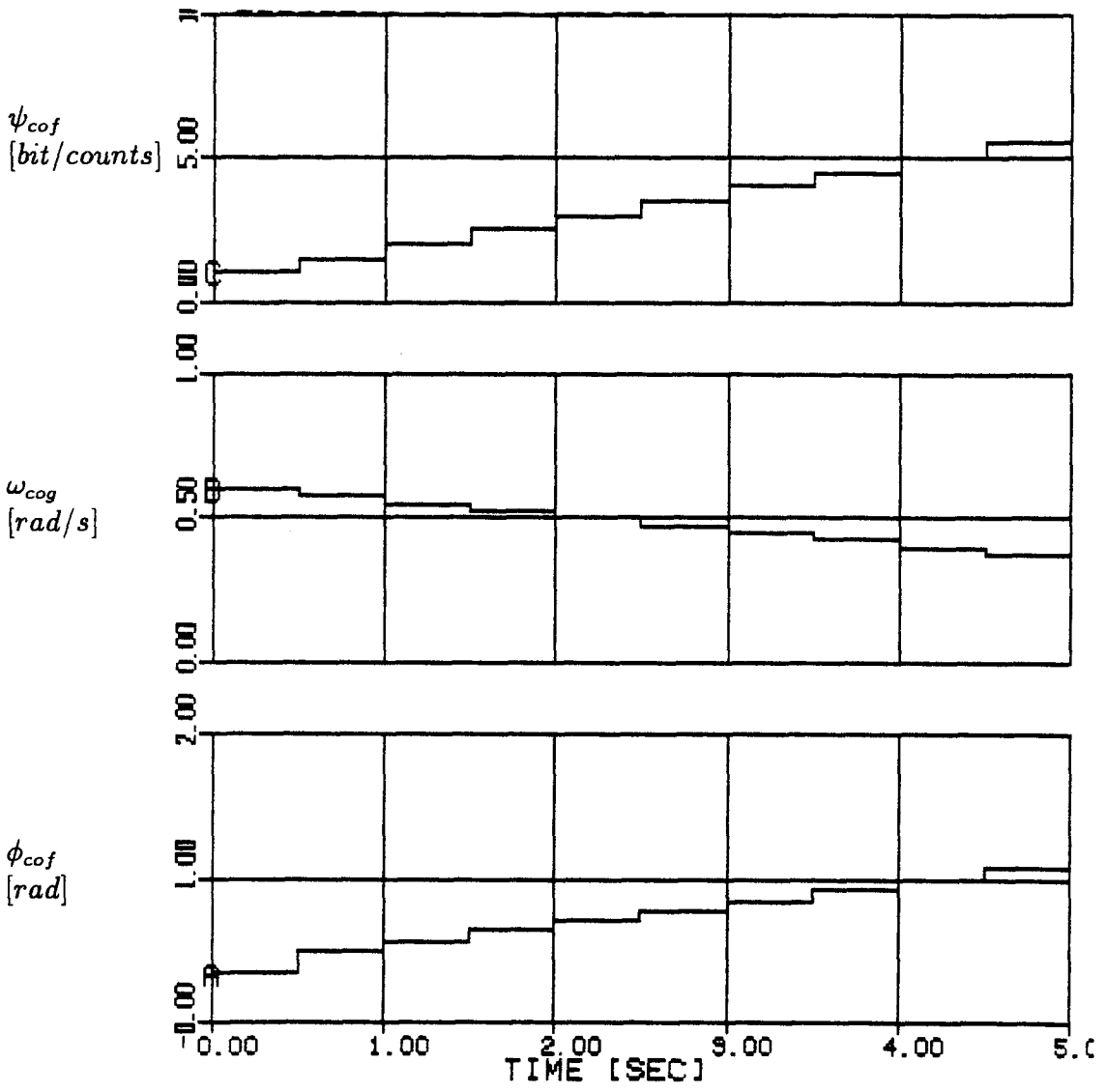


Figure 5.17: Tuning of an Oscillatory Servo-Motor with Voltage Source Amplifier: Controller Attributes

5.2 Experiments using a Commercial Servo-Motor

The same first three experiments in the simulated system are replaced using the commercially available servo-motor system.

5.2.1 Slow Physical Servo-Motor System

The response before, while and after tuning an initially ill-tuned physical controller is shown in Figure 5.18. The controller attributes are shown in Figure 5.19

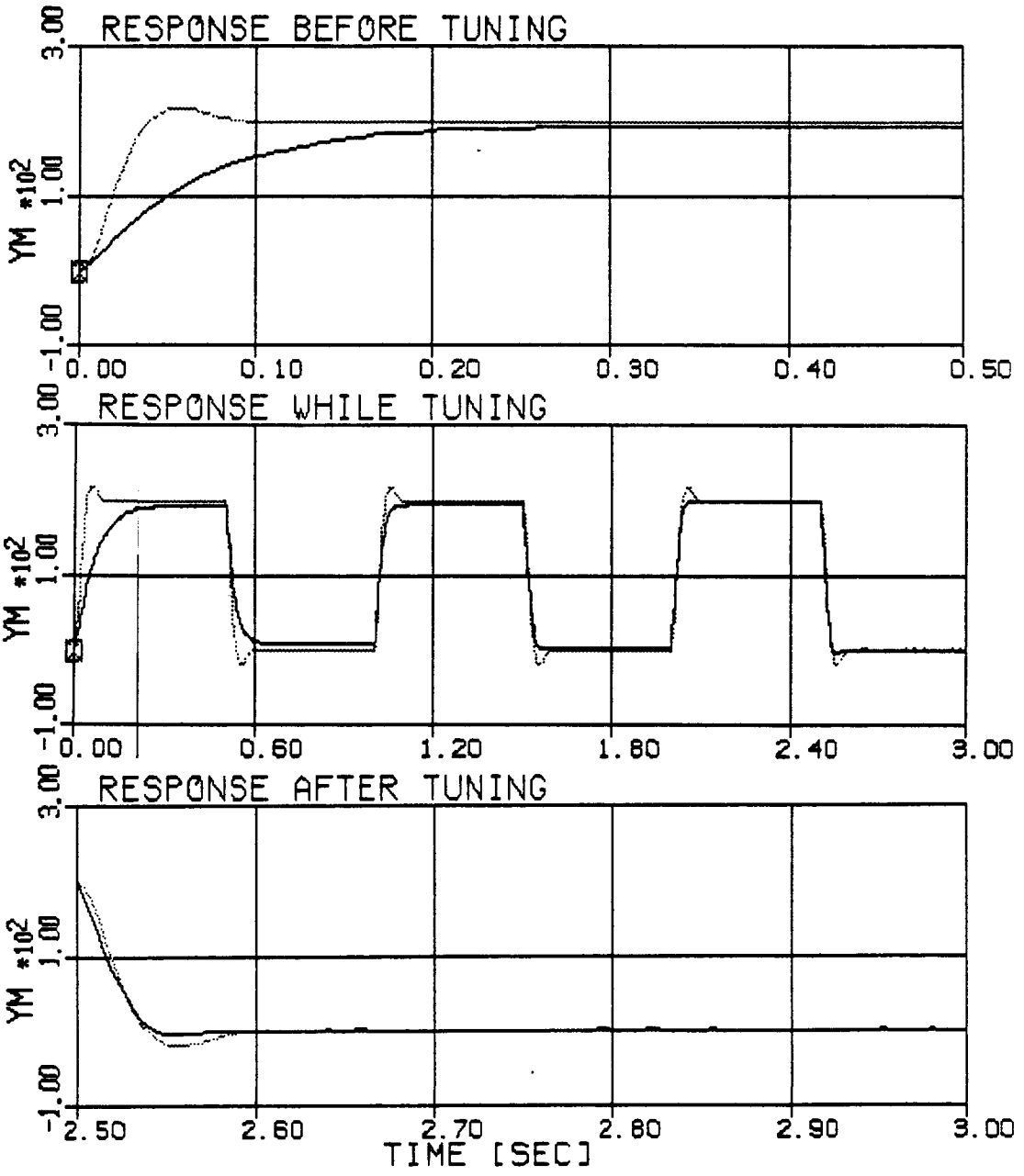


Figure 5.18: Tuning of a Slow Servo-Motor: Time Response of the Physical System

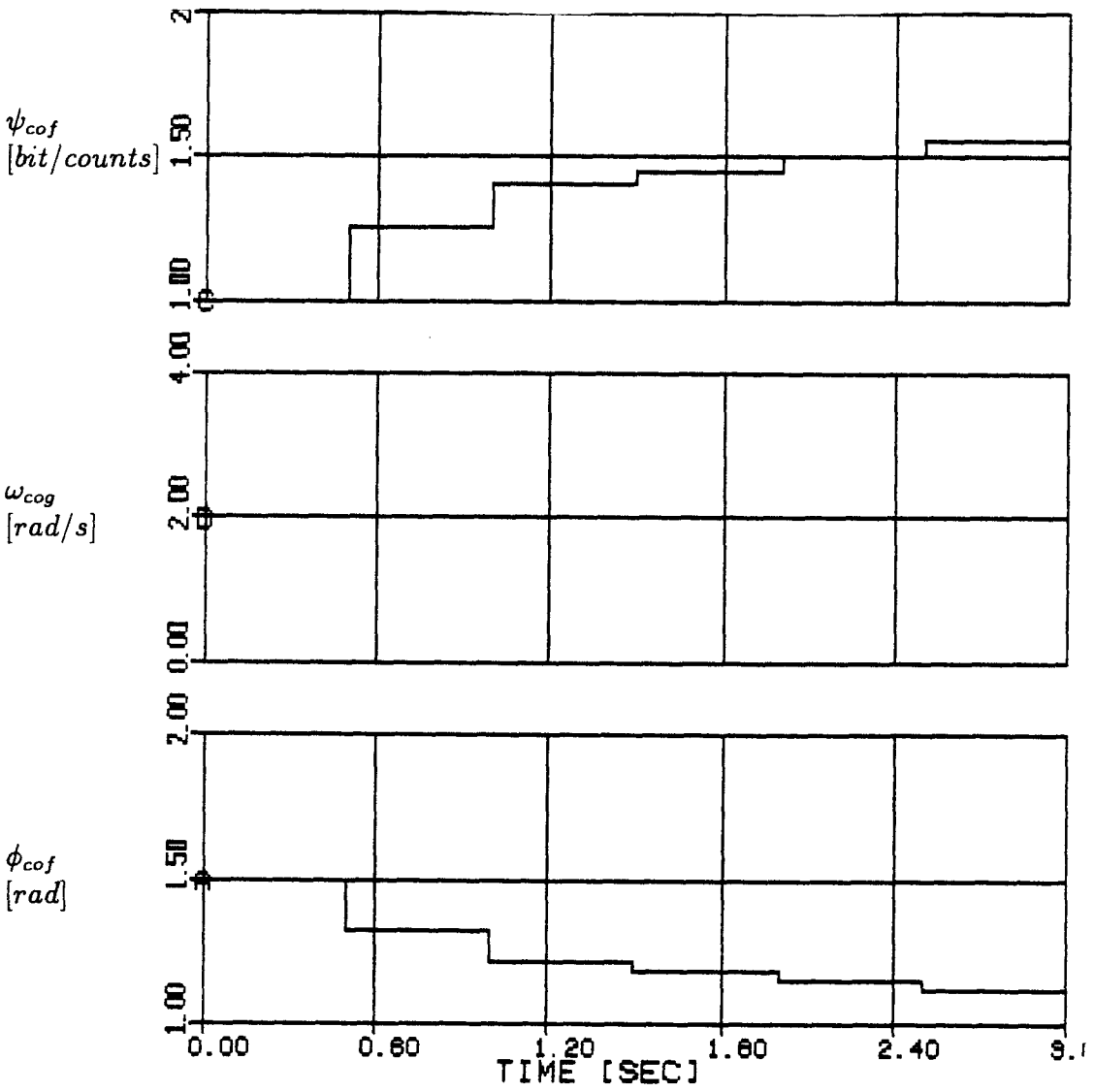


Figure 5.19: Tuning of a Slow Servo-Motor: Controller Attributes of the Physical System

5.2.2 Physical Oscillatory Servo-Motor

The response before, while and after tuning an initially ill-tuned physical controller are shown in Figure 5.20. The controller attributes are shown in Figure 5.21 .

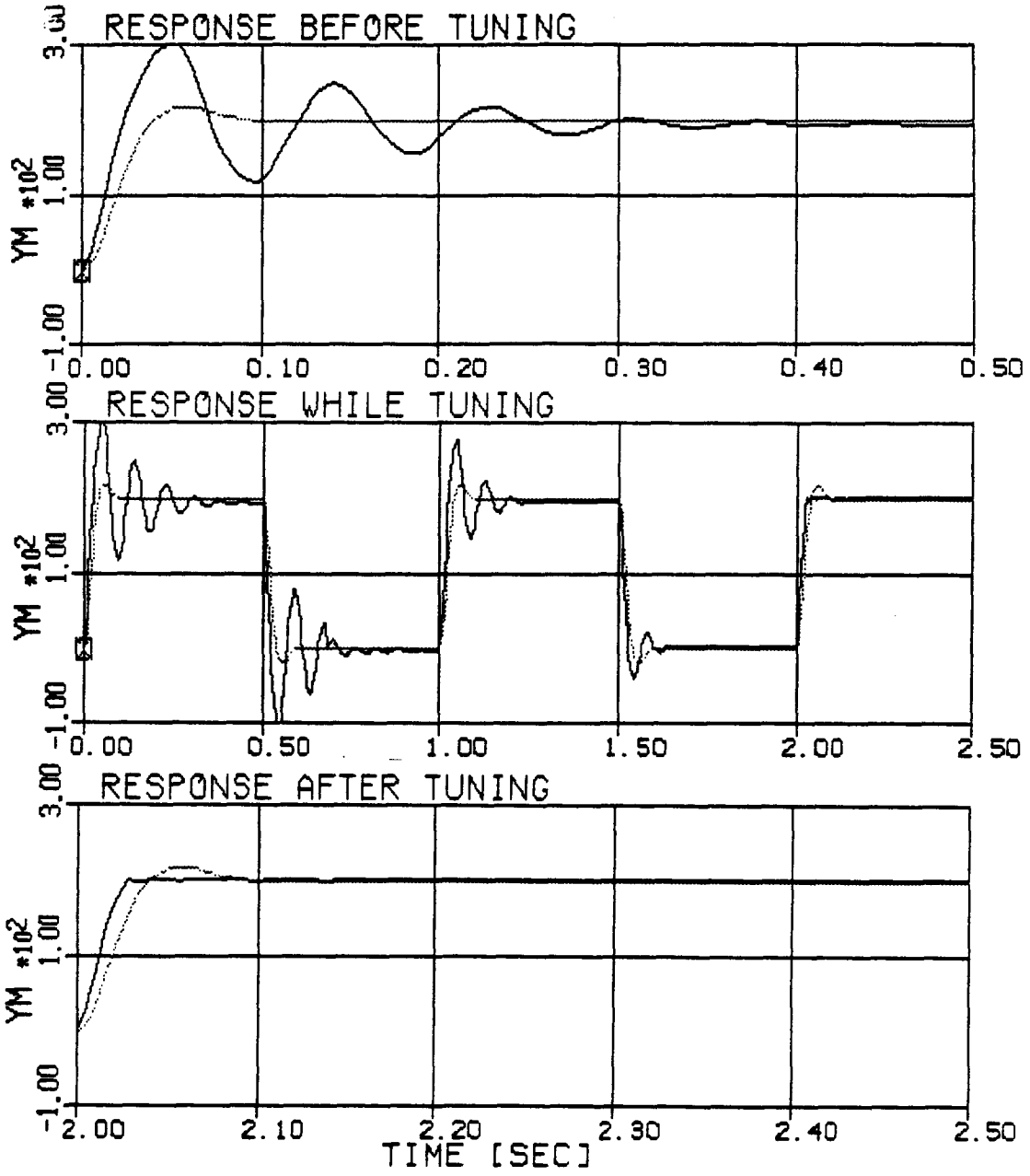


Figure 5.20: Tuning of an Oscillatory Servo-Motor: Response of the Physical System

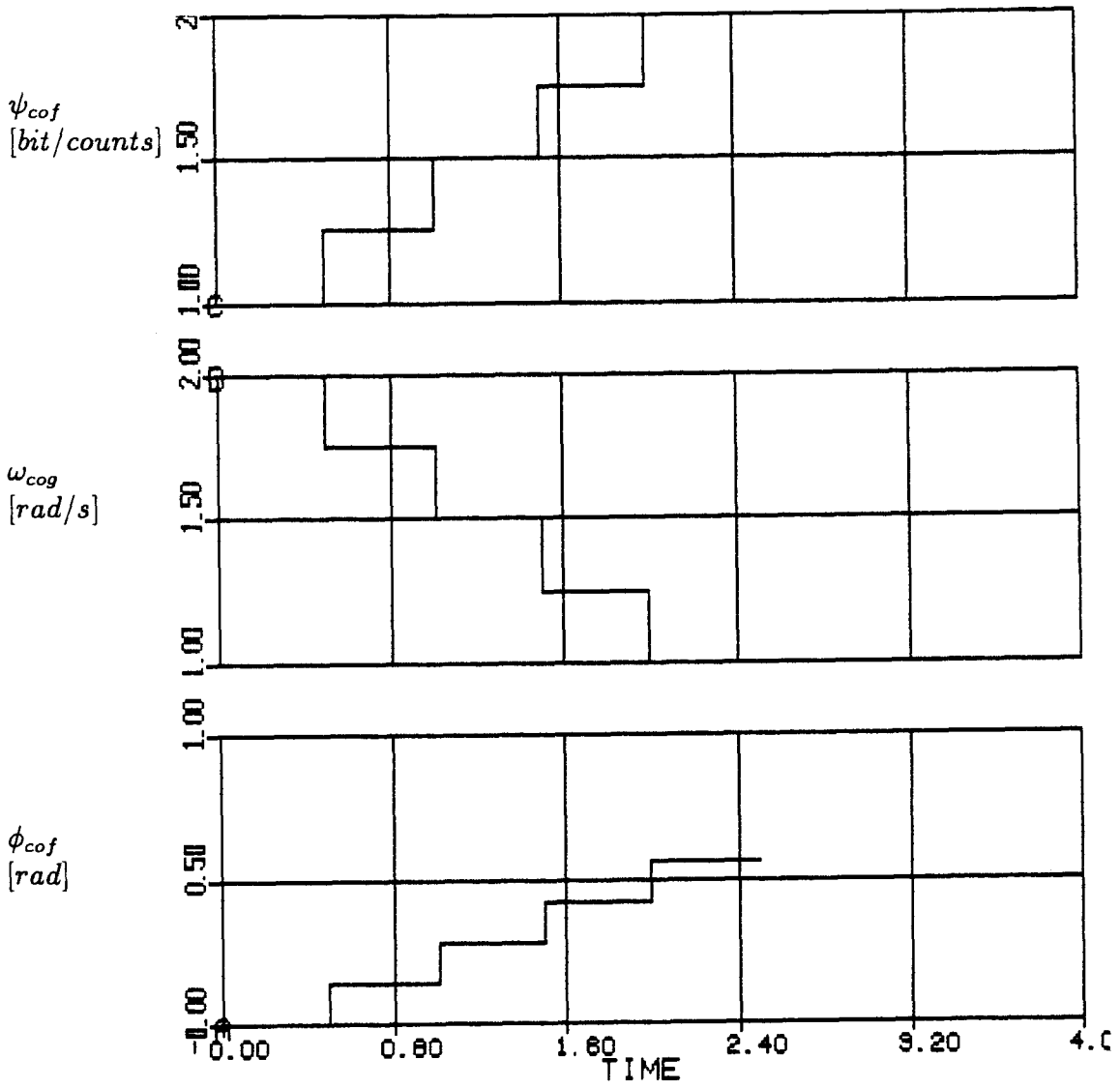


Figure 5.21: Tuning of an Oscillatory Servo-Motor: Controller Attributes of the Physical System

5.2.3 Motor with Increased Inertia

Additional inertia to the physical motor leads to sluggish and oscillatory response. The response before, while and after tuning are shown in Figure 5.22 (a) . The controller attributes are shown in Figure 5.23

