UNIVERSITY OF MANITOBA

A Microcomputer based Machine Controller for Contouring Applications.

by

CHRISTOPHER G. GADSBY

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements for
the Degree of Masters of Science
in Mechanical Engineering.

Copyright
The University of Manitoba

December, 1986 

A MICROCOMPUTER BASED MACHINE CONTROLLER FOR CONTOURING APPLICATIONS

BY

CHRISTOPHER G. GADSBY

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

© 1987

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

ABSTRACT

A microcomputer based machine controller which is intended for contouring machine applications is presented. The sampled data controller is based largely on standard hardware and was developed by using FORTRAN as the main programming language. The flexibility provided by the high level programming language conveniently allowed the use of digital compensation and real time path modification to significantly improve the performance of the controller. Some experimental results are presented which underscore the effectiveness of the presented control scheme.

Acknowledgments

First and foremost, I would like to thank Drs. S. Balakrishnan and N. Popplewell for acting as my advisors during my course of graduate study. Their guidance and support provided a healthy medium for the research detailed in this Thesis.

Many hands were required to make this project a success and I extend my sincere thanks to each and every person: Mr. D. Schaldemose for his extensive work on the electronic interfacing, Mr. Ron Crampton for making the motor mounts, Mr. M. Kapitoler for setting up the mill table, and Mr. J. Shewchuk for making the encoder mounts.

In addition I must thank Mr. D. R. Young, and Mr. O. B. Wolfe for providing valuable dialogue on the programming aspects of this project. In addition, Mr. Wolfe provided a useful subroutine which was incorporated into the control software.

Last but not least, my thanks and best wishes are extended to Mr. T. Kostyniuk who aided in certain experiments and who will be continuing in this area of research.

Dedication

This thesis is dedicated to my parents, who encouraged me throughout my education, and especially to my Father who shared with me his fascination of the mechanical world.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Nomenclature

Block	Single line of a part program.
BLU	Basic Length Unit.
CAD/CAM	Computer Aided Design, Computer Aided Manufacturing
CLU	Control Loops Unit
CNC	Computer Numerical Control
DAC	Digital to Analoque Converter
DDA	Discrete Differential Analyzer
DPU	Data Processing Unit
EIA	Electronic Industries Association
I/O	Input/Output
MCU	Machine Control Unit
NC	Numerical Control
τ	Loaded Motor Time Constant

CHAPTER 1

INTRODUCTION

1.1 Historical Background

The introduction of Numerical Control (NC) in the early 1950s ushered in a new era in automation. Numerically controlled machine tools, typically lathes and milling machines, offered significant cost savings over traditional hand operations. These cost savings were realized largely by a NC machine's ability to consistently produce complex parts with high accuracy. In the early days of NC, a large lot size was required to justify its cost. However, this minimum economical lot size has been reduced steadily by the improvement of NC technology. Assuming a part of moderate complexity, the production of a single part will offer cost savings over traditional methods when NC technology is used in conjunction with a CAD/CAM computer support package.

NC equipment has been defined by the Electronic Industries Association (EIA) as:

" A system in which actions are controlled by the direct insertion of numerical data at some point. The system must automatically interpret at least some portion of

this data." Early or pre-computer controllers were constructed with sophisticated hard-wired control circuits. These circuits largely depended upon a circuit known as the discrete differential analyzer (DDA). The versatility of such circuits was sufficient that entire control circuits, including linear, circular, and parabolic interpolators, could be constructed in a hardwired manner. The circuits were complex and, with the advent of microcomputers in the early 1970's, numerical control became computer numerical control (CNC). The computer was used originally to replace part of the hardwired circuitry of the controller in order to improve reliability.

The degree of sophistication of modern CNC controllers varies greatly. Some systems still use a microcomputer which merely mimics the hardwired circuitry. On the other hand, other modern controllers have increased the duties of the computer to include essentially all the control tasks.

1.2 Classification of NC Machines

Numerically controlled machines may be classified into various categories according to [1]:

i) the type of machine, point-to-point versus contouring,
ii) the structure of the controller, hardware based NC versus CNC,

iii) the programming method, incremental versus absolute; or
iv) the type of control loop, closed versus open loop control.

A point-to-point machine is one in which only the final location of a programmed move is important and not the complete path. Machines which do not perform material removal during motion can fall into this classification. A drill is typical of a point-to-point machine in which the NC controller moves the table to a prescribed point for the drilling operation. Most machine tools, however, are the contouring variety where the whole path of motion is important because material removal occurs during all the motion. A milling machine or a lathe, for instance, represents a contouring type machine.

The classification of a machine tool as NC or CNC depends purely on whether a computer is employed. Furthermore, the term NC is often ambiguous because it is frequently used instead of the more precise term, CNC. It is possible to further subclassify a machine controller by the type of control scheme adopted. These subclassifications will be detailed later.

Machine tools can be programmed by using incremental or absolute (or sometimes both) programs. An absolute program references all dimensions from a defined starting point whereas an incremental program defines the endpoint of the present move from the

endpoint of the previous move.

An open loop controller does not receive a feedback signal to determine the actual position of a machine's axes. This type of controller has low performance and typically uses stepping motors for actuation. Closed loop controllers, conversely, are much more common and utilize feedback techniques for higher accuracy, feedrates, and accelerations.

The remainder of this thesis will be concerned with the development of an inexpensive, closed-loop, contouring CNC system. This classification of machines offers the greatest versatility and the highest performance.

1.3 Structure of a NC machine tool

As shown in Figure 1.1, an NC or CNC machine tool contains a Machine Control Unit and the Machine Tool itself. The control unit contains all the 'intelligence' of the system and controls the actuators, usually electric motors, of the machine tool. Such a unit contains two main components, a Data Processing Unit (DPU), and a Control Loops Unit (CLU).

The fundamental task of the Data Processing Unit is to provide a sequence of reference speeds, as a function of time, for each axis. The references are

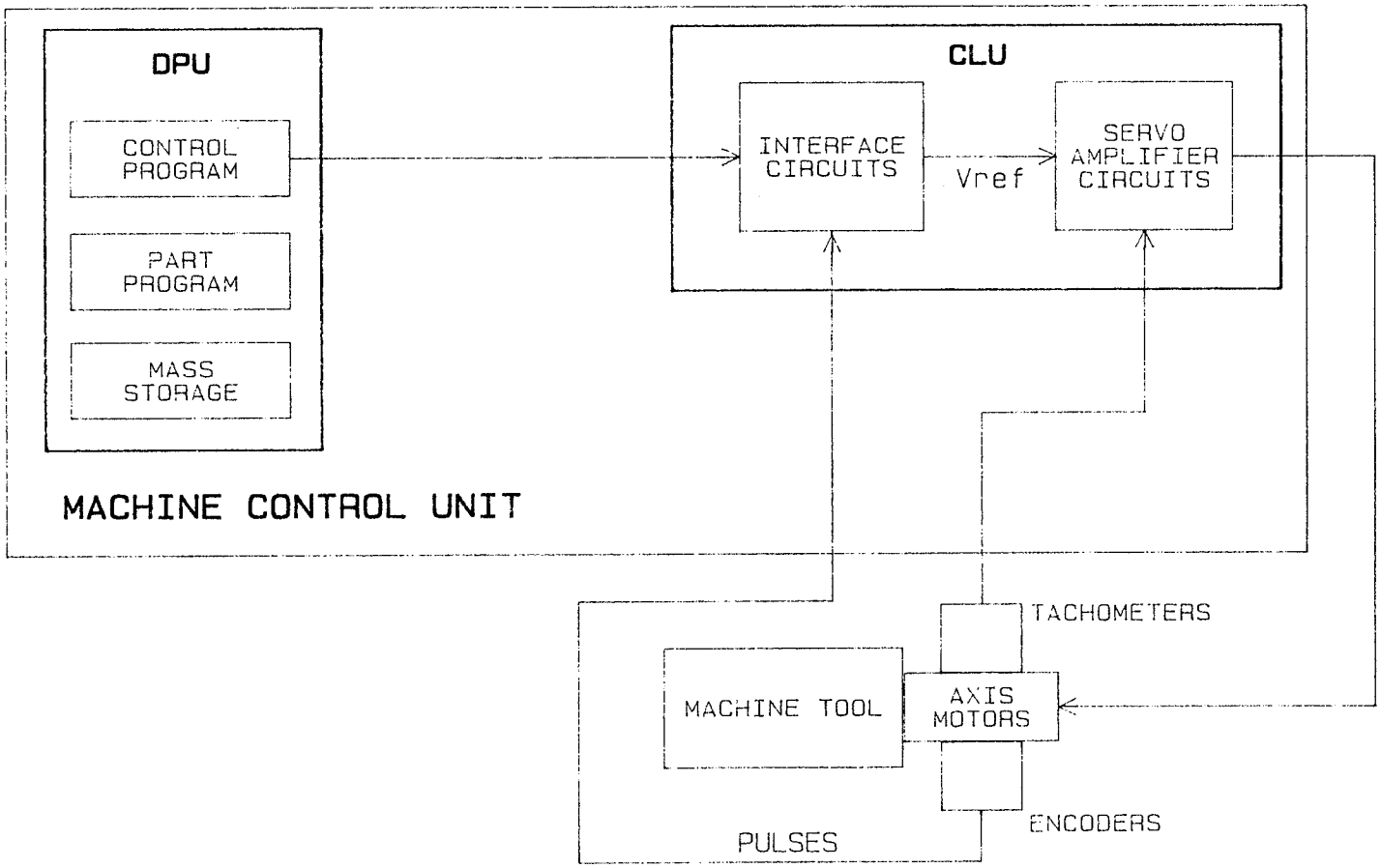
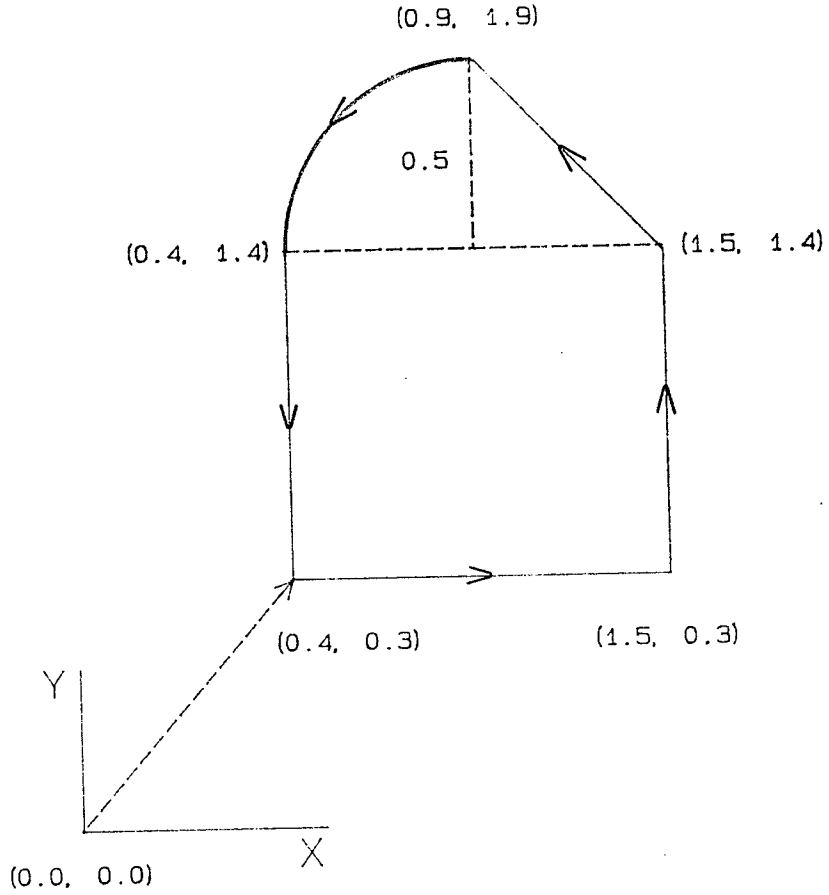


FIGURE 1.1 A CNC MACHINE TOOL'S BASIC STRUCTURE

chosen to specify the machine toolbit's desired path with due regard for the chosen feedrate (toolbit movement speed) along the contour. In general, several axes must move in a coordinated manner to obtain a desired toolbit motion and this process of producing a contour by component axis motion is called interpolation.

