MOTION CONTROL AND SYNCHRONISATION OF MULTI-AXIS DRIVE SYSTEMS

by

CHANGMIN CHEN, BSc, MSc

A Doctoral Thesis
submitted in partial fulfilment of the
requirements for the award of

Doctor of Philosophy

of the Loughborough University of Technology
Department of Manufacturing Engineering

November 1994

To my loving wife, Zhou Jian

DECLARATION

No part of the work described in this Thesis has been submitted in support of an application for any other degree or qualification of this or any other University or any other Institution of learning.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thanks all those people who have helped in one way or another in the completion of my studies here. Amongst them, the following people deserve special recognition.

My sincerest thanks to my supervisor, Professor P R Moore, for his guidance, patience, interest, encouragement and confidence in me, throughout the studies. I also honestly appreciate him for arranging a studentship to me and helping me to obtain other funds. In addition, I want to express my earnest gratitude to him for his careful and critical reading of the draft manuscripts.

My thanks also goes to the members of the Modular Systems Group: Dr J Pu, Dr R Harrison, Mr C B Wong, Mr A Carrot, Mr A Booth, Dr A A West, Mr C D Wright, Mr A S Goh, Professor R H Western, Mr N Armstrong, Mr T I Morley, Mrs M Carden and Miss M Wilkinson, for their help.

I would like to thanks Dr W Wang, Miss F Zheng, Miss G LI, Dr B L Zhang and Dr Q H Wu for their friendship.

My parents, parents-in-law and the family for their support.

My wife, Jian, for her love and support.

ORS Awards VCCP Committee and the Department of Manufacturing Engineering of LUT for their financial assistance.

Quin Systems Ltd for their technical assistance.

Synopsis

Contemporary digital motion control approaches can be used to radically simplify the design, improve the performance, and increase the flexibility of machinery. The three key components of this new motion control technology are highly dynamic servo-drives, software tools and computer hardware. By applying this new digital motion control technology, the traditional machine building blocks (such as mechanical gears, cams and linkages) can be simplified and some cases replaced by reprogrammable software. However, the tight control and flexibility typically required in modern multi-axis machines can be realised only if it can be ensured that the software controlled drives are properly coordinated and synchronised. This thesis reports on the concept and implementation of a new generic 'dual closed-loop' control scheme which offers the potential to improve the motion coordination and synchronisation capabilities of multi-axis motion control systems.

Industrial automation spans a huge spectrum in terms of both the physical structure of machines and the tasks which they perform. Such systems vary enormously in respect to the process they address and the range and complexity of operations they undertake. A study of current control methodologies highlights that a single control method can not satisfactorily solve all the control issues encountered. A combination of control methods is often necessary to attain the desired levels of coordination and synchronisation of these systems. For each axis of motion, an appropriate motion controller is required which can give adequate performance for the particular element of the machine process. The motions involved in the multi-axis machine system are then tightly synchronised through a supplementary control among the axes.

This supplementary control for multi-axis systems is introduced by inter-connecting the independent servo-drives. This control is used to coordinate the actions of the servo-drives, in order to reduce or eliminate the synchronisation error. An accurate mathematical coordination model for the coupled servo-drives is often difficult to attain due to uncertainness in the controlled processes and the different characteristics of the individual servo loops. Human intelligence (reasoning, knowledge and intuition) is good in tackling many coordination problems with little information and fuzzy logic provides a convenient way to represent human linguistic descriptions and make decisions. A fuzzy logic control strategy is presented which achieves synchronisation of the servo-drives through on-line corrective actions via compensation terms.

By combining: a fuzzy logic software synchronisation mechanism; an inter-connected control structure; and the hierarchical decomposition of the tasks in multi-axis motion control systems, a design method for multi-axis motion control has been devised, termed Intelligent Motion Control (IMC). This facilitates the design and control of multi-axis motion control systems in an intelligent manner. The IMC control structure provides for effective motion synchronisation.

A particular implementation of a control structure conforming to this method has been created. The implementation is based on the selective use of modern computer methods and motion controllers. A testbed has been established for verifying the motion synchronisation capability of various control schemes. The control method proposed can be used in manufacturing machines where tight motion synchronisation is required.

KEY WORDS: *Machine Motion Control, Multi-Axis Drive Systems, Motion Synchronisation, Fuzzy Logic Control, Coupling Control Structure.*

CHAPTER ONE
Introduction

Highly competitive markets encourage manufacturing industry to realise improved performance through greater machine flexibility and productivity. Hence, many manufacturers are moving toward a newer category of specialized production equipment featuring "***soft automation***", which means numerical control of machines and processes[1]. Most of the machines and/or processes being automated involve motion; therefore, they will require motion control and, quite often, coordinated motion. A typical industrial robot has five or six axes of motion that must be coordinated. Camshaft grinding, gear hobbing, welding, packaging, material handing, and a host of other applications require two or more axes of coordinated motion.

In this thesis an integrated approach is taken to create a multi-axis motion control and synchronisation scheme to provide the tight control and flexibility necessary to support modern manufacturing operations. In this context the research focuses on the design and implementation of a multi-axis motion control system.

1.1 Motivations and Background -- Design Trends in Advanced Machines

Until recently machine designers were restricted to the use of mechanical transmissions such as cams and linkages to build coordinated motion systems for machines. Although these mechanisms have been progressively improved in design, precision and material used, the underlying principles behind the majority of mechanisms used today can be traced back at least to the time of Leonardo da Vinci (around 1500 AD)[2][3][16]. Nowadays, motion control increasingly involves the use of integrated circuits and microprocessors – basically, computer control[4], reducing the mechanical complexity of machines.

Continued advances in controllers, drives, and feedback devices promise ever greater diversity. Consider:

- Computational–intensive controllers allow smart algorithms with interpretive and expert systems to be integrated.
- Ongoing goals for manufacturers include: more power from smaller motors; reduced inertia for quicker response; and higher flux densities from smaller magnets.
- Position feedback can also be integrated into motor electronics.

The following examples illustrate some of the ways today's motion control approaches can be used to simplify the design, and generally improve the speed and performance of manufacturing operations.

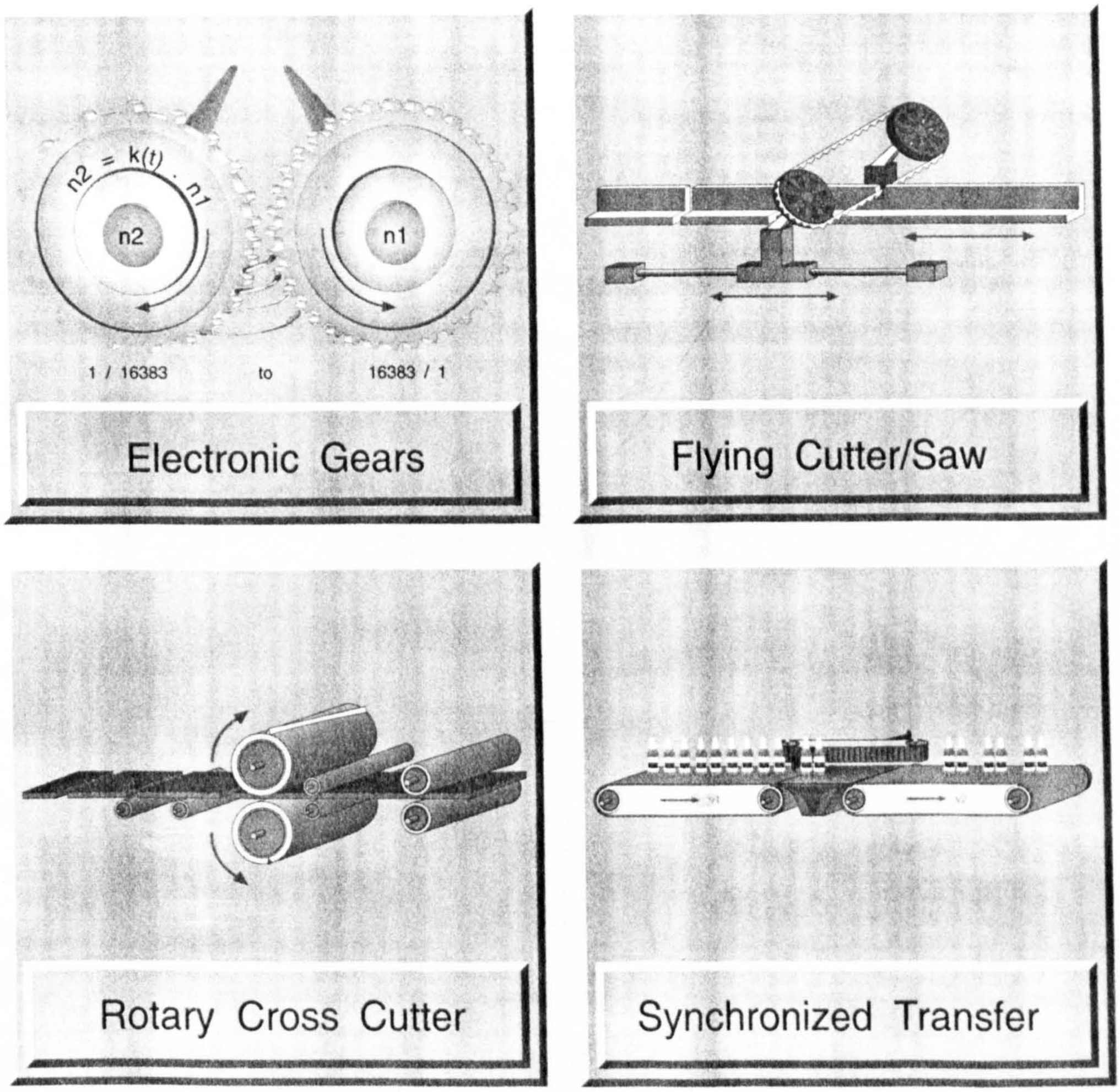


Figure 1.1 – Electronic Gearing for Synchronised Motion. Source: Crane.

The progress of numerically controlled automation systems has resulted in practical manufacturing systems, characterized by a high degree of flexibility and productivity. This in turn has resulted in the establishment of reliable, high quality factory automation systems. Looking into the future, industry will progress towards computerised 'Intelligent Manufacturing Management System' (IMMS)[5]. One of driving forces of this progress is the improved motion control systems which provide one of the crucial building elements of today's automated industrial environments.

1.2 Motion Control Technology, Machine Design & Control

1.2.1 Motion Control Technology

Modern motion control technology has moved a long way since the introduction of power semiconductor devices in the middle of the nineteen fifties [10]. In the course of its dynamic evolution during the last four decades, motion control has grown as a diverse interdisciplinary technology and embraces the areas of power semiconductor devices, converter circuits, electrical machines and fluid/air drives, control theory, and signal electronics. The frontier of this technology has been considerably expanded with the advent of modern powerful micro-computers, VLSI circuits, power integrated circuits, and advanced computer-aided design techniques (see Figure 1.2). Each of the component disciplines is undergoing an evolutionary process, and therefore is contributing to the total advancement of motion control technology.

As a result of the radical evolution of the motion control technology, three essential components for multi-axis motion control, (highly dynamic servo-drives[1], software tools and computer hardware), have been improved considerably.

With continuous improvements of magnetic materials, of cooling and detailed design, servo motors can be built today in AC and in DC versions[10]. To some extent, modern servo drives approach the dynamics of hydraulic/pneumatic servo valves[18]-[24]. These actuators equipped with advanced power converters and modern control techniques produce highly dynamic servo-drives which can achieve high precision motion control.

---

[1] A servo-drive comprises a servomotor, amplifier and controller.

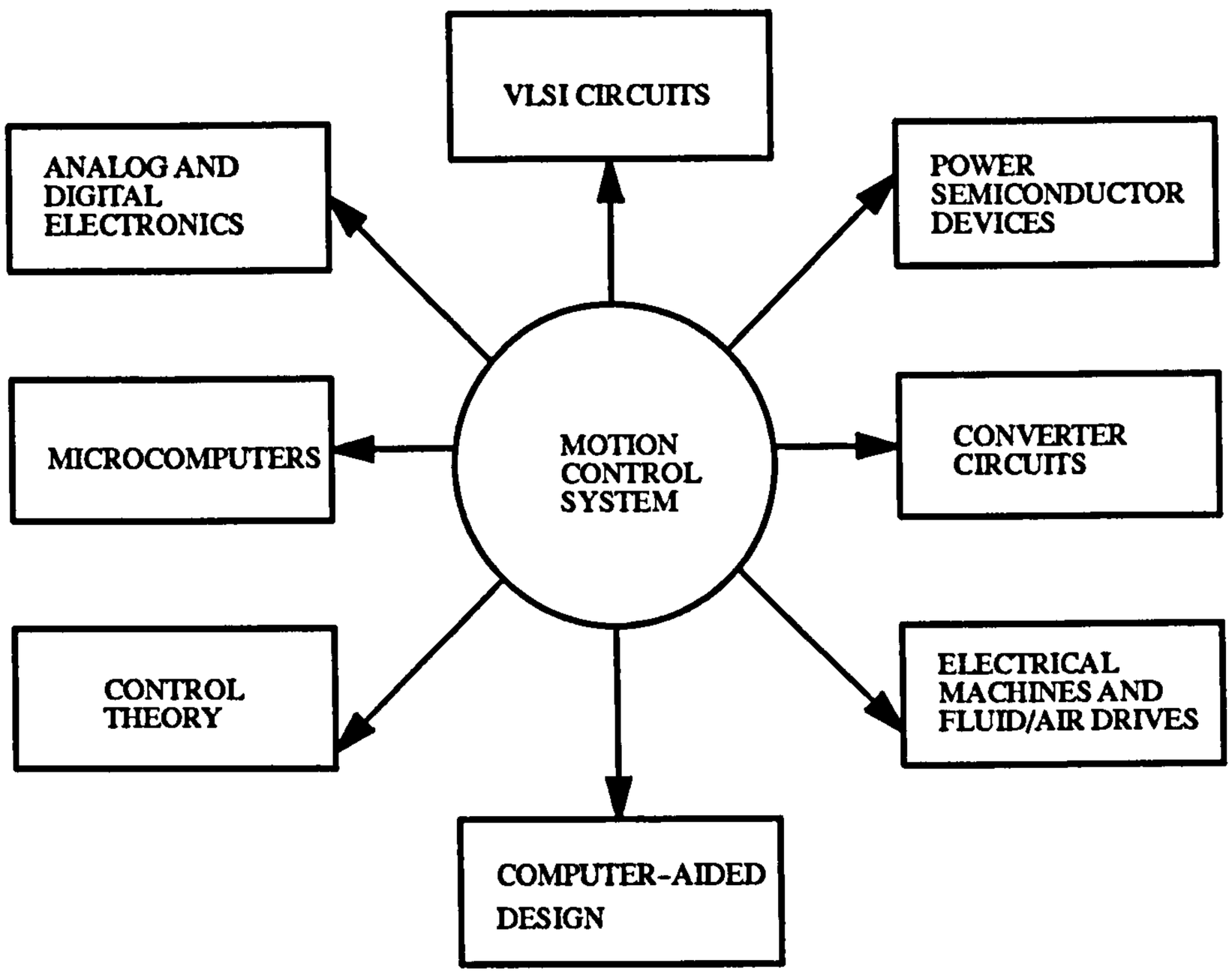


Figure 1.2 Motion control system – an interdisciplinary technology. Source: Bose.

In multi-axis systems, the rules of motion governing the intimate coupling of axes require software tools which allow a fast transformation of mathematical functions. This applies to the design phase as well as to the subsequent operational phase. The necessary mathematics have to be executed frequently in real time, and this calls for controller responses of one millisecond or less. Until recently, experts were of the opinion that assembler programming was the only suitable way to achieve the required short response times. However, there have been important changes in this field: powerful advance language compilers for Pascal or C turn out an essentially more efficient code today, guaranteeing easy maintenance and reliability, although they do not quite come up to the possible efficiency of an expert assembler programmer[16].

Progress in computer hardware by far compensates the present minor disadvantage of advanced languages' lower speed. The processor frequency today ranges from 20 and 30 MHz, math co-processors relieve the main processor during computation-intensive applications, such as for multi-drive systems. The speed of access to the memory chips has kept pace with the speed of the processors[16]. Programmable logic elements and powerful bus systems contribute to actually transfer the modern processors' work

reliably and interference-free from the surrounding noise of highly dynamic drives. When using more advanced microcomputers such as RISC processors, signal processors to motion control, further important developments in the motion control technology may be expected[11]-[15].

1.2.2 A New Approach to Machine Design

Machines with several motions have been designed traditionally according to a philosophy of using mechanisms (such as gears, camshafts, linkages, and belts) linked to a central prime mover to process and combine materials as they move through the machine to produce a finished product. Although there has been continuous development in machine design over the years, with improvements in materials and mechanisms, the underlying design philosophy has not changed. Drawbacks of these mechanical motion controls will limit the extent to which these traditional machine designs can be adapted to meet new requirements, aspects include:

- Extended part changeover time;
- Inaccuracy due to wear;
- Physical time constants which are difficult to overcome; and
- Rudimentary motion profiles.

In recent years, the trends in machine development have been towards higher speed, and more flexible designs[26]. The concept of integrating actuators and sensors with microelectronics to form intelligent modules (or independent drives, independent actuators[21]) also calls for new methods in the area of machine design.

By replacing the central prime mover, cams and mechanical linkages, by a set of independent electro-mechanical drives which deliver power direct to the point of use, flexibility can be introduced. Each software-controlled drive can be programmed to provide tight control of the actuator function (thus replacing the cams) and the drives can be coordinated and synchronised through software (thus replacing the mechanical linkages) [1]. In some instances the independent drives will be used with a mechanism to achieve a specialised motion[25].

The machine built from computer controlled servo-motors can be reconfigured by means of software alone. This can greatly speed the reconfiguration process, and even means that a machine does not have to be stopped to carry out a reconfiguration. The new machine design approach also bring other advantages. The associated reduction in the number of moving parts can lead to an increase in reliability, a reduction in the overall size of the machine, and a reduction in the acoustic noise produced. The machine designer also has more freedom in choosing the layout of the machine, as mechanical links do not have to be accommodated[6].

1.2.3 Motion Control and Synchronisation in Machinery

Motion control problems encountered in modern machinery and automation are often multidimensional: that is, they involve more than one axis[2]. Successful multi-axis machine control requires that the motions of its various parts should be synchronised. Selected examples of motion synchronisation in machines include[7]:

- an x-y table of a machining centre where x- and y- axis must be synchronised;
- tapping where the spindle rotation and the axial feedrate must be synchronised.

In either of the examples listed above, poor synchronisation or coordination result in inferior dimensional accuracy or unusable product. Each of the examples introduces unique coordination problems. The synchronisation involved in x-y motion control problems represents perhaps the simplest case since the dynamics of the x-axis and the y-axis are not strongly coupled and are similar to each other. The synchronisation problem in tapping is more complicated. Reasons for this include: 1) there exists a significant difference between the dynamics of spindle rotational motion and that of spindle feed motion; 2) tapping requires synchronisation of transients such as the simultaneous reversal of spindle rotation and feed; 3) there exists a certain dynamic interaction between the spindle rotational motion and feed motion.

Other applications require motors to run interactively[17]. For example, in the glass industry multiple motors are used to roll sheet glass. Here the motor speeds have to be constant and equal to prevent stretching of the glass. In the textile industry fabrics are

---

[2] In the motion control lexicon, an "axis" is a single servo motion -- linear or rotary.

wound from one reel to another, requiring careful regulation of the fabric tension to prevent damage. Similarly many printing system drives have load sharing requirements.

Generally, in manufacturing machinery, motion synchronisation can be specified as follows:

- Velocity synchronisation -- preserves the velocity ratio between the axes. Used in metal forming, synthetic fibre manufacture, rubber processing and non-woren fabric production etc.
- Position synchronisation (or phase synchronisation) -- preserves the position relationship between the axes. Used in machining, glass bottle production, and in the paper industry, etc.
- Event synchronisation -- enables the slaves to start the motion or change their speeds when the master position equals any specified value. Applied to feed systems for packaging materials, etc.
- Web synchronisation -- is a special class of applications where several motions must be synchronised, however, the ratio between the various speeds vary continuously. Applied to high performance magnetic tape drives, wire-drawing machines, web tension control and more generally, in situations where drives have to be linked in a way which cannot be easily determined analytically, etc.

1.2.4 Motion Coordination and Synchronisation by Software Mechanisms

When mechanically linked systems are replaced by independent drives, the drives must be forced into coordination and synchronisation by an appropriate control regime. For example, in the case of software-based solutions, the synchronisation will be implemented by programming: hence the generic term of software mechanism. Specific applications may be termed software gearboxes, software cams [8] and so on.

There are many different approaches that may be used in the design of a software mechanism[3][6][8][16][17]. However, they tend to share a number of common features, which can be grouped together within two hierarchic of levels [3].

- **Level 1 – the lower or axis level**

For each axis of the mechanism, there is an actuator together with a controller. Each axis/controller system within this level communicates with the higher level, level 2, and not with other axes within level 1.

- **Level 2 – the higher or co-ordination level**

The nature, or function of the mechanism is programmed in this level; for example whether a gearbox is to be emulated, or a cam and cam follower, or perhaps some arbitrary position relationship between the axes is to be satisfied. Given that reference commands are suitable for use with the proposed controllers, the controller gains are calculated (within level 1) according to the desired response of each closed–loop axis system. Level 2 generates the reference signal for each axis in order to meet the synchronisation requirements.

* The overall system performance is not only dependent on the lower level servos, but also determined by the software mechanism. Due to the absence of solid mechanical elements to physically link motions, precise coordination and synchronisation of the interactive motions is one of the challenges which the new design philosophy raises. Much research work has focused on this problem[3][6][7][9][15][17]. The control requirements for such systems are becoming clearer. This thesis describes an alternative solution to these control problems which is believed to lead to improved machine control.

1.3 Research Objectives

The principal objective of this research is to create a multi–axis motion control and synchronisation scheme to provide tight control within a flexible structure for a machine control system with multiple drives. In order to meet this objective, the following secondary objectives were established.

1). To investigate building a multi-axis motion control system

A multi-axis motion control system has been built based on the new machine design philosophy by using mechanical modules (containing electric motors, pneumatic actuators, sensors, power amplifiers), embedded controllers and a system controller.

2). To develop a new control strategy for achieving improved motion synchronisation.

A fuzzy logic control algorithm was proposed to cross-couple independent servo-drives to make them act in a dependent manner to release improved motion synchronisation.

3). To develop a method for multi-axis motion control system design

A design method based on intelligent on-line correction/coordination was developed, which provides tight motion synchronisation.

4). To form real-time multi-axis executable modules within a hierarchical control model.

A set of real-time execution modules which illustrate the design method has been defined and implemented.

1.4 Thesis Organization

This thesis is presented as a series of research tasks. Chapter 1 provides a general background to the need for motion synchronisation for multi-axis control systems, and outlines the research objective. The second task, presented in Chapter 2, appraises basic servo-control algorithms used in motion control. The appraisal of servo-controllers includes: (1) their abilities to eliminate different error sources, and (2) their limitations in motion control. It has been found that a combination of different algorithms are needed to eliminate all errors.

Chapter 3 reviews conventional multi-axis drive system control structures and software mechanisms for achieving coordinated and synchronised motion. These

application–specific methods for motion synchronisation are briefly described, and their advantages and limitations briefly summarised.

A totally new control strategy for multi–axis motion synchronisation is proposed in Chapter 4. It based on a synthesis of the methods reviewed in the previous chapter. A generic 'dual closed–loop' control structure is introduced for machine control. A fuzzy logic coupling algorithm is used to coordinate the servo–drives. The fuzzy logic synchronisation methods are analysed with respect to their characteristics and stability.

In Chapter 5, an Intelligent Motion Control (IMC) design method is developed which embodies intelligent motion synchronisation software mechanism for the multi–axis servo–drives.

Chapter 6 shows the implementation of IMC elements including hardware and software. The simulations and experimental verification are carried out in a simplified IMC testbed and the results are given in Chapter 7. The results confirm the efficiency of the fuzzy logic coupling algorithm.

Chapter 8 to Chapter 9 provide a conclusion to the research work. The evaluation of the IMC method and the possible future extensions to this method are presented In Chapter 8. Chapter 9 briefly summarises the work undertaken and the contribution to new knowledge.

Chapter One: References

[1] Griffin, J. M., "Programmable Motion Control for High Production", Control Engineering, March 1990, pp 96–97.

[2] Hart, I. B., "the world of Leonardo da Vinci", MacDonald & Co., 1961.

[3] Jenkinson M., "The synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

[4] Dunne, E. J., "Look to Motion Control for Manufacturing Solutions", Design News /8–5–91/, pp45–48.

[5] Kusiak, A., "Intelligent Manufacturing Systems", Prentice Hall, 1990.

[6] Danbury, R., Jenkinson, M., "Synchronised servomechanisms – the scalar–field approach", IEE Proc.–Control Theory Appl., Vol. 141, No. 4, July 1994, pp261–273.

[7] Hu, J., Chiu, T. and Tomizuka, M., "On Motion Synchronization of Two Axes Systems", Monitoring and Control for Manufacturing Processes, ASME Conf, 1990, pp267–282.

[8] Crane, J., "Electronic Gearing for Synchronised Motion", Drives & Controls, July/August 1993, pp30–31.

[9] Lo, C. C. "Cross–Coupling Control of Multi–axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

[10] Bose, B. K., "Trends in Motion Control Technology", IETE Technical Review, Vol. 4, No. 8, 1987, pp305–329.

[11] Meshkat, S., "Digital signal Processors Provide Advanced Motion Control", Motion Control, PCIM, December 1987, pp42–75.

[12] Dimitri, D. S., "DSP–Based Motion Controllers Are Moving", Industry Trends, PCIM, May 1991, p30.

[13] Meshkat, S., "Parallel Processing in the Next Generation of Controllers", PCIM, September 1993, pp54–59.

[14] Meshkat, S., "Quantifying System Performance vs DSP MIPS – It's About Time", PCIM, May 1994, pp34–47.

[15] Meshkat, S., "Parallel DSPs Excel in CAM and Gearing Applications", PCIM, February 1994, pp69–73.

[16] Manfred Binder, "Electronic Couplings–Replacement of Mechanical Gears?", PCIM Europe, March/April 1992, pp60–63.

[17] Clarkson, J. C., "New Strategy Improves Multi–Motor Control", Drives & Controls, December 1989/January 1990, pp43–44.

[18] Pu, J., Moore, P. R., Weston, R. H., and Chen, C., "Profile Planning in Digital Motion Control of Servo Pneumatic Drives", Fluid Power Systems and Technology Symposia ,1991 Winter Annual Meeting, USA.

[19] Moore, P. R., Pu, J., Harrison, R., Weston, R. H., and Chen, C., "A General Purpose Digital Motion Controller for Fluid Power Systems", Bath International Fluid Power Workshop, 19–20 September 1991, UK.

[21] Virvalo, T. and Puusaari, P., "New solution of motion control problem in mechatronic system", Computers Elect. Engng, Vol. 18, No. 1, 1992, pp 41–50.

[22] Nevala, K., Lahdenpera, M. and Lammasniemi, J., "An intelligent hydraulic actuator with an integrated electronic control unit", Proc Int conf on Motion Control, The Mechatronics Approach (Edited by Prof H Van Brussel) Antwerp 1989.

[23] "Servo valve becomes digital actuator", Control Eng Mag, June 1986.

[24] Virvalo, T., "Distributed motion control in hydraulics and pneumatics", Mechatronics Vol.2 .No.3. pp 277–288, 1992.

[25] "Communicating roller drivs provide varible feed rate", Eureka on Campus, Spring 1992, p9.

[26] Gillbe, I., "An Intelligent Look at Machine Motion Control", Design Engineering, February 1993.

CHAPTER TWO
Appraisal of Servo-Controllers for Motion Control

2.1 Introduction

Using servo-drives to build a modern manufacturing machine system to achieve high performance (precision and speed) and flexible operation, accurate servo-controllers are needed. This chapter summarises existing servo-controllers for motion control applications and presents an appraisal of these controllers. The appraisal of servo-controllers includes: (1) their abilities to eliminate different error sources, and (2) their practical limitations in a multi-axis motion control scheme.

2.1.1 Error Sources which Affect Motion Control and Synchronisation

Industrial automation spans a huge spectrum of complexity in terms of both the physical structure of machines and the tasks which they perform. However, the performance of the computer controlled machine is mainly dependent upon:

(1) mechanical hardware design (e.g. transmission types, bearings etc.)[2][6][12][24];

(2) process effects (e.g. loading, tool wear, vibration etc..)[16]; and

(3) control system (controllers and drive dynamics)[3][9][34].

Items one and two are outside the scope of this thesis. The research work focuses on the control system, with an emphasis on motion control and synchronisation. In order to deliver motions in a timely manner and coordinated with other machine functions synchronisation of the motions has to be maintained. However, error sources from the controller, drive dynamics and external disturbances will affect the motion synchronisation of the multiple axes. These error sources can be further classified into three categories[3][7][8][34] and are illustrated in Appendix I.

(i) Parameter Mismatch

Parameter mismatch means different change in the respective transfer function, which can result in different changes of output of the respective closed-loop systems.

Therefore, the different output changes of the servo-loops will generate synchronisation errors.

(ii) Disturbances

A disturbance is used here to mean an external action to the loop which changes or disturbs the operation of the controlled variable. Therefore, disturbances (such as frictions, load change etc.) will influence the motion synchronisation.

(iii) Reference Commands

A reference input may result in a steady-state error when it is fed into a particular type of control system[7]. The steady-state error is the error once the transient response has decayed leaving only the continuous response.

In a multi-axis system, for each servo the reference input may be changed from one value to another value from time to time, and reference commands for different servo systems will be different at each update in order to keep a defined relationship.

It is very difficult to have the reference input always match the system type. Therefore, steady-state errors will exist in each servo loop and for multi-axis system, the error value for each servo-loop is different as such synchronisation errors will exist.

Such error sources can be reduced or eliminated by improved design of the servo-control algorithms, and this is the main concern of this chapter. In the following sections, different types of servo control systems for the motion control are briefly appraised.

2.2 Feedback Controller

The term "feedback controller" here means a controller that uses only basic feedback principles. In motion control these controllers may be classified into three basic classes: P, PID and state-feedback controllers.

2.2.1 P Controller

Proportional (P) controller is the most conventional feedback controller in CNC[34]. It sends correction signals proportional to the difference between the reference position and the actual position. The proportional gain, is usually designed so that the closed loop damping ratio (ζ_{closed}) is equal to 0.707. For this damping ratio, the following equation can be used for the selection of the proportional open-loop gain [18].

$$K_{open} = \frac{1}{T + 2\tau} \qquad (2\text{-}1)$$

Where T is the sampling period. τ is the open-loop time constant, and K_{open} is the open-loop gain which is the product of the P-controller gain (K_p) and the other gains in the system such as motor's gain (K_m), gear ratio (K_g), encoder gain (K_e), etc.

2.2.2 PID Controller

In a PID controller the correction signal is a combination of three components: a proportional, an integral and a derivative of the position error. PID controllers use feedback:

- Correct the controlled variable, in a manner proportional to the error;
- Eliminate steady state errors and reject the external disturbances, through operation of the integral term;
- Anticipate major changes in the error and improve system damping using a derivative term.

Implementing an I-controller by itself will cause instability, and it must be combined with a proportional action to enable a stable system. The derivative (D) controller aids in shaping the dynamic response of the system. The combination is known as a PID controller. Since a computer is utilized as the controller, a digital PID is implemented.

There are different ways to design digital PID controllers. We can, for example, formulate the digital PID controller law by approximating the continuous-time PID controller

with backward difference of Euler's or Tustin's methods [1]. In the following analyses, the PID controller law H(z) is formulated based on backward difference approximation, using z–transforms.

$$H(z) = K_p + K_I \frac{T_z}{z-1} + K_D \frac{z-1}{T_z} \qquad (2\text{–}2)$$

Where K_p, K_I, K_D, are the proportional, integral, and derivative gains, respectively. The integral gain K_I is chosen large enough for good disturbance rejection, and the derivative gain K_D is designed to guarantee small overshoot. Usually, the controller gains can be designed based on root locus or frequency domain methods.

The PID controller is applied successfully to most control problems in process control, electrical drive systems and servo mechanisms. The reason for its success is that most of these processes have a dynamic behaviour that can be adequately approximated by a second–order process. The PID controller is insufficient to control processes with additional complexities such as[8]

- time delays;
- significant oscillatory behaviour (complex poles with small damping);
- parameter variations; and
- multiple–input multiple–output systems.

In addition to the basic PID shown in Figure 2.1a , some different PID–structures, shown in Figures 2.1b, 2.1c and 2.1d, were applied in practical servo–controllers[1][22][27] [28][33]. However, since Figure 2.1a represents the most common structure, it is used in the analyses in this chapter.

2.2.3 State–Feedback Controller

The most common measurable states in motion control are the position and the velocity. The axial position is commonly measured by an incremental encoder and associated counter, and the velocity may be measured either by a tachometer [19][22], phase

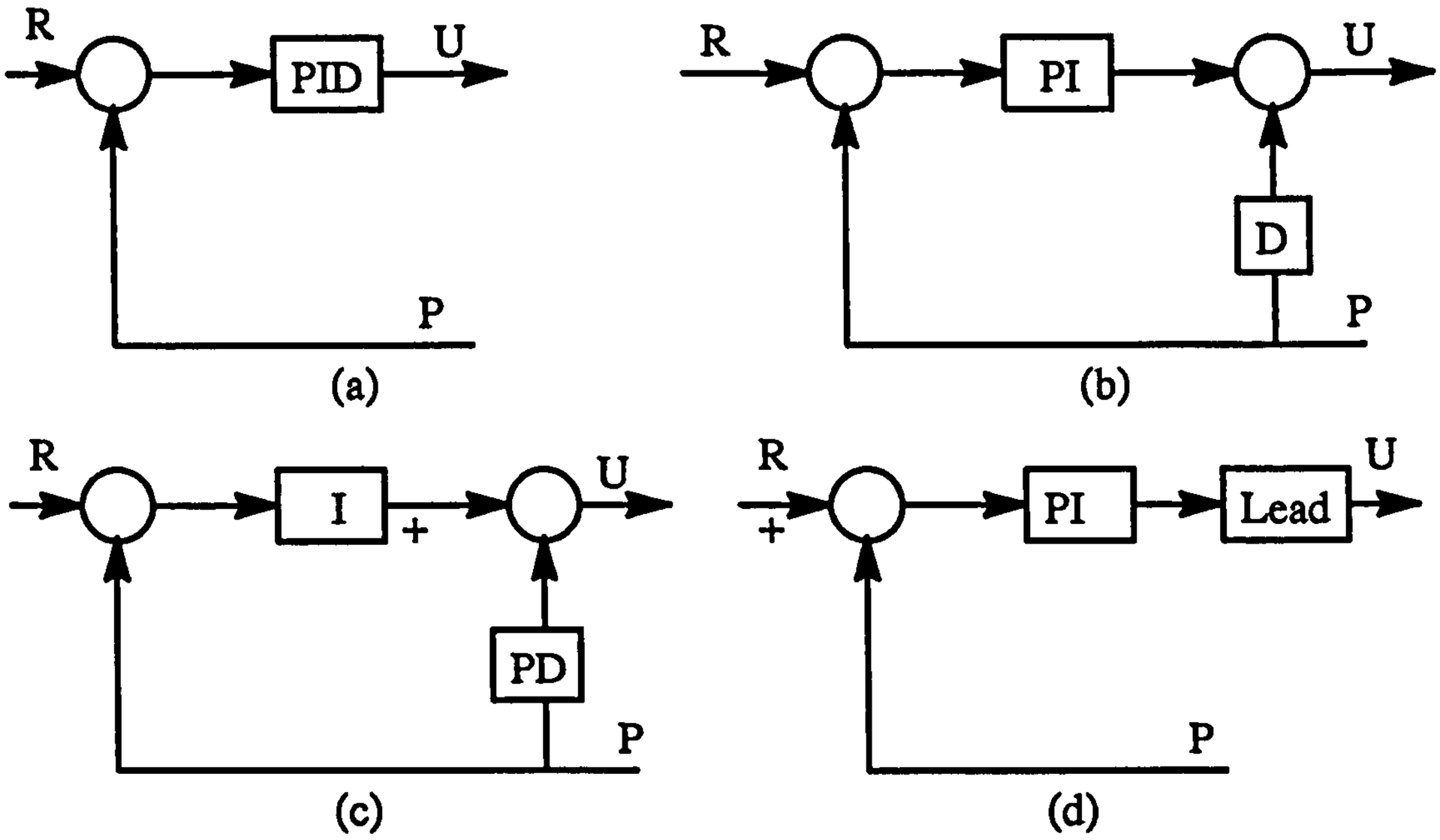


Figure 2.1 Different ways to implement controllers with PID functions. Source: Lo.

measurement of pulse trains[9], or by differentiating the measured position [28][33]. If the feedback states are the position and the velocity, the state-feedback controller is in fact a PD-controller (shown in Figure 2.1b without the integral action).

The major drawback of the state-feedback controller is that it is relatively poor in eliminating the steady-state errors and rejecting external disturbances[34]. The state-feedback controller, however, is similar to PD controller. It will not be discussed further in this chapter.

2.3 Feedforward Controllers

Feedforward controllers have a feedforward term aimed at minimizing the position error during the move. The method is based on anticipating the required motion command signal and providing it as a bias signal. Some approaches to feedforward control are described.

2.3.1 Basic Feedforward Controllers

There are two principal types of feedforward controllers, that are shown in Figure 2.2. The principle of the design in Figure 2.2a[15][17][29][30][31] is simple: Implement in the control computer a transfer function that is the exact inverse of the one of the real control loop, G(z), ie, $G_0^{-1}(z)G(z) = 1$, and then the actual position becomes equal to the required position.

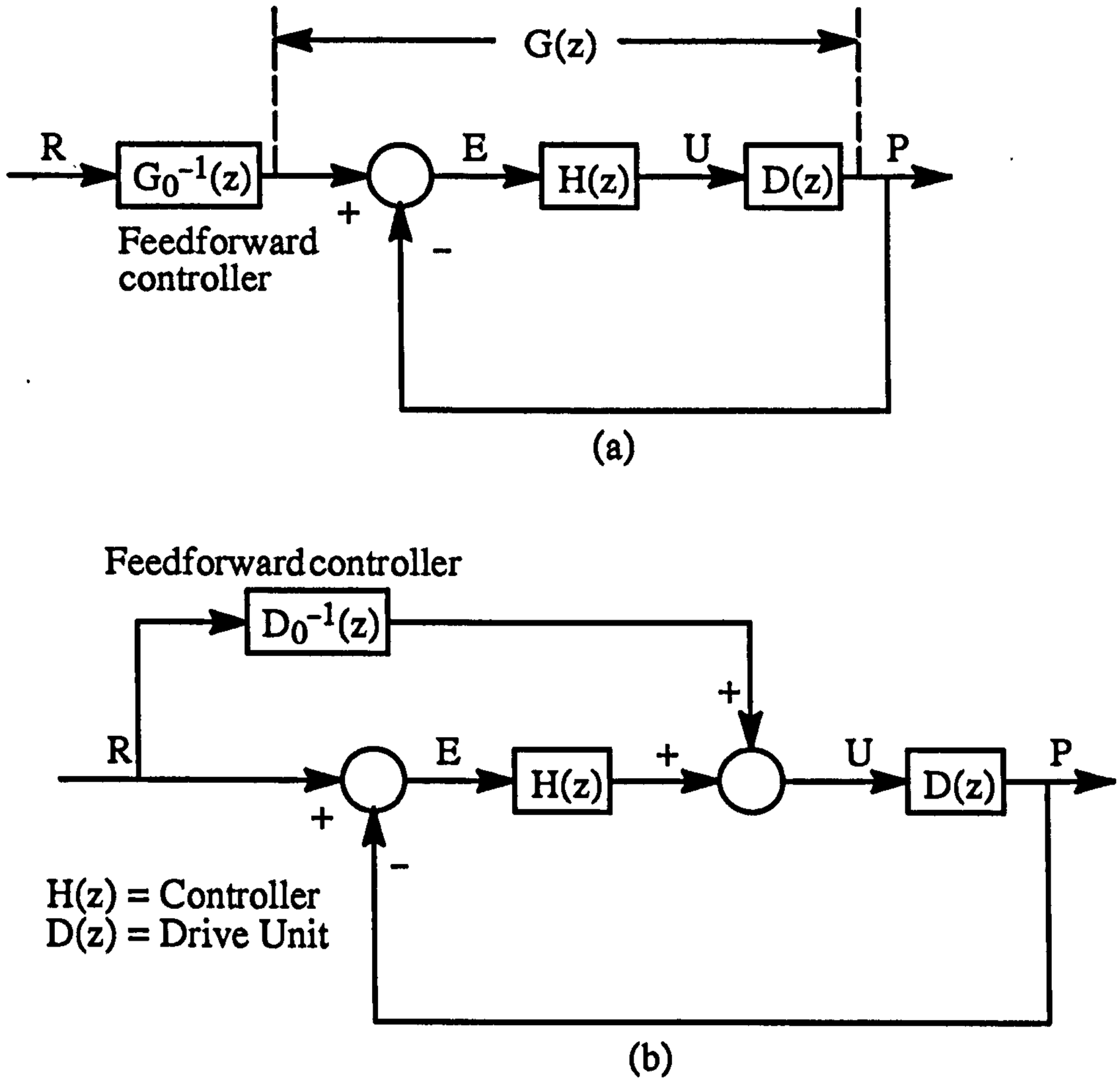


Figure 2.2 Two principle types of feedforward controller. Source: Lo.

The design in Figure 2.2b has the same objective[20][23][25][26]. If we implement an inverse transfer function of the drive unit in the feedforward controller block, as shown in Figure 2.2b, we obtain the following closed-loop equation:

$$\frac{P}{R}(z) = \frac{D_0^{-1}(z)D(z) + H(z)D(z)}{1 + H(z)D(z)} \qquad (2\text{–}3)$$

Where H(z) and D(z) represent the transfer functions of the software controller and the drive unit, respectively. If $D_0(z) = D(z)$, the overall relation between the required position and the actual position becomes 1 : 1. Clearly, the ideal condition cannot be implemented for two reasons. First, it may be impossible to precisely implement the inverse of the function D(z). Secondly, the function D(z) varies with load conditions. Therefore, H(z) should be properly designed to provide good performance.

There are some differences between these two types of feedforward controllers.

(1) The first feedforward controller is the inverse of the feedback control loop, that consists of the controller and the drive and therefore it becomes more complicated if a more comprehensive controller (eg, PID controller) is utilized. Whereas, the later is the the inverse of the drive only, and therefore, the design of its corresponding feedforward controller is simple and independent of the design of the feedback controller.

(2) If the feedforward controller ($G_0^{-1}(z)$ in Figure 2.2a or $D_0^{-1}(z)$ in Figure 2.2b) includes poles located on or outside the unit circle, the design of the feedforward controller must be modified. The modification of the design in Figure 2.2a is easier than that in Figure 2.2b.

Feedforward control minimizes following error and so reduces motion time. However, feedforward control is an open-loop design method which relies on knowledge of invariant system parameters. If the system parameters vary, it is better to use a closed-loop design method, such as robust control.

2.3.2 Zero Phase Error Tracking Controller

A feedforward controller entitled "Zero Phase Error Tracking Controller (ZPETC)" was proposed by Tomizuka[30]. The concept of the ZPETC is based on pole/zero cancellation, ie $G_0^{-1}(z)G(z) = 1$. However, if $G_0^{-1}(z)$ includes unstable poles, it cannot be implemented as a feedforward controller, and therefore must be modified. Let us assume that the closed-loop discrete-time transfer function, which includes the controlled plant and feedback controller, is expressed as:

$$G_{closed}(z^{-1}) = \frac{z^{-d}B^{+}(z^{-1})B^{-}(z^{-1})}{A(z^{-1})} \qquad (2\text{–}4)$$

where Z^{-d} represents a delay of **d** steps (sampling periods) normally caused by the control loop, **A** includes the closed-loop poles, B^+ includes the acceptable closed-loop zeros, and B^- includes the unacceptable closed-loop zeros. The "acceptable" zeros here mean the zeros that are located inside the unit circle, and can be taken as the poles in the feedforward controller. By contrast, unacceptable zeros are located on or outside the unit circle, and cannot be the poles of the feedforward controller since they will cause instability. In practical applications, a zero that is close to the unit circle and is located on or close to the negative real axis (eg, z = –0.97) may introduce an oscillatory mode and, therefore, is regarded as an unacceptable zero.

In unacceptable zeros exist, $G_0^{-1}(z)$ cannot be implemented in the feedforward controller because it will cause a significant oscillation in the control signals. Tomizuka modified the feedforward controller structure as shown in Figure 2.3. The modified feedforward controller $G_f(z)$ has the following form:

$$G_f(z)= \frac{A(z^{-1})\,B^-(z)}{B^+(z^{-1})[B^1(1)]^2} \qquad (2\text{–}5)$$

Multiplying Equations 2–4 and 2–5, yields the overall transfer function

$$\frac{P(k)}{R(k+d)} = \frac{B^-(z)\,B^-(z^{-1})}{[B^-(1)]^2} \qquad (2\text{–}6)$$

The frequency transfer function is given by

$$\frac{P(k)}{R(k+d)} = \left[\frac{B^-(e^{j\omega})}{B^-(1)}\right]\left[\frac{B^-(e^{j\omega})}{B^-(1)}\right] \qquad (2\text{–}7)$$

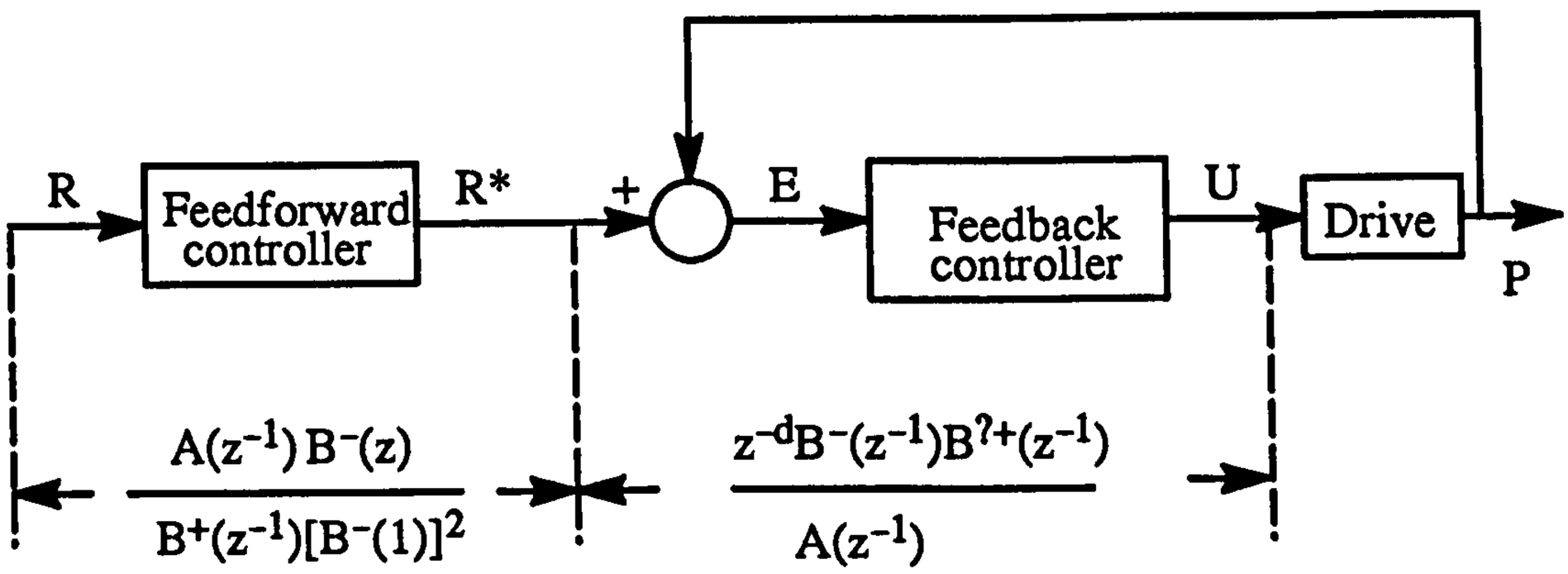


Figure 2.3 Zero phase error tracking control system.
Source: Tomizuka.

Suppose that $B^{-}(e^{-jw}) = Re + jIm$, where Re and Im represent the real and imaginary parts respectively, the frequency transfer function becomes:

$$\left[\frac{B^{-}(e^{-j\omega})}{B^{-}(1)}\right]\left[\frac{B^{-}(e^{j\omega})}{B^{-}(1)}\right] = \frac{1}{[B^{-}(1)]^{2}}[(Re + jIm)(Re - jIm)]$$

$$= \frac{Re^{2} + Im^{2}}{[B^{-}(1)]^{2}}$$

$$= \left\| \frac{B^{-}(e^{j\omega})}{B^{-}(1)} \right\|^{2} \qquad (2\text{–}8)$$

As shown in Equation 2–8, the phase angle of the frequency transfer function is zero. Therefore, zero phase error tracking is provided by the ZPETC method.

The major drawback of the ZPETC method is that it requires precise knowledge of the dynamic behaviour of the axial drive system. However, there might be a difference between the real drive system and the model used in the computer (ie, modeling error), and therefore an additional error source is introduced to the controlled system. Another drawback is that the inverse transfer function in the feedforward controller will cause large control signals. These signals, in practice, will be limited by the permissible

maximum output of the digital-to-analog converter and the maximum voltage of the motor, and therefore the performance of the controlled system is often degraded.

2.4 Other Algorithms

In addition to the two basic classes discussed above, there are other control algorithms that are discussed in this section.

2.4.1 Robust Control

The PID design method described previously is not very effective when the transfer function of the motor and the load includes pairs of complex poles with low damping, like the ones caused by torsional resonances. In that case, high gain causes instability while low gain produces a slow response. Robust control is a special design method aimed at achieving uniform system response in spite of such parameter changes[2].

Robust control addresses this situation by providing a compensation filter of a more general form. The compensation may be, for example, a notch filter, consisting of complex zeros which effectively cancel the complex poles of the motor and load, allowing higher gains in the system.

Robust control is often implemented by several control loops, where the inner loop is aimed at modifying the behaviour of the system and the outer loop controls the position [39].

2.4.2 Optimal Control

The optimal control algorithm can be applied as an efficient control law for servo-controllers[21]. The principle of optimal control is maximisation of a performance index subject to the system differential equations. Optimal control has a closed-form solution for a linear system and a quadratic performance index. In motion control applications if the performance index includes the axial position errors (and their incremental changes), the servo-controller becomes a feedback controller[1]. If the synchronisation error is used in the performance index, the servo controller then becomes an optimal cross-coupling controller[21].

2.4.3 Predictive Control

The basic idea of Generalized Predictive Control (GPC) proposed by Boucher [11] is to make the plant's predicted output coincide with a setpoint or desired known trajectory. Three stages are performed to achieve this goal: (1) first the plant output is predicted; (2) then the future control values which minimize the errors between the predicted outputs and the setpoint values are calculated; and (3) finally the first optimal control value is applied. The process is iterative and normally repeated every sampling period.

The GPC method, however, can be regarded as a particular case of optimal control, in which the performance index includes the errors between the predicted outputs and the desired values[11].

2.4.4 Adaptive Control

Adaptive control adjusts the compensation filter in real time in order to correct for parameter variations. Two methods of adaptive control are the model reference adaptive control (MRAC) and the self-tuning regulator (STR) shown in the block diagrams of Figures 2.4 and 2.5[2].

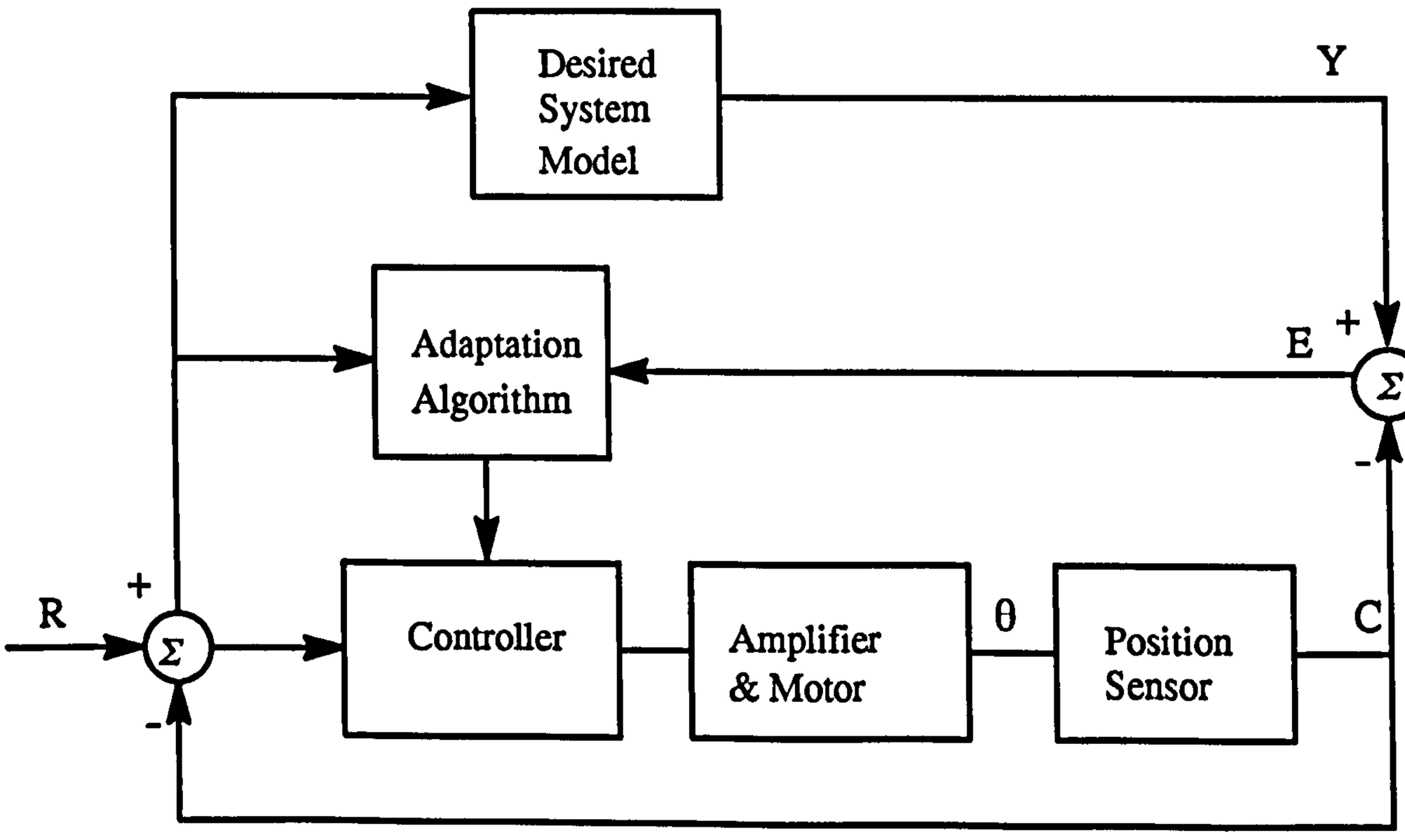


Figure 2.4 Model Reference Adaptive Control

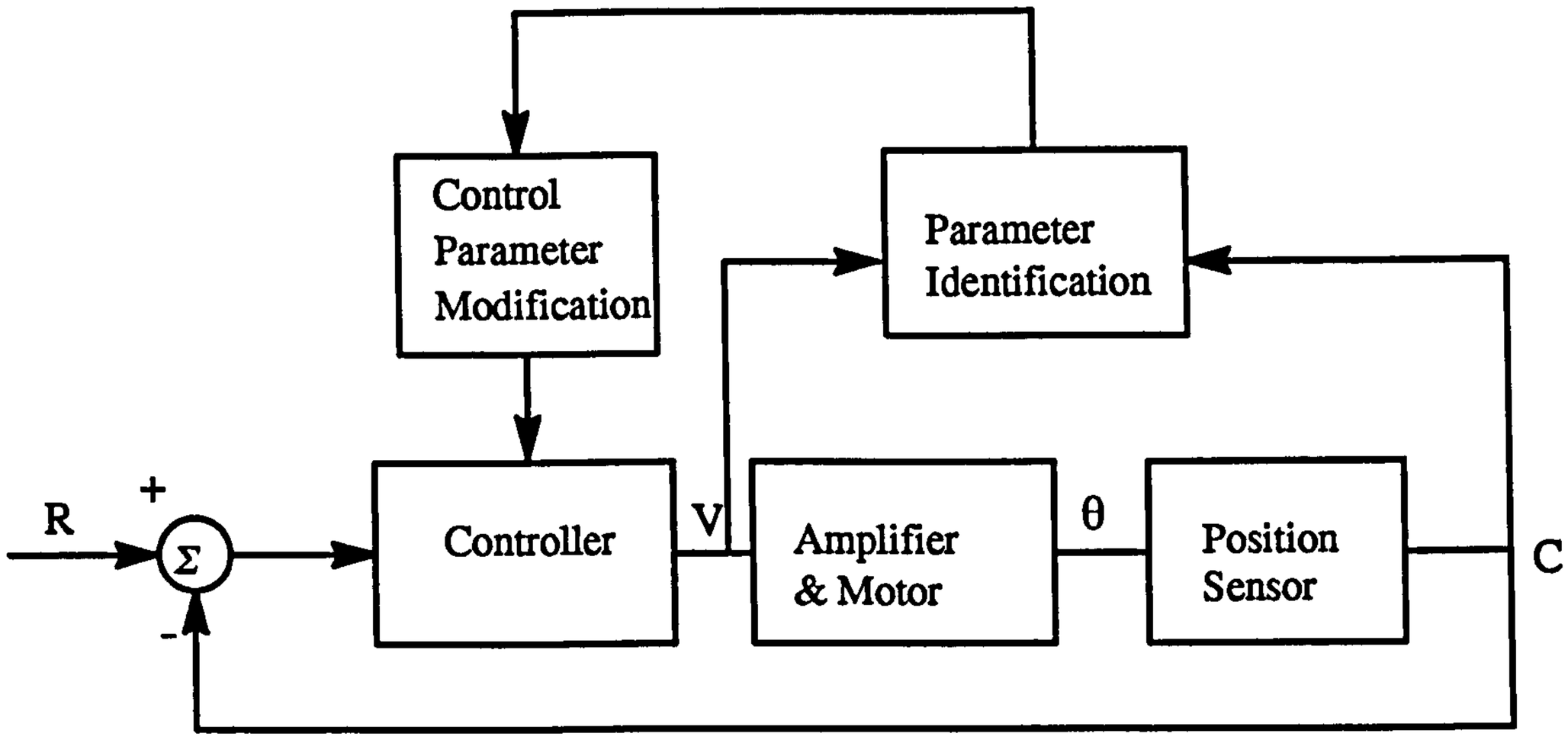


Figure 2.5 Self-Tuning Regulator

The STR divides the adaptive control into two tasks, parameter identification and control modification. The identification algorithm estimates system parameters such as load inertia, friction, etc, in real time. The estimated results are used to modify the compensation parameters including the system gain, zero, etc.

The MRAC aims is to produce the same overall transfer function regardless of the parameter variations. To achieve this, the desired servo model, H(s), is specified and its desired output **Y** is computed. The controller then compares the actual output, **C**, with **Y** to determine the error **E**. On the basis of the information available, the controller modifies the compensation so the estimation error **E** decreases, zero estimation error implies that the system response follows the desired model perfectly.

Although the two design approaches produces similar results, the STR has several advantages over the MRAC, when applied to motion control systems. First, the adaptation algorithm of the MRAC is a nonlinear feedback system that is quite difficult to design. In contrast, the STR divides the design into two smaller tasks of identification and modification that can be performed more readily[2].

The MRAC requirement to keep the system transfer function the same for all parameter values is quite restrictive[2]. It is unreasonable and ill-advised to produce the same

transfer function when the load inertia varies over a 100:1 range. The STR provides such flexibility, and therefore, may be viewed as more general than the MRAC.

The feedforward controller is very sensitive to modeling errors and the variations in system parameters, and therefore, an adaptive algorithm is required if the feedforward control approach is applied to more complex motion control systems[29][30].

An adaptive control algorithm can be applied to a mismatched systems[35]. This adaptive algorithm makes all the individual axial control loops match artificially, and thereby eliminates the synchronisation error caused by mismatched loop parameters.

In addition, the adaptive control algorithm can also be used as an assistant or parallel algorithm for the cross-coupling controller[14].

2.4.5 Sliding Mode or Variable Structure Control

Basically, sliding mode control is an adaptive model-referencing control (MRAC), but is easier to implement by microcomputer than the conventional MRAC system. The sliding mode control is ideally suitable for position servo, such as robot and machine tool drives, where problems related to mechanical inertia variation and load disturbance effect can be eliminated. The control can be extended to multiple drives where close speed or position tracking is desired[37]. In sliding mode control, the "reference model" or a predefined trajectory in the phase plane is stored in microcomputer, and the drive system is forced to follow or "slide" along the trajectory by a switching control algorithm, irrespective of plant parameter variation and load torque disturbance. The microcomputer detects the deviation of the actual trajectory from the reference trajectory and correspondingly changes the switching topology to restore tracking.

The disadvantage of this method is the drastic changes of the manipulated variable. However, this can be avoided by a small modification: a boundary layer is introduced near the switching line which smooths out the control behaviour and ensures the states remaining within the layer. Given that the upper bounds of the model uncertainties etc. are known, stability and high performance of the controlled system are guaranteed[10].

2.4.6 Repetitive Control

In addition to the general servo-controllers, a special purpose controller entitled the "repetitive" controller may be considered to handle the repetitive tasks with the presence of periodic references and periodic disturbance inputs. A repetitive controller was proposed and applied on a disk-drive system and a turning machine[13][29][32]. When utilizing a repetitive controller, for example, the axial errors measured when cutting a part are used to compensate the errors in the next part. This algorithm must be applied together with a conventional P-controller[13] or other servo-control algorithms (eg, adaptive ZPETC in[29]). It cannot be applied independently, and can only be applied to repetitive tasks.

2.5 Conclusions

According to the previous analysis, a comparison of the two basic servo-controllers is surmised in Table 2.1.

Table 2.1 The Evaluation of Servo-controllers[34]

	P control	PID control	Feedforward (with P-controllers)
Tracking nonlinear trajectories	Fair	Fair (low speed) Poor (high speed)	Excellent (1) Fair (2)
Axis mismatch	Fair	Good	Excellent (1) Fair (2)
Disturbances	Poor	Good	Poor
Special problems		Overshoot at stopping	Performance is sensitive to modeling error and saturation

Notes:

1. Assume no difference between theoretical model and real system.

2. Assume 2% difference between theoretical model and real system.

Grading: Excellent, Good, Fair, Poor.

Based on the comparison, the selection of servo-controllers for different motion control applications suggests in the following:

(1) The P controller works well only with low friction, small loading, small mismatch in axial parameters, and low speed operation.

(2) The PID controller has a good disturbance rejection ability and is more robust to mismatched axial parameters. Its drawbacks are poor tracking ability for nonlinear trajectory, and in addition, it may result in overshoot. Therefore, the PID controller is preferred on low-speed machines. Usually, a deceleration is needed at the end of every trajectory segment in order to avoid the problem of overshoot. This increases the total motion time.

(3) The ZPETC is preferred for high-speed motion control when the system model is well known and no varying or nonlinear characteristics exist. High disturbance loads are not allowed with these methods unless the feedback loop already provides an algorithm for good disturbance rejection.

In addition to the basic control approaches, some other modified methods may be also considered as algorithms for servo-controllers. They are summarised below:

(1) An adaptive algorithm can be added to a ZPETC method to cope with the modelling errors and parameter variation. The adaptive algorithm may be limited by the richness of the input signals. Therefore, a careful design of the adaptive algorithm is required if the closed-loop system includes higher order structures.

(2) A combined algorithm of the PID control and ZPETC may have the ability to reject all error sources (see Table 2.1), because of the ZPETC ability in trajectory tracking and the PID control in disturbance rejection.

(3) A Kalman filter is an optimal stochastic linear adaptive controller. However it requires an explicit mathematical model of how control outputs depend on control inputs[36]. If the mathematical model can be developed, the Kalman filter can provide excellent performance for motion control.

Chapter Two: References

[1] Astorm, K. J. and Wittenmark, B., "Adaptive Control", Addison-Wesley Publishing Company, 1989.

[2] Tal, J., "Motion Control Applications", 1989.

[3] Jenkinson, M., "The Synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

[4] Dorf, R. C., "Modern Control System", Fifth Edition, Addison-Wesley, 1989.

[5] Kusiak, A., "Intelligent Manufacturing Systems", Prentice Hall, 1990.

[6] Dunne, E. J., "Look to Motion Control for Manufacturing Solutions", Design News /8-5-91/, pp45-48.

[7] Phillips, C. L. and Harbor, R. D., "Feedback Control Systems", Prentice-Hall International Editions, 1991.

[8] Olsson, G. and Piani, G., "Computer Systems for Automation and Control", Prentice Hall, 1992.

[9] Bollinger, J. G. and Duffie, N. A., "Computer Control of Machines and Processes", Addison-Wesley Publishing Company, 1988.

[10] Bose, B. K., "Sliding mode control of induction motor", IEEE/IAS Annu. Meet. Conf. Rec., 1985, pp479-486.

[11] Boucher, P., Dumur, D. and Rahmani, K., "Generalized Predictive Cascade Control (GPCC) for Machine Tool Drives", Annals of the CIRP, Vol 39/1/90, August, pp357-360.

[12] Chen, J. S., "Real-Time Compensation for Time-Variant Volumetric Error on a Machining Centre", PhD Thesis, The University of Michigan, Ann Arbor Michigan, 1991.

[13] Chew, K. K. and Tomizuka, M., "Steady-State and Stochastic Performance of a Modified Discrete-Time Prototype Repetitive Controller", ASME Transaction, Journal of Dynamic Systems, Measurement and Control, Vol. 112, March 1990, pp35-41.

[14] Chuang, H. Y. and Liu, C. H., "Cross-Coupled Adaptive Feedrate Control for Multiaxis Machine Tools", ASME Transaction, Journal of Dynamic Systems, Measurement and Control, Vol. 113, September 1991, pp451-457.

[15] Haack, B. and Tomizuka, M., "The Effect of Adding Zeros to Feedforward Controllers", ASME Transaction, Journal of Dynamic Systems, Measurement and Control, Vol. 113, March 1991, pp6-10.

[16] Janeczko, J., "Machine Tool Thermal Distortion Compensation", 4th Biennial International Machine Tool Technology conference, September 1988.

[17] Jouaneh, M., Wang, Z. and Dornfield, D., "Tracking of Sharp Corners Using A Robot and A Table Manipulator", University of California, Berkely, Beitrag fur das USA-Japan Symposium on flexible Automation, July 1988.

[18] Koren, Y., "Design of Computer Control for Manufacturing Systems", ASME Transaction, Journal of Engineering for Industry, Vol. 101, No. 3, August 1979, pp326-332.

[19] Koren, Y., "Computer Control of Manufacturing systems", McGraw-Hill, New York, 1983.

[20] Koren, Y., "Robotics for Engineers", McGraw-Hill, New York, 1985.

[21] Kulkarni, P. K. and Srinivasan, K., "Optimal Contouring Control of Multi-Axis Drive Servomechanisms", ASME Transaction, Journal of Engineering for Industry, Vol. 111, May 1989, pp140-148.

[22] Makino, H. and Ohde, T., "Motion Control of the Direct Drive Actuator", Annals of the CIRP, Vol. 40/1/1991, pp375-378.

[23] Markiewicz, B. R., "Analysis of Computer Torque Drive Method and Comparison with Conventional Position Servo for a Computer-Controlled Manipulator", NASA Tech, 1973, Memo pp33-601, J.P.L.

[24] Ni, J., Zhang, B. L. and Wu, S. M., "On-Line Identification of Volumetric Errors of Multi-Axis Machines", Manufacturing International 1988, Atlanta, Georgia, 1988, pp77-83.

[25] Ono, Y. and Kuwahara, H., "The New Design of Motor, Position Sensor and Position Control System for Direct Drive Manipulators", ASME Proceedings, Robotics: Theory and Applications, Anaheim, California, December 1986, pp123-128.

[26] Paul, R. P., "Robot Manipulators: Mathematics, Programming, and Control", MIT press, Cambridge, Mass, 1981.

[27] Pritschow, G. and Philipp, W., "Direct Drives for High-Dynamic Machine Tool Axes", Annals of the CIRP, Vol. 39/1/1990, pp413-416.

[28] Schepper, F and Yamazaki, K., "Application of ASIC-Technology to Mechatronics Control: Development of the Flexible Servo Peripheral Chip", Annal of the CIRP, Vol. 37/1/1988, pp389-392.

[29] Taso, T. C and Tomizuka, M., "Adaptive and Repetitive Control Algorithms for Noncircular Machining", Proceedings of the 1988 American Control Conference, June 1988, pp115-120.

[30] Tomizuka, M., "Zero Phase Error Tracking Algorithm for Digital Control", ASME Transactions, Journal of Dynamic Systems, Measurement, and Control, Vol. 109, March 1990, pp1-8.

[31] Torfs, D., Swevers, J. and De Schutter, J., "Quasi-Perfect Tracking Control of Non-Minimal Phase Systems", Proceedings of the 30th IEEE Conference on Decision and Control, Brighton, UK, 1991.

[32] Tung, E., Anwar, G. and Tomizuka, M., "Low Velocity Friction Compensation and Feedforward Solution Based on Repetitive Control", Proceedings of the 1991 American Control Conference, June 1991, pp2615-2620.

[33] Yamazaki, K., "Development of Flexible Actuator Controller for Advanced Machine Tool and Robot Control", Annals of the CIRP, Vol. 36/1/1987/, pp285-288.