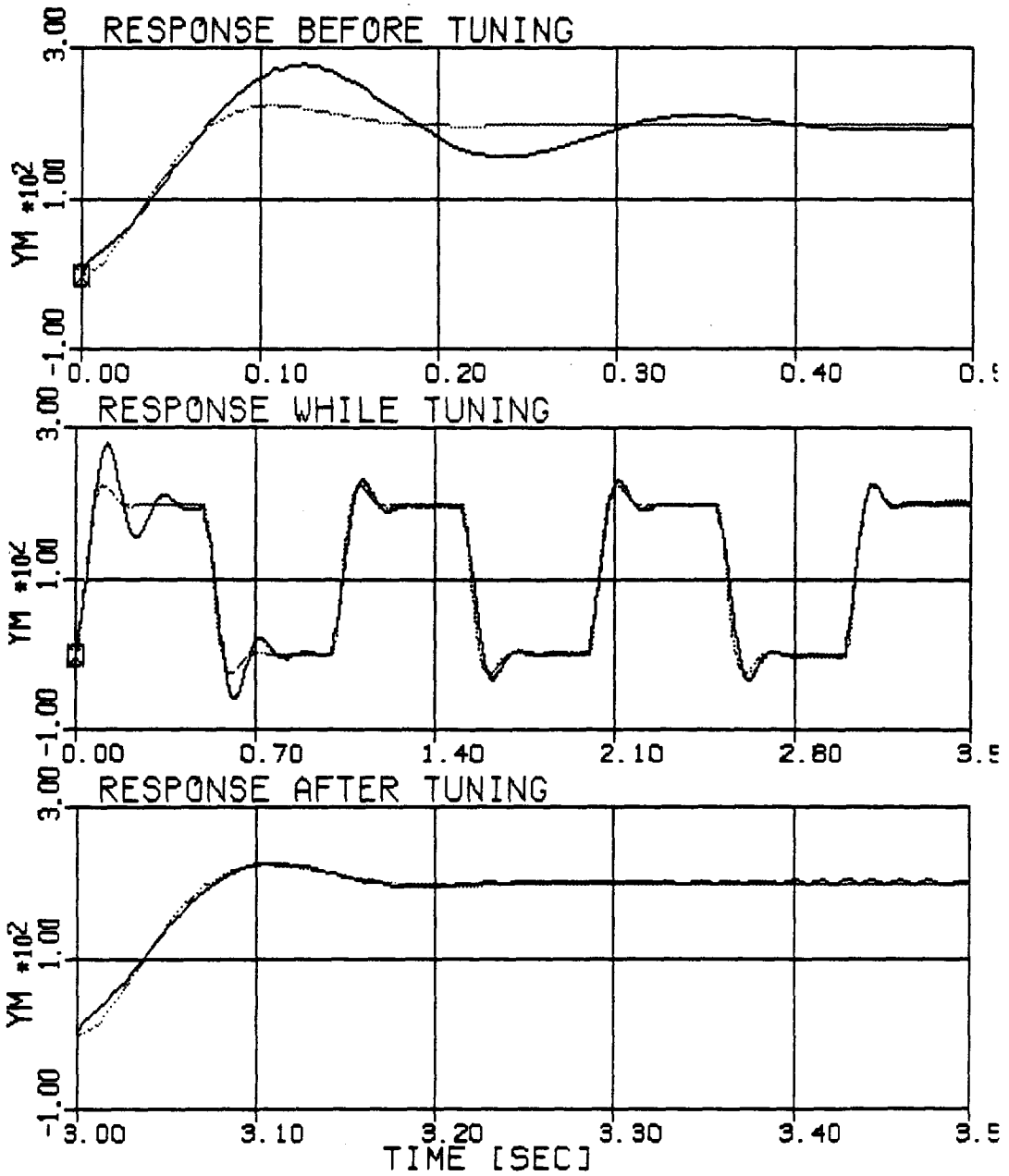


Figure 5.22: Tuning for Increased Motor Inertia: Physical System Response

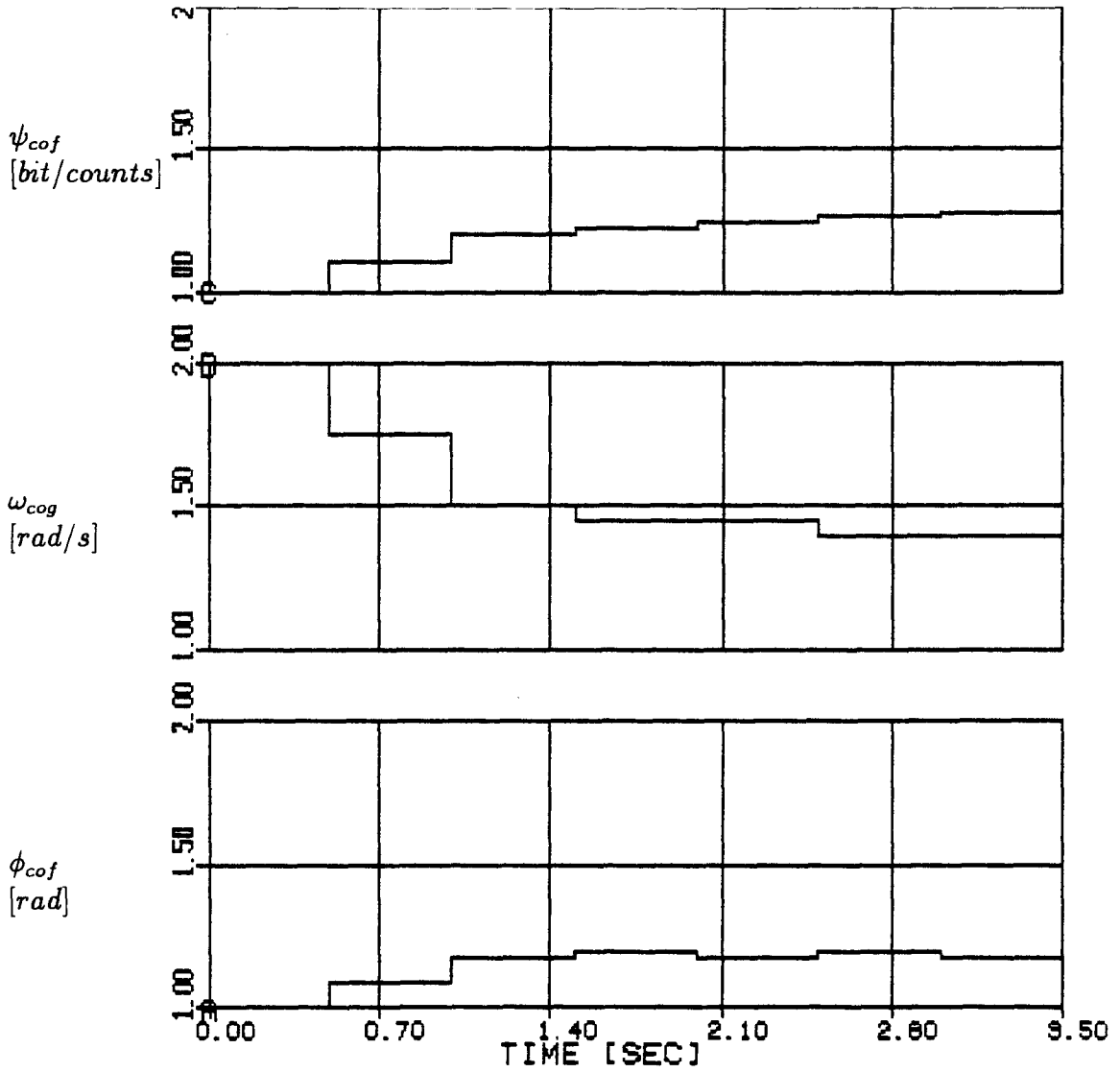


Figure 5.23: Tuning for Increased Motor Inertia: Physical System Controller Attributes

Chapter 6

RULE GENERATING BY SELF LEARNING

6.1 Introduction

The knowledge presented in the previous chapters simulates tuning actions taken by an expert operator and are expressed as a set of linguistic rules. This knowledge might be replaced or modified by a set of rules generated automatically in a preliminary phase of the learning process to be used later in the the fuzzy tuner level.

The learning process might be useful in the following scenarios :

- Compensating for total lack of tuning knowledge (for example when implemented in a new process.)
- Adding to an incomplete tuning knowledge base.
- Modifying incorrect rules.
- Deleting unnecessary or irrelevant rules.

A learning process is demonstrated in this chapter by replacing the knowledge- based ruleset that was introduced in the previous chapters.

6.2 Self Generated Rulset Development

6.2.1 Approach

Learning process results either in tuning rules similar to those generated by a human operator, or directly in a decision table. The first approach is adopted in this research since it enables us to integrate an already existing knowledge with new automatically generated rules. Furthermore, when the new and self-generated ruleset is available, we can use an already developed algorithm to calculate the decision table, leaving the fuzzy tuner unchanged in the highest level, by just switching from tuning to learning mode.

In both the learning and tuning modes, the servo-motor and the reference model are excited by test signal. The measured response is evaluated to perform the same normalized performance parameters as described in Chapter 3.

First, the controller attributes are **preset** corresponding to a set of reference performance parameters. Next the system is perturbated by changing the controller attributes (one at a time), and the difference between the performance parameters of the preset and perturbed systems is calculated and classified to form a **sensitivity index**. This index is used to trigger a rule out of a predefined rule patterns.

A complete mathematical treatment includes the computation of the sensitivity of each performance parameter to each tuning action in the entire working space (all the combinations of the components of the controller attributes). This treatment is beyond the scope of this research. However, a much simpler algorithm is obtained if we assume that each performance parameter is changed **monotonously** when the system is perturbated. This assumption is justified over a wide range (see Figure 4.2 throgh Figure 4.6) for the particular system.

Based on this assumption, the sensitivity is computed when the system is perturbated from a **single, representative** working point. Further simplification is obtained if we

define a finite number of patterns for the rules, and then trigger an appropriate pattern according to the sensitivity index.

The learning system is based on the same servo-expert algorithm used in the tuning mode as described on Chapter 3.4. Here, however, a sensitivity index triggers rules out of the rule generator while in the previous case a performance index triggers a tuning action out of the decision table. Once the new, and self-generated rules are avialable, the same fuzzy tuner algorithm, as described in Chapter 3.5, is used to calculate the decision table.

A block diagram of the system in the learning mode is shown in Figure 6.1

The code of the subprogram LEARN is listed in Appendix B-3-8

6.2.2 Controller Presetting

The presetting of the controller attribute can enrich the system response with information. For example, the system should be oscillated if we want to generate rules relating any tuning action to the damped natural frequency, or a steady error will be present if we generate rules relating offset to any tuning action.

Even though an automated presetting process might be applied by systematically scanning the controller attributes and using the performance parameters as a feedback, a manual, trial and error process is used in this research for the sake of simplification.

6.2.3 System Perturbation

As in the presetting process, for simplicity, a trial and error process was adopted rather than an automated one, to perturb the preset system. Each of the controller attributes is changed, one at a time, leaving all the remaining attributes unchanged (having the preset values). This process results in L sets of response parameters, each of them representing

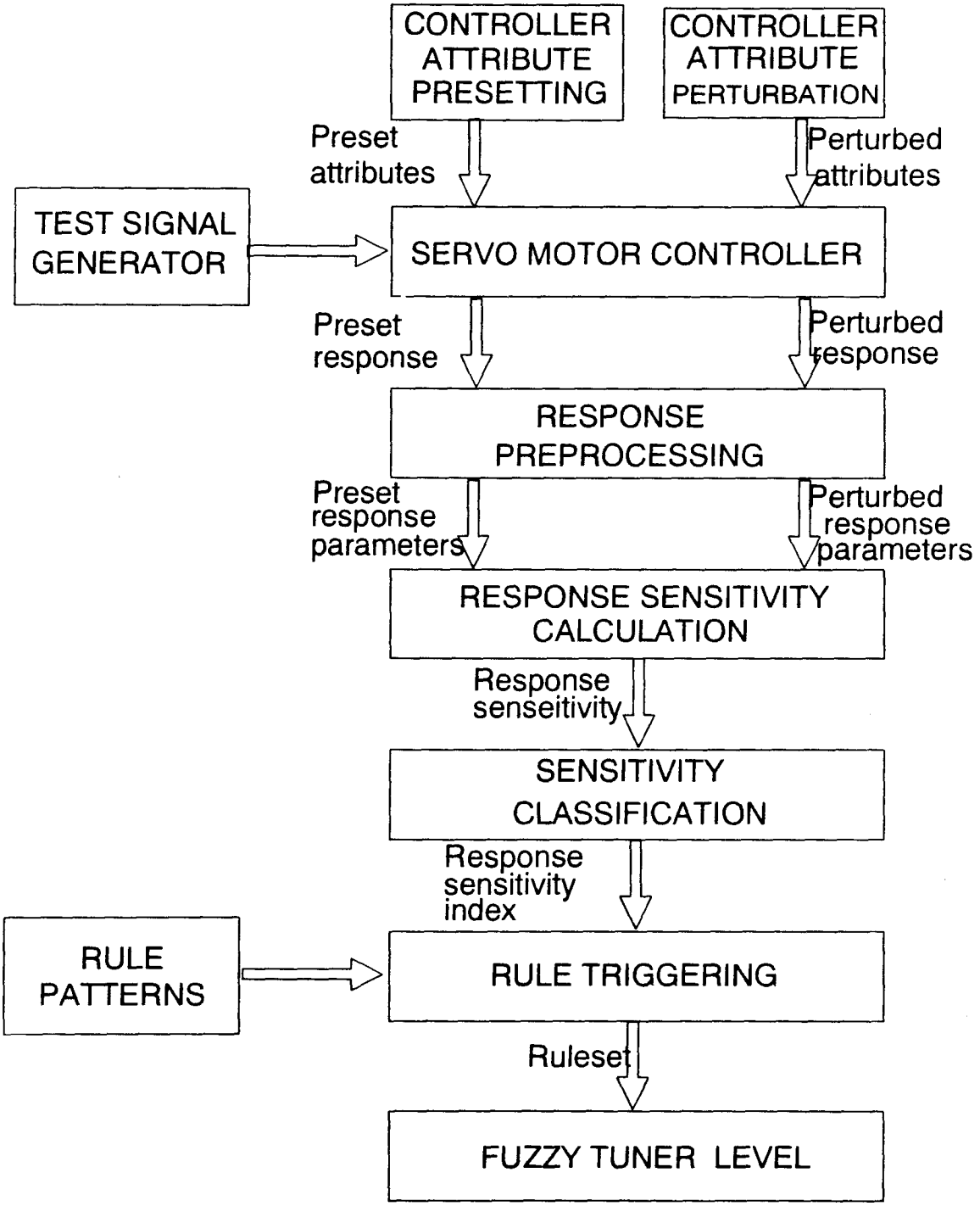


Figure 6.1: The System in Learning Mode

the change in the response due to change in one controller attribute near the preset values.

6.2.4 Response Sensitivity Index

The performance parameters, for both the preset and the perturbed systems, are computed using the response preprocessor and evaluator as described in section 3.4, and the difference between them are classified to form the *sensitivity index.*

The preset system is denoted the superscript ^{0}. Let $\odot^0$ be a vector containing the controller attribute of the preset system, (see equation 3.29 on page 62) and the vector P^0 be its response (see equation 3.21 on page 57).

Let $\odot$ be the perturbed attribute vector. We define the normalized l'th perturbed attribute:

$$\delta\odot_{(l)} = \frac{\odot_{(l)} - \odot^0_{(l)}}{\odot^0_{(l)}} \tag{6.1}$$

and the normalized change in the performance parameter i:

$$\delta P_{(i)} = \frac{P_{(i)} - P^0_{(i)}}{P^0_{(i)}} \tag{6.2}$$

The performance sensitivity $S^0_{(i,l)}$ is computed as follows:

$$S^0_{(i,l)} = \frac{\delta P_{(i)}}{\delta \odot_{(l)}} = \frac{P_{(i)} - P^0_{(i)}}{\odot_{(l)} - \odot^0_{(l)}} \cdot \frac{\odot^0_{(l)}}{P^0_{(i)}} \tag{6.3}$$

for all i and all l.

The superscript ^{0} was added to the performance sensitivity to emphasise that it is calculated near the preset attribute.

Rearrange equation 6.3 as:

$$\odot_{(l)} = \odot^0_{(l)} + \frac{P_{(i)} - P^0_{(i)}}{S^0_{(i,l)}} \cdot \frac{\odot^0_{(l)}}{P^0_{(i)}} \tag{6.4}$$

Scanning over all of the components of $\odot^0$ in the range from $\odot_{(l)_{min}}$ to $\odot_{(l)_{max}}$ (see equation 3.30 on page 63) we can map the sensitivity for all i and l and generate rules which are based on this performance sensitivity array.

However, as mentioned earlier, an automated scanning process, using appropriate resolution is beyond the goals of this research. For simplicity we compute this performance sensitivity at one point only (the preset values of the controller attributes).

Recall that a smaller $P_{(i)}$ value indicates better performance. We see that negative $S^0_{(i,l)}$ indicates improvements of the i th response parameter due to the l th controller attribute, while positive $S^0_{(i,l)}$ indicates decreasing in performance relative to the preset system.

$S_{(i,l)} = 0$ indicates that the performance parameter i is not sensitive to the controller attribute l.

6.2.5 Response Sensitivity Classification

The response sensitivity is classified now according to J predefined thresholds $TH_{(j)}$, resulting in $J + 1$ subsets. Each of these are assigned an integer value $Is_{(i)}$ and will be served as a sensitivity index. For simplicity two threshold values are used in this research to form three subsets:

$$Is_{(i)} = \begin{cases} -1 & \text{if} \quad S^0_{(i,l)} \leq TH_{(1)} \\ 0 & \text{if} \quad TH_{(1)} < S^0_{(i,l)} \leq TH_{(2)} \\ +1 & \text{if} \quad TH_{(2)} < S^0_{(i,l)} \end{cases} \tag{6.5}$$

6.2.6 Rule Pattern

Tuning rules are arranged in predefined patterns, each of them containing 5 rules arranged in an array form $\widetilde{RU}_{(i,j,l)} = m$ as defined in equation 3.48 on page 73, and triggered by the performance sensitivity index.

An example for a pattern triggered by negative $I_{(i)}$ is shown in equation 6.6.

$$\begin{aligned}
\widetilde{RU}_{(i,1,l)} &= -2 \quad (If\ PV_{(i,l)} = UNSTF \quad \text{then}\ TA_{(l)} = NEGHI) \\
\widetilde{RU}_{(i,2,l)} &= -1 \quad (If\ PV_{(i,l)} = POOR \quad \text{then}\ TA_{(l)} = NEGLO) \\
\widetilde{RU}_{(i,3,l)} &= -1 \quad (If\ PV_{(i,l)} = MODRT \quad \text{then}\ TA_{(l)} = NEGLO) \\
\widetilde{RU}_{(i,4,l)} &= 0 \quad (If\ PV_{(i,l)} = INSPC \quad \text{then}\ TA_{(l)} = NOCNG) \\
\widetilde{RU}_{(i,5,l)} &= 1 \quad (If\ PV_{(i,l)} = OVRSP \quad \text{then}\ TA_{(l)} = POSLO)
\end{aligned} \tag{6.6}$$

Example for a pattern triggered by positive $I_{S_{(i)}}$ is shown in equation 6.7 :

$$\begin{aligned}
\widetilde{RU}_{(i,1,l)} &= 2 \quad (If\ PV_{(i,l)} = UNSTF \quad \text{then}\ TA_{(l)} = POSHI) \\
\widetilde{RU}_{(i,2,l)} &= 1 \quad (If\ PV_{(i,l)} = POOR \quad \text{then}\ TA_{(l)} = POSLO) \\
\widetilde{RU}_{(i,3,l)} &= 1 \quad (If\ PV_{(i,l)} = MODRT \quad \text{then}\ TA_{(l)} = POSLO) \\
\widetilde{RU}_{(i,4,l)} &= 0 \quad (If\ PV_{(i,l)} = INSPC \quad \text{then}\ TA_{(l)} = NOCNG) \\
\widetilde{RU}_{(i,5,l)} &= -1 \quad (If\ PV_{(i,l)} = OVRSP \quad \text{then}\ TA_{(l)} = NEGLO)
\end{aligned} \tag{6.7}$$

and the pattern triggered by $I_{S_{s_i}} = 0$ is:

$$\widetilde{RU}_{(i,j,l)} = 0 \quad (If\ PV_{(i,l)} = INSPC \quad \text{then}\ TA_{(l)} = NOCNG) \tag{6.8}$$

6.3 Experimental Procedure and Results

6.3.1 Procedure

First the Servo expert level, as described in chapter 3.4 is applied to the simulated servo-motor described in section 3.3, and the controller attributes are preset to bring the system into oscillation. In addition an external load is presented. Next the servo expert level is executed in the learning mode. The system is preset and perturbated and the sensitivity index is calculated to trigger the appropriate rule pattern.

Then the fuzzy tuner algorithm is executed as described in Section 3.5, using the self-learning ruleset instead of the knowledge-based ruleset (see Section 3.5.1) and the same membership functions to calculate the decision table.