The desired toolbit path is provided by a Part Program. The data program specifies the toolbit path required to produce the desired part. The toolbit's path is specified by codes representing an ordered set of linear and circular arc segments. Each line of the program is called a block. Figure 1.2 portrays a simple cutter path geometry and a partial program. Each program line specifies the endpoint location of a linear or circular cutter motion and the 'G' code merely specifies the path between the present position and the motion's endpoint. The motion of the cutter is expressed as integers in the dimensions of Basic Length Units (BLUs). For example, a motion of 1 inch would be represented by 10000 BLUs on a machine with a BLU size of 0.0001 inches. A BLU may be considered the minimum resolution of machine toolbit motion.

The Control Loops Unit drives the axes motors so



Bracketed values represent the absolute coordinates of the motion segment endpoints.

```
N01 G01 X 04000 Y 03000 I00000 J00000
N02 G01 X 15000 Y 03000 I00000 J00000
N03 G01 X 15000 Y 14000 I00000 J00000
N04 G01 X 09000 Y 19000 I00000 J00000
N05 G03 X 04000 Y 14000 I00000 J05000
N06 G01 X 04000 Y 03000 I00000 J00000
N07 G01 X 00000 Y 00000 I00000 J00000
```

The N codes specify the part program line number.

The G codes specify the cutter path:

- G01 implies linear motion,
- G02 implies clockwise circular motion, and
- G03 implies counterclockwise circular motion.

The X and Y values specify the cutter endpoint positions absolute coordinates in the dimensions of BLUs. For the example toolpath shown, the machine has a BLU size of 0.0001 inches.

The I and J values represent the absolute values of the offset from the initial point of a circular arc motion to the centre of the arc. The I and J parameters are only used for circular motion.

FIGURE 1.2 CUTTER PATH PROGRAMMING

that the machine's toolbit travels along the path specified by the Data Processing Unit. The Control Loops Unit uses the speeds passed by the Data Processing Unit to provide reference positions. In order to intercept the reference positions, the CLU receives feedback signals from each axis and uses them to determine the actual position of each axis. The actual positions are compared to the reference positions. Differences between the reference and actual axis positions are used as speed reference signals by the axis motor amplifiers. Notice that the described differences represent the position errors of the system and, hence, the motor amplifiers may be considered to be position error driven. This error driven aspect of the axis drives indicates that each axis of the drive system may be considered a traditional error-driven closed loop system.

Performance limitations of a machine drive are linked directly to the error driven nature of a closed loop servo system. As the motors of a system may be considered to be driven by the difference, or error, between the reference and actual axis positions, a finite position error must exist whenever the axis motor is in motion. The use of classical control theory provides a way of predicting and minimizing this

position error.

1.4 Modern Machine Controllers

Modern machine controllers are generally dedicated to a particular type of machine tool. Consequently, they offer very little flexibility because a user cannot customize the machine tool for a given application. It would be more desirable, in principle, to have an 'open' design which would allow the user to completely change the controller's personality to suit the application at hand. Therefore, an ideal controller should allow the use of different control schemes, varying from simple contouring to adaptive or optimum control capabilities. Such flexible controllers could be programmed in unusual coordinate frames like those in robots [2] or in more traditional Cartesian coordinates.

It is possible presently to purchase commercial controllers which have limited flexibility. These controllers are intended typically for retrofitting hand-operated milling machines for CNC use. The major limitation of such control units is their expense and lack of flexibility. These devices are targeted directly at the traditional machine tool market and offer limited flexibility for use in traditional machine applications. Furthermore, manufacturers of these

retrofit control units construct the hardware and write the software for their own computer system. This, 'from the ground up' approach tends to add to the final purchase cost because of high development costs. (It would be cheaper to use existing software development tools and standard hardware when developing a machine control unit.)

1.5 Thesis Goals

The goal of this thesis was to design and construct a machine control system which offers flexible machine control at a cost below $6,000, excluding the servo motors and amplifiers. To minimize the cost, common off-the-shelf hardware was utilized as far as possible. Existing computer languages and compilers were adopted to maximize flexibility yet minimize the software development time.

Software was implemented largely in the high level language of FORTRAN to allow simple programming and the use of FORTRAN's powerful mathematical capabilities. Unfortunately, FORTRAN does not have powerful interface capabilities and low level input output support was provided by ASSEMBLER subroutines. Both the FORTRAN and the ASSEMBLER software developed were compatible, allowing the two languages to communicate easily.

The control unit was based on a common IBM PC or compatible microcomputer, supported by an Intel 8088 central processor and Intel 8087 math coprocessor. Although a genuine IBM PC was used for the experimental apparatus, any true compatible could easily be used. The IBM microcomputer was chosen because of the complementary wide array of support software offered for this unit which sells for less than $2500 excluding the required interface cards. The choice of FORTRAN allowed the control program to employ real time, floating point arithmetic with the 8087 support. Floating point calculations not only simplified the programming of the controller but also significantly increased its versatility. The IBM microcomputer required only one custom interface card to control two axes of motion. Future refinements would allow the same two cards to control a total of four axes of motion. Most machine control systems use multiple processors in which a single processor is dedicated to each axis and a master processor exists to coordinate the motions along the axes. The developed control unit operated by using only a single processor, an Intel 8088, for all system tasks. This hardware simplification was possible by increasing the software sophistication of the system beyond that which is normally found in machine control

units. This sophistication included innovative mathematical techniques (which significantly improved the drive system's accuracy) and a thorough treatment of the practical aspects of servo system lag.

It is important to understand the intended market for the designed control unit. This controller was intended as an inexpensive unit for custom machine tool or retrofit applications. Costs were minimized by using advanced control techniques to offset the performance limitations offered by non-custom hardware. Also, the use of commercial languages offered maximum flexibility with least development time.

Chapter 2 of this thesis will introduce the techniques employed in the drive control and will present details of the mathematical control model. Chapter 3 will elaborate upon the actual construction of the designed controller. It should be noted, however, that the controller was implemented, for demonstrative purposes, as a two axis controller on a small milling machine. Performance tests will be described in which the validity of the control scheme was determined.

CHAPTER 2

DIGITAL-ANALOGUE CONTROL LOOPS FOR NUMERICAL CONTROL

2.1 Introduction

This chapter will provide a general introduction to two major control techniques commonly used for machine axes control. In addition, the advantages of sampled data drives will be highlighted and a detailed mathematical analysis of a sampled data drive will be presented. The analysis will indicate the advantages offered by a control scheme which requires the controller to use real-time, floating-point mathematical calculations.

The chapter will present the theoretical basis for the machine drive's control. The theory has been based on several inherent assumptions. The most important assumption is that the mechanical machine drive, itself, is ideal and does not suffer backlash, or the dynamic consequences of structural flexibility. This assumption is generally acceptable for a good quality machine tool. However, for completeness, Chapter 5 will provide a short discussion on the control of non-ideal machines.

2.2 Introduction to Control Schemes

2.2.1 Performance Considerations

It is imperative that a CNC controller be able to maintain a specified accuracy over the entire cutting speed range of the controller. Hence, accuracy is the major consideration in evaluating the performance of a machine tool controller [3]. The performance of a closed loop machine drive controller may be described effectively by two types of error, Position Errors and Path Errors.

Position errors are caused by the error driven nature of a closed loop controller and by the finite response time of an electro-mechanical servo drive. They are not usually of great concern because position errors often do not contribute to the inaccuracy of the cutter's path. For instance, consider a steady state line motion in which the reference position simply leads the actual position by a finite steady value. Although the cutter's actual position lags behind the reference position, the actual position still lies on the desired path.

Path errors are position errors in which the cutter is off the desired path. These errors are caused typically by different response characteristics of

interrelated motors. For example, consider Figure 2.1 where it is desired to accelerate a cutter along a 45 degree path from rest to some desired feedrate. If the motor driving the X axis motion responds more quickly to the reference speed than that for the Y axis, a path error will be generated as shown. Typically, if the interpolation routines are assumed accurate, most path errors can be attributed to a mismatch in related axis servo systems [3]. Chapter 4 will provide a discussion of the errors caused by the mismatch of interrelated axes servo systems. This Chapter will focus primarily on individual axis errors.

There are two major techniques employed in machine tool control. The oldest technique is the reference pulse technique which is a byproduct of the hardwired logic, NC machine tool. The other major technique is the sampled data one in which use is made of sampled data techniques for the control of the machine axes. The following sections will focus on these two techniques and will outline their respective advantages and disadvantages.