[34] Lo, C. C. "Cross-Coupling Control of Multi-axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

[35] Park, H. A., " Adaptive Matching and Preview Controllers for Feed Drive Systems", ASME Transaction, Journal of Dynamic Systems, Measurement and Control, Vol. 113, June 1991, pp316-320.

[36] Kosko, B., "Neural Networks and Fuzzy Systems", Prentice-Hall Inc, 1992.

[37] Benito, F. R. and Hedrick, J. K., "Control of Multivariable Non-linear Systems by the sliding Mode", Int, J. Control, 1987, Vol. 46, No. 3, pp1019-1040.

[38] Palm, R., "Sliding Mode Fuzzy Control", IEEE Conference on Fuzzy Systems, 1992, San Diego, pp519-526.

[39] Munro, N., et al, "Robust control system analysis and synthesis", IEE International Conference on Control'94, March 1994, Vol. 1, pp583-587.

CHAPTER THREE

Conventional Multi-Axis Servo-Drive System Control Structures and Software Mechanisms for Achieving Coordinated and Synchronised Motion

3.1 Introduction

Motion control problems encountered in modern manufacturing and automation are often multidimensional: that is, they involve more than one axis. The previous chapter evaluates the servo-controllers which form the foundation of multi-axis control. The coordinated and synchronised control of the multiple axes is another challenging problem, since the performance of a single axis control system implemented with a conventional controller will be influenced and possibly degraded by long system-time delay, dead zone and/or saturation of actuator mechanisms, model and/or parameter uncertainties, process noise and/or external disturbances. When such a system is operating, ineffective coordination may produce unqualified products, or damage to the products and/or the machine. With the progress in motion control technology, the control structures used in the multi-axis system have been continuously modifying and more comprehensive software mechanisms have been developed. This chapter gives a comprehensive review of these methods.

3.1.1 Master-Slave Approach

Multi-axis motion synchronisation can be achieved by either the ***master-slave*** approach or the ***equal-status*** approach.

The master-slave technique is designed to reduce the synchronisation error under the assumption that slaves can follow the master instantly[6][17]. Figure 3.1 shows that the system can attain very good synchronisation performance despite a variable following error of the master. However, in a discrete-time implementation, there is an inherent time delay introduced by using the present output of the master as the future reference to the slaves. Figure 3.2 shows the time delay effect. When the slave has been disturbed and the master is running rapidly, this time delay will produce large synchronisation

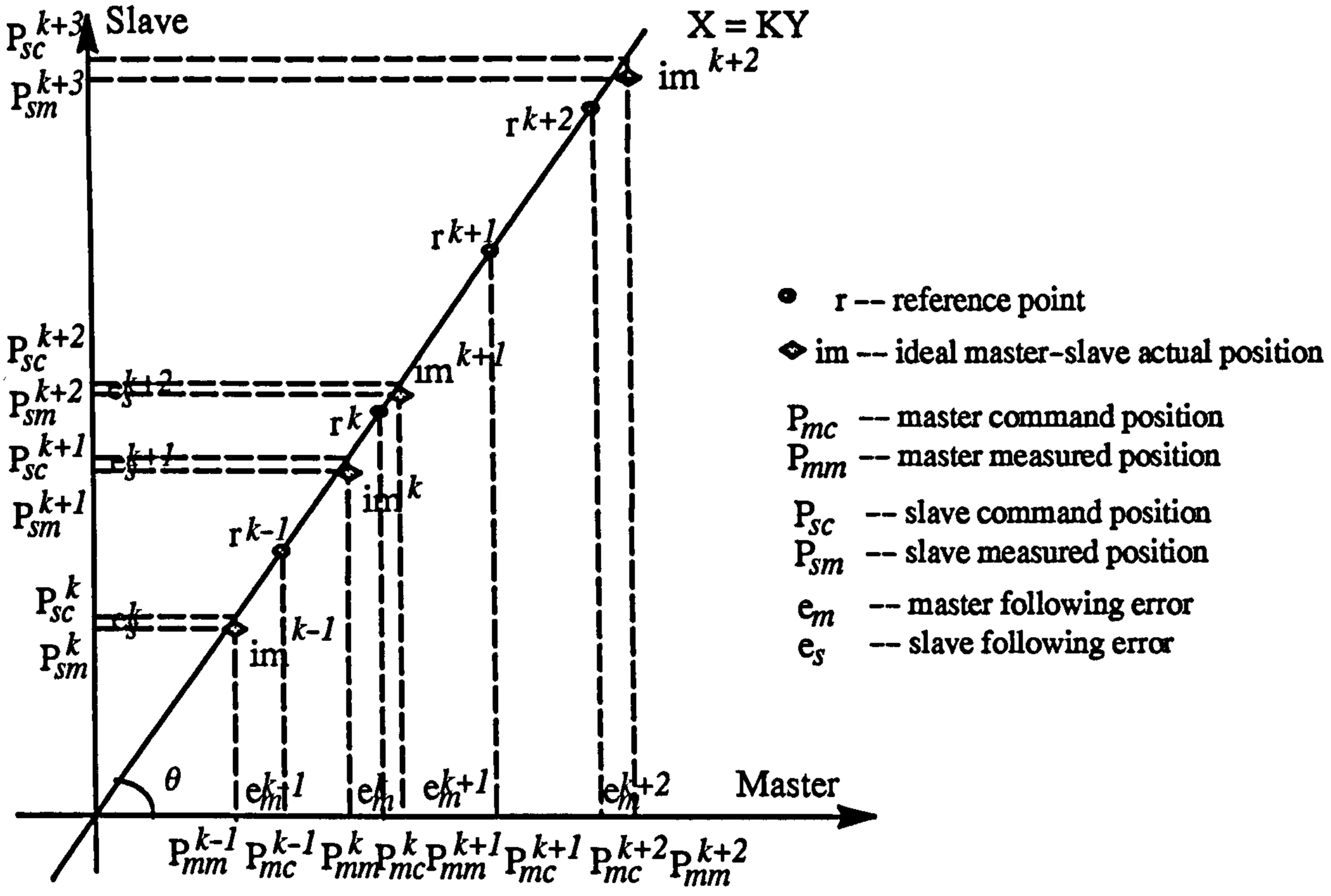


Figure 3.1 Ideal Master-Slave Performance

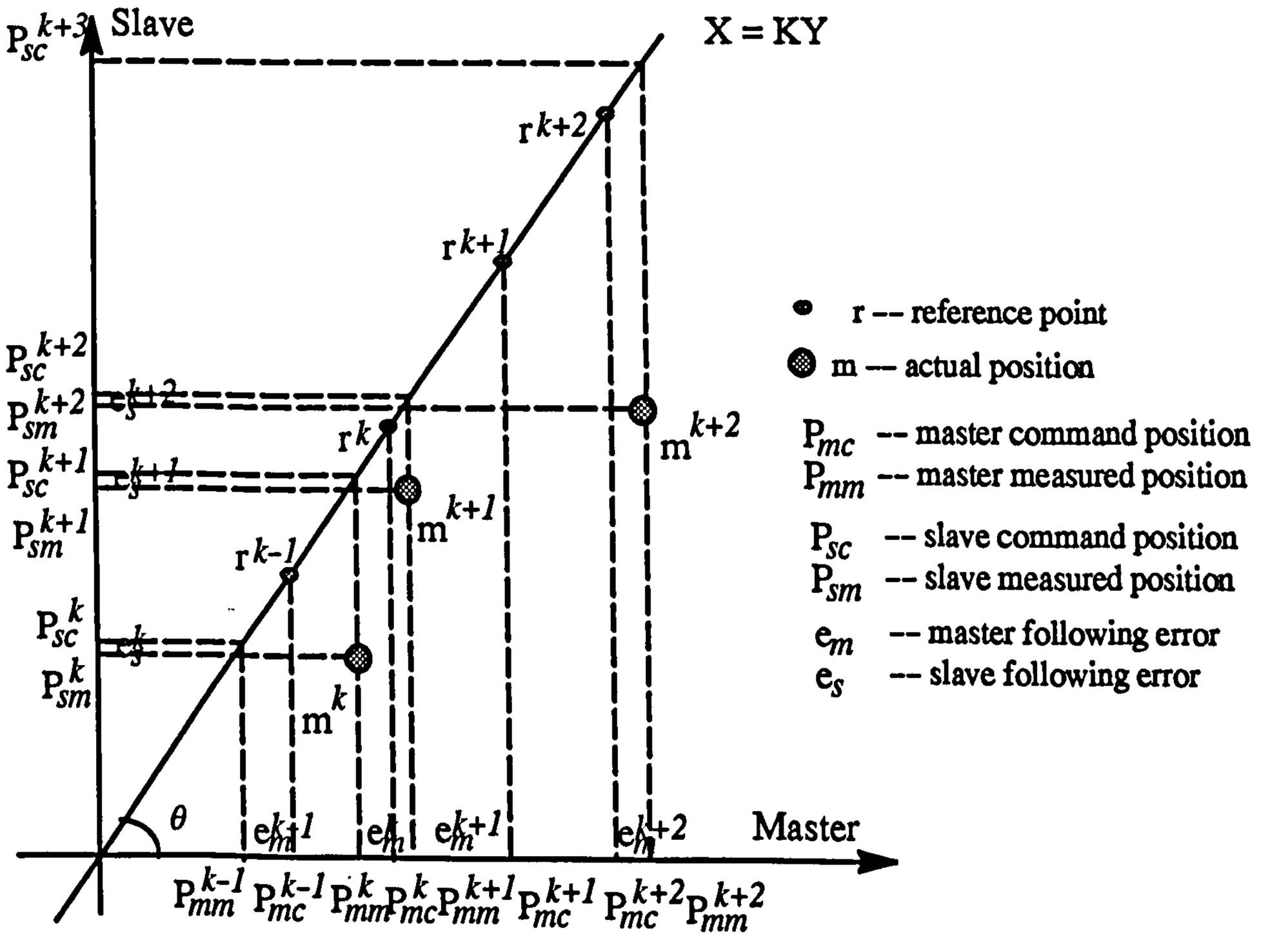


Figure 3.2 Master-Slave with One Sample Time Delay

errors[20]. In addition, if the slaves have a slow response (or say, big dynamic response lag), then the synchronisation performance of the master–slave will be worse.

3.1.2 Equal Status Approach

In the equal–status approach, the synchronising controller treats individual axes in similar manner without favouring one axis over another[7][13][22]. When the dynamics are significantly different among multiple axes, the equal–status approach may not be the best because the synchronisation speed of the overall system is set by the slowest axis. In this case, it would be more sensible to take the master–slave approach. The slow axis is under conventional servo control and acts as the master for the faster axes[7].

3.2 Independent Drives Control

Independent drives control is the simplest multi–axis control structure, because it only needs to simply combine all the servo–loops. As can be seen in Figure 3.3, a multi–axis servo control system can be configured around a single microprocessor that generates the motion commands for all the axes as well as tend to the I/O, communications and operator interface. This single microprocessor strategy is somewhat limited when managing high speed applications because the microprocessor has to handle a wide variety of tasks at simultaneously.

For example, if two of the motors are moving, the microprocessor is generating two motion commands. While generating these commands, a host computer may send over a request via the communications port and the operator may be requesting a change to start up the third motor. It is easy to see that as the number of requested activities goes up, the microprocessor has less time to spend with each individual task. This type of microprocessor, time–management bottle-neck is one of the most challenging engineering software problems for the motion control design engineer.

The block diagram in Figure 3.4 is expanded again to reconfigure the pieces of a multi–axis motion control system. Here a master processor has been added and each motion control axis has its own microprocessor for command generation.

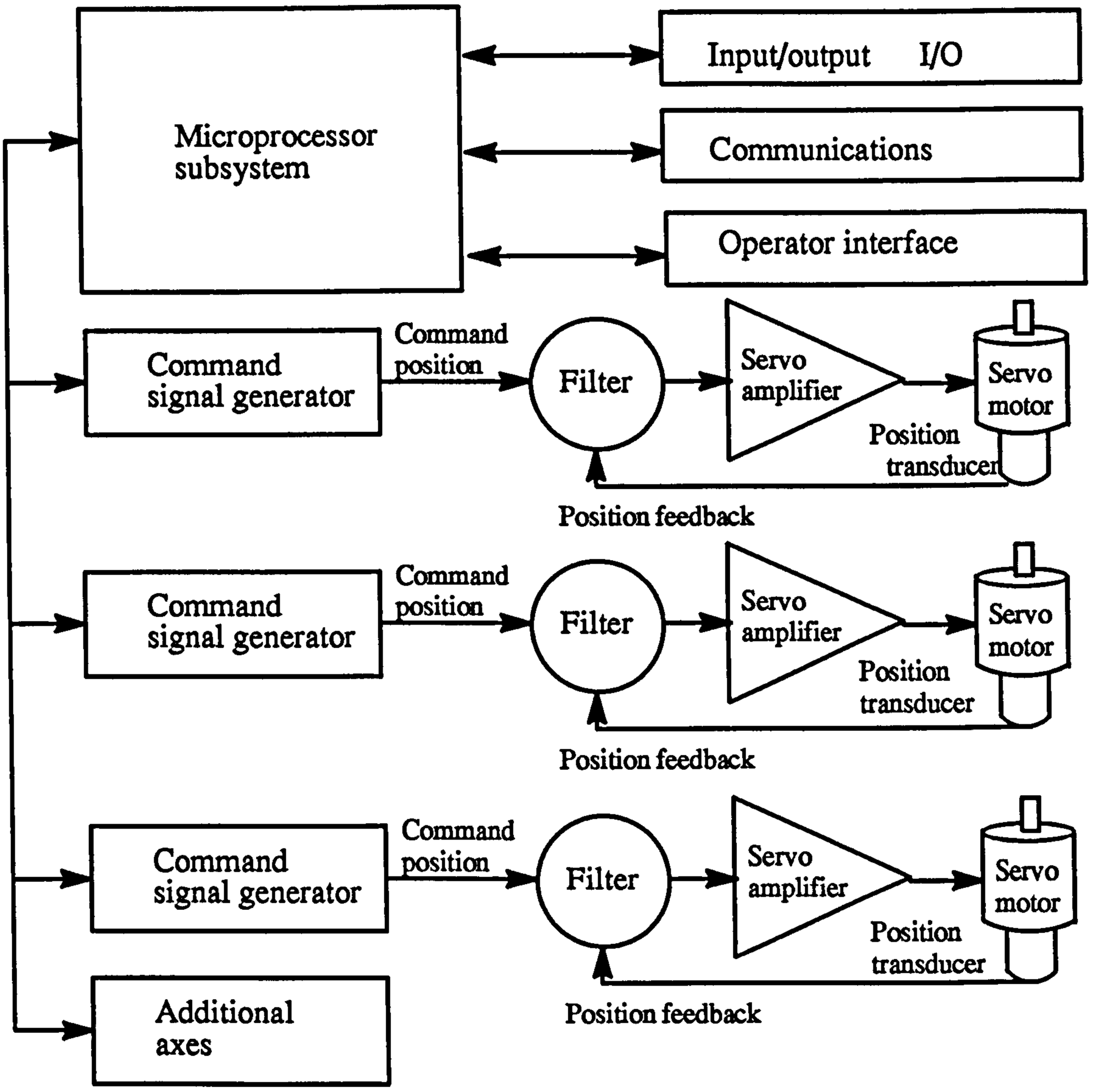


Figure 3.3 Multi-axis Motion Control System

This configuration allows higher speed and increasingly complex applications to be handled without running into the time–management bottle-neck. The master processor tends to the I/O, communications and operator interface. The difficult task of command generation has been delegated to the individual axis microprocessors. This configuration is better suited to handle multiple axes.

With this ***open–loop*** structure, all axes in the system receive their demand position from some high level entity as shown in Figure 3.3, or generate it on their own processors. None of the axis positions are used to generate the demand position of another axis. The position relationship is defined by the user, according to which the demanded position for each axis is generated. When a move command is signalled by a supervisory entity, each axis controller runs independently. The motion synchronisation is attained by a

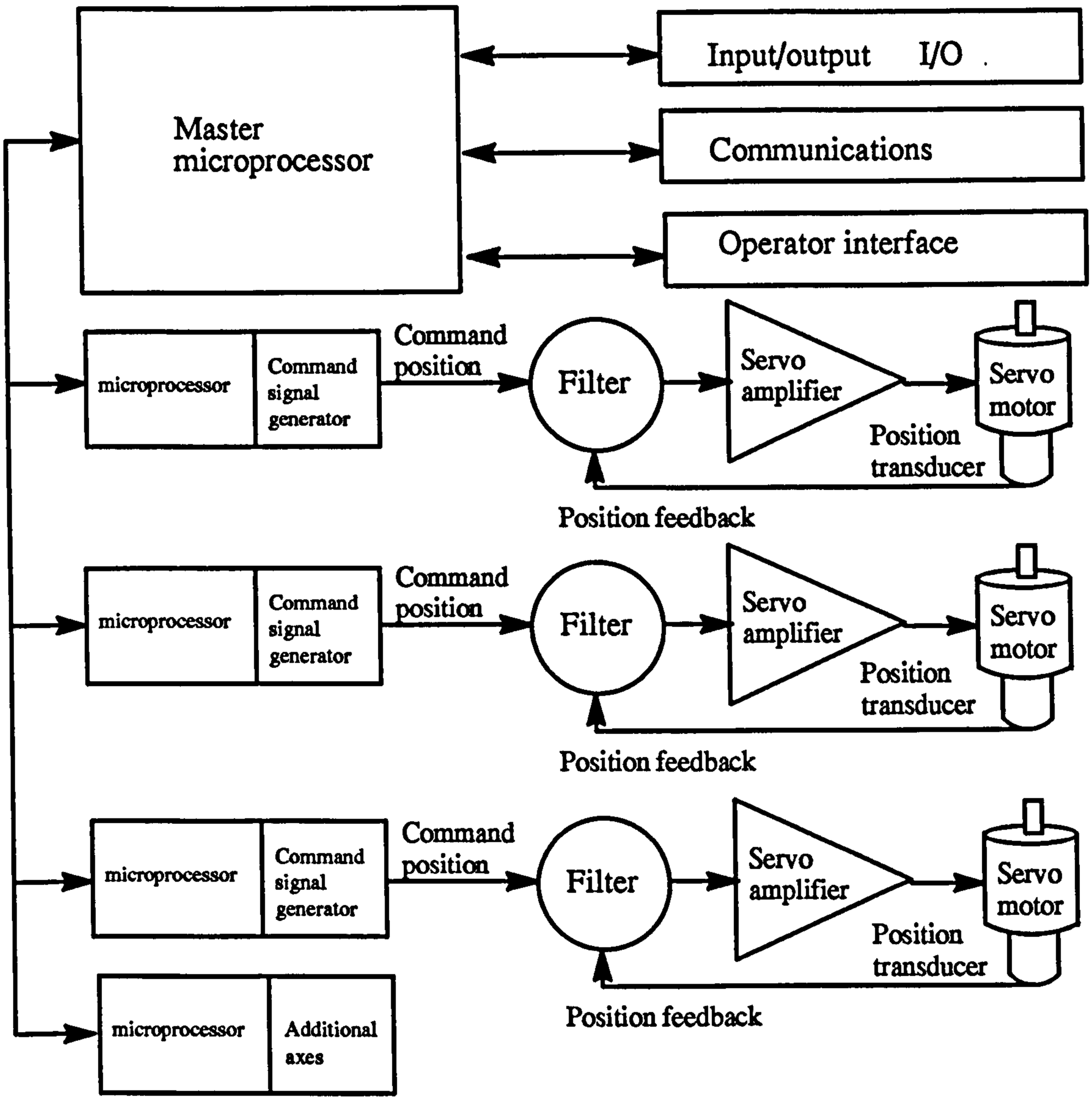


Figure 3.4 Multi-microprocessor Architecture in a Multi-Axis Motion Control System

time switch, at each time tick all axes send out their reference commands. Malfunction of one axis will not directly affect the behaviour of other axes. The resulting motion synchronisation among the axes is determined by the performance of individual servo-drives as shown in Figure 3.5.

3.3 Encoder Tracking Technique

Encoder tracking is a technique that monitors a reference motor with an encoder and controls a secondary motor[17]. The method provides for programmable motor-to-encoder ratios that determine relative speed between the axes. A dynamic change in ratio produces acceleration or deceleration. Applications for encoder tracking

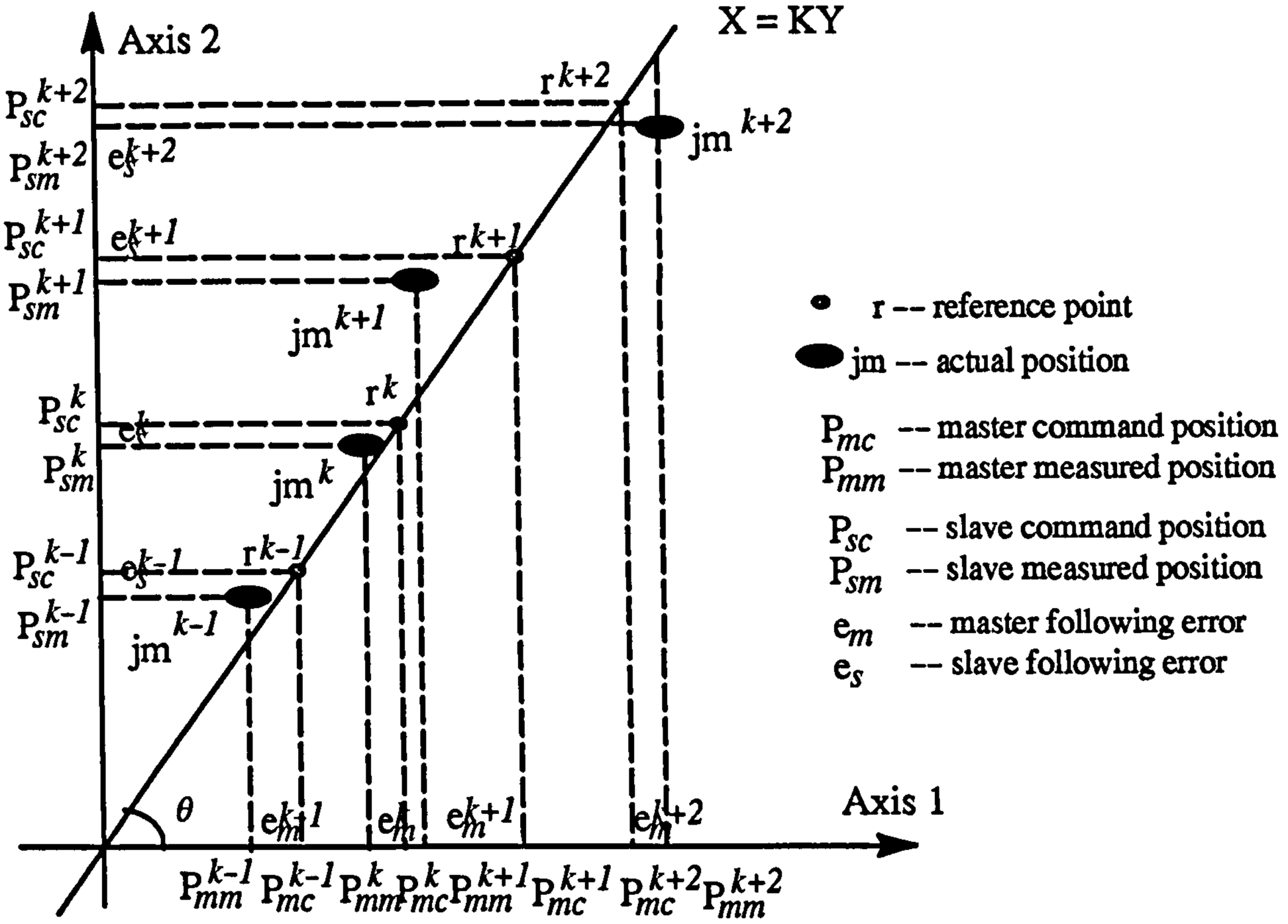


Figure 3.5 An Independent Two Axis System

include electronic gearboxes, conveyor belts, electronic cams and web control processes[6][18]. In principle, the encoder tracking method is a type of master-slave approach.

Many control applications require the coordination or synchronisation of two or more axes of motion in a specific pattern. The most familiar example is found in X-Y plotters, but simultaneous motion in two axes need not always describe a plane figure. Control of a constant speed ratio between two pumps or the motion of a welding head with respect to a moving conveyor belt are other examples. However, these applications all require that the motion profile of a second axis be based on the position, speed, or acceleration of a reference axis. A basic indexer or follower is directly coupled to the drive shaft of the primary motion system with an incremental encoder. The secondary servomotor either follows the primary position and velocity exactly or it follows a complex profile constructed by programming several move segments end to end. Figure 3.6 gives a basic encoder tracking system configuration[6].

Before the introduction of tightly coupled indexers, primary and secondary motion profiles were specified individually, with some form of master clock being used to

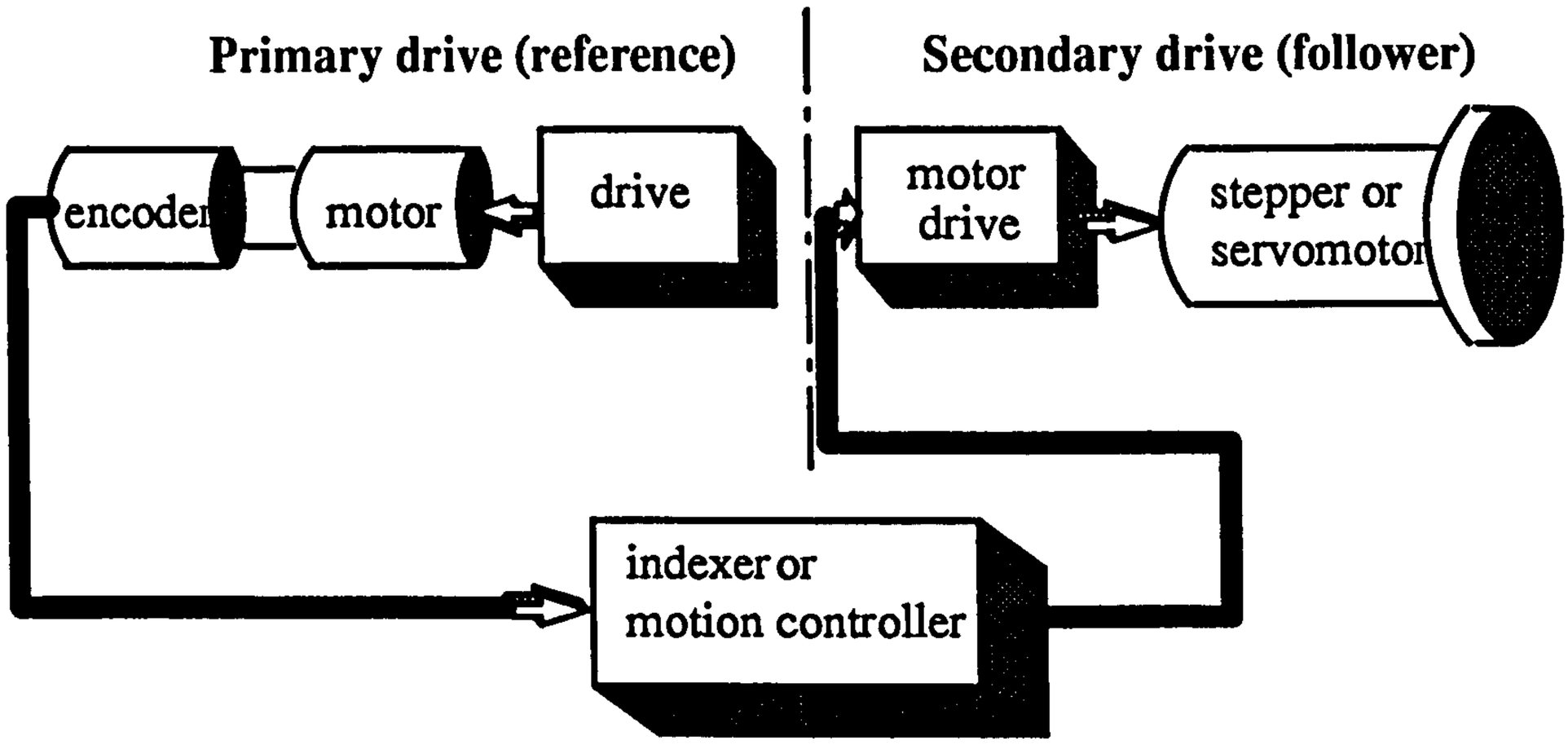


Figure 3.6 Encoder Tracking Basic System Configuration

ostensibly synchronise the activity[17]. As a result, small deviations or errors from the ideal profile described for either axis often resulted in a gradual drift between primary and secondary motion as shown in Figure 3.5. Over a period of time, this accumulated error could cause process malfunction, or worse still, physical damage to the machinery that was being controlled. And the problem obviously becomes more acute at higher operating speeds, where even a small error can equate to a large displacement. In programming an indexer, the absolute motion profile of the secondary axis is not specified. Instead, the indexer is programmed in terms of ratio, offset, or more complex relationships between the measured (reference) motion and the controlled (secondary axis) motion. This eliminates the need for precise absolute motion specifications for both the reference and secondary motion profiles[6].

Choosing an encoder is very important in the encoder tracking method. The precision required in the position of primary motion determines the minimum encoder resolution. For example, if the motion of a primary system must be measured to 0.0254 mm accuracy, and the encoder rotates one turn/0.106 m, a post-quadrature resolution of 4,000 steps (or 1,000 lines) per revolution is needed. The choice of encoder resolution affects the following-motion smoothness. In general, the higher the encoder resolution, the smoother and more accurate the secondary motion.

A very short sample time for secondary motion is also required for this method. Some control systems can provide sample rates at 2 kHz, allowing the torque demand for the secondary motor to be recalculated every 500 μs or less[17]. This fast servo loop update

rate minimises following errors by enabling the secondary motor to react very quickly to any change in primary axis position or velocity, which is especially important for relatively short moves, and also significantly reduces jitter when the motors are stationary.

Clearly the following errors of the slaves (following axes) must be significantly better than those of the master (primary axis) and with no disturbances in the slave loops in order to make this method worthwhile[19] (as also shown in Figure 3.1). In addition, in a discrete-time implementation, there is an inherent time delay introduced when using the present output of the master as the future reference to the slaves. This is illustrated in Figure 3.2. Furthermore the encoder-tracking method also introduces the dynamics of the primary axis into the following axes. This dynamic influence will affect the performance of following axes. In such cases, sending the demand position of the master (primary axis), instead of the real position, to the slaves (following axes) can give better results[14]. If implemented in this way, the encoder-tracking method is very similar to the previous independent drives control.

3.4 Dual-Loop Control Technique

Dual-Loop control is a technique which includes two control loops as shown in Figure 3.7. The inner loop is a normal control loop; the outer loop is condition loop which is used for synchronising the motions. The condition check sensor may be the motor sensor, or may be a remote sensor. Where an auxiliary sensor is used, data can be used as supplementary feedback to adjust the motor speed.

Dual-loop design was firstly introduced to solve the problem of non-ideal mechanical coupling through the use of two sensors, one on each side of the coupler. Nowadays, this method is not only used in backlash compensation and slip couplers, it is also applied to feed systems for packaging materials, high performance magnetic tape drives, web tension control and more generally, in situations where drives have to be linked in a way which cannot be easily determined analytically[17][24].

However, designing a dual-loop control system is not an easy task, since control actions among the axes need to be well coordinated to avoid control action conflicts leading to some axis saturation, degradation of system performance or damage to the product or

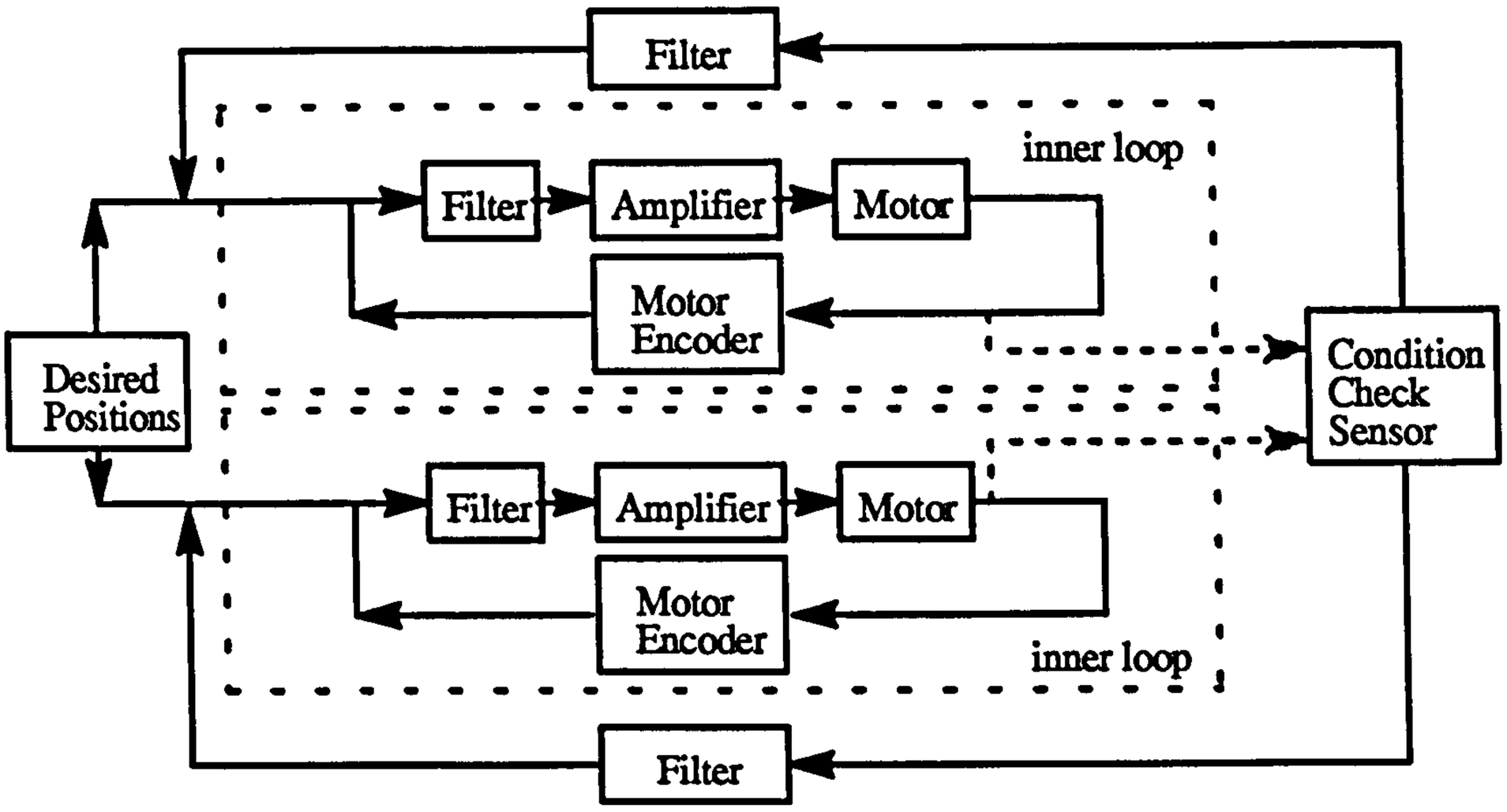


Figure 3.7 Dual Loop Control System

machine[1][17]. Improper design may also introduce resonance modes to the system and limits its gain and bandwidth when using it for web synchronisation, if the relatively flexible material is coupled by a sensor[1][17].

In tension control applications, conventionally the PID control law has been employed to maintain even tension[14], but the PID method presents the following problems[23]. First, the running speed cannot be very fast, or the tension becomes unstable. Moreover, fine adjustments according to the thickness or the material of the film used must be performed by an experienced worker familiar with the traits of the machine. This situation can easily be corrected and stabilized by applying fuzzy control.

When fuzzy control is applied to maintain consistent tension in the process of discharging and rolling film, a fuzzy inference adjusts the motors on the discharge and roll sides to the proper speed based on its conditions. The conditions of the fuzzy inference are rotation speed variations and their differentials, tensions and their differentials. A set of inference rule is used to calculate the outputs (motor speeds) of the fuzzy inference. Table 3.1 shows the original problems and the improved results and effects achieved by applying fuzzy control[23].

Table 3.1 The Original Tension Control Problems And The Improvements Achieved By Employing Fuzzy Operation

Problem	Improved Result	Effect
Tension is not stable when the film is running at high speed.	Tension is stable running at any speed.	Uniform quality achieved.
Adjustments made by experienced workers are necessary to maintain consistent tension.	No fine adjustments are required, system can be operated by anyone.	Improved productivity.
Setting up the operation takes a long time.	Setup is quick because fine adjustment is unnecessary.	Time and labour saved.

For some applications, conventional dual–loop control would not work. For example, moving advertisement hoardings, where the pictures are unwound from a roll on one side and fed to a roll on the other side. It becomes difficult to engineer when they carry a large number of advertisements[24].

Where many different images have to be shown, the rolls will differ considerably in diameter as they unroll and roll up. The material on which the pictures are mounted is also likely to stretch, and change in length, according to temperature and humidity. In this case, a master and intelligent slave method has to be used[24]. Figure 3.8 describes a system which communicates with the roller drives to provide variable feed rate.

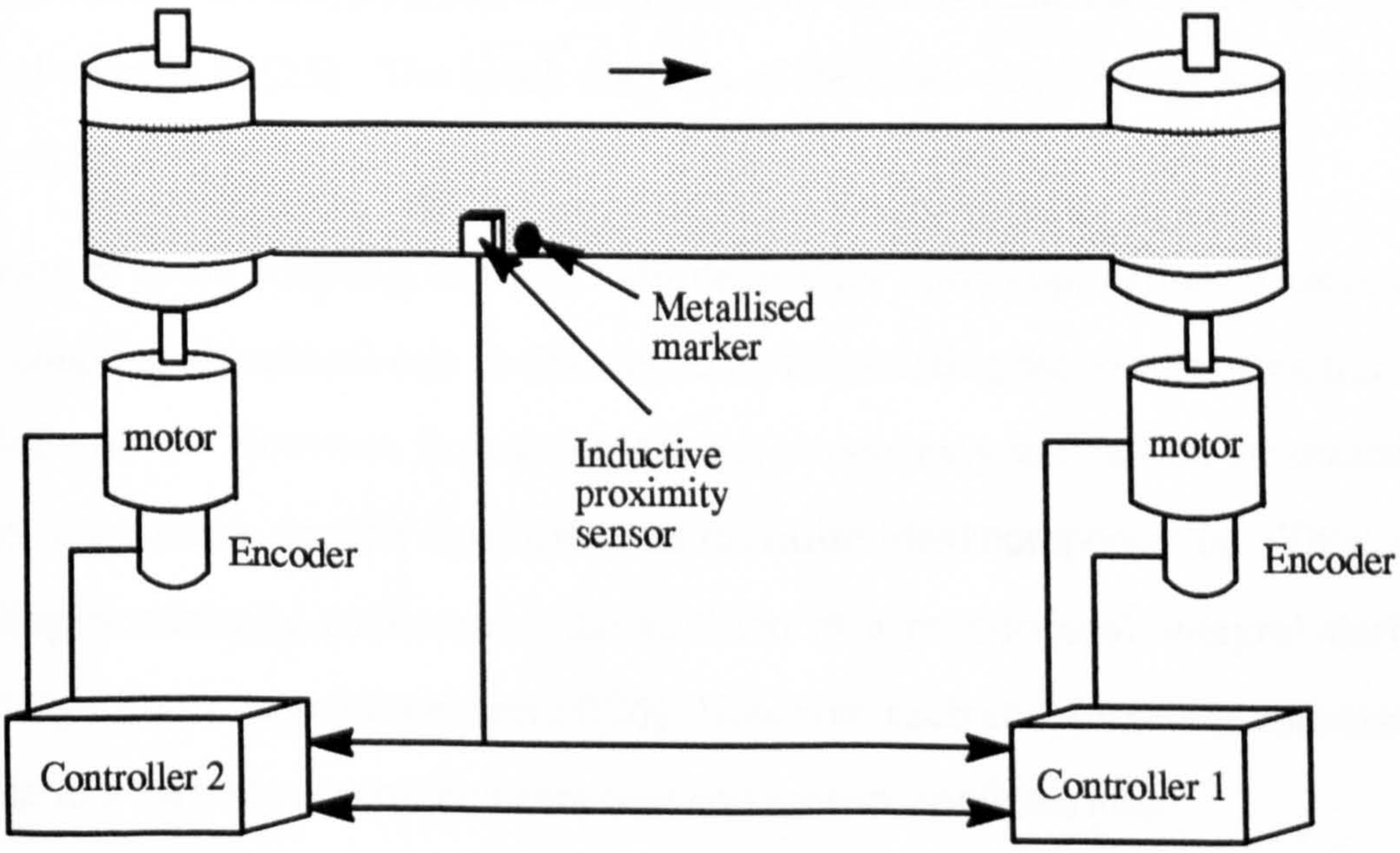


Figure 3.8 Roller Drive System

The master drive (controller 1) powers the take up roller which regulates belt speed and position. Correct positioning is monitored by means of an inductive proximity sensor detecting metallised marks on the side of the belt, and this information is used to correct the position control loop as the system runs. The feed roller drive (controller 2) is used to regulate belt tension, to avoid wrinkles or tearing, and works by monitoring motor current and using this information to determine the appropriate torque. Both axes constantly exchange digital information as they are working.

3.5 Cross-Coupling Technique

The cross-coupling control architecture was first proposed by Koren [25]. Most of the work in cross coupling control concentrates on machine tool control[26]. Some work in cross coupling control can be found in the field of robotics, particularly in the domain of mobile robot control[27]. In machine tool servo control, the main idea of the cross-coupling control is to build in real time a contour error model based on the feedback information from all the axes as well as the interpolator, to find an optimal compensation law, and then to feed correction signals to the individual axes, whereby an error in any axis affects the control loops of all axes. The cross-coupling controllers consist of two parts: (1) the contour-error model, and (2) a control law. Consequently, the differences between the various cross-coupling control systems that were proposed by many other researchers who followed the original work are in the contour error model or in the control law[28][29][30][31][32][33][34], but all of them are based on the same original concept in [25]. The block diagram of the cross-coupled system is shown as Figure 3.9.

The type of cross coupling tends to vary dependant upon application. Conventional cross coupling algorithms can be developed by formulating the transfer function of the coupled system. However, in practice it is not always easy to describe the coupling by means of a discrete transfer function so as to realise ideal compensation. Thus, cross-coupling is normally obtained by the addition of a proportional-integral-derivative (PID) algorithm or a variation thereof[26]. However, such an approach is not ideal when subject to a variable control environment and system nonlinearities.

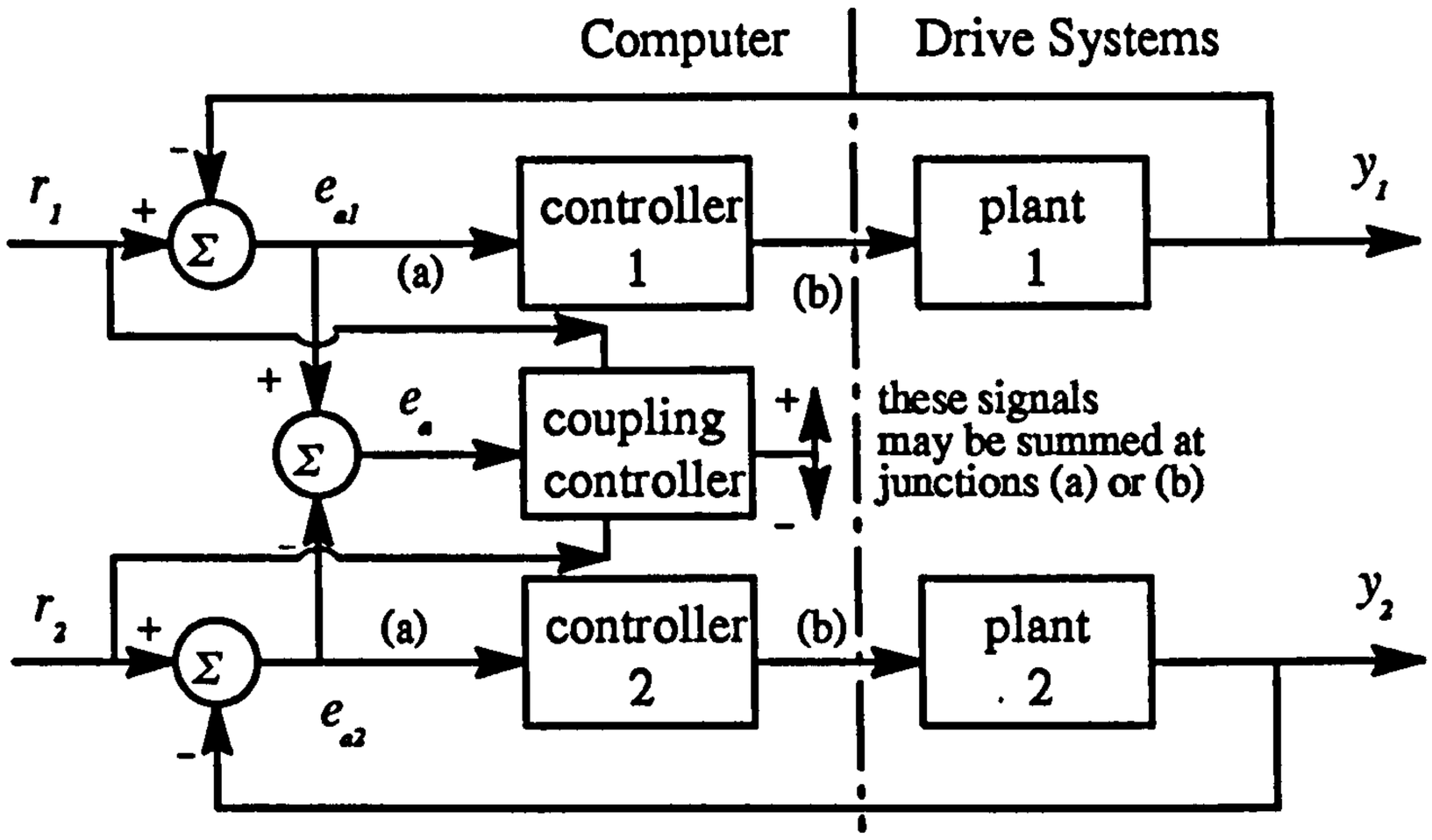


Figure 3.9 Diagram of a cross-coupled system

Cross-coupling methods consider the whole system as a single unit, rather than in terms of its individual loops. The control structure involves cross coupling between the axes. For a two axes system, the coupling controller responds to the synchronization error, ie, the difference between the two motion errors, to generate a coordination signal for the two motion control axes. The cross-coupled structure is suitable for systems where the time response of each individual controlled variable takes on a smaller role relative to the intervariable dependence $y = f(x)$.

The cross-coupling controller has to combine other control algorithms which control the individual axes. These control algorithms for each axes could be normal PID controllers, or adaptive feed-forward controllers[36]. The cross-coupling technique is an "equal-status" control approach. It couples axes in a more symmetrical manner than master-slave synchronisation, and provides useful insight into the more general coupling strategy that we seek[13].

3.6 MIMO Techniques

The synchronising control problem involves a multiple number of motion control axes. Therefore, any multivariable control technique is naturally a good candidate to provide the basis for synchronised control. Synchronising control is a special type of MIMO (Multiple Input, Multiple Output) control problems. A fundamental question from the viewpoint of control system design is then how a multivariable control theory can be

customized or extended to achieve the objective of synchronisation. The use of robust control and linear quadratic (LQ) optimal control in synchronising control from the "equal-status" viewpoint was explored by some researchers [7].

In the robust control approach, a system is considered as consisting of several independent single-input, single-output (SISO) subsystems with relative degree one. To minimize the difference between these outputs, two synchronising control methods are developed. In the first method, the lagging axis is accelerated to "catch" the leading axis while in the second method, the leading axis is slowed down. To implement the two methods, some new switching functions which depend on the synchronisation error are introduced into the control law. When tracking performance of each axis is not as important as synchronisation, it is reasonable to take the latter method.

The attractive point in utilizing the LQ framework for synchronising control is the rich body of knowledge about its theory. A term expressing the synchronising objective can be explicitly introduced in the performance index. Synchronising performance can be improved by increasing the weighting factor on this term. However, MIMO systems are not always favoured because of their increased complexity.

3.7 System Interconnection Technique

In some industrial applications, eg steel rolling mills, paper plants, and hydraulic press systems[38], a number of identical motors are employed with identical inputs, and identical outputs are expected. The problem of output equalization is also relevant if the individual systems (motors) are not at all identical. For example, due to different loading conditions, some parameters in principally identical motors may vary, eg segmented conveyor belts with different loads[39]. Sometimes among a number of non-identical systems there is one 'master', and the outputs of the other 'slave' should be identical to the output of the master during a transient time[40].

The system interconnection method introduces a special control structure for the MIMO motion control system with the same inputs and outputs requirements[37]. Figure 3.10 describes an interconnection strategy with built-in adaptive controllers, which achieves synchronisation of the scalar linear MIMO systems. The closed-loop network forces all outputs to follow the same signal asymptotically while maintaining the open-loop

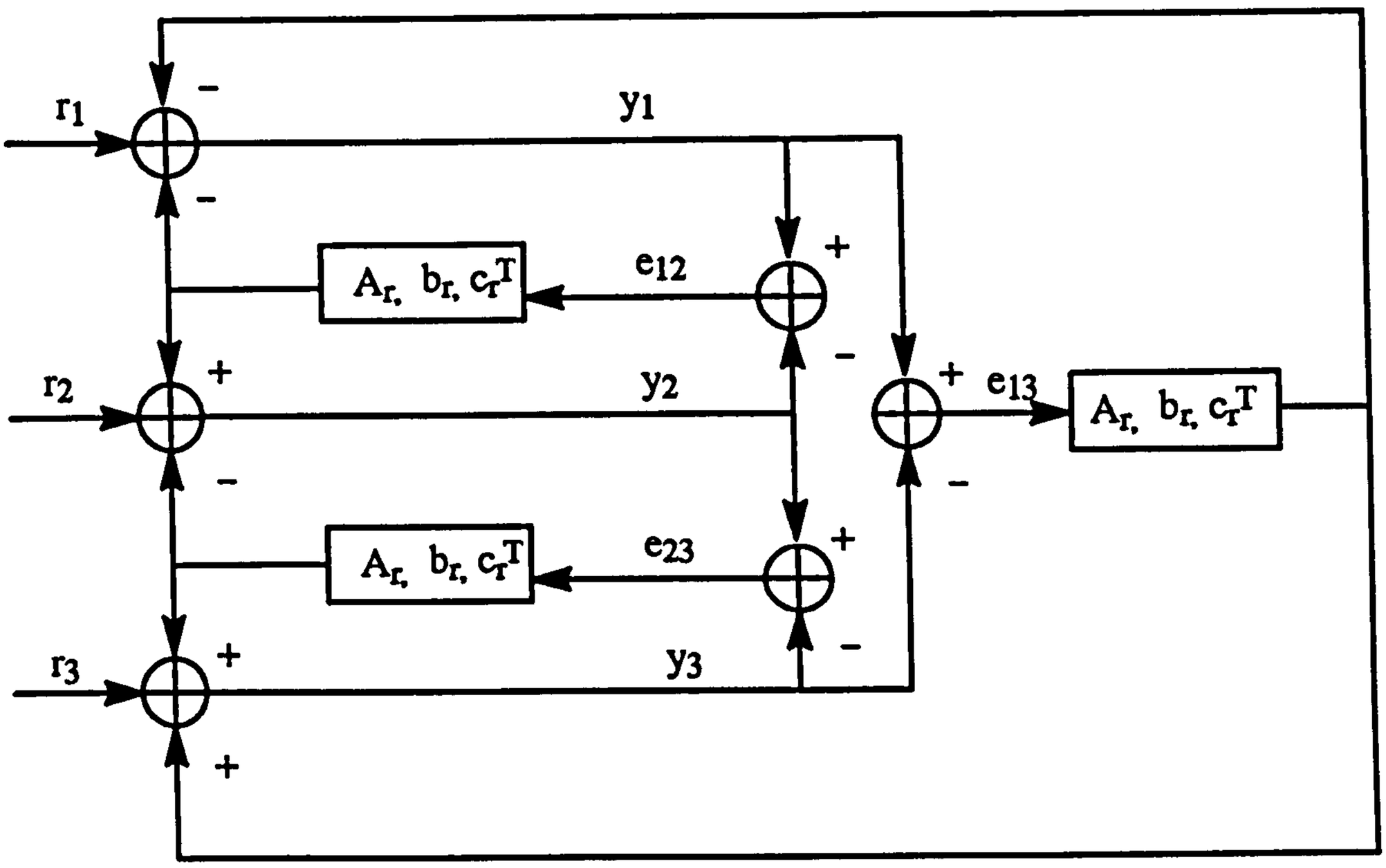


Figure 3.10 An Inter-connection Structure for Multiple Axes. Source: Schmid.

characteristics. In the design of the output feedback controllers, no knowledge of system parameters are assumed, but each axis system must have the same poles and be high-gain-stable.

3.8 Scalar Field Control Technique

Scalar field control is another kind of axis coupling strategy created through a more physical appreciation of requirements[4][13]. A physical surface Φ can provide a potential energy field which acts on a ball. Due to the influence of gravity, the ball seeks out an equilibrium position on a path. The forces on the ball acting to take it to the required path may be resolved in the coordinate axis directions. These resolved forces are related to the slopes of the surface resolved in these directions. Mathematically, the respective resolved slopes may be expressed using partial derivatives. It is possible to use a similar idea to make a position control system achieve a desired position relationship. The partial derivative expressions may then be used as a form of error upon which the controllers act. Figure 3.11 shows the approach, a 2-axis mechanism. The error signal for axis 1 now becomes $-\partial\Phi/\partial y_1$, and for axis 2 becomes $-\partial\Phi/\partial y_2$.

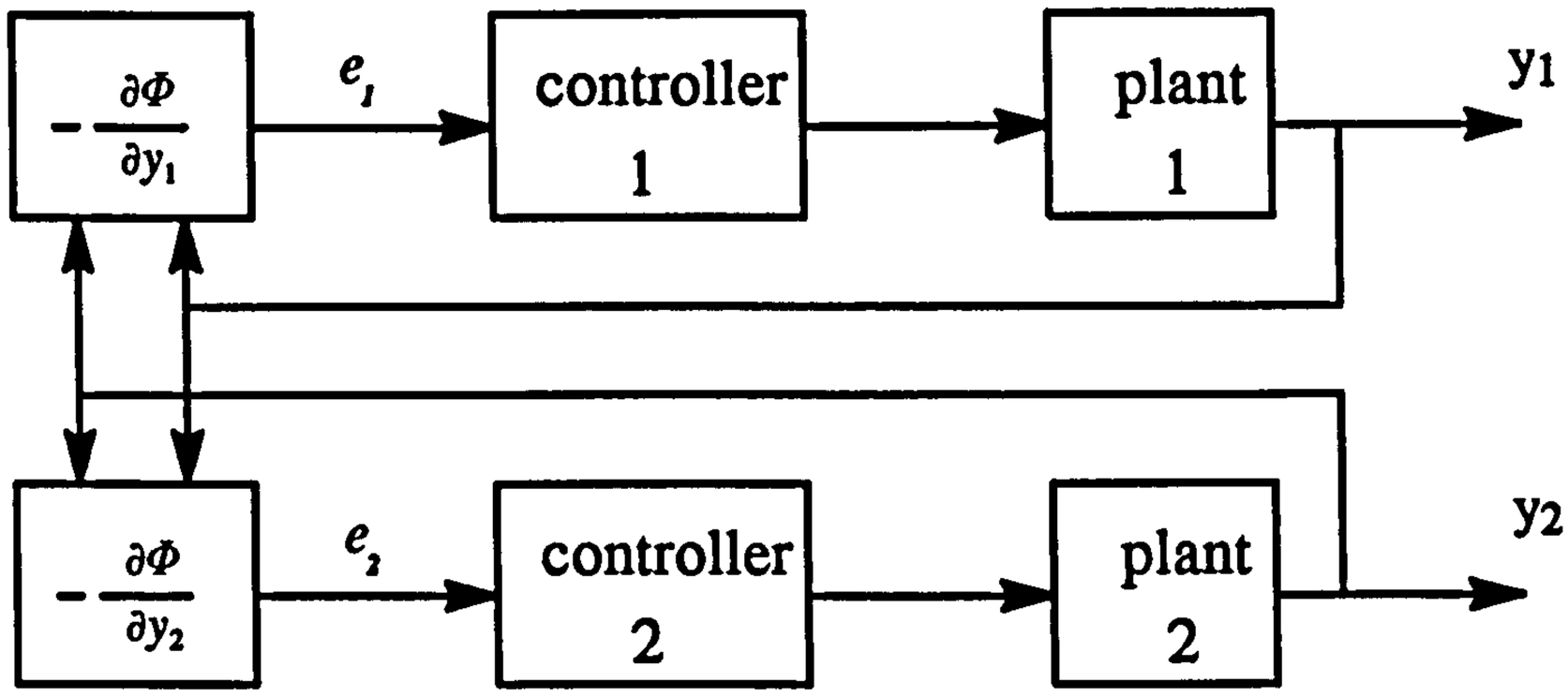


Figure 3.11 A 2-axis System with a Scalar Field Control Mechanism.

Source: Jenkinson.

The mechanism of Figure 3.11 lacks the reference signals of the normal software mechanisms. This is because Φ only tends to propel the output position state[1] of the system towards the mechanism path[2], in order to achieve the desired position relationship.

A scalar field Φ could represent a potential energy field, the vector field $\mathbf{F}$[3] ($\mathbf{F}$=grad Φ) may be regarded as a conservative force field. The partial derivative terms proposed for this controller are the components of grad Φ as, for example, in the case of a field $\Phi(x,y)$, grad Φ may be expressed as

$$grad\ \Phi(x,y) = \frac{\partial \Phi}{\partial x}\ i + \frac{\partial \Phi}{\partial y}\ j \quad ,$$

---

1 · In a N-dimension coordinate system consisted by axis position output y_i (i=1, ... ,n), the point p(y_1 , ... , y_n) represents the output position state of the system.

2. The position relationship of a mechanism can be conveniently represented by a mechanism path[13].

3. At a point in space within the vector field F, there exists a vector whose magnitude is the gradient of Φ (the scalar field), and which points in the direction in which the maximum change of the derivative of Φ occurs [10].

the proposed controller thus behaves rather like the force field **F**; the controller tends to propel the output position state of the system towards the desired mechanism path, which is arranged to coincide with the set of global nadirs of the surface, or scalar field. The aim of the mechanism controlled by the method is to reduce relative following-errors in all conditions — whether reference or disturbance related. This technique introduces a form of coupling between the (physically uncoupled) servomotors which closely emulates the type of coupling found in a real mechanism. Building a precise scalar field in real-time is crucial to the synchronisation accuracy, particularly when nonlinearities exist in the system. It does, however, couple axes with reference to a logical analogy, and provides a very useful insight into the multi-axis synchronisation issues.

3.9 Conclusions

Industrial automation spans a huge spectrum of complexity in terms of both physical structure of machines and the tasks which they perform. A study of current control methodologies highlights that a single control method can not satisfactorily solve all the multi-axis control issues. A combination of these control methods is necessary to attain tight coordination and synchronisation control. The basic nature of motion synchronisation control is through coupling or interconnecting the individual axes. The software mechanisms or algorithms tend to require more intelligence[41] in order to solve the nonlinearities and uncertainness of the controlled processes, and the different characteristic of the individual servo drive loops. In the next chapter, a new control strategy for multi-axis motion control and synchronisation is proposed to tackle these problems.

Chapter Three: References

[1] Baron, W. and Tal, J., "Motion Synchronisation with Intelligent Controllers", Motor-Con. October 1986 Proceedings, pp351-359.

[2] Bullock, T. B., "A Programmable Industrial Computer Coordinates Axis Motions." Official Proc of the First International IMS 1985 Conf (Intelligent Manuf Syst) 1985, pp 204-211.

[3] Crane, J., "Electronic Gearing for Synchronised Motion", Drives & Controls, July/August 1993, pp30-31.

[4] Danbury, R., Jenkinson, M., "Synchronised servomechanisms - the scalar-field approach", IEE Proc.-Control Theory Appl., Vol. 141, No. 4, July 1994, pp261-273.

[5] "Software Cam Controls 40 Axes Simultaneously", Drives and Controls Magazine, November 1990, p131.

[6] Rathkey, J., "How to synchronize servomotors", Machine Design February 1989, pp 109-111.

[7] Hu, J., Chiu, T. and Tomizuka, M., "On Motion Synchronization of Two Axes Systems", Monitoring and Control for Manufacturing Processes, ASME Conf, 1990, pp267-282.

[8] Clarkson, J. C., "New Strategy Improves Multi-Motor Control", Drives and Controls Magazine, December 1989/January 1990.

[9] Seaward, D. R., Johnson, R. C., "Technology transfer from academia to industry of a phase synchronised drives project", IMechE conference on High Speed Machinery, pp 11-18, November 1988.

[10] Jeffrey, A., "Mathematics for Engineers and Scientists", 4th edition, Van Nostrand Reinhold, 1989.

[11] Koren, Y., "Cross-Coupled Biaxial Computer Control for Manufacturing System", Journal of Dynamic Systems, measurement, and Control December 1980, V 102, pp 265-272.

[12] Kulkarni, P. K., Srinivasan, K., "Cross Coupled Compensators for Contouring Control of Multi-axial Machine Tools", NAMRC XIII Proc, 1985, p558-566.

[13] Jenkinson, M., "The synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

[14] Digital Motor Control System, Programmer's Reference Manual, Quin Systems Ltd, December 1992.

[15] Meshkat, S., "Parallel DSPs Excel in CAM and Gearing Applications", PCIM, February 1994, pp6973.

[16] "Motion Control Primer", Industrial Indexing Systems Ltd, 1987.

[17] Scott, R., "Axis Synchronization by Encoder Following", Drives and Controls Magazine, pp 70-72, September, 1991.

[18] Manfred Binder, "Electronic Couplings-Replacement of Mechanical Gears?", PCIM Europe, March/April 1992, pp60-63.

[19] Seaward, D. R., "Continuous Phase Synchronised Drives (For a Rod-making Machine)", PhD Thesis, University of Aston in Birmingham, August 1989.

[20] Moore, P. R., Chen, C., "The Synchronisation of Servo Drives Using Fuzzy Logic Control", IEE Colloquium on Configurable Servo Control Systems, October 1994, UK, Digest No: 1994/176.

[21] Tal, J., "Motion Control Applications", 1989.

[22] Tomizuka, M., Hu, J. S., Chin, T. C., "Synchronization of Two Motion Control Axes Under Adaptive Feedforward Control", Transactions of the ASME, Vol 114, June 1992, pp 196–203.

[23] "Fuzzy Guide Book", Cat. No. P30–E1–2, 1992.

[24] "Communicating roller drivs provide variable feed rate", Eureka on Campus, Spring 1992, p9.

[25] Koren, Y., "Cross–Coupled Biaxial Computer Control for Manufacturing System", Journal of Dynamic Systems, measurement, and Control December 1980, V 102, pp 265–272.

[26] Lo, C. C. "Cross–Coupling Control of Multi–axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

[27] Feng, F., Koren, Y., and Borenstein, J., "Cross–Coupling Motion Controller for Mobile Robots", IEEE Control systems, December 1993, PP 35–43.

[28] Borenstein, J. and Koren, Y., "Motion Control Analysis of a Mobile Robot", ASME Transaction, Journal of Dynamic Systems, Measurement and Control, Vol. 109, June 1987, pp73–79.

[29] Burhoe, J. C., and Nwokah, O. D., "Multivariable Control of a Biaxial Machine Tool", Proceedings of the Symposium on Dynamic Systems, Measurement, and Control, ASME Winter Annual Meeting, LSan Francisco, California, December 1989, pp1–6.

[30] Chuang, H. Y. and Liu, C. H., "Cross–Coupled Adaptive Feedrate Control for Multiaxis Machine Tools", ASME Transaction, Journal of Dynamic Systems, Measurement and Control, Vol. 113, September 1991, pp451–457.

[31] Koren, Y. and Lo, C. C., "Variable–Gain Cross–Coupling Controller for Contouring", Annals of the CIRP, Vol. 104, August 1991, 371–374.

[32] Kulkarni, P. K. and Srinivasan, K., "Cross–Coupled Control of Biaxial Feed Drive Servomechanisms", ASME Transactions, Journal of Engineering for Industry, Vol. 111, May 1989, pp225–232

[33] Liu, C. H. and Chan, W. M., "Microprocessor–Based Cross–Coupled Biaxial Controller for A Two–Axis Positioning System", IEEE Transactions, Journal of Industrial Electronics and Control Instrumentation, 1985, pp327–332.

[34] Masory, O. and Wang, J., "Improving Contouring System Accuracy by Two–stage Actuation", NAMRC Proceedings, 1991.

[35] Tomizuka, M., "Design of Digital Tracking Controllers for Manufacturing Applications", Manufacturing Review 1989 Vol. 2, Part 2, pp134–141

[36] Tomizuka, M., Hu, J. S., Chin, T. C., "Synchronization of Two Motion Control Axes Under Adaptive Feedforward Control", Transactions of the ASME, Vol 114, June 1992, pp 196–203.

[37] Schmid, S., and Pratzel–Wolters, D., "Synchronization Through System Interconnections", IMA Journal of Mathematical Control & Information (1992) 9, pp 161–178.

[38] D'Azzo, J. J. and Houpis, C. H., "Feedback Control system analysis and Synchesis", New York: McGraw-Hill, 1966.

[39] Pratzel-Wolters, D. and Schmid, S., "Adaptive Speed Synchronisation of Segmented Conveyor Belts", Proceedings of the ECMI 1990 Conference, Lahti, 1991.

[40] Vakilzadeh, I. and Mansour, M., "Synchronisation of 'n' integral-plus-double time constant plants with non-identical gain and time constants", J. Franklin Inst. 327, 579-593, 1990.

[41] Moore, P. R., and Chen, C., "Fuzzy Logic Coupling and Synchronised Control of Multiple Independent Drives", Control Engineering Practice, to be published.

CHAPTER FOUR
A New Control Strategy for Multi-Axis Motion Control and Synchronisation

4.1 Introduction

A new control strategy for multi-axis motion synchronisation is proposed in this chapter, based on a synthesis of the methods reviewed in Chapter 3. This control strategy has a new control structure and uses a new control law. After analysing the principle of the motion synchronisation using servo-drives, a generic 'dual closed-loop' control structure is introduced for modern manufacturing machine motion control. The requirements for intelligent integration of motion control mechanism in this control structure leads to develop a fuzzy logic coupling algorithm to coordinate the servo-drives.

4.2 An Intelligent Integrated Approach to Motion Control and Synchronisation

4.2.1 Limitations of Some Traditional Views When Designing Modern Motion Control and Synchronisation Systems

In principle, the function and performance which a machine can attain is highly dependent upon the proper coordination and synchronisation of the axes' motions within the machine, regardless of the design philosophy used. However, this principle is frequently overlooked by many modern machine system designers, or assume that acceptable performance can be automatically obtained as long as each individual axis is 'well' designed and the coordination and synchronisation logic of the axes of motion is correct.

Normally, people who work on computer based machine design assume that low level motion control system is perfectly designed and the design task of the motion control systems is a control engineer's job. The control engineers usually design the motion control system without modifying the process, and they try to design the controller be as robust and adaptable as possible.

Practically, all axes' motions are linked within a manufacturing process. In order to cope with environmental variation and system nonlinearities, adaptive control is often introduced. The function of adaptive control is to sense manufacturing conditions and adjust the motions accordingly. The types of adaptive control and how they are used will vary, of course, among machines and control manufacturers, as described in Chapter 2.

Motion coordination and synchronisation control problems arise because the individual motion axes can not easily attain the desired output on time. Control parameter adjustment may not result in the desired performance[1], if the controller is incorrectly designed. It may be necessary to introduce a new control block within the system that will compensate for the original system's limitations. Such as a tracking control which utilises desired signals in variety of ways to recover delays caused by dynamics of servo-drives[3]. In addition, one or more control functions to deal with synchronisation should also be included in the control system design[2].

However, some uncertainness of the manufacturing process and inaccuracies of the system modelling will reduce the tracking and/or adaptive method's efficiency. Ideal tracking requires precise knowledge of the dynamic behaviour of the axial drive system [3]. While feedforward, adaptive and learning based control applied to each motion axis can enhance the performance of individual axes of a machine, this in turn improves the system coordination and synchronisation. The resulting control system, however, has high order dynamics, and requires large amounts of computation time. This restricts the application when a fast response is desirable[4]. Also, the common difficulty with these approaches lie in the attempt to formulate the input–output relationship by means of mathematical models, which may be difficult in many cases.