Finally the servo-expert algorithm is applied to the simulated, ill-define servo-motor in the usual way, using the self-learning ruleset-based decision table.

6.3.2 Results

The controller attributes were preset as follows:

$\odot^0_{(1)} = \omega_{cog} = 250[rad/s]$

$\odot^0_{(2)} = \phi_{cof} = 40[deg]$

$\odot^0_{(3)} = \psi_{cof} = 4[bit/count]$

$\odot^0_{(4)} = \omega l_{cog} = 0$

The system was then perturbated by the following changes:

$\delta \odot_{(1)} = 200[rad/s]$

$\delta \odot_{(2)} = -20[deg]$

$\delta \odot_{(3)} = -2[bit/count]$

$\delta \odot_{(4)} = 5[bit/count]$

$$\delta \odot_{(4)} = 5[bit/count]$$

Figure 6.2 shows the response of a typical preset (a) and perturbed ((b) through (e)) system.

(a) (b) (c) (d) (e)

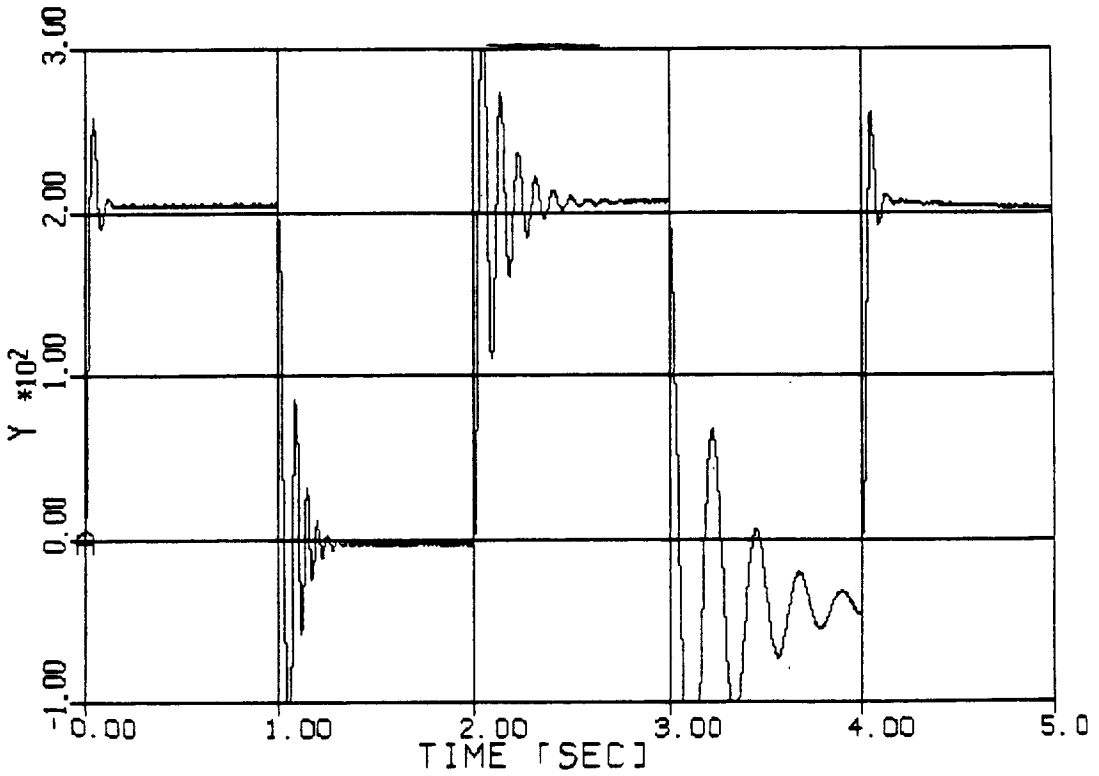


Figure 6.2: The Response of Preset and Perturbed Systems

(a) The response of the preset system
(b) Response of system perturbed by ϕ_{cof}
(c) Response of system perturbed by ω_{cog}
(d) Response of system perturbed by ψ_{cof}
(e) Response of system perturbed by ωl_{cof}

Typical self-generated ruleset given in Table 6.1, for a symmetrical pair of thresholds:

$TH_{(1)} = -0.25$

$TH_{(2)} = +0.25$

These thresholds play an important role in the ruleset generation. For example if they are zero or even too small, then no tuning action (NOCNG) will be presented, which will result in a very sensitive system. On the other hand, large thresholds might lead to deficient ruleset.

For convenience, the linguistic form of the self-generated ruleset is presented:

Table 6.1: Self Generated Ruleset

Rules for Condition Variable:OFFST

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	NEGHI	NOCHG	POSHI	POSHI
POOR	NEGLO	NOCHG	POSLO	POSLO
MODRAT	NEGLO	NOCHG	POSLO	POSLO
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	POSLO	NOCHG	NEGLO	NEGLO

Rules for Condition Variable:DMPRT

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	POSHI	NEGHI	POSHI	NOCHG
POOR	POSLO	NEGLO	POSLO	NOCHG
MODRAT	POSLO	NEGLO	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	NEGLO	POSLO	NEGLO	NOCHG

Rules for Condition Variable:RISTM

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	NEGHI	NOCHG	POSHI	NOCHG
POOR	NEGLO	NOCHG	POSLO	NOCHG
MODRAT	NEGLO	NOCHG	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	POSLO	NOCHG	NEGLO	NOCHG

Rules for Condition Variable:OVSHT

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	POSHI	NEGHI	POSHI	NOCHG
POOR	POSLO	NEGLO	POSLO	NOCHG
MODRAT	POSLO	NEGLO	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	NEGLO	POSLO	NEGLO	NOCHG

Rules for Condition Variable:DMPFR

	PHCOF	FRCOG	GNCOF	LFCOG
UNSATF	NEGHI	NOCHG	POSHI	NOCHG
POOR	NEGLO	NOCHG	POSLO	NOCHG
MODRAT	NEGLO	NOCHG	POSLO	NOCHG
IN_SPC	NOCHG	NOCHG	NOCHG	NOCHG
OVRSPC	POSLO	NOCHG	NEGLO	NOCHG

Comparing with the knowledge based ruleset in table 4.1 on page 94 we see that the rules relating the controller attributes to the rise time and to the average damped natural frequency are the same, while the rules relating some of the controller attributes to the remaining performance parameters have changed (for example, see the rules relates $PHCOF$ to $OFFST$)

A typical system response when the self-generated ruleset is applied to an initially slow and oscillatory system is demonstrated in Figure 6.3 and Figure 6.4

Comparing these figures with Figure 5.1 on page 98 and Figure 5.3 on page 100 even though the tuning process results in a different final controller setting and a different final response, all the performance parameters are in specification.

6.4 Summary

In this chapter a self-generated ruleset was presented. A preset and a perturbed system were introduced and the difference between their response was used to express sensitivity index. The response of an ill-defined system was demonstrated when the tuning actions were triggered using a decision table that was calculated based on a self generated-ruleset

.

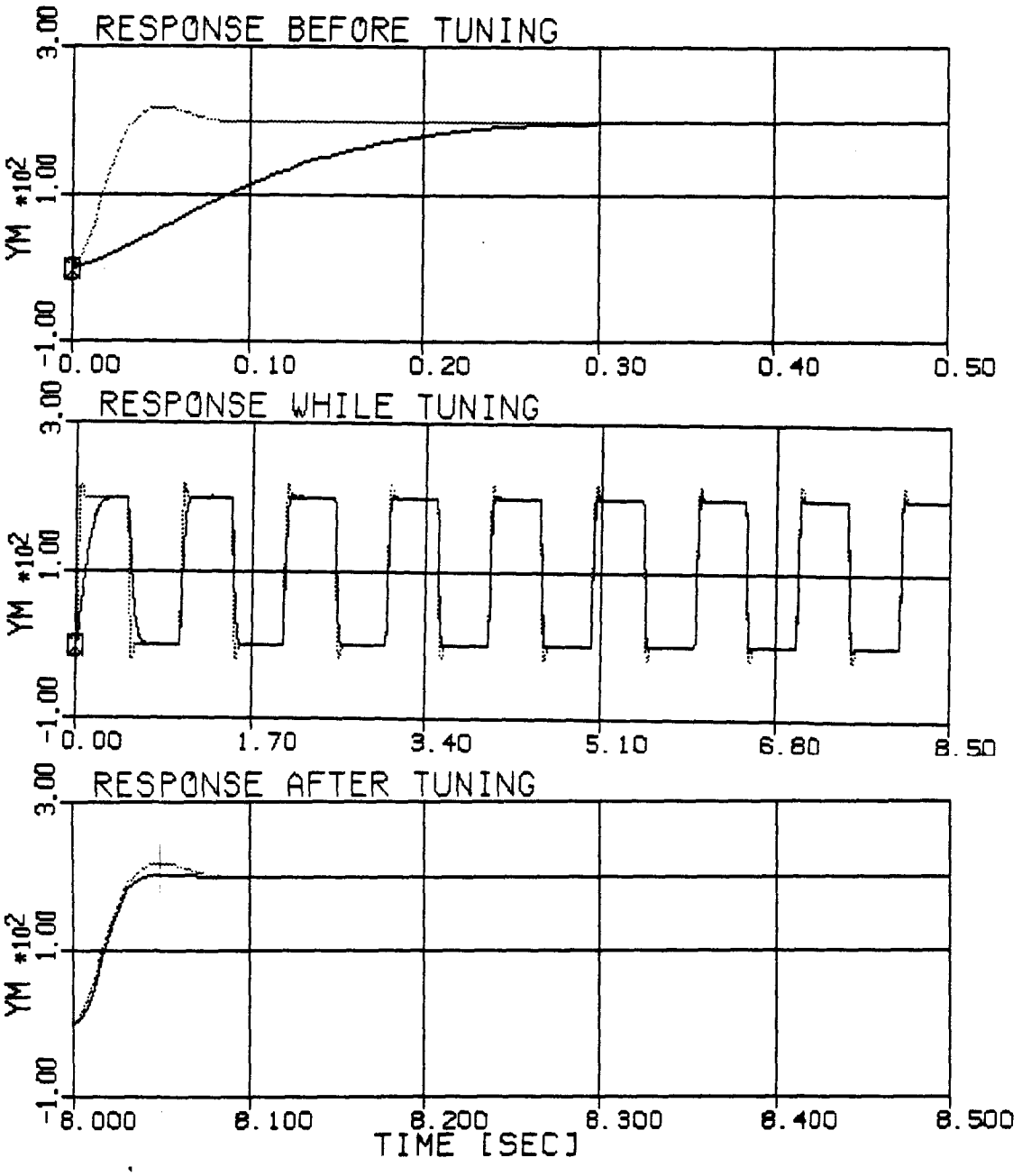


Figure 6.3: Tuning of a Slow Servo-Motor using a Self-Learning Ruleset

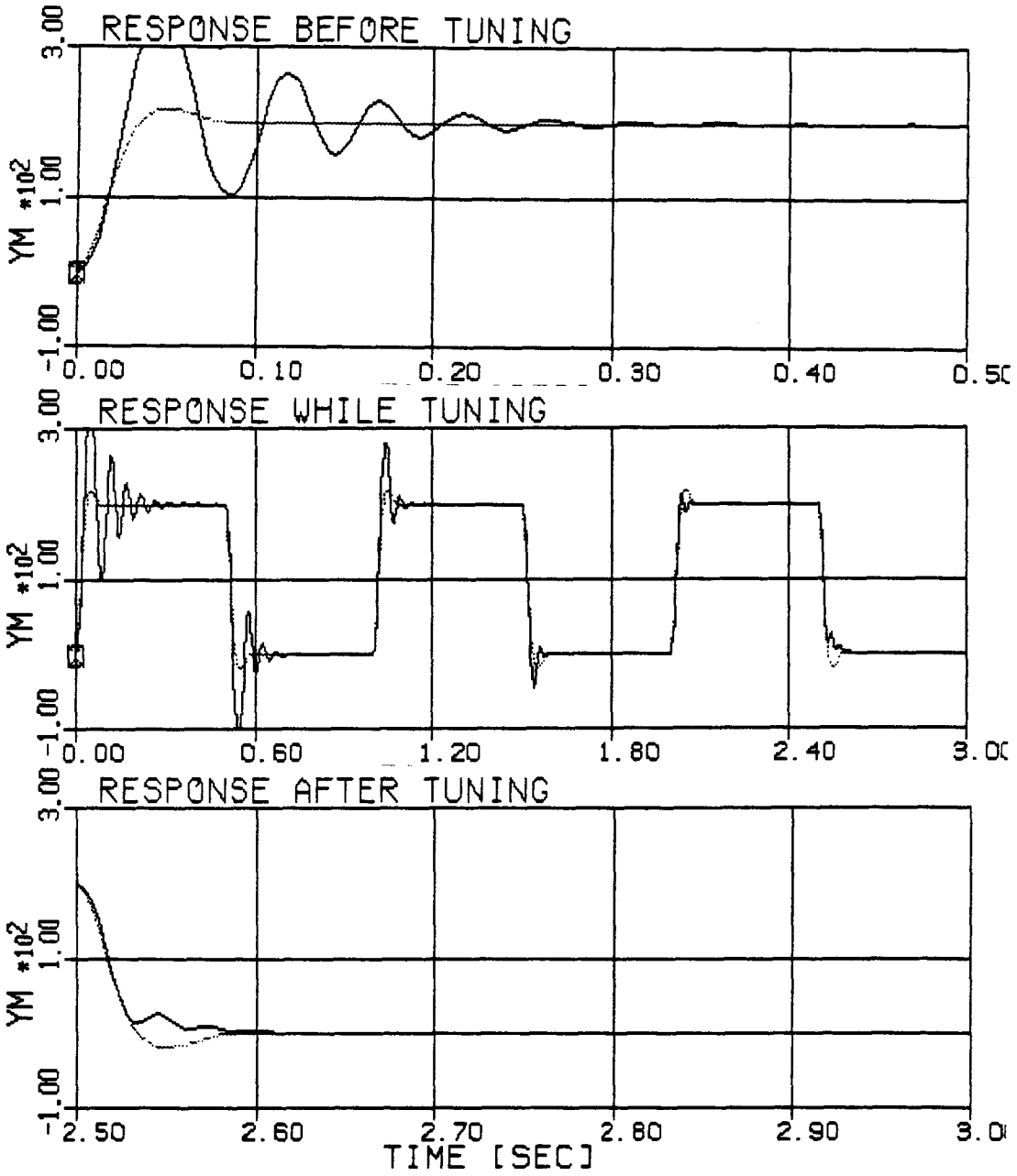


Figure 6.4: Tuning of an Oscillatory Servo using a Self-Learning Ruleset

Chapter 7

CONCLUSIONS and RECOMMENDATIONS

7.1 Introduction

In the privious chapters the background, the motivations and the approach to fuzzy tuning were introduced. The experimental system, that consists of three hierarchical levels, was developed and used to evaluate the performance of the tuner. In the preliminary phase, knowledge on tuning actions was gained to simulate the exists knowledge of a human operator. This knowledge, expressed in terms of fuzzy rules of the form If...Then, was used to calculate dthe ecision table by applying fuzzy logic operations on the membership functions of the condition and action variables. Tuning actions were matched with the actual performance in this decision table.

Furthermore, self-learning can be used to improve the knowledge base of the tuner.

This concluding chapter outlines the accomplishments and the main contributions of this thesis, review the advantages and limitations of the fuzzy tuning approach, and suggests some directions for future work.

7.2 Summary of Accomplishments

7.2.1 The Development of the Experimental System

A three level hierarchical system was developed:

- Simulated servo-motor in the lower level.

- Servo-expert algorithm, to evaluate the response and update the controller, was developed in the intermediate level.

- Knowledge based tuner, in the highest level, was developed to generate a decision table using fuzzy logic operations.

This hierarchical structure combined the advantages of both hard algorithmic control and the knowledge-based soft tuning. In particular, both the simulated hard controller and the physical one are used for lineraization and decoupling the nonlinear and coupled dynamics of the high bandwidth servo in the low level. In the top level, fuzzy tuner takes the advantage of experienced human operator's knowledge to tune the parameters of the controller.

Servo expert in the intermediate level interfaces between the servo and the tuner. A test signal was applied simultaneously to both the servo and the reference model. Performance parameters were calculated and performance indicators were generated using the deviation of the actual from the desired performance parameters. In the same level, tuning actions were used to update the attributes of the controller by using standard control design techniques. These updated attributes were mapped into the parameters of the controller that are to be used in the lowest level. .

7.2.2 Rule Base for Tuning

Tuning knowledge was simulated, gained and expressed mathematically as a set of fuzzy rules. Triangular membership functions were assigned to the fuzzy variables. This knowledge is based either on the relationships between the tuning actions and the corresponding performance as observed by the operator, or by a self learning algorithm. Knowledge on tuning process replaces the knowledge on the mathematical model of the system that is used in the conventional tuning techniques.

7.2.3 Tuning the Simulated System

Recalling that the knowledge generation process was based on extremely simplified model, the simulated servo-motor system was able to be tuned successfully even when the system was spoiled far beyond the one that was used to generate the tuning knowledge toward a complicated system. This demonstrates the independence of the performance on the a priory knowledge of the mathematical model of the system.

7.2.4 Implementation on a Physical System

The tuner was implemented and demonstrated successfully on a physical commercially available servo-motor system. Also this demonstrate the effectiveness of this tuning approach in the practice environment where effects like friction, disturbances and limitation of real-time computation are present.

7.2.5 Self Learning

Simple self-generating ruleset algorithm was developed and demonstrated successfully. This may open doors to further investigations in order to make the system more intelligence.

7.3 Main Contributions of the Research

The implementation of the fuzzy tuner algorithm to a physical system is the main contribution of this work[10]. This demonstrates the feasibility and the potential to achieve the goal of automating the tuning actions that are taken by an experienced operator in order to conserve and spread this knowledge in a cost effective way.

Two features, firstly used in this research, can help in the fuzzy ruleset generating:

Using the controller attributes rather than its parameters as the action variables of the tuning process is found to be much more efficient and results in simpler **If..Then** rules especially when the controller structure is more complex than simple PID.

In addition, using a reference model for performance specification guarantees the feasibility of the specifications, at least with respect to the particular reference model. This will provide some buffer against unrealistic specification. Furthermore, response evaluation should be simplified if it is carried out in a "relative sense" with respect to the response of the reference model response and not in an "absolute sense" when tuning knowledge is gained [27].

Demonstration of a simple self-generated rule mechanism based on the already developed inference and tuning will open doors to further research.

7.4 Advantages and Limitations

The main advantage of the approach that was presented in this research over the conventional tuning techniques is the independence on the a priory knowledge of the mathematical model of the system. This was achieved by using the prior knowledge on the tuning procedure rather than knowledge of the mathematical model of the system. Even though other knowledge-based approachs may be applied (like neural network) to match performance with tuning actions, the fuzzy logic approach has the advantage that it utilizing directly the linguistic rules that are obtained by a human operator..

Some limitations of this approach need further investigation and research. Currantly, there is no mathematical method to deal with the stability problem. The design of the controller is carried out on in the time domain and there is no direct information on the stability margin (phase and gain).

In addition, there are too many degrees of freedom to play with. The selection of

the optimal cardinality (or resolution) of the fuzzy condition and action variables, the tuning sensitivity parameter, and the performance threshold to be chosen are currantly open questions.

The best way to assign the membership function grad to the system variable is another field that needs further investigation.

The tuning process of the controller in this research is based on a preliminary experimental phase and assumed (unknown) time-invariant system. Is this approach applicable in on-line, time varying system ?.

7.5 Future Developments

The logical next step of this study is the implementation of the tuning structure on servo-motor based machines, particularly in robot applications.

Learning and self-generated rules deserve further investigation in modifying the human tuning knowledge.

Study of an on-line adaptation mechanism for automated controller rather than a separate, experimental phase is also an interesting field.

Bibliography

[1] *Advanced Continous Simulation Language (ACSL)* Reference manual. Mithell and Gauthier Associates, Concord, Mass. 01742.

[2] Arzen, K.E.: *"Realization of Expert System based feedback control."* Ph.D. Thesis CODEN: LUTFD2/TFRT-1029. Department of automatic control, Lund Institue of Technology, Lund Sweden. ,1987.

[3] Assiian, S. and Mamdani, E.H.: *"An experiment in linguistic syntesis with a fuzzy logic controller."*, Int. J. Man-Mach. Stud., 7,pp.1-13, 1974.

[4] Astrom, K.J. and Wittenmark, B.: *Adaptive Control* Lund Institue of Technology, Lund Sweden.

[5] Astrom, K.J and Wittenmark,B.: *Ön the control of constant but unknown systems."* 5th IFAC World Congress. Paris.