2.2.2 Reference Pulse Techniques [1]

Figure 2.2 shows a block diagram of a single axis

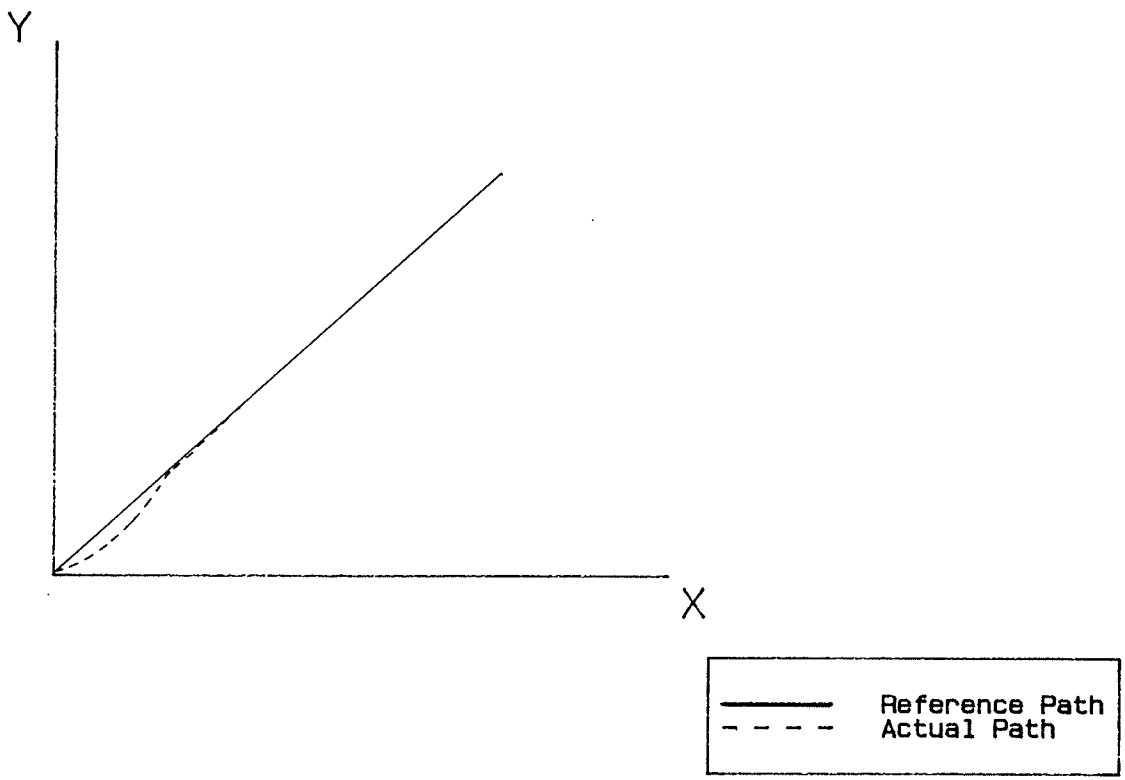


FIGURE 2.1 CONTOUR ERROR CAUSED BY SERVO MISMATCH

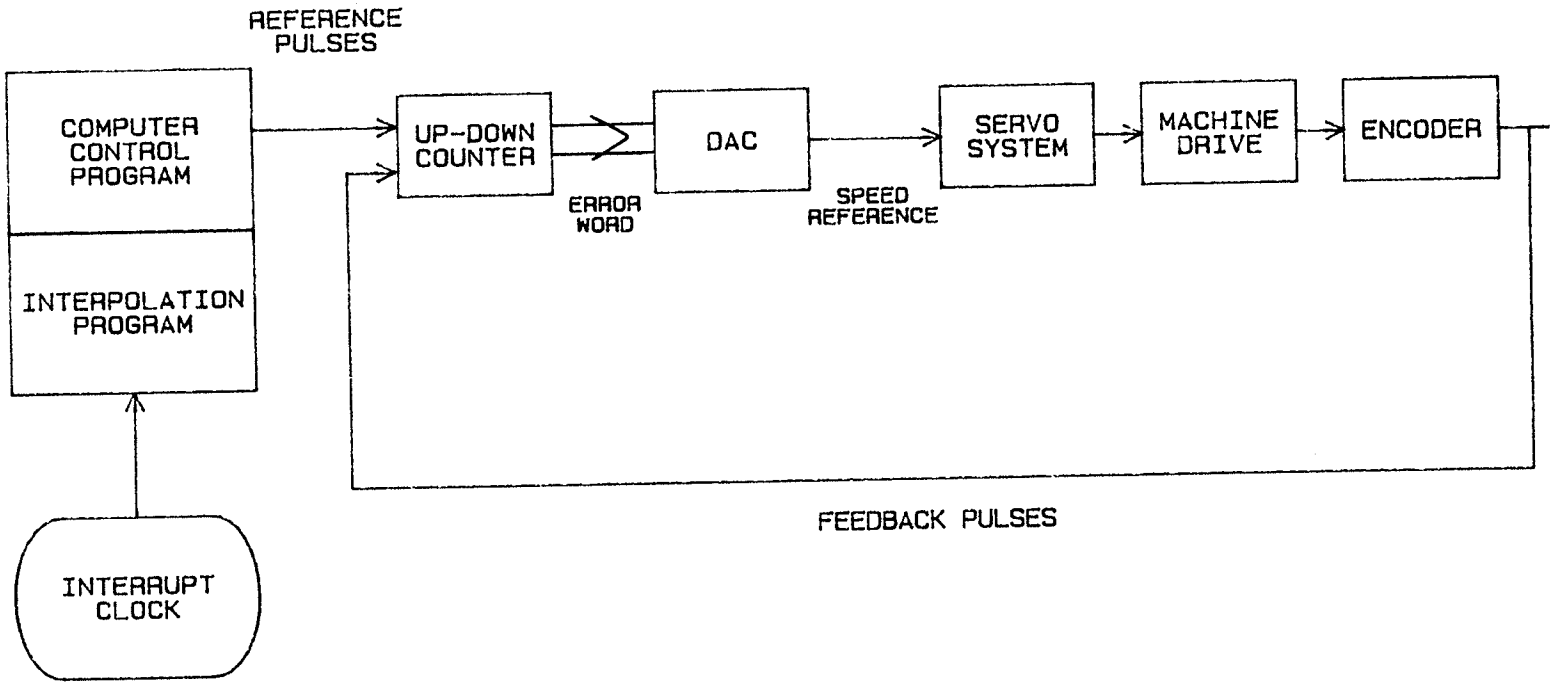


FIGURE 2.2 CONCEPTUAL BLOCK DIAGRAM OF THE REFERENCE PULSE TECHNIQUE

CNC system which employs the Reference Pulse Technique. The microcomputer contains a control program which provides a sequence of reference pulses to the up/down counter for each axis. The up/down counter also receives a series of feedback pulses from the optical encoder mounted on the axis motor. The optical encoder outputs a known number of pulses per revolution. The accumulated difference between these two pulse streams is equal to the position error for the axis. The error is converted to a voltage by the axis DAC. This voltage serves as a speed reference for the servo motor's amplifier and will cause the motor to rotate and reduce the position error. For a single axis and a given, constant steady state reference pulse frequency, the motor will turn at a constant speed with a finite position error between the reference and actual position. Furthermore, this position error will be related linearly to the reference pulse frequency. The position error will be just sufficient to cause the DAC to produce a motor speed which results in a feedback pulse frequency of the same frequency as the input stream.

The interpolation algorithms used in the Reference Pulse Technique are based upon an iterative strategy and

they are controlled by an external interrupt clock. A single iteration of the routine is executed at each interrupt which, depending upon the desired feedrate, may or may not provide a single output pulse to the axis motor. Each iteration can provide only a single pulse or no pulse. The maximum speed of the axis is reached when every iteration produces a reference pulse. Thus, the maximum speed along any axis is restricted by the frequency of the interrupt clock which, in turn, is constrained by the time required for the microcomputer to perform a single iteration of the interpolation routine. This speed constraint will be shown to be particular to the Reference Pulse Technique.

The Reference Pulse Technique exists as a carry-over from NC technology which was popular before the introduction of microcomputers in machine tool control. Traditionally, hard-wired circuits would perform the interpolation calculations by using discrete digital analyzers. References [1,4,5,6] provide details of interpolation algorithms for reference pulse control. Reference pulse interpolation has been restricted to linear, circular, and parabolic interpolations because of the limited algorithm types which may be implemented in hardwired logic.

The major drawback of the Reference Pulse Technique lies in the limitations imposed by the microcomputer's iterative speed. The maximum speed along an axis is constrained by this iterative speed because each iteration can only cause a single pulse to be output. For example, to obtain a feedrate of 30 inches per minute on a machine drive with a BLU of 10^{-4} inches, 300000 interrupts per minute are required. This feedrate corresponds to an interrupt frequency of 300000/60 or 5000 Hz. Interrupt frequencies of this order prevent complex mathematical computations being calculated in real time by present day microcomputer hardware. Thus, although a microcomputer has more computational power than a discrete differential analyzer, merely mimicing the hardwired logic does not offer an attractive solution because the limited intersample time, offered by the Reference Pulse Technique, precludes complex real time calculations. This deficiency imposes a practical limit on possible interpolation capabilities. The Sampled Data Technique will now be introduced and compared to the Reference Pulse Technique.

2.2.3 Sampled Data Techniques

Figure 2.3 represents the major components of a single axis, CNC system which employs a Sampled Data Technique. Such a CNC system uses the microcomputer as part of the control loop. The up/down counter of the Reference Pulse Technique is replaced now by a software-hardware counter. A small external counter is used to count the encoder pulses in a given sample period. (A single pulse is generally considered to be equal to one BLU [1] even though it will be shown later that an integer multiple of pulses equal to a single BLU is much better.) For the purpose of the present discussion, the traditional view will be considered in which a single encoder pulse equals a single BLU.

The number of pulses transferred from the external counter to the microcomputer represents the incremental position change, Δp, for that sample period. The microcomputer accumulates these increments and compares the accumulated position with the reference position determined by the interpolation routine during each sample period. The difference between the reference and accumulated positions represents the error in the drive position. The ensuing error word is scaled, passed to the DAC and causes the motor to rotate in a direction

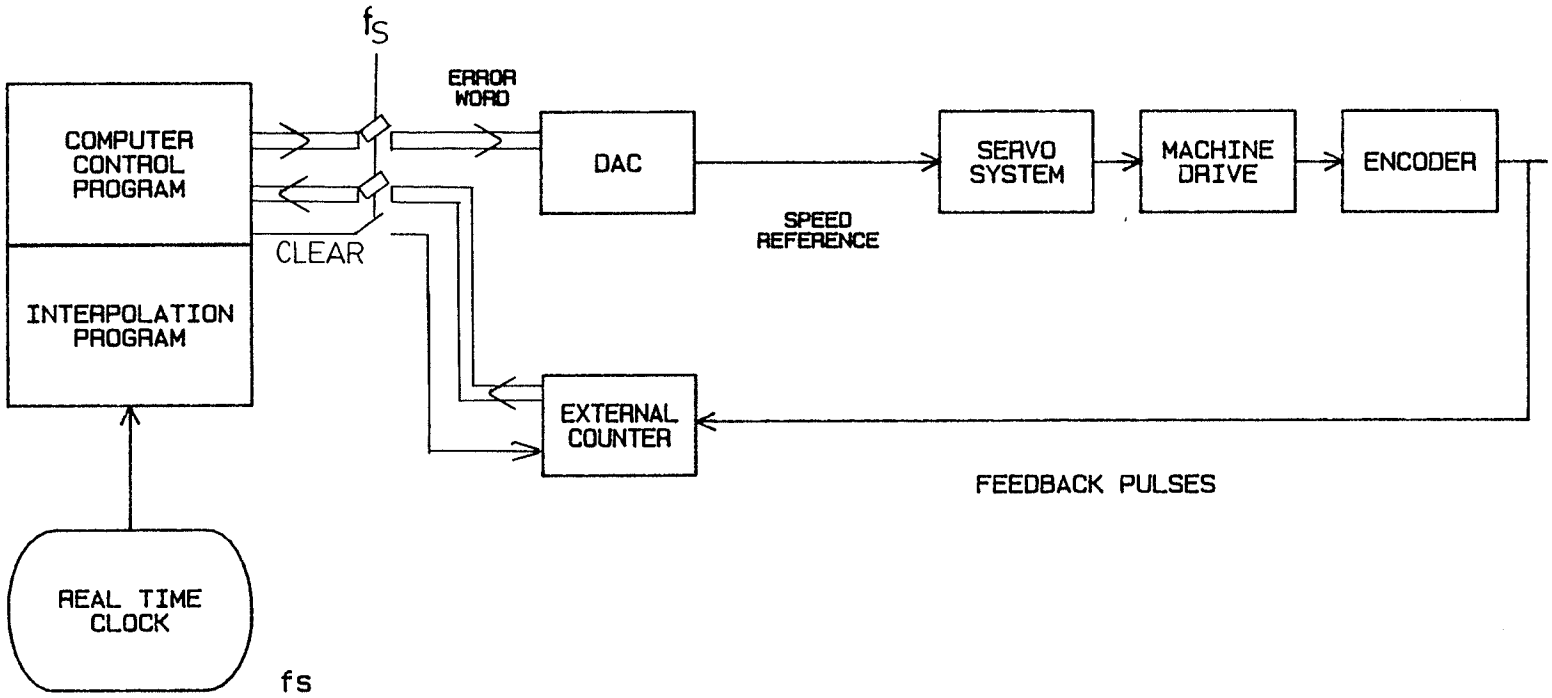


FIGURE 2.3 CONCEPTUAL BLOCK DIAGRAM OF THE SAMPLED DATA TECHNIQUE

which reduces the error.