Another drawback of these methods is that reducing the axial errors does not necessarily reduce the synchronisation error[5]. Consider, (for example, a machine tool), the case in Figure 4.1. Improvements in the axial control strategy shift the actual cutter location from point P to point P'. Although the axial errors E_x' and E_y' at point P' are smaller than E_x and E_y at point P, the contour error at P' is larger than that at P. (Further improvements will shift the cutter location to point P" and remedy the situation. However, this requires case–by–case analysis, and might cause instabilities.). Figure 4.2 shows another general case with a linear motion relationship for two axes. This figure shows some probable steps to reduce each axis' following error for approaching the

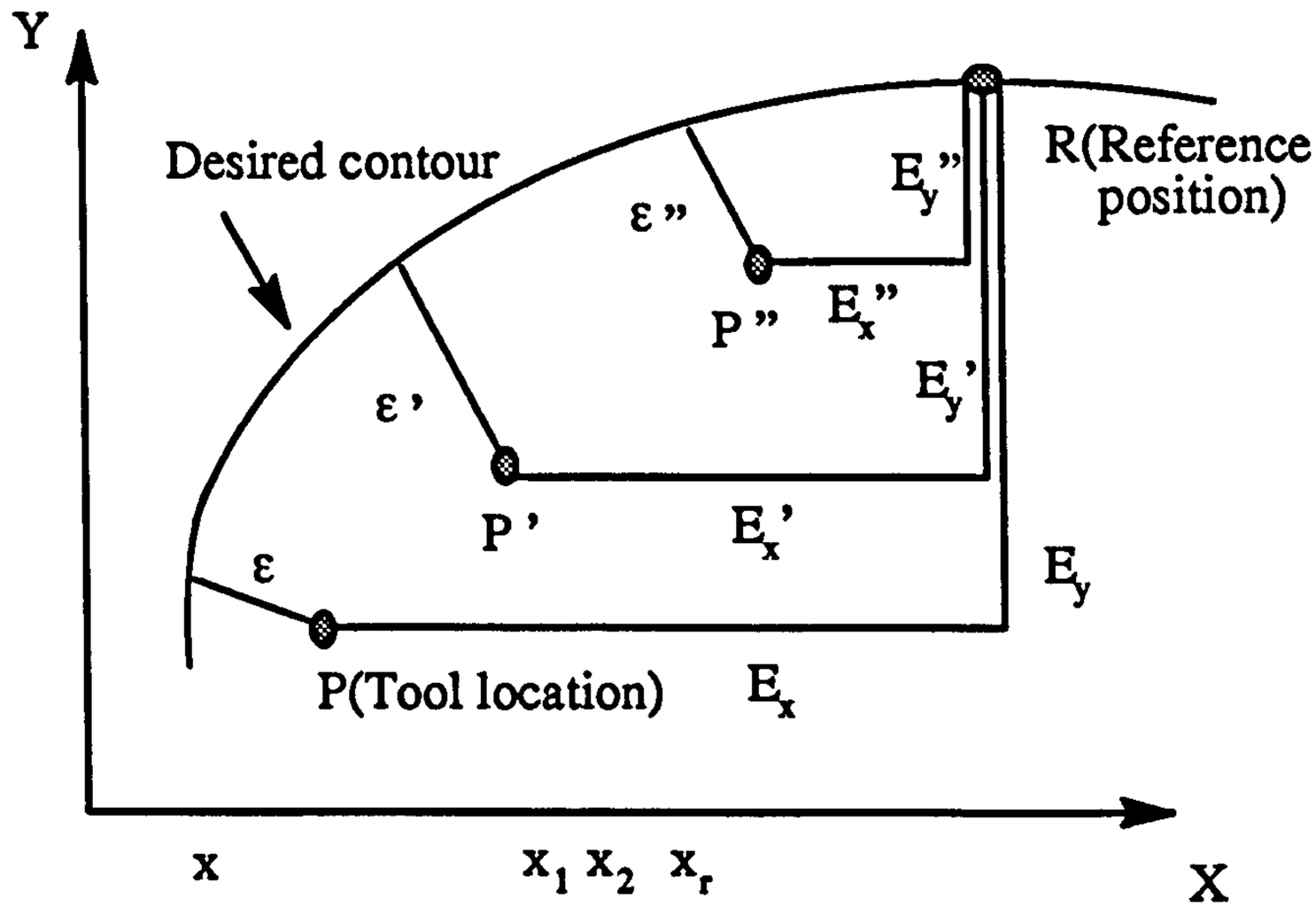


Figure 4.1 The axial and contour errors for different cutter locations

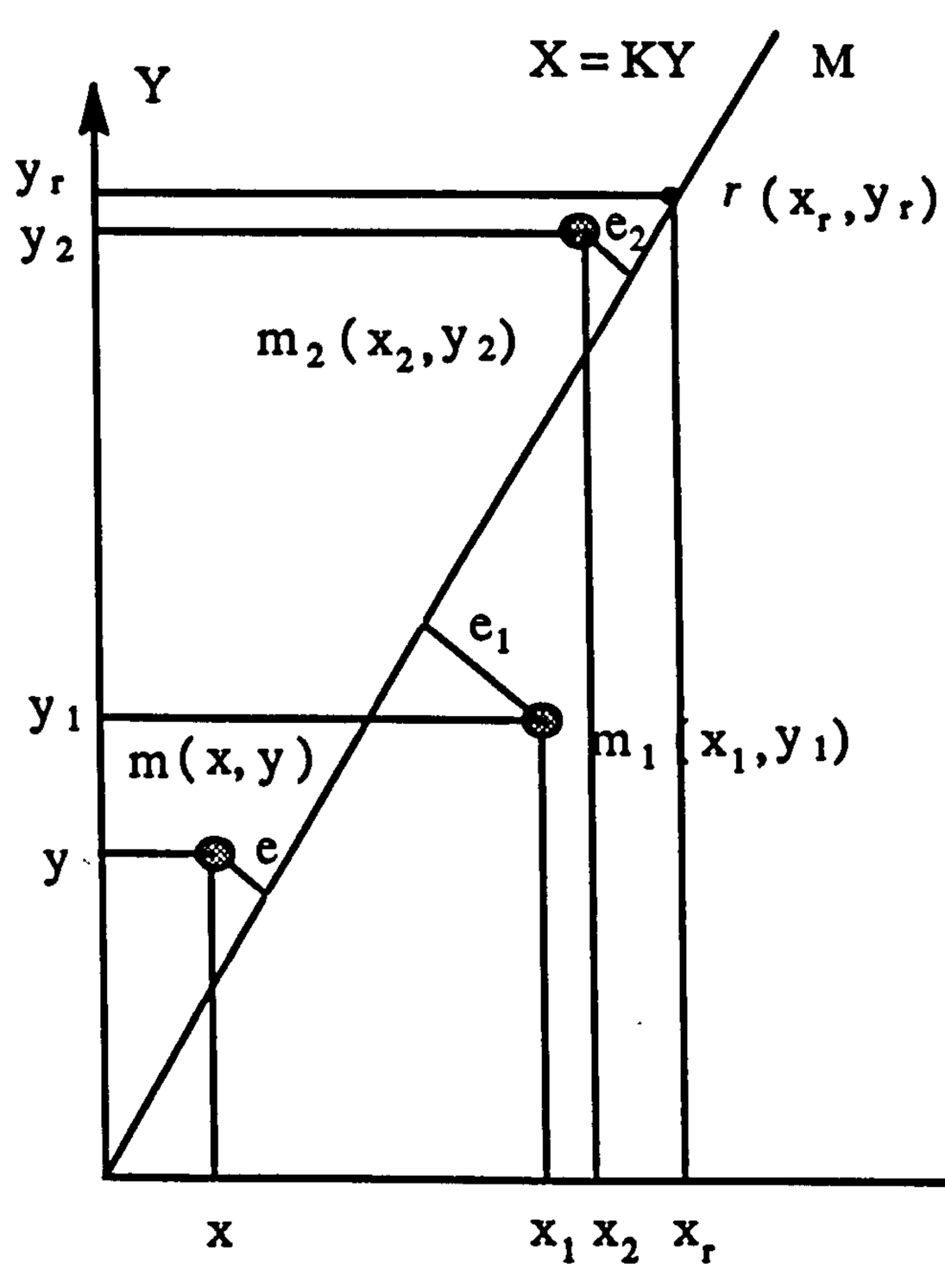


Figure 4.2 A Two Axis System with Linear Relationship X=KY

reference point r. Although the axial errors $(x_1 - x_r)$ and $(y_1 - y_r)$ at point m_1 are smaller than $(x - x_r)$ and $(y - y_r)$ at point m, the synchronisation error e_1 at m_1 is larger than that e at m.

4.2.2 Synthesis of the Currently Developed Multi-axis Motion Control and Synchronisation Methods

The study of current control methodologies for multi-axis motion control and synchronisation, described in Chapter 3, highlights that the basic nature of motion synchronisation control is through coupling or interconnecting the individual axes. Through the coupling, the dynamic information of the motions can be used to apply relevant actions back to the individual motion loops to achieve tight control, whereby an error in any axis affects the control loops of all axes. The major differences among the methods are the different connection types and the synchronisation control laws.

4.2.2.1 The Connections Between Motion Axes

The connections between the motion axes can be grouped as:

- Direct reference command coupling;
- Compensation coupling;
- Combination coupling.

(1) Direct Reference Command Coupling

'Independent drives control' and 'encoder tracking' approaches belong to this group. The 'independent drives control' has no feedback information coupling. The link among the axes are the mathematical expressions governing the motion relationships. It is the weakest coupling mechanism. Any error sources will separate the coupling. Actually, there is no real coupling in 'independent drives control'.

'Encoder tracking' has partial feedback, where the master's actual position is used as the slave's reference. This is a real direct reference command coupling. The feedback information is used in a forward chain linking of the axes.

(2) Compensation Coupling

'Cross-coupling control' and 'dual-loop control' can be included in this category. The compensation coupling uses compensation terms to couple the axes. The coupling controller is a compensator or coordinator, which is designed independently of the individual axes and dedicates the synchronisation control. The 'cross-coupling' methods provide compensation terms for each axis, but the 'dual-loop control' method usually only applies the compensation term to the 'slave' axes to maintain a specified synchronisation requirement.

(3) Combination Coupling

The 'MIMO method', 'system interconnection' method and 'scalar field control' are examples in this group. Combination coupling considers the whole system as a single entity, rather than in terms of its individual loops. In these methods, the couplings are introduced in different ways. The 'MIMO' method uses synchronisation control laws to 'couple' the axes. The 'system interconnection' uses the interconnected structure to link the axes. The 'scalar field control' uses scalar fields to couple the axes.

4.2.2.2 Control Laws for Motion Synchronisation

(a) Conventional cross-coupling algorithms can be developed by formulating the transfer function of the coupled system. However, in practice it is not always easy to describe the coupling by means of a discrete transfer function so as to realise ideal compensation. Thus, cross-coupling is normally obtained by the addition of a proportional-integral-derivative (PID) algorithm or a variation thereof[5]. However, such an approach is not ideal when subject to variable control environments and system nonlinearities.

(b) Commonly available MIMO system design methods will not introduce coupling into already decoupled systems (independent drives) that we wish to couple together[16]. Conventional multivariable control methods have to be extended to achieve the objective of synchronisation. A switching function or term expressing the synchronising objective has to be introduced into the system control law[6]. However, MIMO control approaches often tend to be discarded because of their increased complexity.

(c) The synchronisation control law in 'scalar field control' is hidden in the defined scalar fields. Building a precise scalar field in real-time is crucial to the synchronisation accuracy, particularly when the non-linearity and some uncertainness of the controlled processes exists in the system.

4.2.3 The Philosophy Behind Motion Coordination and Synchronisation Control

From the motion control viewpoint, coordination and synchronisation of any motions can only be obtained through some constraint. Mechanical systems achieve motion coordination and synchronisation over a wide range of normal and abnormal operating condition using cams, gears and linkages etc.[7]. The mechanical coupling is introduced by solid members, structure and parts configuration. Any disturbance on one axis will transmit to all axes and the relative reactions will also reflect back to all axes. As long as the transmission parts are robust to withstand the disturbance and reactions, the synchronisation of the motions can be maintained. When the mechanically linked systems are replaced by independent servo drives, the servo drives must be forced into coordination and synchronisation by an appropriate control system[8]. Hence the need for methodologies and tools to structure the control among the motions. Coupling or interconnecting the individual axes is the basic structure for motion synchronisation control.

The principle behind a coupling architecture is that independent intelligent drives perform in a dependent manner. This structure introduces reciprocal actions which provide the constraint.

In the mechanical systems, the constraint is the direct restriction force, because of the rigid structure. In software mechanisms, the constraint which regulates the motions are control actions which are based on the feedback information of the states of the whole system. These real-time reciprocal actions between the servo-drives can be provided by a closed-loop control.

4.2.4 Dual Closed-Loop Control in a Manufacturing Machine

High accuracies or compliant motion may be achieved when real-time closed-loop compensation is introduced in a manufacturing machine control system[9]. Figure 4.3 presents a conceptual view of a machine[10]. Such a machine employs servo-drives

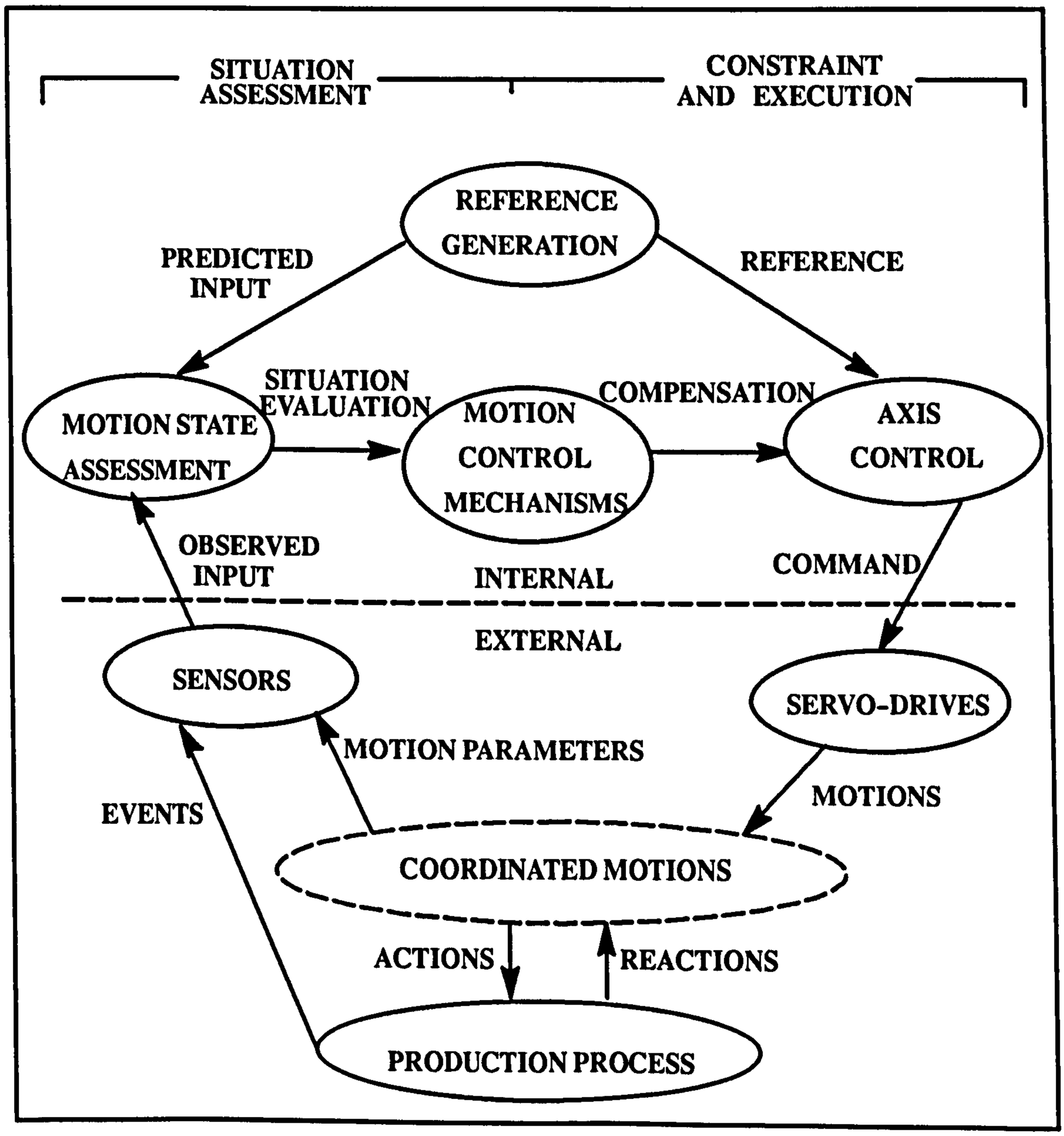


Figure 4.3 A Closed-loop Motion Control Machine System

with which motions can be generated. The motions perform actions on the production process. The machine also possesses sensors to monitor the current states of the coordinated motions and the process. A closed-loop motion control system is formed in the machine by inputting sensory data to Motion State Assessment (MSA) module[1], passing the processed information to the Motion Control Mechanisms (MCM) module, which provides the control of the coordination and synchronisation of the motions, and finally closing the loop through the Axis Control (AC) module which plans motions to be performed through the machine's servo-drives using continuous control algorithms.

In this machine each axis itself is under closed-loop control, an another supervisory closed-loop control for the all axes implies a 'dual closed-loop' control structure for the machine motion control system. This structure represents the motion synchronisation control philosophy described in previous section. It can be used as a general structure for a machine motion control system which requires a tight motion synchronisation control, therefore, this control structure is generic.

4.2.5 The Requirements for Intelligent Integration of Motion Control Mechanisms

The crucial part of a 'dual closed-loop' structure is the 'motion control mechanism' which determines the system's performance. In the 'dual closed-loop' structure, the closed-loop of each axis is designed to cope with the assigned task. Since different tasks are given to different axes, each axis' servo loop has a unique characteristic. Another challenging issue is the uncertainty of the controlled processes. This mechanism has to provide smooth and rapid action in order to react to the change of system state, whilst not causing system oscillation.

Conventional control based on modern analytical methods determines the control action in relation to a number of data inputs using a single set of equations to express the entire control process. Expressing these complex multi-axis motion synchronisation issues in the form of an explicit mathematical expression is very difficult, perhaps impossible[11][12]. As a general rule, a good engineering approach should be capable

---

1 · When we use the term ***module***, we will be describing a software component capable of performing one or more functions.

of making effective use of all the available information. If the mathematical model of a system is too difficult to obtain, then the most important information comes from two sources: 1) sensors which provide numerical measurements of key variables and 2) human experts who provide linguistic descriptions about the system and control instructions[22]. In the control of motion synchronisation between independent servo-drives, the sensor information and human intelligence (which is good in tackling many coordination problems) are available. The relationship states between the independent axes are easily described linguistically. Correction of deviations between the actual and preplanned trajectory, human intelligence can define appropriate actions based on a linguistical description. Because fuzzy inference control has proved ideal for expressing sophisticated knowledge of experts and incorporating valuable intuition[11][13][22], fuzzy logic is used as a tool to represent human linguistic descriptions as a basis for decision making.

4.3 A Fuzzy Logic Motion Synchronisation Algorithm

The basic principle of enforcing independent axes into coordination and synchronisation is that the lagging axis is accelerated to "catch" the leading axis and the leading axis is slowed down to "wait" for the lagging axis. These reciprocal actions have to take place in such a way that ensures undefined deviations from the preprogrammed reference path are avoided.

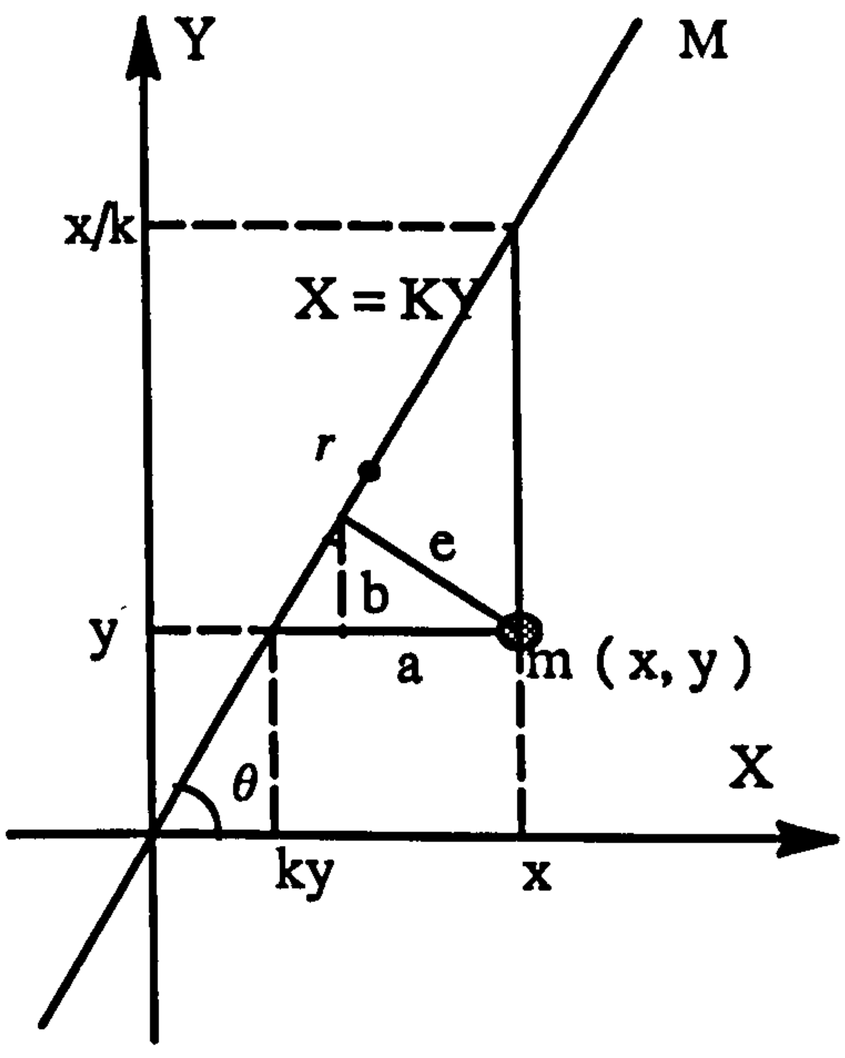
A fuzzy system consists of a bank of fuzzy associative memory (FAM) "rules" operating in parallel, and operating to different degrees[13]. Each rule represents ambiguous expert knowledge or learnt input-output transformations. An FAM rule can also represent the behaviour of a specific mathematical model. The system nonlinearly transforms exact or fuzzy state inputs to fuzzy-set outputs. These output fuzzy sets are usually "defuzzified" with a centriod operation to generate exact numerical outputs.

Expression for motion synchronisation are straight forward as it is only necessary to derive a set of FAM rules which produce the coupling between the axes. With execution of these 'rules', it is possible to obtain accurate synchronised motion control. In this section, a method using fuzzy logic to establish coupling between decoupled

independent drives in a multi-axis configuration is described. This fuzzy logic coupling algorithm is implemented within the Motion Control Mechanisms module.

4.3.1 Basic Structure of the Fuzzy Logic Synchronisation Algorithm

The desired relationship between N synchronised axes can be geometrically described as a hyperline or hyperpath in N-dimensional coordinates. Figure 4.4 shows a two-axis system with a linear relationship. For the reasons previously discussed the actual point $m(x, y)$ may not lie on the path M.



$$e = (x - ky) \sin\theta$$

$$\sin\theta = y / \sqrt{y^2 + (ky)^2}$$

$$= 1 / \sqrt{1 + k^2}$$

$$e = (x - ky) / \sqrt{1 + k^2}$$

$$\frac{a}{b} = \frac{1}{k}$$

$$a = e \sin\theta$$

$$a = (x - ky) / (1 + k^2)$$

$$b = (x - ky) k / (1 + k^2)$$

Figure 4.4 A Two Axis System with Linear Relationship, X=KY

In order to meet synchronisation requirements, the actual point m should be '*forced*' back to the path M (ie. to eliminate e.). For each axis, the modified value will depend on the other axis'. For example, if axis X reduces by a, then axis Y should increase by b. If axis X reduces by less than a, then, axis Y must increase by more than b. In an extreme case, when one axis has already reached saturation, the other axis has to take complete responsibility to maintain synchronisation. For a two-axis control system as described in Figure 4.5, the compensations can be added to the local controllers' output u_i or their input r_i. The direct modification of the controllers' reference command is a straightforward method which does not involve changing the system configuration. It is, however, very difficult to design a high level mechanism which can map the inputs (synchronisation error e and references r_i) to the outputs (modified values rm_{im} for each

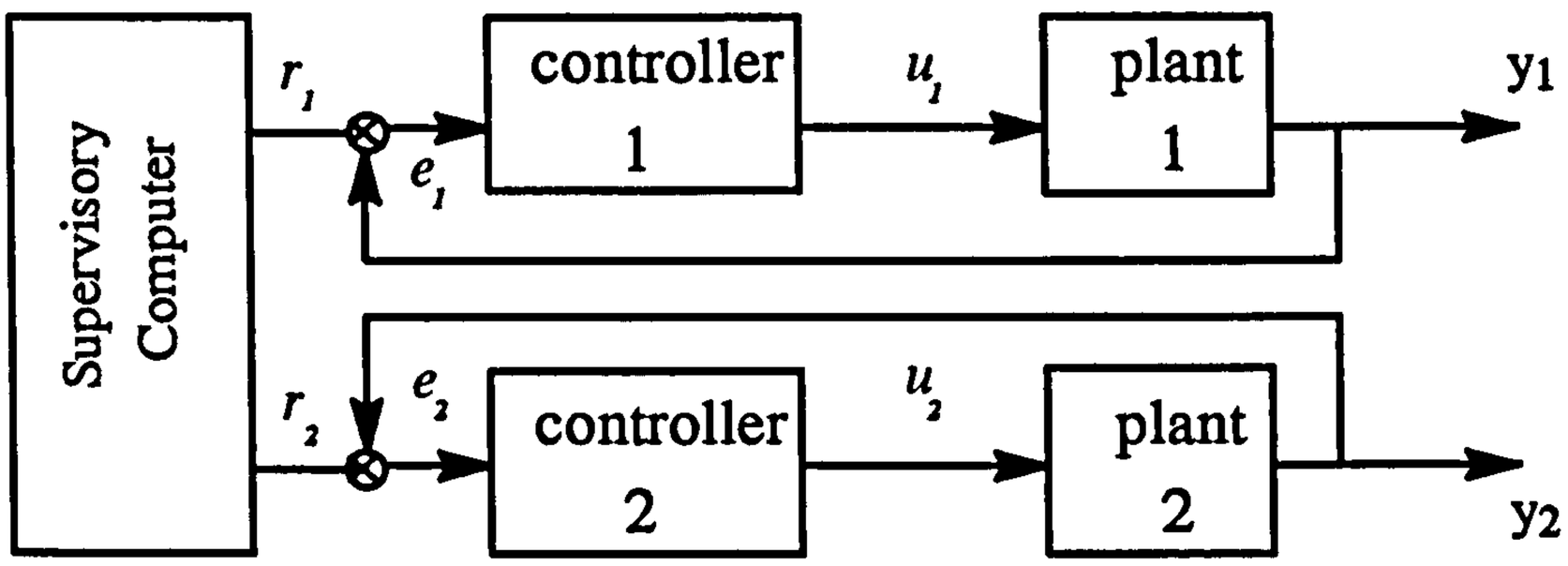


Figure 4.5 A Two-axis Motion Control System

axis' reference). Here, we use a set of fuzzy logic rules to couple the servo axes. Figure 4.6 shows the basic control structure with a fuzzy rule-based coupling mechanism.

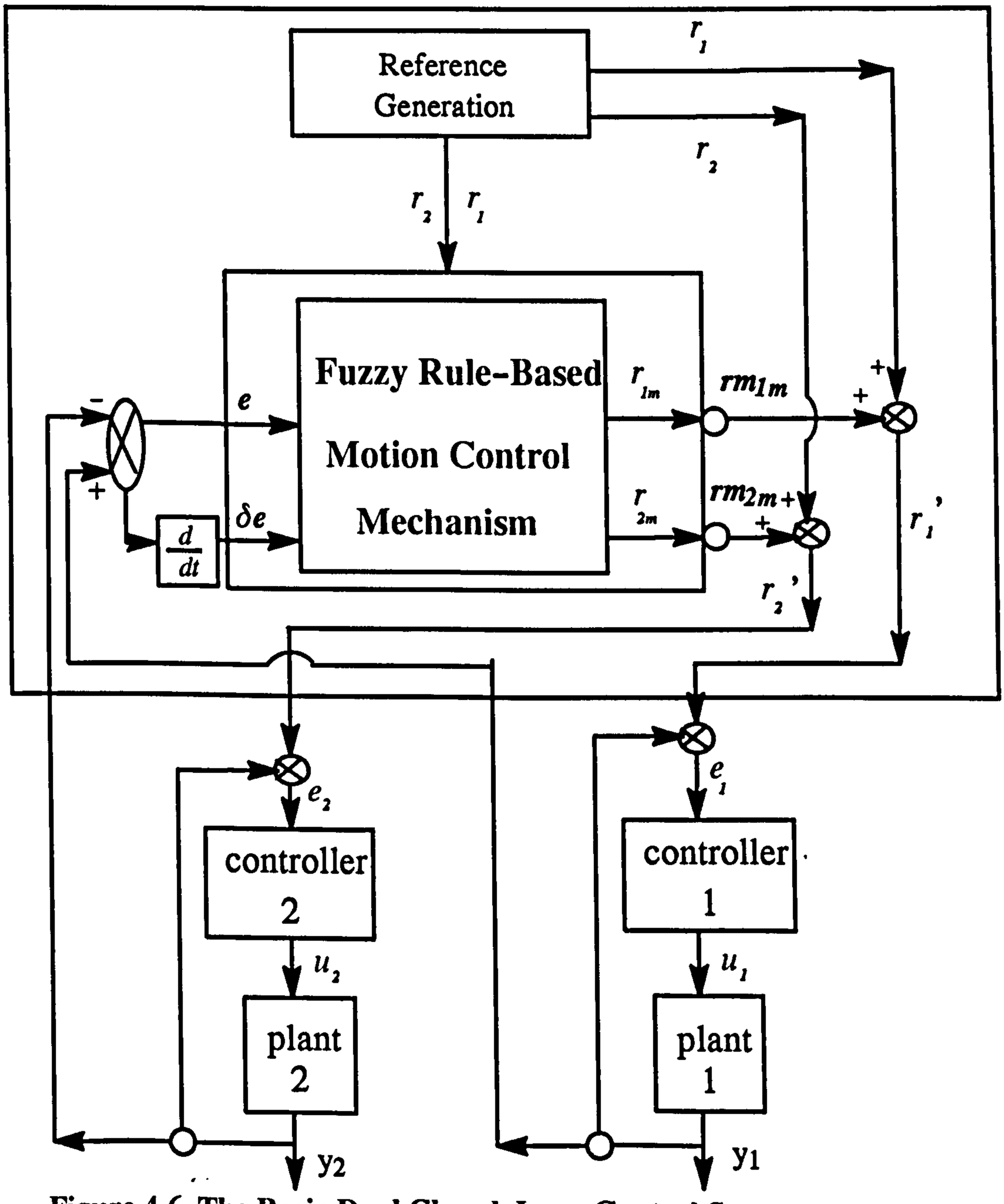


Figure 4.6 The Basic Dual Closed-Loop Control Structure with a Fuzzy Rule-Based Coupling Mechanism

The block diagram of the proposed coupling algorithm, as shown in Figure 4.6, has an architecture where axis position feedback is provided to the supervisory level. The design issue here is how to map the measured synchronisation error vector to the demand position compensation vector so that synchronisation can be controlled as accurately as possible. The procedure involves the following elements:

- The synchronisation error and the rate of change of error are described by fuzzy membership functions which have trapezoidal or triangle profiles.
- The demand position compensation for each axis is also represented by a fuzzy membership function.
- The mapping rules between synchronisation error and corrective actions are heuristically constructed from the input–output data with the objective to reduce or eliminate the synchronisation error.

The conventional method to accomplish these procedures involves establishing: (1) fuzzification; (2) a rule base; (3) an inference mechanism; and (4) defuzzification modules.

(1) Fuzzification

Fuzzification is the process of assigning or calculating a value to represent an input's degree of membership in one or more qualitative groupings, called "fuzzy sets". For the fuzzy coupling, the input variables are chosen as

e: the synchronisation error

δe: the rate of change of synchronisation error,

while the output variables (assuming a two axis system) are selected to be

r_{1m}: the demand position compensation to axis one

r_{2m}: the demand position compensation to axis two.

The fuzzy variables are distinguished from their finite variables by putting a sign (–) above the variable. Therefore, the fuzzified variables of e, δe, r_{1m} and r_{2m} are $\bar{e}$, $\overline{\delta e}$, $\bar{r}_{1m}$ and $\bar{r}_{2m}$, respectively. These fuzzy variables are defined by the universes of discourse (E, δE, R_{1m} and R_{2m} respectively) which have the ranges shown in Figure 4.7. Each

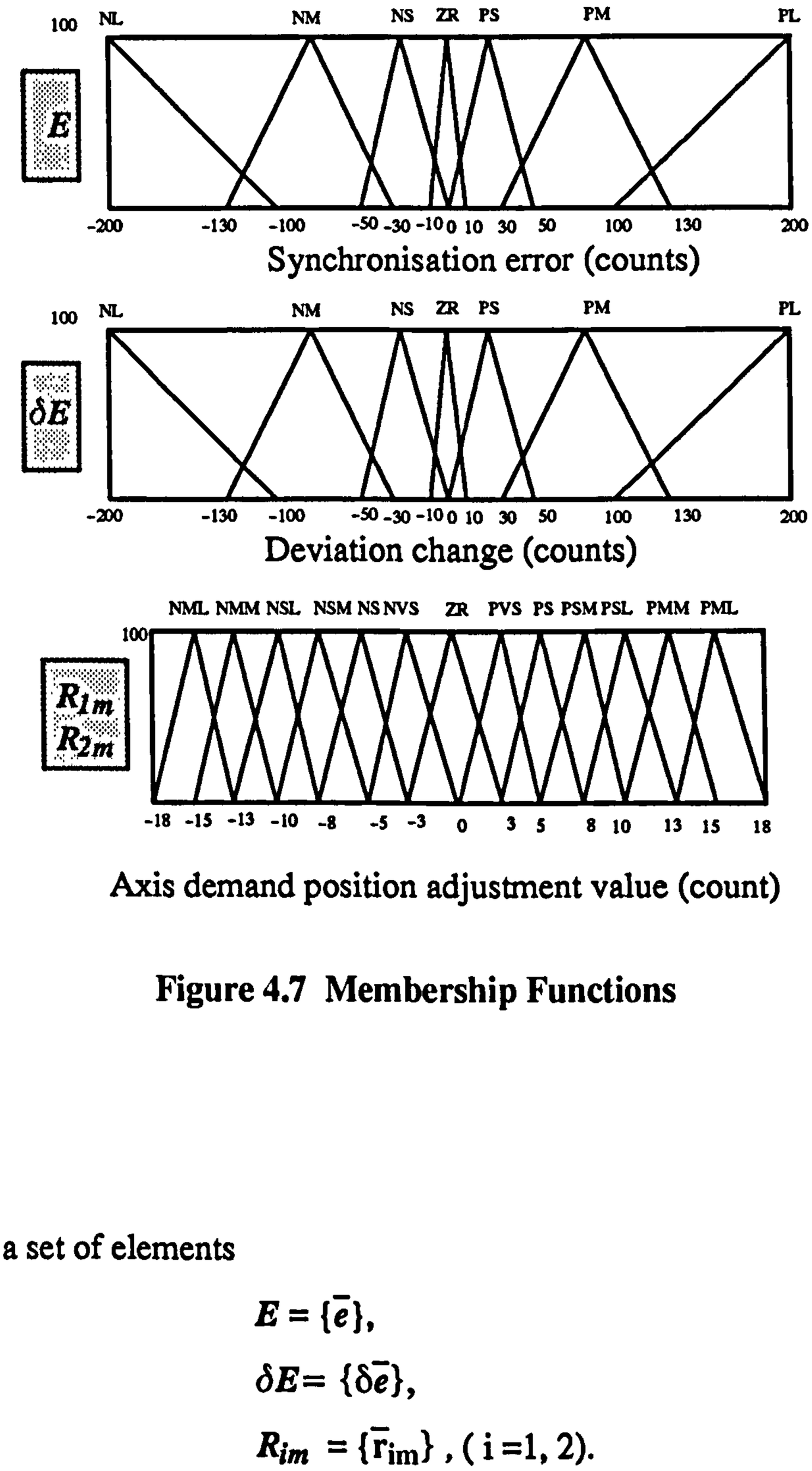


Figure 4.7 Membership Functions

universe is a set of elements

$$E = \{\bar{e}\},$$
$$\delta E = \{\delta\bar{e}\},$$
$$R_{im} = \{\bar{r}_{im}\}, (i = 1, 2). \qquad (4\text{–}1)$$

The fuzzy subsets, E^k, dE^k, R^k_{1m} and R^k_{2m}, are defined by a set of ordered pairs, ie

$$E^k = \{ (\bar{e}, \mu_E{}^k(\bar{e})) \} \subset E$$
$$\delta E^k = \{ (\overline{de}, \mu_{dE}{}^k(\delta\bar{e})) \} \subset \delta E \qquad (4\text{–}2)$$
$$R^k_{im} = \{ (\bar{r}_{im}, \mu_{Rim}{}^k(\bar{r}_{im})) \} \subset R_{im},$$

where $\bar{e}$, $\overline{\delta e}$, $\bar{r}_{1m}$ and $\bar{r}_{2m}$ are the elements of the universes and $\mu_E{}^k(\bar{e})$, $\mu_{dE}{}^k(\delta\bar{e})$ and $\mu_{Rim}{}^k(\bar{r}_{im})$ are the corresponding membership values which give the degree to which the element is a member of the subset.

As shown in Figure 4.7, we have chosen seven fuzzy subsets for $\bar{e}$, $\overline{\delta e}$ and thirteen fuzzy subsets for the output $\bar{r}_{1m}$ and $\bar{r}_{2m}$. The primary fuzzy sets defined are abbreviated as follows:

PML: positive medium large
PMM: positive medium medium
PSL: positive small large
PL: positive large
PSM: positive small medium
PS: positive small
PVS: positive very small
ZR: zero.

NML: negative medium large
NMM: negative medium medium
NSL: negative small large
NL: negative large
NSM: negative small medium
NS: negative small
NVS: negative very small

(2) Rule Base Construction and Derivation

The fuzzy algorithm requires that we articulate or estimate the FAM rules which construct the coupling among the servo-drives. With the help of the previous linguistic variables, the coupling algorithm is developed using production rules of the type "if A then B". The fuzzy control rules provide a natural framework for the characterisation of human behaviour and decisions analysis.

The kth fuzzy rule is constructed as

R^k : IF $\bar{e}$ is E^k, $\overline{\delta e}$ is δE^k ; THEN $\bar{r}_{1m}$ is $R^k{}_{1m}$ and $\bar{r}_{2m}$ is $R^k{}_{2m}$.

Here, we use one example to show the rule derivation procedure for motion synchronisation control. Consider two motors under position control mode with a 1:1 ratio relationship.

For example, IF $\bar{e}$ is ZR, $\overline{\delta e}$ is NL;

In this case, when $\bar{e}$ is ZR, the actual positions of these two motors are very close, ie the synchronisation error e is very small. However, at this time $\overline{\delta e}$ is NL, and

δe = current (e) – past interval synchronisation error (e_p),

Since the synchronisation error e is very small, one case which may have $\overline{\delta e}$ at NL is that e is negative (< 0), and e_p is positive (> 0).

If $e < 0$, ie. $y_1 < y_2$.

If $e_p > 0$, ie. $y_{1p} > y_{2p}$. (in past interval)

Therefore, from the past interval to current time, the speed of motor 1 is reducing, and the speed of motor 2 is rising.

Another case which can make $\overline{\delta e}$ at NL is that e is positive (> 0), and e_p is also positive (> 0) but with a large value.

If $e > 0$, with a small value, ie. y_1 is slight big than y_2.

If $e_p > 0$, with a large value, ie. y_{1p} is much big than y_{2p}. (in past interval)

Therefore, we have the same conclusion as the previous case: during the last interval, the speed of motor 1 is reducing, and the speed of motor 2 is rising.

Based on the above analysis, we can derive one rule which is used to control the motion synchronisation (slowing axis has a positive compensation value, rising axis has a negetive compensation value.):

IF $\bar{e}$ is ZR, $\overline{\delta e}$ is NL ; THEN $\bar{r}_{1m}$ is PML and $\bar{r}_{2m}$ is NML.

Appropriate rules are readily derived using such an analogy.

Therefore, the overall rule base has the form

$$R = \{R^1, R^2, \ldots, R^k, \ldots, R^{49}\}. \qquad (4\text{-}3)$$

Figure 4.8 shows a rule base which is composed of a set of sub-rule-bases. This property can be used to implement the proposed algorithm.

$\bar{e}$ \ $\overline{\delta e}$; $\bar{r}_{1m}$ / $\bar{r}_{2m}$	NL	NM	NS	ZR	PS	PM	PL
NL	PML / NML	PMM / NMM	PSL / NSL	PSM / NSM	PSM / NSM	PS / NS	PS / NS
NM	PMM / NMM	PSM / NSM	PSM / NSM	PS / NS	PS / NS	PVS / NVS	PVS / NVS
NS	PSM / NSM	PS / NSM	PVS / NVS	PVS / NVS	PVS / NVS	ZR / ZR	ZR / ZR
ZR	PSM / NSM	PS / NS	PVS / NVS	ZR / ZR	NVS / PVS	NS / PS	NSM / PSM
PS	ZR / ZR	ZR / ZR	NVS / PVS	NVS / PVS	NVS / PVS	NS / PS	NSM / PSM
PM	NVS / PVS	NVS / PVS	NS / PS	NS / PS	NSM / PSM	NSL / PSL	NMM / PMM
PL	NS / PS	NS / PS	NSM / PSM	NSM / PSM	NSL / PSL	NMM / PMM	NML / PML

Figure 4.8 The FAM Bank for the Fuzzy Logic Coupling Mechanism

(3) Evaluation of Rules

To govern the system's behaviour, a set of rules that have the form of if–then statements have been developed, as shown in Figure 4.8. The if side of a rule contains one or more conditions, called "antecedents"; the then side contains one or more actions, called "consequences".

Each antecedent has a degree–of–truth (membership) value assigned to it as a result of fuzzification. During rule evaluation, strengths are computed based on antecedent values and then assigned to the rules' fuzzy outputs. Generally, a minimum function is used so that the strength a rule is assigned the value of its weakest or least true antecedent. Often, more than one rule applies to the same specific action, in which case the common practice is to use the strongest or most true rule, ie use a maximum function[14][15].

For the result of the *k*th fuzzy rule, the membership grade function $\mu_{Rim}{}^{k}(\bar{r}_{im})$ can be obtained by

$$\mu_{Rim}{}^{k}(\bar{r}_{im}) = \text{MIN}\,[\mu_{E}{}^{k}(\bar{e}), \mu_{dE}{}^{k}(\delta\bar{e}), \mu_{Rim}{}^{k}(\bar{r}_{im})]. \qquad (4\text{–}4)$$

From the results of the 49 rules, the final membership grade function $\mu_{Rim}(\bar{r}_{im})$ is obtained using the MAX operator:

$$\mu_{Rim}(\bar{r}_{im}) = \text{MAX}\,[\mu_{Rim}{}^{1}(\bar{r}_{im}), \mu_{Rim}{}^{2}(\bar{r}_{im}), \ldots, \mu_{Rim}{}^{k}(\bar{r}_{im}), \ldots, \mu_{Rim}{}^{49}(\bar{r}_{im})]. \qquad (4\text{–}5)$$

(4) Defuzzification of Outputs

Even though the rule–evaluation process assigns strengths to each specific action, further processing, or "defuzzification", is required for two reasons. The first is to decipher the meaning of vague (fuzzy) actions using membership functions. The second is to resolve conflicts between competing actions which may have been triggered by certain conditions during rule evaluation. Defuzzification employs compromising techniques to resolve both the vagueness and conflict issues.

One common defuzzification technique, the "centre–of–gravity method"[14], consists of several steps. Initially, a centroid point $r_{im}{}^{j}$ is determined for each output membership function. Then, the membership functions are limited in height by the applied rule

strength ($\mu_{Rim}(\bar{r}_{im}{}^{j})$), and the area $S(\mu_{Rim}(\bar{r}_{im}{}^{j}))$ of the membership function $\bar{r}_{im}{}^{j}$ is computed. Finally, the defuzzified output r_{im} is derived by a weighted average of the centroid points and the computed areas, with the areas serving as the weights. The centre-of-gravity method is illustrated in the following equation.

$$r_{im} = \frac{\sum_{j=1}^{p} S(\mu_{Rim}(\bar{r}_{im}{}^{j})) \cdot r_{im}{}^{j}}{\sum_{j=1}^{p} S(\mu_{Rim}(\bar{r}_{im}{}^{j}))} \qquad (4\text{–}6)$$

where p is the number of fuzzy subsets of the output.

(5) System Outputs

The actual compensation for each axis depends on the ratio of the relationship between the axes. For the two axes with a relationship X=KY, the compensation terms for the axis commands are

$$rm_{1m} = \frac{\sum_{j=1}^{p} S(\mu_{R1m}(\bar{r}_{1m}{}^{j})) \cdot r_{1m}{}^{j}}{\sum_{j=1}^{p} S(\mu_{R1m}(\bar{r}_{1m}{}^{j}))} \cdot C_x$$

$$rm_{2m} = \frac{\sum_{j=1}^{p} S(\mu_{R2m}(\bar{r}_{2m}{}^{j})) \cdot r_{2m}{}^{j}}{\sum_{j=1}^{p} S(\mu_{R2m}(\bar{r}_{2m}{}^{j}))} \cdot C_y \qquad (4\text{–}7)$$

where

$$C_x = 1 / (1 + k^2)$$

$$C_y = k / (1 + k^2)$$

4.3.2 A Fuzzy Rule Base for Closed Loop Master-Slave Linked Motion Synchronisation

In the previous section, we assume both two axes' commands can be adjusted to achieve tight synchronisation control, which can be considered as an "Equal-Status" approach. In the "Master-Slave" approach, the slaves' commands can be changed, since the master's actual position is used as a reference for all slaves axes, therefore, some modifications is needed to the general fuzzy logic synchronisation algorithm described in the previous section.

(a) The Synchronisation Error

In master-slave configurations, the slave has to take complete responsibility in preserving synchronisation. In this case, error **b** (rather than **e**) is reduced or eliminated to control the synchronisation. Figure 4.9 illustrates the relationships.

The input variables are chosen as

b: the synchronisation error

δb: the rate of change of the synchronisation error,

while the output variable is selected to be

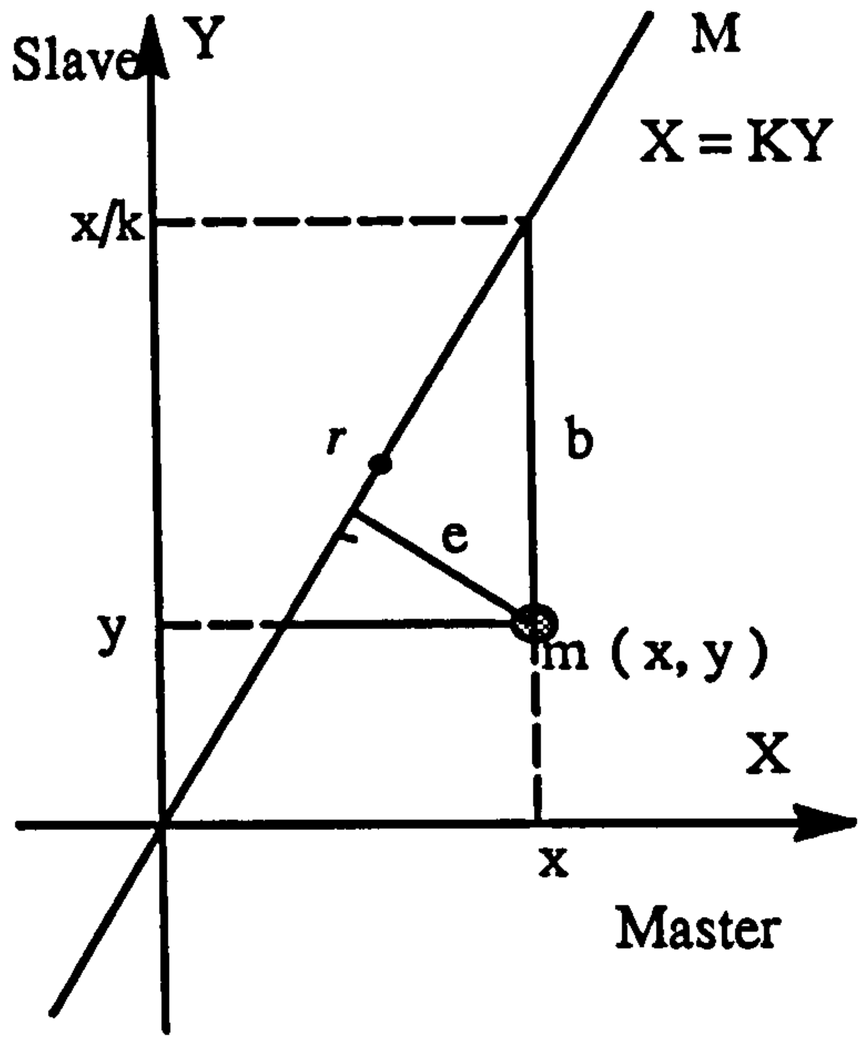
v: the slave demand position compensation term for the next interval.

(b) Predictive Action for Slaves

In order to reduce synchronisation error, a compensation term has to provide a corrective action towards the desired synchronisation path. A control scheme is described here where the slave command consists of the reference command plus a compensation term. The synchronisation error can be reduced by providing such a compensation term to the slave command. This predictive action must be designed to ensure the error is always reduced.

(c) Closed-Loop Master-Slave Structure

Because one axis follows the output of another axis, the method is still classed as master-slave. However, the addition of the compensation term implies a closed-loop



$$e = (x - ky) / \sqrt{1 + k^2}$$

$$a = 0$$

$$b = y - x/k$$

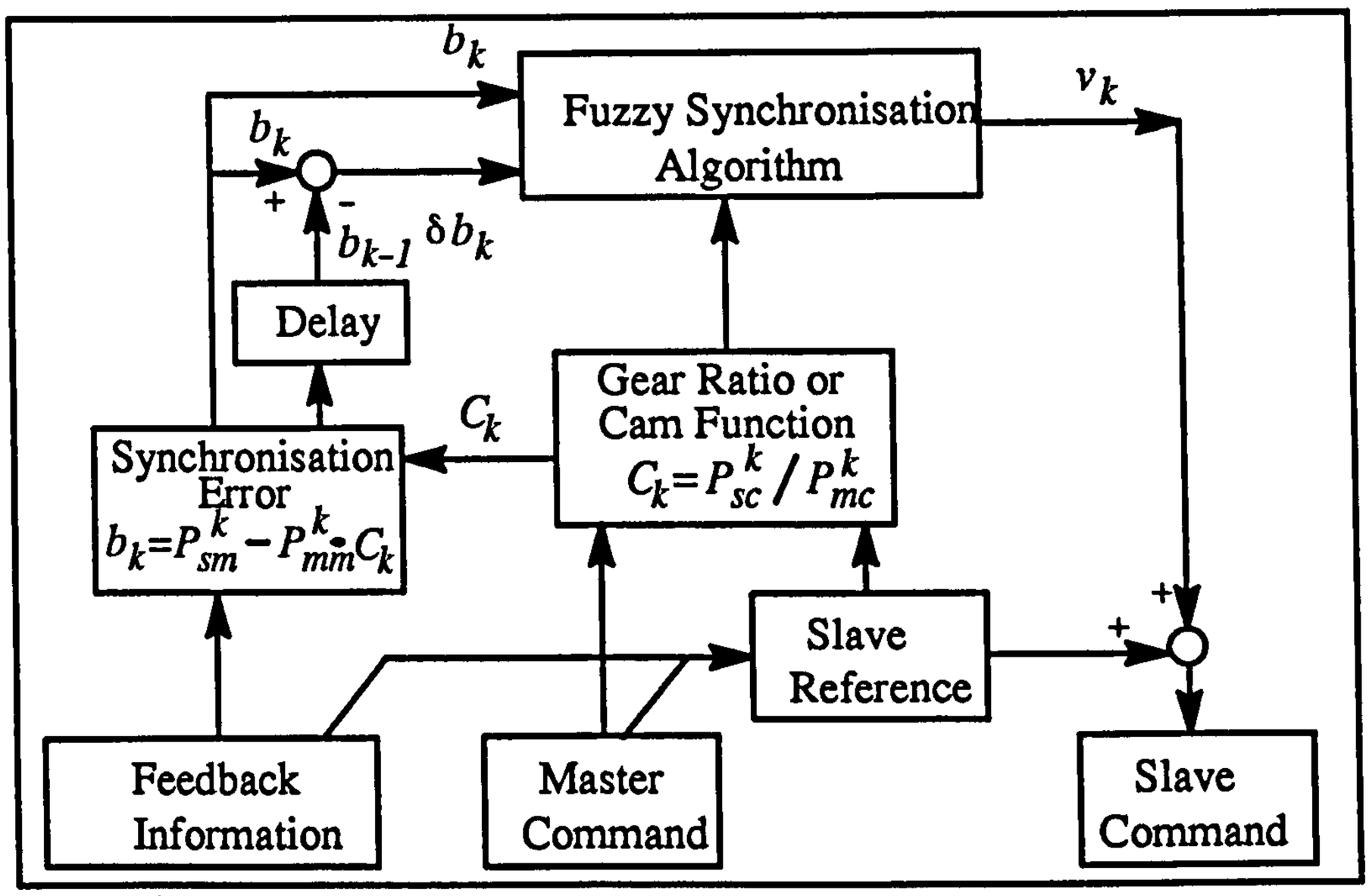
Figure 4.9 A Two Axis Master-Slave System with a Linear Relationship, X=KY

master-slave method. Figure 4.10 shows the basic structure of the close-loop master-slave strategy.

(d) A Fuzzy Logic Rule Base for Closed-Loop Master-Slave Systems

For the closed-loop master-slave problem, the target of reducing the synchronisation error is achieved by the slave drives dynamically matching the master drive by modifying the input command to the slave's. This predictive action is based on the synchronisation error ***b***, the rate of change of the synchronisation error δb and knowing the relationship between the master and slave axes.

The synchronisation error ***b*** from the previous interval can not be removed. However, it is a goal to be corrected during the next interval to prevent accumulated errors. The



where,

P^k_{mc} -- master command position

P^k_{mm} -- master measured position

P^k_{sc} -- slave command position

P^k_{sm} -- slave measured position

Figure 4.10 Fuzzy Logic Based Closed-Loop Master-Slave Structure

rate of change of the synchronisation error δb represents the trend in the error change. It is important to use this information in a preemptive action. Based on an understanding of these input variables, the derivation of the fuzzy logic rules is not difficult. Here, heuristic rules are used to generate the required slave command for minimum synchronisation error. If the state point is to the left side of the path, then slave axis has to slow down requiring a negative compensation in the next interval. If the state point is to the right side of the path, then slave axis has to speed up requiring a positive compensation term.

Figure 4.11 shows a typical rule base. This property can be used to implement the proposed closed-loop master-slave algorithm.

$\dot{b}$ \ $\overline{\delta b}$	NL	NM	NS	ZR	PS	PM	PL
NL	PML	PMM	PSL	PSM	PSM	PS	PS
NM	PMM	PSM	PSM	PS	PS	PVS	PVS
NS	PSM	PS	PVS	PVS	PVS	ZR	ZR
ZR	PSM	PS	PVS	ZR	NVS	NS	NSM
PS	ZR	ZR	NVS	NVS	NVS	NSM	NSM
PM	NVS	NVS	NS	NS	NSM	NSM	NMM
PL	NS	NS	NSM	NSM	NSL	NMM	NML

Figure 4.11 A Fuzzy Rule Base for Closed-Loop Master-Slave Mechanism

With the exception of the modifications outlined above, the procedure to produce the fuzzy algorithm is the same as described in the previous section.

4.3.3 Fuzzy Logic Coupling Mechanisms for Nonlinear Motion Synchronisation

For general nonlinear motion synchronisation, the most important task is to identify accurately the synchronisation error in a sufficiently short time. We may choose an iterative method (eg Newton-Raphson) to find the synchronisation error[5][16][17]. However, an iterative approach generally needs a large amount of computation and is normally only suitable for off-line calculations, in non real-time systems. Several other methods are suggested in [16] with more general discussions. Here, a method proposed in [5] was chosen to calculate the synchronisation error. In this method, the nonlinear path is locally approximated by a circle as shown in Figure 4.12. The reasons for selecting this method are

- reasonable accuracy in calculation of the synchronisation error,
- simplicity in real-time computation, and
- suitable for the proposed fuzzy logic coupling algorithm.

For nonlinear motion synchronisation, the proposed fuzzy logic synchronisation algorithm is used with an "Equal-Status" viewpoint. The fuzzy logic synchronisation algorithm described in section 4.3.1 has to be generalised to provide for nonlinear relationship.

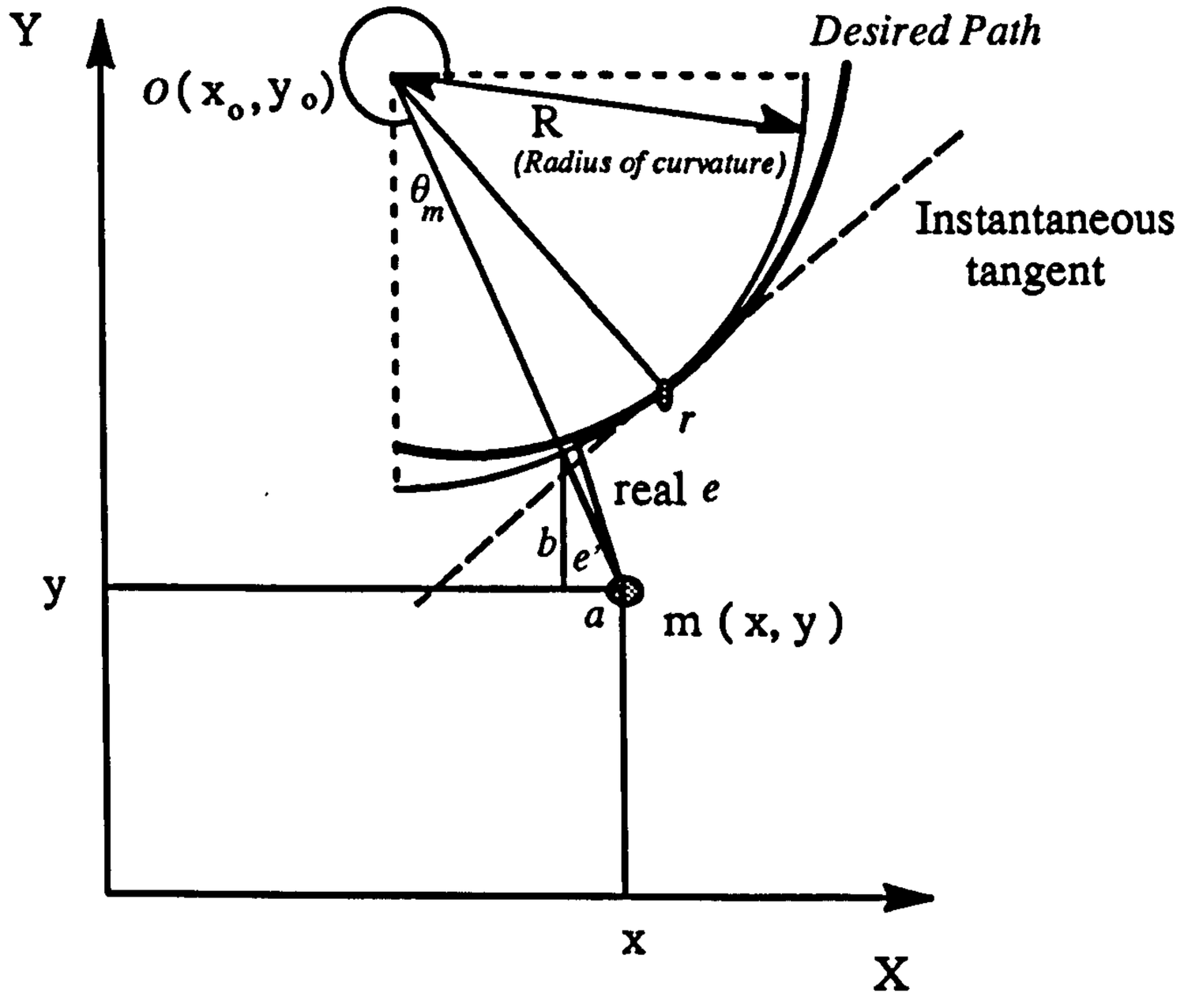


Figure 4.12 The Synchronisation Error Model

(a) The Synchronisation Error

In Figure 4.12, the synchronisation error is approximated by the distance between the actual position and the circle, and can be calculated by the following equation:

$$e \approx e' = \sqrt{(x - x_o)^2 + (y - y_o)^2} - R \qquad (4\text{-}8)$$

where R is the instantaneous radius of curvature; (x_o, y_o) is the corresponding centre of the circle; and the actual position is m(x,y).

(b) The Coefficients for Compensation Terms

The fuzzy algorithm described in section 4.3.1 is for a linear relationship between the axes. When it is used for nonlinear relationships, the coefficients for the system outputs change due to the nonlinear path. These coefficients are directly related with θ_m and expresses as

$$C_x = \cos\theta_m;$$
$$C_y = -\sin\theta_m; \qquad (4\text{–}9)$$

In Appendix II, a very detailed derivation of the above equations (4–9) is given.

With the exception of these modifications, the procedure to produce the fuzzy algorithm is unaltered.

4.4 Analysis of the Fuzzy Logic Synchronisation Algorithm

4.4.1 Characteristic Analysis of the Fuzzy Logic Synchronisation

Basically, each control system maps inputs to outputs, these input–output transformations are then used geometrically to define control surfaces. The fuzzy control surface characterises the system's fuzzy–set value definitions and its bank of FAM rules. In this section, we study the control surfaces and contrast them with the scalar field control approach to analyse the coupling characteristics.

4.4.1.1 Fuzzy Control Surfaces

Different sets of FAM rules yield different fuzzy controllers, and hence different control surfaces[13]. Figure 4.13 shows a FAM bank which is configured using common sense and engineering judgment and experience (ie speed up the slow axis and slow down the fast axis to maintain synchronisation between the axes). Each entry in this linguistic matrix represents one FAM rule.

The entire FAM bank determines how the system maps input fuzzy sets to output fuzzy sets. The fuzzy–set membership functions shown in Figure 4.14 determines how each finite input value translates to each fuzzy–set value. So both the fuzzy–set value

$\overline{\delta e}$ / $\overline{R_{1m}}$ / $\overline{R_{2m}}$ / $\overline{e}$	NL	NM	NS	ZR	PS	PM	PL
NL	PL / NL	PL / NL	PM / NM	PL / NL	PM / NM	PM / NM	PL / NL
NM	PL / NL	PL / NL	PM / NM	PM / NM	PM / NM	PM / NL	PL / NM
NS	PM / NM	PM / NM	PM / NM	PS / NS	PS / NM	PS / NM	PM / NM
ZR	PM / NM	PS / NS	PS / NS	ZR / ZR	NS / PS	NS / PS	NM / PM
PS	NM / PM	NS / PM	NS / PM	NS / PS	NM / PM	NM / PM	NM / PM
PM	NL / PM	NM / PL	NM / PM	NM / PM	NM / PM	NL / PL	NL / PL
PL	NL / PL	NM / PM	NM / PM	NL / PL	NM / PM	NL / PL	NL / PL

Figure 4.13 The FAM Bank of a Fuzzy Logic Coupling Mechanism

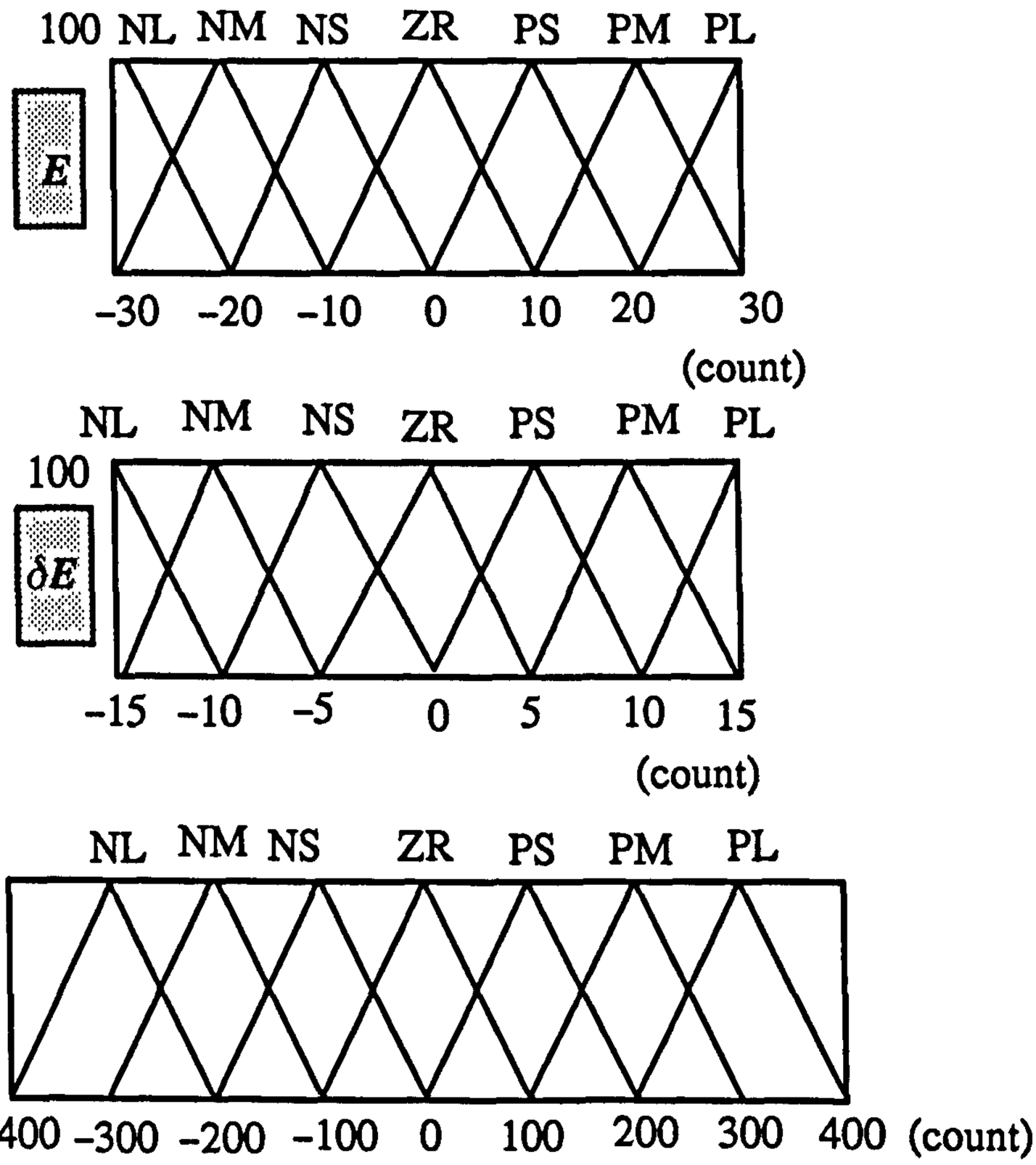


Figure 4.14 Fuzzy Input/Output Membership Functions

definitions and the FAM bank determine the defuzzified outputs for any finite set of input values.

The set of all possible inputs and outputs defines a FAM surface in the input-output product space. Figure 4.15 shows the control surfaces of the fuzzy logic coupling mechanism. The control outputs are plotted against the inputs e and δe. If the deviation

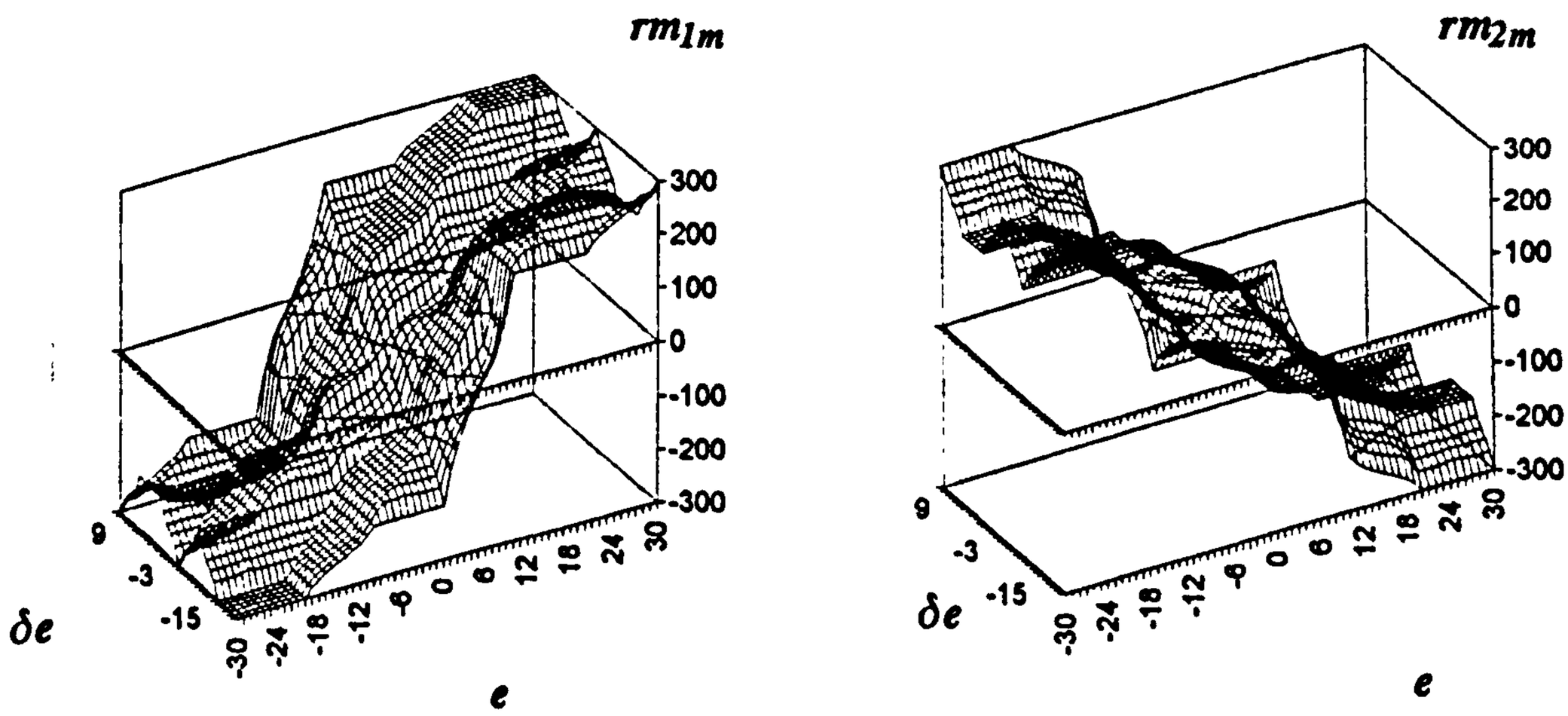


Figure 4.15 Control Surface of the Fuzzy Logic Coupling Mechanism

change $\delta e = 0$, we obtain a control line instead of control surface. Figure 4.16 shows the control lines of the fuzzy logic coupling mechanism for $\delta e = 0$.