[6] Astrom, K.J.,Anton, J. and Arzen, K.E.: *"Expert control"*. Automatica 22 no. 3:277-286, 1986.

[7] Astrom, K.J. and Wittenmark,B.: *"On self-tuning regulators."* Automatica 9: 185-199, 1973.

[8] Astrom, K.J. : *"Theory and applications of adaptive control - survey."* Automatica 19: 471-486, 1983.

[9] Astrom, K.J. : *"Adaptive feedback control"* Proc. IEEE 75: 185-217., 1987.

[10] Barlev, S. and de Silva,C.W.,:*Implementation and Evaluation of Fuzzy Tuner for an Ill-Defined Servo-Motor Systems* Proc. 3rd AMSE Northwest GSTC, 1992 (In Press).

[11] Barr, A. and Feigenbaum,E.A edc. : *the Handbook of Artificial Intellegence.* Los Altos, Calif.:William Kaufmann, 1982.

[12] Braae, M. and Rutherford, D.A. :*"Theoretical and Linguistic Aspects of the Fuzzy Logic Controller."* Automatica Vol 15 pp. 553-557, 1979.

[13] Brachman, R.J. and Smith, B.C Special issue on Knowledge Representation, SIGART 70, 1980.

[14] Carter,G.A., Rutherford,D.A. : *A herustic adaptive controller for a sinter plant.* IFAC Symp. on Automation in Mining. Met and Met Processing, Johannesburg, 1976.

[15] Clarke, D.W. and Gawthrop,P.J. :*" A Self-tuning controller."* IEEE Proc.122: 929-934 , 1975.

[16] Clarke, D.W.,Mohtady,C. andTuffs, P.S. : *"Generalized Predictive Control-Part I. The basic algorithm."* Automatica 23: 13,7-148, 1987.

[17] Clarke, D.W., Mohtady,C. and Tuffs,P.S. : *"Generalized Predictive Control-Part II. Extension and interpatation."* Automatica 23: 149-160 ,1987.

[18] Cohen, G.H. and Coon, G.A. : *"Theoretical consideration of retarded control."* Trans. ASME 15:827-834, 1953.

[19] Czogala,E. and Pedrycz, W. : *"Control Problems in Fuzzy Systems."* Fuzzy Sets and Systems 7 pp. 257-273. North-Holland Publishing Company, 1982.

[20] Czogala, E. and Pedrycz, W. : *On identificationin fuzzy systems and its applications in control problems*, Fuzzy Sets and Systems 6 73-83, 1981.

[21] Davis, R. and Lenat, D.B. : *"Knowledge-Based System in Artificial Intelligence."* McGraw-Hill,New York, 1982.

[22] de Silva, C.W and MacFaralen, A.G.J. : *Knowledge-based control approach for robotic manipulators* Int. J. Control 50,(1) 249-273, 1989.

[23] de Silva, C.W and MacFaralen, A.G.J. : *Knowledge-based control with application robots.* Lecture Notes in Control and Information Sciences, Springer-Verlag, Vol. 123 Berlin, 1989.

[24] de Silva, C.W. : *Control Sensors and Actuators*,Prentice-Hall, Inc, Englewood Cliffs,Nj., 1989.

[25] de Silva, C.W. and Van Winssen, J.C. : *"Least Squares Adaptive Control for Trajectory Following Robots."*,ASME Journal of Dynanic Systems, Measurement, and Control, 109(2),104-110 ,1987.

[26] de Silva, C.W. : *An Analytical Framework for Knowledge-Based Tuning of Servo Controllers."* International Journal of Engineering Applications of Artificial Intelligence, Vol. 4, No.3, pp.177-189, 1991.

[27] de Silva, C.W. and Barlev, S. :*Hardware Implementation and Evaluation of a Knowledge-Based Tuner for a Servo Motor.* Proc. IFAC Symposium on ACASP, Grenoble, France, 1992. (Invited Paper, In Press)

[28] De Keyser, R.M.C. and Van Cauwenberghe, A.R. : *"Extended prediction self-adaptive control."* Preprint 7th IFAC Symposium on Identification and System Parameters Estimation: pp.1255-1260. York, UK, 1985.

[29] Dubois,D. and Prade, H. : *Fuzzy Sets and Systems*, Academic Press, Orlando, 1980.

[30] Duda, R.O. and Shortliffe, E.H : *Expert systems research.* Science 220, 261-268, 1983.

[31] Francis, J.C. and Leitch, R.R.,: *"ARTIFACT: A Real-time Shell for Intelligent Feedback Control"*, Research and Development in Expert System, Bramer, M.A. edc, Cambridge University Press, 1985.

[32] *DMC-400-10 Series, User Manual*, GALIL Motion Control,inc. Palo Alto, CA.

[33] Gawthrop, P.J. : *"Continuous Time Self-Tuning."* Letchworth, England: Research Studies Press, 1986.

[34] Goff, K.W. : *"Artifical Intelegence in Process Control."*, Mechanical Engineering, ASME, 107(10),53-57, 1985.

[35] Gregory, P.C., ed. : *Proc Self Adaptive Flight Control* Symposium. Wright-Patterson Air Force Base, Oaio: Wright Air Development Center., 1959.

[36] Grimble, M.J. : *"Implicit and explicit LQG self-tuning controllers."* Paper 14.4/F-3, 9th IFAC World Congress. Budapest., 1984.

[37] Gupta,M.M.,ed. : *Adaptive Metods for Control System Design.* New York : IEEE Press, 1986.

[38] Hayes-Roth, F., Watermann, D., and Lenat, D. : *Building Expert Systems.* Reading. Mass.: Addison-Wesley, 1983.

[39] Hirai H., Asai K., Kitajima S. : *Fuzzy Automation and its application to learning control systems.*Mem. Fac. Engng. Osaka City Univ. 10,December, 1968.

[40] Hirota,K. : *"Extended fuzzy expressions of probabilistic sets."* in: M. Gupta, R. Ragade and R. Yager.edc Advances in Fuzzy Sets Theory and Applications(North Holland,Amsterdam, 1979.

[41] Holmblad, I.P. and Ostergaard, J.J. : *"Fuzzy logic control: Operator experience applied in automatic process control"* F.L.S Review, 45, 11-16 ,F.L. Smidth and Co.,Vigerslev Alle 77,DK-2500 Valby, Copenhagen, 1981.

[42] Kaufmann,A.M. : *Introduction to Fuzzy sets Theory.* Academic Press. New York, 1975.

[43] Kickert, W.J.M. and Van Nauta Lemke,H.R.: *Application of a fuzzy controller to a warm water plant."* Automatica 12,301-308, 1976.

[44] King, P.J., Mamdani,E.H. : *The application of fuzzy control system to industrial processes*, IFAC World Congress, MIT, Boston ,1975.

[45] Kloeden, P.E. : *Fuzzy Dynamical Systems* Fuzzy Sets and Systems 7 pp.275-296. North Holland Publishing Company, 1982.

[46] Kiszka, J.B., Gupta,M.M, Nikiforuk,P.N. : *"Some Properties of Control System"* Elsevier Science Publisher B.V. (North-Holland), 1985.

[47] Kiszka, J., Kochanska, M., Sliwinska,D. : *The Influence of Some Fuzzy Implication Operators on the Accuracy of a Fuzzy Model* 1985,Fuzzy Sets and Systems,15,16, 1985.

[48] Kornblugh, R.D. : *An Experimental Evaluation of Robotic Manipulator Dynamic Performance under Model Reference Adaptive Control*, S.M. Thesis, Dept. Mech. Engineering Massachusetts Institue of Technology, Cambridge, September, 1984.

[49] Landau, Y.D. : *Adaptive control-The model reference approach.* New York: Marcel Dekker, 1979.

[50] Larsen, P.M. : *Industrial application of fuzzy logic control".* Int.J. Man-Mach. stud12,pp.3-10, 1980.

[51] MacMillen, G.K. *Tuning and Control Loop Performance*, Research Triangle Park, N.C.: Instrumemt Society of America.

[52] Mamdani, E.H and Gaines, B.R. eds, : *Fuzzy Resoning and its Application*, Academic Press, London, 1981.

[53] Mamdani ,E.H. : *"Process control using fuzzy logic, designing for human-computer communication"* (Academic Press, London, 1983) pp.311-336

[54] McMillan, G.K. Tuning and Control Loop Performance. Research Triangle Park, N.C.:Instrument Society of America, 1983.

[55] Mamdani, E.H., Ostergaard, J.J. and Lembessis, E. : *"Use of fuzzy logic for implementing rule-based controllers for industrial process"* in Zimmerman, H.J., Zadeh, L.A. and Gains, B.R. (eds) : Fuzzy sets and decision analysis-Vol 20 (TIMS Studies in Management Science,North Holland, 1984.)

[56] Mamdani, E.H. : *Application of fuzzy algorithms for the control of a simple dynamic plant.*, Proc. IEEE 1585-1588, 1974.

[57] Mandic, N.J., Scharf, E.M., Mamdani, E.H. : *"Practical application of heuristic fuzzy rule-based controller to the dynamic control of a robot arm."* IEE Proceedings, Vol. 132, Pt. D. No. 4, 1985

[58] Mishkin, E. and Braun, L.: *Adaptive Control Systems.* New York: McGraw-Hill, 1961.

[59] Narendra, K.S. and Y.-H. Lin :*"Design of stable model reference adaptive controllers."* In Application of Adaptive Control, edc. New York: Academic Press, 1980.

[60] IEEE : *"Mini-issue on NASA's advanced control low program for F-8 DFBW aircraft."* IEEE Trans. Automat Contr. AC-22:752-806 Institue of Technology, Sweden, 1977.

[61] Osburn, P.V., Whitaker, and A.Kezer : *"New developments in design of model reference adaptive control systems."* ISA 29th Annual Metting, paper 61-39. New York: Institue of Aeronautical Sciences, 1961.

[62] Osregaard, J.J. : *Fuzzy logic Control of a Heat Exchanger Process.* Publ.No.7601,Elec. Power Engng. Dept. Tech. University of Denmark,DK 2800, Lyngby,January, 1976.

[63] Parks, P.C. : *"Lyopanov redesign of model reference adaptive control systems."* IEEE Trans. Automat Contr.AC-11: 362-367, 1966.

[64] Peterka, V. : *"Adaptive digital regulation of noisy systems."* Preprints 2nd IFAC Symposium on Identification and Process Parameters Estimation. Prague, 1970.

[65] Rich, E.: *Artificial Intelligence.* Mcgraw-Hill, New York, 1983.

[66] Procyk, T.J. and Mamdani, E.J. : *"A linguistic self-organising process controller."* ,Automatica,15,pp15-30, 1979.

[67] Scharf, E.M. and Mandic, N.J. : *Development of learning algorithms for applications in control of robot arms.",*SERC report GR/B 7940.0, 1984.

[68] Stein, G. : *"Adaptive flight control-A pragmatic view." In Application of Adaptive Control,* edc. K.S. Narendra R.V. Monopoly. New York: Academic Press., 1980.

[69] Tong, R.M. : *Some problems with the design and implemention of fuzzy controllers.* Report CUED/F-CAMS/TR 127, Cambridge Univ. Control Eng. Dept., 1976.

[70] Tong R.M. : *Some properties of fuzzy feedback systems.* IEEE Trans. Syst. Man. Cybrenet. 10 pp.327-330., 1980.

[71] Tong R.M., Beck, M.B. : *Fuzzy control of the activated sludge wastewater treatment process.* Automatica 16,659-701, 1980.

[72] Van Amerongen, J., Van Nauta Lemke, H., Van der Veen, J.C.T. : *"An autopilot for Ships Designed with Fuzzy Sets.",* Proc. Fifth IFAC/IFIP Int. Conf. on Digital Computer Appl. Process Control, 1977.

[73] Whatley, M.J. and Pott, D.C : *"Adaptive gain improves reactor control."* Hydrocarbon Processing. May: 75-78, 1984.

[74] Wieslander, J. and Wittenmark,B. : *"An approach to adaptive control using real time identification."* Automatica 7: 211-217, 1971.

[75] Yamazaki,T. and Mamdani,E.H. : *"On the performance of a rule-based self-organizing controller.",*Proceedings of IEEE confrence on application of adaptive and multivariable control, Hull,UK, 1982.

[76] Ydstie, B.E. : *"Extended horizon adaptive control."* Paper 14.4/E-4 9th IFAC World Congress. Budapest, 1984.

[77] Zadeh, L.A. : *"Fuzzy sets."*, Inform. Contr.,Vol.8,pp 338-353, 1965.

[78] Zadeh, L.A. : *"Outlline of a new approach to the analysis of complex systems and decision processes"*. IEEE Trans. Systems. Man and Cybernetics SMC-3:28-44., 1973.

[79] Ziegler, J.G. and Nichols,N.B.: *"Optimum setting for automatic controllers."* Tran. ASME 64:759-798, 1942.

Appendix A

The Physical Servo-Motor

A.1 General Specification of the Servo-Motor System

System Specifications

Feature	Units	Motor Type Motor-5-500	Motor-50-1000
Resolution	Count/rev	2000	4000
Maximum speed with standard 12V supply*	rpm	3000	1200
Continuous torque with standard 12V supply*	oz-in Nm	6 0.04	15 0.1
Controller sampling frequency			
400 series	ms	0.5	0.5
600 series	ms	1	1
Servo System Bandwidth	Hz	100	100
Dimension (motor & encoder)			
Length	inch	3"	4.5"
OD	inch	1.68"	2.125"
Shaft diameter	inch	0.1562"	0.25"
Shaft length	inch	0.562"	1.00"

System Requirements

—DOS 2.0 or higher on IBM* PC/XT/AT or compatible
—Hercules, CGA or EGA Graphics Adaptor
—512K Bytes memory
—DMC-400 or DMC-600 Series controller must be installed

*IBM is a registered trademark of International Business Machines

Ordering Information

Software

SDK-400 Software	Must be purchased with DMC-400 series controller
SDK-600 Software	Must be purchased with DMC-600 series controller
SDK-Demo Software	Demonstration Software. Does not require controller installation.

Complete Systems

SDK-400-1	DMC-400-10, ICB-930, MOTOR-5-500, PS-12, SDK-400 Software
SDK-400-2	DMC-400-10, ICB-930, MOTOR-50-1000, PS-12, SDK-400 Software

Examples of SDK Systems

DMC-620	DMC-430
ICB-960	ICB-933
MOTOR-5-500 (x2)	MOTOR-50-1000 (x3)
PS-12	PS-12
SDK-600 Software	SDK-400 Software

SDK software may be used with single or multi-axis versions of the DMC-400 or DMC-600 controllers. SDK software is programmable for either Galil hardware or other motors, encoders, and drivers. Consult factory for options.

A.2 Motor

A.2.1 Specifications of the 50/1000 DC Motor

Motor 50/1000 Specifications

Motor Parameter		English Units		Metric Units	
T_c	Continuous stall torque	oz-in	30	Nm	0.21
T_p	Peak torque	oz-in	205	Nm	1.45
K_t	Torque constant	oz-in/A	10.8	Nm/A	0.076
K_e	Back EMF constant	Volt/Krpm	7.9	Volt/(rad/s)	0.076
J_m	Motor moment of inertia	oz-in-s^2	$3.7 \cdot 10^{-3}$	kg-m^2	$2.6 \cdot 10^{5}$
R	Armature resistance	Ω	1.59		
L	Armature inductance	mH	2.5		
T_m	Electromechanical time constant	ms	7		
R_{th}	Thermal resistance	°C/W	7.7		
r	Encoder resolution	Degree	0.09	counts/rev	4000
V_s	Recommended supply	Volts	35		
ω_o	Maximum speed with V_s	rpm	3750		

Features

- 200 oz-in peak torque
- 3750 RPM maximum speed
- Includes 1000 pulses per revolution TTL-level incremental encoder
- Low cost

Operating Guidelines

For continuous operation, limit torque to 30 oz-in and limit current to 2.8 A.

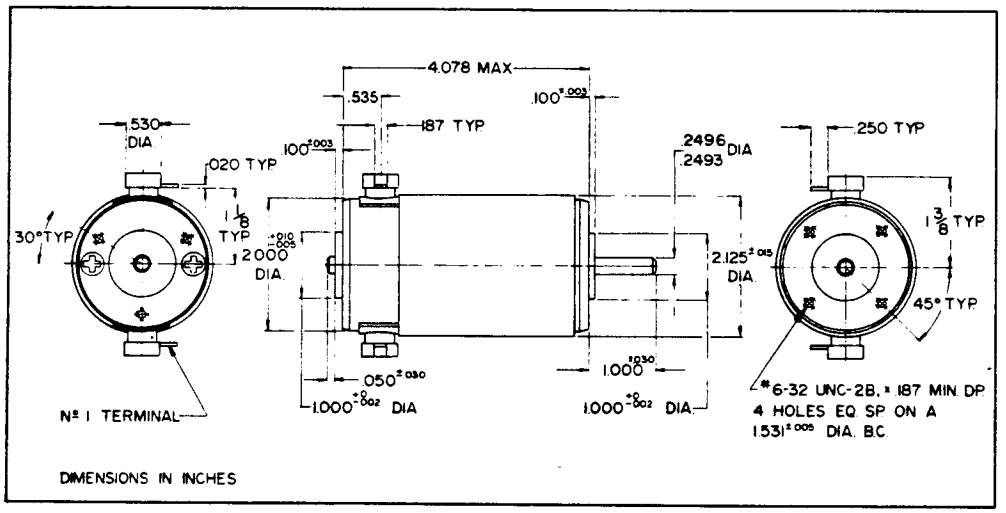
For intermittent operation, limit RMS current to
Irms <2.8 A

Encoder Pinout

1	Phase A	2	+5V
3	Ground	4	Ground
5	Ground	6	Ground
7	+5V	8	Phase B
9	+5V	10	Index

10 pin ribbon, mating connector Berg 65-692-001 or equivalent

A.2.2 Mechanical Drawing of the 50/1000 DC Motor



A.3 Specification of the DMC400 Controller

Features

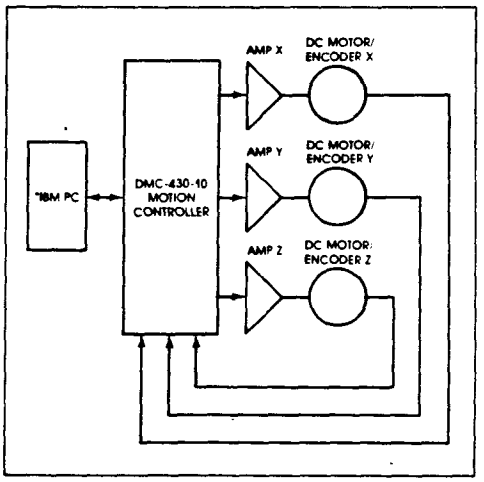
- IBM* PC/XT/AT compatible
- For servo motors with incremental encoder feedback
- Controls motion of up to 3 independent axes
- Position and velocity control
- Programmable velocity profiling
- Change position, velocity, acceleration "on-the-fly"
- Incremental position mode for continuous path
- Position "learn mode"
- 2 KHz sample and update
- 250,000 counts/sec maximum speed
- Programmable digital filter with gain, damping and integration—eliminates Tach
- ± Overtravel limits, home inputs, emergency stop
- Programmable torque and error limits
- PWM or Analog output
- PC communication software available

General Description

The DMC-400-10 Series are general purpose motion control cards for servo motors with encoder feedback. The DMC-400-10 controls one motor, the DMC-420-10 controls two motors, and the DMC-430-10 controls three motors. Each controller plugs into the IBM PC/XT/AT or compatible bus.

For each axis, the controller contains a microprocessor dedicated to the time-intensive motion control tasks. The controller functions include quadrature decoding of the encoder, generating the velocity profile and position trajectory, digital filtering of the control signal and generation of a ± 10 volt analog motor command signal. In addition, the controller provides overtravel, homing, emergency stop, and error handling functions.

The DMC-400-10 is programmable, accepting ASCII commands from the IBM PC host. Each separately addressable axis responds to over 40 instructions for specifying system parameters and motion profiles. Controller status and motor position can also be interrogated at any time.