Figure 2.4 shows a flowchart of the software required to implement a sampled data controller for a single axis. The Data Processing Unit reads a single block of the part program from a mass storage device, usually a tape or a disk drive, and passes the requested move to the appropriate interpolator. The interpolator calculates the corresponding feedrates for the axes and passes the reference speeds to the Control Loops Unit. The Control Loops Unit waits for an external interrupt and, when it occurs, the CLU loads the incremental position changes for each axis and adds them to the current axis position accumulators. New reference positions are calculated from the requested axis speeds and the resulting position errors are used as speed reference signals for the servo motor's amplifiers.

The sampling frequency, fs, employed in the Sampled Data Technique is considerably slower than the interrupt frequency of the Reference Pulse Technique. The maximum rotational speed of the motor is not limited by the sampling frequency because each iteration outputs an entire error word and not just a single pulse. The sampling frequency ranges typically between 10 and 100 Hz and is considerably slower than the interrupt clock

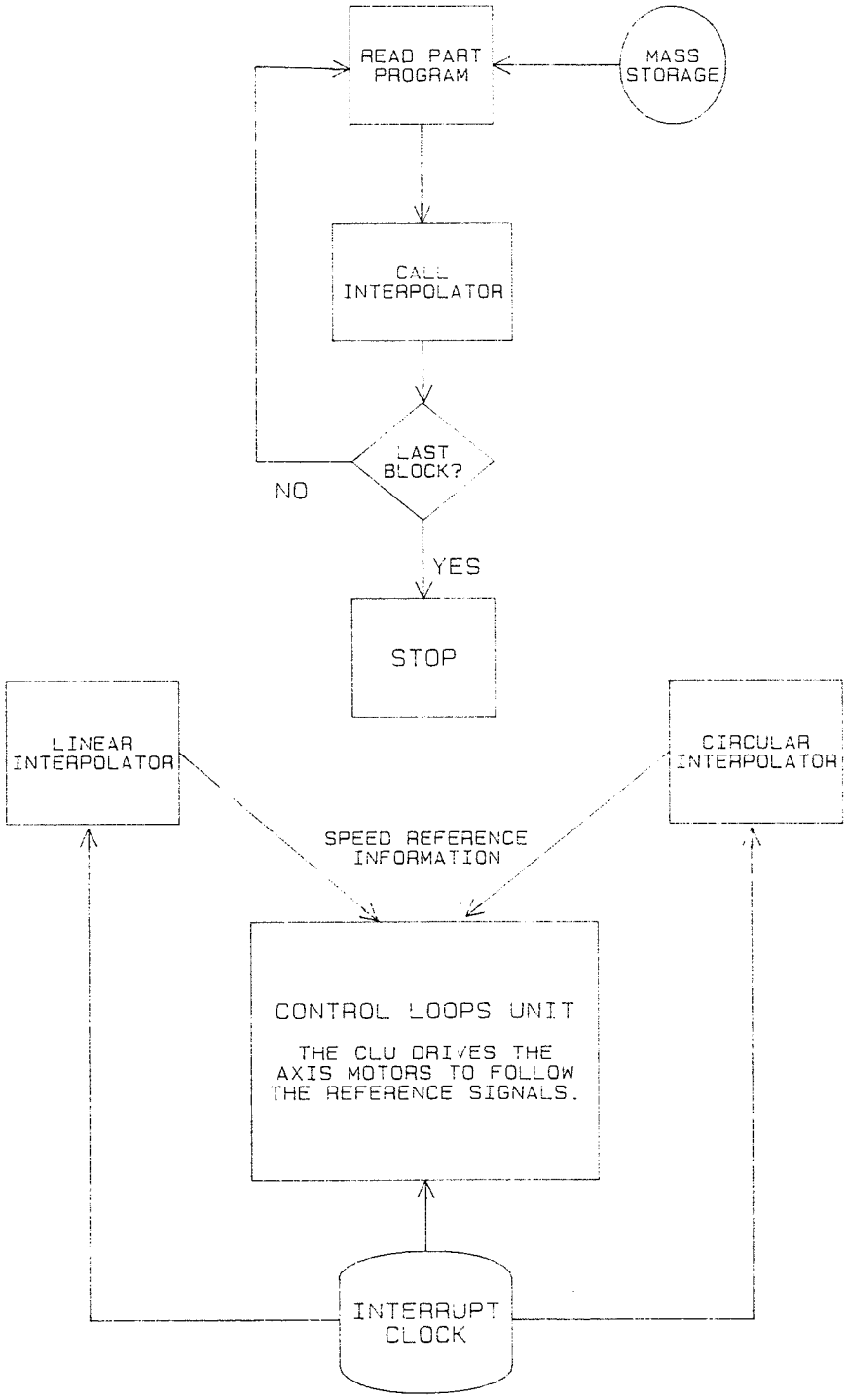


FIGURE 2.4 SOFTWARE SCHEME FOR A SAMPLED DATA CONTROLLER

frequency of the Reference Pulse Technique. The longer intersample time allows a comparatively complex interpolation scheme to be implemented in real time. The only disadvantage of the Sampled Data Technique is that the drive is sampled at a much lower frequency than the Reference Pulse Technique. This reduction causes the dynamic performance of the servo control system to be degraded slightly in comparison to the Reference Pulse Technique. However, this slight decrease is overshadowed by the advantage of having a significantly longer interrupt period. One important advantage is that the control loop is closed within the microcomputer. This implies that the microcomputer has available internally the axes position errors which can be used in the control calculations. Recall that the control loops are closed outside the microcomputer in the Reference Pulse Technique so that the actual angular orientations of the different axes are unknown to the microcomputer. The ability to observe the actual orientations will be shown to be extremely beneficial from a control viewpoint. Hence, the Sampled Data Technique was chosen for this project and the corresponding mathematical model will be developed next.

2.3 Sampled Data Control of a Single Axis

2.3.1 System Model

Figure 2.5 presents a detailed block diagram of a single axis, sampled data control system. Each element of the system will be considered in turn. The microcomputer compares the actual position, P, with the reference position, R. The resulting position error pulse count is scaled by a proportional factor, Kp, before being passed to the Digital to Analogue Converter (DAC). The DAC converts the error into an analogous voltage with a gain of Kd Volts/Pulse error. The DAC outputs this voltage to the motor drive which accepts it as a speed reference signal. The motor drive acts as a low pass filter and can be modelled, therefore, as a first order block with a time constant equal to that of the loaded drive [1,7]. By convention [8], the time constant shown in Figure 2.6 is the period required for the inertially loaded motor drive to attain 63.2% of a requested step change in speed. The gain of the motor's drive system Km, on the other hand, is represented in units of Rads/sec/volt. The encoder produces pulses with an encoder gain of Ke pulses/rad. Such pulses are accumulated in a small external counter during the intersample period. At each sampling instant, the