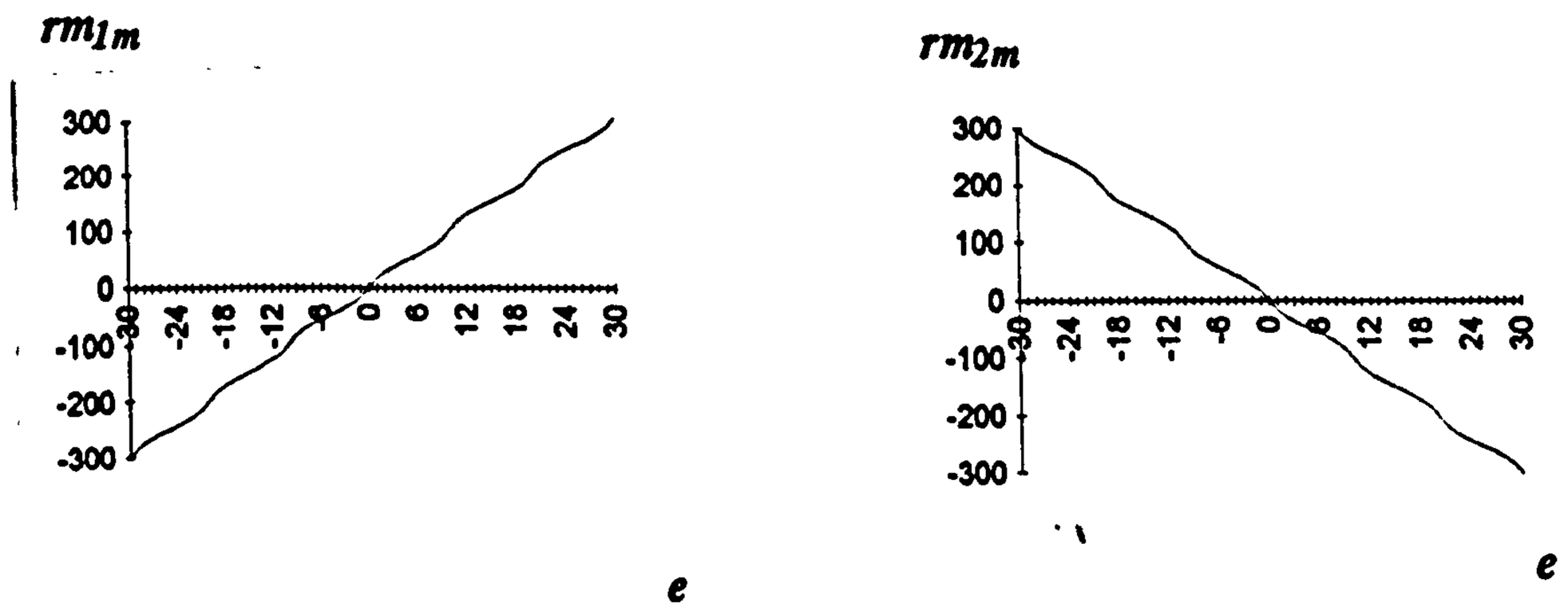


Figure 4.16 Control Lines of the Fuzzy Logic Coupling Mechanism for $\delta e = 0$

If we consider the control lines in Figure 4.16 as the basic profile, one may find that due to δe the control lines need to vary from the basic line. The control surface is the collection of these control lines as shown in Figure 4.15. From the maps, we can also see that for a synchronisation error on an axis which is slow receives a positive compensation, whereby an axis which is fast receives a negative compensation. The fuzzy logic coupling mechanism acts in this way to reduce the synchronisation error through the linguistic representation of the coupling rules.

4.4.1.2 Scalar Field Control

From the ***Gradient of Scalar Function*** mathematical definition, if the scalar function $\Phi(x,y)$ is a continuously differentiable function with respect to the independent variables x and y,the gradient of Φ is defined by the vector[18]

$$grad\ \Phi(x,y) = \frac{\partial \Phi}{\partial x}\ i + \frac{\partial \Phi}{\partial y}\ j \qquad (4\text{–}10)$$

A scalar field Φ could represent a potential energy field, the vector field **F** (**F**=grad Φ) may be regarded as a conservative force field. Scalar field control is one kind of axis coupling strategy created through a physical appreciation of requirements. A physical surface Φ can provide a potential energy field which acts on a 'ball' which is on the surface. Due to the influence of gravity, the ball seeks out an equilibrium position on a path. The forces on the 'ball' acting to take it to the required path may be resolved in the coordinate axes directions. These forces are related to the slopes of the surface resolved in these directions. Mathematically, the respective resolved slopes may be expressed using partial derivatives. A similar idea is used to make a position control system achieve a desired position relationship[16]. The partial derivative expressions ($\partial\Phi/\partial x$, $\partial\Phi/\partial y$) may then be used as a form of error upon which the controllers act.

From the ***total differential theorem***, the total differential $d\Phi$ of the function $\Phi(x,y)$ in the region Ψ is given by

$$d\Phi = \frac{\partial \Phi}{\partial x}\ dx + \frac{\partial \Phi}{\partial y}\ dy \qquad (4\text{–}11)$$

When applied the differential ***de*** for the above equation, then becomes

$$\frac{d\Phi}{de} = \frac{\partial\Phi}{\partial x}\frac{dx}{de} + \frac{\partial\Phi}{\partial y}\frac{dy}{de} \qquad (4\text{–}12)$$

From Figure 4.4, $de = dx\boldsymbol{i} + dy\boldsymbol{j}$, and

1) when $de<0$, (ie. the state point is to the left side of the path), then

$$\frac{dx}{de} = \sin\theta \quad , \quad \frac{dy}{de} = -\cos\theta \quad . \qquad (4\text{–}13)$$

2) when $de>0$, (ie. the state point is to the right side of the path), then

$$\frac{dx}{de} = -\sin\theta \quad , \quad \frac{dy}{de} = \cos\theta \quad . \qquad (4\text{–}14)$$

From the above equations, we can see that if the control actions for the axes ($rm_{1m}=k_1\partial\Phi/\partial x$, $rm_{2m}=k_2\partial\Phi/\partial y$) have been defined, then the control action directly to the synchronisation error ***e*** ($k_3\partial\Phi/\partial e$) can be calculated based on them using equation (4–12), where k_i (i=1,2,3) are proportional gains. By integrating of $\partial\Phi/\partial e$, an approximate surface Φ against ***e*** can be generated using the data in Figure 4.15. For the condition $\delta e = 0$, Figure 4.17 shows the approximate curve derived with respect to the synchronisation error, where $k_1=1$, $k_2=1$, and $k_3=1$. The slopes of the curve are anti-proportional to the synchronisation error, which means that the resulting control action tries to reduce the synchronisation error.

If we use a real scalar function $\Phi=\frac{1}{2}(ke^2)$ with a proportional gain k to compare with the fuzzy approximated curve, in Figure 4.18 (in this case k=3.6445) we can see that the two curves are very similar, which illustrates that linguistic rules can be used to achieve the same control action as that achieved with a scalar field.

In the ball analogy of scalar field control[16], positive damping was necessary to bring the ball to rest on the path. Consequently a derivative term is often introduced into a conventional controller; frequently resulting in a controller of the form $(k_p + k_D s)$, where

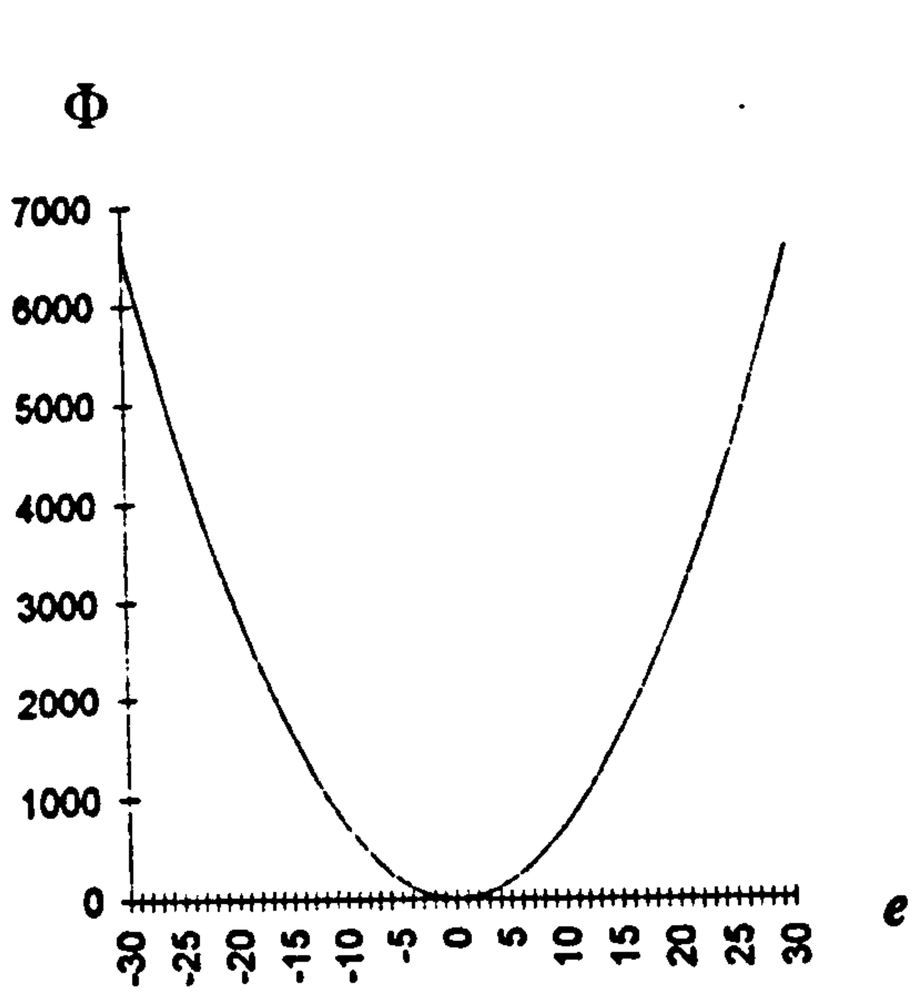


Figure 4.17 An approximate curve for function Φ when $\delta e = 0$

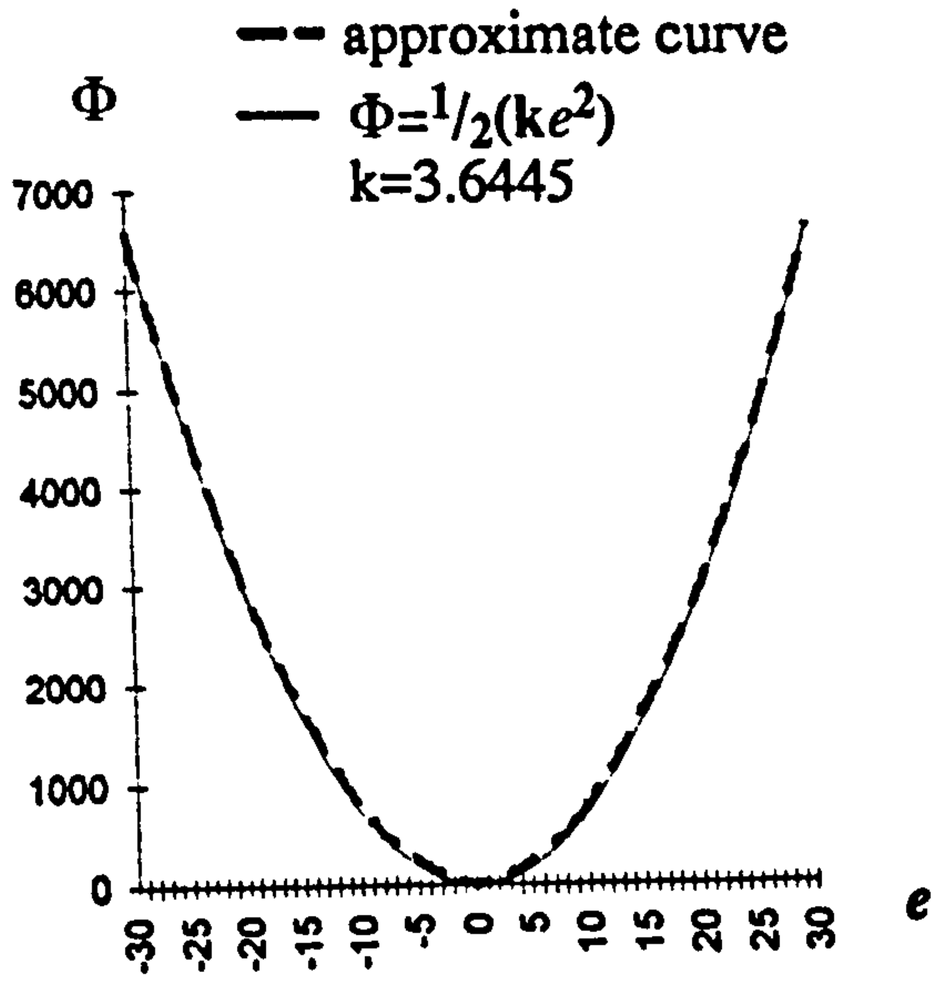


Figure 4.18 Comparison between the fuzzy approximated curve and the scalar function $\Phi=\frac{1}{2}(ke^2)$

k_p represents a proportional gain, and k_D denotes a derivative gain. This conventional controller therefore acts upon the absolute error signal e_a and its time derivative de_a/dt. In the scalar field control, the error signal is represented by $-\partial\phi/\partial x$, and its time derivative is $d(-\partial\phi/\partial x)/dt$. The proposed fuzzy logic coupling algorithm naturally includes the error signal e and its time derivative δe. If the control rules and fuzzy reasoning are treated as a nonlinear mapping from e and δe to the control action rm_{im}, then rm_{im} can be represented as

$$rm_{im} = f(e, \delta e). \qquad (4\text{–}15)$$

The above equation implies that rm_{im} is the output of a nonlinear P.D. controller[4]. Therefore, the developed fuzzy logic coupling algorithm is a cross-coupling controller with a nonlinear P.D. control law.

Here, we may say that scalar field control can mimic the '***potential energy***' effect, but the fuzzy logic coupling also includes the '***kinetic energy***' effect (assuming $\delta e \neq 0$), as shown in Figure 4.15. The δe term which represents the higher order synchronisation errors of the system (ie 'velocity synchronisation error') is useful where the synchronisation error is changing rapidly, for example, when disturbances occur on one

axis, or the required motion involves a step change in position. From the '***work–energy theorem***', both kinds of energy have the same effect[19]. Human operators can naturally consider the '***kinetic energy***' effects in control actions. The fuzzy logic control which is based on human knowledge representation normally requires both error and error change to evaluate the control inputs, which may result in fuzzy logic coupling having a more effective and adaptive capability[4].

4.4.2 Stability Issues of Multi-axis Systems with the Fuzzy Logic Coupling Mechanism

The multi-axis motion control system we consider is assumed to be a steady-state system, because each motion is furnished with a stabilizing controller, which causes the whole system to be in steady-state[20]. In order to achieve tight motion synchronisation with the independent servo-drives, an interconnected or coupled control structure and on-line compensation are necessary. See section 4.2.2.

However, with the proposed fuzzy logic coupling mechanism, the compensation term is generated by a fuzzy algorithm, therefore, the fuzzy system has to be stable as well. Stability analysis of fuzzy logic control systems has been difficult because fuzzy systems are essentially nonlinear. Recently, some useful stability techniques [23]–[27], which are based on nonlinear stability theory, have been reported. Therefore, stability analysis of fuzzy control systems became easy. But, generally speaking the analysis of fuzzy control systems by means of nonlinear control theory can never be general, simply because a general theory of nonlinear control systems does not exist. It is quite difficult to develop a method for analysing the stability of fuzzy logic coupling mechanism, because the whole system includes not only a fuzzy logic algorithm, but also has a number of interconnected servo-drives. This will be one aspect of the future work. Currently, the fuzzy logic coupling system is analysed by simulation tests and tuned manually.

4.5 Conclusions

A new coupling strategy for multi-axis motion synchronisation, which embeds human intelligence, has been proposed. Advantages of the new scheme are:

1) The characteristics of individual servos can be handled separately by input/output variable membership definitions and rule base derivation, which enables the multi-axis system designer to use the state of art control techniques for each axis, then integrate the axes as a whole multi-axis system in an intelligent way.

2) The uncertainness of the controlled process can be described linguistically and implemented in fuzzy logic. When some information of the processes can be described in linguistic values, these information can be incorporated into the proposed fuzzy logic synchronisation control algorithm to enhance the synchronisation performance.

3) No mathematical modelling is necessary. Therefore, all types of motion synchronisation can be implemented easily and use the same framework.

The calculation of the proposed algorithm may slow down the possible sampling-rate of the software mechanism. However, with present advances in computer power, especially digital fuzzy processors have been developed[21], we do not see it is a drawback of the proposed mechanism.

Chapter Four: References

[1] Luggen, W. W., "Fundamentals of Computer Numerical Control", Third Edition, Delmar Publishers Inc., 1994.

[2] Hu, J., Chiu, T. and Tomizuka, M., "On Motion Synchronization of Two Axes Systems", Monitoring and Control for Manufacturing Processes, ASME Conf, 1990, pp267–282.

[3] Tomizuka, M., "Design of Digital Tracking Controllers for Manufacturing Applications", Manufacturing Review 1989 Vol. 2, Part 2, pp134–141

[4] Li, Y. F. and Lau, C. C., "Development of Fuzzy Algorithms for Servo Systems", IEEE Control Systems Magazine, April 1989, pp65–71.

[5] Lo, C. C. "Cross-Coupling Control of Multi-axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

[6] Hu, J., Chiu, T. and Tomizuka, M., "On Motion Synchronization of Two Axes Systems", Monitoring and Control for Manufacturing Processes, ASME Conf, 1990, pp267–282.

[7] Hunt, K. H., "Mechanics and Motion", English Universities Press, 1960.

[8] Draper, C. M., Holding, D. J., "The specification and fast prototyping of a distributed real-time computer control system for a modular independently driven high-speed machine", 2nd International Conference on Software Engineering for Real Time Systems, September 1989, pp 199–203.

[9] Richard Quintero and A. J. Barbera, "A Real-Time Control System Methodology for Developing Intelligent control Systems", NISTIR 4936, October 1992.

[10] Chen, C., "Control and Synchronisation of Multi-axis Drive Systems (Synchronisation part)", Internal Report, Department of Manufacturing Engineering, Loughborough University of Technology, August, 1993.

[11] "Fuzzy Guide Book", Cat. No. P30-E1-2, 1992.

[12] Chuen Chien Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I and Part II", IEEE Transactions on systems, Man, and Cybernetics, Vol 20 No 2, March/April, 1990, pp 404–435.

[13] Bart Kosko, "Neural Networks and Fuzzy Systems", Prentice-Hall Inc, 1992.

[14] Zhang, S. J., Wong, Y. S. and Poo, A. N., "Analysis and Design of Inference Mechanisms for Fuzzy Fedback Control", IEEE, 1993, pp9–17.

[15] Bouslama, F. and Ichikawa, A., "Fuzzy control rules and their natural control laws", Fuzzy Sets and Systems 48 (1992), pp65–86.

[16] Jenkinson M., "The synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

[17] Burden, R. L., "Numerical Analysis", 4th edition, PWS-Kent Publishing, 1989.

[18] Jeffrey, A., "Mathematics for Engineers and Scientists", 4th edition, Van Nostrand Reinhold, 1989.

[19] Meriam J. L., "Engineering Mechanics Statics and Dynamics", SI Version, John Wiley & Sons, 1980.

[20] Wladyslaw Findeisen, Mieczyslaw Brdys, Krzysztof Malinowski, Piotr Tatjewski, and Adam Wozniak, "On-Line Hierarchical Control for Steady-State Systems", IEEE Transactions on Automatic Control, Vol. AC-23, No.2, April 1978, pp189-209.

[21] "Clearly Fuzzy", OMRON, 1991.

[22] Wang Li-Xin, "Stable Adaptive Fuzzy Control of Nonlinear Systems", IEEE Transaction on Fuzzy Systems, Vol.1 No.2, May 1993, pp146-155.

[23] Langari, R. and Tomizuka, M., "Analysis and synthesis of fuzzy linguistic control systems", in 1990 ASME Winter Annual Meet., 1990, p35-42.

[24] Kitamura, S. and Kurozumi, T., "Extended circle criterion, and stability analysis of fuzzy control systems", in Proc. Int. Fuzzy Engin. Symp. 1991, Vol. 2, 1991, pp634-643.

[25] Tanaka, K. and Sugeno, M., "Stability analysis and design of fuzzy control systems", Fuzzy Sets and Syste., Vol. 45, No. 2, 1992, pp135-136.

[26] Hara, F. and Ishibe, M., "Simulation study on the existence of limit cycle oscillation in a fuzzy control system", in Proc. Korea-Japan Joint Conf. Fuzzy Systems and Engin., 1992, pp25-28.

[27] Tanaka, K. and Sano, M., "A Robust Stabilization Problem of Fuzzy Control Systems and Its Application to Backing up Control of a Truck-Trailer", IEEE Transactions on Fuzzy Ssytems, Vol. 2, No. 2, May 1994, pp119-134.

CHAPTER FIVE
Intelligent Motion Control (IMC) Structure Design

5.1 Introduction

IMC (Intelligent Motion Control) is a method to facilitate the design and control of multi-axis motion control systems in an intelligent manner. Intelligent motion control is defined as utilizing feedback from the physical environment to manifest "intelligent behaviour" in real-time via computerised real-time coordinated and synchronised motion control of the machine's electro-mechanical actuators and sensors. Such multi-axis systems are termed ***dual-closed loop motion control systems*** and are distinguished from ***open-loop motion control systems*** in that open-loop systems do not have the capacity to alter their behaviour in real-time based on sensory feedback from the environment. The requirements and developments of the method are described in this chapter.

5.1.1 Intelligent Software Mechanisms for Manufacturing Machines

For a typical manufacturing machine the high-speed concurrent manipulation and synchronisation of workpieces, tools and sensors may be involved. Thus, a number of motion and sensing elements (which will be referred to as machine components) will be required to operate in a coordinated manner, the number and type of these components being chosen to accomplish the specific producing function. The individual components will each require their own subset of information processing and control functions. The individual components will be required to interact with other components forming one or more 'close coupled' component groups, performing logically separate parts of the overall producing function. The term 'closely coupled' is used here to imply a 'close relationship between'; an example of such a relationship occurs when more than one machine component is involved in locating a workpiece or tool according to some specified position, velocity or force profile. The individual components of a 'closely coupled' component group may, in fact, be operating at different physical locations in a specific machine. Indeed, the individual components may also be distributed at remote

locations along a production line but be required to operate in a synchronised manner (a common requirement in the packaging and process industries)[1]. Thus, the individual components of a 'closely coupled' component group can be considered to be logically related, but may or may not be physically linked to each other.

The components of a 'closely coupled' group are inter-linked by a software mechanism. The motion of all of the axes defined in a software mechanism can then be coordinated. Therefore, the intelligent motion control of the multiple axes depends on the intelligence of this software mechanism. The fuzzy logic coupling mechanism developed in the previous chapter will be used as one of the essential building blocks in the IMC structure.

5.2 Hierarchical Design of Multi-Axis Machine Systems

Industrial automation spans a huge spectrum of complexity in terms of both the physical structure of machines and the tasks which they perform. A design method for machine systems, termed Intelligent Motion Control (IMC), has been developed based on the hierarchical modelling of the machine systems.

5.2.1 Task Oriented Decomposition

Every method seeks to devise a model of the problem domain which explicitly represents and emphasises the most critical components of the problem while simplifying those aspects which have a lesser impact on the solution being sought. Every model is, after all, a simplification of the real world. The aim is to choose an appropriate abstraction (or set of abstractions) that highlights the parts of the problem that make a difference in the understandability, quality and efficiency of the solution. That is why it is so important to choose a method which matches the problem domain[2]. In the domain of manufacturing machine control systems the author believes ***tasks***[1] are the driving factor.

Fundamental to the control of electro-mechanical devices is an understanding of the task it is to perform. For tasks that involve complicated sequences of steps and the complex

---

1 · *Tasks* are the activities which a machine performs together with the sequencing and synchronisation of these activities.

coordination of many operations, the exact nature of that task becomes critical to solving the control problem. The task determines the use and synchronisation of sensors and actuators, the choice of computing hardware, and nature of the world model. Effective design of control systems for complex decision oriented problems is well accomplished through careful task analysis and decomposition.

5.2.2 Computer Control of Machines -- Levels of Abstraction

The control structures of both conventional and independently-driven manufacturing machines are based on a hierarchy of operations [3]. The machine function is defined as the sequence of processing tasks necessary to transform raw materials into a product. The task level synchronises and coordinates the various axes required to accomplish a particular task. The axis level comprises a set of controlled motion profiles. These profiles are transformed into actuator profiles and used to manipulate the product as required. The levels of abstraction are discussed here from the perspective of physical movement tasks (rather than logical operations such as switch closures), as follows:

Level 1. Servo -- The Servo Level functions as the interface to the machine's actuators and sensors, It can be thought of as the device driver level. Task commands from level 2 are converted into voltages and in turn currents to drive electro-mechanical devices interfaced at this level. The Servo Level generally operates using high bandwidth synchronous closed-loop control. Each actuator may be paired with one or more sensors to effect closed-loop feedback control. The Servo Level periodically samples sensors and sends out drive signals to effect stable control and smooth movements.

Level 2. Sync (synchronisation) -- The Synchronisation or Sync Level accepts commands from Level 3 and decomposes them into regularly spaced "move" commands to the Servo Level. Sync deals with machine dynamics and performs functions such as the fuzzy logic coupling algorithm. Sync generally produces set-point output commands to the Servo Level in a synchronous fashion (evenly spaced in time). Sensory data, such as inputs from proximity, force, and torque sensors, are used to produce or modify reference commands in real-time.

Level 3. Task (coordination) -- The Task Level accepts task commands from Level 4 and converts them into tasks for the machine's subsystems (an axis or a group of axes).

This level is responsible for coordinating the actions of the subsystems within a single machine. This Level deals with machine kinematics and is typically responsible for generating reference commands to be further decomposed by the Sync Level. The control system components are typically hosted on a single multiprocessor backplane at this level and below. The backplane host is typically connected to a local area network (LAN) for communication with higher levels.

Level 4. Machine -- This Level coordinates the actions of a small group of machines and people. Generally the machine controllers communicate over some type of local area network. Level 5 controllers are typically hosted on computer workstations in factory applications.

The above levels of abstraction have their derivations from the RCS architecture [2]. They are, however, defined here in a manner more suitable for high speed, algorithmically complex servo controlled manufacturing machinery.

5.3 The IMC Design Method

An IMC motion control system consists of a set of runtime control software modules conforming to the above modelling. The philosophy behind the IMC design method provides the means for controlling the machine motions through the intelligent software mechanisms among the servo-drives. The IMC design method is proposed to improve the multi-axis motion control capability of the UMC[2] methodology which was devised by the Manufacturing Systems Integration (MSI) Research Institute in the Department of Manufacturing Engineering, Loughborough University of Technology[5][6].

5.3.1 The UMC Methodology

UMC (Universal Machine Control) is a methodology to facilitate the design and implementation of machine control systems in a generalised manner. A UMC control system consists of a set of runtime software modules conforming to a reference architecture and off-line tools to aid with the configuration and management of the

---

2. *UMC* is an abbreviation of *Universal Machine Control*. This is the name by which the machine control methodology is known. An terminology of the UMC methodology is described in Appendix V.

runtime system. The philosophy behind the UMC reference architecture provides the means for integrating proprietary control products (such as motion controllers, I/O controllers and sensors) and presenting a unified view of these products to the runtime software. In addition to the off-line tools specific to UMC, proprietary design products and tools can be employed by the system builder and the applications programmer. These products can come from a wide range of sources and include off-line modelling, evaluation and programming tools.

The UMC methodology provides a consistent approach for system configuration and for the integration of additional functionality as new requirements evolve. The methodology allows the control system to develop over a period of time and to adapt to evolving requirements and technologies.

Features of the UMC methodology:

- **Consistent device interface.** The key difference between UMC and typical proprietary control systems is that a UMC control system can incorporate components from a wide range of different manufacturers and provides a consistent interface to these components.

- **Ease of control system modification or upgrade.** A UMC control system can readily accommodate either physical or logical changes to the machine, making future modifications or upgrades relatively straightforward. This is an enormously important advantage and is brought home by considering the large number of changes which a typical machine undergoes from initial development to obsolescence.

- **Data visibility.** The architecture has been designed so that all system data is readily accessible to software running outside the control system. A vast range of machine information is therefore available if required.

5.3.2 UMC Machine Overview

A UMC machine consists of a number of concurrently executing processes together with mechanisms for communication, co-ordination and synchronisation of the processes. This approach encourages the user to divide the overall machine requirements at the application level into easily programmable application tasks, which relate to a functional

decomposition of the whole machine operation. The application tasks together have a heterarchical relationship although functionally one task can be a master task. Events are used to synchronise the tasks, control the use of shared resources and for passing data between the tasks.

Any of the application tasks can control any subset of the handlers in a particular UMC machine. The subsets of handlers used by the tasks can overlap, allowing sharing of handled external devices between tasks.

This decomposition of dependencies is specified as a set of UMC configuration data which is stored within a database in the UMC off-line environment. The UMC configuration data contains a high level description of the physical and logical machine relating to:

- device configuration information
- tasks
- handlers
- events

The UMC configuration data includes descriptions of individual "handled" external devices. This includes the data needed to describe the external device in terms of the task communication capabilities, and the data required to initialise and run the handler.

The UMC methodology allows for the programming of control systems made up from essentially any external devices for which suitable handlers can be provided. Handlers thus provide a standardized interface between tasks and the external devices which they control.

Application tasks, written in a high level language, utilize language interfaces for inter-process communication provided as part of the implementation. These interfaces provide the means for the establishment of communications to other tasks and handlers, issuing commands to handlers, provision for inter-task co-ordination, and access to all information modules. The interface for a particular programming language is provided by means of a library of functions or sub-routines specific to that language.

The position co-ordinates of UMC axes may be referenced by means of named locations. Each task can use an optional location information module to contain the axis

co-ordinates for named locations. The locations may define position co-ordinates for some or all of the axes controllable by a given task, as specified in the machine information module.

5.3.3 UMC Reference Architecture

5.3.3.1 Introduction

The UMC reference architecture provides a common framework for UMC machines. The reference architecture is simple in both concept and structure. See Figure 5.1.

5.3.3.2 Reference Architecture Levels and Building Blocks

The UMC reference architecture explicitly defines three hierarchical levels, namely, machine level, task level and handler level.

- **Machine Level**

  This is the top level in the UMC reference architecture. It consists of a machine information module together with utilities to configure and manage a UMC machine. The machine information module describes a particular UMC machine at runtime.

- **Task Level**

  The task level consists of application tasks and utility tasks, together with any associated task or location information modules. Application tasks are the application specific components in the UMC reference architecture, and are user defined. Complete applications are divided into concurrent application tasks with defined synchronisation and data access mechanisms. Since the design of a particular UMC machine is seen as very application dependent, the application task structure is deliberately not explicitly defined. The task level therefore supports the concept of multiple tasks which can be arranged both hierarchically and heterarchically in a user defined manner, as determined by the requirements of individual applications.

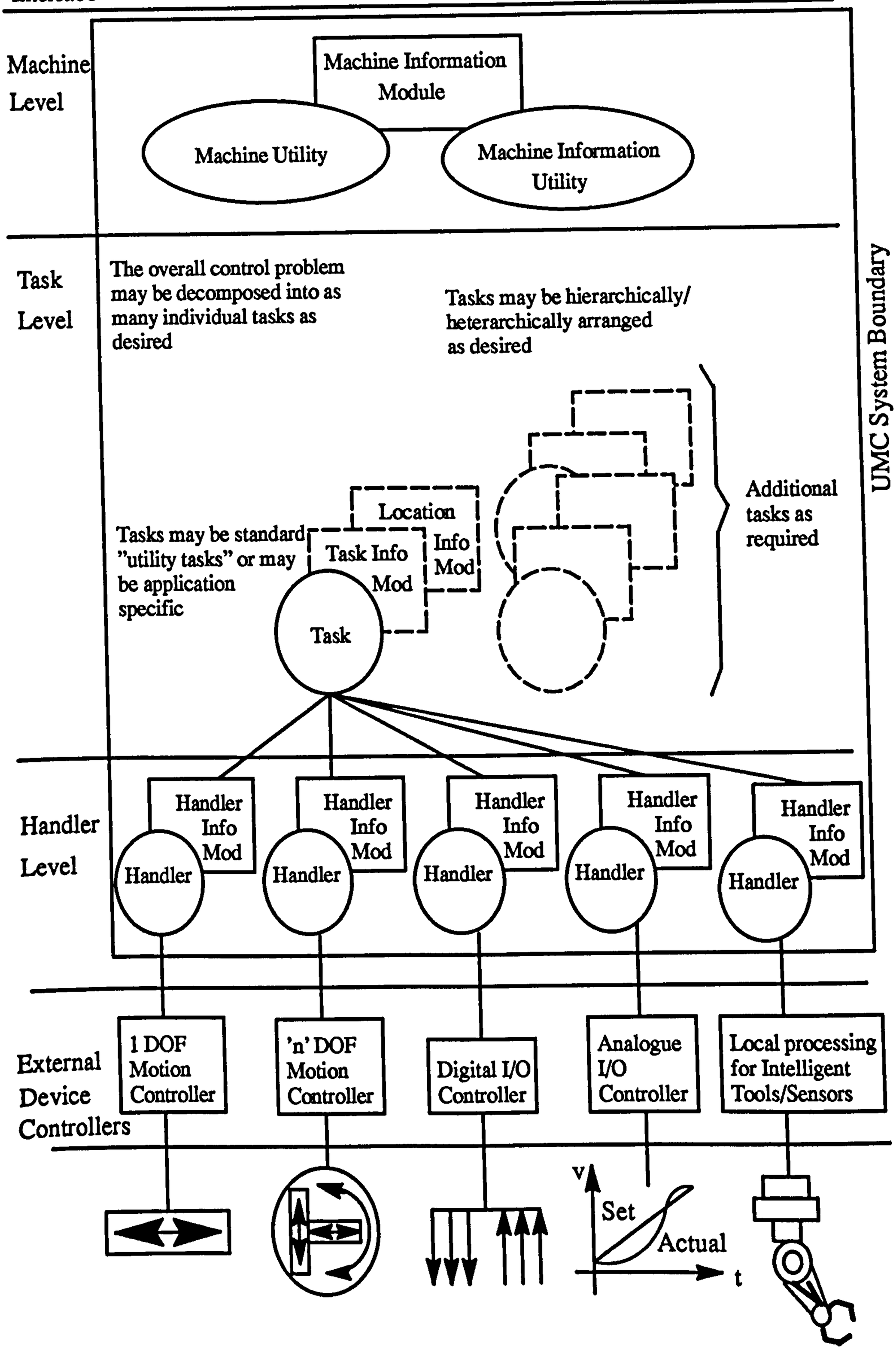


Figure 5.1 The UMC Run-time Architecture. Source: Harrison.

- **Handler Level**

  The handler level is the lowest level and consists of handlers and associated handler information modules. This level provides isolation between the internal unified UMC system representation and external devices. A defined type of handler is required to cater for each generic class of external device.

5.3.3.3 Reference Architecture Communication

- **Machine to Above**

  No current specification.

- **Tasks to Global Data View**

  Consistent data access control mechanisms to any global information modules within the runtime system.

- **Task to Task**

  Defined data transfer and synchronisation mechanisms. The implementation platform affects specific use.

- **Task to Handler**

  Provides for the communication of UMC commands. This allows tasks to control virtual devices by means of a generic command set. Note that particular external devices may not be able to support all the capabilities of a generic device type.

- **Handler to Device**

  External device specific communication.

5.3.4 Real Time Performance of the Current UMC

The current UMC axis handler provides motion control with the option of defined velocity profiles when these are supported by the motion control devices. The use of intelligent I/O devices supervised via the UMC handlers is the intended philosophy for

time critical hard real-time processes beyond the performance capabilities of the UMC environment. For example, in the case of high speed synchronised drives, a dedicated motion controller would typically be utilised for each axis of motion. Pre-calculated position demand profiles may be stored in the motion controller local memory and drive synchronisation is achieved heterarchically via a common clock [7]. In this supervisory role the UMC system performs device set-up, monitoring and adjustments via appropriate handler functions.

The UMC system does not currently support tight synchronisation of multi-axis movement at the task level, since the handler does not have the capacity to alter its behaviour in real-time based on sensory feedback from the environment. From this viewpoint, within the current UMC system boundary multi-axis motion control systems are restricted to an implementation that is of the "open-loop" type. It should be emphasised that nearly all current commercial multi-axis motion control systems are of this "open-loop" type. Therefore, although interpolation, contouring and programmable transmission capabilities are being progressively incorporated into the system since the addition of such capabilities is intrinsic in the UMC approach, the tight motion synchronisation for multi-axis systems is still attained by external devices outside the UMC system boundary. Normally, performance limitations within the UMC system boundary need not however be a limiting factor on overall control system performance. As advances occur in state of the art external device performance these can be rapidly and consistently interfaced to the UMC system via the implementation of appropriate handlers. Nevertheless, for the multi-axis system this implementation may result a complex handler which might be inefficient in real-time control when fast communication between task level to device level is required.

Inter-device synchronisation under software control is a prerequisite for many applications as described in previous chapters (ie, between closely-coupled axes, axes and I/O, and selected I/O conditions). The interrupt based communication mechanisms at the device level can be very fast and efficient, however, synchronising hardware at the element or handler level is generally inappropriate because standard communication mechanisms and the logic functions (if they existed) would not provide the required application dependant functionality. This application logic has to be expressed in the

user task processes, but the existing task to device communication mechanisms are in no way fast enough for efficient 'real-time' synchronisation[8].

The current UMC hierarchy introduces time delays because all task to device communication takes place through handler process. Advantages of using handlers are:

- Supports device data visibility
- Application tasks are hardware independent
- Devices are shareable between user task processes
- Task independence from hardware error recovery

Disadvantages of the handler approach:

- Large delays are incurred when device/hardware has to be synchronised at the task level.
- Devices are not shareable between different UMC machines.

In the following section, the IMC design method is introduced to replace the current handler approach in order to provide tight motion synchronisation within UMC system boundary.

5.3.5 IMC Structure

The IMC structure is based on the UMC reference architecture, since IMC modelling of the multi-axis machine system is much similar to UMC modelling. The Machine Level and Task Level remain unchanged. The Handler Level of UMC is replaced by Sync Level and Servo Level in order to introduce couplings between the servo axes to provide tight motion synchronisation. Figure 5.2 shows the IMC run-time structure.

The dual closed-loop control structure described in Chapter 4 is used to construct the Sync Level and Servo Level. In this way the fuzzy logic coupling mechanism is embedded in Sync Level. Therefore the IMC has a closed-loop control structure between tasks and the external devices, so that IMC can provide tight motion synchronisation control without having to rely on some dedicated control software or

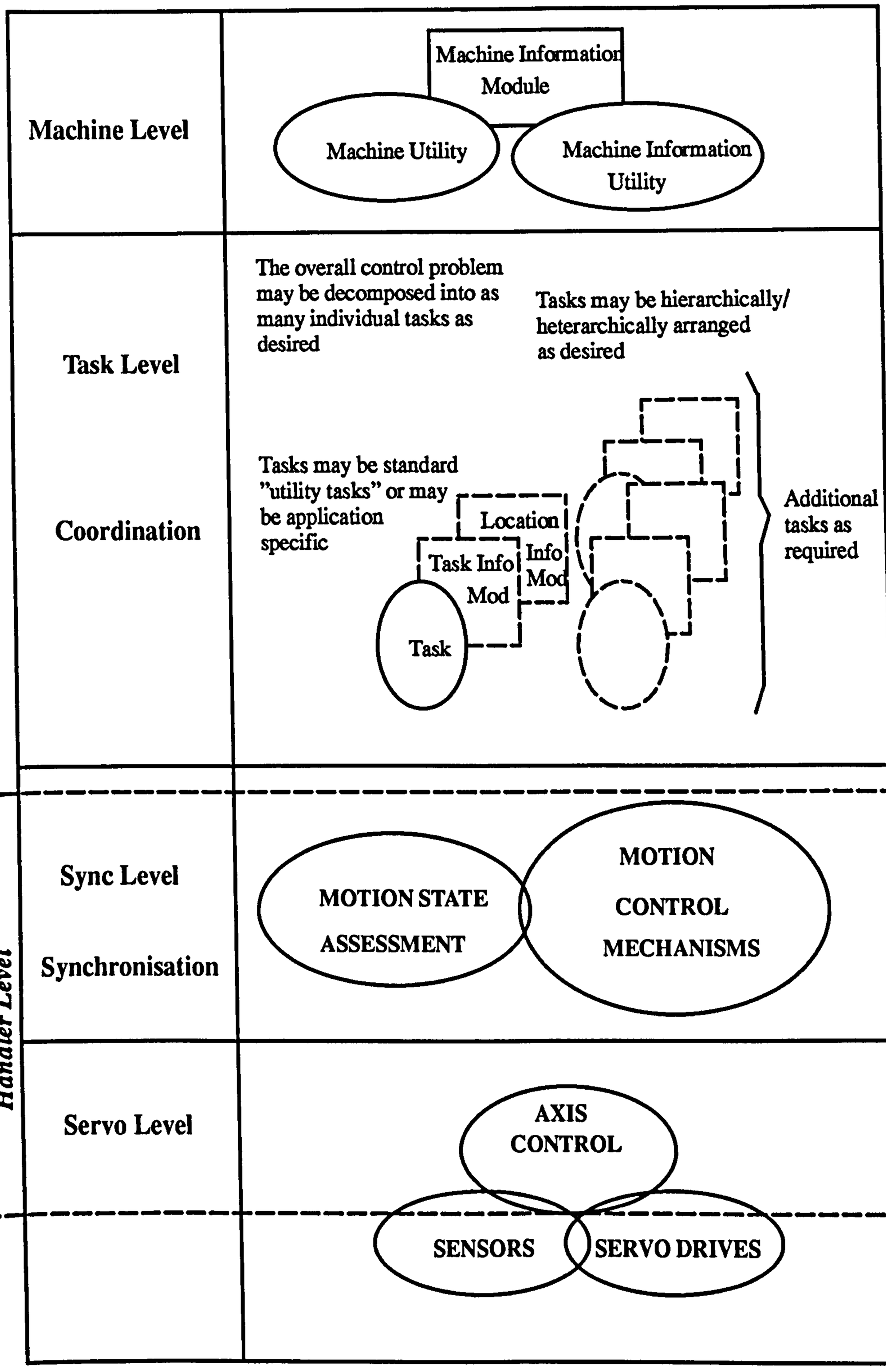


Figure 5.2 The IMC Run-Time Structure

controllers (such as, the PTS[7]). Therefore, the functions of the current UMC system will be expanded, and the real-time performance will be improved[9].

5.3.5.1 The Task-Device Levels Communication in the IMC Structure

The current UMC machine is implemented on 680x0 family processors running the OS-9 operating system. OS-9 can respond to external events via interrupts which generate a hardware signal to the processor. It causes the processor to suspend execution of the current program, and execute a separate function (called an "interrupt handler"). The interrupt handler carries out any tasks that require "immediate" attention, and/or sends a software signal to the program that wants to know about the external event. Interrupt based communication mechanisms are very fast and efficient[10].

Figure 5.3 shows the current UMC task level hardware synchronisation mechanism[8]. Through this mechanism, a hardware input (from either axis or I/O hardware) is read by the task, the necessary action is decided upon and an appropriate output is sent to the other hardware. Apart from the initial hardware interrupt to the first driver, which takes place in OS-9 system state, all the other actions are carried out in OS-9 user state. Each process waits in the active process queue until it is given a time slice (OS-9 is a time slice based operation system), it then carries out all its processing until its time slice is complete or it voluntarily gives it up. Substantial (and indeterminate) delays can therefore be accrued.

In the IMC structure, the UMC handler is replaced by Sync level and Servo level. Since the modules in Sync level tightly link with the modules of Task level and the modules in Servo level include the device, the "distance" between tasks and devices becomes short. Figure 5.4 outlines the communication mechanism in the IMC structure. The task can overpass Sync level and Servo level to reach the device.

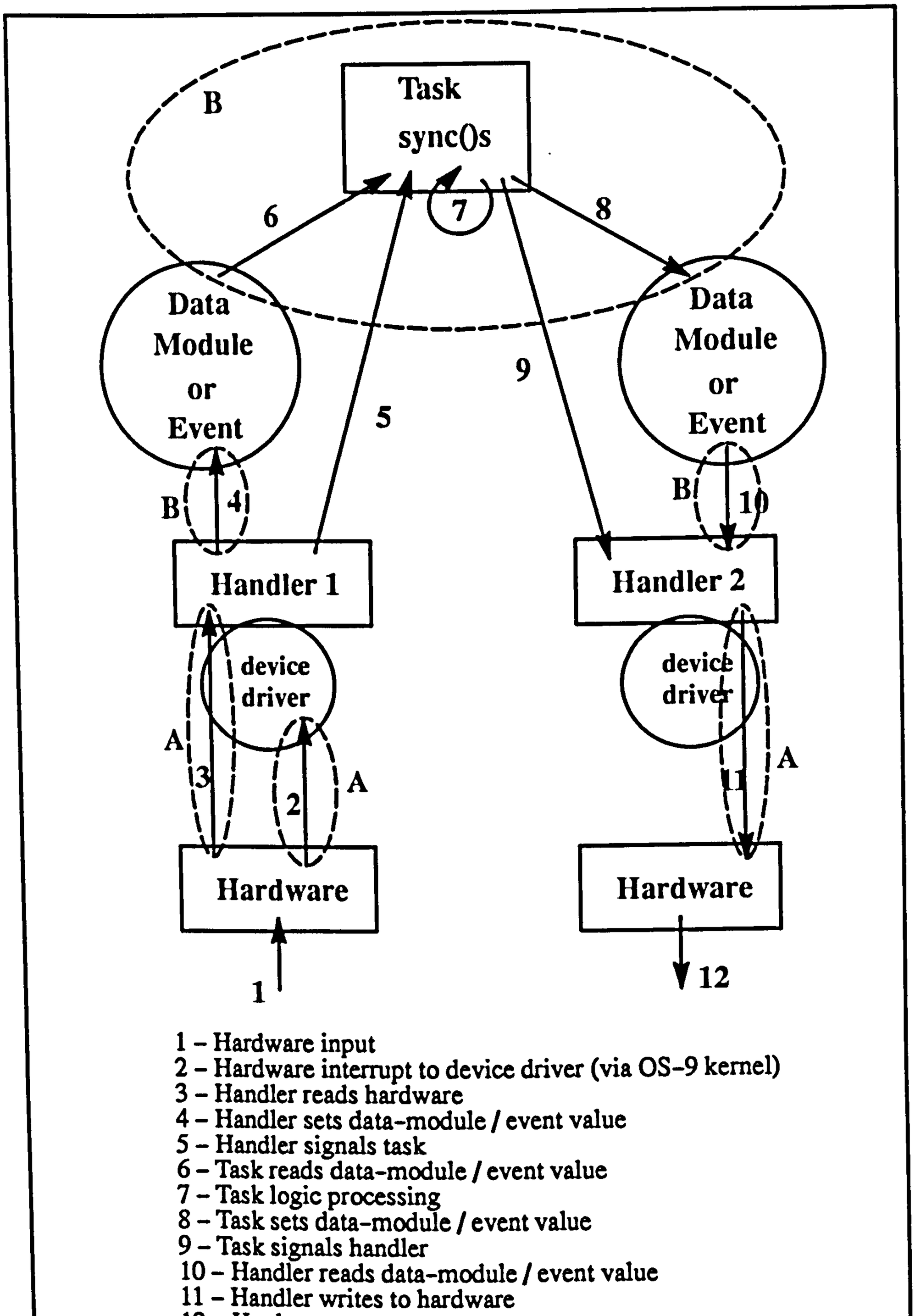
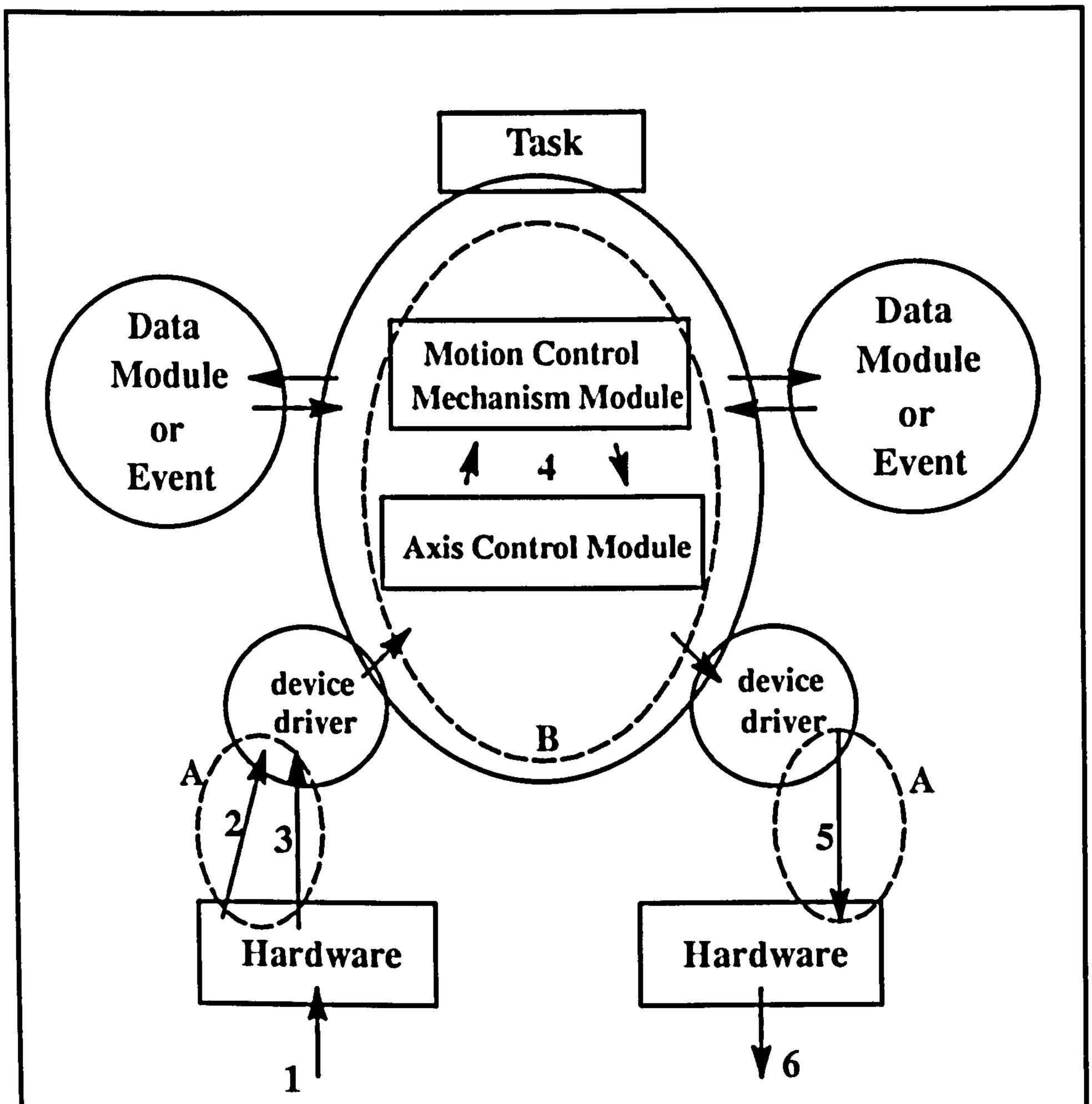


Figure 5.3 The Current UMC Task Level Hardware Synchronisation. Source: Armstrong.



1 – Hardware input
2 – Hardware interrupt to device driver (via OS-9 kernel)
3 – Task reads hardware
4 – Task logic processing
5 – Task writes to hardware
6 – Hardware output

A – Fast – OS-9 system state
B – Slow – OS-9 user state

Figure 5.4 The Task-Device Communication in the IMC Structure

5.4 Summary

The IMC method aims to facilitate the design and control of multi-axis motion control systems in an intelligent manner.

The IMC structure provides a common framework to design intelligent motion control systems for applications which require tight motion synchronisation.

Since IMC is based on the UMC methodology, all facilities of UMC can be used to build IMC systems.

The next chapter looks at the selection of suitable enabling technology and describes the implementation of the IMC method using selected hardware and software.

Chapter Five: References

[1] Weston, R. H., "Open, Integrated and Model-Driven Product Realisation", Advanced School on 'Mechatronics' Udine, Italy, March, 1992.

[2] Quintero, R., and Barbera, A. J., "A Real-Time Control System Methodology for Developing Intelligent control Systems", NISTIR 4936, October 1992.

[3] Rathkey, J., "How to synchronize servomotors", Machine Design February 1989, pp 109-111.

[4] Draper, C. M., Holding, D. J., "The specification and fast prototyping of a distributed real-time computer control system for a modular independently driven high-speed machine", 2nd International Conference on Software Engineering for Real Time Systems, September 1989, pp 199-203.

[5] Harrison, R., "A Generalised Approach to Machine Control", PhD Thesis, Loughborough University of Technology, July 1991.

[6] Harrison, R., et al., "UMC Conceptual Specification", 4th Working Draft, June 2 1992.

[7] Anon, "Programmable Transmission System Reference Manual", Quin Systems Ltd, Wokingham, 1990.

[8] Armstrong, N. A., "A Generic UMC Task-Level Hardware Synchronisation Mechanism", Internal Report, MSI Research Institute, Department of Manufacturing Engineering, Loughborough University of Technology, March, 1992.

[9] Moore, P. R., and Chen, C., "Fuzzy Logic Coupling and Synchronised Control of Multiple Independent Drives", Control Engineering Practice, to be published.

[10] Dayan, P. S., "The OS-9 Guru", Galactic Industrial Limited, 1992.

CHAPTER SIX
Implementation of Intelligent Motion Control (IMC) Elements

6.1 Introduction

The previous chapter described the essential features of an intelligent approach to motion control IMC which is derived from the UMC architecture conceived at Loughborough University. From UMC to IMC, the major change is to the handler mechanism. IMC can therefore be considered as a 'special case' UMC, which provides tight motion synchronisation at task level. The realisation of IMC only necessitates the implementation of the elements in Sync level and Servo level. Elements in Machine Level and Task Level are unaltered. This chapter provides an in-depth discussion of implementation methods for the Sync level and Servo level of IMC and documents the design of the real-time execution modules.

6.1.1 Hardware Selected Ensuring Adequate Real-Time Performance

IMC can be considered as a 'special case' of UMC, as such, it is very convenient to implement IMC in the same environment as that of UMC. However, since the motion state information is needed to feedback to the Sync level in IMC, fast communication between Sync level and Servo level is required. Generally speaking, in order to tightly control the motions the sample rate for the Sync level should be as fast as in the Servo level[1]. But for some applications, such as web synchronisation, the Sync level closes a loop (for example, the tension loop) which may be slower in nature than the position loop at the Servo level.

Computation speed and the real-time response of the local operating system are two issues that will influence the co-ordination and synchronisation performance and as such require consideration when selecting hardware for an IMC system. The current UMC machine is implemented on 680x0 family processors running the OS-9 operating system. The 680x0 family processors with a 68881 floating-point coprocessor have powerful calculation capability, and OS-9 can respond to external events via interrupts

which generate a hardware signal to the processor. Interrupt based communication mechanisms are very fast and efficient[2].

6.2 Software Development System

The OS-9 operating system, on a Syntel VM022 processing platform, is used to implement the IMC elements. At the heart of the VM022 is a 68020 processor running OS-9, together with a 68881 floating-point coprocessor. The operating system OS-9 was originally conceived around 1979 by Microware Systems Corporation, and has become one of industry's standard operating system environments for real-time applications. It presents a multi-user, multi-tasking kernel with comprehensive input/output (I/O) capabilities, and is robust enough to support real-time processes. The version OS-9/680x0 can be used across the range of the 680x0 microprocessor family. To fully exploit its potential, the 'C' programming language has been chosen by the originators for the operating system.

Interfacing to the outside world is achieved by using position independent modules of code called 'device drivers'. Upon receiving an interrupt signal, the drivers access the I/O memory locations of the interfacing hardware and buffer the values into system memory. The system memory is then accessible to an awaiting C program. Since OS-9 is designed to be interrupt driven, accurately timed acquisitions can be achieved with a robustness which is not always available with some other operating systems, such as PC-DOS and UNIX [2]. The drivers are also re-entrant, enabling the same driver to be shared by many processes, each with its own device.

6.3 Motion Controller

The Quin Systems multi-axis digital motor control modules -- DSC-1M -- are used as motion controllers[3]. They are used for multi-axis operation in a G64 bus based OS9 computer system. The DSC-1 controller comprises hardware and software to control one servo system. They may be used as part of a multi-axis servo system in conjunction with other modules. They receive commands via the systems bus from the host processor.

6.4 Interfacing

6.4.1 Interfacing Hardware

For machine control with multi-axis something more is needed – a machine controller. It is a unit that has enough processing capacity to coordinate operations of individual servo-drives. As operational elements of a machine the independent servo-drives should operate according to the machine controller's commands. This leads to a structure where the physically distributed independent servo-drives are connected to the machine controller.

Figure 6.1 describes the interconnection which uses a parallel interface[13]. The chosen motion controllers (DSC-1M) are interfaced to the OS-9 computer system through G-64 bus[3].

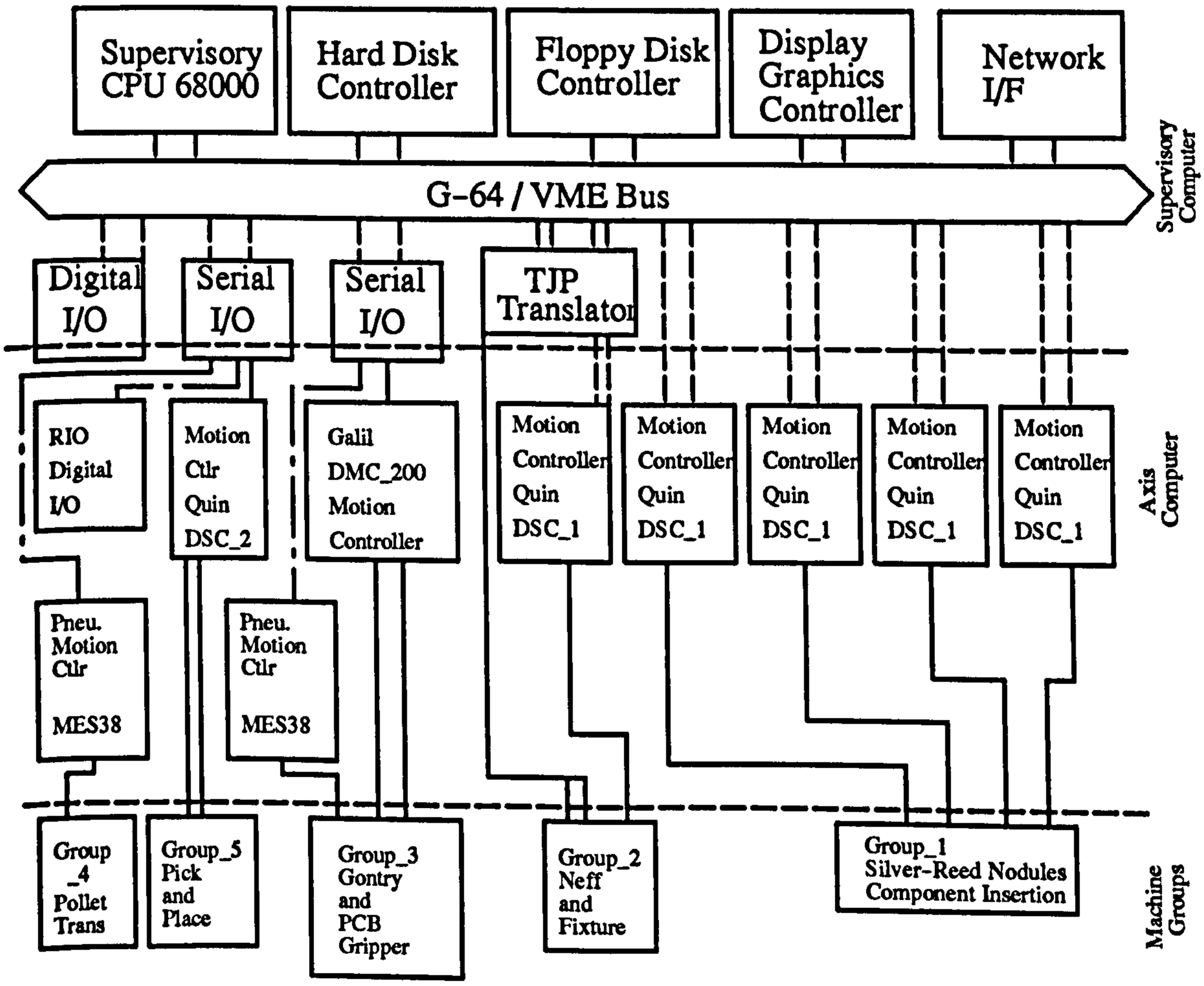


Figure 6.1 Loughborough University's Control Architecture. Source: Harrison.

6.4.2 Interfacing Software

Communication with the servo module is performed by the dsc1 device driver. The DSC-1 uses device driver functions to provide all the interface functions between the axis controller module(s) and the host OS9 system. They allow commands and data to be sent to and from the module, and provide the facilities for setting up position data transfers between axes to support the position mapping and broadcast functions. They also provide comprehensive mechanisms for signalling between the axis cards and the host system.

6.5 Real-Time Multi-Axis Execution Modules

6.5.1 Motion Control Applications -- Real-time Systems

A real-time system is a system which has timing constraints associated with its actions. The timing constraints are of course dependent upon the application. Stankovic [5] has defined a real-time system to be a system where the "Correctness depends not only on the logical result of computation but also on the time at which the results are produced".

Consider a real-time system composed of several processes (activities in the computer system) which perform one or more actions in order to control the motion of a mechanical system. The system may consist of several layers. An IMC structure has a machine level, a task level, a sync level and a servo level. Some important characteristics of the real-time system include:

(1) The granularity of the timing requirements for various actions.

(2) Whether the timing requirements are periodic or aperiodic.

(3) Whether the timing requirements are soft or hard.

(4) Reliability requirements of the system.

Most processes in the considered motion control system are periodic and can be divided into "computations" and "sampling/actuating" processes[4]:

I. Periodic computations { Ci, Di, Ti} Ci<Di<Ti;

II. Periodic actuating or Sampling {Cj, Dj, Tj} Cj<Dj<Tj

where C is a worst case execution time, D is the deadline and T is the period.

The type I is required to produce a result before the end of its period (due to other requirements such as communication latencies or safety margins the deadline may be required to be smaller than the period). The type II has to perform the sampling or actuating action at periodic intervals of time with only small deviation allowed from the period, ie the deadline is usually much smaller than the period.

Soft vs hard timing requirements usually refer to whether there is a meaning or not to completing the process even if the deadline of the process has expired. For a soft deadline this can be interpreted such that the meaning of completing the process diminishes towards zero as a function of time when the deadline has expired. For a hard deadline the meaning drops to zero directly after the deadline has been reached. Another characteristic is the criticality of a deadline. Most hard deadlines are critical, ie the computer system has failed if these deadlines are not met. However, the system usually can afford to miss a soft deadline.

The periodic processes performing motion control do not have hard deadlines in the sense that the system has failed if one single deadline is missed. If at any time there is no new position data available to be sent, the device driver maintains the previous position value for each axis. However, repeatedly missed deadlines can lead to degraded accuracy and even cause instability in the controlled motion. We therefore consider the deadlines of this process to be critical and avoid classifying them as hard or soft. A real-time control system should provide a guarantee that critical deadlines are always met.

In the following sections, real-time execution modules in the Sync and Servo level of IMC have been defined.

6.5.2 Axis Control

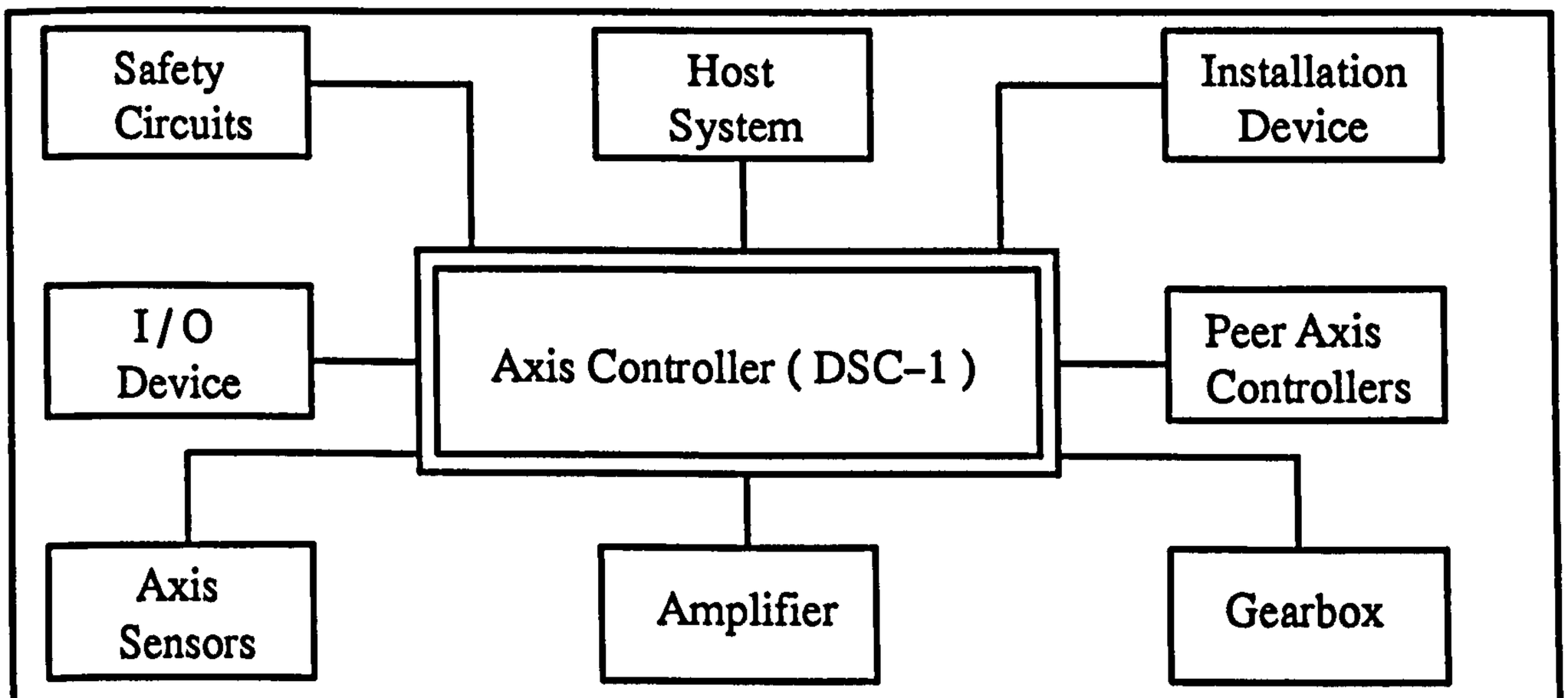
(1) Functionality Description of Axis Control Module

Axis Control module is in the lowest level of IMC. It plans the motion commands, broadcasts the commands and servos the axis motors. Since different servo-drives may be integrated into the system, the concrete implementation of this module may be different. However, the functions of the axis control module are same and can be defined as follows:

- Combination of the reference with the compensation term
- Broadcast of the motion commands to each channel
- Implementation of position control algorithms

(2) Environment Model of an Axis Controller (DSC-1M)

The DSC-1 module is a complete control system for a servo motor. It has its own processor with independent programs and data memory. It is normally supplied with comprehensive firmware which performs all the real-time functions and interfacing for the servo system. The module includes an input for an incremental position encoder, two analogue outputs, one as the main speed control signal for the power amplifier and one as an auxiliary output, a spare counter/timer input and output, seven digital inputs, seven digital outputs, and two RS-232 serial ports. Figure 6.2 shows the environment model of DSC-1.



- Drive --- Drive, power amplifier.
- Axis Sensors --- Digital incremental position encoders which provide two signals in quadrature.
- Safety Circuits --- ISO-1 module, which isalates the DSC-1 servo controller from external equipment
- Gearbox --- Gearbox, transmission, ...
- Peer Axis Controllers --- Other axis controllers.
- Installation Device --- PC, terminal, ...
- I / O Device --- External switches, a PLC, ...