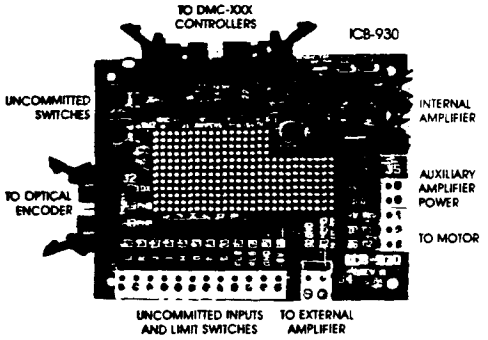


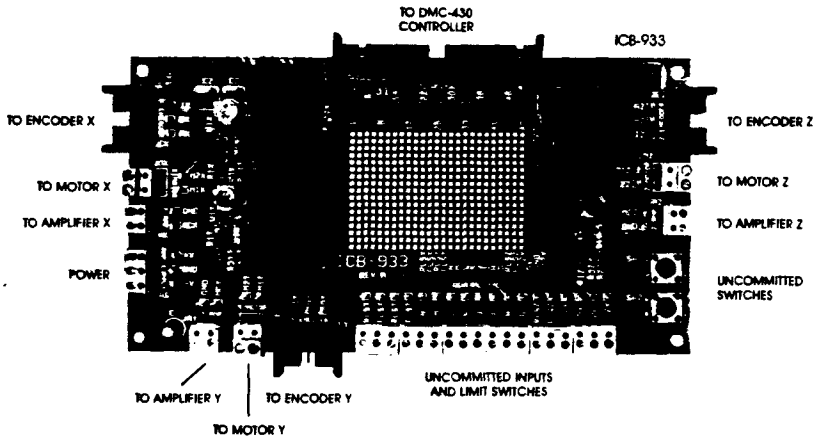
Three-Axis Motion Control System

A.4 Specification of the Interconnection Board

Features

- Interfaces directly to DMC-100, -200, -300, -400 controllers
- Convenient connection points for controller, encoder, motor and external amplifier
- Contains up to 3 on-board 20 Watt amplifiers
- Convenient test points and jumpers
- Eight uncommitted input points
- Two uncommitted pushbutton switches
- Breadboarding area





General Description

The ICB-930 is a compact circuit card that connects the DMC-100, DMC-200, DMC-300, DMC-400 controllers with other system elements such as motor, encoder, amplifier, power supply and external switches. The card includes ribbon conectors, test pins, two uncommitted pushbutton switches, screw type terminals and breadboarding area, allowing system components to be easily connected with maximum flexibility. The ICB-930 also contains a 1 Amp, 30 volt linear amplifier suitable for driving small motors.

The ICB-933 connects the DMC-420 and DMC-430 motion controllers with other system elements such as motors, encoders, amplifiers, power supply and external switches. The ICB-933 contains three 1 Amp, 30 volt linear amplifiers suitable for driving small motors.

*IBM PC is a registered trademark of International Business Machines Corp

On-Board Amplifiers

The ICB-930 contains a single 20 Watt linear amplifier and the ICB-933 contains three 20 Watt linear amplifiers with current feedback. An external supply of ± 10 to ± 35 Volts is required for the ICB-933. The external supply should be connected to the JAS connector and not to the ± 12V outputs. For the ICB-930, the ± 12 Volt source can come directly from the Galil Controller. Care should be taken to insure the average power dissipated across each amplifier is less than 20 Watts. The gain of each amplifier is .2 Amps/Volt.

Dimensions

ICB-933: 8.2" x 4", ⅛" diameter x 4 mounting holes at corners. Each mounting hole ¼" from sides.

ICB-930: 4.5" x 3.5", ⅛" diameter x 4 mounting holes at corners. Each mounting hole ¼" from sides.

A.5 Interface and Communication Software

A.5.1 Inerface to C

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>

char comm(in, val)
char far in[13];
long far *val;

{
    char res;
    int i, j, k, n, b, c;
    int t1, t;
    long t2, total, d;

    k = 0;
    n = 1006;
    t2 = 0;
    total = 0;
    d = 1;
    res = inp(n);
/*lab1: b = inp(n+1)/4;*/
/*    if (b/2 == b/2.0)*/
```

```
/*        goto lab1;*/
lab2: b = inp(n+1);
    c = b/2;
    if (c/2 == c/2.0)
        goto lab2;
    j = 12;
    for (i=0; i<j; i++) {
        if (in[i] != ' ') {
lab3:       b = inp(n+1);
            c = b/2;
            if (c/2 == c/2.0)
                goto lab3;
            res = in[i];
            outp(n, toascii(res));
        }
        else
            goto lab4;
    }
lab4: b = inp(n+1);
    c = b/2;
    if (c != b/2.0)
        goto lab4;
    t1 = inp(n);
    res = toascii(t1);
    if ((res != ':') && (res != '?')) {
        if ((t1 >= 65) && (t1 <= 70)) {
```

```
            t2 = (t1-55) * 1048576 / d;
            total = total + t2;
        }
        else if ((t1 >= 48) && (t1 <= 57)) {
            t2 = (t1-48) * 1048576 / d;
            total = total + t2;
        }
        k = k + 1;
        d = 16 * d;
        goto lab4;
    }
    if (total >= 8388608)
        *val = total - 16777216;
    else
        *val = total;
    return(res);
}
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>

int brk(dum)
int far dum;
{
    int l;
```

```
    l = kbhit();
    printf("the keboard is hit = %d\n", l);
    return(l);
}

\end{verbatin}
\newpage
\subsection{Inteface to FORTRAN}
\begin{verbatim}
      SUBROUTINE DMC
C                BLOCK TO BE MOVED AFTER TRANSLAT STAGE
c      @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
          interface to character function comm[c] (send,valu)
          CHARACTER*13,send [far, reference]
          INTEGER*4,valu  [far, reference]
          end
C     @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
      DIMENSION CNTRLD(5)
C%      CHARACTER comm
      CHARACTER*13,CC,PP
      CHARACTER po
      CHARACTER*7,parval
      CHARACTER*4,TP
      CHARACTER*2,GN,GI,PL,ZR,DB,PA
      CHARACTER*13,send
      INTEGER*4,valu
```

```
LOGICAL BG
LOGICAL EX
INTEGER   CNTRLD,YR0
INTEGER*4,dd,ee,pos


ENTRY DMCINIT
GN='GN'
GI='KI'
PL='PL'
ZR='ZR'
DB='DB'
TP='TP; '
PA='PA'
PP='PA200;'
CC =      'RS;                    '
po = comm(CC,dd)
CC =      'M0;                    '
po = comm(CC,dd)
CC =      'TL127;                 '
po = comm(CC,dd)
CC =      'OE1;                   '
po = comm(CC,dd)
CC =      'AC130000000;           '
po = comm(CC,dd)
CC =      'SP250000;              '
```

```
      po = comm(CC,dd)
      RETURN

      ENTRY DMCPRINT
      PRINT *,'              --- NOW IN DMC MODE ---'
      PRINT *,'              **********************'
      PRINT *,'       Type DMC command (Capital letters followed by ";")'
      PRINT *,'            or PA to change step amplitude                '
      PRINT *,'            or BG to start run                            '
      PRINT *,'            or EX to Exit without run                     '
      RETURN


      ENTRY DMCREAD(BG,EX,YR0)
      READ 10,CC
   10 FORMAT(13A)

      IF    (CC.EQ.'PA' )       THEN
         PRINT *,'STEP AMPLITUDE=?'
         READ  12,YR0
   12    FORMAT(I7)
         CALL DMCBARLEV (PA,YR0,PP)
         po = comm(PP,ee)
         print *,po,PP,dd
         RETURN
      ELSEIF(CC.EQ.'EX' )       THEN
```

```
   EX=.TRUE.
   RETURN
ELSEIF(CC.EQ.'BG' )        THEN
   BG=.TRUE.
   RETURN
ELSE
   po = comm(CC,dd)
   print *,po,CC,dd
ENDIF
   RETURN

ENTRY DMCSTART
CC='SH;                    '
po= comm(CC,dd)
CC='SV;                    '
po= comm(CC,dd)
CC='PR0;                   '
po= comm(CC,dd)
CC='BG;                    '
po= comm(CC,dd)
RETURN

ENTRY DMCP
CC='AB;                    '
po= comm (CC,dd)
CC= PP
```

```
      po= comm (CC,dd)
      CC='BG;                    '
      po= comm (CC,dd)
      RETURN


      ENTRY DMCM
      CC='AB;                    '
      po = comm (CC,dd)
      CC='PA0;                   '
      po = comm (CC,dd)
      CC='BG;                    '
      po = comm (CC,dd)
      RETURN
c
      ENTRY DMCTUNING  (CNTRLD)
C
      CALL DMCBARLEV (GN,CNTRLD(1),parval)
      po = comm(parval,ee)
c
      CALL DMCBARLEV (GI,CNTRLD(2),parval)
      po = comm(parval,ee)
c
      CALL DMCBARLEV (ZR,CNTRLD(3),parval)
      po = comm(parval,ee)
c
```

```
      CALL DMCBARLEV (PL,CNTRLD(4),parval)
      po = comm(parval,ee)
c
c
      CALL DMCBARLEV (DB,CNTRLD(5),parval)
      po = comm(parval,ee)
c
      return


      entry position(pos)
      po = comm(TP,pos)
c
      RETURN

      ENTRY DMCSTOP(t,rltm)
C     STOP MOTION AT END OF RUN

      CC='ST;                    '
      po = comm (CC,dd)
      CC='MO;                    '
      po = comm (CC,dd)
      CC='TP;                    '
      po = comm (CC,dd)
C
C      print *,'T = ',T,'RLTM = ',RLTM,'End position = ',dd
```

```
      RETURN
      END

      SUBROUTINE DMCBARLEV(PAR,VAL,parval)
c
      integer val
      character*2,par
      character digit1
      character digit2
      character digit3
      character*7,parval
      character bl
      character sc
      bl=' '
      sc=';'
c
      if (val.lt.100) go to 1299
      i3=int(val/100)
      call chract (i3,digit3)
c
      i3100=i3*100
      i2=int((val-i3100)/10)
      call chract (i2,digit2)
      i1=int(val-i3100-i2*10)
c
      call chract (i1,digit1)
```

```
      parval=par//digit3//digit2//digit1//sc//bl
C      print *,parval
      return
c
 1299 if (val.lt.10) go to 129
      i2=int(val/10)
      call chract (i2,digit2)
c
      i1=int(val-i2*10)
      call chract(i1,digit1)
      parval=par//digit2//digit1//sc//bl
C      print *,parval
      return
c
 129  i1=val
      call chract(i1,digit1)
      parval=par//digit1//sc//bl
C      print *,parval
c
      return
      end
c
c
c
      subroutine chract(ii,ch)
      character ch
```

```
c
      if (ii.eq.0) go to 10
      go to (1,2,3,4,5,6,7,8,9) ii
      return
1     ch='1'
      return
2     ch='2'
      return
3     ch='3'
      return
4     ch='4'
      return
5     ch='5'
      return
6     ch='6'
      return
7     ch='7'
      return
8     ch='8'
      return
9     ch='9'
      return
10    ch='0'
      return
      end
```

```
      SUBROUTINE TIM
      integer*2 ihr,imin,isec,i100th
c
      ENTRY INIT1TIME(CINT2)
      RETURN

      ENTRY INITTIME
C      II=-1
      isec=0
      i100th=0
C      RLTM=0.
      CALL SETTIM(ihr,imin,isec,i100th)
      RETURN
C
      ENTRY REALTIME(RLTM)
C      II=II+1
C      J=0
   1  CALL GETTIM(ihr,imin,isec,i100th)
C      J=J+1
      sec=isec
      th=i100th
      RLTM=SEC+TH/100.
C      CINTI=II*CINT
      IF (RLTM.LT.CINT2) go to 1
      return
      end
```

```
      FUNCTION COMM(CC,dd)
C%          FUNCTION COMMDUMMY(CC,dd)
      integer*4,dd
      CHARACTER*13,CC
      return
      end
```

Appendix B

Programs Listing

B.1 Main Program coded in ACSL

```
PROGRAM FUZZY
                    "TYPE DECLARATION"
                    "----------------"
      ARRAY     CNDQNT(5),DECTBL(5,5,5),CNTRLC(5),CNTRLZ(5),CNTRLD(5),...
                PIKO(20),PIKU(20),TPIKO(20),TPIKU(20),...
                CNGATR(5),ATRNET(5),PRFORM(5),...
                THOSC(3),THSPD(3),THOFS(3),THOVS(3),THDFR(3),...
                CNVAR(5),CNQNT(5),ACVAR(5),ACQNT(5)
      INTEGER   EY,EP,CNDQNT,CNTRLD,PIKO,PIKU,Y,YDMC,EDMCL,YM,YR,YRO,...
                K,LO,LU,IPHMAX,IFRMAX,IHFRGN,ILFRGN
      LOGICAL   DMC,BEGIN,EXIT,STP,LGD,DEMO
                    "CONTROLLER ATTRIBUTES INITIALIZATION"
                    "------------------------------------"
      CONSTANT PHMAXO =  57.29    $"PHase at Cross-Over frequency"
      CONSTANT FRMAXO = 314.      $"Cross-OVer Frequency"
      CONSTANT HFRGNO =   1.      $"HIgh Frequency Gain"
      CONSTANT LFRGNO =   0.      $"LO Frequency Gain"
                    "TUNER INITIALIZATION"
                    "--------------------"
      CONSTANT DPHMAX =   0.
      CONSTANT DFRMAX =   0
      CONSTANT DHFRGN =   0.
      CONSTANT DLFRGN =   0.
      CONSTANT THOFS = 0.1,0.2,0.3
      CONSTANT THOSC = 0.1,0.2,0.3
      CONSTANT THSPD = 0.1,0.2,0.3
      CONSTANT THOVS = 0.1,0.2,0.3
      CONSTANT THDFR = 0.1,0.2,0.3
      CONSTANT PIKMIN=0.04
                    "PROCESS PARAMETERS"
                    "------------------"
      CONSTANT Ktm= 0.076         $ "Torque constant      [Nm/A]"
      CONSTANT Ka = 0.2           $ "Amplifier gain       [Amp/Volt]"
      CONSTANT R  = 2.            $ "Armature resistence  [Ohm]"
      CONSTANT L  = 0.004         $ "Armature inductance  [H]"
      CONSTANT Jm = 2.6E-5        $ "Motor inertia        [Kg*m^2]"
      CONSTANT Tl = 0.            $ "External Load        [Nm]"
      CONSTANT Ks = 0.            $ "Spring constant      [Nm/rad]"
      CONSTANT Kd = 0.
      CONSTANT ENCDR = 636.62       $ "Encoder gain: 4N/2Pi=4*1000/2Pi"
      CONSTANT V2B = 0.078125     $"Volt to Bit gain"
                    "Reference signal"
```

```
                    "----------------"
      CONSTANT TSTEP = 0.05
      CONSTANT YR0   =100
                    "MODEL INITIALIZATION"
                    "--------------------"
      CONSTANT ZETAM  = 0.7       $"Model dumping"
      CONSTANT WNM    = 200       $"Model B.W "
      CONSTANT OFFSTM = 0.02
      CONSTANT ACCURM = 0.
      CONSTANT YMDLY0=0.1
      CONSTANT YMRIS0=0.95
                   "SIMULATION CONTROL"
                   "------------------"
      CONSTANT TF     = 0.25        $ "Teminated Time"
      CONSTANT DTSMP  = 0.0005      $ "# Samplin time"
      CONSTANT KSTEP=3
      CONSTANT CINT=0.005
      CONSTANT DLY=100
      CONSTANT DMC =.FALSE.
      CONSTANT LGD =.FALSE.
      CONSTANT DEMO=.FALSE.
      NSTEPS    NSTP   = 1           $ "# Of integration steps in CINT"
"INITIAL SECTION"
"***************"
INITIAL
      BEGIN  =.FALSE.
      EXIT   =.FALSE.
      STP    =.FALSE.
      ATRNET(1) = PHMAX0/57.29
      ATRNET(2) = FRMAX0
      ATRNET(3) = HFRGN0
      ATRNET(4) = LFRGN0
      CNGATR(1) = DPHMAX/57.29
      CNGATR(2) = DFRMAX
      CNGATR(3) = DHFRGN
      CNGATR(4) = DLFRGN
      Tcm   = R*Jm/Ktm/Ktm
      Tce   = L/R
      CINT2=2.*CINT
      YHI=YR0+DLY
      YLO=-DLY
"     CALL INIT1TIME(CINT2)"
      CALL INIT1EVAL (      =CINT,PIKMIN,YR0)
      CALL INITLINGU
      CALL DECISION (DECTBL=)
      CALL DESIGN    (CNTRLC=ATRNET)
      CALL ZTRANS    (CNTRLZ=DTSMP,CNTRLC)
                     "DMC-400 SERVO SYSTEM INITALIZATION"
                     "----------------------------------"
      CALL DMCINIT
      IF  (.NOT.DMC) GO TO L1
      CALL DMCPRINT
      L0..CALL DMCREAD   (=BEGIN,EXIT,YR0) $"INITIALIZION"
      IF(     EXIT ) GO TO L5       $"EXIT RUN"
      IF(.NOT.BEGIN) GO TO L0       $"CONTINUE RUN"
      CALL DMCCONTROL (CNTRLD=CNTRLZ)
      CALL DMCTUNING  (=CNTRLD)
      CALL DMCSTART
      L1..CONTINUE
      K=0 $  TMODL=0.$ YR=0 $ YDMC=0
      CALL INITPRNT(ATRNET)
```

```
L100..CONTINUE  $ "Step loop initialization"
      IF(DMC)EDMCL=YR-YDMC
      EY=0       $ Ep= 0      $ U=0.      $ U1p=0.    $ U2=0.       $ U2p=0.
      YM=YR      $ Y=YR       $ TM=0.
      WD=0.      $ YMI=0.     $ RISINM=0.$ RISINS=0.$OFSINS=0.
      LO=1       $ LU=1       $ OSCDMP=0.$ PIOWDM = 0.
      DMPFRS=0. $ DMPRTS=0. $ STLTMS=0.$ OVSHTS=0.
      TLO=TL*(-1)**K
      YR0R2=YR0/ENCDR/2
      DO L50 JJ=1,20
      PIKO(JJ)=0
      PIKU(JJ)=0
      TPIKO(JJ)=0.
      TPIKU(JJ)=0.
L50.. CONTINUE
      CALL MODELPARMT(DMPFRM,DMPRTM,STLTMM,OVSHTM,PIOWDM,TMDLY,TMRIS,...
                      YMDLY0,YMRIS0 = YR0,YR,WNM,ZETAM,CINT,PIKMIN,TSTEP)
      CALL INIT2EVAL(=K,YR,PIOWDM)
      K=K+1
            "Reference signal"
            "----------------"
          Y0=YR/ENCDR
          YR=-YR0*((-1)**K-1)/2
      IF  (.NOT.DMC) GO TO L15
          IF (YR.EQ.0) GO TO LZ
                       GO TO LP
          LZ..CALL DMCM
                       GO TO L15
          LP..CALL DMCP
    L15..CONTINUE
END  $ "OF INITIAL"
DYNAMIC $ "DYNAMIC SECTION"
"-------------------------"
      TIME=T+(K-1)*TSTEP
                           "PROCESS DYNAMICS"
                           "----------------"
      IF(DMC) GO TO L13
"        IF (TIME.EQ.0)  CALL INITTIME(=CINT)"
"        CALL REALTIME(REALTM=)"
                            "SIMULATED SERVO"
                            "---------------"
DERIVATIVE
      CINTERVAL CINT   = 0.005       $ "Communication time interval"
                      "PROCESS DYNAMIC"
                      "---------------"
        M      = Ka*U+TlO+TLSP+TLDP   $ "Total load          [Nm]"
        DYY    = Ktm*INTEG(M/Jm,0.)   $ "Motor speed         [rad/sec]"
        TLDP   = -Kd*DYY
        YY     = INTEG(DYY,Y0)        $ "Motor position      [rad]"
        TLSP   = -Ks*(YY-YR0R2)
"       DYY    = REALPL(TCM,M)"
"       YYE    = REALPL(TCE,DYY)/Ktm"
"        YY     = INTEG(YYE,Y0)"          $ "Motor position      [rad]"
                      "ENCODER DYNAMIC"
                      "---------------"
        Y      = INT(ENCDR*YY)
END $ "Of derivative"
        CONSTANT NOISE=1.
```