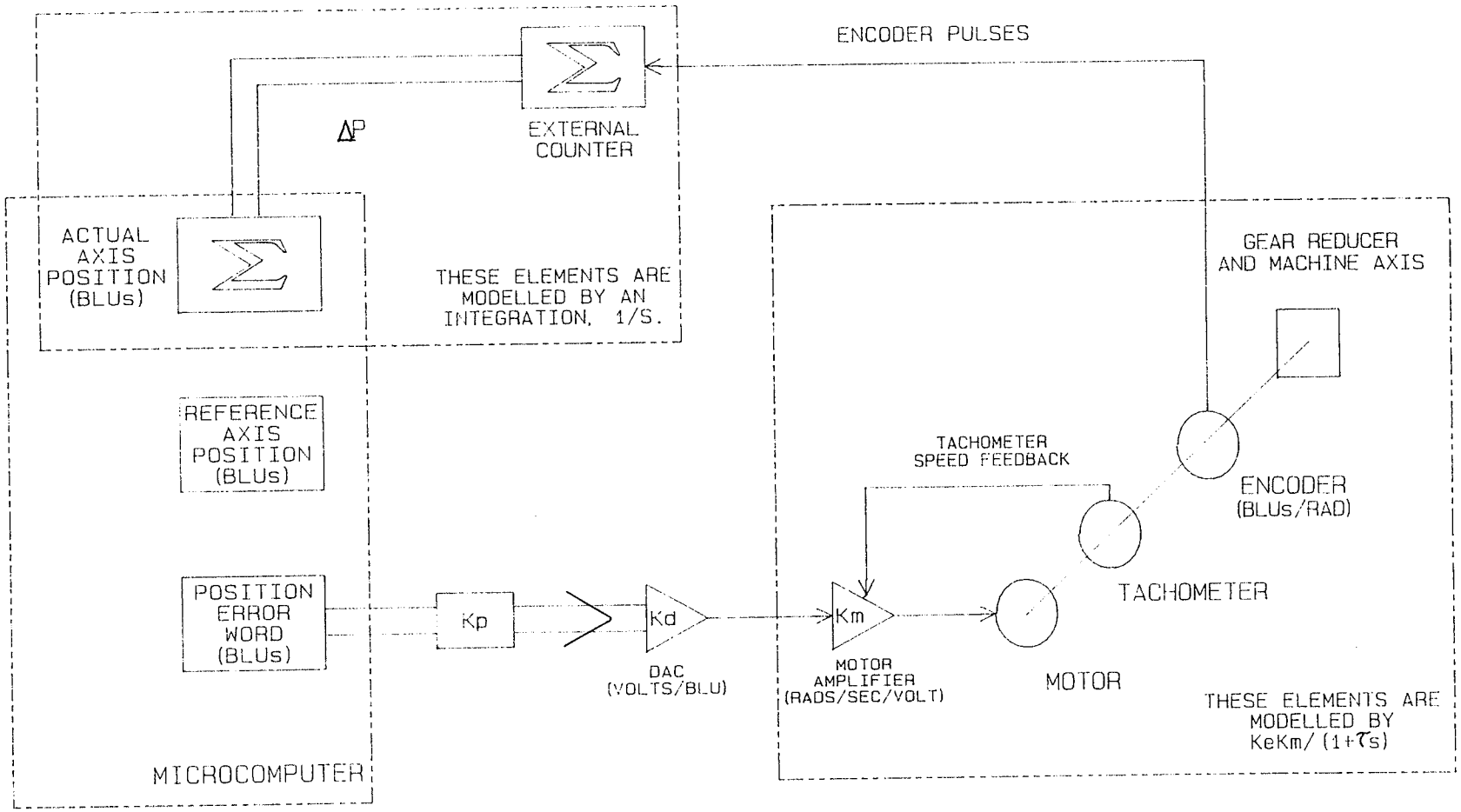


FIGURE 2.5 DIAGRAM OF THE DRIVE SYSTEM'S ELEMENTS

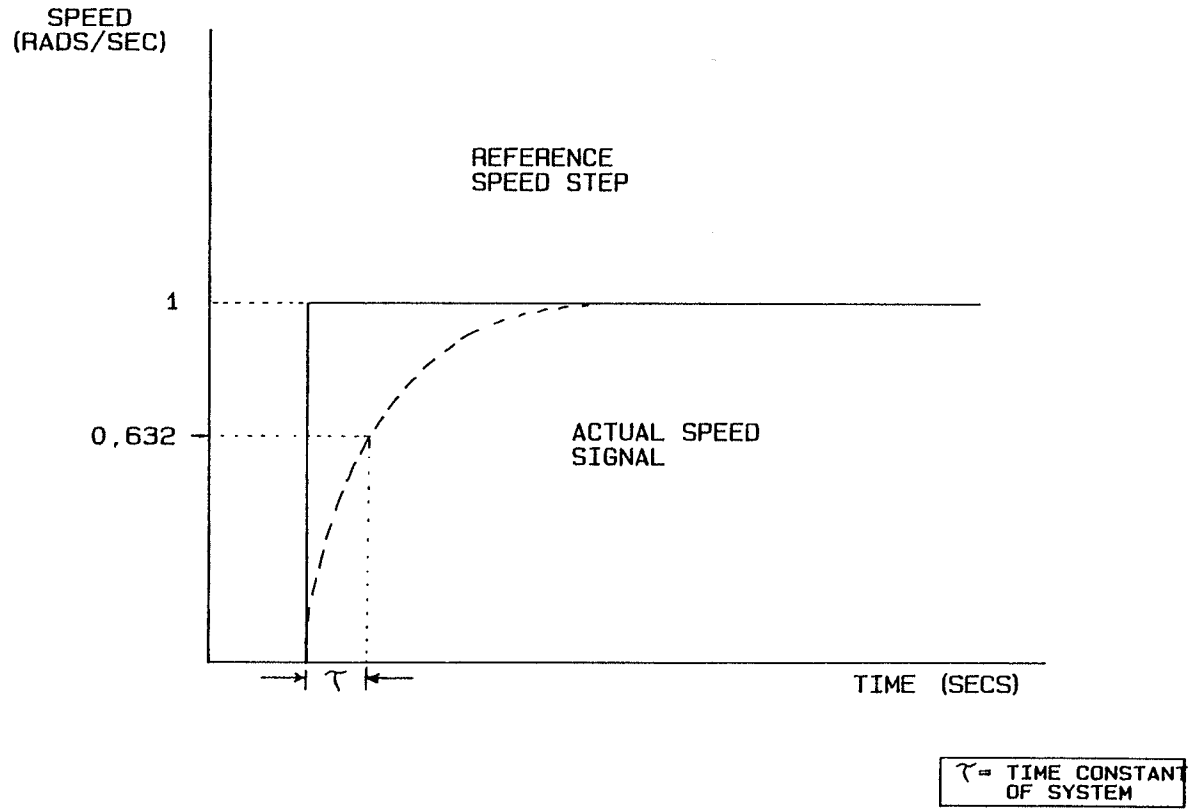


FIGURE 2.6 DETERMINATION OF THE MOTOR SYSTEM'S TIME CONSTANT

external counter value, Δp, is added to an absolute accumulator and then the external counter is cleared. Thus, the absolute accumulator contains the absolute position count. This process of accumulating incremental position changes is represented essentially by an integration.

Figure 2.7 presents the machine drive's block diagram. The first order representation of this system contains several approximations which must be considered separately. Typical servo amplifiers have internal loops which generally include a tachometer, a current limiting loop (to prevent demagnetization of the motor), a dynamic braking loop, and compensation loops to improve the speed response of the motor. With the exception of the current limiting loop and the braking loop, all loops tend to linearize the response of the motor over small reference speed changes. Large speed changes, however, tend to cause the current limitation loop to produce a disproportionately greater response time which is a non-linear effect. Also the dynamic braking loop forces the motor to generate electricity when a request is made for the motor to slow down. This electricity is converted to heat through a resistance so that the braking introduces another non-linear effect into the

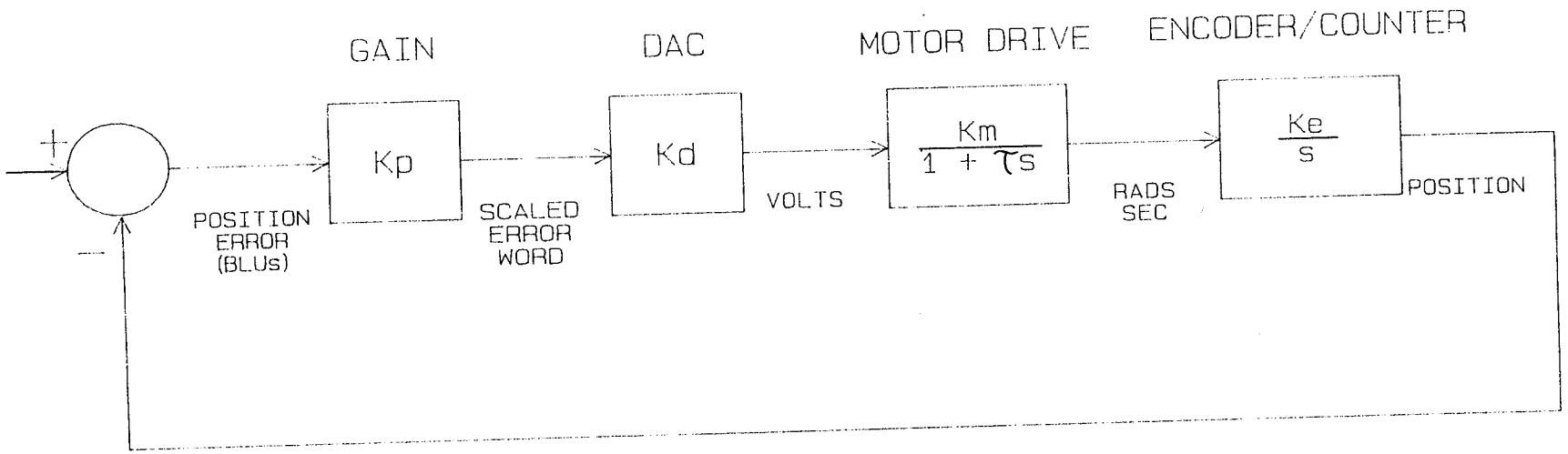


FIGURE 2.7 MATHEMATICAL BLOCK DIAGRAM OF A SAMPLED DATA, MACHINE DRIVE SYSTEM

control system. These effects undermine, but do not invalidate, the usefulness of the linear drive model.

The effect of extraneous external loads on the motor is another aspect which must be considered when constructing the drive's representation. Cutting forces and the friction of the leadscrews are fed back as torque loads to the motor through the gearbox. As these forces change, different motor current requirements result. The forces are referenced through a gearbox so that all the cutting forces are reduced by a factor of N, say, where N is the gearbox reduction factor. The value of N depends upon the specific application but a value of 10 was chosen here for the experimental system. Also, the motor is subjected to inertial loads which depend upon the weight of the workpiece being moved on the machine tool's table. Fortunately, these inertial loads are reduced by a factor of N^2 by the gearbox [1]. For a lightly loaded mill, the external loads will only cause small variations in the time constant of the loaded servo drive system.

The non-linearities of the amplifiers and unpredictable nature of the external loads prevent an exact determination of the machine drive's transfer function. Fortunately, a simple first order model

consisting of an integrator and a low pass filter is considered an acceptable model [7,9]. The open-loop, forward-path transfer function, Gf, of the machine drive's servo system may be written as:

$$Gf = Kp \times \frac{Kd \times Km \times Ke}{s\,(1 + \tau s)} \qquad (2.1)$$

where:

Kd = DAC Gain (Volts/pulse)
Km = Motor Gain (rads/sec volt)
Ke = Encoder Gain (pulses/rad)
Kp = Proportional Gain (unitless)
s = Laplace operator
τ = Loaded Motor Time Constant (secs)

The various gains of the machine drive system may be gathered simply into a single term Kt where Kt = Kp x Kd x Km x Ke.

System parameters are found easily from the hardware specifications of the equipment or from straightforward experiments. An encoder of 100 pulses/rotation was used experimentally and this value corresponds to an encoder gain, Ke, of 100/(2*pi) or 15.92 pulses/rad. The DAC produced 10 Volts fullscale which accords to an error word of 127. Hence the DAC gain, Kd, is 10/127 or 0.078 Volts/pulse. The motor rotated at a constant 3000 RPM for a reference voltage

of 10 Volts so that the motor's gain, Km, is 3000/(10*60)*(2*pi) or 31.42 rads/sec volt. The proportional factor, Kp, is not hardware specific and it is chosen by the designer for a desired damping factor of the system. By temporarily assuming Kp to be unity, the overall system's total gain, Kt, would be merely the product of the various gains or Kt = 1.0 x 0.078 x 31.42 x 15.92 = 39.02 sec^{-1}. A loaded motor's time constant may be obtained experimentally by providing a reference step speed to the motor's amplifier. The time required for the motor's speed to reach 63.2% of the requested value represents, by convention, the time constant of the drive. For the experimental system, the published motor time constant was 0.030 seconds and tests to be detailed in Chapter 4 will verify this value for small reference signal changes.

The procedure for determining the transfer function of a drive system has now been introduced. Equation (2.1) will form the basis of a mathematical analysis which uses classical control theory. A simple proportional analysis will be presented next which will serve later as an introduction to a more advanced analysis.

2.3.2 Proportional Analysis

It is generally accepted that a system can be considered essentially continuous if its inputs and outputs are sampled sufficiently quickly. While various authors [7] disagree on the degree of quickness, a sampling frequency 5 to 10 times the system's bandwidth will capture the relevant spectral information. To aid in understanding why the continuous approximation of a discrete series is reasonable, consider a situation in which external forces are causing a machine tool's toolbit to vibrate with large deflections at a frequency more than twice the control system's bandwidth. It is impossible for the control system to correct for such an external forcing frequency because the machine drives will not respond at the desired frequency. Hence, the drives' bandwidth represents the only frequency range of interest. Thus a sampling frequency for the controller of 5 to 10 times the highest frequency should not only assure a fair approximation but it also safely satisfies the Nyquist Criterion [10].