Figure 6.2 Environment Model of DSC-1

(3) Function Charts in Axis Control Module

The software in the Axis Control Module was written in a mixture of 68000 assembly language and C. The dsc1 device drive (including interrupt service routine) and DSC–1 control functions were written in assembly language due to the low–level activities it performs, and in order to be as quick and efficient as possible. High–level language C access to the driver is provided through a library of device driver interface C functions, which may be linked with the other C functions at compilation time. The function chart is shown in Figure 6.3. Appendix III includes the C–code of these functions.

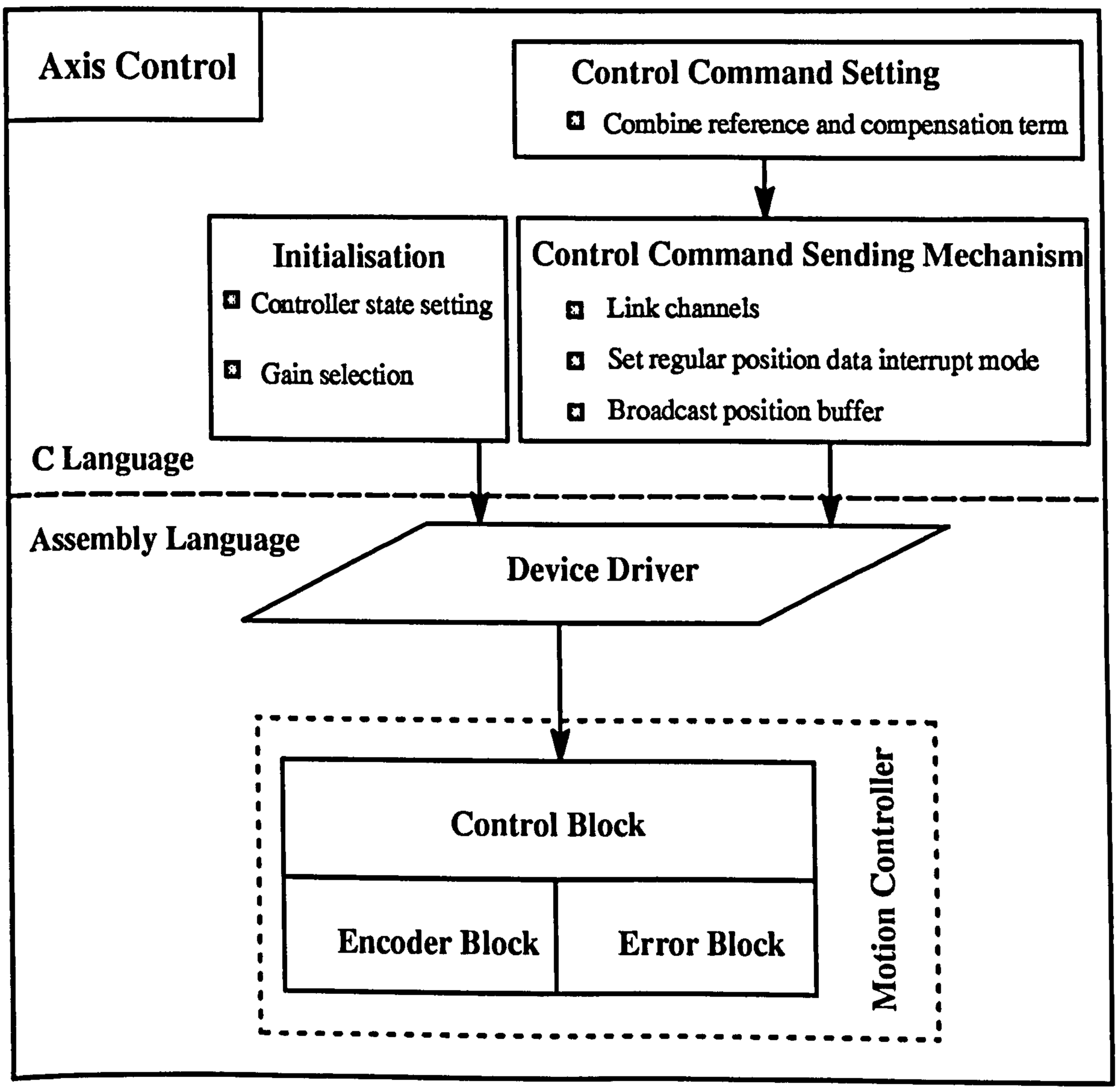


Figure 6.3 Function Chart of Axis Control Module

6.5.3 Motion State Assessment Module

The motion state assessment module maintains the best estimate of the state of the machine in real-time. It includes the following operations:

- Sensor data capture
- Sensory processing

(1) Sensor data capture

For event synchronisation, the motion control systems must be able to coordinate their movement with an external motion event. With precision servo control, the purpose is to be able to do this coordination quickly and accurately, often 'on-the-fly'. The actual methods used to synchronise the motion under direct control with external events have tremendous impact on the speed and accuracy of the entire process.

When the events occur at a regular time, then continuous synchronisation is required. The external event is usually a timer or some trigger[3]. The interval of the events is the servo sample time. Because the events can be pre-planed and are cyclical, the data capture can be set cyclically.

Another class of application is the real-time capture of axis position under control at the occurrence of an external event(s)[6]. This is commonly used in registration applications, such as printing or cutting on the fly, and to even out spacing of objects on a conveyer for packaging or other processing. This mechanism is also used during the homing-search move for any axis with an incremental sensor, even if the application requires no further synchronisation.

The above applications require the ability to receive a signal from a sensor at random, then take proper action based on the state of the controller, especially position, at the time of the signal. In digital electronic terms, the controller must be able to detect a signal edge, copy the contents of the position register(s) into working register(s), and perform some action based on that position information[6][7][8].

There are three methods for capturing position at a signal transition in a digital control system[6][7]. One is software polling of the input, and copying the contents of the position counter into a storage register (usually RAM) when a change in the input is detected. In a typical motion controller, which is cycling between tasks such as servo loop closure, trajectory update, move planning, error condition checking and monitoring

inputs, the potential delay in detecting the input signal can be 10 msec or more. Because the system is moving, the position captured on detection of the input signal can be substantially different from the actual position at the time of the signal. At a speed of 100 mm/sec (slow for many applications) a 10 msec delay means a 1 mm error. An error this large would be enough to destroy the quality of colour printing with multiple passes referenced to a common registration mark. To keep sufficient accuracy, this latency has often caused system designers to slow down the motion when expecting an input signal. Many machine tools require a two-pass homing procedure for this reason. The first pass is fast, but inaccurate; the second pass is short and slow to get the required accuracy.

Another method of position capture is to bring the signal to an interrupt input on the motion control processor. On receipt of the interrupt, the processor suspends whatever calculations it is performing, figures out which interrupt it received, and executes the proper instructions for that interrupt[2]. Here, that means reading the position register(s). This process usually involves a delay of several microseconds between receiving and reading the position. This is a factor of a thousand better than the polled method, but it still can limit speed and/or accuracy. To keep delays this short, it must be able to 'snapshot' the position registers in between servo cycles. If not, there will always be a potential of one servo cycle delay (approximately 1 or 4 millisecond). At 100 mm/sec, a 1 μsec delay can introduce a 0.1 micron error. A 10 μsec delay could introduce a 1 micron error at this speed[6].

The third method of capturing position on an input signal is to devote dedicated hardware to the process[6][7]. The easiest way to do this is to have the input signal directly latch the contents of the active position counter into a buffered register. Because this capture requires no software intervention, the only delays are the hardware delays – the gate delays of the position circuits. These delays are usually in tens of nanoseconds; a factor of a thousand better than the interrupt method, and a factor of a million better than the polled method.

A special register should be dedicated just to this function. Many motion controllers latch the contents of the position counter into a buffered register every servo cycle. But because the registration inputs are asynchronous to the control functions, we cannot use the same latching register for registration and the servo cycle without possibly degrading servo position information accuracy.

Because the hardware capture delays can be smaller than the minimum time interval between position counts, the hardware technique, used properly, can guarantee to capture position on the exact count of the input signal. Therefore, hardware position capture imposes no additional speed or accuracy limitations on the motion control system. There is no need to slow down the system when the registration signal is expected, unless physical constraints dictate this need.

It is important to realise the difference between the delay in capturing the position on a trigger, and the delay in starting to react to it. In many systems, particularly those with a polled position capture, these delays are the same – as soon as the controller sees the trigger, it captures the position and starts to calculate the response. However, the reaction delay can often be much longer than the delay in capture. The total response time (delay to start, calculation time, and physical response time) must be short enough to finish in time for the required synchronised action (e.g. printing, cutting, inserting).

The problem of sensor data capture mentioned at this point is a general discussion. When using the DSC–1 motion controllers to implement the Axis Control Module for continuous synchronisation control, a DSC–1 command is used to tell the servo modules to transmit the host processor various data values continuously, on each servo time step. Therefore, in the Sync level of IMC, state feedback information is available.

(2) Sensory Processing

Figure 6.4 gives a very general Sensory Processing module[9]. The Sensory Processing function is responsible for filtering incoming sensor data, comparing the data stream with predicted values supplied by the Locations in the Task Level, integrating sensor data over time, extracting a historical trace of the data values, performing some special functions on the data to allow data fusion from multiple sensors, and applying windows to the data stream in order to detect sensory input which has exceeded an event threshold.

In multi–axis motion synchronisation applications, the main function of the Sensory Processing is for providing the detected synchronisation error to the Motion Control Mechanisms.

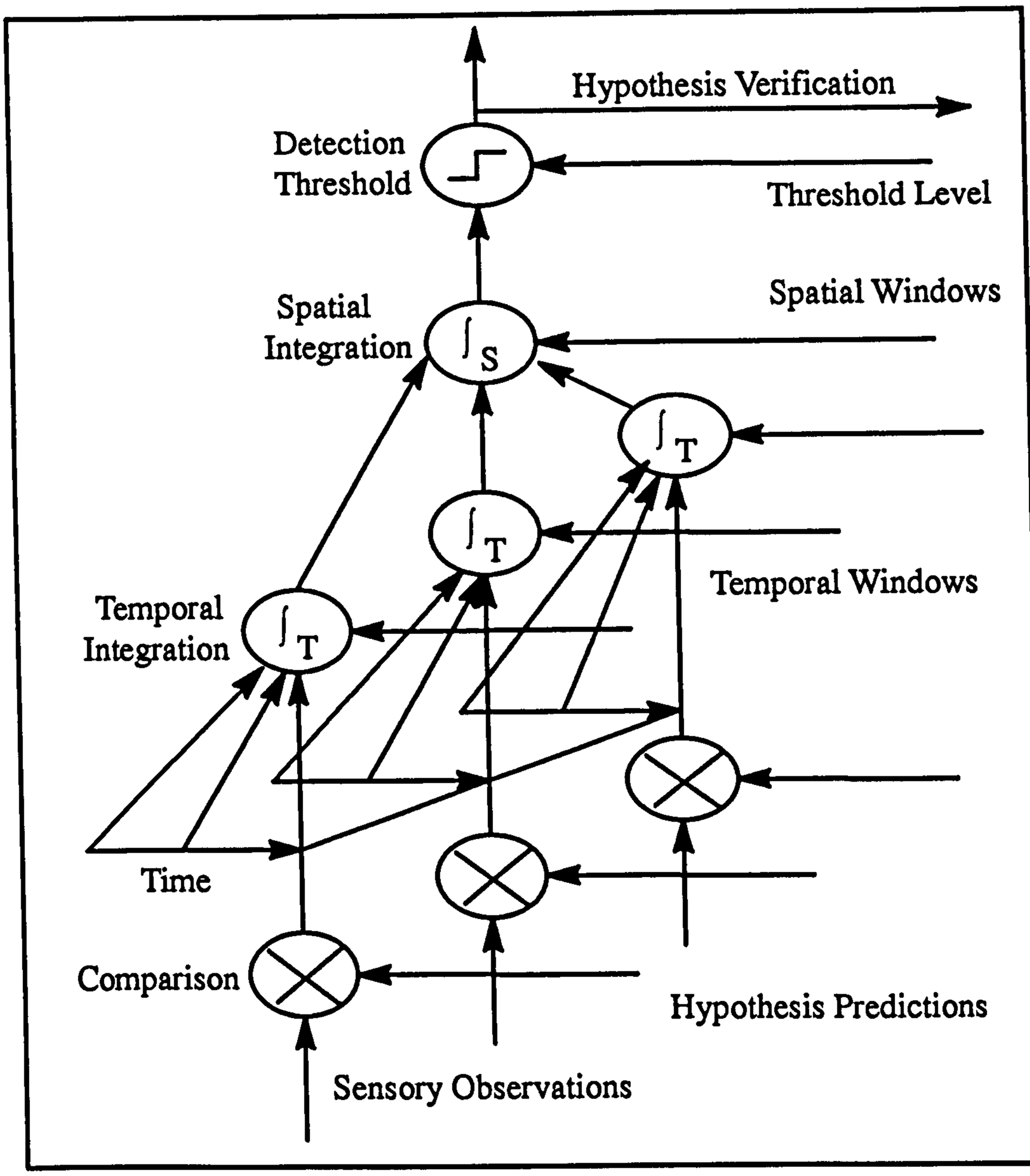


Figure 6.4 Sensory Processing. Source: Quintero.

6.5.4 Motion Control Mechanisms

The motion control mechanism is constructed by a coupling or other type of control algorithm. The 'closely coupled' components of the motion control system are inter-linked by the software mechanism. The motion of all of the axes defined in a software mechanism can then be coordinated through the coupling control algorithm. The control algorithms can be simple cross-coupling algorithms, a scalar field control algorithm and a fuzzy logic coupling algorithm as developed in Chapter 4. It is these

algorithms that control the motion synchronisation among the axes. Figure 6.5 shows the motion control mechanism module. The following section gives a more detailed description of the implementation of the fuzzy logic coupling algorithm.

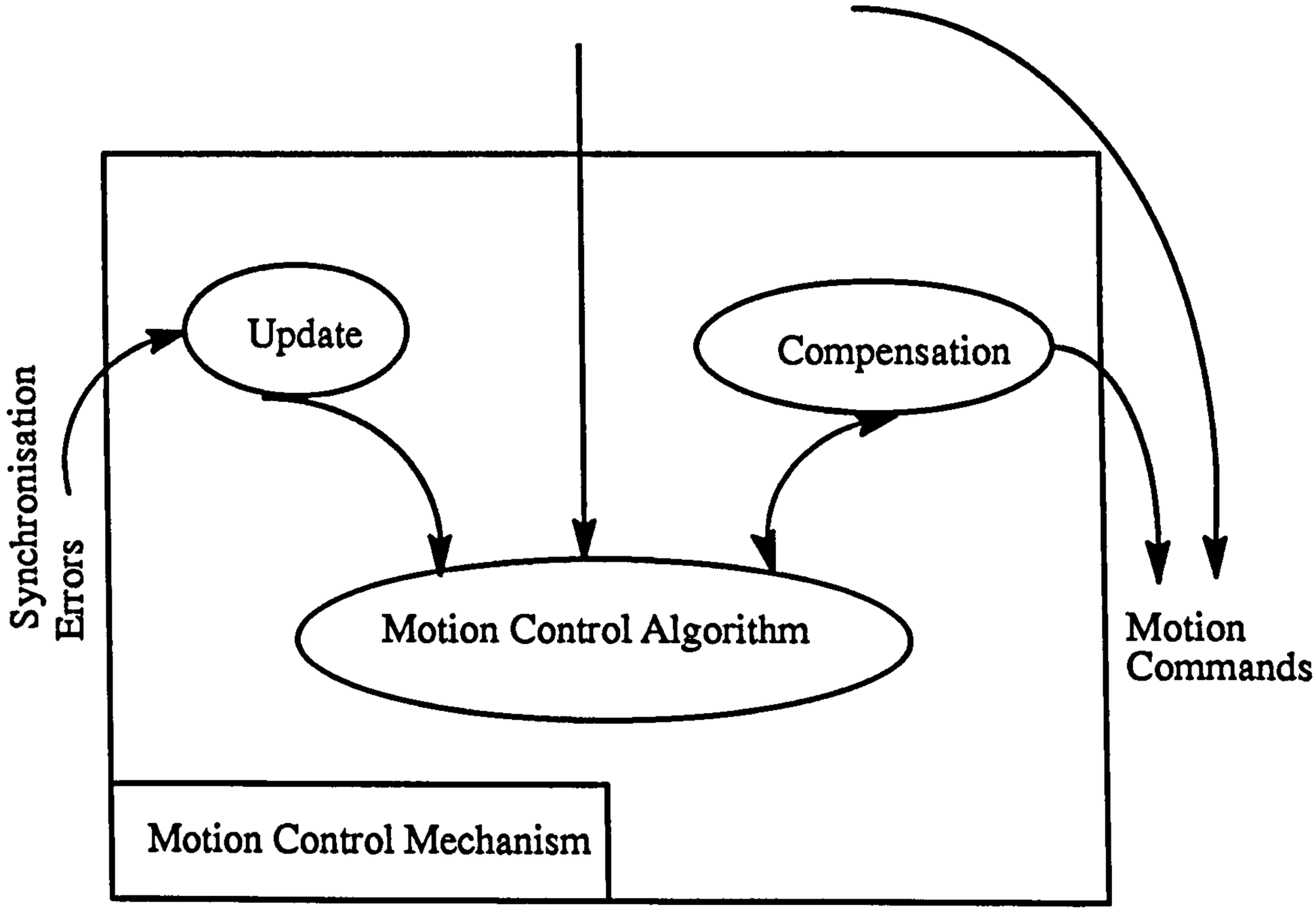


Figure 6.5 The Motion Control Mechanism Module

6.5.4.1 Implementation of the Fuzzy Logic Coupling Control Algorithm

(i) Organization of the Fuzzy Logic Coupling Control Algorithm

Figure 6.6 illustrates the flow of data through the fuzzy logic coupling control algorithm. System inputs (the synchronisation error and the rate of change of synchronisation error) undergo three transformations to become system output (the compensation terms for the reference commands of axes). First, a fuzzification process that uses predefined membership functions maps each system input into one or more degrees of membership. Then, the rules in the rule base (also predefined) are evaluated by combining degrees of

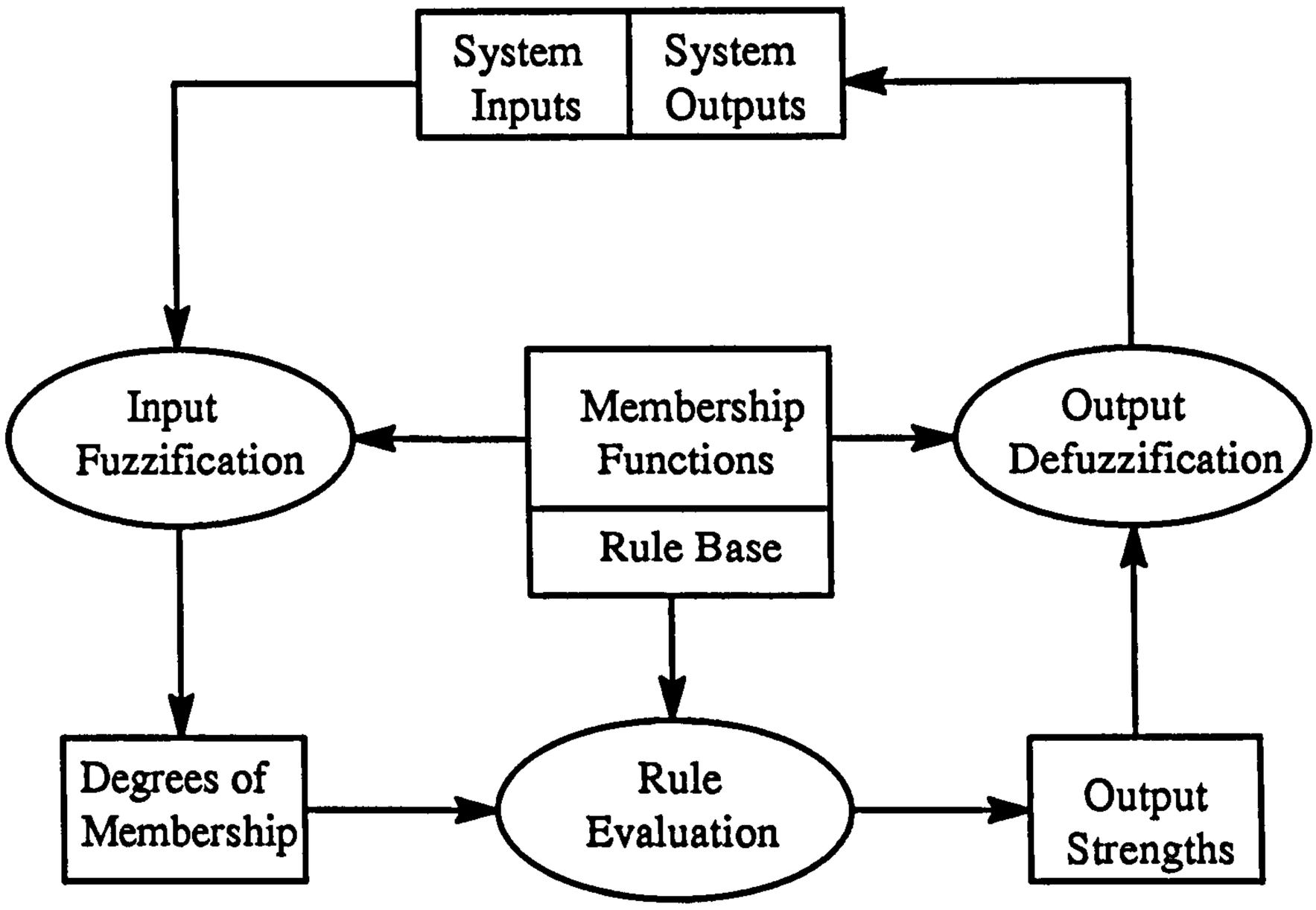


Figure 6.6 Fuzzy-system data flow. Source: Viot.

membership to form output strengths. And lastly, the defuzzification process computes system outputs based on strengths and membership functions[10].

(ii) Data Structures of the Fuzzy Logic Coupling Control Algorithm

To implement the fuzzy control algorithm in C programming language, the following types of data must be accommodated:

- System inputs,
- Input membership functions,
- Antecedent values,
- Rules,
- Rule-output strengths,
- Output membership functions,
- System outputs.

Figure 6.7 illustrates an overall linked-list arrangement of system-input and membership-function nodes. The details of these structures are shown in Figure 6.8. The system-input node is straight-forward and contains an input name, and a membership-function pointer. The membership-function structure contains two points and two slope values that describe a trapezoidal membership function. This information is used to calculate antecedent values (degrees of membership), as shown in Figure 6.9. The resulting antecedent value is stored in the "value" field of the membership-function structure.

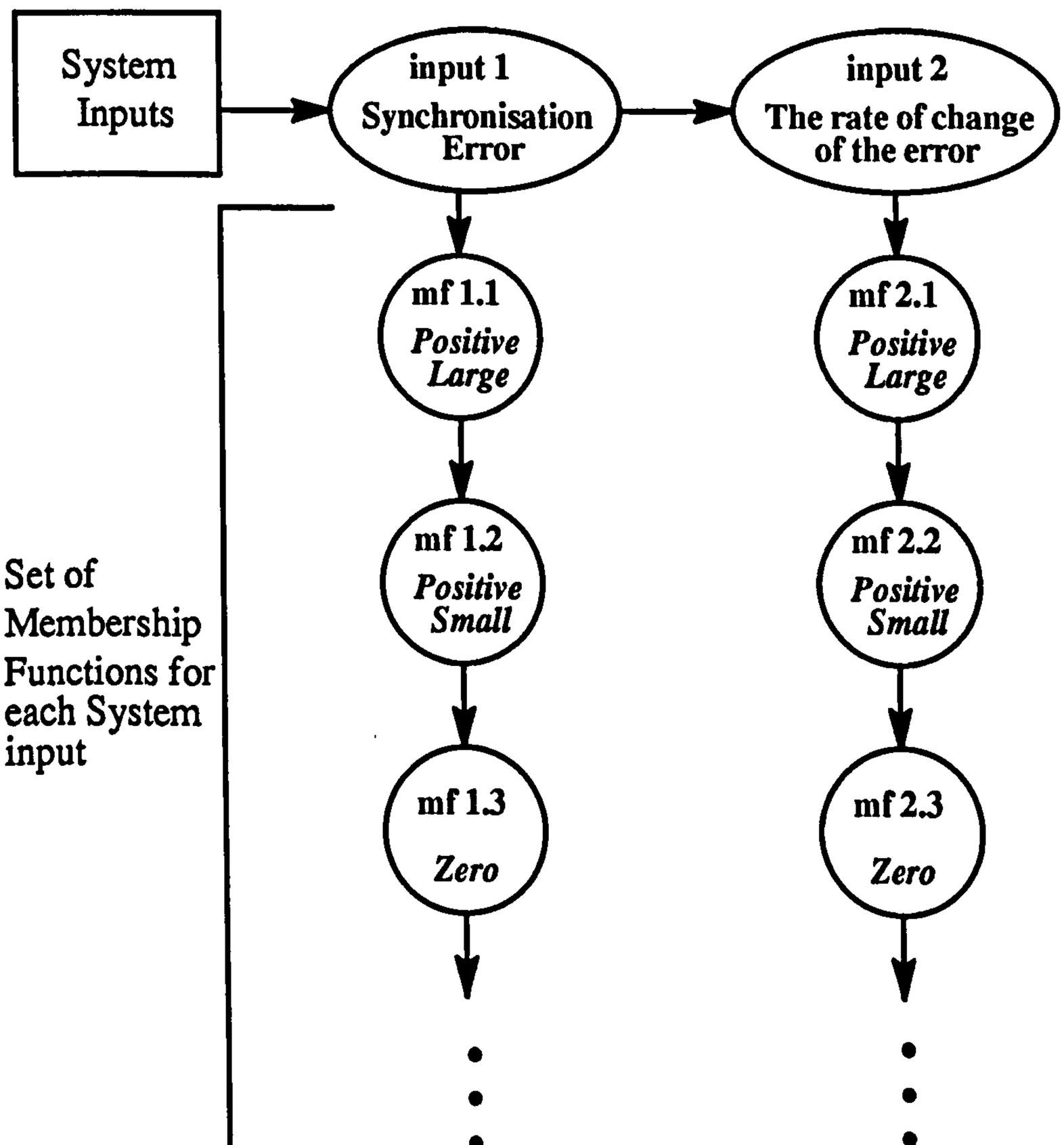


Figure 6.7 Input-data Arrangement. Example: mf 1.2 is the second membership function of system input 1.

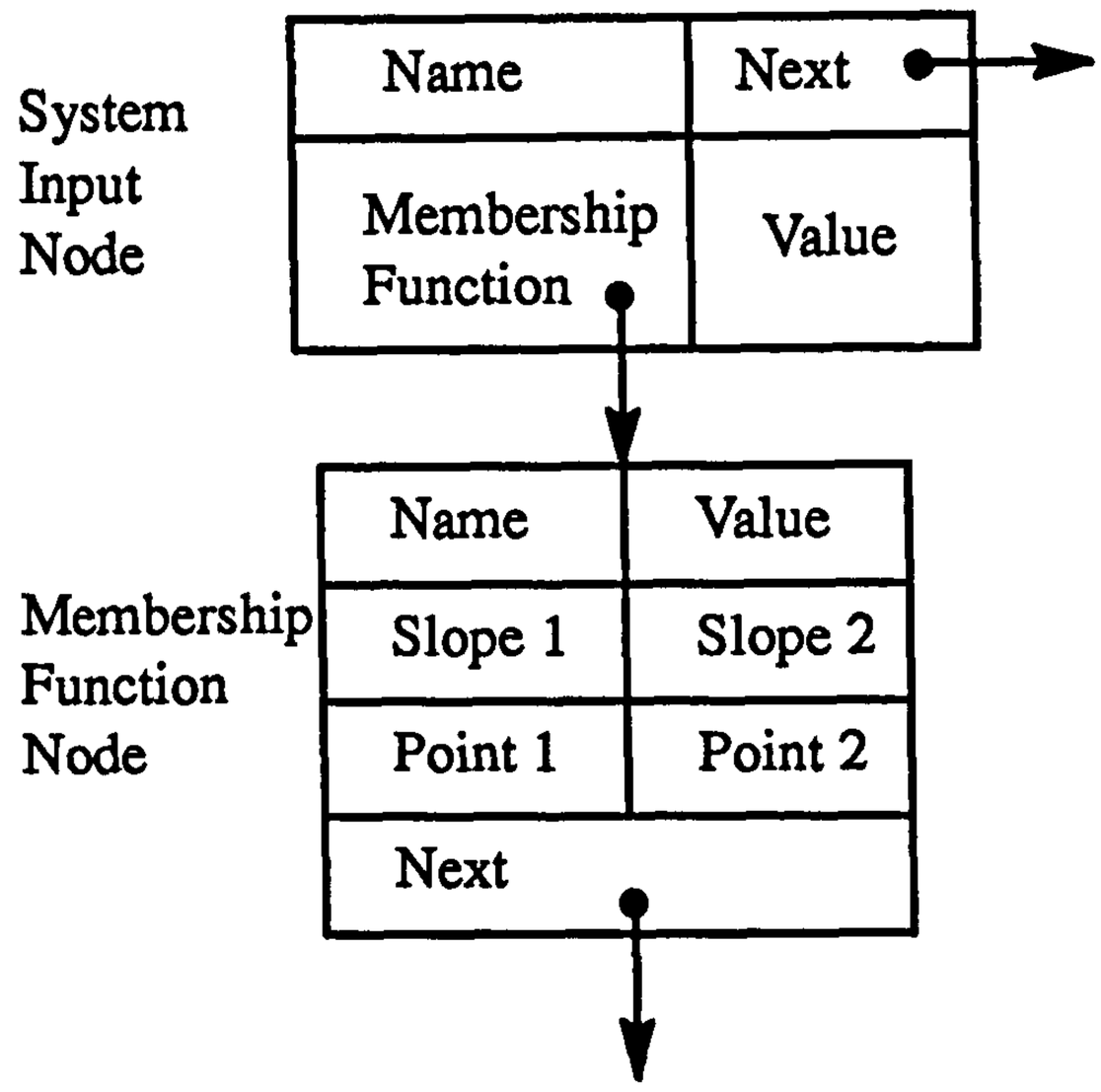
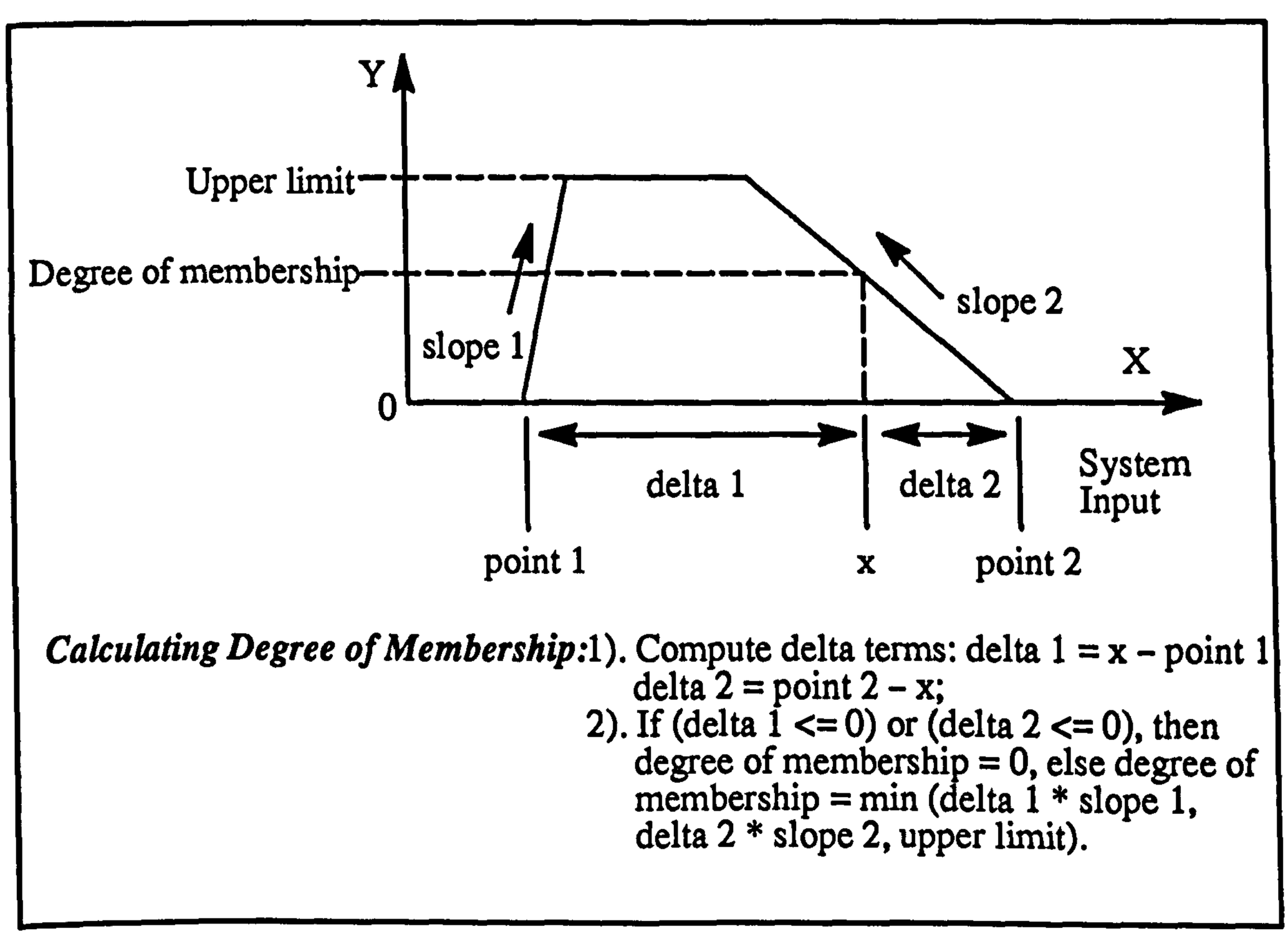


Figure 6.8 More detailed data structure. Source: Viot.



***Calculating Degree of Membership*:**1). Compute delta terms: delta 1 = x – point 1
delta 2 = point 2 – x;
2). If (delta 1 <= 0) or (delta 2 <= 0), then degree of membership = 0, else degree of membership = min (delta 1 * slope 1, delta 2 * slope 2, upper limit).

Figure 6.9 Calculating the Degree of Membership. Source: Viot.

Rules can be represented by two sets of pointers; see Figure 6.10. The first set indicates which antecedent values are used to determine the rule's strength, and the second set points to output locations where the strength is to be applied.

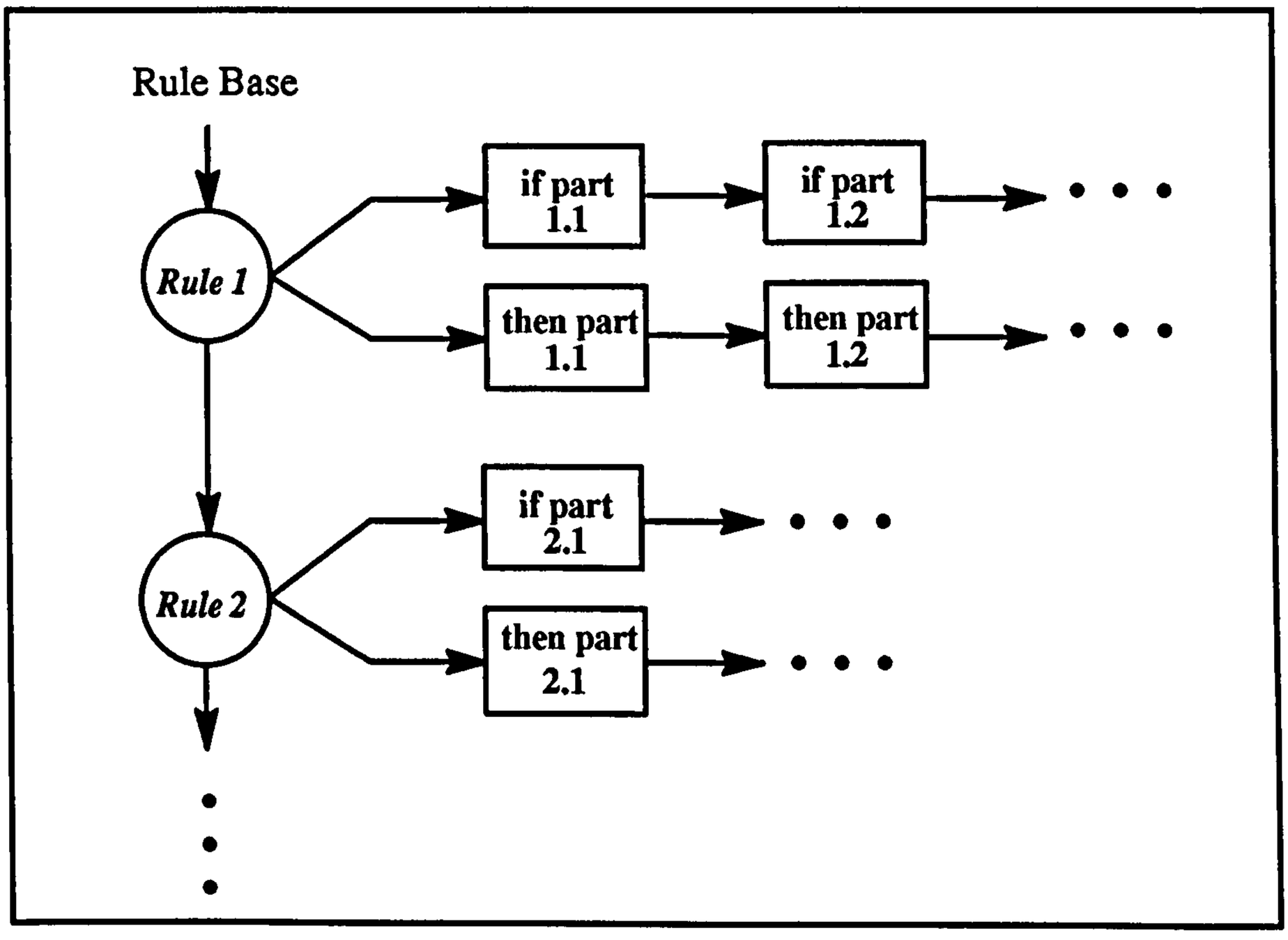


Figure 6.10 Rule-base Structure. Example: if part 1.1 is the first antecedent of rule 1. Source: Viot.

Figure 6.11 shows a data arrangement similar to the input-data structure which handles outputs and output membership functions.

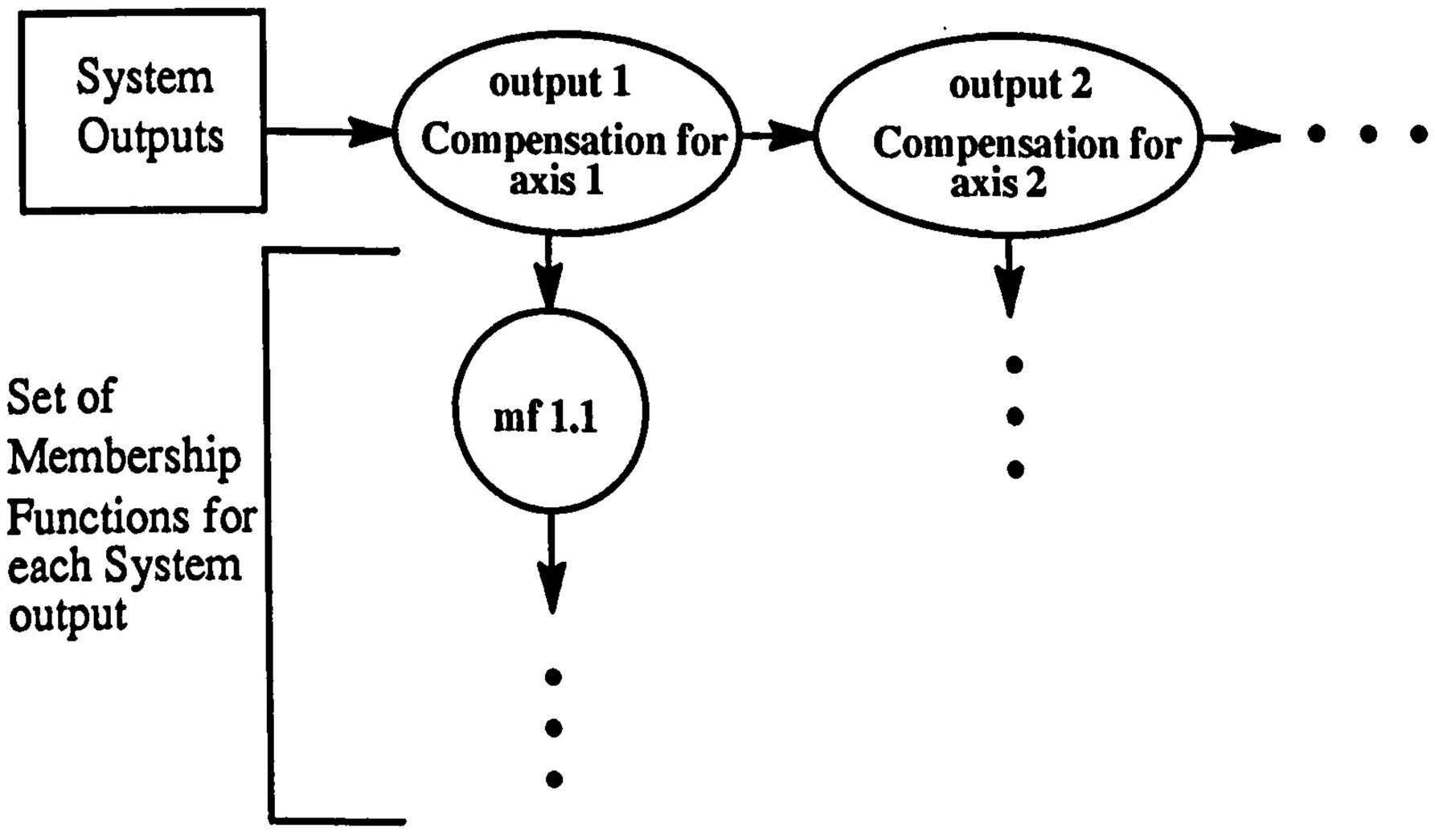


Figure 6.11 Output-data Arrangement (similar to the input-data structure shown in Figure 6.7)

Appendix III includes the C-code definition of these data structures. The implementation of fuzzy systems at the assembly language level is explained in [11][12].

6.6 Summary

A particular implementation of the IMC control structure conforming to the IMC design method has been created. The implementation is conducted in the same environment as that of UMC. Actually, only the modules in Sync level and Servo level are implemented, since the elements in Machine Level and Task Level are the same as for UMC.

The motion controllers DSC-1M are used to form the basic part of the Axis Control module for continuous synchronisataion control the axes. Although the whole implementation is based on the DSC-1M modules, the defined execution modules can be applied in a generalised way.

The special implementation issue of the fuzzy logic coupling control algorithm is included. The software mechanism which is based on this control algorithm is verified in the next chapter.

Chapter Six: References

[1] Middleditch, A. E., "Design Criteria for Multi–Axis Closed Loop Computer Numerical Control Systems", Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control, March 1974, pp36–40.

[2] Dayan, P. S., "The OS–9 Guru", Galactic Industrial Limited, 1992.

[3] "Digital Motor Control System, Programmer's Reference Manual", Quin Systems Ltd, Issue 10, December 1992.

[4] Torngren M. and Wikander J., "Real–Time Control of Physically Distributed Systems –– Application : Motion Control", Computers Elect. Engng, Vol.18, No.1, pp 51–72, 1992.

[5] Stankovic J A, "Misconceptions about real–time computing: a serious problem for next generation systems", Comput Mag , Oct. 1988, pp 10–19.

[6] Wilson, S. C., "Better Synchronisation to External Motion Events Improves System Accuracy", PCIM, June 1993, pp42–48.

[7] Scott, R., "Axis Synchronization by Encoder Following", Drives and Controls Magazine, pp 70–72, September, 1991.

[8] Meshkat, S, "Parallel DSPs Excel in Cam and Gearing Applications", PCIM, Feb. 1994, pp69–73.

[9] Quintero, R., and Barbera, A. J., "A Real–Time Control System Methodology for Developing Intelligent control Systems", NISTIR 4936, October 1992.

[10] Viot, G., "Fuzzy Logic in C", Dr. Dobb's Journal, February 1993, pp40–49.

[11] Sibigtroth, J. M., "Creating Fuzzy Micros", Embedded Systems Programming, December, 1991.

[12] Sibigtroth, J. M., "Implementing Fuzzy Expert Rules", AI Expert, April 1992.

[13] Harrison, R., "A Generalised Approach to Machine Control", PhD Thesis, Loughborough University of Technology, July 1991.

CHAPTER SEVEN

Verification of the Software Mechanism of Intelligent Motion Control (IMC)

7.1 Introduction

An experimental rig was developed in order to test the various software coupling mechanisms. This comprised a computer system which was interfaced with two DC servo–drive position control systems. An overall system schematic is shown in Figure 7.1. Two DSC–1M controllers are connected to the OS–9 based computer through a G–64 bus. When each DSC–1M motion controller is sequently linked with a PWM drive

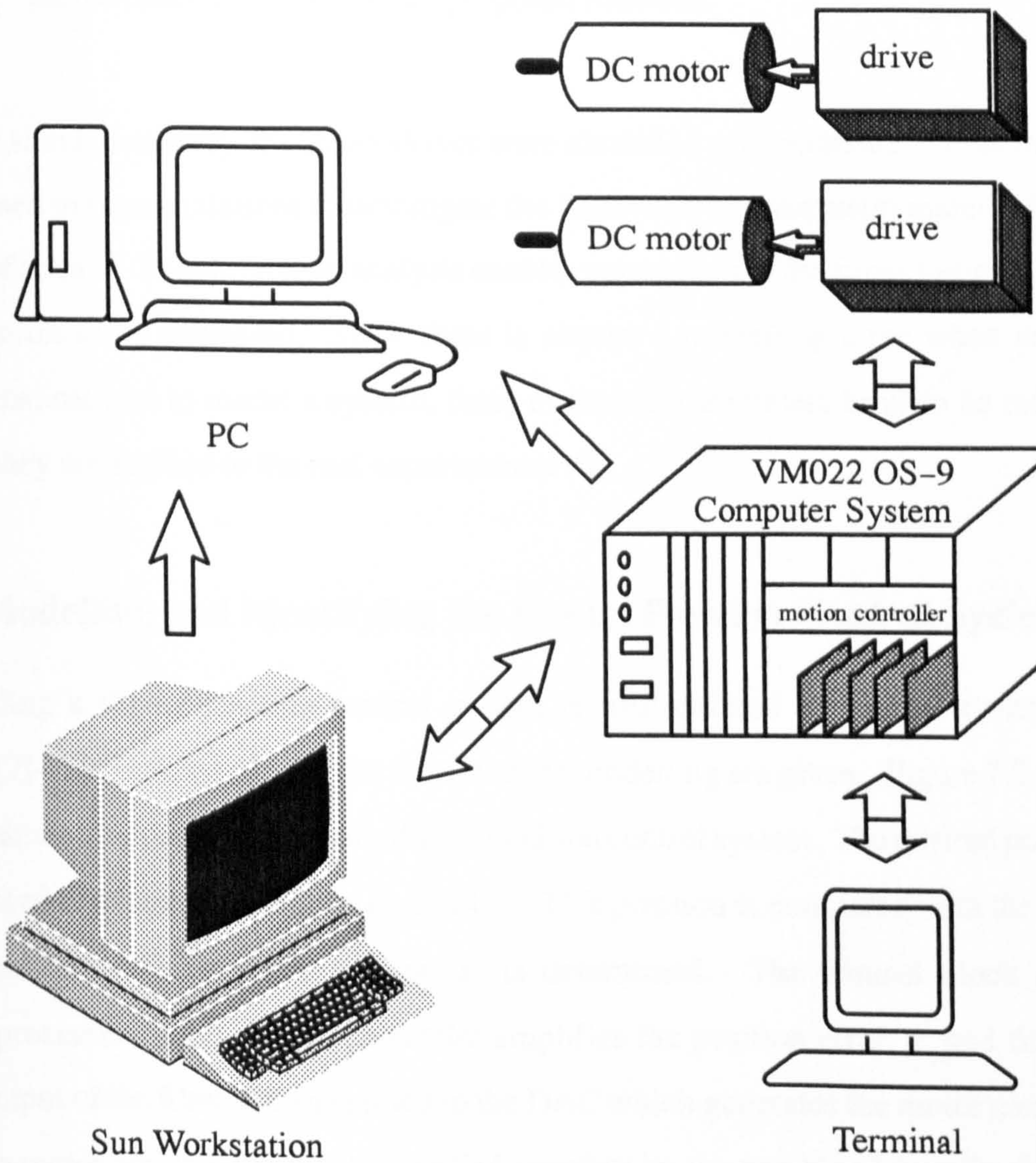


Figure 7.1 - System Schematic

amplifier, a DC motor, and a feedback encoder to form a complete servo-drive system, a simple multi-axis motion control system is established. The C-codes are written in a Unix-based Sun workstation environment, then, compiled by a UNIX-OS9 cross compiler and the result object codes run in the OS-9 based computer. Logged data is analysed by using Excel which is a DOS based software package.

Before running the programs in the real system, a simulation study is carried out. The reasons for the simulation study are:

(i) simulation is the commonly used method to verify fuzzy systems[1];

(ii) simulating the system before linking to a real system is a safe and easy method for debugging;

(iii) simulations can extend the current hardware's limits to show the performance boundary of the proposed methods.

For the simulation study, the servo-drives were identified and modelled. These models were used to run simulations to investigate the behaviour of the system under different types of control. The simulation analysis enables parameters of the fuzzy logic coupling mechanism to be evaluated. Since there is always a modelling error when using a mathematical tool to model a system, these evaluated parameters have to be refinded when they are applied to the real experimental rig.

7.2 Modelling and Identifying the Digital Position Control Systems

Modelling a digital position control system is well defined in control system text books[2]-[5]. In Appendix IV, the details of the modelling are given. Figure 7.2 shows a representative block diagram of a digital position control system. The desired position, expressed in encoder quadrature counts is r. This position is compared with the actual feedback, c, and the position error, x, is determined. The control block of the microprocessor-based motion controller amplifies the position error, x, and filters it. The output of the filter is then applied to the DAC which generates the motor command v. The motor and the driver are modelled together by the combined transfer function M(s). This is the transfer function between the motor command, v, and the angular position of the motor, θ. The motor angular position is sensed by the encoder which

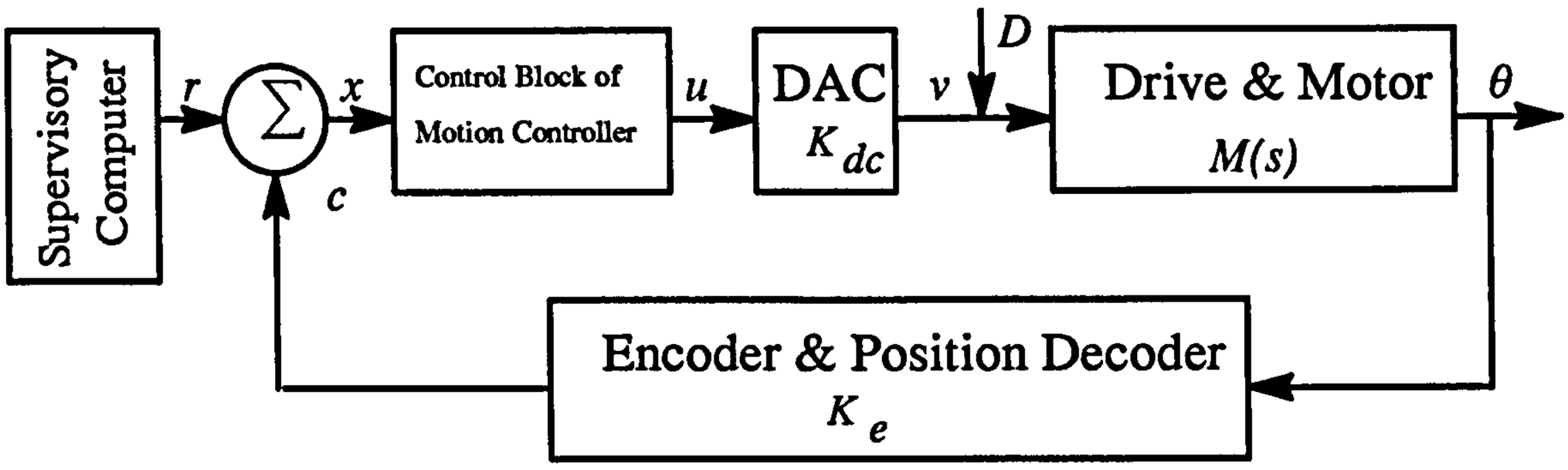


Figure 7.2 Modelling position control system elements

generates two signals in quadrature, channels A and B. These two signals are then applied to the position decoder, which generates the position feedback, ***c***.

7.2.1 System Identification

For the simulation and experimental studies, two position control systems were chosen which have the same configuration. The sampling period chosen for the implementations was 4 ms, corresponding to a sampling frequency of 250 Hz which is the same as the motion controller used for the experiments. The motors are DC motors, which are driven by a PWM drive. The characteristic of the motors is given in Table 7.1[6]. The

Table 7.1 Characteristic of the DC Motor

CHARACTERISTICS	MOTOR RATINGS					MECHANICAL DATA									ELECTRICAL DATA						
	Continuous Output Power (Watts)	Continuous Stall Torque (Nm)	Peak Torque (Nm)	Max Stall Current (Amps)	Max No Load Speed (r.p.m.)	Rotor Moment of Inertia ($Kg\ m^2$)	Mechanical Time Constant (ms)	Damping Constant (Nm/rad/sec)	Static Friction Torque (Nm)	Max Shaft Radial Load (kg)	Max Shaft Axial Load (Kg)	Max Armature Temperature (^{o} C)	Thermal Resistance (Am to Amb)(^{o}C/W)	MKotor Weight (Kg)	Torque Constant (K_T) (Nm/A)	Voltage Constant (K_E) (V/K r.p.m.)	Armature Resistance @ 25 °C (Q)	Terminal Resistance @ 25 °C (Q) at 4 Amps	Electrical Time Constant (ms)	Max Pulse Current (Amps)
Motor Type No S540-1A	*80*	*0.2*	*1.0*	*2.97*	*6000*	*2.0* $x10^{-5}$	*8.4*	*6.7* $x10^{-6}$	*2.5* $x10^{-2}$	*4.54*	*2.27*	*155*	*5.0*	*1.0*	*7.1* $x10^{-2}$	*7.5*	*1.24*	*1.6*	*2.1*	*23*

encoders have N=250 lines per revolution. The control block of the motion controller is shown in Figure 7.3. The DSC-1 has a PID with velocity feedback and feed-forward control algorithm.

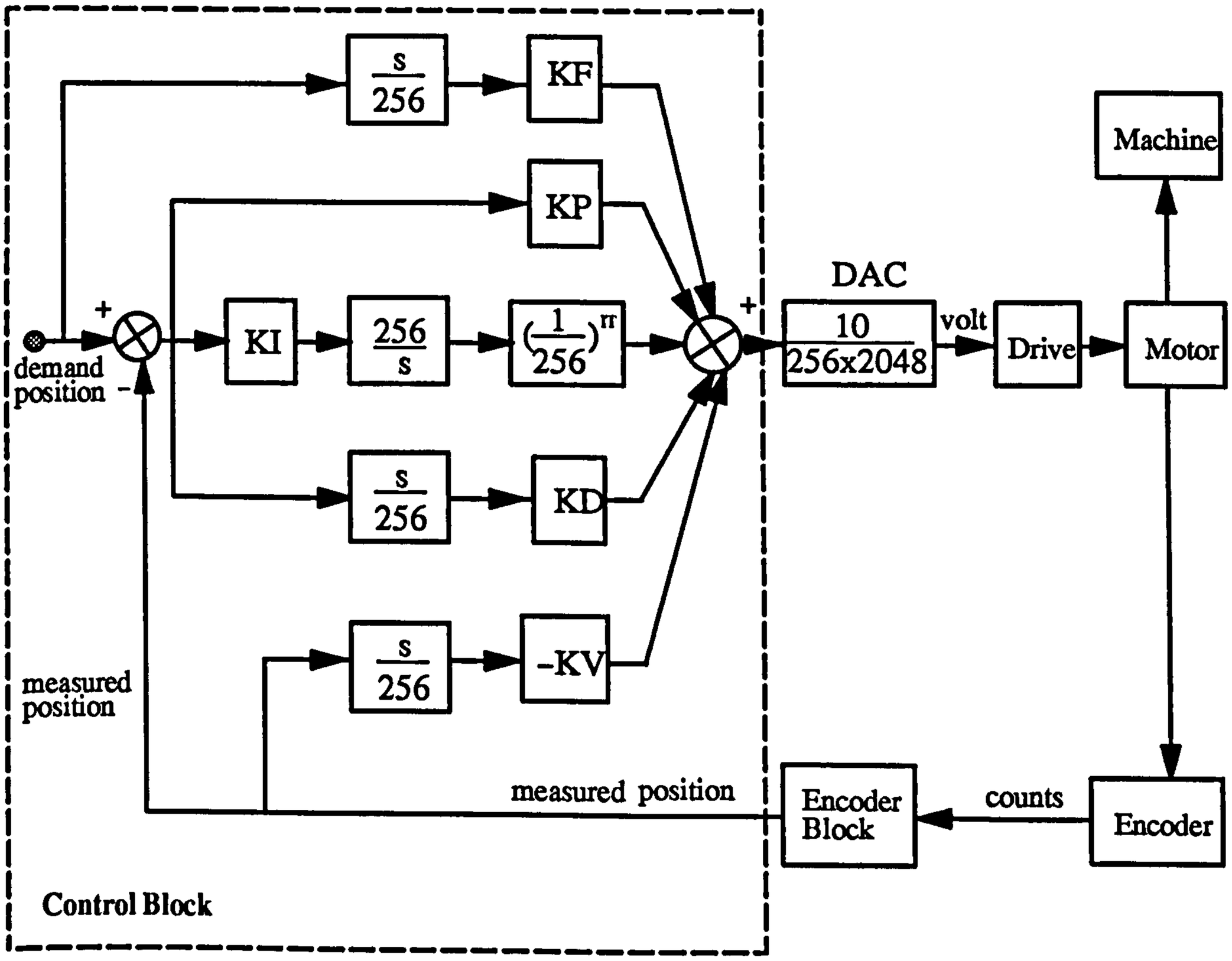


Figure 7.3 The DSC-1 Servo-drive System. Source: Quin.

With reference to Appendix IV, the system parameters can be obtained as following.

From Table 7.1, T_m=0.0084 s, T_e=0.0021 s, and the transfer function of the motor is:

$$M(s)=\frac{1/K_T}{s(0.0084s+1)(0.0021s+1)}$$

$$=\frac{1/0.071}{s(0.0084s+1)(0.0021s+1)} \quad [\frac{rad}{volt}] \qquad (7\text{-}1)$$

The position is sensed by an encoder with a line density of N=250 lines/rev. Since the encoder has two channels in quadrature, it is possible to increase the resolution to 4N.

In this case the effective resolution is 1000 quadrature counts per revolution. This corresponds to a feedback gain of:

$$K_e = \frac{1000}{2\pi} \quad [\frac{\text{counts}}{\text{rad}}] \qquad (7\text{–}2)$$

From the Figure 7.3, the DAC gain is:

$$K_{dc} = \frac{10}{256 \times 2048} \quad [\frac{\text{volts}}{\text{count}}] \qquad (7\text{–}3)$$

The sampling time(T) is 4ms. Therefore, the model of the ZOH is

$$F(s) = e^{-sT/2} = e^{-0.002\,s} \qquad (7\text{–}4)$$

At this point, it is best to describe all of the system elements by block diagram. This is done in Figure 7.4.

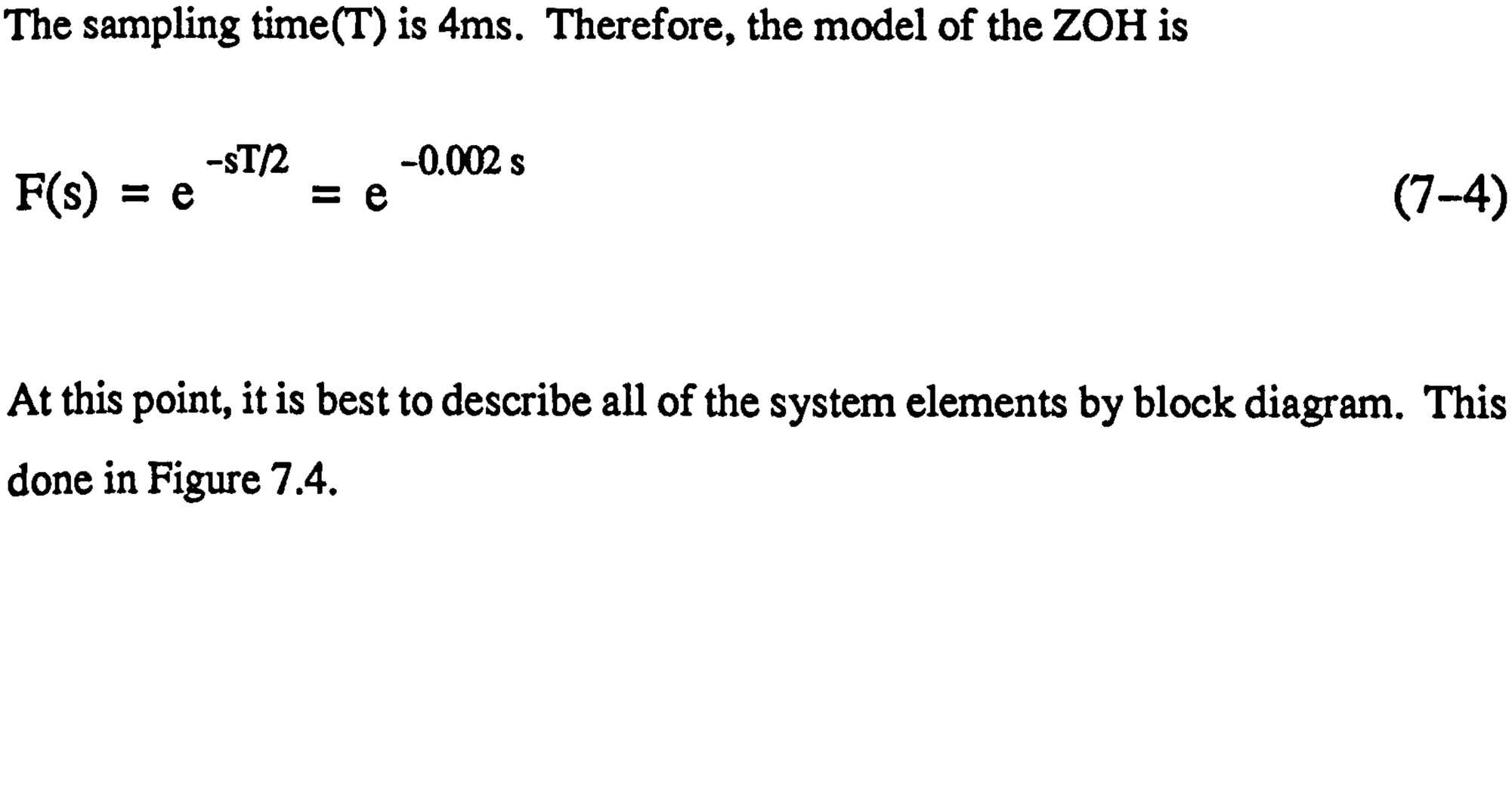


Figure 7.4 System Elements

7.3 Case Studies

Programs have been written to execute two kinds of motion synchronisation – a 2–axis system with a linear position relationship and a 2–axis system with a nonlinear circular position relationship. The position relationships of these systems are shown in Figure 7.5.

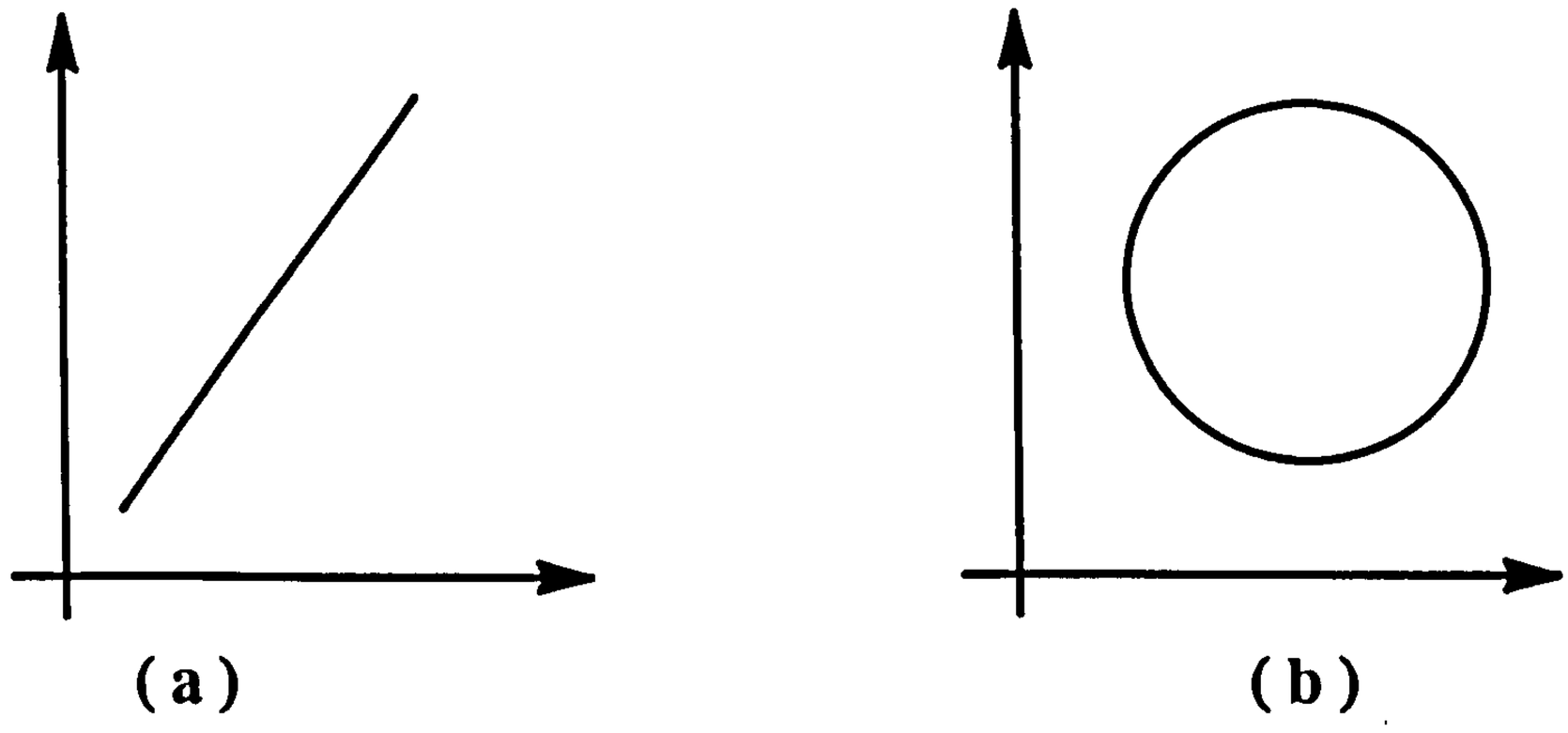


Figure 7.5 (a) Gearing path, (b) Circular path.

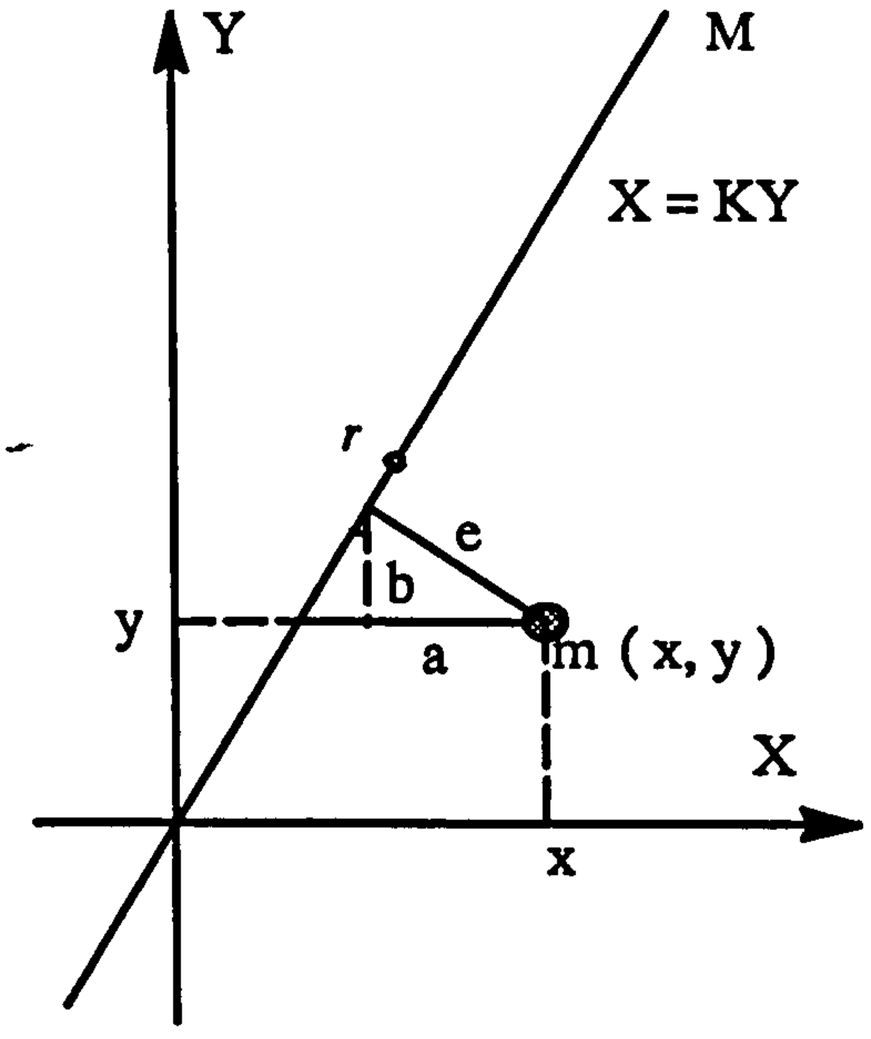
7.3.1 Linear Gear Implementation

A linear position relationship of a 2–axis system can be represented by a gearbox path (x=ky). As shown in Figure 7.6, an actual position point **m** (x,y) may not be on the path, and a synchronisation error (**e**) occurs. In order to meet synchronisation requirements, the actual point **m** should be "forced" back to the path **M** (ie to eliminate **e**). For each axis, the modified value will depend on the other axis'.