```
        YN = Y+NOISE*GAUSS(0.0,1)
                    "DISCRETE CONTROLLER DYNAMIC"
                    "-------------------------*"
DISCRETE CNTRLR
PROCEDURAL
      INTERVAL  DTSAMP = 0.0005       $ "Sampling time"
      EY     = YR-Y                    $"Error [counts]"
      U1     = CNTRLZ(4)*U1p+CNTRLZ(1)*EY-CNTRLZ(1)*CNTRLZ(3)*Ep
      IF(ABS(U1).LT.1.E-10) U1=0.
      U2     = U2p+EY*DTSMP            $"Integral part"
      U      = BOUND(-10.,10.,V2B*QNTZR(1.,U1+CNTRLZ(2)*U2))...
                                       $"Command [Volt]"
      Ep     = EY                      $"E time shifting"
      U1p    = U1                      $"U1 time shifting"
      U2p    = U2                      $"U time shifting"
END
END $"Of discrete controller"
           GO TO L16
L13..      CALL POSITION(YDMC=)
           Y=YDMC+EDMCL
L16..      CALL MODEL   (YM=T)
"           CALL LOGD(LGD)"
           CALL PIKDETEC (LO,LU,PIKO,PIKU,TPIKO,TPIKU,STP=T,Y,TMRIS)
           CALL INTEGRAL (RISINM,RISINS,OFSINS=T,Y,YM,TMDLY,TMRIS)
      TERMT(T.GT.TSTEP)                $ "Teminated Time"
      IF(STP) GO TO L5
END $"OF DYNAMIC"
TERMINAL
      CALL PEAKCHECK(LO,LU)
      CALL SDMPFR  (DMPFRS=LO,LU,TPIKO,TPIKU)
      CALL SDMPRT  (DMPRTS=LO,LU,PIKO,PIKU,TPIKO,TPIKU)
      CALL SSTLTM  (STLTMS=LO,LU,TPIKO,TPIKU)
      CALL SOVSHT  (OVSHTS=PIKO,YR,YRO)
      YROT=(TSTEP-2.*PIOWDM)*YRO
      OFSINM=YROT*OFFSTM
      OFSINS=ABS(ABS(OFSINS)-YROT)
      RISINS=ABS(RISINS)
      RISINM=ABS(RISINM)
      CALL  FOFFST(CNDQNT,PRFORM=OFSINS,OFSINM,THOFS)
      CALL  FDMPRT(CNDQNT,PRFORM=DMPRTS,DMPRTM,THOSC)
      CALL  FRISIN(CNDQNT,PRFORM=RISINS,RISINM,THSPD)
      CALL  FDMPFR(CNDQNT,PRFORM=DMPFRS,DMPFRM,THDFR)
      CALL  FOVSHT(CNDQNT,PRFORM=OVSHTS,OVSHTM,THOVS)
      IF(CNDQNT(1).GE.4.AND.CNDQNT(2).GE.4.AND.CNDQNT(3).GE.4.AND....
         CNDQNT(4).GE.4.AND.CNDQNT(5).GE.4.AND..NOT.DEMO)   GO TO L5
      CALL TUNEPHMAX  (IPHMAX,ATRNET=CNDQNT,CNGATR)
      CALL TUNEFRMAX  (IFRMAX,ATRNET=CNDQNT,CNGATR)
      CALL TUNEHIFGN  (IHFRGN,ATRNET=CNDQNT,CNGATR)
      CALL TUNELOFGN  (ILFRGN,ATRNET=CNDQNT,CNGATR)
      CALL PRNT(CNDQNT,PRFORM,DECTBL,ATRNET...
                ,IPHMAX,IFRMAX,IHFRGN,ILFRGN,K)
       CALL DESIGN     (CNTRLC=ATRNET)
       CALL ZTRANS     (CNTRLZ=DTSMP,CNTRLC)
       IF(.NOT.DMC) GO TO L14
```

```
      CALL DMCCONTROL (CNTRLD=CNTRLZ)
      CALL DMCTUNING  (=CNTRLD)
 L14..CONTINUE
      IF (K.LT.KSTEP)GO TO L100
  L5..CONTINUE
      CALL FINALPRNT(CNDQNT)
      IF (DMC) CALL DMCSTOP(=T,REALTM)
      LSTSTP = K*TSTEP-TSTEP
      TF     = K*TSTEP
END $ "OF TERMINAL"
END $"OF PROGRAM"
```

B.2 Servo-Expert sub-programs listing

B.2.1 Sub-Program MODEL

```
      SUBROUTINE MODEL1
      INTEGER YR0,YR,YM,YMDR0,YMDR1
      LOGICAL YMDLY,YMRIS

      ENTRY MODELPARMT(YR0,YR,WNM,ZETAM,CINT,PIKMIN,TSTEP,
     #                 DMPFRM,DMPRTM,STLTMM,OVSHTM,PIOWD,TMDLY,TMRIS,
     #                                                 YMDLY0,YMRIS0)
      PI=3.1416
      TMRIS=0.
      TMDLY=0.
      DMPRTM=0.
      YMDR1=0.
      TM=0.
      IF(YR.NE.YR0)YMDR1=YR0
      YMDLY=.FALSE.
      YMRIS=.FALSE.
      YMDLAY=YMDLY0*YR0
      YMRISE=YMRIS0*YR0
      SQTDMP=SQRT(1.-ZETAM*ZETAM)
      PHI=ACOS(ZETAM)
      ZETAWN=ZETAM*WNM

      DMPFRM=WNM*SQTDMP
      OVSHTM=EXP(-PI*ZETAM/SQTDMP)
      PIOWD=PI/DMPFRM
      YMP2=1.-EXP(-ZETAWN   *PIOWD)*SIN(DMPFRM   *PIOWD+PHI)/SQTDMP
      YMP1=1.-EXP(-ZETAWN*2.*PIOWD)*SIN(DMPFRM*2.*PIOWD+PHI)/SQTDMP
      TJ=3.
    1 CONTINUE
      YMP0=1.-EXP(-ZETAWN*TJ*PIOWD)*SIN(DMPFRM*TJ*PIOWD+PHI)/SQTDMP
      DMPRTM=DMPRTM+ABS((YMP0-YMP1)/(YMP1-YMP2))
      IF(ABS((YMP0-YMP1)).LT.PIKMIN) GO TO 2
      YMP2=YMP1
      YMP1=YMP0
      TJ=TJ+1.
      GO TO 1
```

```
    2 CONTINUE
      STLTMM=(TJ-1.)*PIOWD
      DMPRTM=DMPRTM/(TJ-2.)

   3  YM=YR0*EXP(-ZETAWN*TM)/SQTDMP*SIN(DMPFRM*TM+PHI)
      IF(YR.EQ.YR0) YM=YR0-YM
      YMDR0=YR0-ABS(YR-YM)
      IF(YMDR0.GE.YMDLAY)THEN
        IF(.NOT.YMDLY)THEN
          TMDLY=TM
          YMDLY=.TRUE.
         ELSEIF(YMDR0.GE.YMRISE)THEN
          IF(.NOT.YMRIS)THEN
            TMRIS=TM
            YMRIS=.TRUE.
          ENDIF
         ENDIF
      ENDIF
      TM=TM+CINT
      IF (TM.LE.TSTEP) GO TO 3
      RETURN

      ENTRY MODEL(T,YM)
      YM=YR0*EXP(-ZETAWN*T)/SQTDMP*SIN(DMPFRM*T+PHI)
      IF(YR.EQ.YR0) YM=YR0-YM
      RETURN
      END
```

B.2.2 Sub-Program PREPROCESS

```
      SUBROUTINE PREPROCESS
      DIMENSION PIK0(20),PIKU(20),TPIK0(20),TPIKU(20)
      LOGICAL STP
      INTEGER Y,YM,YR,YR0,Y1,Y2,DY1,DY2,DYYY,DYYYM1,L0,LU,
     # PIK0,PIKU
C


c                  PEAKS DETECTION
c                  ---------------
      ENTRY PIKDETEC(T,Y,TMRIS,L0,LU,PIK0,PIKU,TPIK0,TPIKU,STP)
```

```
      IF(T.LE.TMRIS) GO TO 10
      DY1=Y-Y1
      DY2=Y1-Y2
      IF (DY1*DY2)20,15,10
   15 IF (DY1.EQ.0.AND.DY2.EQ.0) GO TO 10

      IF(DY1.NE.0)THEN
      DYYY=DY1
      ELSE
      DYYY=DY2
      ENDIF

      IF(DYYY*DYYYM1.LT.0)GO TO 21
      DYYYM1=DYYY
      GO TO 10
   20 CONTINUE
      DYYY=DY1
   21 PEAKM1=PEAKS
      PEAKS=FLOAT(Y2-YR)/FLOAT(YR0)
      DYYYM1=DY1

      IF (ABS(PEAKS-PEAKM1).LT.PIKMIN) GO TO 10
      IF(DYYY.GT.0)GO TO 25
      IF(YR.EQ.YR0)THEN
      LO=LO+1
      PIKO(LO)=Y1
      TPIKO(LO)=T
      ELSE
      LU=LU+1
      PIKU(LU)=Y1
      TPIKU(LU)=T
      ENDIF
      GO TO 10

   25 IF (ABS(1.-PEAKS/PEAKM1).LT.PIKMIN) GO TO 10
      IF(YR.EQ.YR0)THEN
      LU=LU+1
      PIKU(LU)=Y1
      TPIKU(LU)=T
      ELSE
```

```
      LO=LO+1
      PIKO(LO)=Y1
      TPIKO(LO)=T
      ENDIF
   10 CONTINUE

      IF(LO.GT.20.OR.LU.GT.20)THEN
          PRINT *,'LO>20 OR LU>20 !!',LO,LU
          STP=.TRUE.
      ENDIF
      Y2   = Y1
      Y1   = Y
      RETURN

      ENTRY INTEGRAL (T,Y,YM,TMDLY,TMRIS,RISINM,RISINS,OFSINS)
      YRMYRO=YR-YRO
      YSN=FLOAT(Y +YRMYRO)
      YMN=FLOAT(YM+YRMYRO)
            IF(T.LT.TMRIS)THEN
                  RISINS=YSN
                  RISINM=YMN
             ELSEIF(T.GT.PI2OWM)THEN
                  OFSINS=OFSINS+(YSN+YSO)*CINT2
             ENDIF
      YSO=YSN
      YMO=YMN
      RETURN
      END
```

B.2.3 Sub-Program EVALUATION

```
      SUBROUTINE EVALUATION
      DIMENSION PIKO(20),PIKU(20),TPIKO(20),TPIKU(20)
      LOGICAL STP
      INTEGER Y,YM,YR,YRO,Y1,Y2,DY1,DY2,DYYY,DYYYM1,LO,LU,
     # PIKO,PIKU
C
      ENTRY INIT1EVAL(CINT,PIKMIN,YRO)
      STP=.FALSE.
      CINT2=CINT/2.
```

```
      RETURN
C
      ENTRY INIT2EVAL(K,YR,PIOWDM)
      Y1   = YR
      Y2   = YR
      DYYY  = -YR0*(-1)**K
      DYYYM1= (-1)**K
      LO    = 1
      LU    = 1
      PEAKS = 1.
      MM=0
      YSO=0
      YMO=0
      RISINS=0.
      RISINM=0.
      OFSINS=0.
      PI2OWM=2.*PIOWDM
      RETURN

      ENTRY PEAKCHECK(LO,LU)

   1  IF(LU.GT.LO)THEN
      LU=LU-1
      GO TO 1
      ENDIF

   2  IF((LO-LU).GT.1)THEN
      LO=LO-1
      GO TO 2
      ENDIF
      RETURN

      ENTRY SDMPFR(LO,LU,TPIKO,TPIKU,DMPFRS)
      IF(LU.EQ.1) THEN
      DMPFRS=0.
      RETURN
      ELSEIF(LU.EQ.2.AND.LO.EQ.2) THEN
      DMPFRS=3.14/(TPIKU(2)-TPIKO(2))
      RETURN
      ENDIF
```

```
      IF(LO.EQ.(LU+1))THEN
                DMPFRS=6.28*(LO-2)/(TPIKO(LO)-TPIKO(2))
      ELSE
                DMPFRS=3.14*(2*LO-3)/(TPIKU(LU)-TPIKO(2))
      ENDIF
      RETURN

      ENTRY  SDMPRT(LO,LU,PIKO,PIKU,TPIKO,TPIKU,DMPRTS)
      IF (LO.LT.3)THEN
      DMPRTS=0.
      RETURN
      ENDIF
      JJJ=3
      N=0
      PIKD=FLOAT(PIKU(2)-PIKO(2))
40    CONTINUE
      IF (JJJ.GT.LO) GO TO 41
      JJ1=JJJ-1
      PIKN=FLOAT(PIKO(JJJ)-PIKU(JJ1))
      IF(PIKD.EQ.0.)PRINT*,'PIKD=0.!!!'
      DMPRTS=DMPRTS+ABS(PIKN/PIKD)
      N=N+1

      IF (JJJ.GT.LU) GO TO 41
      PIKD=PIKN
      PIKN=FLOAT(PIKU(JJJ)-PIKO(JJJ))
      IF(PIKD.EQ.0.)PRINT*,'PIKD=0.!!!'
      DMPRTS=DMPRTS+ABS(PIKN/PIKD)
      N=N+1

      PIKD=PIKN
      JJJ=JJJ+1
      GO TO 40
41    IF(N.EQ.0)PRINT *,'N=0!!!'
      DMPRTS=DMPRTS/N

      ENTRY SSTLTM(LO,LU,TPIKO,TPIKU,STLTMS)
      IF (LU.LT.2)THEN
      STLTMS=0.
      RETURN
      ELSE
```

```
      IF (LO.EQ.LU) THEN
      STLTMS=TPIKU(LU)
      RETURN
      ELSE
      IF (LO.EQ.(LU+1)) STLTMS=TPIKO(LO)
      ENDIF
      ENDIF
      RETURN


      ENTRY SOVSHT(PIKO,YR,YRO,OVSHTS)
      IF(PIKO(2).NE.0) OVSHTS=ABS(FLOAT(PIKO(2)-YR))/YRO
      RETURN
      END
```

B.2.4 Sub-Program CLASSIFICATION

```
      SUBROUTINE CLASSIFICATION
      DIMENSION THSPD(3),THOSC(3),THOFS(3),THOVS(3),THDFR(3)
     #,PIKO(20),PIKU(20),TPIKO(20),TPIKU(20),PRFORM(5)
      INTEGER CNDQNT(5),PIKO,PIKU,Y,YRO,YR

      ENTRY  FOFFST(OFSINS,OFSINM,THOFS,CNDQNT,PRFORM)
      PRFORM(1) = -(OFSINM - OFSINS)/OFSINM
      IF(PRFORM(1).LE.0.                                  ) GO TO 15
      IF(PRFORM(1).LE.THOFS(1)                            ) GO TO 14
      IF(PRFORM(1).LE.THOFS(2).AND.PRFORM(1).GT.THOFS(1)) GO TO 13
      IF(PRFORM(1).LE.THOFS(3).AND.PRFORM(1).GT.THOFS(2)) GO TO 12
      IF(PRFORM(1).GT.THOFS(3)                            ) GO TO 11
   15 CNDQNT(1)=5
      RETURN
   14 CNDQNT(1)=4
      RETURN
   13 CNDQNT(1)=3
      RETURN
   12 CNDQNT(1)=2
      RETURN
   11 CNDQNT(1)=1
      RETURN
```

```
      ENTRY FDMPRT(DMPRTS,DMPRTM,THOSC,CNDQNT,PRFORM)
      PRFORM(2) =-(DMPRTM - DMPRTS)
      IF (DMPRTS.EQ.0.) GO TO 24
      IF(                    PRFORM(2).LT.0.      ) GO TO 25
      IF(PRFORM(2).GE.0.       AND.PRFORM(2).LE.THOSC(1)) GO TO 24
      IF(PRFORM(2).GE.THOSC(1).AND.PRFORM(2).LE.THOSC(2)) GO TO 23
      IF(PRFORM(2).GE.THOSC(2).AND.PRFORM(2).LE.THOSC(3)) GO TO 22
      IF(PRFORM(2).GE.THOSC(3)                          ) GO TO 21
   25 CNDQNT(2)=5
      RETURN
   24 CNDQNT(2)=4
      RETURN
   23 CNDQNT(2)=3
      RETURN
   22 CNDQNT(2)=2
      RETURN
   21 CNDQNT(2)=1
      RETURN

      ENTRY  FRISIN(RISINS,RISINM,THSPD,CNDQNT,PRFORM)
      PRFORM(3) = (RISINM - RISINS)/RISINM
      ARISIN=ABS(PRFORM(3))
      IF( PRFORM(3).LT.0.                               ) GO TO 35
      IF(ARISIN.LE.THSPD(1)                           ) GO TO 34
      IF(ARISIN.LE.THSPD(2).AND.ARISIN.GE.THSPD(1)) GO TO 33
      IF(ARISIN.LE.THSPD(3).AND.ARISIN.GE.THSPD(2)) GO TO 32
      IF(ARISIN.GE.THSPD(3)                           ) GO TO 31
   35 CNDQNT(3)=5
      RETURN
   34 CNDQNT(3)=4
      RETURN
   33 CNDQNT(3)=3
      RETURN
   32 CNDQNT(3)=2
      RETURN
   31 CNDQNT(3)=1
      RETURN


      ENTRY  FOVSHT(OVSHTS,OVSHTM,THOVS,CNDQNT,PRFORM)
      PRFORM(4) = -(OVSHTM - OVSHTS)/OVSHTM
```

```
      IF(OVSHTS.EQ.0) GO TO 45
      IF(PRFORM(4).LE.0.                                ) GO TO 45
      IF(PRFORM(4).LE.THOVS(1)                          ) GO TO 44
      IF(PRFORM(4).LE.THOVS(2).AND.PRFORM(4).GE.THOVS(1)) GO TO 43
      IF(PRFORM(4).LE.THOVS(3).AND.PRFORM(4).GE.THOVS(2)) GO TO 42
      IF(PRFORM(4).GE.THOVS(3)                          ) GO TO 41
   45 CNDQNT(4)=5
      RETURN
   44 CNDQNT(4)=4
      RETURN
   43 CNDQNT(4)=3
      RETURN
   42 CNDQNT(4)=2
      RETURN
   41 CNDQNT(4)=1
      RETURN

      ENTRY  FDMPFR(DMPFRS,DMPFRM,THDFR,CNDQNT,PRFORM)
      PRFORM(5) = (DMPFRM - DMPFRS)/DMPFRM
      IF(DMPFRS.EQ.0) GO TO 55
      IF(PRFORM(5).LE.0.                                ) GO TO 55
      IF(PRFORM(5).LE.THDFR(1)                          ) GO TO 54
      IF(PRFORM(5).LE.THDFR(2).AND.PRFORM(5).GE.THDFR(1)) GO TO 53
      IF(PRFORM(5).LE.THDFR(3).AND.PRFORM(5).GE.THDFR(2)) GO TO 52
      IF(PRFORM(5).GT.THDFR(3)                          ) GO TO 51
   55 CNDQNT(5)=5
      RETURN
   54 CNDQNT(5)=4
      RETURN
   53 CNDQNT(5)=3
      RETURN
   52 CNDQNT(5)=2
      RETURN
   51 CNDQNT(5)=1
      RETURN
      END
```

B.2.5 Sub-Program TUNING

```
      SUBROUTINE TUNER
      DIMENSION CNDQNT(5),ATRNET(5),CNGATR(5),DECTBL(5,5,5),PRFORM(5)
```

```
      CHARACTER*5,CNVAR(5)
      CHARACTER*6,CNQNT(5)
      CHARACTER*5,ACVAR(5)
      CHARACTER*5,ACQNT(5)
      INTEGER CNDQNT

      ENTRY DECISION (DECTBL)
      OPEN(UNIT=64,FILE='DECISION.TBL')
      do 5 i = 1,5
      DO 5 j = 1,5
      READ (64,14) (DECTBL(i,1,j),1=1,5)
   14 FORMAT(5F10.2)
    5 CONTINUE
      CLOSE(64)
      RETURN

C                   MAXIMUM PHASE
C                   -------------
      ENTRY TUNEPHMAX (CNDQNT,CNGATR,IPHMAX,ATRNET)
C     IF(CNDQNT(2).EQ.1)THEN
C                   IPHMAX=2
C     ELSE
C                   IPHMAX=3
C     ENDIF


C      IF(CNDQNT(3).LE.CNDQNT(2).AND.CNDQNT(2).NE.4)THEN
      IF(CNDQNT(2).LT.CNDQNT(3))THEN
                                    IPHMAX=2
      CNDM=CNDQNT(2)
      ELSE
                                    IPHMAX=3
      CNDM=CNDQNT(3)
      ENDIF
C      IF(CNDQNT(4).LT.CNDM.AND.CNDQNT(4).NE.4)          IPHMAX=4
      IF(CNDQNT(4).LT.CNDM)          IPHMAX=4

      ATRNET(1)=ATRNET(1)+CNGATR(1)*DECTBL(IPHMAX,CNDQNT(IPHMAX),1)

c      IF (ATRNET(1).LE.0.5 )THEN
c      ATRNET(1)=0.5
```

```
c       ELSEIF(ATRNET(1).GE.1.5 )THEN
        IF(ATRNET(1).GE.1.5 )THEN
        ATRNET(1)=1.5
        ENDIF
        RETURN


C                       FREQUENCY AT MAXIMUM PHASE
C                       --------------------------
        ENTRY TUNEFRMAX (CNDQNT,CNGATR,IFRMAX,ATRNET)

C       IF(CNDQNT(2).LT.4) THEN
C                        IFRMAX=2
C       ELSE
C                        IFRMAX=5
C       ENDIF

C        IF(CNDQNT(2).LE.CNDQNT(5).AND.CNDQNT(2).NE.4)THEN
        IF(CNDQNT(2).LE.CNDQNT(4))THEN
                              IFRMAX=2
C        CNDM=CNDQNT(2)
        ELSE
                              IFRMAX=5
        CNDM=CNDQNT(5)
        ENDIF
        IF(CNDQNT(4).LT.CNDM) IFRMAX=4

        ATRNET(2)=ATRNET(2)+CNGATR(2)*DECTBL(IFRMAX,CNDQNT(IFRMAX),2)

        IF(ATRNET(2).LE.20.)THEN
        ATRNET(2)=20.
        ELSEIF(ATRNET(2).GE.600.)THEN
        ATRNET(2)=600.
        ENDIF
        RETURN

C                       HIGH FREQUENCY GAIN
C                       -------------------
        ENTRY TUNEHIFGN (CNDQNT,CNGATR,IHFRGN,ATRNET)

c       IF(CNDQNT(2).EQ.1)     THEN
```

```
c                    IHFRGN=2
c      ELSEIF(CNDQNT(5).NE.4)THEN
c                    IHFRGN=5
c      ELSE
c                    IHFRGN=4
c      ENDIF

C       IF(CNDQNT(2).LE.CNDQNT(4).AND.CNDQNT(2).NE.4)THEN
        IF(CNDQNT(5).LE.CNDQNT(4))THEN
                             IHFRGN=5
        CNDM=CNDQNT(5)
         ELSE
                             IHFRGN=4
         CNDM=CNDQNT(4)
        ENDIF

        IF(CNDQNT(3).LT.CNDM) THEN
                             IHFRGN=3
          CNDM=CNDQNT(3)
        ENDIF
        IF(CNDQNT(1).LT.CNDM)IHFRGN=1

      ATRNET(3)=ATRNET(3)+CNGATR(3)*DECTBL(IHFRGN,CNDQNT(IHFRGN),3)

c!      IF(ATRNET(3).LE.0.1)THEN
c!      ATRNET(3)=0.1
c!      ELSEIF(ATRNET(3).GE.10.)THEN
c!      ATRNET(3)=10.
c!      ENDIF
      RETURN


C                   LOW FREQUENCY GAIN
C                   ------------------
      ENTRY TUNELOFGN (CNDQNT,CNGATR,ILFRGN,ATRNET)

      ATRNET(4)=ATRNET(4)+CNGATR(4)*DECTBL(1,CNDQNT(1),4)
      IF (ATRNET(4).LT.0) ATRNET(4)=0.