The forward path transfer function of the machine drive considered previously and the single integration in equation (2.1) indicates that the drive corresponds to a type '1' position control system [8,11]. Knowledge

of the system's type allows the steady state behaviour to certain specified inputs to be predicted. The two types of inputs of interest here are step speed and ramp speed changes. These inputs correspond to constant feedrates and steadily changing feedrates, respectively. On the other hand, the step and ramp speed inputs correspond to ramp and parabolic position inputs, respectively. Fortunately, the behaviour of a type '1' system is well-known for these inputs [11] and may be summarized, with reference to Figure 2.8, as follows:

Velocity Input	Corresponding Position Input Variation	Speed Error	Position Error
Increasing or Decreasing Ramp	Parabolic	1/Kt * A	Increasing
Constant Step	Ramp	Zero	1/Kt * V

Kt = System Gain
V = Speed
A = Acceleration

TABLE 2.1 - TYPE '1' SYSTEM BEHAVIOUR

Notice that the errors shown in Table 2.1 are inversely proportional to the overall machine drive's gain, Kt. Consequently a high gain will reduce axis position

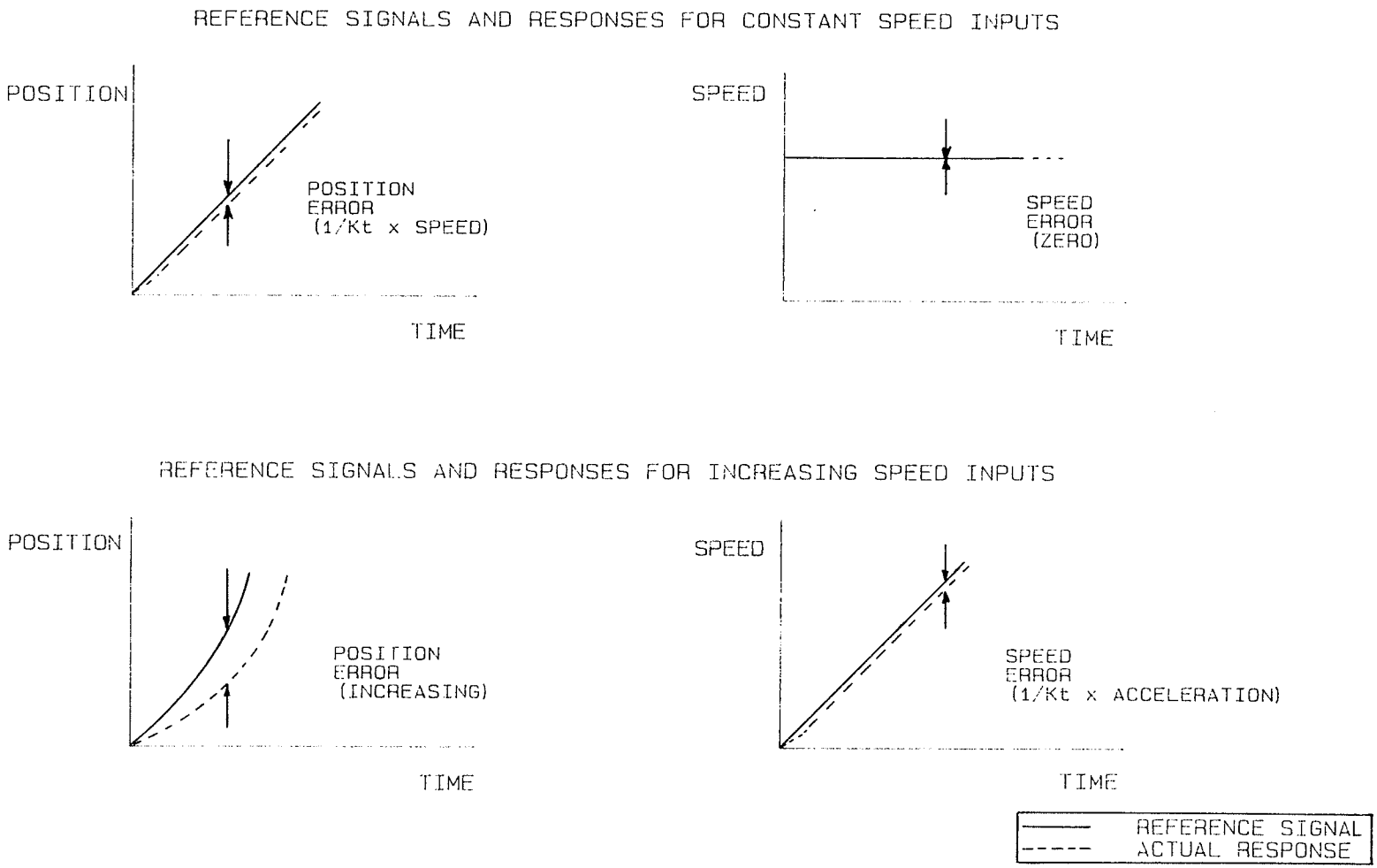


FIGURE 2.8 RESPONSES OF A TYPICAL TYPE '1' CONTROL SYSTEM

errors. Knowledge of the behaviour of the control system to a step or ramp input is advantageous because a typical servo movement contains both step and ramp signals. This knowledge will be used after first completing the mathematical analysis of the control system.

The previously described control scheme represents a basic proportional control in which the system is driven simply by the position error scaled by a proportional constant, Kp. Generally, all published work on servo systems use proportional control. The Kp is chosen to yield a selected closed-loop damping factor. High gains tend to reduce the errors of the machine drive but they also reduce the stability of the drive. These two tendencies are contradictory, of course, because a stable system is desired with a high gain. Much research has been published considering a system's stability as a function of the gain. A damping ratio of 0.707 is considered to provide the best trade-off between system stability and gain [3,9]. It is possible to calculate the damping factor, zeta, for any specified gain by using the following formula [9]:

$$\text{zeta} = \frac{1}{2\,(Kt * \tau)^{1/2}} \qquad (2.2)$$

where:

zeta = damping factor
Kt = overall system gain
τ = system time constant

The preceeding analysis is based on a continuous model of the system so that the sampling rate should be at least 5 times the system's bandwidth. By examining equation (2.2) it can be seen that, for the specified zeta of 0.707, the overall gain Kt is purely a function of the drive system's loaded machine time constant, τ. Having determined Kt, Kp can be specified from equation (2.1) because the remaining gains, Kd, Km, and Ke, are all hardware specific. Equation (2.1) yields a Kt value of 16.7 for the experimental system employed. This value of Kt implies that Kp must be 16.7/39.01 or 0.428 for a zeta of 0.707. Knowing Kp, it is possible to determine the smallest correctable position error of the machine drive. This error is calculated by considering the minimum position error that would set the least significant bit of the DAC and, hence, produce the minimum speed reference to the motor. It is found by determining:

$$\text{Pulse\#}_{min} \text{ such that } \quad \text{Pulse\#} * Kp >= 1 \qquad (2.3)$$

The error is system dependent and, for the constructed machine drive with a simple proportional scheme, the $\text{Pulse\#}_{min}$ was 3 pulses. This value of $\text{Pulse\#}_{min}$ indicates that it is impossible for the constructed drive to maintain a position accuracy greater than 3 Pulse units when using a proportional scheme. The corresponding position error would be acceptable only if the resolution of the drive was finer than its desired accuracy. To maintain real accuracy, the machine drive's resolution must be finer than the programming resolution (expressed in BLUs) and, thus, a BLU should preferably consist of multiple encoder pulses. A factor of 10 is a convenient multiple because it allows the conversion between a system's resolution units and the programming units to be performed by merely shifting the decimal point. The next step in the mathematical analysis will introduce the benefits of compensation.

Consider the Bode plot of the open loop, forward path transfer function presented in Figure 2.9. The Bode form shown is typical of all machine drives. The w_1 marked on the abscissa equals $1/\tau$ where τ is the time constant of a loaded axis drive. Given τ, the correct

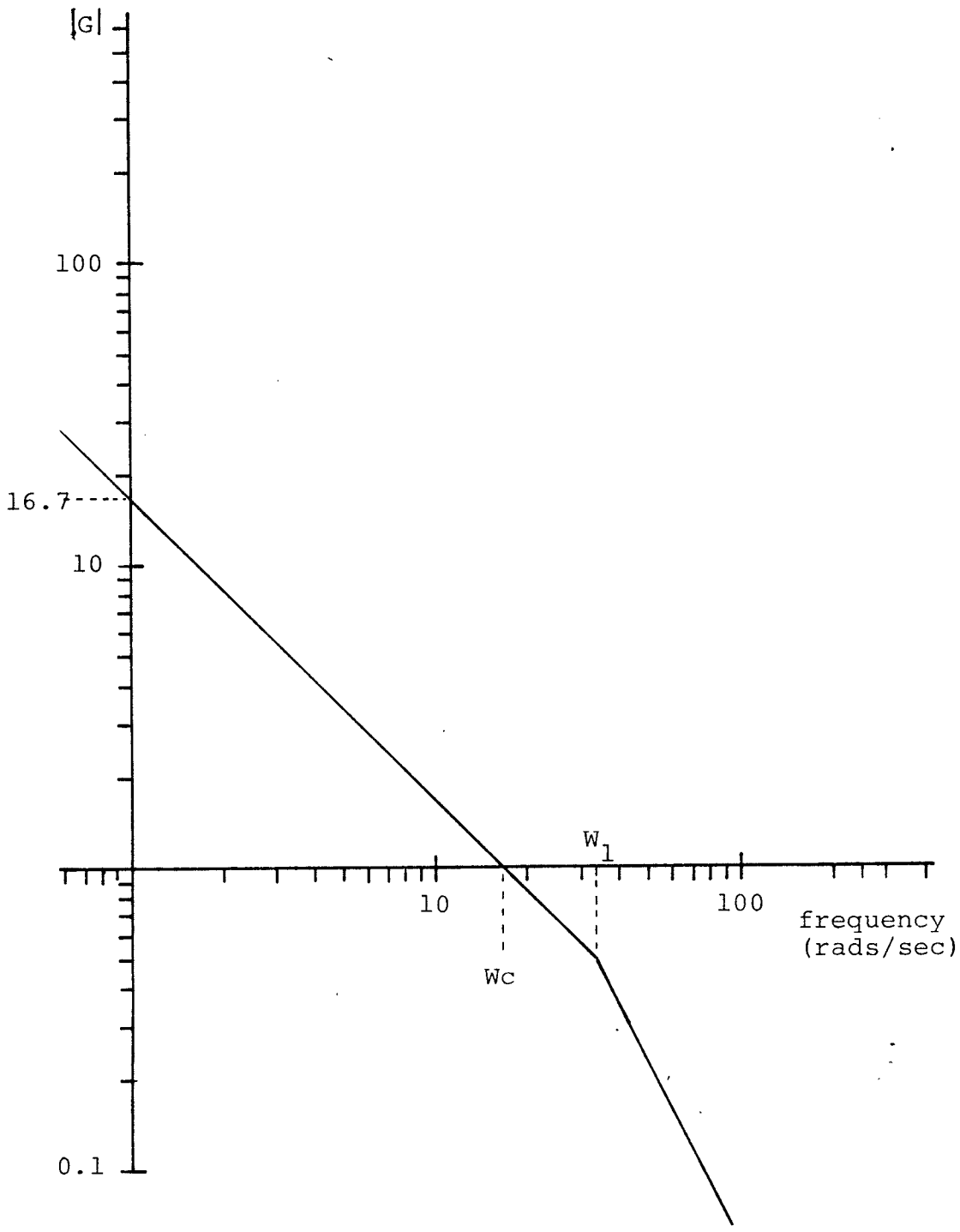


FIGURE 2.9 A TYPICAL OPEN LOOP TRANSFER FUNCTION FOR A SERVO MACHINE DRIVE SYSTEM

total gain, Kt, of the machine drive system can be chosen by using formula (2.2). Once the Bode plot has been determined, the crossover frequency, Wc, will be identical to Kt because the slope of the Bode plot is -45 degrees at this frequency. Consequently the frequency and gain axis intercepts are then identical. (Note that formula (2.2) ensures that the corner frequency $1/\tau$ is beneath the abscissa which is the requirement for equating the numerical values of Wc and Kt.) The crossover frequency, Wc, in Figure 2.9 allows the system's bandwidth to be estimated from [12]:

$$\text{Bandwidth} = Wc/(2*pi) \qquad (2.4)$$

where:

Wc = crossover frequency
pi = 3.14159 . . .