One case is to "force" **m** back to **q** (as the nearest state point to **m** lying within the path **M**). The modified value for each axis will be :

$$a = (x - ky) / (1 + k^2)$$
$$b = (x - ky)\, k / (1 + k^2) \qquad (7\text{–}5)$$

From the above equations (7–5), the compensation for y axis is ***k*** times than that for **x** axis.



$$e = (x - ky) / \sqrt{1 + k^2}$$

$$a = (x - ky) / (1 + k^2)$$

$$b = (x - ky) k / (1 + k^2)$$

Figure 7.6 A Two Axis System with Linear Relationship, X=KY

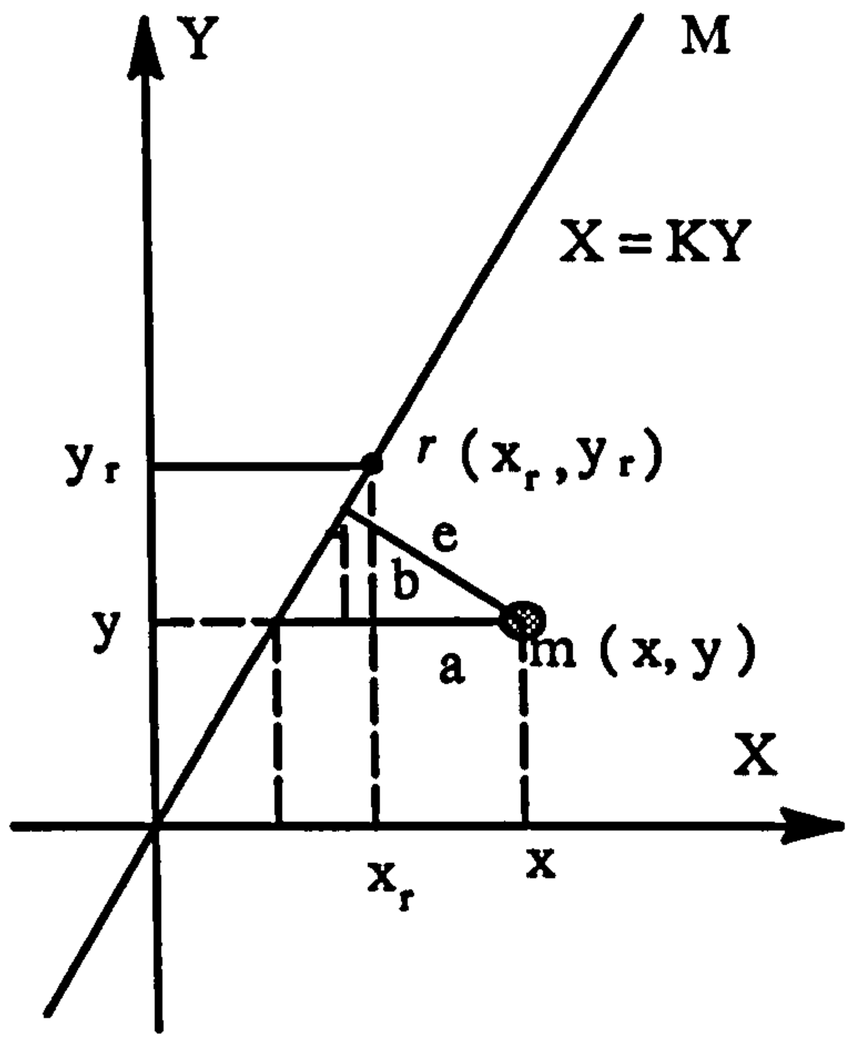
From the synchronisation viewpoint, it does not matter that the m is "forced" back to any specific point in the path. However, in order to reduce the tracking error, it is better to force *m* back to the reference point (*r*). However, the modified values for each axis have to be related to each other. If axis *X* reduces by *a*, then axis *Y* should increase by *b*. If axis *X* reduces by less than *a*, then, axis *Y* must increase by more than *b*.

In the case of trying to approach the reference point *r*, the modified values for each axis will be:

$$a = x - x_r$$

$$b = y - y_r \qquad (7\text{-}6)$$

Figure 7.7 shows this case. The above expressions imply that the compensation term for each axis is trying to completely eliminate the following errors of the axes. However, this approach may increase the synchronisation error during the reduction of the following errors of the axes, since it is only partially true that when the following errors of all axes are reducing, the synchronisation error will reduce. Figure 7.8 shows probable steps in reduction of the following errors of the axes to approach the reference point *r*. Although the axial errors $(x_1 - x_r)$ and $(y_1 - y_r)$ at point m_1 are smaller than



$$e = (x - ky) / \sqrt{1 + k^2}$$

$$a = x - x_r$$

$$b = y - y_r$$

Figure 7.7 A Two Axis System with Linear Relationship, X=KY

$(x - x_r)$ and $(y - y_r)$ at point m, the synchronisation error e_1 at m_1 is larger than that e at m. Therefore, the synchronisation control and tracking control have to be separated. The above expressions (7–6) can not be used to achieve the two control goals.

In an extreme case, when one axis has reached saturation (or in master–slave configuration), the other axis (or slave) has to take complete responsibility to maintain synchronisation. The modified value for the unsaturated or slave axis will be:

$$a = 0$$

$$b = y - x/k \qquad (7\text{–}7)$$

Figure 7.9 gives this derivation. In this case, error **b** instead **of e** is used as the target for elimination to maintain synchronisation.

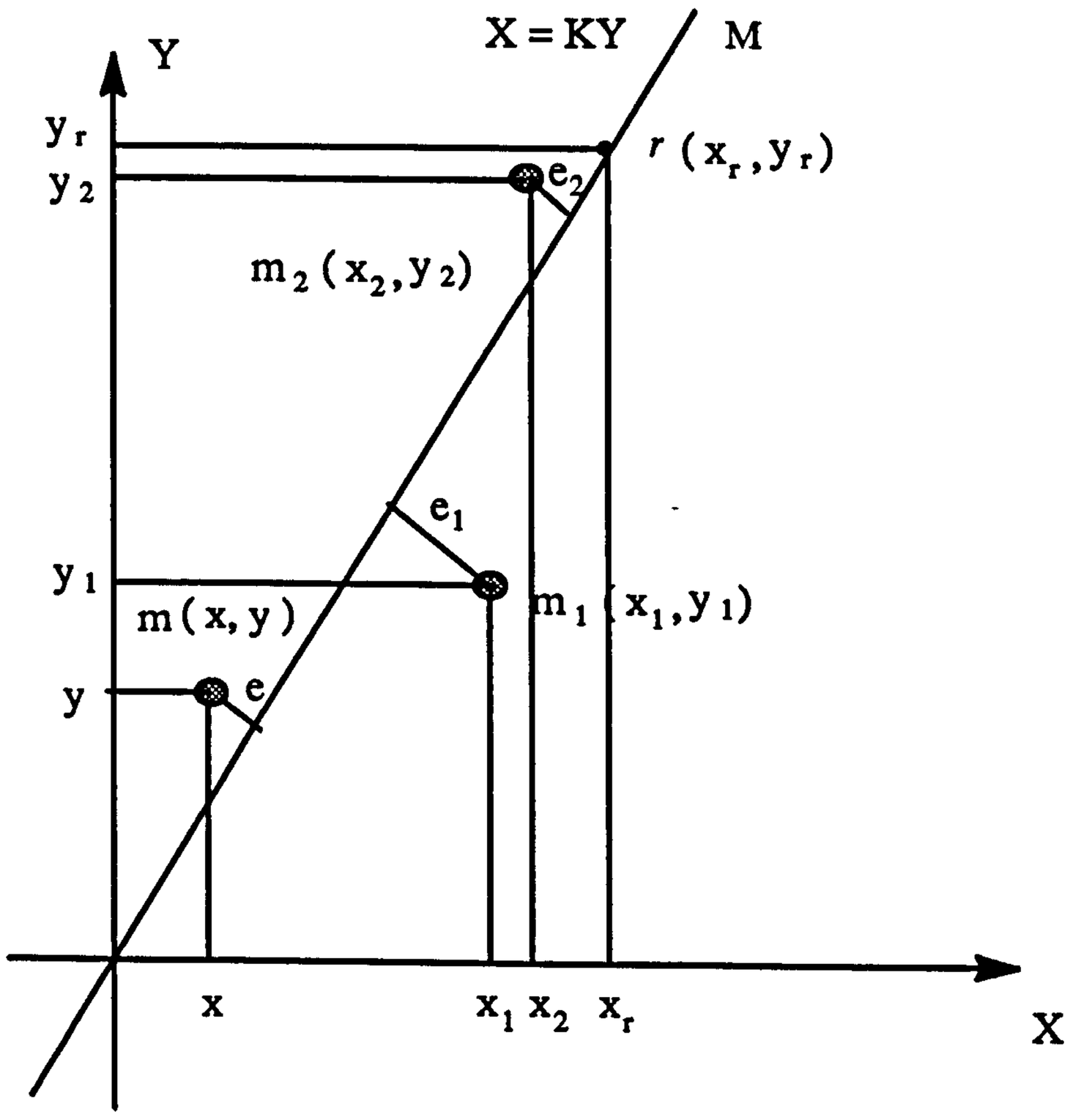
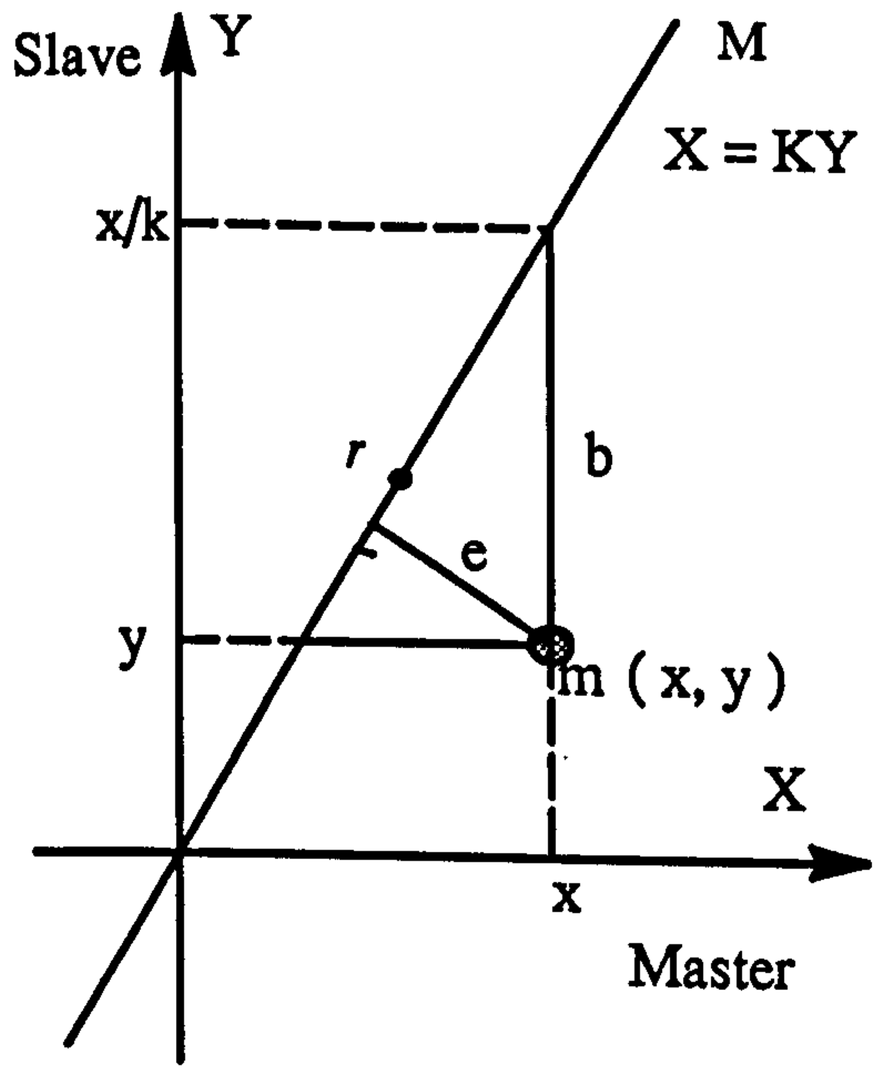


Figure 7.8 A Two Axis System with Linear Relationship, X=KY



$$e = (x - ky) / \sqrt{1 + k^2}$$

$$a = 0$$

$$b = y - x/k$$

Figure 7.9 A Two Axis System with Linear Relationship, X=KY

7.3.2 Circular Path Implementation

The reasons that a circular position relationship of a 2–axis system is chosen to represent a nonlinear relationship between the axes are:

- Such a path might be used to describe a trajectory for a tool on an xy–table. Applications are known where a square path with rounded corners is used (for example, glue–laying[7]).
- A circular path is relatively easy to implementation compared with many other nonlinear relationships[7]. When the position of each axis change continuously in a sinusoidally mode with time, the joint point of the 2–axis' positions will follow a circular path at constant speed.

In Figure 7.10, a graphical analysis for synchronisation control is given. When the synchronisation error is:

$$e = \sqrt{(x - x_o)^2 + (y - y_o)^2} - R \tag{7–8}$$

the modified values for each axis will be:

$$\begin{aligned} a &= e\ cos\theta_m \\ b &= e\ sin\theta_m \end{aligned} \tag{7–9}$$

where

$$\begin{aligned} sin\theta_m &= \frac{y - y_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}} \\ cos\theta_m &= \frac{x - x_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}} \end{aligned} \tag{7–10}$$

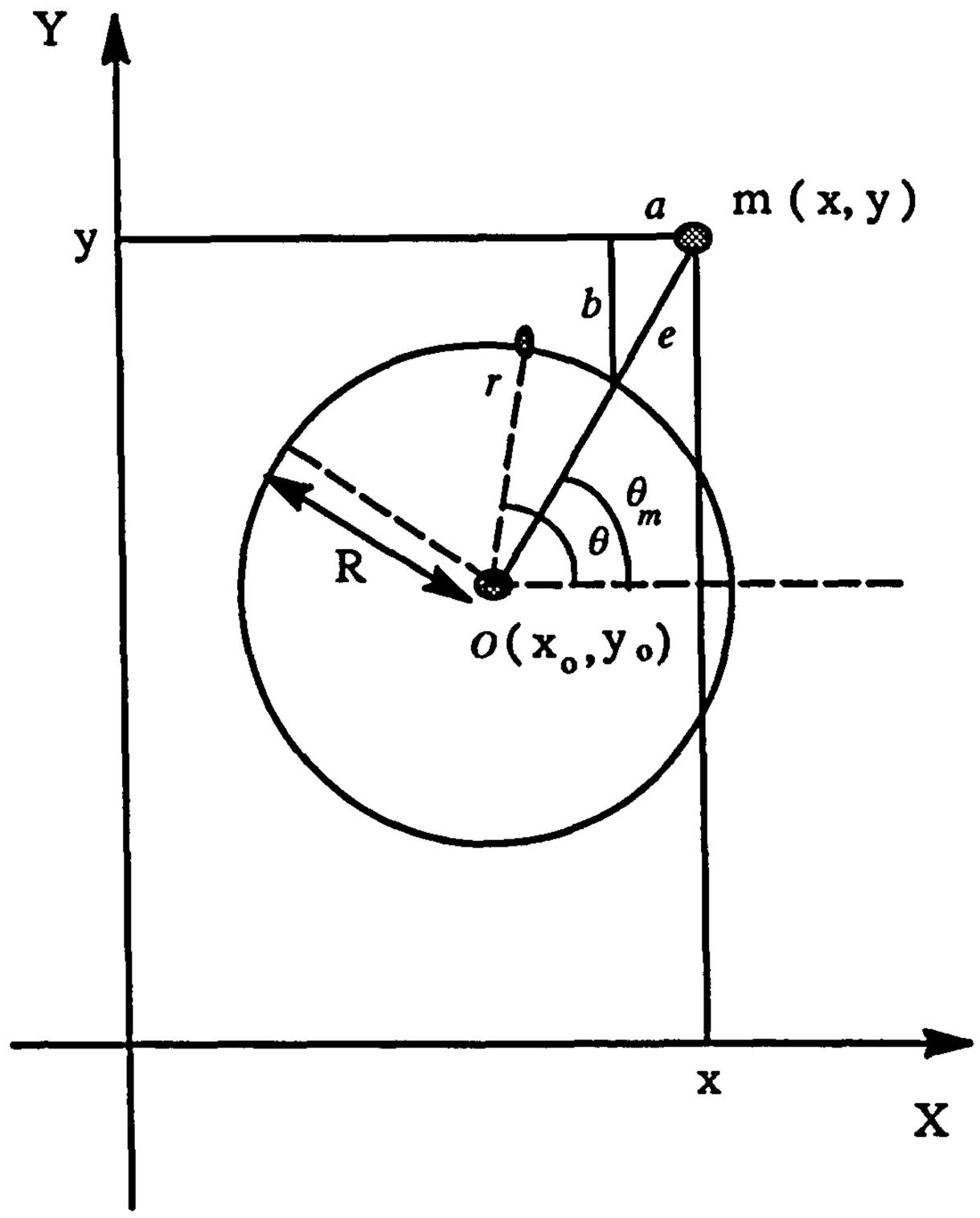


Figure 7.10 A Two Axis System with a Circular Relationship

7.4 Simulations

The C programming language was used to write all software modules for the IMC. For the simulation study, two position control systems were chosen which have the same configuration as shown in Figure 7.3. The sampling period chosen for the implementations was 4 ms, corresponding to a sampling frequency of 250 Hz which is the same as that in the motion controllers used.

The system was simulated using the facilities described in Figure 7.1. The programs were written in a Unix workstation environment, compiled by a Unix-OS9 cross compiler and the object codes run on an OS-9 based target machine. The logged data

was analysed using Excel which is a DOS Windows software package. The system under simulation is based on the schematic of Figure 7.11. The controller and plant

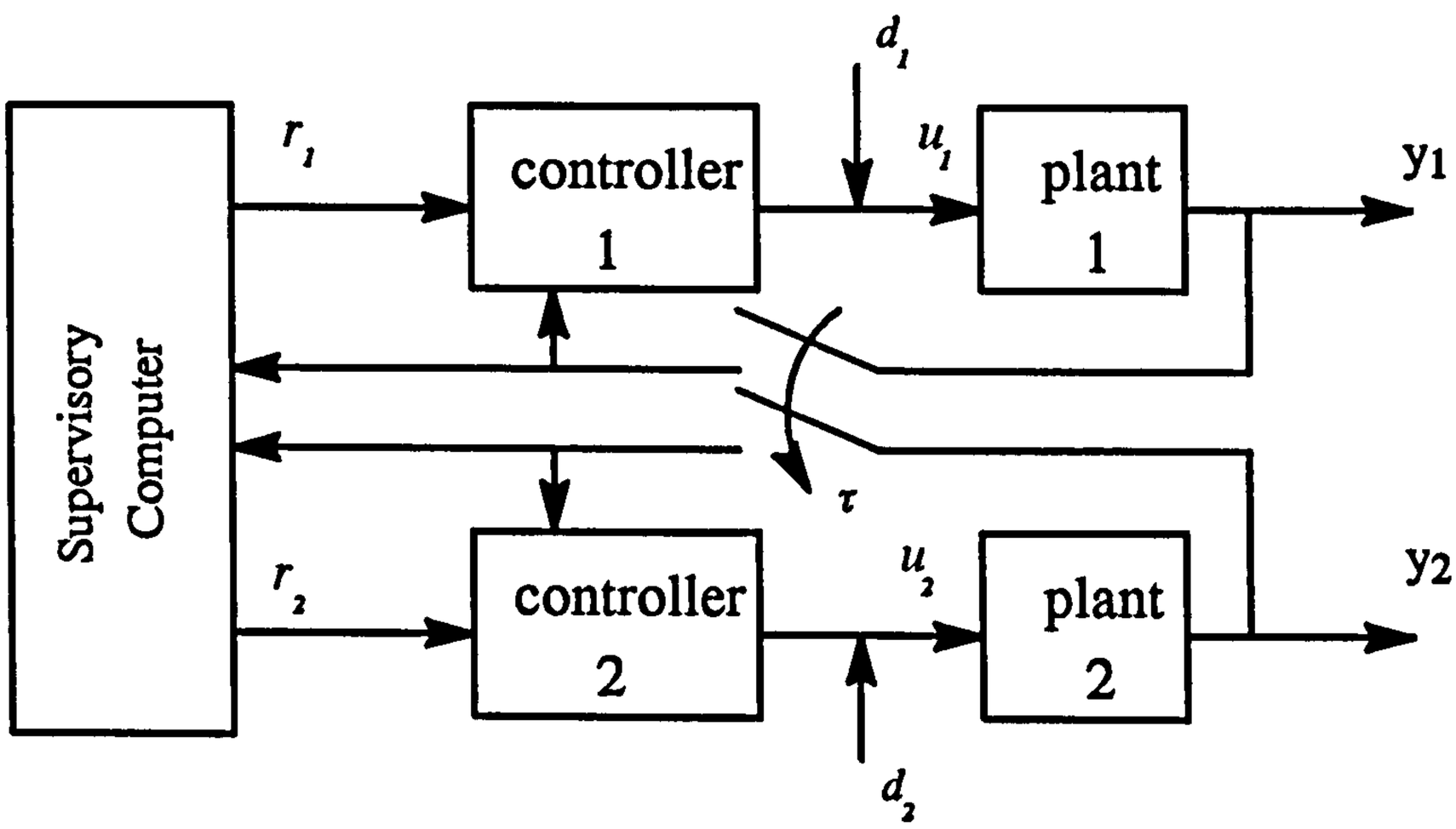


Figure 7.11 Block Diagram of the Simulated System. Souce: Jenkinson.

(drive & motor) are modelled using mathematical functions (for filtering the data) are implemented as software modules. When the system is "running", these modules enact the "response" in order during each "sample time τ (4ms)". The disturbance added to one of the "axes" is simulated by applying a step signal at d_1 or d_2.

7.4.1 Test Simulation Using a Linear Relationship

The two "motors" are assumed to "run" at constant rates where a speed differential may exist. The reference generation module provides commands which are calculated based on a trapezoidal velocity profile defined by the acceleration, velocity, and distance of the requested move. Before adding the disturbance to the "motors", the "controllers" have been tuned to make the individual axes stable with minimum following error. The gains of the "controllers" are given in Table 7.2.

Table 7.2 Simulation Parameters

Simulation Parameter	With Fuzzy Logic Coupling	Without Fuzzy Logic Coupling
Position Control System Gains & Constants	k_p =58 k_i =30 k_d =1.5 k_v =20 k_f =15 k_m=10 k_{dc} =0.000019 k_e =159.135 T_m=0.0084 T_e =0.0021	k_p =58 k_i =30 k_d =1.5 k_v =20 k_f =15 k_m=10 k_{dc} =0.000019 k_e =159.135 T_m=0.0084 T_e =0.0021
Disturbance	D = 0.2 v t = 0.2 s to 0.3s	D = 0.2 v t = 0.2 s to 0.3s
Distance Speed Acceleration Deceleration	S_d = 5000 (counts) S_v = 10000 (counts/s) S_a = 200000 (counts/s^2) S_{ad} = 200000 (counts/s^2)	S_d = 5000 (counts) S_v = 10000 (counts/s) S_a = 200000 (counts/s^2) S_{ad} = 200000 (counts/s^2)

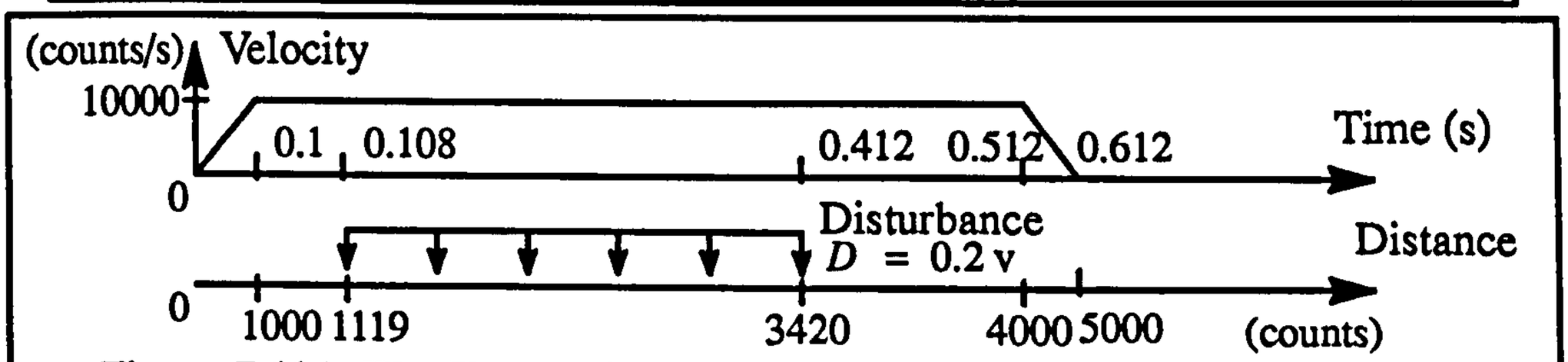
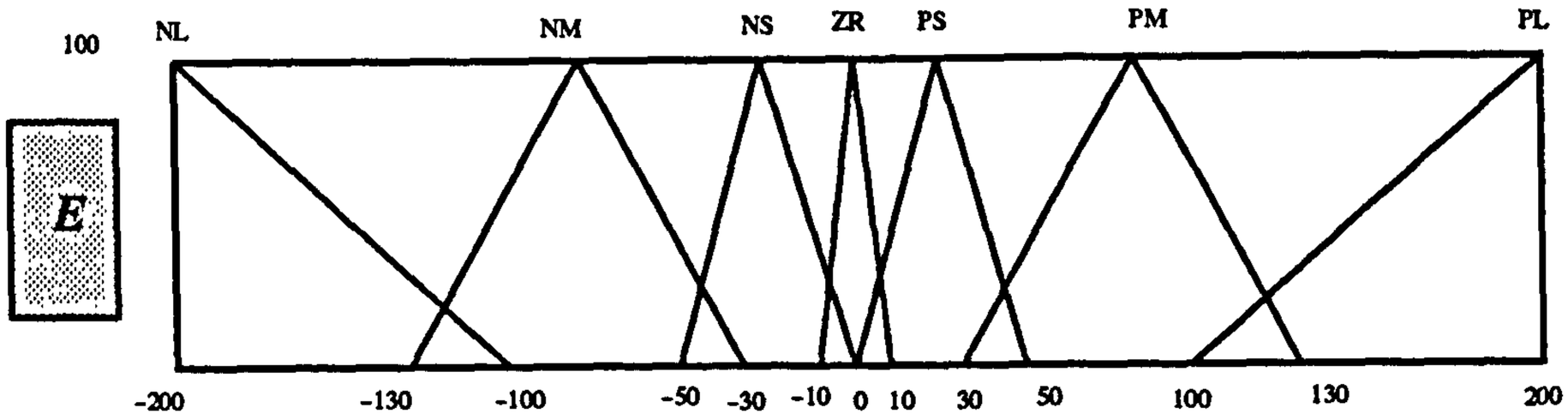
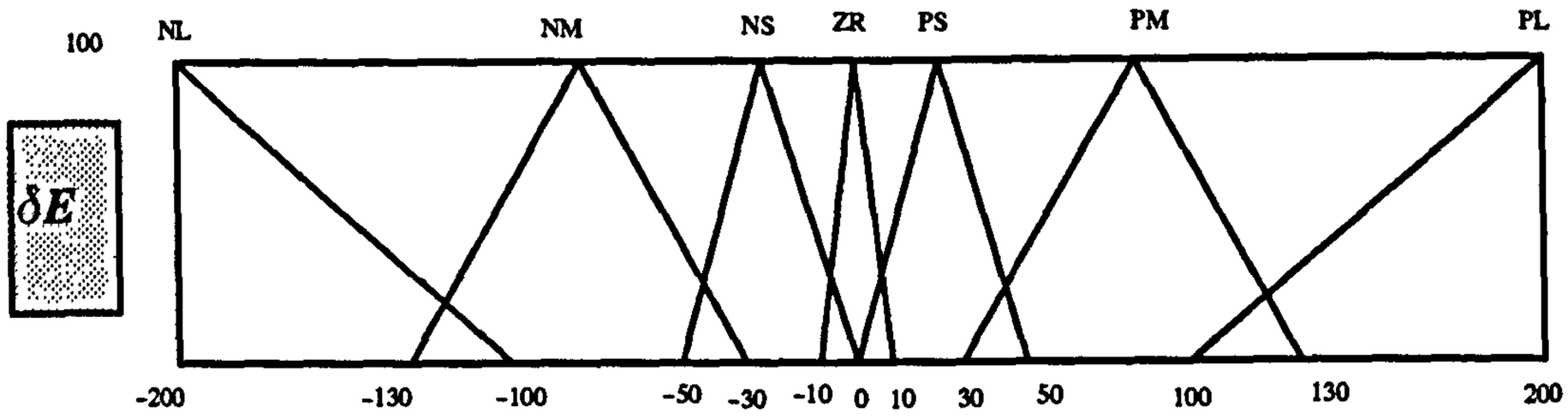


Figure 7.11A The Trapezoidal Velocity Profile and the Disturbance

In the motion control mechanism module, seven fuzzy subsets are selected for its input variables (synchronisation error $\overline{e}$ and the rate of the error change $\overline{\delta e}$) of the fuzzy coupling algorithm. The fuzzy subsets for the outputs (compensation terms for each axis, R_{1m} and R_{2m}) are selected as PML, PMM, PSL, PSM, PS, PVS, ZR, NVS, NS, NSM, NSL, NMM and NML. They form thirteen fuzzy subsets. Their values and dimensions are heuristically set and adjusted to keep the whole system stable and give high synchronisation accuracy. Figure 7.12 shows the fuzzy membership functions for the input–output variables. The fuzzy rule bases are set up using the principles developed in Chapter 4. Figure 7.13 gives the details of the fuzzy rule bases. The fuzzy logic coupling software mechanism uses the heuristic control "rules" which are based on quantised values of $\overline{e}$ and $\overline{\delta e}$. We do not quantise inputs in the classical sense that we assign each input an exact output level. Instead, each linguistic value equals a fuzzy set that overlaps with adjacent fuzzy sets. The fuzzy logic coupling mechanism uses triangular fuzzy–set values, as Figure 7.12 shows. The lengths of the upper and lower bases provide design parameters that we must calibrate for satisfactory performance. A good rule of thumb is adjacent fuzzy–set values should overlap approximately 25 percent[1].





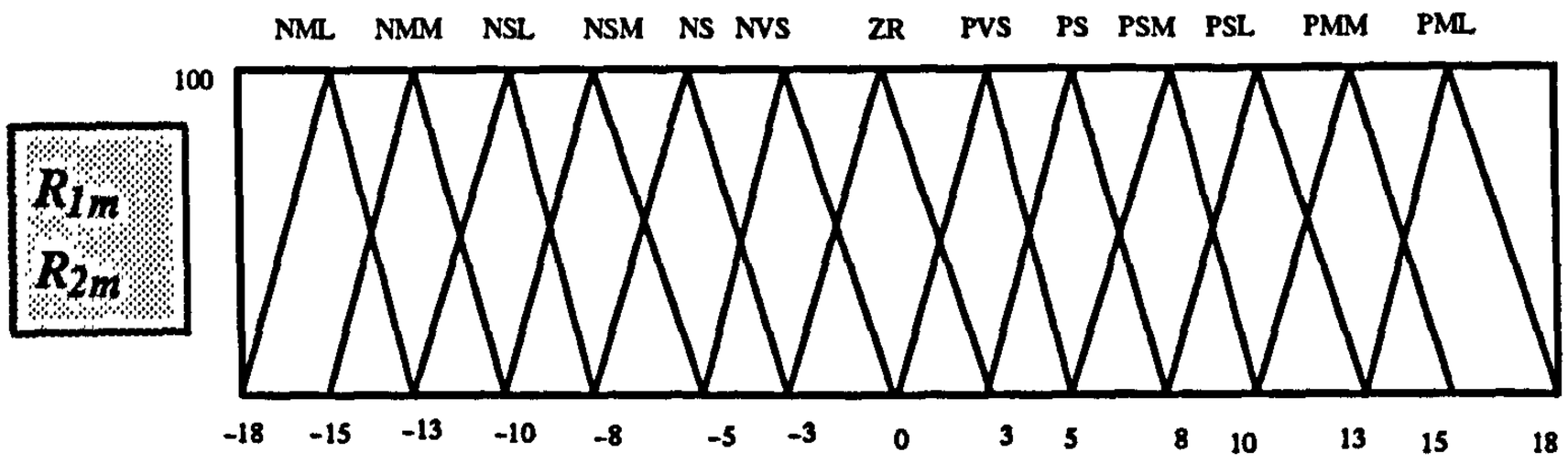


Figure 7.12 Fuzzy Membership Function for Input/Output Variables

$\dot{r}_{1m}$ δe / r_{2m} $\dot{e}$	NL	NM	NS	ZR	PS	PM	PL
NL	PML / NML	PMM / NMM	PSL / NSL	PSM / NSM	PSM / NSM	PS / NS	PS / NS
NM	PMM / NMM	PSM / NSM	PSM / NSM	PS / NS	PS / NS	PVS / NVS	PVS / NVS
NS	PSM / NSM	PS / NSM	PVS / NVS	PVS / NVS	PVS / NVS	ZR / ZR	ZR / ZR
ZR	PSM / NSM	PS / NS	PVS / NVS	ZR / ZR	NVS / PVS	NS / PS	NSM / PSM
PS	ZR / ZR	ZR / ZR	NVS / PVS	NVS / PVS	NVS / PVS	NS / PS	NSM / PSM
PM	NVS / PVS	NVS / PVS	NS / PS	NS / PS	NSM / PSM	NSL / PSL	NMM / PMM
PL	NS / PS	NS / PS	NSM / PSM	NSM / PSM	NSL / PSL	NMM / PMM	NML / PML

Figure 7.13 Fuzzy Rule Bases for Simulation Study

7.4.2 Discussion for Linear Relationship Simulations Results

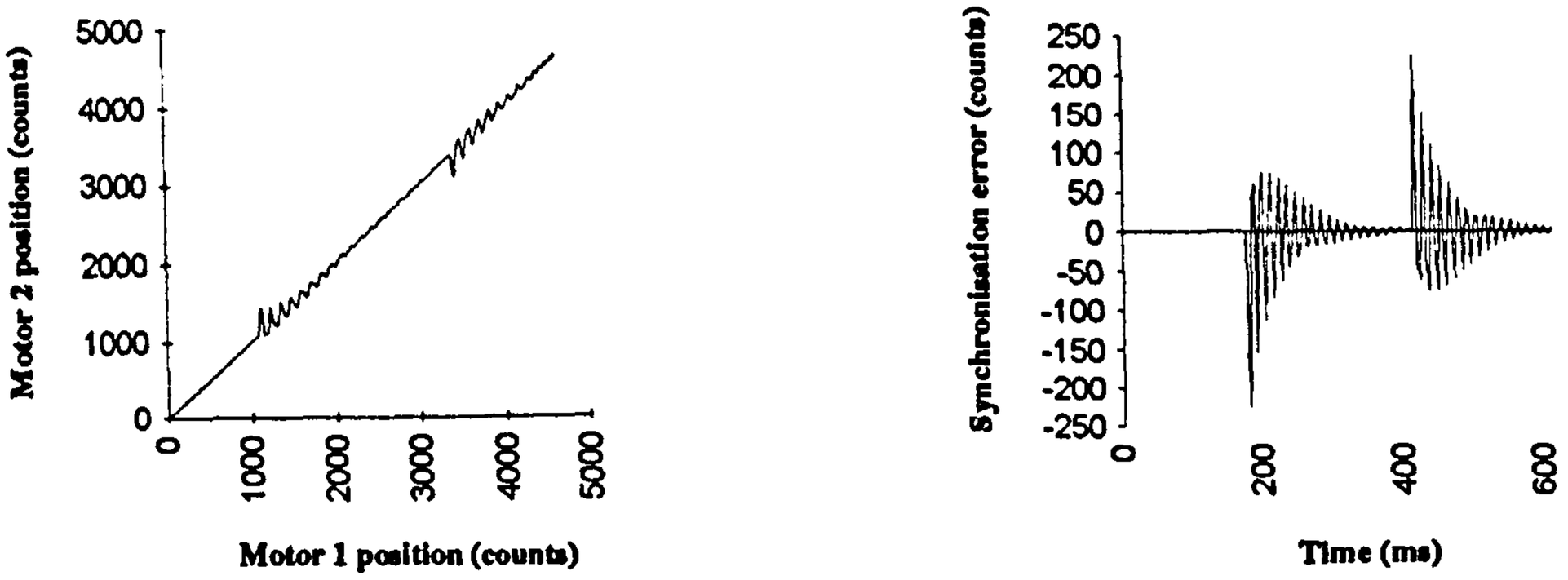
The plots in Figure 7.14 show simulation results for a 1:1 ratio 'gearbox' application. In order to clearly show the influence of the disturbance for the different types of control structure, a large constant disturbance is added to an axis. The suitability of the disturbance for these servo-systems was not a consideration. To which axis the disturbance is applied is arbitrary. In the simulation test, a constant disturbance was added to the axis 2 as shown in Figure 7.11A.

Figure 7.14a shows the case of two independent servo-drives. When a disturbance is added to an axis, a big position synchronisation error occurs. but the disturbed axis gradually reduces the following error caused by the disturbance, then a period the accurate synchronisation returns. The reason for this phenomenon is that even though there is no coupling among the axes, the PID control algorithms included in each axis have the ability to reject the external disturbance. However, in this case, the external disturbance is only 'seen' by the disturbed axis, the other axis does not 'know' what is happening in other parts of the system.

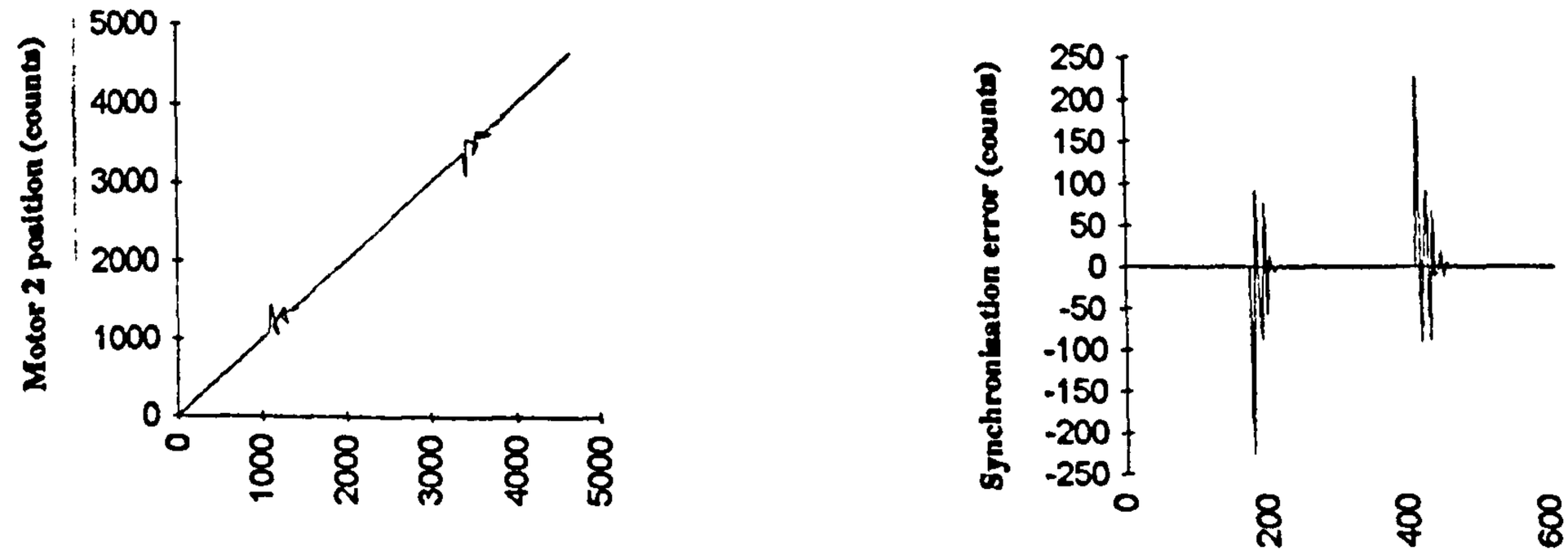
Figure 7.14b shows the response using fuzzy logic coupling in the equal-status structure. In this case, the synchronisation error caused by the disturbance decays very rapidly. When the fuzzy parameters are well tuned, better results can be expected. This will be shown in the following nonlinear case. With the fuzzy coupling mechanism, the disturbance results in a reaction on both axes to minimise the synchronisation error.

Figure 7.14c illustrates the performance of the closed-loop linked master-slave structure. When the slave has been disturbed, the coupling mechanism can still help the slave cope with the disturbance. The result shows that the synchronisation error is reduced much more quicker when compared with the uncoupled axes seeing the same disturbance as in Figure 7.14a.

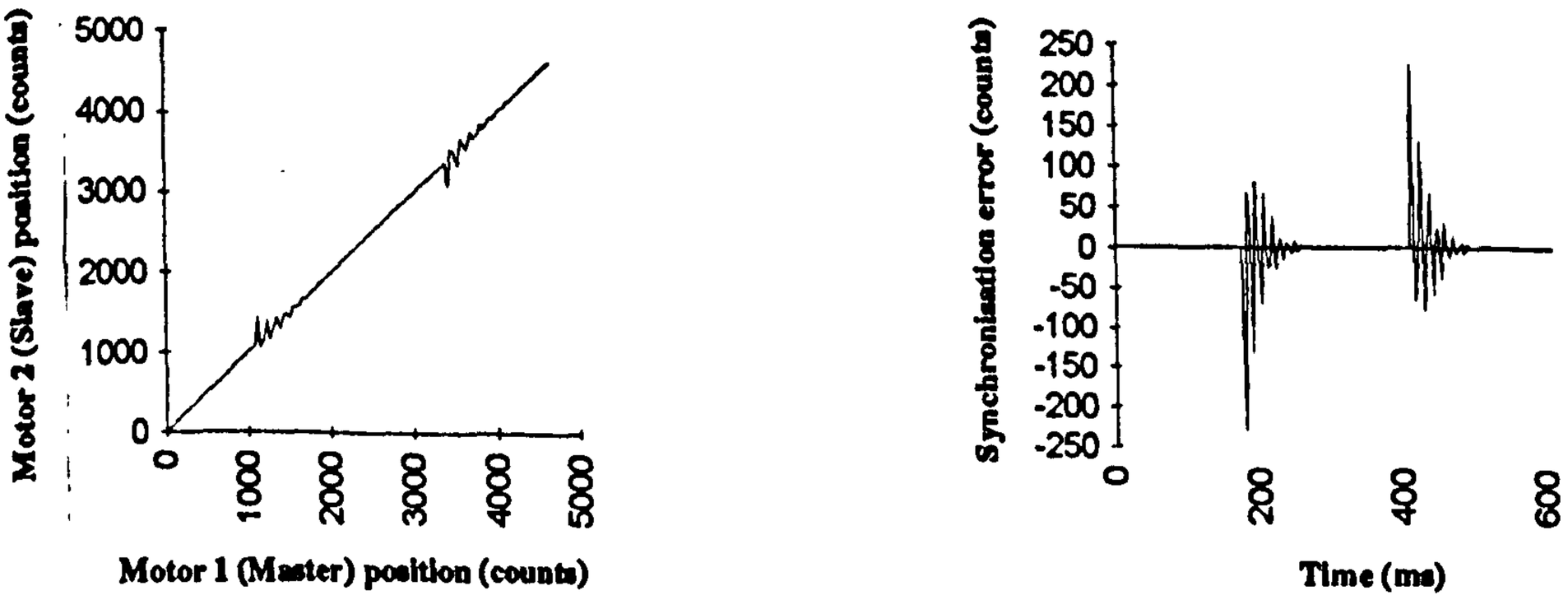
The reasons the synchronisation error can not be completely eliminated when using the fuzzy logic coupling mechanism are: 1) the disturbance is too large and when applied suddenly makes the axis reach its saturation state; 2) the fuzzy parameters and the rule bases needed to be optimized. Tools do not currently exist which can achieve this optimisation process.



(a) Independent Servo-drives



(b) Equal-Status Approach with Fuzzy Logic Coupling



(c) Closed-Loop Linked Master-Slave Approach

Figure 7.14 The simulation results of a linear 'gearbox' application

7.4.3 Test Simulations Using a Circular Path Reference

In order to let the joint point of the two axes' positions "travel" around in a radius R cycle path with a constant angular velocity ω, one axis will follow a sine function and the other axis will follow a cosine function. The positions of the two "motors" can be represented as:

$$m_1 = R \sin\omega t;$$
$$m_2 = R \cos\omega t, \qquad (7\text{–}11)$$

where t is the time.

The simulation is carried out in three phases. Firstly, both "motors" start to move from the original point (zero), then run at a same constant speed towards the circular path. Upon reaching the circular path the two "motors" follow the equations given in (7–11). After the joint point of the two axes' positions "travels" around the circular path twice, the "motors" move at a constant speed to return back to the original point (zero).

Since the same "motors" are used in the linear and nonlinear cases, the control parameters including fuzzy parameters set in the linear case can be used directly in the nonlinear case. Also, the fuzzy rule bases set in the linear case can be used in the nonlinear case. The parameters used in the nonlinear case are given in Table 7.3.

Table 7.3 Simulation parameters for circular path

Simulation Parameter		With Fuzzy Logic Coupling	Without Fuzzy Logic Coupling
Linear Movement	Distance	S_d = 5000 (counts)	S_d = 5000 (counts)
	Speed	S_v = 10000 (counts/s)	S_v = 10000 (counts/s)
	Acceleration	S_a = 200000 (counts/s^2)	S_a = 200000 (counts/s^2)
	Deceleration	S_{ad} = 200000 (counts/s^2)	S_{ad} = 200000 (counts/s^2)
Circular Movement	angular velocity ω	ω = 2.758 (radians/s)	ω = 2.758 (radians/s)
	radius R	R = 5000 (counts)	R = 5000 (counts)
Disturbance		D = 0.2 v t = 2.136 s to 2.14 s	D = 0.2 v t = 2.136 s to 2.14 s

7.4.4 Discussion for Circular Path Simulations Results

The plots in Figure 7.15 show simulation results for a circular path application. The disturbance was added to the axes in the same way with the same magnitude as with the linear case, but the disturbance period was very short, only lasting 4ms. An intermittent disturbance of this type presents most problems for synchronisation control. The simulation results of the linear case verify this viewpoint. As shown in Figure 7.14, the effect of the disturbance only happens at the time of start and end of the disturbance.

The simulation results shown in Figure 7.15 reproduce the phenomenon exhibited in the linear case. The fuzzy logic coupling algorithm demonstrates the ability to reject disturbances better, and has a shorter settling time. Figure 7.15c shows a response for a well tuned fuzzy logic coupling, this clearly illustrates that good synchronisation can be achieved even when disturbances are present.

7.5 Experimental Verification

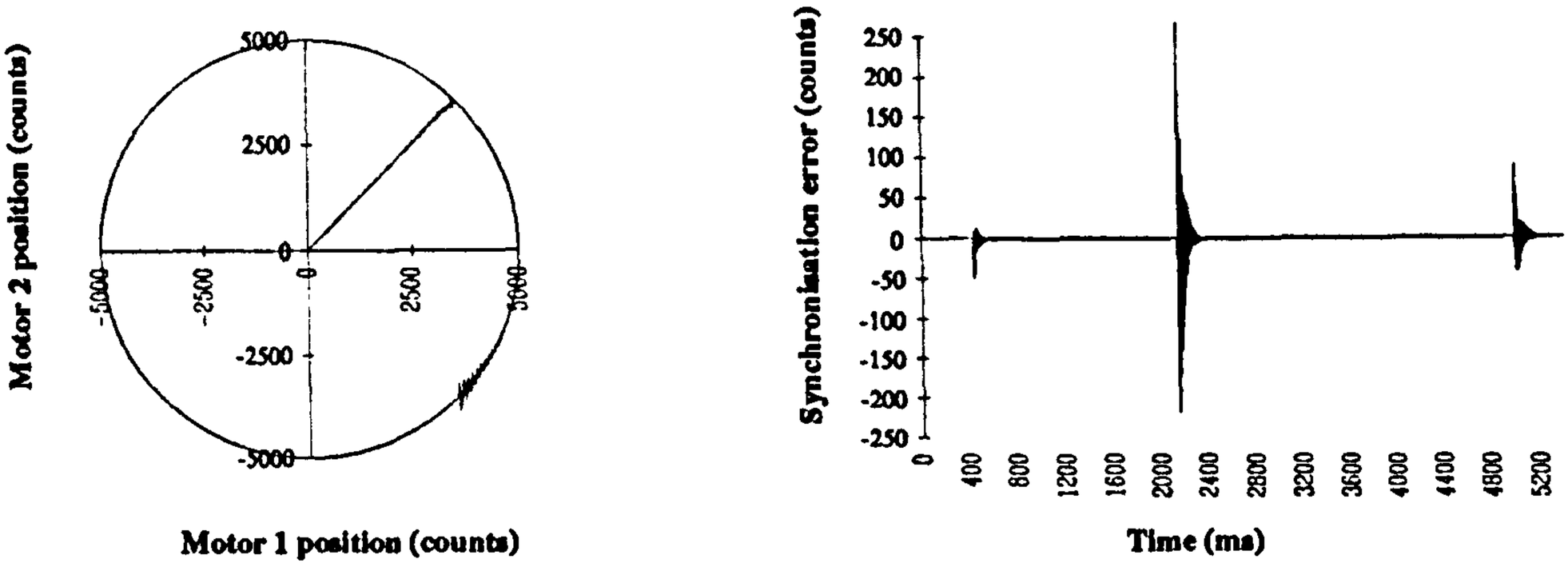
The experimental set-up, as shown in Figure 7.16, consists of two digital motor control systems (Quin Systems, DSC-1M), a low-loss pulse width modulated DC drive (Parker Digiplan Ltd, UD5), two DC servo motors (Electro-Craft, M540SA), and an OS-9 computer system (a Syntel VM022, with a 68020 processor running OS-9, together with a 68881 floating-point coprocessor). The software structure is shown in Figure 7.17, which is constructed from to the modules developed in Chapter 6.

The fuzzy logic parameters used in the simulation are also used for the experiments, with some minor modifications to account for the existence of modelling errors. The gains of the motion controllers were set to minimise the following error in each servo. Table 7.4 shows the gains set, Figure 7.18 gives the fuzzy parameters and Figure 7.19 shows the rule bases.

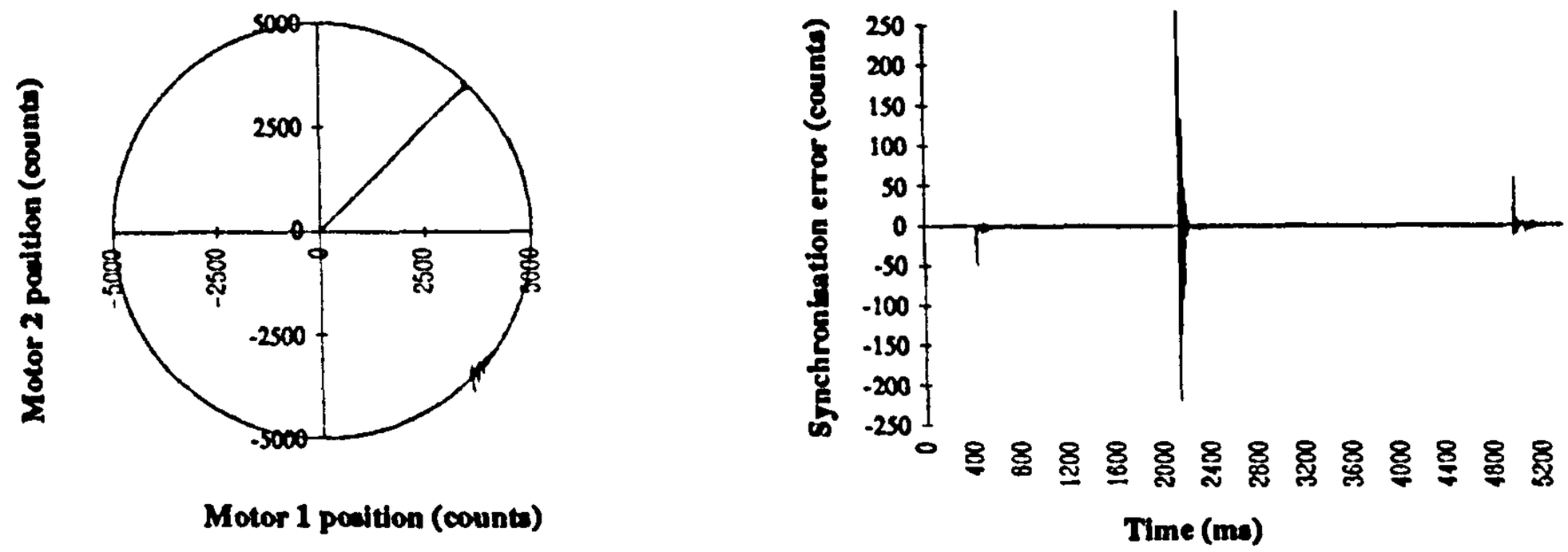
The following experiments were conducted:

(i) to reduce the synchronisation errors caused by the different characteristics of the two servo-systems; and

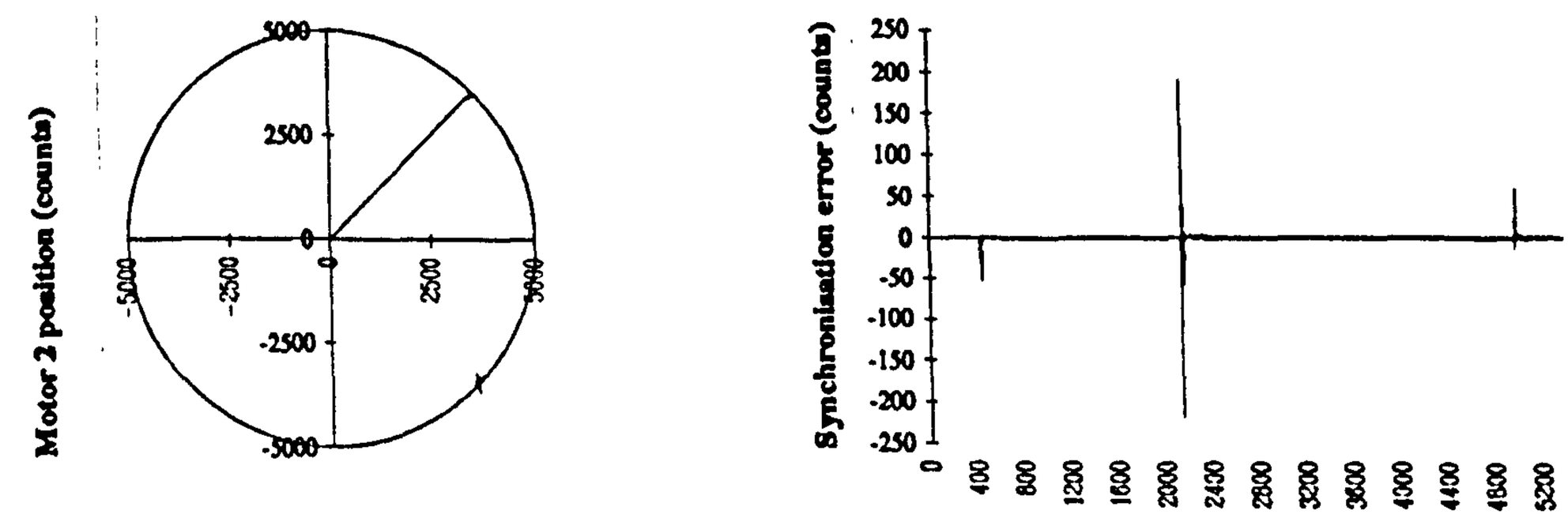
(ii) to reduce the synchronisation error caused by an external disturbance.



(a) Independent Servo-drives



(b) Fuzzy Logic Coupling Control



(c) Best Tuned Fuzzy Logic Coupling Control

Figure 7.15 The simulation results for a circular path application

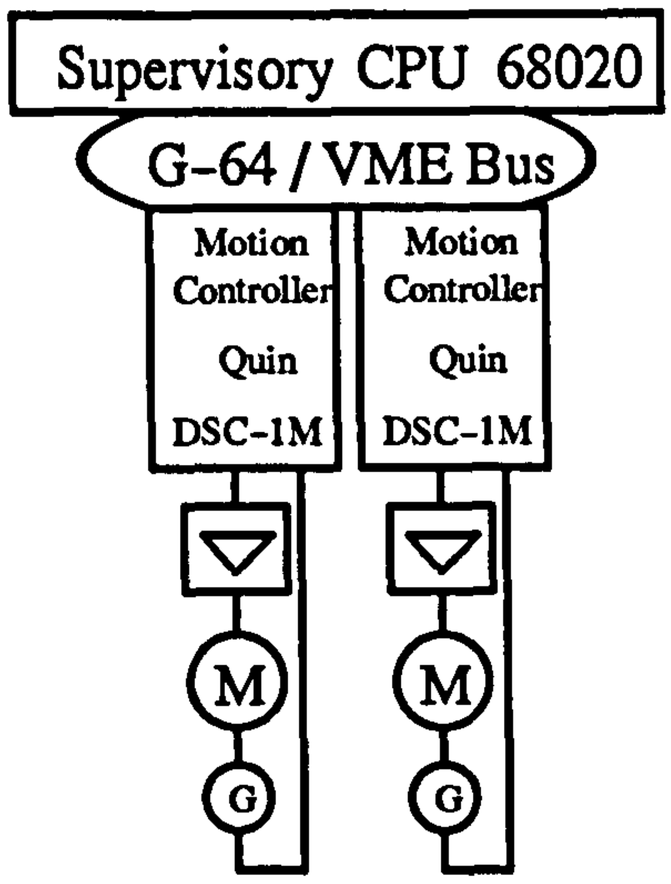


Figure 7.16 Hardware Configuration of the Testbed

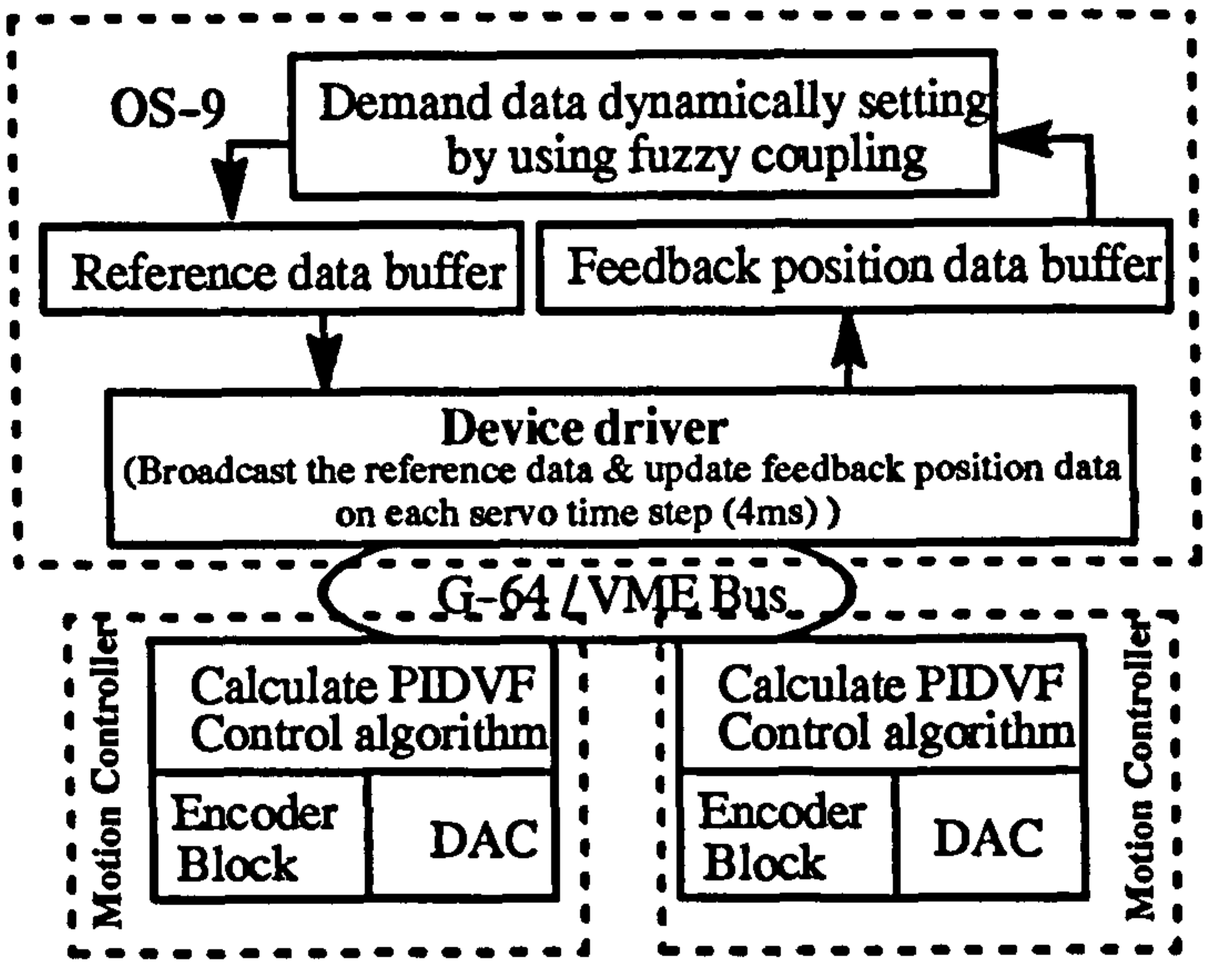


Figure 7.17 Software Structure of the Test Program

Table 7.4 Parameters for Experimental Verification

Parameters		With Fuzzy Logic Coupling	Without Fuzzy Logic Coupling
Position Control System Gains & Constants		motor 1: k_p =205 k_i =5 k_d =50 k_v =1200 k_f =1000	motor 1: k_p =205 k_i =5 k_d =50 k_v =1200 k_f =1000
		motor 2: k_p =180 k_i =10 k_d =50 k_v =1200 k_f =1200	motor 2: k_p =180 k_i =10 k_d =50 k_v =1200 k_f =1200
Linear Movement	Distance	S_d = 5000 (counts)	S_d = 5000 (counts)
	Speed	S_v = 10000 (counts/s)	S_v = 10000 (counts/s)
	Acceleration	S_a = 200000 (counts/s^2)	S_a = 200000 (counts/s^2)
	Deceleration	S_{ad} = 200000 (counts/s^2)	S_{ad} = 200000 (counts/s^2)
Circular Movement	angular velocity ω	ω = 2.758 (radians/s)	ω = 2.758 (radians/s)
	radius R	R = 5000 (counts)	R = 5000 (counts)

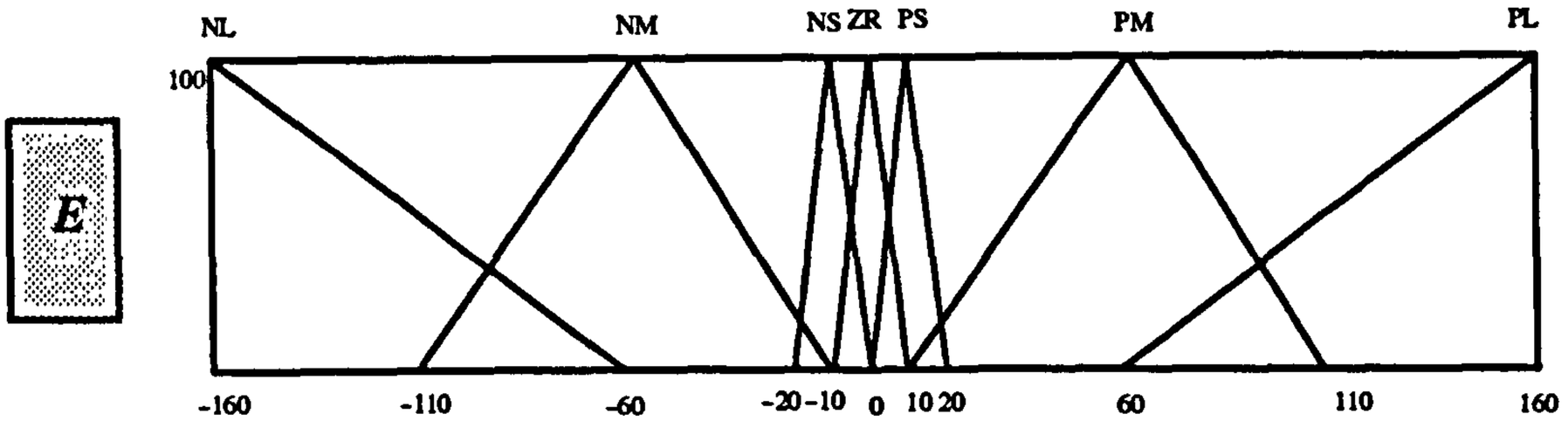


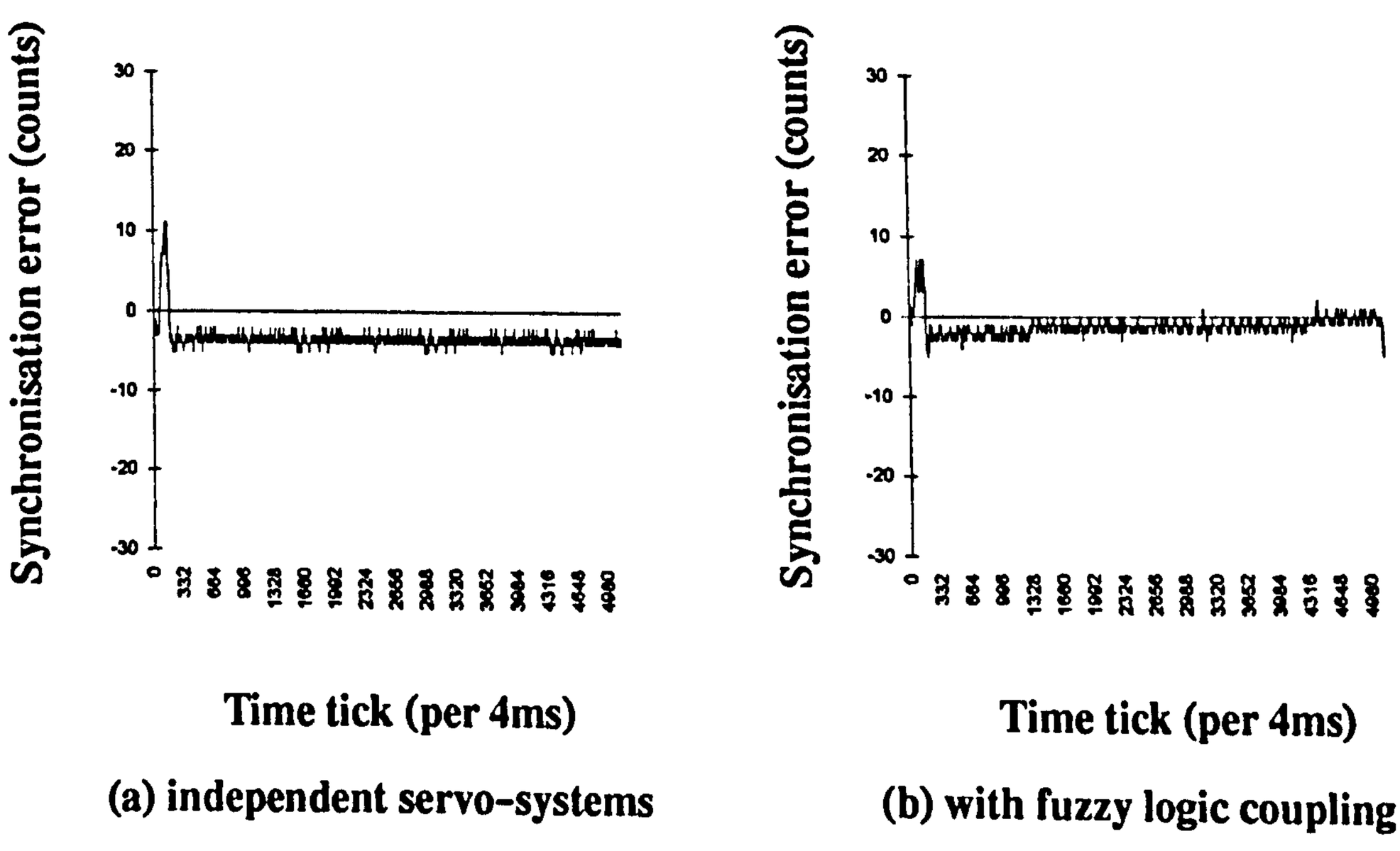
Figure 7.18 Fuzzy Membership Function for Input/Output Variables in the Experimental Studis

$\bar{e}$ \ $\bar{\delta e}$ / $\bar{r}_{1m}$ / $\bar{r}_{2m}$	NL	NM	NS	ZR	PS	PM	PL
NL	PML / NML	PMM / NMM	PSL / NSL	PSM / NSM	PSM / NSM	PS / NS	PS / NS
NM	PMM / NMM	PSM / NSM	PSM / NSM	PS / NS	PS / NS	PVS / NVS	PVS / NVS
NS	PSM / NSM	PS / NSM	PVS / NVS	PVS / NVS	PVS / NVS	ZR / ZR	ZR / ZR
ZR	PSM / NSM	PS / NS	PVS / NVS	ZR / ZR	NVS / PVS	NS / PS	NSM / PSM
PS	ZR / ZR	ZR / ZR	NVS / PVS	NVS / PVS	NVS / PVS	NS / PS	NSM / PSM
PM	NVS / PVS	NVS / PVS	NS / PS	NS / PS	NSM / PSM	NSL / PSL	NMM / PMM
PL	NS / PS	NS / PS	NSM / PSM	NSM / PSM	NSL / PSL	NMM / PMM	NML / PML

Figure 7.19 Fuzzy Rule Bases for Experimental Studies

(1) To reduce the synchronisation error caused by the different characteristics of the servo-systems

As shown in Table 7.4, different gains were set on the two motion controllers to minimise the position following error on each axis. However, when combining these servo-systems to construct a multi-axis system for synchronisation errors may be introduced. The synchronisation errors between the motors can result from the different dynamic characteristics of the servo-systems. The largest synchronisation errors occur during the acceleration phase, because of the differences in the dynamic response of the servo-systems. In the steady state phase, the following errors of the servo-systems gives an approximately constant position synchronisation error. Figure 7.20a shows one test result, with the two motors following the same trapezoidal velocity profile. This synchronisation error can be reduced using the fuzzy logic coupling mechanism. The test result shown in Figure 7.20b illustrates the response (the synchronisation error being significantly reduced).