      RETURN
      END
```

B.2.6 Sub-Program DESIGN

```
      SUBROUTINE DESIGN(ATRNET,CNTRLC)
      DIMENSION CNTRLC(5),ATRNET(4)
      SPHMAX=SIN(ATRNET(1))
      ALFA=(1.-SPHMAX)/(1.+SPHMAX)
      SQRALF=SQRT(ALFA)
      CNTRLC(3)=ATRNET(2)*SQRALF
      CNTRLC(4)=ATRNET(2)/SQRALF
      CNTRLC(1)=1/ALFA*ATRNET(3)
      CNTRLC(2)=ATRNET(4)
      CNTRLC(5)=0.
C      PRINT 100,CNTRLC(1),CNTRLC(2),CNTRLC(3),CNTRLC(4)
C  100 FORMAT(1X,'GNC=',F5.0,5X,'GIC=',F5.0,'ZRC=',F5.0,5X,'PLC=',F5.0)
      RETURN
      END

      SUBROUTINE ZTRANS(DTSMP,CNTRLC,CNTRLZ)
      DIMENSION CNTRLC(5),CNTRLZ(5)
      POZ   = CNTRLC(4)/CNTRLC(3)
      EXDTPZ=EXP(-DTSMP*(CNTRLC(4)-CNTRLC(3)))
      CNTRLZ(1)    = CNTRLC(1)*EXDTPZ
      CNTRLZ(2)    = CNTRLC(2)/7.8
      EXSPL = EXP(-DTSMP*CNTRLC(4))
      CNTRLZ(3)    = EXSPL/EXDTPZ
      CNTRLZ(4)   =  EXSPL
      CNTRLZ(5)=0.
      RETURN
      END

      SUBROUTINE DMCCONTROL (CNTRLZ,CNTRLD)
      DIMENSION CNTRLZ(5),CNTRLD(5)
      INTEGER*4,CNTRLD
C
      CNTRLD(1)=INT(2*CNTRLZ(1))
      IF(CNTRLD(1).GT.255)THEN
          CNTRLD(1)=255
          PRINT*,'MAX. GAIN!'
      ENDIF
      IF(CNTRLD(1).LT.0.OR.CNTRLD(1).EQ.1)THEN
```

```
          CNTRLD(1)  =0
          PRINT*,'MIN. GAIN=',CNTRLD(1)
      ENDIF
C
C CHECK CORRECTNESS OF NEXT LINE (256?)
      CNTRLD(2)=INT(127*CNTRLZ(2))
      IF(CNTRLD(2).GT.127)                CNTRLD(2)=127
      IF(CNTRLD(2).LT.0)                  CNTRLD(2)=0.
C
      CNTRLD(3)=INT(256*CNTRLZ(3))
      IF(CNTRLD(3).GT.255)THEN
          CNTRLD(3)=255
          PRINT*,'MAX. ZERO!'
      ENDIF
      IF(CNTRLD(3).LT.0. )THEN
          CNTRLD(3)=  0
          PRINT*,'MIN. ZERO!'
      ENDIF
C
      CNTRLD(4)=INT(256*CNTRLZ(4))
      IF(CNTRLD(4).GT.255)THEN
          CNTRLD(4)=255
          PRINT*,'MAX. POLE!'
      ENDIF
      IF(CNTRLD(4).LT.0. )THEN
          CNTRLD(4)=0
          PRINT*,'MIN. POLE!'
      ENDIF
C
      CNTRLD(5)=0
C
      RETURN
      END
```

B.2.7 Print Out

```
      SUBROUTINE PRINTOUT
      DIMENSION CNDQNT(5),ATRNET(5),CNGATR(5),DECTBL(5,5,5),PRFORM(5)
      CHARACTER*5,CNVAR(5)
      CHARACTER*6,CNQNT(5)
```

```
      CHARACTER*5,ACVAR(5)
      CHARACTER*5,ACQNT(5)
      INTEGER CNDQNT

      ENTRY INITLINGU
      OPEN(UNIT=74,FILE='LINGUIST.DAT')
      READ (74,15) CNVAR,CNQNT,ACVAR,ACQNT
   15 FORMAT (5A5,5A6,5A5,5A5)
      CLOSE(74)
      RETURN

      ENTRY INITPRNT(ATRNET)
      PHMAXD=ATRNET(1)*57.29
      PRINT  100
  100 FORMAT (20X,'Initial attributes:')
      PRINT  101,PHMAXD,ATRNET(2),ATRNET(3),ATRNET(4)
  101 FORMAT (5X,'PHMAX=',F4.1,5X,'FRMAX=',F4.0,4X,'HFRGN=',F4.2
     @,5X,'LFRGN=',F4.1)
      RETURN

      ENTRY PRNT(CNDQNT,PRFORM,DECTBL,ATRNET,
     @IPHMAX,IFRMAX,IHFRGN,ILFRGN,K)
      PHMAXD=ATRNET(1)*57.29

      PRINT 102,K
  102 FORMAT(20X,'TEST SIGNAL',2X,I3)
      PRINT 103,CNVAR(IPHMAX),CNQNT(CNDQNT(IPHMAX)),PRFORM(IPHMAX)
     @             ,ACVAR(1),DECTBL(IPHMAX,CNDQNT(IPHMAX),1),PHMAXD
  103 FORMAT(1X,A5,' = ',A6,' (',F5.2,') ==> ',A5,'AC= ',F4.1,2X
     @                                          ,'PHMAX=',F7.2)
      PRINT 104,CNVAR(IFRMAX),CNQNT(CNDQNT(IFRMAX)),PRFORM(IFRMAX)
     @              ,ACVAR(2),DECTBL(IFRMAX,CNDQNT(IFRMAX),2),ATRNET(2)
  104 FORMAT(1X,A5,' = ',A6,' (',F5.2,') ==> ',A5,'AC= ',F4.1,2X
     @                                          ,'FRMAX=',F7.2)
      PRINT 105,CNVAR(IHFRGN),CNQNT(CNDQNT(IHFRGN)),PRFORM(IHFRGN)
     @              ,ACVAR(3),DECTBL(IHFRGN,CNDQNT(IHFRGN),3),ATRNET(3)
  105 FORMAT(1X,A5,' = ',A6,' (',F5.2,') ==> ',A5,'AC= ',F4.1,2X
     @                                          ,'HFRGN=',F7.2)
      PRINT 106,CNVAR(1),CNQNT(CNDQNT(1)),PRFORM(1),ACVAR(4)
     @                                 ,DECTBL(1,CNDQNT(1),4),ATRNET(4)
  106 FORMAT(1X,A5,' = ',A6,' (',F5.2,') ==> ',A5,'AC= ',F4.1,2X
```

```
     @                                                    ,'LFRGN=',F7.2)
      RETURN

      ENTRY FINALPRNT(CNDQNT)
      PRINT 107
  107 FORMAT(20X,'Final performance:')
      PRINT 108,CNVAR(1),CNQNT(CNDQNT(1)),CNVAR(2),CNQNT(CNDQNT(2))
     @          ,CNVAR(3),CNQNT(CNDQNT(3)),CNVAR(4),CNQNT(CNDQNT(4))
     @          ,CNVAR(5),CNQNT(CNDQNT(5))
  108 FORMAT(1X,A5,'=',A6,3X,A5,'=',A6,3X,A5,'=',A6,3X,A5,'=',A6
     @,3X,A5,'=',A6)
      RETURN
      END
*eof
```

B.3 Fuzzy Tuner Program listing

B.3.1 Main Program DECISION

```
          PROGRAM DECISION


C         Program DECISION computes, off-line, the Fuzzy Tunner Decision-Table.

C         3 Data base are used:

C      a)  Ruleset Table (RULST), generated in sub-program SUBRULST.
c      b)  Membership Functions table of the Condition Variables (CNDMF), generated
c         in sub-program SUBCNDMF.
C      c)  Membership Functions of the Action Variables (ACTMF), generated
c         in sub-program SUBACTMF.

c          Sub-program SUBRLTION computes the Fuzzy Relations Table (RLTION)
c         using SUP (Sub-program SUP) of MIN (Sub-program MIN) operation.

c          This Relation table is now matched with the Membership Function of
c         Condition Variable by applying the compositionl rule of inference
c         ("SUP" of "MIN"), in sub-Program SUBDCSION, to generate the Fuzzy value
c         of the action. This value is defuzzified in sub-program SUBDEFUZD, using
c          the Center Of Gravity Method.

c          The Decision table is stores in DCSION to be used by the Servo Expert
c         To produce the Tunning Action in real-time.



      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax

      DIMENSION CNDMF(5,5,5),ACTMF(5,5,5),RULST(5,5,5)
     #,BARLEV(5,5,5,5,5),RLTION(5,5,5,5),DCSION(5,5,5)

      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
```

```
      CHARACTER*5,ACTQNT
      INTEGER ACTGRD,RULST
C
      CALL DAT
      CALL SUBCNDMF (CNDMF)
      CALL SUBACTMF (ACTMF)
      CALL SUBRULST (RULST)
C
      CALL SUBRLTION (CNDMF,ACTMF,RULST,RLTION)
      CALL SUBDCSION (RLTION,CNDMF,DCSION)
C
C
      STOP
      END
```

B.3.2 Sub-Program DATA

```
      SUBROUTINE DATA
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
c
      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT
      INTEGER ACTGRD
c
c     Membership Function cardinality for the Condition Variables:
      kmax=5
c
c     Membership Function cardinality for the Action Variables:
      nmax=5
c
C
c     CONDITION AND ACTION VARIABLES AND  QUANTITIES:
c  ------------------------------------------------
c
c     Fuzzy CoNDdition VARiables (CNDVAR), index i, Cardinality imax.
      imax=5
C!      IMAX=1
c
c     OFFSET in the error response:
      CNDVAR(1)='OFFST'
c
c     OSCILlation in error response:
      CNDVAR(2)='DMPRT'
c
c     SPEED of response:
      CNDVAR(3)='RISIN'
c
c     OVer SHoot  of the error response:
      CNDVAR(4)='OVSHT'
c
c     Damped natural frewuency
      CNDVAR(5)='DMPFR'
```

```
c
c     Fuzzy CoNDition QuaNTitie (CONQNT) index j, Cardinality jmax.
c
      jmax=5
c
c                           GRAD
c                          ---------
c     Well UNder SPCisfiaction   (-3)
      CNDQNT(1)='UNSATF'
c
c     UNDer SPCification         (-2)
      CNDQNT(2)='POOR  '
c
c     IN_SPCification            (-1)
      CNDQNT(3)='MODRAT'
c
c     OKEY                       ( 0)
      CNDQNT(4)='IN_SPC'
c
c     Well OVer SPCification      (1)
      CNDQNT(5)='OVRSPC'
c
c     Fuzzy Action Variables index l, Cardinality lmax.
c
c!        LMAX=1
      lmax=5
C
c     Change the PHase LeaD in the C.O freq.
      ACTVAR(1)='PHMAX'
c
c     Change the CRose-OVer Frequency.
      ACTVAR(2)='FRMAX'

c     Change the HIgh Frequency Gain.
      ACTVAR(3)='HFRGN'
c
c     Change the Low  Frequency Gain.
      ACTVAR(4)='LFRGN'
c
c     Change the Dead BaNd Zoon.
      ACTVAR(5)='DBNDZ'
```

```
c
c
c     Fuzzy ACTION Quantities (Grades from -2 to +2), index m, Cardinality mmax.
c
      mmax=5
c
c                          GRAD
c                         ----------
c     Negative HI           (-2)
      ACTQNT(1)='NEGHI'
      ACTGRD(1) = -2
c
c     Negative LO           (-1)
      ACTQNT(2)='NEGLO'
      ACTGRD(2) = -1
c
c     No ChanGe             ( 0)
      ACTQNT(3)='NOCHG'
      ACTGRD(3) = 0
c
c     Positive LO           (+1)
      ACTQNT(4)='POSLO'
      ACTGRD(4) = 1
c
c     Popsitive HI          (+2)
      ACTQNT(5)='POSHI'
      ACTGRD(5) = 2
c
      OPEN(UNIT=74,FILE='LINGUIST.DAT')
      WRITE (74,15) CNDVAR,CNDQNT,ACTVAR,ACTQNT
   15 FORMAT (5A5,5A6,5A5,5A5)
      CLOSE(74)
c
      RETURN
      END
```

B.3.3 Sub-Program CNDMF

```
      SUBROUTINE SUBCNDMF (CNDMF)
c
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
      DIMENSION CNDMF(5,5,5)
      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT
      INTEGER ACTGRD
c
      open(unit=60,file='cndmf.tbl')
c
c    Asume Crisp Values for Condition i, Quantity j (i.e [0 0 0 1 0] ):
c
      DO 2 i = 1,imax
      DO 2 J = 1,jmax
c      DO 2 k = 1,kmax
c      IF (j.eq.k) GO TO 1
c      CNDMF(i,j,k) = 0.
c      GO TO 2
c  1   CNDMF(i,j,k) = 1.
      DO 2 k = j,kmax
      CNDMF(i,j,k)=1.-(k-j)*0.2
      CNDMF(i,k,j)=CNDMF(i,j,k)
  2   CONTINUE
c
      DO 3 i = 1,imax
c
      PRINT 10,CNDVAR(i)
      WRITE(60,10) CNDVAR(i)
  10  FORMAT (1h0,//,'          Membership Function Table for Condition
     @Variable:',A5)
      PRINT 11
      WRITE(60,11)
  11  FORMAT (1H,23X,' 1         2         3         4         5')
c
      PRINT 12
```

```
      WRITE(60,12)
   12 FORMAT (1H,15X,50('-'))
C
      DO 3 J = 1,jmax
      PRINT 13 ,CNDQNT(j),(CNDMF(i,j,k),k=1,kmax)
      WRITE(60,13) CNDQNT(j),(CNDMF(i,j,k),k=1,kmax)
   13 FORMAT (1h,8x,A6,' |',5F10.1)
C

   3  CONTINUE
      close(60)
C
      RETURN
      END
```

B.3.4 Sub-Program ACTMF

```
      SUBROUTINE SUBACTMF (ACTMF)
c
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
      DIMENSION ACTMF(5,5,5)

      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT
      INTEGER ACTGRD

c
      open(unit=61,file='actmf.tbl')
c     Asume Crisp Values for Condition l, Quantity m (i.e [0 0 0 1 0] ):
c
      DO 2 l = 1,lmax
      DO 2 m = 1,mmax
c      DO 2 n = 1,nmax
c
c      IF (m.eq.n) GO TO 1
c      ACTMF(l,m,n) = 0.
c      GO TO 2
c  1   ACTMF(l,m,n) = 1.
c  2   CONTINUE
      DO 2 n = m,mmax
      ACTMF(l,m,n)=1.-(n-m)*0.2
      ACTMF(l,n,m)=ACTMF(l,m,n)
  2   CONTINUE
c
      DO 3 l = 1,lmax
c
      PRINT 10,ACTVAR(l)
      WRITE(61,10)
  10  FORMAT (1h0,//,14X,' Membership  Function  Table  for  the Action
     @Variable:',A5,/,)
c
      PRINT 11, (ACTGRD(n),n=1,nmax)
```

```
      WRITE(61,11)
   11 FORMAT (1H,15X,5I10,/,10X,58('-'))
      WRITE(61,12)
   12 FORMAT (1H,15X,50('-'))
C
C
      DO 3 m = 1,mmax
      PRINT 13 ,ACTQNT(m),(ACTMF(1,m,n),n=1,nmax)
      WRITE (61,13) ACTQNT(m),(ACTMF(1,m,n),n=1,nmax)
   13 FORMAT (1h,8X,A5,' |',5F10.1)
C
    3 CONTINUE
      CLOSE(61)
C     RETURN
      END
```

B.3.5 Sub-Program RULESET

```
      SUBROUTINE  SUBRULST (RULST)
c
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
c
      DIMENSION RULST(5,5,5),RULST1(5,5,5),RULS(5,5,5)

      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT
      CHARACTER*5,RULS
      INTEGER RULST,RULST1,ACTGRD
c
c     Rules for  default: IF CONDITION(i) = OKY, THEN ACTION(1) = NCH .
c
      open(unit=62,file='rulset.tbl')
c
      DO 1 i = 1,imax
      DO 1 j = 1,jmax
      DO 1 l = 1,lmax
      RULST(i,j,l) = 3
      RULS (i,j,l)=ACTQNT(3)
 1    CONTINUE

C     RULES FOR: IF OSCIL = CNDVAR THEN PHMAX = ACTQNT

C     if OSCIL = WUNSPC then PHMAX = POSHI
       RULS (2,1,1) = ACTQNT(5)
       RULST(2,1,1) =  5
C     if OSCIL = UNDSPC then PHMAX = NOCNG
       RULS (2,2,1) = ACTQNT(3)
       RULST(2,2,1) =  3
C     if OSCIL = IN_SPC then PHMAX = NOCNG
       RULS (2,3,1) = ACTQNT(3)
       RULST(2,3,1) =  3
C     if OSCIL = OK     then PHMAX = NOCNG
       RULS (2,4,1) = ACTQNT(3)
```

```
       RULST(2,4,1) =  3
C      if OSCIL = WOVSPC then PHMAX = NOCNG
       RULS (2,5,1) = ACTQNT(3)
       RULST(2,5,1) =  3
C      RULES FOR: IF SPEED = CNDVAR THEN PHCOV = ACTQNT

C      if SPEED = WUNSPC then PHCOV = NEGHI
       RULS (3,1,1) = ACTQNT(1)
       RULST(3,1,1) =  1
C      if SPEED = UNDSPC then PHCOV = NEGLO
       RULS (3,2,1) = ACTQNT(2)
       RULST(3,2,1) =  2
C      if SPEED = IN_SPC then PHCOV = NEGLO
       RULS (3,3,1) = ACTQNT(2)
       RULST(3,3,1) =  2
C      if SPEED = OK     then PHCOV = NOCNG
       RULS (3,4,1) = ACTQNT(3)
       RULST(3,4,1) =  3
C      if SPEED = WOVSPC then PHCOV = POSLO
       RULS (3,5,1) = ACTQNT(4)
       RULST(3,5,1) =  4
C