For the experimental system, if a proportional control scheme was used, with a crossover frequency of 16.7 rads/sec, the bandwidth would be 2.66 Hz. At this point, the system designer has enough information to estimate the step response of the system including the response time, time to first peak etc. These estimations can be reasonably based on a closed loop second order model [12].

2.3.3 Compensation

The accuracy of a machine tool is linked directly

to the gains of the position axes. It would be beneficial if a drive system's gain could be increased without affecting its stability. An unstable or marginally stable drive system is unacceptable because of oscillations or 'hunting' which would occur whenever the cutter was disturbed. Formula (2.2) indicates that, for a proportional control scheme, an increase in the control gain will always undermine the drive's stability. However, if the simple Kp block in Figure 2.7 is replaced by a compensation block Gc, the gain could be increased without damaging the drive's stability.

Consider Figure 2.10, for example, which shows three Bode magnitude plots of the forward path transfer functions of three different drive systems. The various gains were chosen for a drive system with a loaded time constant of 0.030 seconds. This value is representative of the drive system used in the experimental tests of the controller.

G_1 is the original system with a gain of 16.7,
G_2 is the original system with a gain of 200,
G_3 is an improved overall system.

The G_1 has been calculated to correspond to the ideal

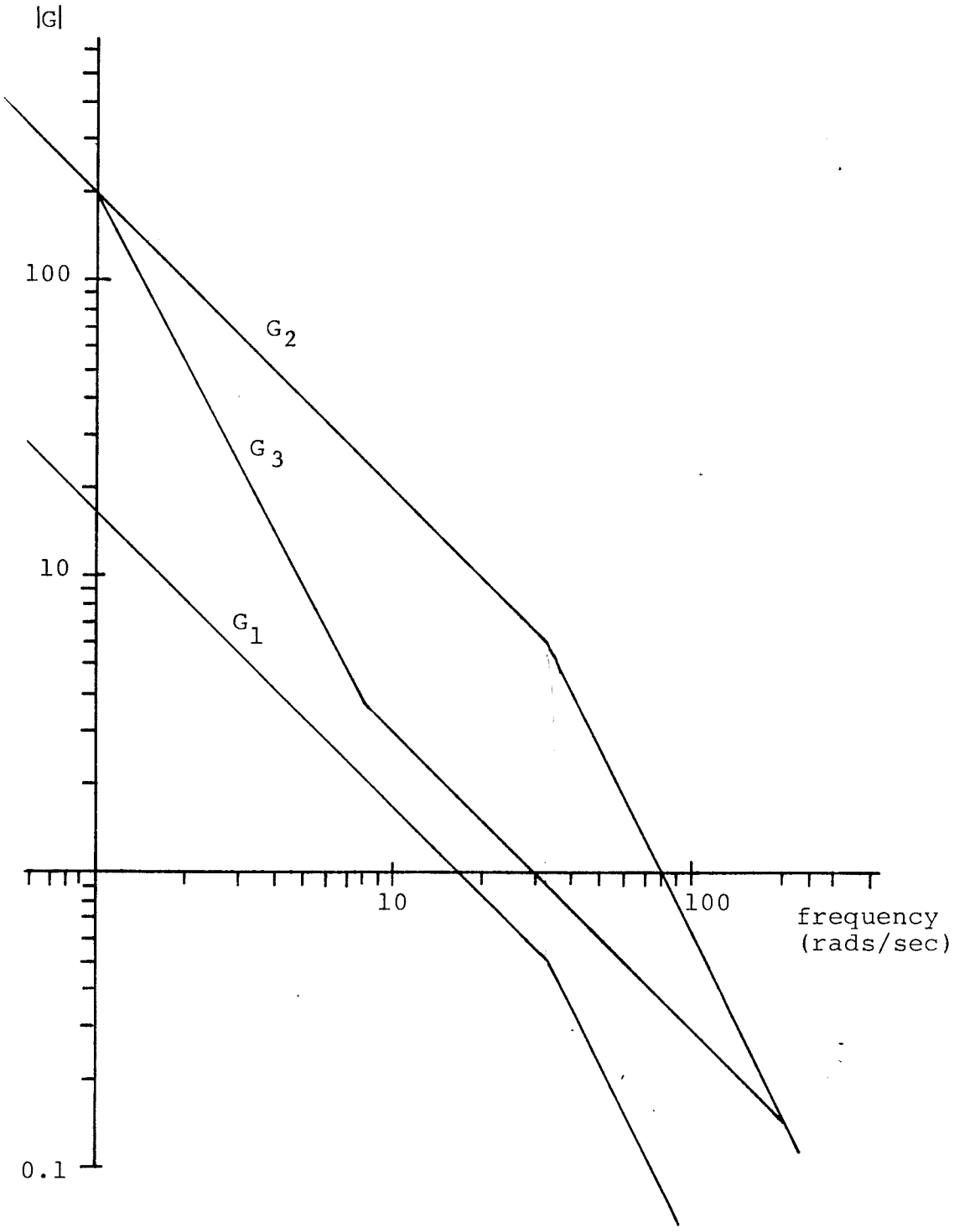


FIGURE 2.10 THE EFFECT OF COMPENSATION ON THE DRIVE SYSTEM'S BODE PLOT

zeta of 0.707. The G_2, on the other hand, would produce an unstable system. Now, G_3 represents G_2 when compensated with a damping factor of 0.7 - a value which is close to the ideal value of 0.707. The realization of G_3 would offer an increase in gain from 16.7 to about 200. If this drive system was available, it would offer a reduction by a factor of over 12 in the steady state speed error from a ramp input. In addition to the greater gain, the bandwidth would be widened which would improve the system's response time. The G_3 drive system would be significantly more accurate than the G_1 system. The advantages of a compensated system should not be minimized even though a computer search did not reveal any literature which examined the role of digital compensation in machine tool control. On the other hand, this technique does need the use of the Sampled Data Control Technique and the availability of a floating point programming unit. The actual realization of this compensator will be demonstrated next.

Consider the forward path transfer function G_3 which is merely 'read from' the Bode diagram by using the asymptotic approximation [11,12]:

$$G_3 = \frac{200\ (1 + s/8)}{s\ (1 + s/1)\ (1 + s/235)} . \qquad (2.5)$$

The unstable and uncompensated original drive system (with a gain of 200) has a transfer function of:

$$G_{\infty} = \frac{200}{s(1 + s/33.3)} . \qquad (2.6)$$

Using the techniques of Bode plot subtraction [12], formula (2.6) implies that the cascade compensator transfer function must be:

$$Gc = \frac{(1 + s/8)\ (1 + s/33.3)}{(1 + s/1)\ \ (1 + s/235)} . \qquad (2.7)$$

The last formula represents a standard lag-lead compensator function [11,12].

Although the lag-lead compensator could be realized by using an analogue network, a cheaper and more elegant implementation is possible. The analysis to this point has approximated the control system as continuous but the system is, of course, a sampled-data control system. This implies that the compensator can be actually realized computationally by using digital control techniques.

By using the techniques of pole-zero mapping [13], the compensator function of formula (2.7) may be transformed in the z-domain as:

$$Gc^*(z) = \frac{0.413351\ (z - 0.923116)\ (z - 0.716770)}{(z - 0.990050)\ (z - 0.095369)} . \qquad (2.8)$$

The above form of compensator may be implemented simply by using direct digital compensation [14]. Details of this transformation and direct digital programming are given in Appendix I. However, real time floating point calculations are required and the actual machine drive position must be available for the calculations. The actual machine drive's position is only available if the Sampled Data Technique is used for the machine's control. Now that the basic control theory and software techniques have been examined, principles of the interface hardware will be introduced.

2.4 Electronic Interface

The drive interface allows the microcomputer to monitor and control each axis of motion. An axis interface must contain a Digital to Analogue Converter to provide a voltage speed reference to the drive's electronics. In addition, an encoder interface is needed to determine the position of the motor. The feedback pulses from the optical encoder must also be decoded to determine the encoder's direction of motion. These pulses are used to increment or decrement a small

external counter which is cleared at every sample instant. The external pulse counter must be bi-directional to accomodate both clockwise and counterclockwise motions of the motor. Furthermore, the counter must have a sufficient word length to contain the pulse count associated with the motor's fastest speed. The pulse count and word length required may be calculated from:

$$PCR = \frac{Mv \times Lc}{60 \times Fi} \times 2 \qquad (2.9)$$

$$WLR = N_{min} \quad \text{where} \quad 2^N >= PCR \qquad (2.10)$$

where:

WLR = Word Length Required (bits)
PCR = Pulse Count Required (BLUs)
Lc = Encoder pulses/motor rotation (BLUs)
Mv = Maximum motor speed (RPM)
Fi = Sampling frequency (Hz.)

The above formula can be substantiated straightforwardly because the numerator represents the highest number of possible encoder pulses per minute and the denominator represents the number of samples (and consequent zeroing of the counter) per minute. The factor of two in formula (2.9) exists because an extra sign bit is needed to separate positive clockwise and negative

counterclockwise movements. A latch is required as part of the interface because it is imperative that the counter is able to operate unhindered while the computer is reading the counter's contents. This requirement is necessary because the motor will often be rotating during a sampling instant. It is also necessary to clear the counter after every sample in order that the counter not overflow. These requirements are most fundamental. A circuit which would accomplish this procedure is shown in Figure 2.11.

The digital optical encoder illustrated in Figure 2.11 outputs two pulse streams which have a 90 degrees phase difference. Channel A leads channel B for encoder rotations in one direction and, for the other direction, channel B leads A. Outputs from the encoder are passed through a simple low-pass filter in order to filter unwanted high frequencies. The bandwidth of this filter should be chosen so that pulses with durations shorter than the fastest encoder pulses are filtered. This precaution prevents extraneous noise from triggering the circuitry and avoids false counts. Schmitt triggers are employed to reshape the pulses. The reshaped pulses are used as inputs to the Flip-Flops of Figure 2.11. These Flip-Flops require both channel A and B of the encoder