(a) independent servo-systems **(b) with fuzzy logic coupling**

Figure 7.20 Experimental Results (the influence by the different characteristics of the servo-systems)

(2) To reduce the synchronisation error caused by external disturbances

As shown in the simulation section, the external disturbances will give rise to a synchronisation error and the fuzzy logic coupling algorithm will act to reduce the error to maintain tight synchronisation. Here, a similar approach was adopted to that taken

in the simulation tests. In order to investigate the effects of disturbances to the different control structures, a method to apply a repeatable disturbance to an axis was required. A simple method was used to introduce the disturbances by fixing a random shaped disk on the rotor of the motor and using a spring-driven roller or plunger to push the disk. Figure 7.21 shows the arrangement of this method. Different shaped projections gave

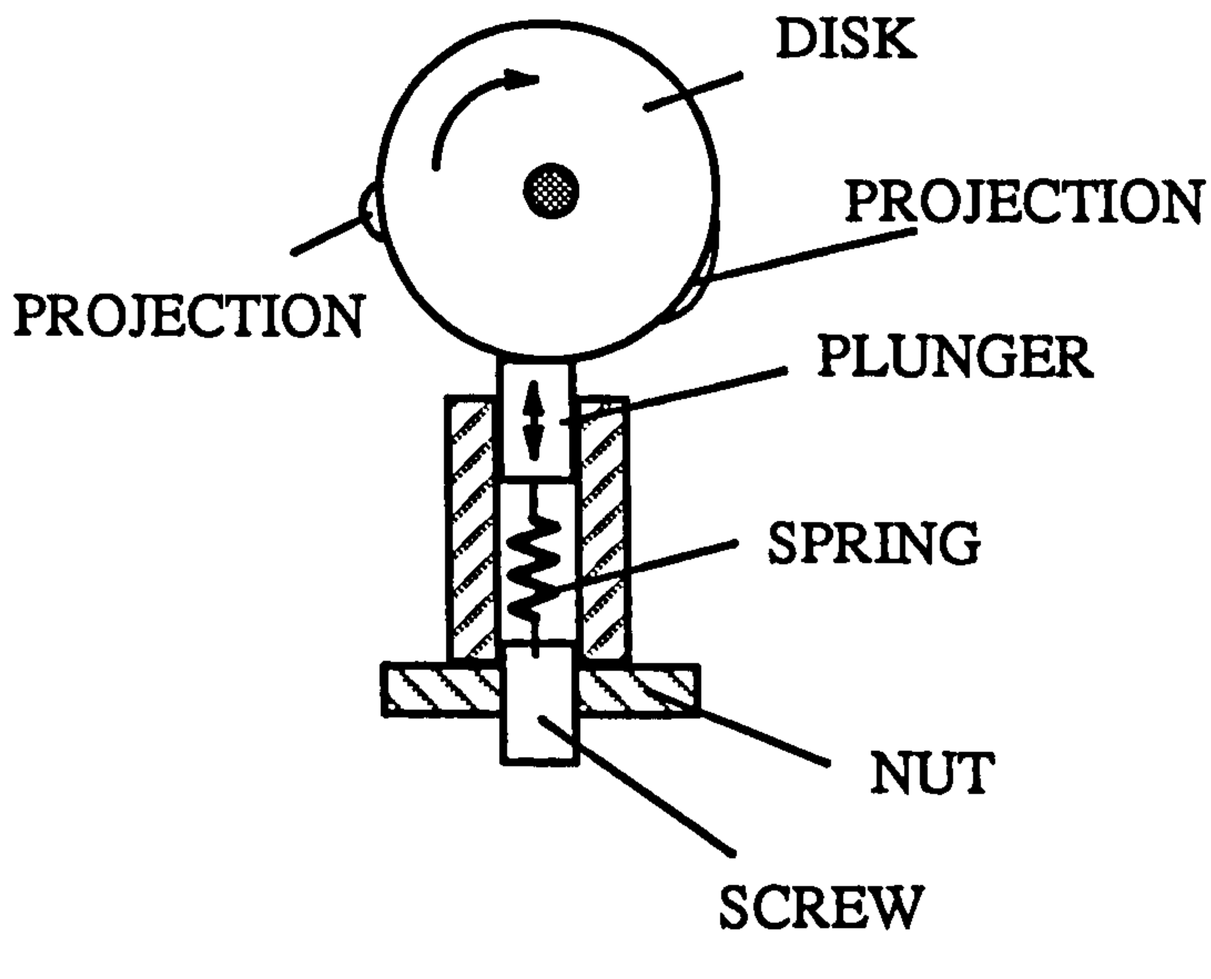


Figure 7.21 A Variable Loading Mechanism

different loadings or disturbances to the rotor. However, this method can only be used for the comparison of the performance of the systems under the same disturbance. It can not be used for the quantitative analysis for the effects of the disturbances, because the exact magnitude of the disturbance is not known.

Experimental results are shown in Figures 7.22 and 7.23 for two systems. Each Figure is split into parts (a) and (b). Part (a) shows the system response without disturbance, and Part (b) shows the effects of the disturbance. The same disturbance is applied for all tests.

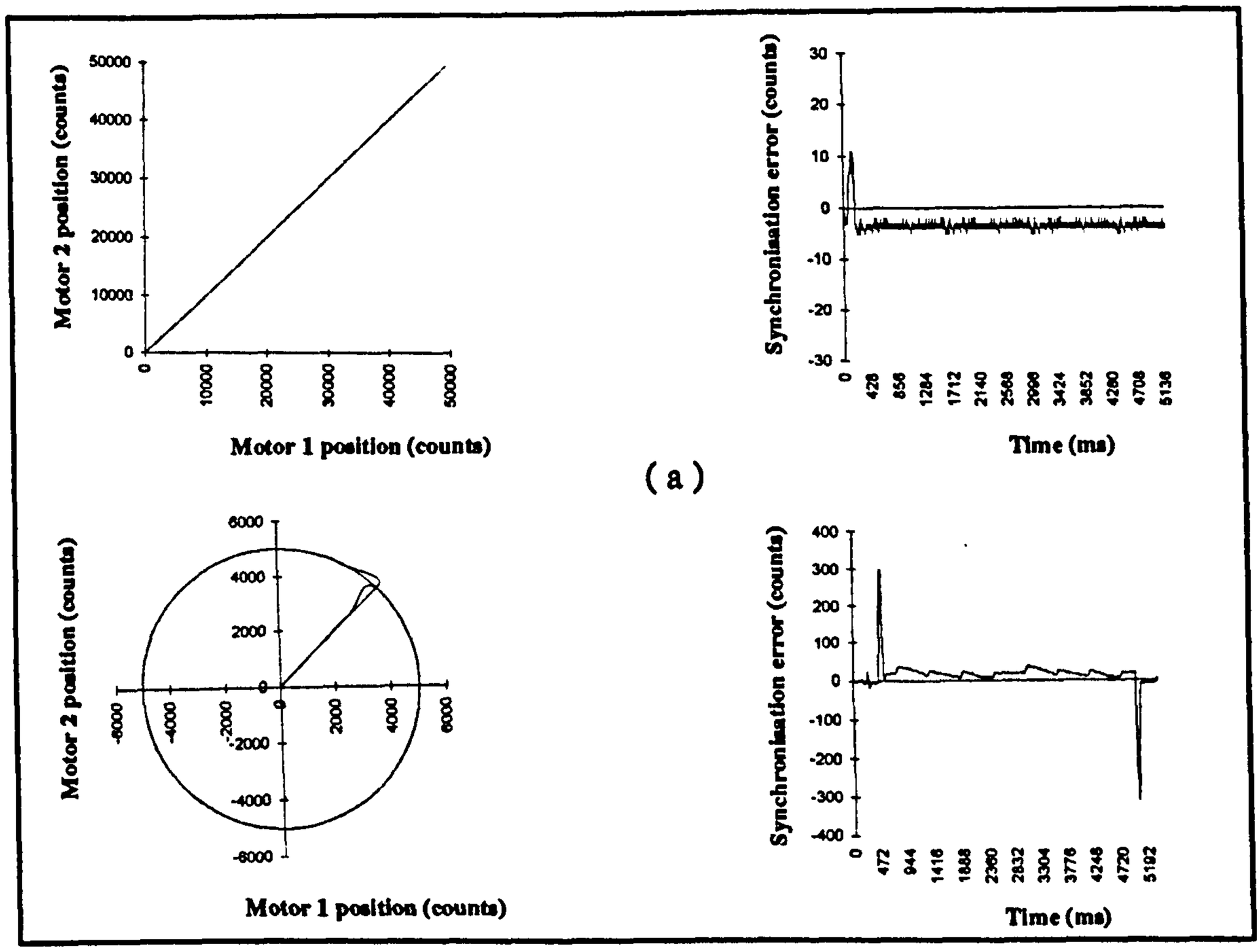


Figure 7.22 Experimental results using independent servo drives

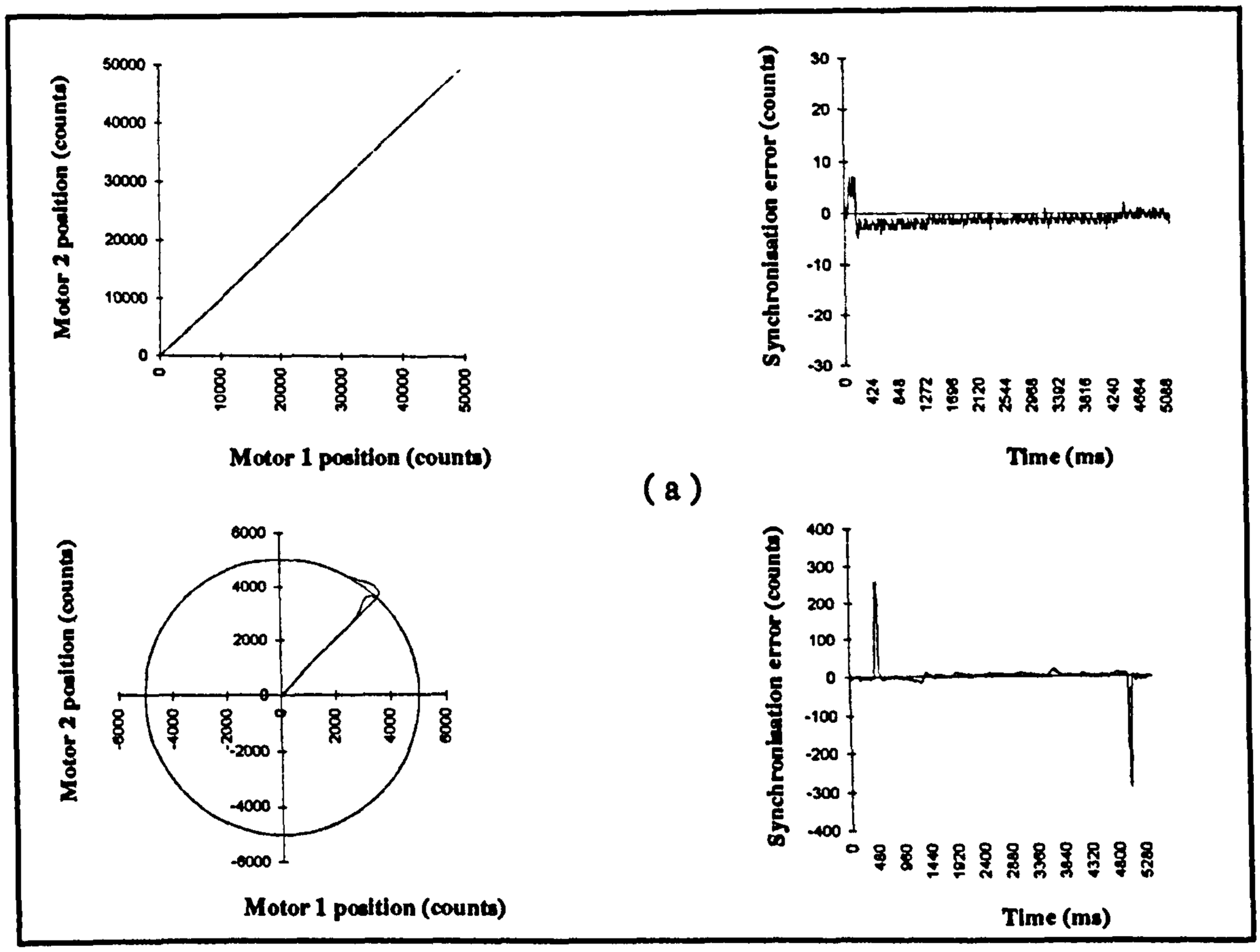


Figure 7.23 Experimental results using fuzzy logic coupling

7.5.1 Discussion of Experimental Results

Figure 7.22a

In the linear case, the result repeats the phenomenon shown in Figure 7.20a. In the nonlinear case (the circular path application), ripple synchronisation error occurred due to mismatched dynamics of the servo–systems. Similar phenomena were observed in [7], which also addressed mismatched dynamics. In addition, overshoot phenomenon occurrs at the ***corner*** where the straight–line motion transfers to the circular path motion, or from the circular path to the straight–line. It is quite a common situation in applications such as contouring[8]. In reference[8], a similar effect is observed when a PID controller is used for '***corner tracking***'.

Figure 7.22b

With the introduction of a disturbance to axis 2, the system performance is degraded, as expected. Large peaks of position synchronisation error are observed. For the straight–line motion, a large load was suddenly applied to axis 2 for a short period. The shape of the synchronisation error curve observed is very common, when an axis is under PID control[8]. A variable high friction disturbance was introduced to the circular motion by the loading mechanism shown in Figure 7.21. As can be seen in Figure 7.22b, the controller does not provide good immunity to disturbance. A similar shape of synchronisation error curve was observed in [8], when the axis was under P controller. Therefore, we may come to the conclusion that the DSC–1 controllers used for the axes were not well tuned, they acted only as P controllers. In reference[8], it was observed that a PID controller had a good ability in disturbance rejection.

Figure 7.23a

The results of Figure 7.23 are included here to highlight the differences between the two approaches. As seen from Table 7.4, the same control parameters were used in the two types of control. When using the fuzzy logic coupling control structure, better synchronisation can be obtained. The plots of Figure 7.23a shows a significant improvement over Figure 7.22a with both linear and nonlinear motion. Nevertheless, the ***corner tracking*** is still poor. Further investigation is needed.

Figure 7.23b

The plots of Figure 7.23b shows the synchronisation errors are significantly smaller when the fuzzy logic coupling was used. This means that the fuzzy logic coupling does enhance the ability to reject disturbance. However, the improvement was not as good as expected, since the unexpectedly highly computational load of this algorithm (especially for the nonlinear motion), the coupling loop sampling time was increased. Currently, the sample time of the coupling loop is 8ms which is twice than that of axis servo loop. Therefore, more efficient hardware and software tools are required to implement and run this algorithm in order to reduce the execution time (ie shorten the sample time). This will be further discussed in the next chapter.

7.6 Conclusions

The software mechanism of IMC, namely the fuzzy logic coupling algorithm which has been developed in the previous chapters, has been verified by the simulation and experimental studies in this chapter. Simulations involving models of these servo-drives have been implemented, which verify the theory of earlier chapters. The performance of the linear and nonlinear motion synchronisation of two DC motors has been tested under normal and disturbance conditions. The results obtained were consistent with the theory and simulations.

Chapter Seven: References

[1] Bart Kosko, "Neural Networks and Fuzzy Systems", Prentice–Hall Inc, 1992.

[2] Tal, J., "Motion Control by Microprocessor", Naples Controls Ltd, 1984.

[3] Olsson, G. and Piani, G., "Computer Systems for Automation and Control", Prentice Hall, 1992.

[4] Bollinger, J. G. and Duffie, N. A., "Computer Control of Machines and Processes", Addison–Wesley Publishing Company, 1988.

[5] Phillips, C. L. and Harbor, R. D., "Feedback Control Systems", Prentice–Hall International Editions, 1991.

[6] Electro–Craft Ltd, UK, "Short Form Product Guide", April 1989.

[7] Jenkinson M., "The synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

[8] Lo, C. C. "Cross–Coupling Control of Multi–axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

CHAPTER EIGHT
Intelligent Motion Control (IMC) Evaluation and Future Extensions

8.1 Introduction

A considerable amount of software has been written in order to enable the concepts of IMC to be tested. The fuzzy logic coupling mechanism, the core element of IMC, has been verified in the previous chapter. This chapter considers the capabilities of the current implementation of IMC, its perceived advantages, limitations, and possible routes to its exploitation.

8.2 The Advantages of the IMC Method

The proposed IMC method is expected to solve some of the deficiencies of the existing methods for multi-axis motion control and synchronisation. Apart from the proposed fuzzy logic coupling algorithm which can provide accurate synchronisation for multi-axis motion control systems, the IMC approach enables the multi-axis system designer to use the state of art single axis controllers for each individual axis, then integrate the axes as a co-ordinated multi-axis system in an intelligent way. The advantages of IMC as compared to the existing methods can be summarised as follows:

- The proposed fuzzy logic coupling algorithm can provide accurate synchronisation for many kinds of multi-axis motion control systems.
- Because human intelligence in coordination can be directly incorporated and fuzzy control has a parallel processing structure, the fuzzy logic rule based approach allows a complex motion synchronisation to be created simply.
- Since the fuzzy logic based software motion synchronisation mechanism can couple any types of servo-drive, it is possible to apply an appropriate control algorithm for each axis which matches the axis dynamic characteristic and assigned task.

- With the fuzzy logic software mechanism, the bus supported motion controllers can be directly used for multi-axis systems where tight motion synchronisation is required. This makes the multi-axis system design very efficient compared to completely designing a conventional MIMO controller.

- IMC methods provide guide lines for designing multi-axis systems for a variety of motion synchronisation requirements, from position synchronisation to web synchronisation.

- IMC control structures support machine control architectures (such as UMC and MOSAIC[10]) to have tight motion synchronisation capability at their task level.

8.3 The Limitations of the Current Implementation of the IMC Method

The IMC method is based on the introduction of a 'dual closed-loop' control structure for multi-axis systems and the development of the fuzzy logic coupling algorithm. The limitations of the current implementation of the IMC method can be categorised in the two groups.

The 'dual closed-loop' control structure requires the axes state information fedback to the higher level (or host processor). The current implementation of IMC requires the motion controller to return to the host processor various data values continuously, at each servo time step. The main limitation of this technique is that a large data buffer is needed to hold all the returned data. If there are more axes, a large storage is required in the supervisory computer. In addition, this configuration requires a very efficient '*handshaking*' between the motion processors and host processor in order to shorten the communication time.

Generally, the fuzzy logic coupling compensations need to be applied to the axes at every servo cycle. Using a high-level language (such as the C language) to write the fuzzy logic programs and running them on conventional microcomputers can result in long execution times when nonlinear motion synchronisation is involved. It may be the case that a combination of assembly language and a high speed processor or special

hardware, such as DSP[11][12] or fuzzy logic processors[6][13], are required to realise optimised loop closure rates.

Fuzzifying the control variables and setting the FAM rules are the major tasks involved with this approach. They determine the system's performance (accuracy and stability). It is quite a difficult and time consuming process to tune manually. Therefore, a method for optimising the fuzzification and rule base setting are needed.

8.4 Future Extensions

The basic premise of the IMC method is that many multi-axis motion control applications have motion synchronisation requirements which can be effectively catered for by an intelligent integrated approach. IMC currently enables the implementation of multi-axis motion control systems in a intelligent manner. In a broader context the IMC method could be naturally extended to create a framework capable of supporting an intelligent approach to machine design. The suggestions listed below provide some possible directions for future work which could be carried out to extend the existing system boundaries.

8.4.1 Parameter Tuning of the Fuzzy Logic Coupling Algorithm

The performance of the fuzzy logic coupling algorithm is partially dependent on the fuzzy parameter tuning. The fuzzy parameter tuning involves: (i) modifying the span of the membership functions which affect the sensitivity of the fuzzy system to process noise[1]; (ii) determining a term set which is at the right level of granularity for describing the values of each (linguistic) variable; (iii) selecting the type of fuzzy variable, (such as monotonic, triangular, trapezoidal, and bell-shaped.), which will affect the type of reasoning to be performed by the inference mechanism[2]; and (iv) adjusting the normalizing gains.

So far, very few methods have been developed for tuning the fuzzy system and no method can handle all the fuzzy parameter tuning[3]. Therefore, more general studies are needed to develop a parameter tuning method for the fuzzy logic coupling algorithm.

8.4.2 The Generation of the Fuzzy Logic Coupling Rules

The approach used in Chapter 4 for the derivation of the fuzzy logic coupling rules is a heuristic method. In this method, a collection of fuzzy control rules is formed by analysing the behaviour associated with synchronisation of multiple axes. The control rules are derived in such a way that the deviation from the desired state can be corrected and the control objective (motion synchronisation) can be achieved. The derivation is purely heuristic in nature and relies on the qualitative knowledge of the system characteristics.

It has been shown in Chapter 4 that the control characteristics of the fuzzy logic coupling algorithm are similar to those of the scalar field control method. Therefore, the physical analogy method which is used in generating a scalar field to introduce coupling into decoupled systems, such as *a ball on a surface*[14], can be applied to generate the fuzzy logic coupling rules. Some of the ideas are shown as follows:

(1) We use the scalar field control concept to derive the first 7 rules (ie for the condition $\delta e = 0$) and obtain a parabolic curve against the synchronisation error e, as shown in Figure 4.21.

(2) Imagine a "ball" is at the bottom of the curve (ie when $e = 0$). In this case, in order to keep the "ball" stationary we have to provide "***force***" to counteract the "***kinetic energy***" when $\delta e \neq 0$. Because of '***energy equivalence***', we can use similar rules to generate the "***force***" to hold back the "ball". Based on these conditions, we obtain another 7 rules (ie when $e = 0$).

(3) When the "ball" is on the parabolic curve with "speed δe ", if δe's direction is toward the bottom 0, it may help the "ball" to go down to the bottom. Therefore, we can reduce some '***potential energy***' requirements. However, if δe is too large, even if we remove all the '***potential energy***' at the point, the "ball" will overshoot the bottom, then a counteracting force should be added. When δe's direction is away from the bottom 0, it will resist the "ball" to go down to the bottom. An '***extra–force***' is needed to push the "ball" down to the bottom. Appropriate rules are readily derived using such an analogy.

8.4.3 Intermittent Motion Synchronisation

The IMC method can also embed intermittent motion synchronisation in which two or more coordinated servo-drives are brought into synchronisation at specific spatial positions or points in time. The problem of intermittent synchronisation is somewhat different and involves enforcing the synchronisation, as a discrete event, upon asynchronous controlled servo-drives[7]. For intermittent motion synchronisation, conventionally the on-line synchronisation error correction is directly added to the measured synchronisation error values to form the servo-controllers command. Since some error sources exist in the system the response to the reference may be imprecise. As such the on-line corrective action from the synchronisation error has to be added in a compensation term which includes the correct reasoning for coping with these errors. However, since very little data about the error sources is available in complex applications, to express this control action using conventional control expressions (equations or logical expressions) is very difficult, if not impossible. The proposed fuzzy logic coupling approach provides a framework to implement the control reasoning for such corrective action.

The ability of motion control systems to execute a pre-programmed motion profile in response to a signal from an external sensor, even though the motor is already moving, is vital to many of today's high-throughput manufacturing processes--particularly those involving web materials. Known as dynamic registration, this control technique is one of the most powerful ways of synchronising plant machinery with the material that is being processed[8].

Now, we look at a typical intermittent motion synchronisation example used in packaging, flying shears and other cyclic cutting applications. Figure 8.1 shows one of these applications[9]. In this example, the Master-slave axes have a Cam path position relationship as shown in Figure 8.2 and the motion is cyclic. That is, the slave follows the same trajectory periodically. A cycle is programmed as the master's position displacement (e.g., one turn of the shaft in a rotary motion or 2 inches of a linear move). To provide an accurate positioning of the knife, a registration sensor measures the marks imprinted on the web. Ideally, knife displacement in one cycle (measured by the knife's marker pulse) must be equal to the distance between adjacent registration marks. However, in an actual application, this may not always be the case. The difference,

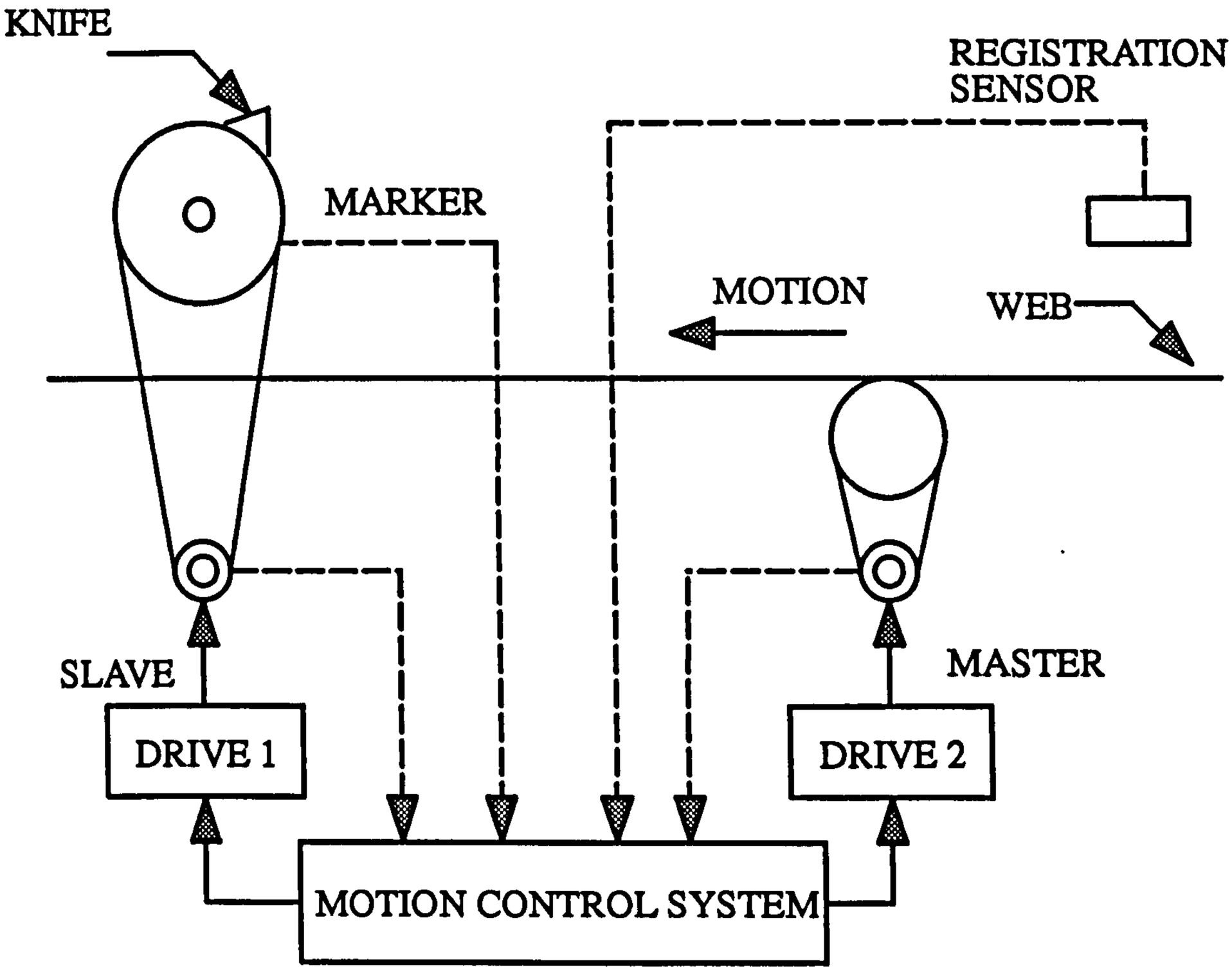


Figure 8.1 Synchronous Cutter Application. Source: Meshcat.

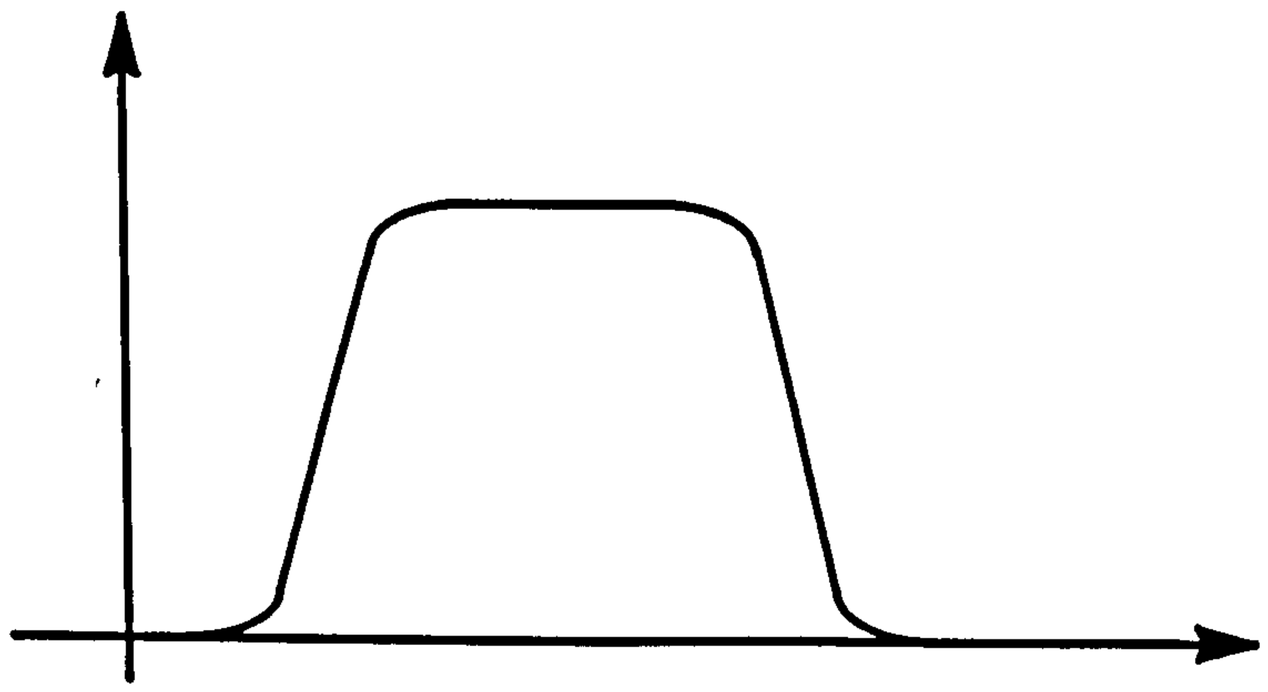


Figure 8.2 An Example Cam Path

defined as the cutting error, is measured as:

Cutting Error = Knife Marker position – Registration Marker Position.

Conventionally the cutting error is directly added onto the servo-controllers commands, in order to maintain the synchronisation. As the afore-mentioned analysis, this approach may not result in precise synchronisation. In order to reduce the synchronisation error, a compensation term has to provide a corrective action towards the ideal zero cutting error. The compensation term can be obtained by a fuzzy rule based control scheme similar to the ***fuzzy rule base for closed loop master–slave linked motion synchronisation*** method described in Chapter 4.

8.4.4 Different Structures of the Fuzzy Logic Coupling

The structure of the proposed fuzzy logic coupling in Chapter 4 can be defined as a *modifying with fuzzy logic coupling*. In this structure, the compensations are added to the local controllers' inputs (reference command). The direct modification of the controllers' reference command is a straightforward way which does not involve changing the system configuration. Another structure of fuzzy logic coupling can be termed *parallel fuzzy logic coupling*. This coupling structure adds the compensations to the local controllers' outputs (control values). The fuzzy logic coupling algorithm provides direct control of the synchronisation error. This may give more effective synchronisation control. However, adding a control value to the axis controller is not easy. This structure is suitable to implement in a multi-axis controller where the addition of control values can be done through software.

8.4.5 Motion Synchronisation Incorporating Process Information

In order to cope with environmental variation and system nonlinearities, adaptive control is often introduced. The function of adaptive control is to sense manufacturing conditions and adjust the motions accordingly. However, some uncertainness of the manufacturing process and inaccuracies of the system modelling can reduce the efficiency of the adaptive method, since the approach relies on an attempt to formulate the input-output relationship by means of mathematical models, which may be difficult

in many cases. When process information can be described in linguistic terms or the operational knowledge of the processes is available, this information can be incorporated into the proposed fuzzy logic algorithm to enhance synchronisation performance.

8.4.6 Extending the Number of Axes of the Systems

Thus far, only 2-axis systems have been used to illustrate the ideas behind the new motion synchronisation control algorithm. However, a motion control system may involve more than two axes or motions which require tight synchronisation. In manufacturing systems, more than two axes may be involved, generally each 'closely coupled' group may only have one master with several slaves, or two or more 'closely coupled' masters and each master having several slaves, or several axes with equal status. For these multi-axis motion control systems, the following issues will occur in the proposed fuzzy logic coupling algorithm.

(1) The motion synchronisation error model

For multi-axis systems which have two or more axes with an equal status or in the master status, the motion synchronisation error model may include a *state synchronisation error* and an *orientation error*[4][5]. The *state synchronisation error* can be a position or speed synchronisation error which depends on the application requirements. The term *orientation error* is used to denote the angular error between the actual orientation and the required orientation in respect to the relationship path. The *orientation error* will contribute to the motion synchronisation error. Therefore, the elimination of the *orientation error* as well as the elimination of the *state synchronisation error* should be chosen as the two control objectives of the multi-axis fuzzy logic coupling algorithm.

In applications where a master axis has several slave axes, each slave axis should have a synchronisation error model relative to the master axis. Therefore, there are several synchronisation error models in the system.

(2) The fuzzy rule base for coupling the axes

Fuzzy control is based on the concept that complex processes are nothing more than a collection of simple processes. As such, complex control processes can be broken down into simple, independent parts[6]. The inference process used in fuzzy control is composed of several rule processes and a single logic sum. As described in Chapter 4, the rule processes are divided into conditions (the antecedent block) and a conclusion (the consequent block). The logic sum is arrived at when a defuzzifier operation converts the results of the rule processes into a single, fixed value. Each rule process is based on different inputs and is largely independent; each independent sum affects only the logical sum. While the individual rules are simple, the aggregation of them allows complex control to be strictly executed. These processes culminate in a defuzzifier operation which unifies the results, and calculates a final value which is then output to each device. This reasoning process is called a parallel processing structure, and it provides many benefits for controlling complex control systems enabling the system to be created simply, but operating efficiently at high speed and with high reliability[6].

In a n-axis system, the fuzzy logic coupling algorithm will have n fuzzy coupling rule bases, which means that each axis has its own rule base. The rule base can be generated by consideration of actions which help eliminate the synchronisation error of the system, based on the synchronisation error models. Because each rule process is separate, each rule within the rule bases can be set and tested individually. Therefore, the complete fuzzy coupling rule base can easily be established. This separate processing of the fuzzy logic coupling algorithm produces an attractive advantage over the conventional coupling approaches when dealing with complex multi-axis motion control systems.

The proposed approach when applied to multi-axis systems, however, will increase the computational load. Nevertheless, with the widely available increase in computational speed of microprocessors, this is no longer seen as a major limitation.

8.5 Summary

Based on the requirements identified in sections 1.2.3 and 4.2.5, the current implementation of IMC demonstrates a number of significant advantages over

conventional motion synchronisation methods for multi-axis motion control systems; particularly with regard to the ability to intelligently integrate different control algorithms or controllers to achieve tight motion synchronisation. Some limitations in the current implementation of the IMC do exist mainly due to constraints imposed by the test platform available. The further development of the IMC method has also been broadly discussed.

Chapter Eight: References

[1] Braae, M., and Rutherford, D. A., "Selection of Parameters for a Fuzzy Logic Controller", Fuzzy Sets Syst., Vol. 2, 1979, p185.

[2] Berenji, H. R., "Fuzzy Logic Controllers. Introduction to Fuzzy Logic Applications in Intelligent Systems", Zadeh, L. A., and Yager, R. eds., pp69–96, Kluwer Academic Publishers, 1991.

[3] Ling, C. and Edgar, T. F., "The Tuning of Fuzzy Heuristic Controllers", Asian Pacific Eng. J., 1993.

[4] Feng, L., Koren, Y., and Borenstein, J., "Cross–Coupling Motion Controller for Mobile Robots", IEEE Control systems, December 1993, PP 35–43.

[5] Lo, C. C. "Cross–Coupling Control of Multi–axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

[6] "Fuzzy Guide Book", Cat. No. P30–E1–2, 1992.

[7] Draper, C. M., Holding, D. J., "The specification and fast prototyping of a distributed real–time computer control system for a modular independently driven high–speed machine", 2nd International Conference on Software Engineering for Real Time Systems, September 1989, pp 199–203.

[8] Scott, R., "Axis Synchronization by Encoder Following", Drives and Controls Magazine, September 1991, pp 70–72.

[9] Meshkat, S., "Parallel DSPs Excel in CAM and Gearing Applications", PCIM, February 1994, pp69–73.

[10] Esprit II Project 5292: MOSAIC, "First complete OMC–Specification V1.0", Document No: AR 3.4.3.1, December 1991.

[11] Meshkat, S., "Digital signal Processors Provide Advanced Motion Control", Motion Control, PCIM, December 1987, pp42–75.

[12] Meshkat, S., "Quantifying System Performance vs DSP MIPS – It's About Time", PCIM, May 1994, pp34–47.

[13] "Fuzzy MicroController", American NeuraLogix, Inc., August 1991.

[14] Jenkinson M., "The synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

CHAPTER NINE
Conclusions

In pursuit of the global aim of creating and demonstrating improved multi-axis motion control and synchronisation methods for machine control systems with multiple independent drives as defined in the original research objectives (Section 1.3, pages 8 and 9), has involved many facets of study. In particular the author has:

(i) Identified the requirements for multi-axis motion synchronisation;

(ii) Surveyed the features of 'existing' software based motion synchronisation and coordination mechanisms;

(iii) Proposed and implemented a novel new 'dual-loop' control scheme for intelligent multi-axis motion control based on a fuzzy logic approach;

(iv) Proposed and implemented a new framework in which to design and build software based intelligent multi-axis motion control and synchronisation in machine systems, namely 'Intelligent Motion Control' (IMC);

(v) Validated the IMC approach through selective simulation and experimental studies.

The key conclusions from these studies, which have been considered in detail within the main body of the thesis, are presented in the following section.

9.1 Summary of Work Undertaken

There is a current trend in manufacturing industry towards greater automation and computer integration, with a view to improving productivity and competetiveness, through improved performance and greater flexibility. Increasingly, manufacturers are seeing the advantages of using more flexible machines in their production processes, making shorter production runs cost effective so that feature variants and product variants can be accommodated. Therefore, traditional machines, typically comprising a control prime mover and mechanical transmission systems, are being replaced with a number of modular independent servo-drives. The key advantages of flexibility, simplicity and intelligent sensor based operation are compelling reasons for

incorporating advanced motion controls in modern machine designs. When moving to eliminate mechanical components, the intelligent motion solution is microprocessor based to control position, speed or torque of a motor to a high level of precision. The microprocessor interacts with its environment and with a human operator, and co-ordinates synchronises multiple axes of motion to a programmed specification.

This thesis has reviewed a number of control schemes commonly used for these types of applications. It has been found that in addition to the control algorithms for each individual axis, these axes also require coupling or interconnection in order to coordinate and synchronise with each other or external events in an optimum manner. It has also been found that existing control schemes for multi-axis motion synchronisation have some deficiencies because of: i). limitations in the control structure; ii). the uncertainness of the controlled processes; iii). the different characteristics of the individual servo-drive loops; and iv). the complex control situations where servo-drives have to be linked in a way which cannot be easily determined analytically.

First of all, this research study established a 'dual closed-loop' control structure which introduces an interconnection to enable the axes to perform in a dependent manner. In this structure, each motion is furnished with a stable closed loop servo-controller to action with the assigned task. A supervisory system provides a secondary closed-loop which provides reciprocal actions to constrain the axes of motion into a synchronisation mode. The reciprocal actions are generated by a fuzzy logic coupling algorithm which incorporates human knowledge and reasoning in coordination. With the axes of motion under such a control scheme, the computer controlled multi-axis system runs as if there existed a mechanical mechanism enforcing the motion synchronisation. Complex motion synchronisation problems can be implemented by defining appropriate fuzzy logic coupling rules. The fuzzy rule based solution can be applied in a wide range of control applications including tasks that conventional methods can not easily handle.

Through hierarchical decomposition of the tasks in machine control systems, it was found that motion synchronisation is a genuine requirement of a machine with multiple axes of motion. However, a number of 'open' machine control systems (such as UMC and MOSAIC) have not included a synchronisation level in their control architectures, which makes it difficult to provide tight synchronisation of the external devices from the

task level. A modified machine control architecture is proposed when an intelligent motion synchronisation mechanism is available. A multi-axis motion control system design method has been developed to include a synchronisation level within the machine control architecture. The approach is termed Intelligent Motion Control (IMC), which facilitates the design and control of multi-axis motion control systems in an intelligent manner.

To verify the concepts advanced in the thesis, simulation and experimental studies were conducted. The results show the effectiveness of the proposed motion synchronisation method. However, further development of the method is necessary to fully exploit its capabilities. A number of proposals have been made which could be implemented with appropriate enabling technologies.

9.2 Contributions to Knowledge

The major contributions to new knowledge that the author feels have resulted from this research study are summarised below:

(i) The research study has identified and extended the design requirements that are considered as essential for software based motion co-ordination and synchronisation mechanisms for use in modern high performance machine systems;

(ii) The research proposes and implements a novel 'dual-loop' control algorithm for generic application in motion synchronisataion scenarios using a fuzzy logic based approach. To the author's knowledge this is the first design and implementation of this type in the domain of motion control. This new control scheme has been shown to accommodate a range of application requirements in a very flexible and effective manner; demonstrating a number of significant attributes over existing control methods;

(iii) A new framework to embody the design and control of multiple independent servo-drives that require co-ordination and synchronisation has been proposed and implemented namely 'Intelligent Motion Control' (IMC). IMC provides for structured hierarchical approach in which to build software based machine control systems that require tightly co-ordinated and synchronised motion control; and

(iv) This new control scheme for motion control and synchronisation of multi-axis drive systems has been evaluated in carefully selected simulation and experimental studies to demonstrate the validity of the approach in meeting the requirements and demands of next generation machine systems.

Appendix I
Error Sources which Affect Motion Control and Synchronisation

Error sources from the controller, drive dynamics and external disturbances will affect the motion synchronisation of the multiple axes. These error sources can be further classified into three categories[1][3] [4][6].

(1) mismatch in axial-loop parameters;

(2) external disturbances, and

(3) reference commands.

A1.1. Parameter Mismatch

A mismatch in axial-loop parameters causes motion synchronisation errors. Take a general example, a closed-loop control system, shown in Figure A1.1[3]. A process, represented by the transfer function G(s), whatever its nature, is subject to a changing environment, aging, ignorance of the exact values of the process parameters, and other natural factors that affect a control process. In order to illustrate the effect of parameter variations, let us to consider a change in the process so that the new process is G(s) + ΔG(s). Then, we have

$$C(s) + \Delta(s) = \frac{G(s) + \Delta G(s)}{1 + (G(s) + \Delta G(s))\, H(s)}\, R(s) \qquad \text{(A1-1)}$$

Then the change in the output is

$$\Delta C(s) = \frac{\Delta G(s)}{(1 + GH(s) + \Delta GH(s))\,(1 + GH(s))}\, R(s) \qquad \text{(A1-2)}$$

When GH(s) >> ΔGH(s), as is often the case, we have

$$\Delta C(s) = \frac{\Delta G(s)}{(1 + GH(s))^2} R(s) \qquad \text{(A1-3)}$$

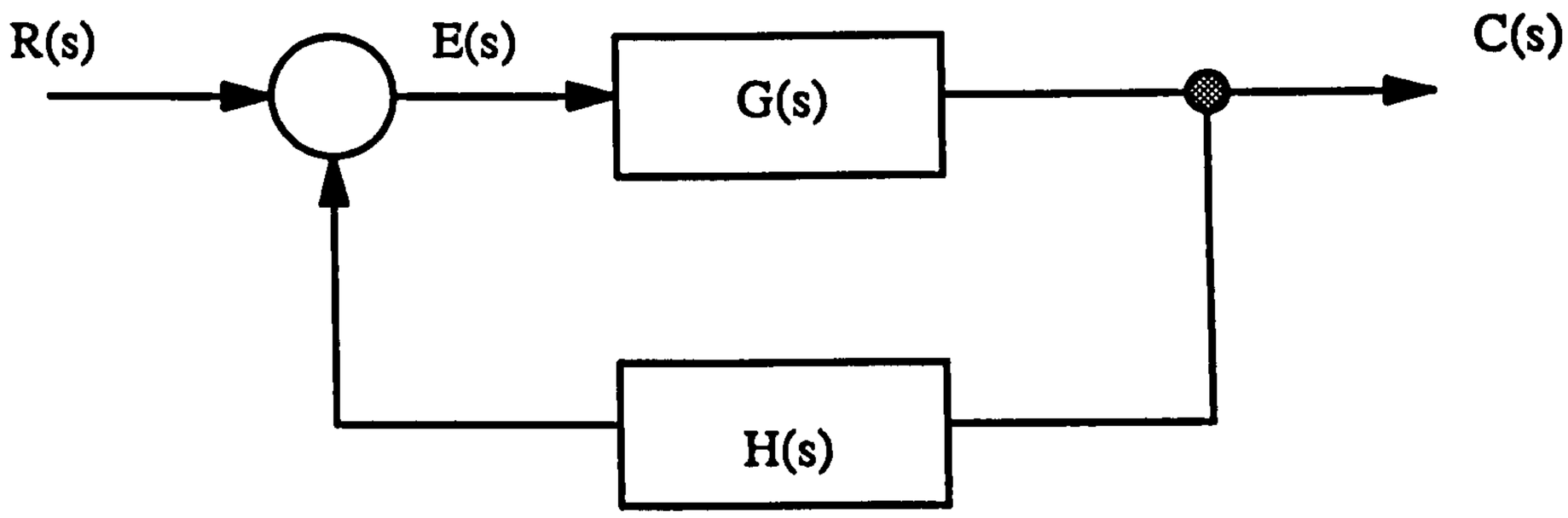


Figure A1.1 A closed-loop control system

Examining Equation (A1-3), for multiple servo-loops, parameter mismatch means different change in the respective transfer function, which can result in different changes of output of the respective closed-loop systems. Therefore, the different output changes of the servo-loops will generate synchronisation errors.

A1.2. Disturbances

A disturbance is used here to mean an external action to the loop which changes or disturbs the operation of the controlled variable. Therefore, disturbances (such as frictions, load change etc.) will influence the motion synchronisation.

As a specific example of a system with a unwanted disturbances, let us consider the speed control system for a steel rolling mill[2]. Rolls passing steel through are subject to large

sudden load changes or disturbances. As a steel bar approaches the rolls (see Figure A1.2), the rolls turn unloaded. However, when the bar engages in the rolls, the load on the rolls suddenly increases. This loading effect can be approximately by a step change of disturbance torque.

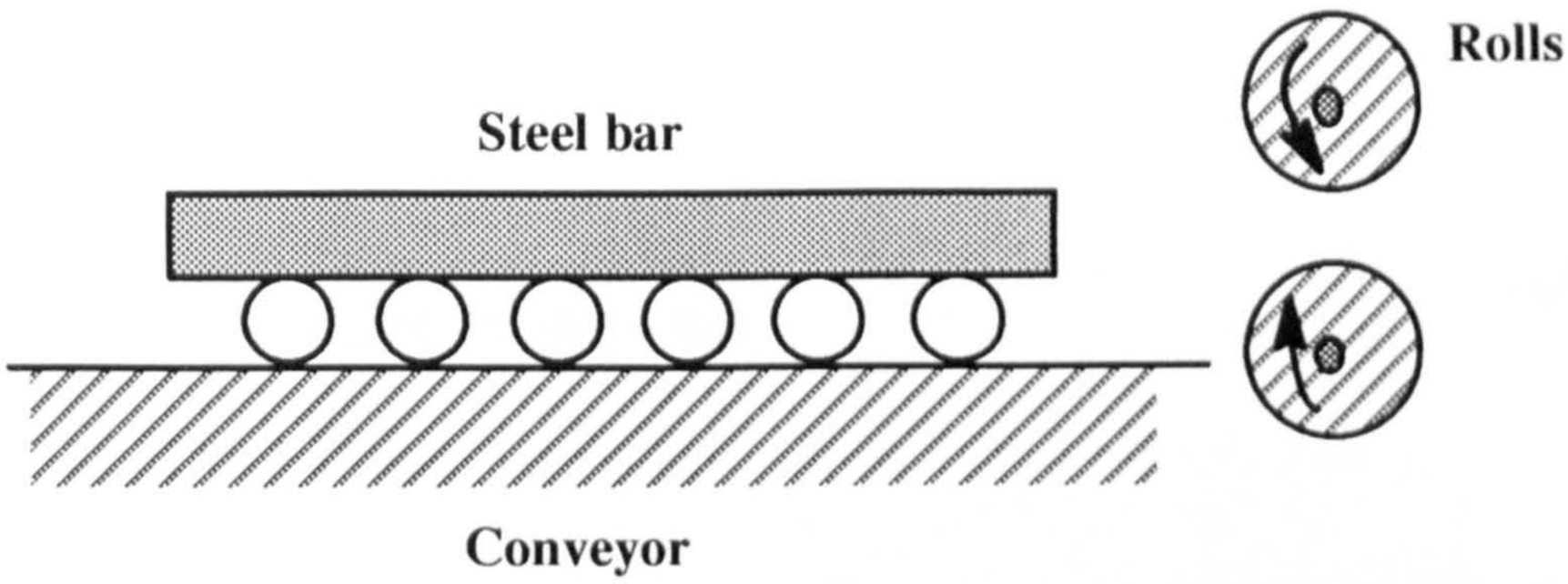


Figure A1.2 Steel rolling mill

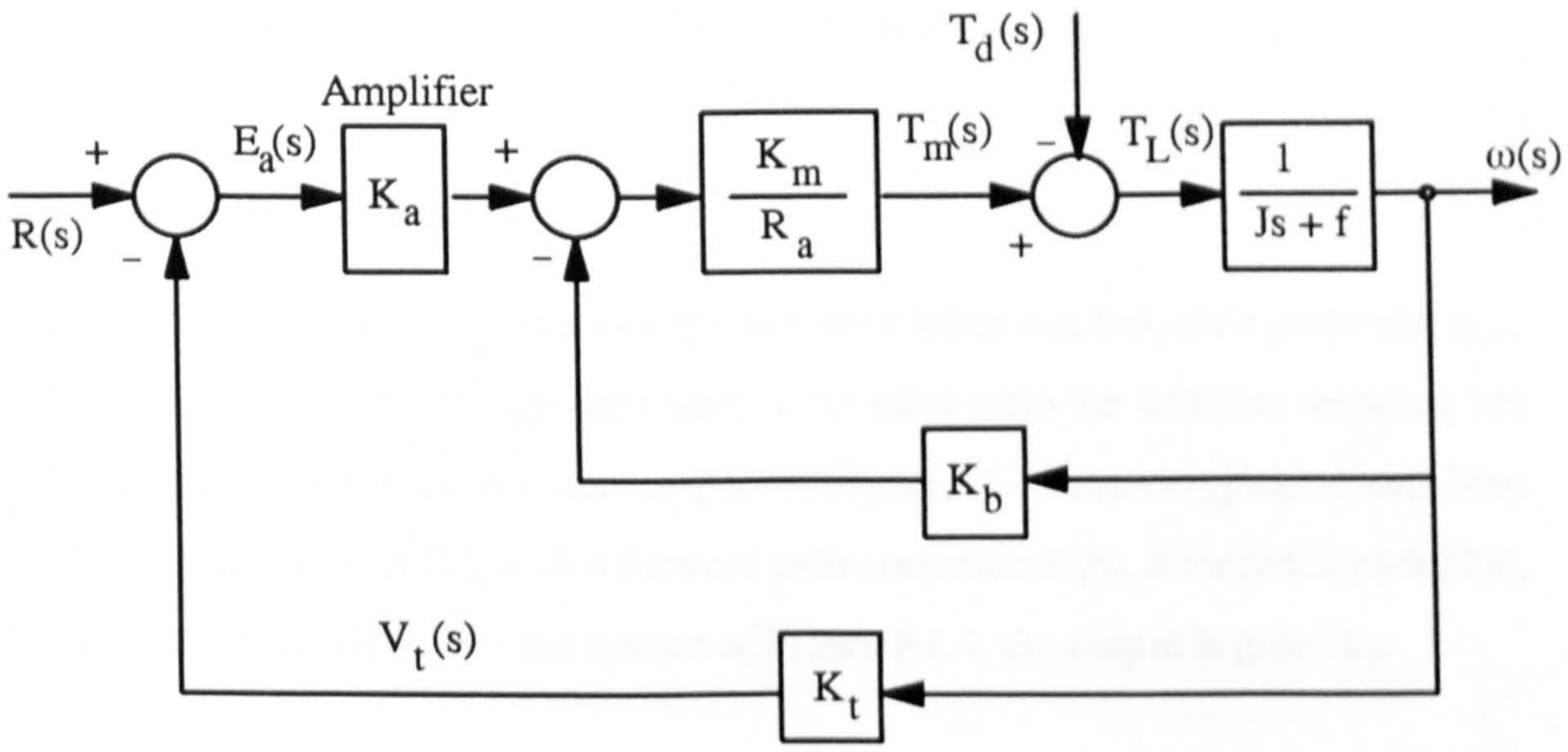


Figure A1.3 Closed-loop speed tachometer control system

The output for the speed control system of Figure A1.3 due to the load disturbance[2], when the input R(s) = 0, may be obtained using Mason's formula[5]

$$\omega(s) = \frac{-1}{Js + f + (K_m/R_a)(K_tK_a + K_b)} T_d(s) \qquad \text{(A1-4)}$$

The steady-state error in speed due to the load torque $T_d(s) = D/s$ is found using the final-value theorem, and we have:

$$\lim_{t\to\infty} \omega(t) = \lim_{s\to 0} (s\omega(s))$$

$$= \frac{-1}{f + (K_m/R_a)(K_tK_a + K_b)} D \qquad \text{(A1-5)}$$

According to Equation (A1-5), load change will result in speed change as you would intuitively expect. There will be a speed synchronisation error among the rolls and conveyor because of the different loads applied on them.

A1.3. Reference Commands

A reference input may result in a steady-state error when it is fed into a particular type of control system. The steady-state error is the error once the transient response has decayed leaving only the continuous response. Figure A1.4 shows a typical closed-loop position control system[1], with a forward path controller D(s), a feedback path H(s), and a plant model G(s). For the system of Figure A1.4, the output is given by

$$C(s) = \frac{D(s)G(s)}{1 + D(s)G(s)H(s)} R(s) \qquad \text{(A1-6)}$$

The error for the system is the difference between the input and the output, that is,

$$\text{System error} = e(t) = r(t) - c(t) \qquad \text{(A1-7)}$$

The steady-state system error is, by definition, the steady-state value of e(t). Denoting the steady-state error by e_{ss}, then, by the final value theorem of the Laplace transform

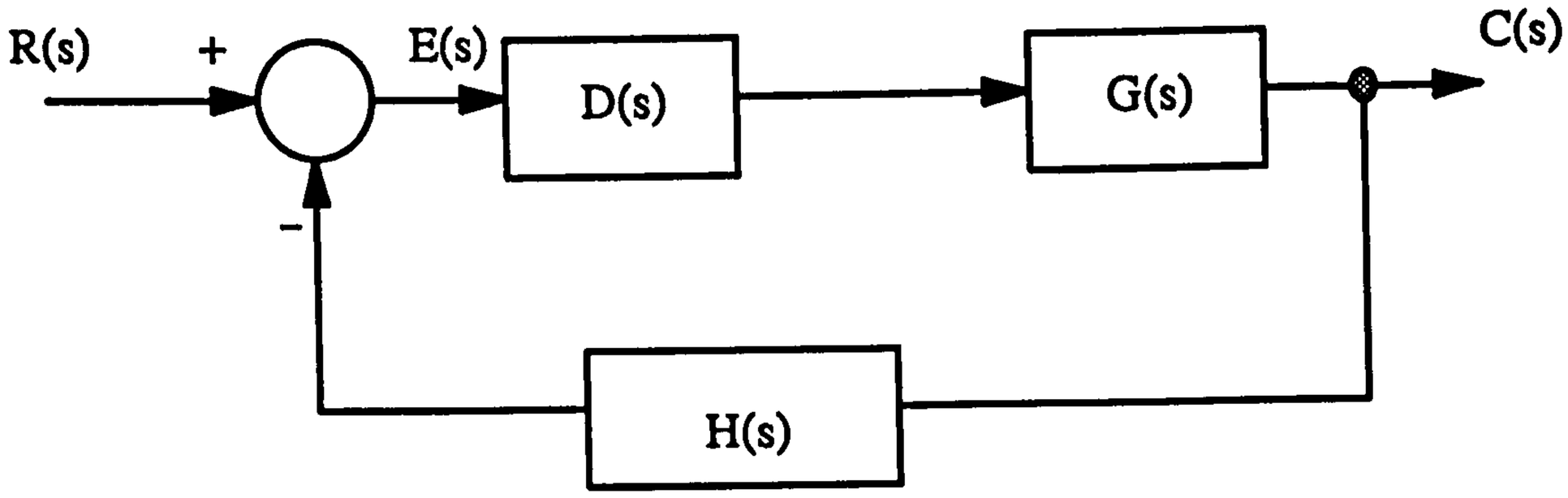


Figure A1.4 Typical Closed-loop Position Control System

[3].

$$e_{ss} = \lim_{s \to 0} sE(s) \qquad \text{(A1-8)}$$

provided that e(t) has a final value.

For the system of Figure A1.4,

$$E(s) = \frac{1 + D(s)G(s)H(s) - D(s)G(s)}{1 + D(s)G(s)H(s)} R(s) \qquad \text{(A1-9)}$$

and

$$e_{ss} = \lim_{s \to 0} SR(s) \frac{1 + D(s)G(s)H(s) - D(s)G(s)}{1 + D(s)G(s)H(s)} \qquad \text{(A1-10)}$$

Table A1.1 gives the steady-state errors for three different input functions for a unity feedback system (H(s) = 1).

Table A1.1 Steady-state Errors [3]

System type \ Input R(s)	Step Response (1/s)	Ramp Response $1/s^2$	Parabolic input $1/s^3$	
N				
0	$\frac{1}{1+Kp}$	∞	∞	$Kp = \lim_{s\to 0} sD(s)G(s)$
1	0	$\frac{1}{Kv}$	∞	$Kv = \lim_{s\to 0} sD(s)G(s)$
2	0	0	$\frac{1}{Ka}$	$Ka = \lim_{s\to 0} sD(s)G(s)$

In a multi-axis system, for each servo the reference input may be changed from one value to another value from time to time, and reference commands for different servo systems will be different at each update in order to keep a defined relationship.

It is very difficult to have the reference input always match the system type. Therefore, steady-state errors will existed in each servo loop and for multi-axis system, the error value for each servo-loop is different as such synchronisation errors will exist.

Appendix I: References

[1] Jenkinson, M., "The Synchronization of Actuators Using Scalar Field Control", PhD Thesis, University of Bristol, 1992.

[2] Richard, C. Dorf, "Modern Control System", Fifth Edition, Addison-Wesley, 1989.

[3] Phillips, C. L. and Harbor, R. D., "Feedback Control Systems", Prentice-Hall International Editions, 1991.

[4] Olsson, G. and Piani, G., "Computer Systems for Automation and Control", Prentice Hall, 1992.

[5] Bollinger, J. G. and Duffie, N. A., "Computer Control of Machines and Processes", Addison-Wesley Publishing Company, 1988.

[6] Lo, C. C. "Cross-Coupling Control of Multi-axis Manufacturing Systems", PhD Thesis, The University of Michigan, 1992.

Appendix II

The Derivation of Coefficients for Compensation Terms in Nonlinear Motion Synchronisation

The fuzzy algorithm described in section 4.3.1 is for a linear relationship between the axes. When it is used for nonlinear relationships, the coefficients for the system outputs change due to the nonlinear path. These coefficients are directly related with θ_m, but they can be expressed as

$$C_x = \cos\theta_m;$$
$$C_y = -\sin\theta_m;$$

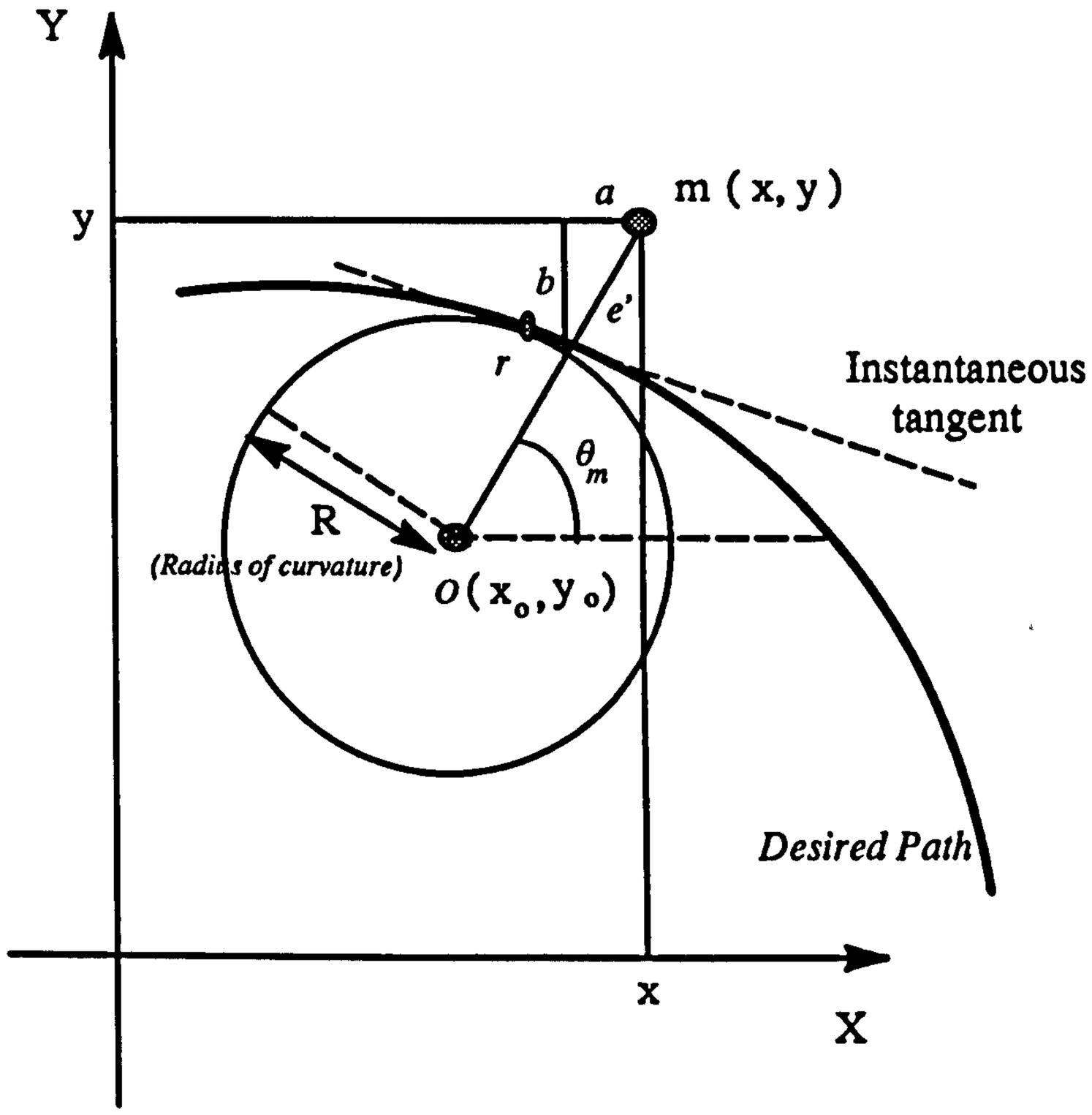
A detailed derivation of the above equations is given as follows.

(i) When $\theta_m = 0^o$ to 90^o.

This situation is the case shown in Figure A2.1. In order to eliminate the positive synchronisation error *e'*, axis *X* should reduce by *a*, and axis *Y* should also reduce by *b*. To eliminate a negative synchronisation error *e'*, axis *X* should increase by *a*, and axis *Y* should also increase by *b*. Compared with the linear relationship case as shown in Figure 4.4, to eliminate a positive synchronisation error *e*, axis *X* should reduce by *a*, and axis *Y* should increase by *b*. For elimination of a negative synchronisation error *e*, axis *X* should increase by *a*, and axis *Y* should reduce by *b*. Since the fuzzy rule base will be remained as the same as for a linear relationship, the system output for axis *Y* should change sign. Therefore, when $\theta_m = 0^o$ to 90^o, the coefficients for the system outputs are

$$C_x = \cos\theta_m;$$
$$C_y = -\sin\theta_m;$$

(ii) When $\theta_m = 90^o$ to 180^o



$$a = e' \cos\theta_m$$

$$b = e' \sin\theta_m$$

$$\sin\theta_m = \frac{y - y_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

$$\cos\theta_m = \frac{x - x_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

Figure A2.1 A Two Axis System with a Nonlinear Relationship When $0_m = 0^o$ to 90^o

Figure A2.2 shows this situation. In order to eliminate the positive synchronisation error ***e'***, axis ***X*** should increase by ***a***, and axis ***Y*** should reduce by ***b***. To eliminate a negative synchronisation error ***e'***, axis ***X*** should reduce by ***a***, and axis ***Y*** should increase by ***b***. Compared with the linear relationship case as shown in Figure 4.4, therefore, the system outputs for axis ***X*** and ***Y*** should both change sign. Therefore, when $\theta_m = 90^o$ to 180^o, the coefficients for the system outputs are

$C_x = \cos\theta_m$; (in this case, $\cos\theta_m$ is negative.)