C      RULES FOR: IF OVSHT = CNDVAR THEN PHMAX = ACTQNT

C      if OVSHT = WUNSPC then PHMAX = POSHI
       RULS (4,1,1) = ACTQNT(5)
       RULST(4,1,1) =  5
C      if OVSHT = UNDSPC then PHMAX = POSLO
       RULS (4,2,1) = ACTQNT(4)
       RULST(4,2,1) =  4
C      if OVSHT = IN_SPC then PHMAX = POSLO
       RULS (4,3,1) = ACTQNT(3)
       RULST(4,3,1) =  4
C      if OVSHT = OK     then PHMAX = NOCNG
       RULS (4,4,1) = ACTQNT(3)
       RULST(4,4,1) =  3
C      if OVSHT = WOVSPC then PHMAX = NEGLO
       RULS (4,5,1) = ACTQNT(2)
       RULST(4,5,1) =  2
```

```
C     RULES FOR: IF OSCIL = CNDVAR THEN COVFR = ACTQNT

C     if OSCIL = WUNSPC then PHCOV = NEGHI
       RULS (2,1,2) = ACTQNT(1)
       RULST(2,1,2) =  1
C     if OSCIL = UNDSPC then PHCOV = NEGLO
       RULS (2,2,2) = ACTQNT(2)
       RULST(2,2,2) =  2
C     if OSCIL = IN_SPC then PHCOV = NEGLO
       RULS (2,3,2) = ACTQNT(2)
       RULST(2,3,2) =  2
C     if OSCIL = OK     then PHCOV = NOCNG
       RULS (2,4,2) = ACTQNT(3)
       RULST(2,4,2) =  3
C     if OSCIL = WOVSPC then PHCOV = NOCNG
       RULS (2,5,2) = ACTQNT(4)
       RULST(2,5,2) =  4

C     RULES FOR: IF OVSHT = CNDVAR THEN COVFR = ACTQNT

C     if OVSHT = WUNSPC then PHCOV = NEGHI
       RULS (4,1,2) = ACTQNT(1)
       RULST(4,1,2) =  1
C     if OVSHT = UNDSPC then PHCOV = NEGLO
       RULS (4,2,2) = ACTQNT(2)
       RULST(4,2,2) =  2
C     if OVSHT = IN_SPC then PHCOV = NEGLO
       RULS (4,3,2) = ACTQNT(2)
       RULST(4,3,2) =  2
C     if OVSHT = OK     then PHCOV = NOCNG
       RULS (4,4,2) = ACTQNT(3)
       RULST(4,4,2) =  3
C     if OVSHT = WOVSPC then PHCOV = NOCNG
       RULS (4,5,2) = ACTQNT(4)
       RULST(4,5,2) =  4

C     RULES FOR: IF DMPFR = CNDVAR THEN COVFR = ACTQNT
```

```
C      if DMPFR = WUNSPC then PHCOV = NEGHI
        RULS (4,1,2) = ACTQNT(1)
        RULST(4,1,2) =  1
C      if DMPFR = UNDSPC then PHCOV = NEGLO
        RULS (4,2,2) = ACTQNT(2)
        RULST(4,2,2) =  2
C      if DMPFR = IN_SPC then PHCOV = NEGLO
        RULS (4,3,2) = ACTQNT(2)
        RULST(4,3,2) =  2
C      if DMPFR = OK     then PHCOV = NOCNG
        RULS (4,4,2) = ACTQNT(3)
        RULST(4,4,2) =  3
C      if DMPFR = WOVSPC then PHCOV = NOCNG
        RULS (4,5,2) = ACTQNT(4)
        RULST(4,5,2) =  4

C      if DMPFR = WUNSPC then FRMAX = NEGHI
        RULS (5,1,2) = ACTQNT(1)
        RULST(5,1,2) =  1
C      if DMPFR = UNDSPC then FRMAX = NEGLO
        RULS (5,2,2) = ACTQNT(2)
        RULST(5,2,2) =  2
C      if DMPFR = IN_SPC then FRMAX = NEGLO
        RULS (5,3,2) = ACTQNT(2)
        RULST(5,3,2) =  2
C      if DMPFR = OK     then FRMAX = NOCNG
        RULS (5,4,2) = ACTQNT(3)
        RULST(5,4,2) =  3
C      if DMPFR = WOVSPC then FRMAX = POSHI
        RULS (5,5,2) = ACTQNT(4)
        RULST(5,5,2) =  4
c
c       RULES FOR: IF OFFST = CNDVAR THEN HFRGN = ACTQNT

C      if OFFST = WUNSPC then HFRGN = POSHI
        RULS (1,1,3) = ACTQNT(5)
        RULST(1,1,3) =  5
C      if OFFST = UNDSPC then HFRGN = POSLO
        RULS (1,2,3) = ACTQNT(4)
        RULST(1,2,3) =  4
```

```
C     if OFFST = IN_SPC then HFRGN = POSLO
       RULS (1,3,3) = ACTQNT(4)
       RULST(1,3,3) =  4
C     if OFFST = OK     then HFRGN = NOCNG
       RULS (1,4,3) = ACTQNT(3)
       RULST(1,4,3) =  3
C     if OFFST = WOVSPC then HFRGN = NEGLO
       RULS (1,5,3) = ACTQNT(2)
       RULST(1,5,3) =  2

C     RULES FOR: IF OSCIL = CNDVAR THEN HFGRGN = ACTQNT

C     if OSCIL = WUNSPC then HFRGN = NEGHI
C       RULS (2,1,3) = ACTQNT(1)
C       RULST(2,1,3) =  1
C     if OSCIL = UNDSPC then HFRGN = NOCNG
C       RULS (2,2,3) = ACTQNT(3)
C       RULST(2,2,3) =  3
C     if OSCIL = IN_SPC then HFRGN = NOGNG
C       RULS (2,3,3) = ACTQNT(3)
C       RULST(2,3,3) =  3
C     if OSCIL = OK     then HFRGN = NOCNG
C       RULS (2,4,3) = ACTQNT(3)
C       RULST(2,4,3) =  3
C     if OSCIL = WOVSPC then HFRGN = NOGNG
C       RULS (2,5,3) = ACTQNT(3)
C       RULST(2,5,3) =  3


C     RULES FOR: IF RISE = CNDVAR THEN HFRGN = ACTQNT


C     if RISE  = WUNSPC then HFRGN = POSHI
      RULS (3,1,3) = ACTQNT(5)
      RULST(3,1,3) =  5
C     if RISE  = UNDSPC then HFRGN = POSLO
      RULS (3,2,3) = ACTQNT(4)
      RULST(3,2,3) =  4
C     if RISE  = IN_SPC then HFRGN = POSLO
      RULS (3,3,3) = ACTQNT(3)
      RULST(3,3,3) =  4
```

```
C     if RISE  = OK      then HFRGN = NOCNG
      RULS (3,4,3) = ACTQNT(3)
      RULST(3,4,3) =  3
C     if RISE  = WOVSPC then HFRGN = NEGLO
      RULS (3,5,3) = ACTQNT(2)
      RULST(3,5,3) =  2


C     RULES FOR: IF OVSHT = CNDVAR THEN HFRGN = ACTQNT


C     if OVSHT = WUNSPC then HFRGN = POSHI
      RULS (4,1,3) = ACTQNT(5)
      RULST(4,1,3) =  5
C     if OVSHT = UNDSPC then HFRGN = POSLO
      RULS (4,2,3) = ACTQNT(4)
      RULST(4,2,3) =  4
C     if OVSHT = IN_SPC then HFRGN = POSLO
      RULS (4,3,3) = ACTQNT(3)
      RULST(4,3,3) =  4
C     if OVSHT = OK      then HFRGN = NOCNG
      RULS (4,4,3) = ACTQNT(3)
      RULST(4,4,3) =  3
C     if OVSHT = WOVSPC then HFRGN = NEGLO
      RULS (4,5,3) = ACTQNT(2)
      RULST(4,5,3) =  2
C
C     RULES FOR: IF DMPFR = CNDVAR THEN HFGRGN = ACTQNT

C     if DMPFR = WUNSPC then HFRGN = POSHI
       RULS (5,1,3) = ACTQNT(5)
       RULST(5,1,3) =  5
C     if DMPFR = UNDSPC then HFRGN = POSLO
       RULS (5,2,3) = ACTQNT(4)
       RULST(5,2,3) =  4
C     if DMPFR = IN_SPC then HFRGN = POSLO
       RULS (5,3,3) = ACTQNT(4)
       RULST(5,3,3) =  4
C     if DMPFR = OK      then HFRGN = NOCNG
       RULS (5,4,3) = ACTQNT(3)
```

```
       RULST(5,4,3) =  3
C      if DMPFR = WOVSPC then HFRGN = NEGHI
       RULS (5,5,3) = ACTQNT(1)
       RULST(5,5,3) =  1

c      RULES FOR: IF OFFST = CNDVAR THEN LFRGN = ACTQNT

C      if OFFST = WUNSPC then LFRGN = POSHI
       RULS (1,1,4) = ACTQNT(5)
       RULST(1,1,4) =  5
C      if OFFST = UNDSPC then LFRGN = POSLO
       RULS (1,2,4) = ACTQNT(4)
       RULST(1,2,4) =  4
C      if OFFST = IN_SPC then LFRGN = POSLO
       RULS (1,3,4) = ACTQNT(4)
       RULST(1,3,4) =  4
C      if OFFST = OK     then LFRGN = NOCNG
       RULS (1,4,4) = ACTQNT(3)
       RULST(1,4,4) =  3
C      if OFFST = WOVSPC then LFRGN = NEGLO
       RULS (1,5,4) = ACTQNT(2)
       RULST(1,5,4) =  2

       DO 2 i = 1,imax
       PRINT 10,CNDVAR(i)
       WRITE(62,10) CNDVAR(i)
 10    FORMAT(1h0,//,30X,'Rules for Condition Variable:',A5,/)
       PRINT 11,(ACTVAR(l),l=1,lmax)
       WRITE (62,11) (ACTVAR(l),l=1,lmax)
 11    FORMAT(1H,22X,5A10,/,20x,55('-'))
c
       DO 2 J = 1,jmax
       PRINT 12,CNDQNT(j),(RULS(i,j,l),l=1,lmax)
c       PRINT 13,CNDQNT(j),(RULST(i,j,l),l=1,lmax)
       WRITE(62,12) CNDQNT(j),(RULST(i,j,l),l=1,lmax)
 12    FORMAT(1h,15x,A6,'|',5A10)
 13    FORMAT(1h,15x,A6,'|',5I10)
 2     CONTINUE
c
       CLOSE(62)
       RETURN
```

```
END
```

B.3.6 Sub-Program RELATION

```
      SUBROUTINE RELATION (CNDMF,ACTMF,RULST,RLTION)
c
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
      DIMENSION  CNDMF(5,5,5),ACTMF(5,5,5),BARLEV(5,5,5,5,5)
     @,RULST(5,5,5),RLTION(5,5,5,5)

      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT
      INTEGER ACTGRD,RULST

      OPEN(UNIT=63,FILE='RELATION.TBL')
c
      DO  1 i = 1,imax
      DO  1 l = 1,lmax
c
      IF(i.ne.1.or.l.ne.1) go to 110
      PRINT 10, CNDVAR(i),ACTVAR(l)
      WRITE(63,10) CNDVAR(i),ACTVAR(l)
   10 FORMAT(1H1,/////,10X,'Development of the Fuzzy Realation Table
     !for:',A5,' ==>',A5)
  110 continue
c
      DO 2 j = 1,jmax
c
c
      DO 3 k = 1,kmax
      DO 3 n = 1,nmax
c
      BARLEV(i,l,j,k,n) =
     # MIN(CNDMF(i,j,k),ACTMF(l,RULST(i,j,l),n))
c
   3  CONTINUE
c
      IF(i.ne.1.or.l.ne.1) go to 112
```

```
      PRINT 11 ,CNDVAR(i),CNDQNT(j),ACTVAR(l),ACTQNT(RULST(i,j,l))
      WRITE(63,11) CNDVAR(i),CNDQNT(j),ACTVAR(l),
     # ACTQNT(RULST(i,j,l))
  11  FORMAT(1h,//,25X,'IF  ',A5,' =',A6,' Then',A5,' =',A5)
      PRINT 12, (ACTGRD(n),n=1,nmax)
      WRITE (63,12) (ACTGRD(n),n=1,nmax)
  12  FORMAT(1h,/,22X,5I10,25X,55('_'))
 112  continue
c
      DO 2 k=1,kmax
c
      IF(i.ne.1.or.l.ne.1) go to 113
      PRINT 13,k,(BARLEV(i,l,j,k,n),n=1,nmax)
      WRITE(63,13) k,(BARLEV(i,l,j,k,n),n=1,nmax)
  13  FORMAT (1h,10X,I10,'|',5F10.1)
 113  continue
c
   2  CONTINUE
c
      DO 5 k = 1,nmax
      DO 5 n = 1,kmax
      RLTION(i,l,k,n)=BARLEV(i,l,1,k,n)
      DO 5 j = 1,jmax-1
c
      RLTION(i,l,k,n) = MAX(RLTION(i,l,k,n),BARLEV(i,l,j+1,k,n))
c
   5  CONTINUE
      IF(i.ne.1.or.l.ne.1) go to 115
c
      PRINT 14,CNDVAR(i),ACTVAR(l)
      WRITE(63,14) CNDVAR(i),ACTVAR(l)
  14  FORMAT(1h,//,17x,' Composite Relation Table for ',A5,' ==> ',A5)
      PRINT 15, (ACTGRD(n),n=1,nmax)
      WRITE(63,15) (ACTGRD(n),n=1,nmax)
  15  FORMAT (1H,/,22X,5I10,/,15X,58('-'))
 115  continue
c
      DO 7 k = 1,kmax
      IF(i.ne.1.or.l.ne.1) go to 116
      PRINT 16,k,(RLTION(i,l,k,n),n=1,nmax)
      WRITE(63,16) k,(RLTION(i,l,k,n),n=1,nmax)
```

```
  16  FORMAT(1h,10X,I10,'|',5F10.1)
 116  continue
   7  CONTINUE
c
   1  CONTINUE
c
      CLOSE(63)
      RETURN
      END
```

B.3.7 Sub-Program DECISION

```
      SUBROUTINE SUBDCSION (RLTION,CNDMF,DCSION)
c
      INTEGER ACTGRD
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
c
      DIMENSION CNDMF(5,5,5),RLTION(5,5,5,5),SHANY(5,5,5,5,5)
     !,YAEL(5,5,5,5),DCSION(5,5,5)

      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT

c
      OPEN(UNIT=64,FILE='DECISION.TBL')
      DO 1 i = 1,imax
c
      DO 1 l = 1,lmax
      DO 1 j = 1,jmax
c
      DO 2 n = 1,nmax
      DO 2 k = 1,kmax
c
      SHANY(i,l,j,k,n) = MIN(CNDMF(i,j,k),RLTION(i,l,k,n))
c
   2  continue
c!      do 100 k=1,kmax
c! 100  print 110,i,l,j,k, (shany(i,l,j,k,n),n=1,nmax)
c! 110  format(1h,2x,4i5,5f5.0)
c
      DO 3 n = 1,nmax
      YAEL(i,l,J,n)=SHANY(i,l,J,1,n)
      DO 3 k = 1,kmax-1
c
      YAEL(i,l,j,n) = MAX(YAEL(i,l,j,n),SHANY(i,l,j,k+1,n))
c
   3  CONTINUE
```

```
c
C      PRINT 10, (YAEL(i,l,j,n),n=1,nmax)
C      WRITE(64,10) (YAEL(i,l,j,n),n=1,nmax)
C  10  FORMAT(1h,10X,5F10.1)
   1  CONTINUE
c
      CALL SUBDFUZD (YAEL,DCSION)
c
      DO 5 i = 1,imax
c
      PRINT 11,CNDVAR(i)
C      WRITE(64,11) CNDVAR(i)
  11  FORMAT(1H,///,25X,'Fuzzy Decision Table for Condition:'
     @,2x,A5,/,20x,55('*'))
      PRINT 12, (ACTVAR(l),l=1,lmax)
C      WRITE(64,12) (ACTVAR(l),l=1,lmax)
  12  FORMAT(1H,/,40X,'ACTION VARIABLE',/,20x,5a10,/,10x,60('-'))
c
      DO 5 j = 1,jmax
c
      PRINT 13,CNDQNT(J),(DCSION(i,l,j),l=1,lmax)
      WRITE(64,14) (DCSION(i,j,l),l=1,lmax)
  13  FORMAT(1H,12X,A6,'|',5F10.1)
  14  FORMAT(5F10.2)

c
   5  CONTINUE
c
      CLOSE(64)
      RETURN
      END
```

B.3.8 Sub-Program DEFUZZY

```
      SUBROUTINE SUBDFUZD(YAEL,DCSION)
c
      INTEGER ACTGRD
      COMMON/COM1/ CNDVAR(5),CNDQNT(5),ACTVAR(5),ACTQNT(5),ACTGRD(5)
      COMMON/COM2/ imax,jmax,kmax,lmax,mmax,nmax
      DIMENSION YAEL(5,5,5,5),DCSION(5,5,5)

      CHARACTER*5,CNDVAR
      CHARACTER*6,CNDQNT
      CHARACTER*5,ACTVAR
      CHARACTER*5,ACTQNT

c
      DO 1 i = 1,imax
      DO 1 l = 1,lmax
      DO 1 j = 1,jmax
c
      DCSION(i,l,j) = YAEL(i,l,j,1)*ACTGRD(1)
      DO 1 n = 2,nmax
      DCSION(i,l,j) = DCSION(i,l,j)+YAEL(i,l,j,n)*ACTGRD(n)
   1  CONTINUE
      DCSION(i,l,j) = DCSION(i,l,j)/nmax
c
      RETURN
      END
*eof
```

B.4 Subprogram LEARN

```
      SUBROUTINE LEARN(K,PRFRS,THOLRN,CNGATR,RULSET)
      COMMON/COM1/ CNVAR,CNQNT,ACVAR,ACQNT
      DIMENSION RULSET(5,5,4),PRFRS(5),PRFRS0(5),CNGATR(4)
     #,THOLRN(5),DPRFRM(5),RULS(5,5,4)
      CHARACTER*5,CNVAR(5)
      CHARACTER*6,CNQNT(5)
      CHARACTER*5,ACVAR(4)
      CHARACTER*5,ACQNT(5)
      CHARACTER*5,RULS

      INTEGER RULSET

      OPEN(UNIT=70,FILE='RULS.LRN')
      OPEN(UNIT=71,FILE='RULSET.LRN')
      IF (K.EQ.1)THEN
           DO 1 I=1,5
         PRFRS0(I)=PRFRS(I)
         DO 1 J=1,5
         DO 1 L1=1,4
         RULSET(I,J,L1)=3
    1    RULS(I,J,L1)=ACQNT(3)
         RETURN
      ENDIF

      L=K-1

      DO 2 I=1,5
      DPRFRM(I)=(PRFRS(I)-PRFRS0(I))/ABS(PRFRS(I))

      IF(DPRFRM(I)*SIGN(1.,CNGATR(L)).LE.-THOLRN(I))THEN
         RULSET(I,1,L)= 5
         RULSET(I,2,L)= 4
         RULSET(I,3,L)= 4
         RULSET(I,4,L)= 3
         RULSET(I,5,L)= 2
      ELSEIF(DPRFRM(I)*SIGN(1.,CNGATR(L)).GE.+THOLRN(I))THEN
         RULSET(I,1,L)= 1
         RULSET(I,2,L)= 2
         RULSET(I,3,L)= 2
```

```
         RULSET(I,4,L)= 3
         RULSET(I,5,L)= 4
       ENDIF


         DO 6 J=1,5
    6    RULS(I,J,L)=ACQNT(RULSET(I,J,L))

    2 CONTINUE


       DO 3 i = 1,5
       WRITE(70,10) CNVAR(i)
   10  FORMAT(1h0,//,30X,'Rules for Condition Variable:',A5,/)
       WRITE (70,11) (ACVAR(l),l=1,4)
   11  FORMAT(1H,22X,4A10,/,20x,55('-'))

       DO 3 J = 1,5
       WRITE(70,12) CNQNT(j),(RULS(i,j,l),l=1,4)
   12  FORMAT(1h,15x,A6,'|',4A10)
       WRITE(71,13)(RULSET(i,j,l),l=1,4)
   13  FORMAT(4I10)
    3  CONTINUE
       CLOSE(70)
       CLOSE(71)
       RETURN
       END
```