$C_y = -\sin\theta_m$;

(iii) When $\theta_m = 180^o$ to 270^o

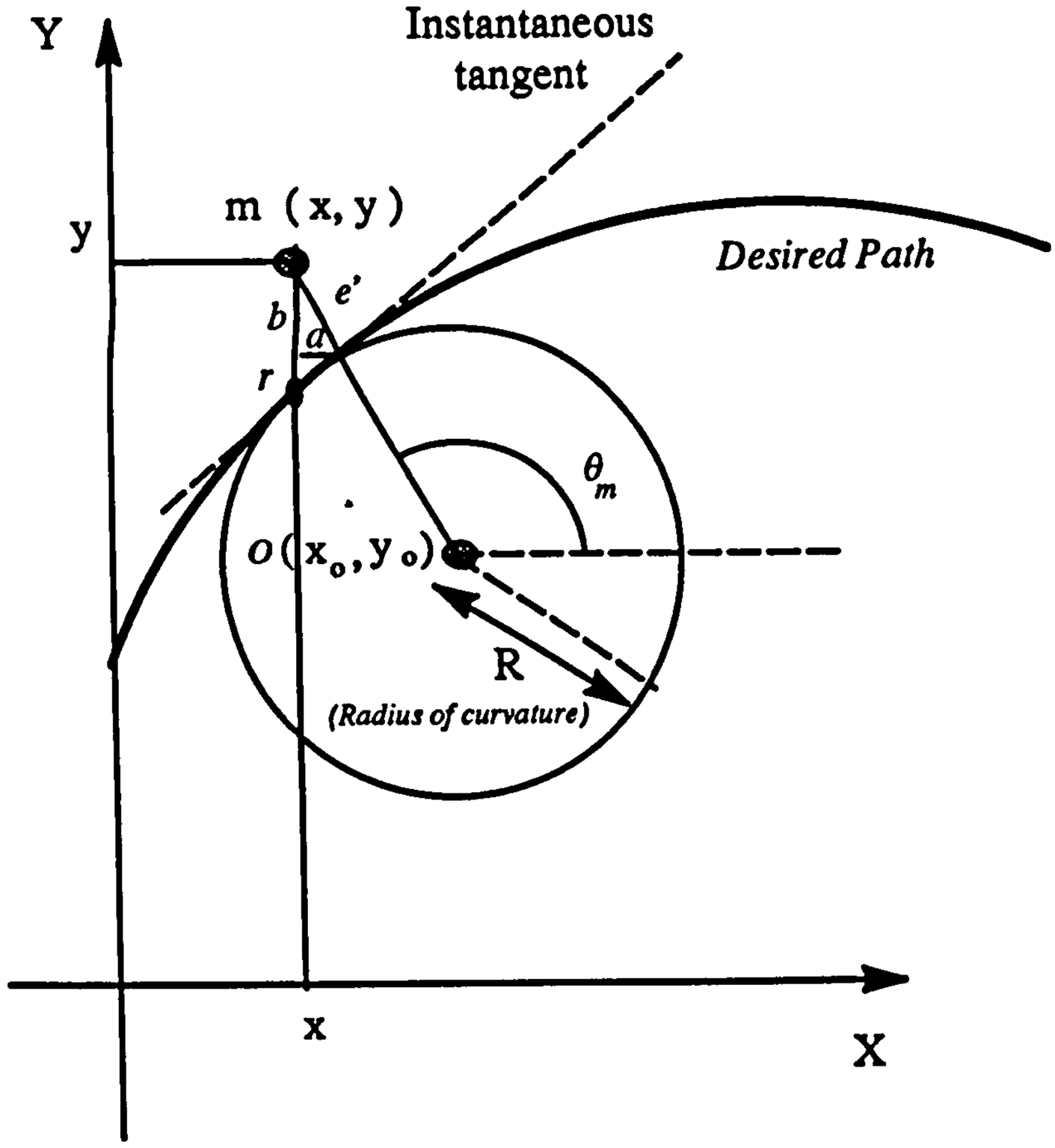
In Figure A2.3, to eliminate a positive synchronisation error ***e'***, axis ***X*** should increase by ***a***, and axis ***Y*** should also increase by ***b***. To eliminate a negative synchronisation error ***e'***, axis ***X*** should reduce by ***a***, and axis ***Y*** should also reduce by ***b***. Compared with the linear relationship case as shown in Figure 4.4, therefore, the system output for axis ***X*** should change sign. Therefore, when $\theta_m = 180^o$ to 270^o, the coefficients for the system outputs are

$C_x = \cos\theta_m$; (in this case, $\cos\theta_m$ is negative.)

$C_y = -\sin\theta_m$; (in this case, $\sin\theta_m$ is negative.)

(iv) When $\theta_m = 270^o$ to 360^o

Figure A2.4 shows this situation. To eliminate a positive synchronisation error ***e'***, axis ***X*** should reduce by ***a***, and axis ***Y*** should increase by ***b***. To eliminate a negative synchronisation error ***e'***, axis ***X*** should increase by ***a***, and axis ***Y*** should reduce by ***b***. Compared with the linear relationship case as shown in Figure 4.4, therefore, the system outputs for axis ***X*** and ***Y*** do not change sign. Therefore, when $\theta_m = 270^o$ to 360^o, the coefficients for the system outputs are



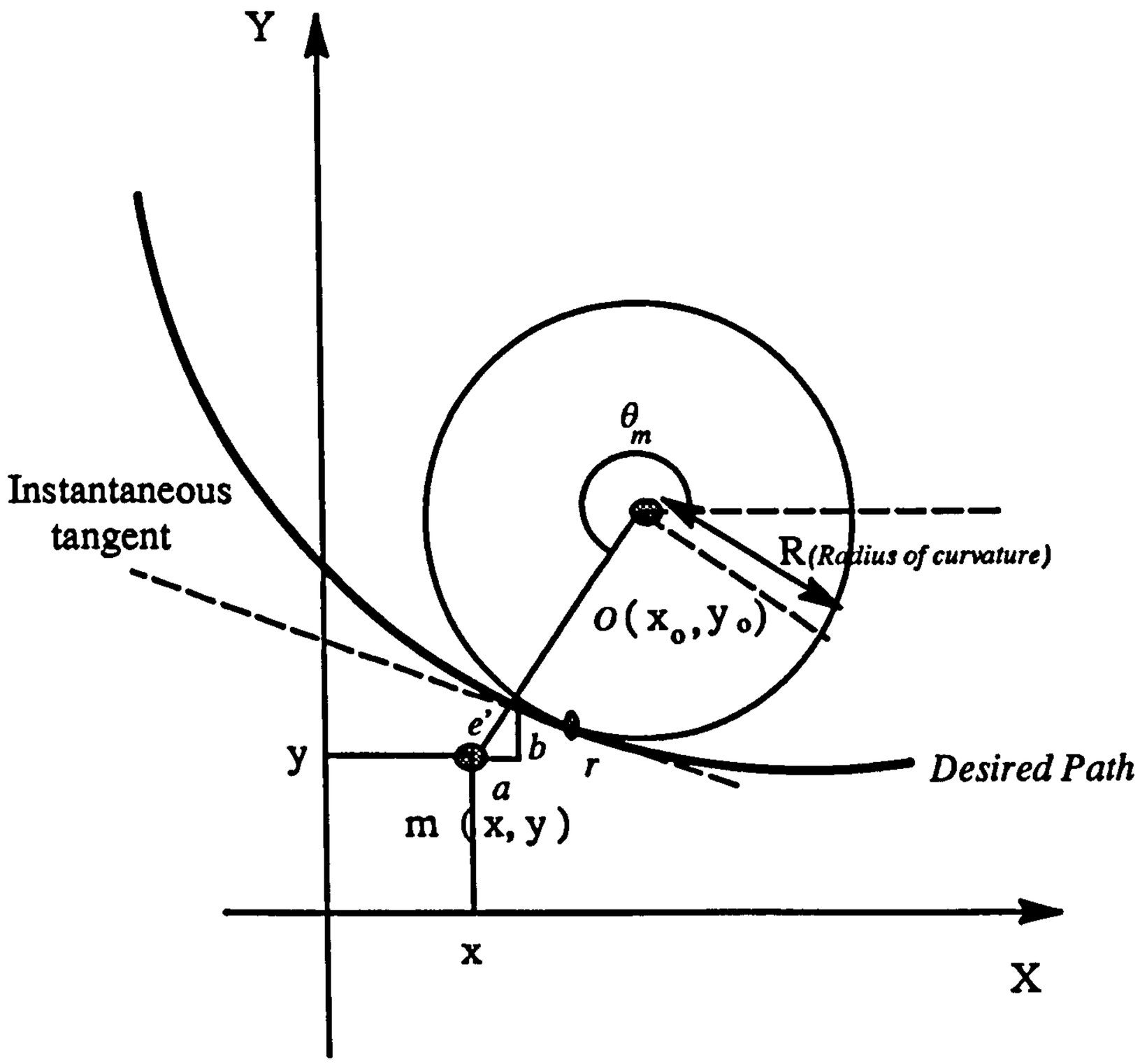
$$a = -e' \cos\theta_m$$

$$b = e' \sin\theta_m$$

$$\sin\theta_m = \frac{y - y_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

$$\cos\theta_m = \frac{x - x_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

Figure A2.2 A Two Axis System with a Nonlinear Relationship When $0_m = 90^o$ to 180^o



$$a = -e' \cos\theta_m$$

$$b = -e' \sin\theta_m$$

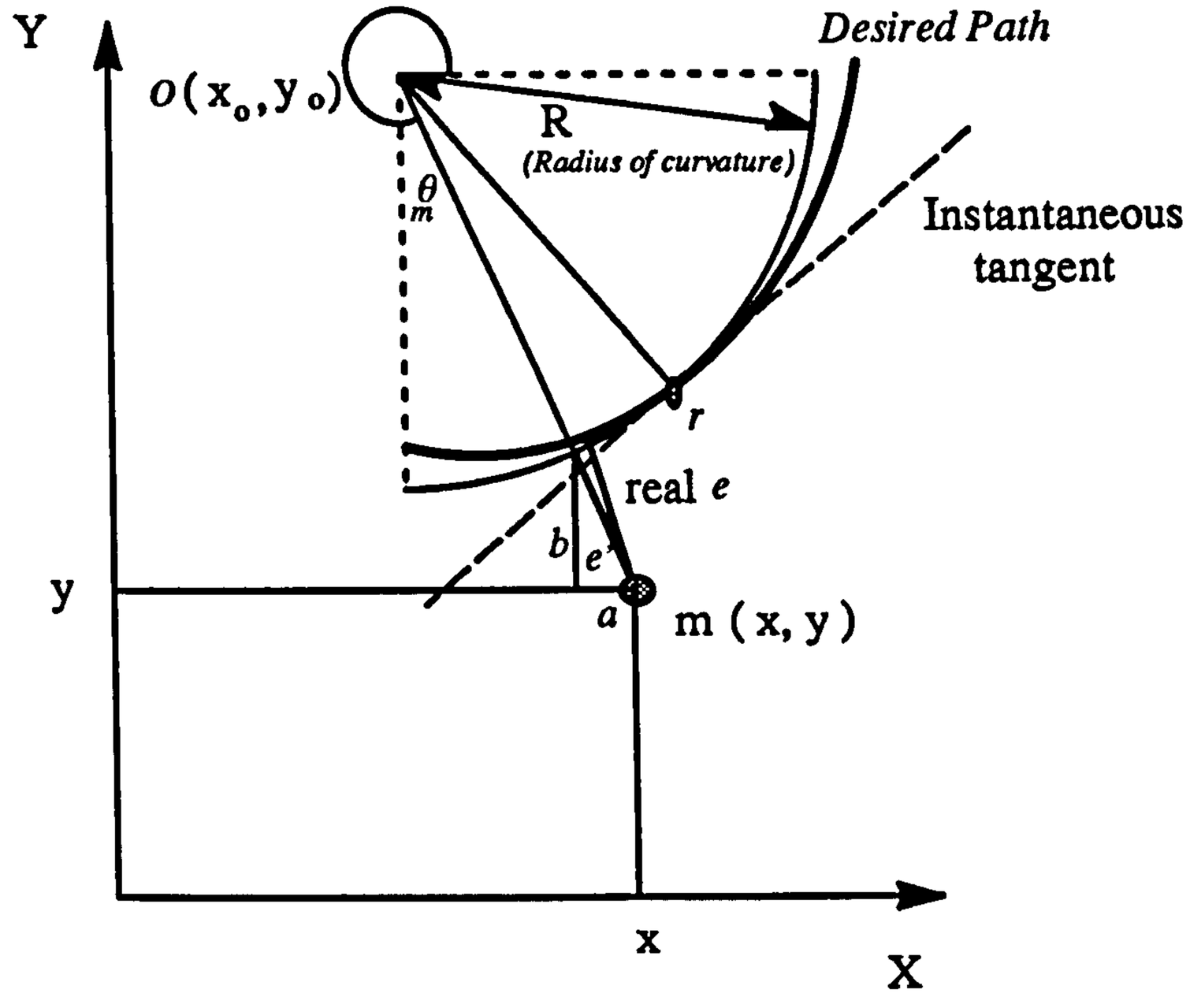
$$\sin\theta_m = \frac{y - y_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

$$\cos\theta_m = \frac{x - x_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

Figure A2.3 A Two Axis System with a Nonlinear Relationship When $\theta_m = 180^o$ to 270^o

$C_x = \cos\theta_m$;

$C_y = -\sin\theta_m$; (in this case, $\sin\theta_m$ is negative.).



$$a = e' \cos\theta_m$$

$$b = -e' \sin\theta_m$$

$$\sin\theta_m = \frac{y - y_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

$$\cos\theta_m = \frac{x - x_o}{\sqrt{(x - x_o)^2 + (y - y_o)^2}}$$

Figure A2.4 A Two Axis System with a Nonlinear Relationship When $\theta_m = 270^o$ to 360^o

Appendix II: References

[1] Jeffrey, A., "Mathematics for Engineers and Scientists", 4th edition, Van Nostrand Reinhold, 1989.

Appendix III
C-code of the Modules in Sync & Servo Levels of IMC

The C-code listed in this appendix was used to implement the modules in Sync and Servo levels of IMC which are defined in Chpater 6.

A3.1 Headers and Data Structures

```
#include <errno.h>
#include <stdio.h>
#include <math.h>


#define NULL   (void *)0
#define MAXNAME 10        /* max number of characters in
                    names */
#define UPPER_LIMIT  100  /* max number assigned as
                     degree of membership */
#define max(A,B) ((A)>(B)? (A):(B)) /* max Macro
                                    definition */
#define min(A,B) ((A)>(B)? (B):(A)) /* min Macro
                                   definition */


/* Fuzzy Inference data structures definition */


/* io_type structure builds a list of system inputs and a list of
system outputs.  After initialization, these lists are fixed, except
for value field which is updated on every inference pass.  */

struct io_type{
  char name[MAXNAME];    /*name of system
                  input/output */
  int value;             /* value of system input/output */
  struct mf_type  /*  list of membership functions for */
    *membership_functions;  /*this system input/output*/
  struct io_type *next;  /*pointer to next input/output*/
  };


/* Membership functions are associated with each system
input/output. */

struct mf_type{
  char name[MAXNAME];  /* name of membership
                 function(fuzzy set) */
  int value;             /* degree of membership or output
                 strength   */
  int point1;            /* leftmost x-axis point of mem.
                  function */
```

```
  int point2;           /* rightmost x-axis point of mem.
                  function */
  int slope1;           /* slope of left side of
                  membership function */
  int slope2;           /* slope of right side of
                  membership function */
  struct mf_type *next;   /* pointer to next membership
                     function  */
  };

/* Each rule has an if side and a then side.  Elements making up if
side are pointers to antecedent values inside mf_type structure.
Elements making up then side of rule are pointers to output strength
values, also inside mf_type structure.  Each rule structure contains
a pointer to next rule in rule base. */


struct rule_element_type{
  int *value;          /* pointer to antecedent/output
                  strength value */
  struct rule_element_type *next;  /* next
        antecedent/output element in rule */
  };

struct rule_type{
  struct rule_element_type *if_side;  /* list of
                     antecedents in rule */
  struct rule_element_type *then_side;  /* list of
                              outputs in rule */
  struct rule_type *next;    /* next rule in rule base */
  };
```

A3.2 C Subroutine Support Programs

A3.2.1 Subroutines for the Fuzzy Processees

```
fuzzification()

{  /* Fuzzification--Degree of membership value is
calculated for each membership function of each system input.
Values correspond to antecedents in rules.   */

  struct io_type *si;       /* system input pointer */
  struct mf_type *mfsi; /* membership function pointer */
  for(si=System_Inputs; si != NULL; si=si->next)
     for(mfsi=si->membership_functions; mfsi != NULL;
        mfsi=mfsi->next)
compute_degree_of_membership(mfsi,si->value);
}    /* END OF FUZZIFICATION */


rule_evaluation()

{  /* Rule Evaluation--Each rule consists of a list of
pointers to antecedents (if side), list of pointers to outputs (then
side), and pointer to next rule in rule base.  When a rule is
evaluated, its antecedents are ANDed together, using a minimum
function, to form strength of rule.  Then strength is applied to
```

```
each of listed rule outputs.  If an output has already been assigned
a rule strength, during current inference pass, a maximum function
is used to determine which strength should apply.*/

 struct rule_type *rule;
 struct rule_element_type *ip;  /* pointer of antecedents
                                (if-parts) */
 struct rule_element_type *tp; /* pointer of consequences
                                (then-parts) */
 int strength;           /*strength of rule currently being
                       evaluated */
 for(rule=Rule_Bases; rule != NULL; rule=rule->next)
   {
    strength = UPPER_LIMIT; /*max rule strength allowed*/
      /* process if-side of rule to determine strength */
    for(ip=rule->if_side; ip !=NULL; ip=ip->next)
        strength = min(strength,*(ip->value));
      /* process then-side of rule to apply strength */
    for(tp=rule->then_side; tp !=NULL; tp=tp->next)
         *(tp->value) = max(strength,*(tp->value));
   }
}    /* END RULE EVALUATION */


defuzzification()

{
  struct io_type *so;       /* system output pointer */
  struct mf_type *mfso;       /* output membership
                              function pointer */
  int sum_of_products;      /* sum of products of area &
                             centroid */
  int sum_of_areas; /* sum of shortend trapezoid area */
  int area;
  int centroid;
 /* compute a defuzzified value for each system output */
  for(so=System_Outputs; so != NULL; so=so->next)
    {
       sum_of_products = 0;
       sum_of_areas = 0;
      for(mfso=so->membership_functions; mfso != NULL;
mfso=mfso->next)
       {
         area = compute_area_of_trapezoid(mfso);
         centroid = mfso->point1 + (mfso->point2
            -mfso->point1)/2;
         sum_of_products += area * centroid;
         sum_of_areas += area;
       }
     if(sum_of_areas == 0)
        so->value = 0; /* position adjustment is zero */
     else
        {so->value = sum_of_products/sum_of_areas;}
             /* weighted average */
     }
}    /* END DEFUZZIFICATION */


compute_degree_of_membership(mf,input)
struct mf_type *mf;  /* membership function pointer */
```

```
int input;
{ /* Compute Degree of Membership--Degree to which input is a member
of mf is calculated as follows: 1.Compute delta terms to determine
if input is inside or outside membership function.  2. If outside,
then degree of membership is 0.  Otherwise, smaller of delta_1 *
slope1 and delta_2 * slope2 apllies.  3. Enforce upper limit. */

  int delta_1;
  int delta_2;
  delta_1 = input - mf->point1;
  delta_2 = mf->point2 - input;
  if ((delta_1 <= 0) || (delta_2 <= 0))  /* input outside
                          mem. function ? */
    mf->value = 0;   /* the degree of membership is 0 */
  else{
    mf->value = min( (mf->slope1*delta_1),
          (mf->slope2*delta_2) );
    mf->value = min(mf->value, UPPER_LIMIT);} /* enforce
                               upper limit */
}  /* END DEGREE OF MEMBERSHIP */

compute_area_of_trapezoid(mf)
struct mf_type *mf;
{  /* Compute Area of Trapezoid--Each inference pass
produces a new set of output strengths which affect the areas of
trapezoidal membership functions used in center-of-gravity
defuzzification.  Area of trapezoid is h*(a+b)/2 where
h=height=output_strength=mf->value,
b=base=mf->point2-mf->point1, a=top= must be derived from h,b, and
slopes1&2. */

  int run_1;
  int run_2;
  int base;
  int top;
  int area;
  base = mf->point2 -mf->point1;
  run_1 = mf->value/mf->slope1;
  run_2 = mf->value/mf->slope2;
  top = base - run_1 -run_2;
  area = mf->value * (base + top)/2;
  return(area);
}  /* END AREA OF TRAPEZOID */


initialize_system()
{ /* Gains for DSC-1 Controllers */
 static int kp[] = { 205, 180 },
            ki[] = { 5, 10 },
            kd[] = { 50, 50 },
            kv[] = { 1200, 1200 },
            kf[] = { 1000, 1200 },

  /* parameters of DSC-1 */
            sv[] = { 10000, 10000 }, /* set velocity */
            sa[] = { 200000, 200000 },/*set acceleration
                                 */
```

```
/*****************************************************
*                                                    *
*    initialize the fuzzy inference system           *
*                                                    *
*****************************************************/

/* initialize the membership functions for the System inputs */
 static struct mf_type SI1mf[7] = {
     { "NLip1", 0, -260, -60, 1, 1, &SI1mf[1]},
     { "NMip1", 0, -110, -10, 2, 2, &SI1mf[2]},
     { "NSip1", 0, -20, 0, 10, 10, &SI1mf[3]},
     { "ZRip1", 0, -10, 10, 10, 10, &SI1mf[4]},
     { "PSip1", 0, 0, 20, 10, 10, &SI1mf[5]},
     { "PMip1", 0, 10, 110, 2, 2, &SI1mf[6]},
     { "PLip1", 0, 60, 260, 1, 1,NULL}
     };          /* the membership functions for the
         System input 1 */

static struct mf_type SI2mf[7] = {
     { "NLip2", 0, -260, -60, 1, 1, &SI2mf[1]},
     { "NMip2", 0, -110, -10, 2, 2, &SI2mf[2]},
     { "NSip2", 0, -20, 0, 10, 10, &SI2mf[3]},
     { "ZRip2", 0, -10, 10, 10, 10, &SI2mf[4]},
     { "PSip2", 0, 0, 20, 10, 10, &SI2mf[5]},
     { "PMip2", 0, 10, 110, 2, 2, &SI2mf[6]},
     { "PLip2", 0, 60, 260, 1, 1,NULL}
     };          /* the membership functions for the
         System input 2 */
/* initialize the inputs--synchronisation error and the error change
rate */
 static struct io_type System_Inputs[2] = {
     { "Vel_Syn_Er", 0, &SI1mf[0], &System_Inputs[1]},
     { "Syn_Er_Rt", 0, &SI2mf[0],NULL}
     };


/* initialize the membership functions for the System Outputs */
 static struct mf_type SO1mf[19] = {
     { "NBLop1", 0, -155, -115, 5, 5, &SO1mf[1]},
     { "NBMop1", 0, -135, -95, 5, 5, &SO1mf[2]},
     { "NBSop1", 0, -115, -75, 5, 5, &SO1mf[3]},
     { "NMLop1", 0, -95, -55, 5, 5, &SO1mf[4]},
     { "NMMop1", 0, -75, -35, 5, 5, &SO1mf[5]},
     { "NSLop1", 0, -45, -25, 10, 10, &SO1mf[6]},
     { "NSMop1", 0, -35, -15, 10, 10, &SO1mf[7]},
     { "NSop1", 0, -25, -5, 10, 10, &SO1mf[8]},
     { "NVSop1", 0, -10, 0, 20, 20, &SO1mf[9]},
     { "ZRop1", 0, -5, 5, 20, 20, &SO1mf[10]},
     { "PVSop1", 0, 0, 10, 20, 20, &SO1mf[11]},
     { "PSop1", 0, 5, 25, 10, 10, &SO1mf[12]},
     { "PSMop1", 0, 15, 35, 10, 10, &SO1mf[13]},
     { "PSLop1", 0, 25, 45, 10, 10, &SO1mf[14]},
     { "PMMop1", 0, 35, 75, 5, 5, &SO1mf[15]},
     { "PMLop1", 0, 55, 95, 5, 5, &SO1mf[16]},
     { "PBSop1", 0, 75, 115, 5, 5, &SO1mf[17]},
     { "PBMop1", 0, 95, 135, 5, 5, &SO1mf[18]},
     { "PBLop1", 0, 115, 155, 5, 5,NULL}
     };          /* the membership functions for the
       System Output 1 */
```

```
static struct mf_type SO2mf[19] = {
    { "NBLop2", 0, -155, -115, 5, 5, &SO2mf[1]},
    { "NBMop2", 0, -135, -95, 5, 5, &SO2mf[2]},
    { "NBSop2", 0, -115, -75, 5, 5, &SO2mf[3]},
    { "NMLop2", 0, -95, -55, 5, 5, &SO2mf[4]},
    { "NMMop2", 0, -75, -35, 5, 5, &SO2mf[5]},
    { "NSLop2", 0, -45, -25, 10, 10, &SO2mf[6]},
    { "NSMop2", 0, -35, -15, 10, 10, &SO2mf[7]},
    { "NSop2", 0, -25, -5, 10, 10, &SO2mf[8]},
    { "NVSop2", 0, -10, 0, 20, 20, &SO2mf[9]},
    { "ZRop2", 0, -5, 5, 20, 20, &SO2mf[10]},
    { "PVSop2", 0, 0, 10, 20, 20, &SO2mf[11]},
    { "PSop2", 0, 5, 25, 10, 10, &SO2mf[12]},
    { "PSMop2", 0, 15, 35, 10, 10, &SO2mf[13]},
    { "PSLop2", 0, 25, 45, 10, 10, &SO2mf[14]},
    { "PMMop2", 0, 35, 75, 5, 5, &SO2mf[15]},
    { "PMLop2", 0, 55, 95, 5, 5, &SO2mf[16]},
    { "PBSop2", 0, 75, 115, 5, 5, &SO2mf[17]},
    { "PBMop2", 0, 95, 135, 5, 5, &SO2mf[18]},
    { "PBLop2", 0, 115, 155, 5, 5,NULL}
    };                  /* the membership functions for the
              System Output 2 */


/* initialize the outputs--Axes reference adjustment amount */
 static struct io_type System_Outputs[2] = {
     { "Vel1_adj", 0, &SO1mf[0],&System_Outputs[1]},
     { "Vel2_adj", 0, &SO2mf[0],NULL}
     };

/* initialize the rule base */

/* Rule 1: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=ZRip2, Then
Vel1_adj=PSMop1,Vel2_adj=NSMop2; */
 static struct rule_element_type is1[2] = {
     { &SI1mf[0].value, &is1[1]},
     { &SI2mf[3].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts1[2] = {
     { &SO1mf[12].value,&ts1[1]},
     { &SO2mf[6].value,NULL}
     };  /* Then Side */

/* Rule 2: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=NSip2, Then
Vel1_adj=PSMop1,Vel2_adj=NSMop2; */
 static struct rule_element_type is2[2] = {
     { &SI1mf[1].value, &is2[1]},
     { &SI2mf[2].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts2[2] = {
     { &SO1mf[12].value,&ts2[1]},
     { &SO2mf[6].value,NULL}
     };  /* Then Side */

/* Rule 3: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=ZRip2, Then
Vel1_adj=PSop1,Vel2_adj=NSop2; */
 static struct rule_element_type is3[2] = {
     { &SI1mf[1].value, &is3[1]},
     { &SI2mf[3].value,NULL}
     };  /* If Side */
```

```
 static struct rule_element_type ts3[2] = {
     { &SO1mf[11].value,&ts3[1]},
     { &SO2mf[7].value,NULL}
     };  /* Then Side */

/* Rule 4: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=PSip2, Then
   Vel1_adj=PSop1,Vel2_adj=NSop2; */
 static struct rule_element_type is4[2] = {
     { &SI1mf[1].value, &is4[1]},
     { &SI2mf[4].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts4[2] = {
     { &SO1mf[11].value,&ts4[1]},
     { &SO2mf[7].value,NULL}
     };  /* Then Side */

/* Rule 5: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=NMip2, Then
Vel1_adj=PSop1,Vel2_adj=NSop2; */
 static struct rule_element_type is5[2] = {
     { &SI1mf[2].value, &is5[1]},
     { &SI2mf[1].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts5[2] = {
     { &SO1mf[11].value,&ts5[1]},
     { &SO2mf[7].value,NULL}
     }; /* Then Side */
/* Rule 6: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=NSip2, Then
Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
 static struct rule_element_type is6[2] = {
     { &SI1mf[2].value, &is6[1]},
     { &SI2mf[2].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts6[2] = {
     { &SO1mf[10].value,&ts6[1]},
     { &SO2mf[8].value,NULL}
     }; /* Then Side */

/* Rule 7: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=ZRip2, Then
Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
 static struct rule_element_type is7[2] = {
     { &SI1mf[2].value, &is7[1]},
     { &SI2mf[3].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts7[2] = {
     { &SO1mf[10].value,&ts7[1]},
     { &SO2mf[8].value,NULL}
     }; /* Then Side */
/* Rule 8: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=PSip2, Then
Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
 static struct rule_element_type is8[2] = {
     { &SI1mf[2].value, &is8[1]},
     { &SI2mf[4].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts8[2] = {
     { &SO1mf[10].value,&ts8[1]},
     { &SO2mf[8].value,NULL}
     }; /* Then Side */

/* Rule 9: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=PMip2, Then
Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
```

```
 static struct rule_element_type is9[2] = {
     { &SI1mf[2].value, &is9[1]},
     { &SI2mf[5].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts9[2] = {
     { &SO1mf[10].value,&ts9[1]},
     { &SO2mf[8].value,NULL}
     }; /* Then Side */

/* Rule 10: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=NLip2, Then
 Vel1_adj=PSMop1,Vel2_adj=NSMop2; */
 static struct rule_element_type is10[2] = {
     { &SI1mf[3].value, &is10[1]},
     { &SI2mf[0].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts10[2] = {
     { &SO1mf[12].value,&ts10[1]},
     { &SO2mf[6].value,NULL}
     }; /* Then Side */
/* Rule 11: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=NMip2, Then
 Vel1_adj=PSop1,Vel2_adj=NSop2; */
 static struct rule_element_type is11[2] = {
     { &SI1mf[3].value, &is11[1]},
     { &SI2mf[1].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts11[2] = {
     { &SO1mf[11].value,&ts11[1]},
     { &SO2mf[7].value,NULL}
     }; /* Then Side */

/* Rule 12: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=NSip2, Then
 Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
 static struct rule_element_type is12[2] = {
     { &SI1mf[3].value, &is12[1]},
     { &SI2mf[2].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts12[2] = {
     { &SO1mf[10].value,&ts12[1]},
     { &SO2mf[8].value,NULL}
     }; /* Then Side */
/* Rule 13: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=ZRip2, Then
 Vel1_adj=ZRop1,Vel2_adj=ZRop2; */
 static struct rule_element_type is13[2] = {
     { &SI1mf[3].value, &is13[1]},
     { &SI2mf[3].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts13[2] = {
     { &SO1mf[9].value,&ts13[1]},
     { &SO2mf[9].value,NULL}
     }; /* Then Side */

/* Rule 14: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=PSip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is14[2] = {
     { &SI1mf[3].value, &is14[1]},
     { &SI2mf[4].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts14[2] = {
     { &SO1mf[8].value,&ts14[1]},
     { &SO2mf[10].value,NULL}
```

```
    }; /* Then Side */

/* Rule 15: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=PMip2, Then
 Vel1_adj=NSop1,Vel2_adj=PSop2; */
 static struct rule_element_type is15[2] = {
     { &SI1mf[3].value, &is15[1]},
     { &SI2mf[5].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts15[2] = {
     { &SO1mf[7].value,&ts15[1]},
     { &SO2mf[11].value,NULL}
     }; /* Then Side */

/* Rule 16: If Vel_Syn_Er=ZRip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=NSMop1,Vel2_adj=PSMop2; */
 static struct rule_element_type is16[2] = {
     { &SI1mf[3].value, &is16[1]},
     { &SI2mf[6].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts16[2] = {
     { &SO1mf[6].value,&ts16[1]},
     { &SO2mf[12].value,NULL}
     }; /* Then Side */

/* Rule 17: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=NMip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is17[2] = {
     { &SI1mf[4].value, &is17[1]},
     { &SI2mf[1].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts17[2] = {
     { &SO1mf[8].value,&ts17[1]},
     { &SO2mf[10].value,NULL}
     }; /* Then Side */

/* Rule 18: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=NSip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is18[2] = {
     { &SI1mf[4].value, &is18[1]},
     { &SI2mf[2].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts18[2] = {
     { &SO1mf[8].value,&ts18[1]},
     { &SO2mf[10].value,NULL}
     }; /* Then Side */

/* Rule 19: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=ZRip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is19[2] = {
     { &SI1mf[4].value, &is19[1]},
     { &SI2mf[3].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts19[2] = {
     { &SO1mf[8].value,&ts19[1]},
     { &SO2mf[10].value,NULL}
     }; /* Then Side */

/* Rule 20: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=PSip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is20[2] = {
     { &SI1mf[4].value, &is20[1]},
     { &SI2mf[4].value,NULL}
```

```
    };  /* If Side */
 static struct rule_element_type ts20[2] = {
    { &SO1mf[8].value,&ts20[1]},
    { &SO2mf[10].value,NULL}
    }; /* Then Side */

/* Rule 21: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=PMip2, Then
 Vel1_adj=NSMop1,Vel2_adj=PSMop2; */
 static struct rule_element_type is21[2] = {
    { &SI1mf[4].value, &is21[1]},
    { &SI2mf[5].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts21[2] = {
    { &SO1mf[6].value,&ts21[1]},
    { &SO2mf[12].value,NULL}
    }; /* Then Side */

/* Rule 22: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=NSip2, Then
 Vel1_adj=NSop1,Vel2_adj=PSop2; */
 static struct rule_element_type is22[2] = {
    { &SI1mf[5].value, &is22[1]},
    { &SI2mf[2].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts22[2] = {
    { &SO1mf[7].value,&ts22[1]},
    { &SO2mf[11].value,NULL}
    }; /* Then Side */

/* Rule 23: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=ZRip2, Then
 Vel1_adj=NSop1,Vel2_adj=PSop2; */
 static struct rule_element_type is23[2] = {
    { &SI1mf[5].value, &is23[1]},
    { &SI2mf[3].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts23[2] = {
    { &SO1mf[7].value,&ts23[1]},
    { &SO2mf[11].value,NULL}
    }; /* Then Side */

/* Rule 24: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=PSip2, Then
 Vel1_adj=NSMop1,Vel2_adj=PSMop2; */
 static struct rule_element_type is24[2] = {
    { &SI1mf[5].value, &is24[1]},
    { &SI2mf[4].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts24[2] = {
    { &SO1mf[6].value,&ts24[1]},
    { &SO2mf[12].value,NULL}
    }; /* Then Side */

/* Rule 25: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=ZRip2, Then
 Vel1_adj=NSMop1,Vel2_adj=PSMop2; */
 static struct rule_element_type is25[2] = {
    { &SI1mf[6].value, &is25[1]},
    { &SI2mf[3].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts25[2] = {
    { &SO1mf[6].value,&ts25[1]},
    { &SO2mf[12].value,NULL}
    }; /* Then Side */

/* Rule 26: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=NMip2, Then
```

```
Vel1_adj=PSMop1,Vel2_adj=NSMop2; */
static struct rule_element_type is26[2] = {
    { &SI1mf[1].value, &is26[1]},
    { &SI2mf[1].value,NULL}
    };  /* If Side */
static struct rule_element_type ts26[2] = {
    { &SO1mf[12].value,&ts26[1]},
    { &SO2mf[6].value,NULL}
    }; /* Then Side */
/* Rule 27: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=PMip2, Then
Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
static struct rule_element_type is27[2] = {
    { &SI1mf[1].value, &is27[1]},
    { &SI2mf[5].value,NULL}
    };  /* If Side */
static struct rule_element_type ts27[2] = {
    { &SO1mf[10].value,&ts27[1]},
    { &SO2mf[8].value,NULL}
    }; /* Then Side */
/* Rule 28: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=NMip2, Then
Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
static struct rule_element_type is28[2] = {
    { &SI1mf[5].value, &is28[1]},
    { &SI2mf[1].value,NULL}
    };  /* If Side */
static struct rule_element_type ts28[2] = {
    { &SO1mf[8].value,&ts28[1]},
    { &SO2mf[10].value,NULL}
    }; /* Then Side */
/* Rule 29: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=PMip2, Then
Vel1_adj=NSLop1,Vel2_adj=PSLop2; */
static struct rule_element_type is29[2] = {
    { &SI1mf[5].value, &is29[1]},
    { &SI2mf[5].value,NULL}
    };  /* If Side */
static struct rule_element_type ts29[2] = {
    { &SO1mf[5].value,&ts29[1]},
    { &SO2mf[13].value,NULL}
    }; /* Then Side */
/* Rule 30: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=NLip2, Then
Vel1_adj=PMLop1,Vel2_adj=NMLop2; */
static struct rule_element_type is30[2] = {
    { &SI1mf[0].value, &is30[1]},
    { &SI2mf[0].value,NULL}
    };  /* If Side */
static struct rule_element_type ts30[2] = {
    { &SO1mf[15].value,&ts30[1]},
    { &SO2mf[3].value,NULL}
    }; /* Then Side */
/* Rule 31: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=NMip2, Then
Vel1_adj=PMMop1,Vel2_adj=NMMop2; */
static struct rule_element_type is31[2] = {
    { &SI1mf[0].value, &is31[1]},
    { &SI2mf[1].value,NULL}
    };  /* If Side */
static struct rule_element_type ts31[2] = {
    { &SO1mf[14].value,&ts31[1]},
    { &SO2mf[4].value,NULL}
    }; /* Then Side */
```

```
/* Rule 32: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=NSip2, Then
 Vel1_adj=PSLop1,Vel2_adj=NSLop2; */
 static struct rule_element_type is32[2] = {
     { &SI1mf[0].value, &is32[1]},
     { &SI2mf[2].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts32[2] = {
     { &SO1mf[13].value,&ts32[1]},
     { &SO2mf[5].value,NULL}
     }; /* Then Side */

/* Rule 33: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=NLip2, Then
 Vel1_adj=PMMop1,Vel2_adj=NMMop2; */
 static struct rule_element_type is33[2] = {
     { &SI1mf[1].value, &is33[1]},
     { &SI2mf[0].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts33[2] = {
     { &SO1mf[14].value,&ts33[1]},
     { &SO2mf[4].value,NULL}
     }; /* Then Side */

/* Rule 34: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=NLip2, Then
 Vel1_adj=PSMop1,Vel2_adj=NSMop2; */
 static struct rule_element_type is34[2] = {
     { &SI1mf[2].value, &is34[1]},
     { &SI2mf[0].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts34[2] = {
     { &SO1mf[12].value,&ts34[1]},
     { &SO2mf[6].value,NULL}
     }; /* Then Side */

/* Rule 35: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=NLip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is35[2] = {
     { &SI1mf[4].value, &is35[1]},
     { &SI2mf[0].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts35[2] = {
     { &SO1mf[8].value,&ts35[1]},
     { &SO2mf[10].value,NULL}
     }; /* Then Side */

/* Rule 36: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=NLip2, Then
 Vel1_adj=NVSop1,Vel2_adj=PVSop2; */
 static struct rule_element_type is36[2] = {
     { &SI1mf[5].value, &is36[1]},
     { &SI2mf[0].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts36[2] = {
     { &SO1mf[8].value,&ts36[1]},
     { &SO2mf[10].value,NULL}
     }; /* Then Side */

/* Rule 37: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=NLip2, Then
 Vel1_adj=NSop1,Vel2_adj=PSop2; */
 static struct rule_element_type is37[2] = {
     { &SI1mf[6].value, &is37[1]},
     { &SI2mf[0].value,NULL}
     };  /* If Side */
```

```
 static struct rule_element_type ts37[2] = {
     { &SO1mf[7].value,&ts37[1]},
     { &SO2mf[11].value,NULL}
     }; /* Then Side */
/* Rule 38: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=NMip2, Then
 Vel1_adj=NSop1,Vel2_adj=PSop2; */
 static struct rule_element_type is38[2] = {
     { &SI1mf[6].value, &is38[1]},
     { &SI2mf[1].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts38[2] = {
     { &SO1mf[7].value,&ts38[1]},
     { &SO2mf[11].value,NULL}
     }; /* Then Side */
/* Rule 39: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=NSip2, Then
 Vel1_adj=NSMop1,Vel2_adj=PSMop2; */
 static struct rule_element_type is39[2] = {
     { &SI1mf[6].value, &is39[1]},
     { &SI2mf[2].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts39[2] = {
     { &SO1mf[6].value,&ts39[1]},
     { &SO2mf[12].value,NULL}
     }; /* Then Side */

/* Rule 40: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=PSip2, Then
 Vel1_adj=NSLop1,Vel2_adj=PSLop2; */
 static struct rule_element_type is40[2] = {
     { &SI1mf[6].value, &is40[1]},
     { &SI2mf[4].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts40[2] = {
     { &SO1mf[5].value,&ts40[1]},
     { &SO2mf[13].value,NULL}
     }; /* Then Side */
/* Rule 41: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=PMip2, Then
 Vel1_adj=NMMop1,Vel2_adj=PMMop2; */
 static struct rule_element_type is41[2] = {
     { &SI1mf[6].value, &is41[1]},
     { &SI2mf[5].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts41[2] = {
     { &SO1mf[4].value,&ts41[1]},
     { &SO2mf[14].value,NULL}
     }; /* Then Side */

/* Rule 42: If Vel_Syn_Er=PLip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=NMLop1,Vel2_adj=PMLop2; */
 static struct rule_element_type is42[2] = {
     { &SI1mf[6].value, &is42[1]},
     { &SI2mf[6].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts42[2] = {
     { &SO1mf[3].value,&ts42[1]},
     { &SO2mf[15].value,NULL}
     }; /* Then Side */
/* Rule 43: If Vel_Syn_Er=PMip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=NMMop1,Vel2_adj=PMMop2; */
 static struct rule_element_type is43[2] = {
```

```
    { &SI1mf[5].value, &is43[1]},
    { &SI2mf[6].value,NULL}
    };  /* If Side */
static struct rule_element_type ts43[2] = {
    { &SO1mf[4].value,&ts43[1]},
    { &SO2mf[14].value,NULL}
    }; /* Then Side */

/* Rule 44: If Vel_Syn_Er=PSip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=NSMop1,Vel2_adj=PSMop2; */
 static struct rule_element_type is44[2] = {
    { &SI1mf[4].value, &is44[1]},
    { &SI2mf[6].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts44[2] = {
    { &SO1mf[6].value,&ts44[1]},
    { &SO2mf[12].value,NULL}
    }; /* Then Side */
/* Rule 45: If Vel_Syn_Er=NSip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
 static struct rule_element_type is45[2] = {
    { &SI1mf[2].value, &is45[1]},
    { &SI2mf[6].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts45[2] = {
    { &SO1mf[10].value,&ts45[1]},
    { &SO2mf[8].value,NULL}
    }; /* Then Side */

/* Rule 46: If Vel_Syn_Er=NMip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=PVSop1,Vel2_adj=NVSop2; */
 static struct rule_element_type is46[2] = {
    { &SI1mf[1].value, &is46[1]},
    { &SI2mf[6].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts46[2] = {
    { &SO1mf[10].value,&ts46[1]},
    { &SO2mf[8].value,NULL}
    }; /* Then Side */

/* Rule 47: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=PLip2, Then
 Vel1_adj=PSop1,Vel2_adj=NSop2; */
 static struct rule_element_type is47[2] = {
    { &SI1mf[0].value, &is47[1]},
    { &SI2mf[6].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts47[2] = {
    { &SO1mf[11].value,&ts47[1]},
    { &SO2mf[7].value,NULL}
    }; /* Then Side */

/* Rule 48: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=PMip2, Then
 Vel1_adj=PSop1,Vel2_adj=NSop2; */
 static struct rule_element_type is48[2] = {
    { &SI1mf[0].value, &is48[1]},
    { &SI2mf[5].value,NULL}
    };  /* If Side */
 static struct rule_element_type ts48[2] = {
    { &SO1mf[11].value,&ts48[1]},
    { &SO2mf[7].value,NULL}
    }; /* Then Side */
```

```
/* Rule 49: If Vel_Syn_Er=NLip1 & Syn_Er_Rt=PSip2, Then
 Vel1_adj=PSMop1,Vel2_adj=NSMop2; */
 static struct rule_element_type is49[2] = {
     { &SI1mf[0].value, &is49[1]},
     { &SI2mf[4].value,NULL}
     };  /* If Side */
 static struct rule_element_type ts49[2] = {
     { &SO1mf[12].value,&ts49[1]},
     { &SO2mf[6].value,NULL}
     }; /* Then Side */


/* list of all rules in rule base */
 static struct rule_type Rule_Bases[49] = {
     { &is1[0], &ts1[0], &Rule_Bases[1]},
     { &is2[0], &ts2[0], &Rule_Bases[2]},
     { &is3[0], &ts3[0], &Rule_Bases[3]},
     { &is4[0], &ts4[0], &Rule_Bases[4]},
     { &is5[0], &ts5[0], &Rule_Bases[5]},
     { &is6[0], &ts6[0], &Rule_Bases[6]},
     { &is7[0], &ts7[0], &Rule_Bases[7]},
     { &is8[0], &ts8[0], &Rule_Bases[8]},
     { &is9[0], &ts9[0], &Rule_Bases[9]},
     { &is10[0], &ts10[0], &Rule_Bases[10]},
     { &is11[0], &ts11[0], &Rule_Bases[11]},
     { &is12[0], &ts12[0], &Rule_Bases[12]},
     { &is13[0], &ts13[0], &Rule_Bases[13]},
     { &is14[0], &ts14[0], &Rule_Bases[14]},
     { &is15[0], &ts15[0], &Rule_Bases[15]},
     { &is16[0], &ts16[0], &Rule_Bases[16]},
     { &is17[0], &ts17[0], &Rule_Bases[17]},
     { &is18[0], &ts18[0], &Rule_Bases[18]},
     { &is19[0], &ts19[0], &Rule_Bases[19]},
     { &is20[0], &ts20[0], &Rule_Bases[20]},
     { &is21[0], &ts21[0], &Rule_Bases[21]},
     { &is22[0], &ts22[0], &Rule_Bases[22]},
     { &is23[0], &ts23[0], &Rule_Bases[23]},
     { &is24[0], &ts24[0], &Rule_Bases[24]},
     { &is25[0], &ts25[0], &Rule_Bases[25]},
     { &is26[0], &ts26[0], &Rule_Bases[26]},
     { &is27[0], &ts27[0], &Rule_Bases[27]},
     { &is28[0], &ts28[0], &Rule_Bases[28]},
     { &is29[0], &ts29[0], &Rule_Bases[29]},
     { &is30[0], &ts30[0], &Rule_Bases[30]},
     { &is31[0], &ts31[0], &Rule_Bases[31]},
     { &is32[0], &ts32[0], &Rule_Bases[32]},
     { &is33[0], &ts33[0], &Rule_Bases[33]},
     { &is34[0], &ts34[0], &Rule_Bases[34]},
     { &is35[0], &ts35[0], &Rule_Bases[35]},
     { &is36[0], &ts36[0], &Rule_Bases[36]},
     { &is37[0], &ts37[0], &Rule_Bases[37]},
     { &is38[0], &ts38[0], &Rule_Bases[38]},
     { &is39[0], &ts39[0], &Rule_Bases[39]},
     { &is40[0], &ts40[0], &Rule_Bases[40]},
     { &is41[0], &ts41[0], &Rule_Bases[41]},
     { &is42[0], &ts42[0], &Rule_Bases[42]},
     { &is43[0], &ts43[0], &Rule_Bases[43]},
     { &is44[0], &ts44[0], &Rule_Bases[44]},
     { &is45[0], &ts45[0], &Rule_Bases[45]},
     { &is46[0], &ts46[0], &Rule_Bases[46]},
     { &is47[0], &ts47[0], &Rule_Bases[47]},
```

```
    { &is48[0], &ts48[0], &Rule_Bases[48]},
    { &is49[0], &ts49[0],NULL}
    };

/* Initialize the DSC-1 controllers */

/* Open paths to both channels */
  for (chan=0; chan<=1; chan++)
  {
    strcpy (string, "/");
    strcat (string, moname[chan]);
    if ((motor[chan] = open (string, S_IREAD+S_IWRITE))
== -1)
      printf ("Error opening path to %s : error %d\n",
    string, errno);
    else printf ("Path number for %s is %d\n", string,
        motor[chan]);
  }


  /* Initialise channel parameters */
  for (chan=0; chan<=1; chan++) {

  if (motor[chan] != -1)
  {
    if ((err = _ss_sleep (motor[chan], (1<<31)+256)) ==
                                    -1)
     printf ("Error settimg sleep time on %s : error
    %d\n",moname[chan], errno);
     else printf ("Sleep time set to 1 sec on %s\n",
        moname[chan]);

   if ((err = _ss_tries (motor[chan], 50)) == -1)
     printf ("Error setting number of tries for dsc1
signal on %s\n", moname[chan]);
     else printf ("Number of tries for dsc1 signals on
      %s set to 50\n", moname[chan]);

   if (!data_ready (motor[chan]))
   { if (errno == E_NOTRDY)
        printf ("No data ready from %s\n",
     moname[chan]);
        else printf ("Error testing %s for data ready :
           error %d\n",moname[chan], errno);
   }
   else printf ("Data ready from %s\n", moname[chan]);

   if (!rdy_for_data (motor[chan]))
   { if (errno == E_NOTRDY)
     printf ("Not ready to receive data on %s\n",
 moname[chan]);
  else printf ("Error testing %s for rfd : error
        %d\n", moname[chan], errno);
}
else printf ("Ready to receive data on %s\n",
   moname[chan]);

if ((err = _ss_sbase (motor[chan], sigsize*(chan+1)))
     == -1)
  printf ("Error setting up signal base value on %s :
 error %d\n", moname[chan], errno );
else printf ("Signal base set to %d on %s\n",
 sigsize*(chan+1), moname[chan]);
_ss_sigon (motor[chan]);  /* Enable signals */
```

```
status[chan] = _gs_stat (motor[chan]);
error[chan] = _gs_errs (motor[chan]);
printf ("Status of %s is %04X; error status is
%04X\n",moname[chan], status[chan],
                        error[chan]);


send_cmd (motor[chan], _setpc);
send_cmdp (motor[chan], _setkp, kp[chan]);
send_cmdp (motor[chan], _setki, ki[chan]);
send_cmdp (motor[chan], _setkd, kd[chan]);
send_cmdp (motor[chan], _setkv, kv[chan]);
send_cmdp (motor[chan], _setkf, kf[chan]);
send_cmdp (motor[chan], _setvel, sv[chan]);
send_cmdp (motor[chan], _setacc, sa[chan]);
send_cmd (motor[chan], _initdac);
send_cmd (motor[chan], _setzero);


 /* Set up data storage buffer */
if ((err = _ss_dbuf (motor[chan], all_data,
dmdata[chan], dmsize)) == -1)
 printf ("Error  %d setting up buffer\n",  errno );

if ((err = send_cmd (motor[chan], _getconst)) != -1)
{ for (i=0; i<7; i++)
  { if ((code = read_data (motor[chan], &value)) !=
-1)
      printf ("%d %s = %-7d\n", i, parname[i],
      value);
    else printf ("Error reading parameter %d : error
     %d\n", i, errno);
  }
}

/* Set up the position broadcast facilities */

 send_buf (motor[0], posbuf, psize); /* preload data */
   for (chan=0; chan<=1; chan++)
   send_cmdp (motor[chan], _execmap, 0);/* put both
                  channels into map mode */


}        /* END INICIALIZE  */
```

A3.2.2 Subroutines of DSC-1

```
read_status (chan)
int chan;
{    if ((status[chan] = _gs_stat (motor[chan])) == -1)
         printf("Error reading %s status : error %d\n",
              moname[chan], errno);
     if ((error[chan] = _gs_errs (motor[chan])) == -1)
         printf("Error reading %s error status : error
%d\n", moname[chan], errno);
     printf ("Status of %s is %04X; error status is
   %04X\n",moname[chan], status[chan],
error[chan]);
}
```

```
wait_for_idle (path)
int path;
{     int stat;
      do
      {     tsleep (20);
            stat = _gs_stat (path);
            errchk ();
      }
      while ((stat & 0x7f00) != 0 );
      return (0);
}

/* Send command without parameter to dscl */
send_cmd (path, code)
int path;
unsigned char code;
{     int err;

      /* Send command to dscl */
      if ((err = _ss_send (path, code)) == -1)
            printf ("Error sending %s command to %s : error
  %d\n",cmdname[code], name[path], errno );
      errchk ();
      return (err);
}

/* Send command with parameter to dscl */
send_cmdp (path, code, param)
int path, param;
unsigned char code;
{     int err;

      /* Send command to dscl */
      if ((err = _ss_send(path, code|flagbit, param)) ==
 -1)
            printf ("Error sending %s command to %s : error
       %d\n",cmdname[code], name[path], errno );
      errchk ();
      return (err);
}

/* Send data to dscl - for example profile data points */
send_data (path, value)
int path, value;
{     int err;

      /* Send data to dscl */
      if ((err = _ss_send (path, flagbit, value)) == -1)
            printf ("Error sending data to %s : error
  %d\n", name[path], errno );
      errchk ();
      return (err);
}

/* Read command/data value from dscl */
read_data (path, valptr)
int path, *valptr;
{     short code;

      /* Read data from dscl */
      if ((code = _gs_read (path, valptr)) == -1)
            printf("Error reading data from %s : error
 %d\n", name[path], errno );
      errchk ();
      return (code);
}
```

```
data_ready (path)     /* Test path for data ready */
int path;
{     int err;

      errchk ();
      if ((err = _gs_rdy (path)) == -1) return (false);
      else return (true);
}

buffer_ready (path)     /* Test for buffer ready */
int path;
{     int err;

      errchk ();
      if ((err = _gs_prdy (path)) == -1) return (false);
      else return (true);
}

rdy_for_data (path)  /* Test if path is ready to receive data */
int path;
{     int err;

      errchk ();
      if ((err = _gs_rfd (path)) == -1) return (false);
      else return (true);
}

send_buf (path, ptr, size)  /* Set up broadcast position data buffer
*/
int path, *ptr, size;
{     int err;

      errchk ();
      while (!buffer_ready(path)) tsleep(10);
      err = _ss_pbuf (path, ptr, size);
      return(err);
}

trap (signal)
register int signal;
{ register int chan;
  chan = (int) signal / sigsize;
  if (chan == 0)
  {  close (motor[0]);
     close (motor[1]);
     exit (signal);
  }
  signal -= chan-- * sigsize; /* Get signal offset and decrement
 chan */
  if (signal > 0 && signal < 127) /* User signal */
     usig[chan] = signal;
else
{  signal -= 128; /* Subtract system signal offset */
   switch (signal)
   {     case statsig:
         {     status[chan] = _gs_stat (motor[chan]);
               sflag[chan] = True;
               break;
         }
         case errsig:
         {     error[chan] = _gs_errs (motor[chan]);
               eflag[chan] = True;
               break;
         }
         case abortsig:
```

```
{     abflag[chan] = True;
      break;
}
case flushsig:
{     flflag[chan] = True;
      break;
}
case refsig:
{     referr[chan] = _gs_rdsig (motor[chan],
                        refsig);
      rflag[chan] = True;
      break;
}
case wrapsig:
{     wrapcnt[chan] = _gs_rdsig (motor[chan],
                        wrapsig);
      wflag[chan] = True;
      break;
}
case snapsig:
{     snapdata[chan] = _gs_rdsig (motor[chan],
                          snapsig);
      pflag[chan] = True;
      break;
}
default:
{     close (motor[0]);
      close (motor[1]);
      exit (signal);
            }
      }
   }
}
errchk ()
{     register int chan, temp;
      for (chan=0; chan<=1; chan++)
      {     if (eflag[chan])
            {     eflag[chan] = False;
                  if ((temp = error[chan] >> 8))
                  {     if (temp < e_limit) temp = e_limit;
                        temp -= e_limit - 1;
                        printf ("Error on %s : %s\n",
                  moname[chan],  errname[temp] );
                  }
                  if ((temp = error[chan] & 0xff))
                        printf ("Error on %s : %s\n",
      moname[chan],  errname[temp] );
}
if (sflag[chan])
{     sflag[chan] = False;
      temp = status[chan];
      printf ("%d", chan+1); /* Print channel
                        number */
      if (temp & 0xff)
            write (1,"W ",2);
      else if (temp & motoroff << 8)
            write (1,": ",2);
      else if (temp & initmode << 8)
            write (1,"I ",2);
      else if (temp & stopmode << 8)
```

```
                write (1,"S ",2);
            else if (temp & mapmode << 8)
                write (1,"X ",2);
            else if (temp & prflmode << 8)
                write (1,"P ",2);
            else if (temp & movemode << 8)
                write (1,"M ",2);
            else if (temp & vcmode << 8)
                write (1,"V ",2);
            else write (1,"> ",2);
    }
    if (abflag[chan])
        { abflag[chan] = False;
          printf ("%s aborted\n", moname[chan]);
        }
        if (flflag[chan])
        { flflag[chan] = False;
          printf ("%s flushed input buffer on an
           error\n",moname[chan]);
        }
        if (rflag[chan])
        { rflag[chan] = False;
          printf ("%s reference error = %d\n",
moname[chan], referr[chan]);
        }
        if (wflag[chan])
        { wflag[chan] = False;
          printf ("%s wraparound count = %d\n",
 moname[chan], wrapcnt[chan]);
            }
            if (pflag[chan])
            { pflag[chan] = False;
              printf ("%s position snapshot = %d\n",
          moname[chan], snapdata[chan]);
            }
            if (usig[chan])
            { usig[chan] = 0;
            }
        }
        return (0);
}
```

Appendix III: References

[1] Viot, G., "Fuzzy Logic in C", Dr. Dobb's Journal, February 1993, pp40–49.

[2] "Digital Motor Control System, Programmer's Reference Manual", Quin Systems Ltd, Issue 10, December 1992.

Appendix IV
Modeling Digital Position Control Systems

A digital position control system is represented by the block diagram of Figure A4.1. The desired position, expressed in encoder quadrature counts is ***r***. This position is compared with the actual feedback, ***c***, and the position error, ***x***, is determined. The control block of the microprocessor–based motion controller amplifies the position error, ***x***, and filters it. The output of the filter is then applied to the DAC which generates the motor command ***v***. The motor and the drive are modeled together by the combined transfer function M(s). This is the transfer function between the motor command, ***v***, and the angular position of the motor, θ. The motor angular position is sensed by the encoder which generates two signals in quadrature, channels A and B. These two signals are then applied to the position decoder, which generates the position feedback, ***c***.

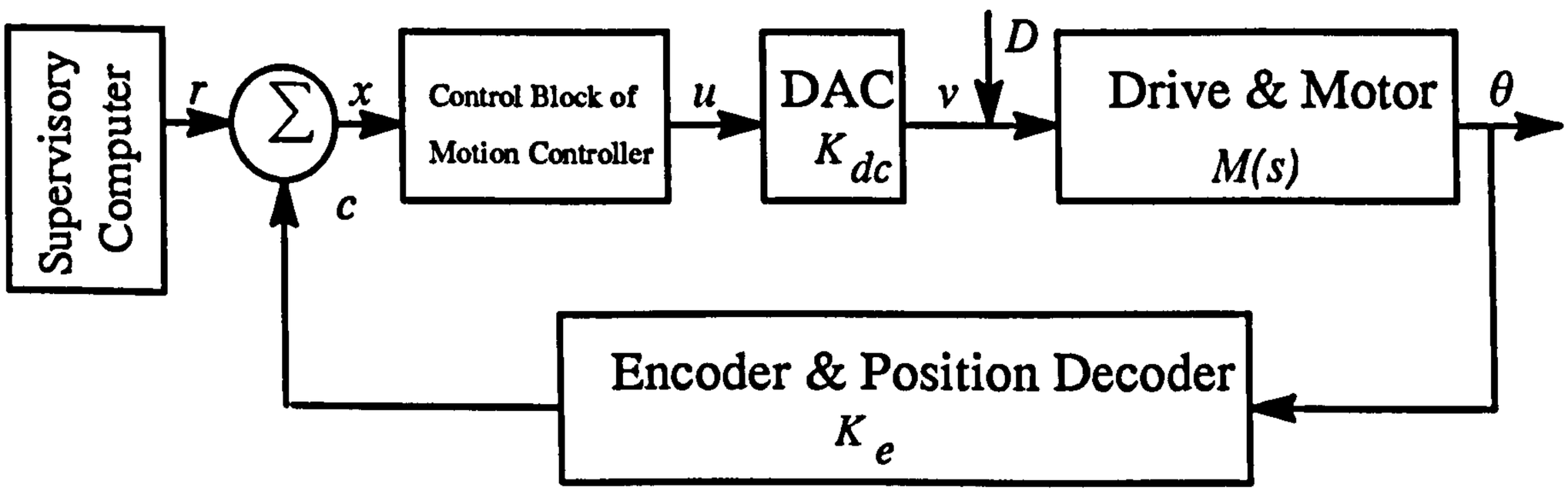


Figure A4.1 Modelling position control system elements

A4.1 Drive and Motor

The motors under consideration may be of different types. The most common motors are DC motors, of both the brushed and the brushless types. When a DC motor is driven by a voltage amplifier, the transfer function can be represented as:

$$M(s)=\frac{K_m}{s\,(\,sT_m+1\,)\,(\,sT_e+1\,)} \qquad [\,\frac{\text{rad}}{\text{volt}}\,] \qquad (A4\text{–}1)$$

The term s in the denominator indicates integration due to the fact that there is position feedback. The two time constants T_m and T_e are the mechanical and electrical time constants of the motor. When the electrical time constant is short, in comparison with the system response time, it can be ignored. In that case, M(s) can be simplified to the form:

$$M(s)=\frac{K_m}{s\,(\,sT_m\,+1\,)} \qquad [\,\frac{\text{rad}}{\text{volt}}\,] \qquad \text{(A4-2)}$$

When the motor is driven by an amplifier with current feedback, which acts as a current source, the corresponding transfer function can be represented as:

$$M(s)=\frac{K_m}{s^{\,2}} \qquad [\,\frac{\text{rad}}{\text{volt}}\,] \qquad (\,\text{A4-3})$$

A4.2 Digital to Analogue Converter (DAC)

The DAC generates a voltage *v*, which is proportional to *u*, the microprocessor output. In general, if the DAC output voltage is within $+V_m$, and the DAC has n bits, its gain, K_{dc}, is:

$$K_{dc}=\frac{2\,V_m}{2^{\,n}} \qquad [\,\frac{\text{volt}}{\text{count}}\,] \qquad \text{(A4-4)}$$

A4.3 Encoder and Position Decoder

The encoder and the position decoder generate a count, *c*, which is proportional to the motor angular position θ. Let the encoder line density be N lines per revolution. Due to the quadrature sensing, the position decoder generates 4N counts per revolution. This corresponds to feedback gain, K_e, of:

$$K_e=\frac{4\,N}{2\pi} \qquad [\,\frac{\text{count}}{\text{rad}}\,] \qquad \text{(A4-5)}$$

A4.4 Motion Controller

The motion controller operates by sampling the position of the motor at regular intervals, and calculating a motor demand signal according to some control algorithm. The algorithm used is of the following form.

$$u = k_p x_i + k_i \sum x_i + k_d (x_i - x_{i-1}) + k_v (c_i - c_{i-1}) + k_f (r_i - r_{i-1}) \qquad \text{(A4-6)}$$

where k_p = proportional gain constant

k_i = integral gain constant

k_d = differential gain constant

k_v = velocity feedback gain constant

k_f = velocity feed-forward gain constant

The dynamic behaviour of the system depends on these gain constants, and on the mechanical characteristics of the system being controlled. Tuning the control system to obtain good performance with a particular mechanical configuration requires set up of these gain constants.

Appendix IV: References

[1] Tal, J., "Motion Control by Microprocessors", Galil Motion Control, 1984.

[2] "Digital Motor Control System, Programmer's Reference Manual", Quin Systems Ltd, December 1992.

[3] Phillips, C. L. and Harbor, R. D., "Feedback Control Systems", Prentice-Hall International Editions, 1991.

[4] Olsson, G. and Piani, G., "Computer Systems for Automation and Control", Prentice Hall, 1992.

[5] Bollinger, J. G. and Duffie, N. A., "Computer Control of Machines and Processes", Addison-Wesley Publishing Company, 1988.

Appendix V
Definitions of UMC Terms

A5.1 General Definitions

Architecture

Architecture relates to the art of designing and realizing a complex system. *Architecture* is a powerful word, and is evocative of structure and style. The structure and interconnection of building blocks is the basis of *architecture.*

A strong motive for enforcing a consistent *architecture* is to enable design reuse. This is based on the reuse of architectural building blocks and frameworks.

Process

An executing program.

Reference Architecture

A *reference architecture* describes the function of system parts, as opposed to how they operate internally. A *reference architecture* is composed of a set of generic guide-lines, constraints which provide a framework together with definitions of generic building blocks.

It is important to put the consideration of *architecture* in the context of the control application being considered. For the reuse of control system building blocks to be worthwhile there must be common characteristics in the range of applications to be addressed by system users. Clearly there are big differences across machine control applications but there are also many major commonalities. These commonalities can be addressed by a *reference architecture.*

View

A *view* is a distinct aspect or dimension of an *architecture* which can be considered independently or coupled to other *views.*

A5.2 UMC Specific Definitions

Application Task — *Application tasks* are user defined *processes* which collectively describe the application of a particular *UMC machine*. The scope of each *application task* is defined by the application programmer.

Axis — An *axis* is a programmably positioned mechanical system with one degree of freedom. Any *axis* which interfaces to a UMC machine does so via an *axis handler*. One or more axes may be associated with one *axis handler*. Axes may be linear or rotary and may be powered in any appropriate way such as electrically, pneumatically or hydraulically. If an *axis* is to be used by a *UMC machine* it must be at least position controllable via a motion controller.

Axis Group — *Axes associated* with one or more *handlers* can be logically grouped together to achieve collective movement in many degrees of freedom (typically as part of a distributed manipulator). The axes of an *axis group* need not be mechanically coupled (i.e. may belong to separate kinematic chains). Theoretically there is no restriction on the number of axes in a *group* and individually handled sets of axes may be part of more than one *axis group*.

Channel — A *channel* is a unit of input or output. *Channels* are controlled and monitored by *ports*. Output *channels* can be set by appropriate *commands* via the *port handler* of the port to which the *channel* belongs.

Components — A collective term for the *events*, *handlers* and *tasks of a UMC machine*.

Emulation Handler — A special version of *handler* which is used to collect runtime data.

Event

Events provide the mechanisms for interprocess communication coodination and synchronisation within a *UMC machine*.

External Device

Any software (drivers, *processes*, etc.) or hardware (dedicated motion or I/O controllers etc.) which can be interfaced to a *UMC machine* via a *handler* (axes of motion or binary/analogue *ports* etc.).

Handler

Each *external device* to be accessed by a *UMC machine* must be interfaced to the higher levels of the runtime software via a *handler*. Two types of *handlers*, *axis handlers* (both single and multi-axis) and *port handlers* are currently defined. A *handler* provides the mechanism for interfacing an *external device* to a *UMC machine* in a unified and device independent manner. A specific type of *handler* must be written for each type of *external device*.

With a *handler* in place an *external device* is referred to as a *virtual device* and may then be controlled using a standard set of *UMC commands*. Some of these commands are totally generic and may be used to control all *external devices*, but many are designed to control a particular class of device. This concept is applicable to many types of proprietary building block such as PLCs, robot arms and intelligent tools or sensors. The ultimate aim is to identify common control requirements within each of these groups and create standard *UMC command* sets and *handlers* to suit.

Handler Information Module

An *information module* used for communication between *tasks* and *handlers*.

Handler Level

The handler level forms the bottom level of the *UMC reference architecture*. It contains all of the *handlers* and associated *handler information modules*.

Information Module

An *information module* is an area of memory used as a means of storing data which is to be shared between *processes*. Each *information module* is normally associated with a single *process* and provides the means of storing *process* variables and data in a form which is visible to the rest of the system. Data fields within the *information modules* have defined meanings and can be written to or read by the appropriate *processes*.

Location

A *location* is a named set of *axis* positions or coordinates used to specify a configuration or pose of an *axis group* associated with a *UMC machine*. *Locations* are used as a shorthand means of referencing by name and allow for modification of the axis position data by teaching or editing as a separate activity, without the need to modify the task.

Location Information Module

Location information module optionally exist for each *task* and contain position data for the axes being controlled by the *task*.

Machine Utility

A *process* which is used to load, modify and unload a *UMC machine*.

Machine Information Module

A data storage area used for machine configuration and runtime data.

Machine Information Utility

A UMC utility to retrieve and display information relating to the components of loaded *UMC machines*.

Machine Level

This is the top level of the *UMC reference architecture*. It contains the *machine information module* and machine utilities.

Port

A *port* is an *external device* which controls and monitors *channels*. A *port* is interfaced to a *UMC machine* via a *port*

handler. The configuration and number of *channels* is related to the specification of the external device, which might typically be a PLC or an intelligent I/O board.

Task — Application control logic is comprised of *processes* called *tasks*.

Task Information Module — *Task information modules* are optional and may contain application related information in a user defined format.

Task Level — This is the middle level of the UMC reference architecture. It contains all of the tasks, with associated task information modules and location information modules.

UMC — *Universal Machine Control*. This is the name by which the machine control methodology described in this document is known.

UMC Axis — A *UMC axis* refers to the *view* of and *axis* as seen by a *task*, which is the *view* presented to it by the *axis handler*. Hence the *task* has no knowledge of the actual *axis*, only a *view* of a *virtual device*. Replacing the *axis* device with an alternative which possesses comparable capabilities will not change this *view*. Hence the *external device* can be a hardware device, a software emulation, or a combination of both.

UMC Command — A *handler* is instructed to perform an operation when a *task* issues it with a *UMC command*. A set of *UMC commands* exists for each type of *handler*. The *handler* may or may not be able to execute an individual UMC command depending on the capabilities of the *external device*.

UMC Configuration Data — This is an off-line set of setup data for particular *UMC machines*. This data is used to produce all of the *information modules* when a *UMC machine* is loaded by the *machine*

	utility. It contains the information to "create" (and "uncreate") all the *information modules*, *tasks* and *handlers* in a *UMC machine*.
UMC Function Library	A library of UMC functions.
UMC Library Functions	Functions which *tasks* may call. These include functions to communicate *commands* to the *handlers*, functions to access the *information modules* and functions to manage *events*.
UMC Machine	A *UMC machine* consists of a number of concurrently executing *processes* together with mechanisms for communication, co-ordination and synchronisation of the *processes*.
UMC Port	See *UMC Axis*.
UMC System Boundary	A UMC system consists of building blocks which conform to the *UMC reference architecture*. The *UMC system boundary* encompasses these building blocks and separates them from anything not conforming to the *reference architecture*.
Utility Task	*Utility tasks* are generic *tasks* designed to be used in unmodified form. They perform device independent activities and serve as development aids during *handler* or machine development. A *utility task* can temporarily replace an unwritten user *task* and if appropriate, may be used in the final machine.
Virtual Device	This is the *view* a *task* has of an *external device*, when communicating with the *handler* which interfaces to the *external device*.

Appendix V: References

[1] Harrison, R., et al., "UMC Conceptual Specification", 4th Working Draft, June 2 1992.