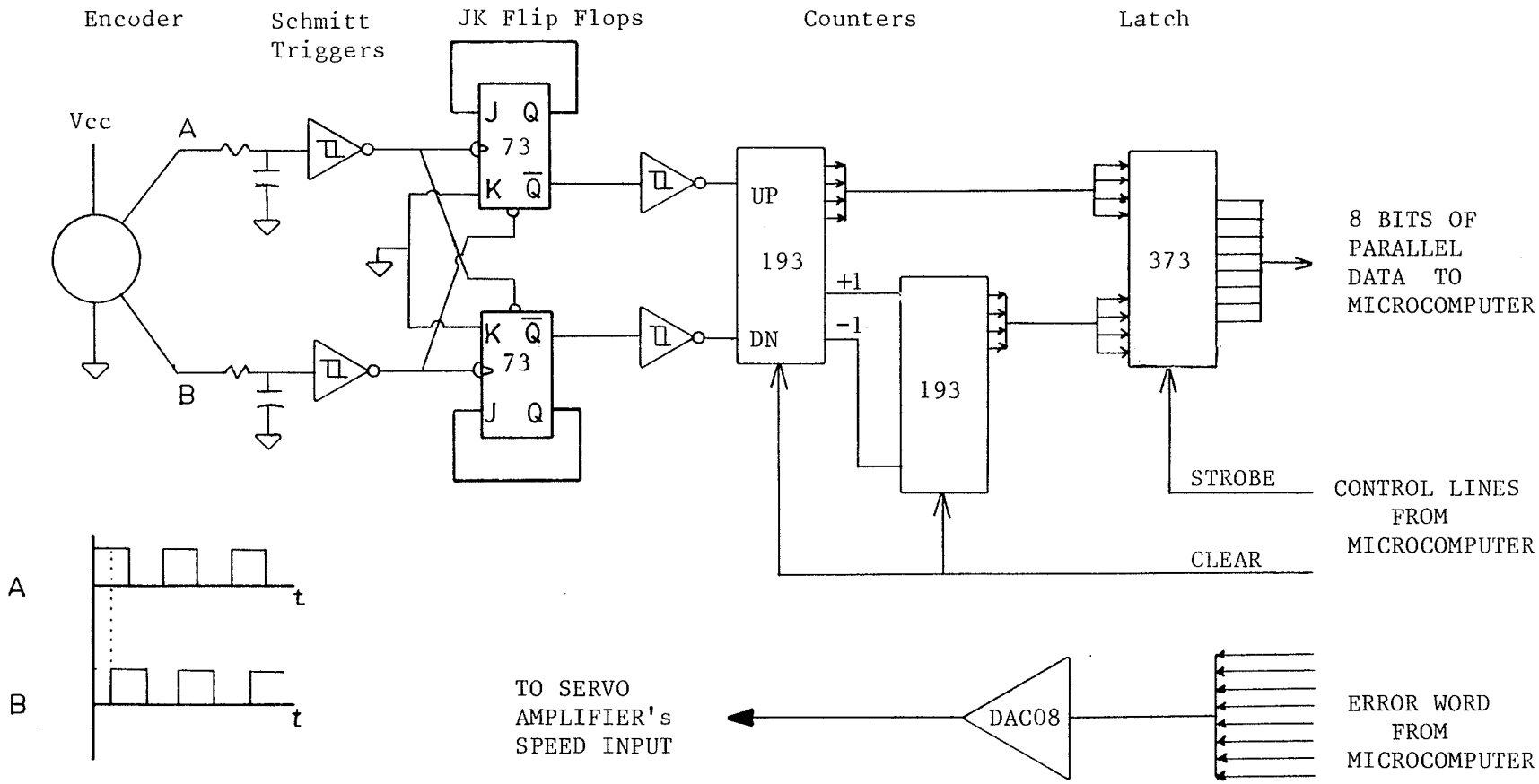


FIGURE 2.11 BASIC ENCODER INTERFACE CIRCUIT

to change states before a pulse is accepted as valid. This procedure effectively rejects false pulses caused by mechanical vibration from the machine which, in turn, causes the encoder to vibrate around a channel transition point. The Flip-Flops also direct the pulse stream to the appropriate terminal of the cascaded counters. The counter outputs are buffered by a latch and are connected to the microcomputer. A control line must exist to 'strobe' the latch in order to load the counter's contents. Immediately after loading the counter, the 'strobe' must also clear the external counter.

The described interface is very basic and a more elaborate interface, with improved features, will be introduced in Chapter 3. References [1,15,16] provide an excellent treatment on the use of incremental optical shaft encoders.

2.5 Chapter Summary

This chapter has introduced the mathematics and basic hardware and software which is required for the implementation of a sampled data, machine drive controller. Digital compensation techniques have been shown to offer significant improvements in drive

performance. The Sampled Data Control Technique and a floating point unit are necessary to implement digital compensation. Chapter 3 will provide details of an experimental control system which uses the described techniques.

CHAPTER 3

EXPERIMENTAL SETUP

3.1 Introduction

A two-axis contouring machine controller was developed and installed on a small milling machine to test the control scheme discussed in Chapter Two. The mill, Model LC30A manufactured by the Long Chang Machinery Company Ltd., was originally a manual machine but was modified for use as a CNC machine.

Figure 3.1 shows the modified milling machine. With reference to Figure 3.1, the handwheels of the table's leadscrews were replaced with couplers. The milling machine's leadscrews were of such a pitch that a complete rotation of either of the table leadscrews produced 0.1 inches of translational motion for its respective axis. The output shaft of a tenfold gear reducer was attached to each coupler and the axis servo motors were mounted on the gear reducer input shafts. A reduction factor of 10 was chosen in order to get the desired torque-speed range for the mill. Rotary optical incremental encoders were mounted on the ends of the

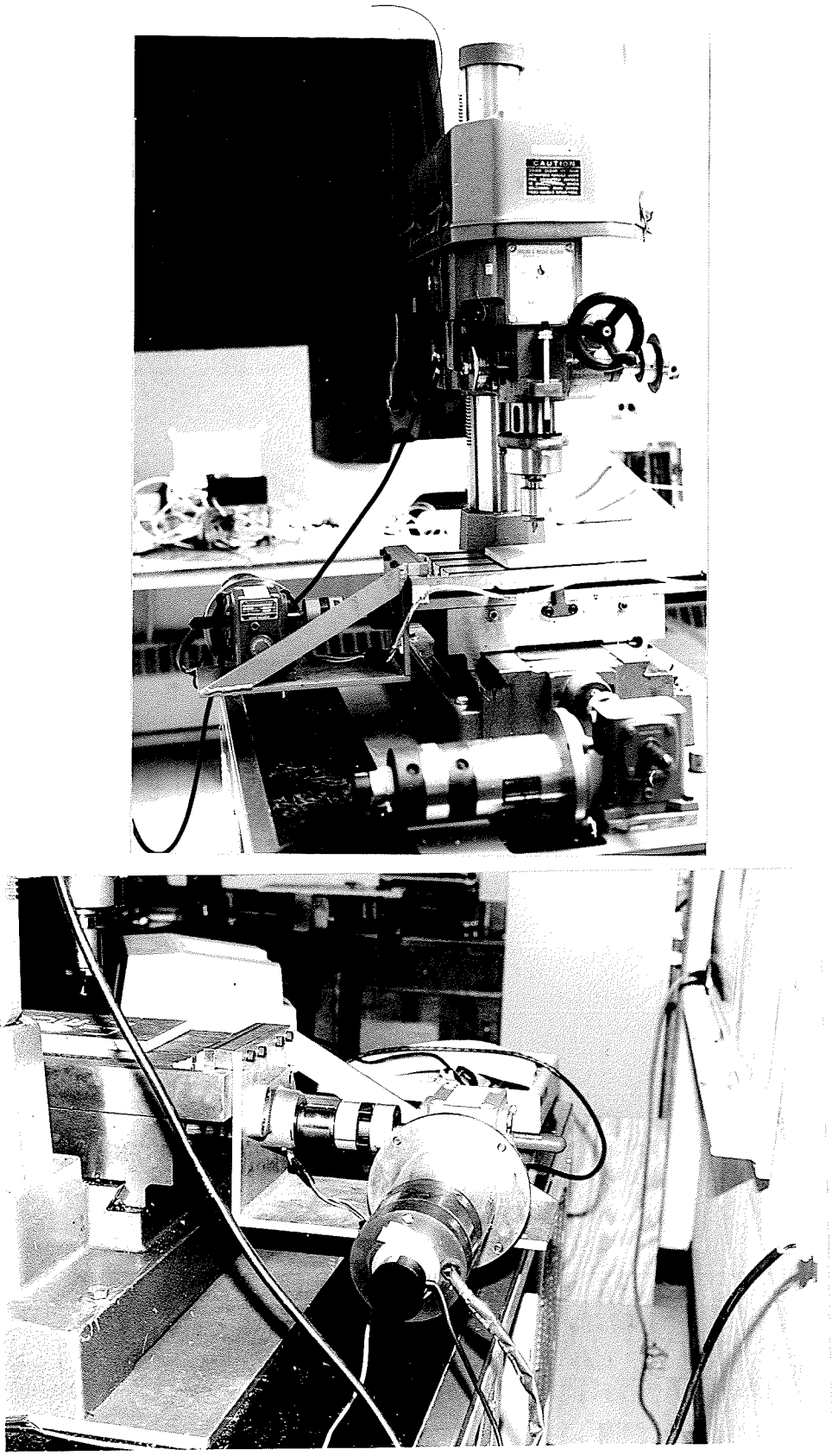


Figure 3.1 Photograph of Modified Milling Machine

motor shafts. The encoders produced 100 pulses per revolution and their mounting position provided a resolution of 1/100 of a rotation of the motor's output shaft. Thus, the final resolution for the gear reducer on the output shaft was 1/10 x 1/100 or, 1/1000 of a rotation. Considering that a single complete rotation of the axis leadscrews produced a translation motion of 0.1 inches, the basic machine drive's resolution was 0.1 inches/rotation x 1/1000 rotation/pulse, or 0.0001 inches per pulse. At this resolution, the speed range of the machine varied from 100 BLUs/second to 4000 BLUs/second, corresponding to 0.010 inches/second and 0.400 inches/second, respectively. The speed could be adjusted between these two extremes in increments of 0.010 inches/second.

It is important to note that the positioning control loop was closed at the motor shaft. This means that any backlash or other errors in the machine axes would be undetectable and, hence, uncorrectable by the control system. Normally, it is desirable to close the control loop around as many of the drive components as possible. However, to fully test the control scheme's accuracy and isolate the backlash of the milling machine's leadscrews, the encoder was mounted on the

motor shaft. For discussion purposes, references to accuracy or position error assume that the drive line components beyond the motor shaft are 'perfect' and do not contribute to the position errors.

An IBM PC microcomputer was used as the control computer. To provide the computational throughput required by the Sampled Data Control Technique, a 8087 mathematical coprocessor and a commercial input-output (I/O) board were installed in addition to a custom made encoder interface. The hardware specifications for the computer system are detailed in Appendix II and the major components of the machine control unit will be considered separately next.

3.2 Software

3.2.1 The Software Structure

The focus of this project was to implement machine control by employing software techniques. To ensure flexibility, FORTRAN 77 was used for all the sophisticated manipulations. However, FORTRAN has extremely limited input-output capabilities so that all low level input-output was performed by ASSEMBLER subroutines which were linked to the FORTRAN mainline. The Sampled Data control technique adopted relied on

sampling at constant time intervals. Normally, this technique is implemented by using an external clock which interrupts the computer at a constant frequency. However, the IBM PC has an internal time-of-day clock which is interrupt driven. This clock is assigned permanently to the highest-priority interrupt of the computer. It was possible to reprogram the computer to ensure that this highest priority interrupt drove the control program. This technique guaranteed an exact sampling frequency. A somewhat arbitrary but quite high sampling frequency of 100 Hz. was chosen to test whether complex interpolation schemes could be implemented in real time. This frequency met the requirements of being at least 5 times higher than the drive system's bandwidth.

The main program was constructed by using modular programming. Named common blocks were utilized to minimize the passing of parameters between the code modules. Data was separated into convenient categories such that each subroutine was granted access to the common blocks needed. Each major module will be considered in turn. Appendix III contains documentation and listings of the control programs.

3.2.2 The Data Processing Unit

3.2.2.1 Introduction

As described previously, the Data Processing Unit's function is to provide speed reference signals for the Control Loops Unit. The Data Processing Unit consists of a decoding subroutine and all the interpolation subroutines. The decoding subroutine uses the part program as input and invokes the appropriate interpolation subroutine for that given block.

Linear and circular interpolation routines were implemented as part of the Data Processing Unit. These two interpolation profiles define the minimum interpolator set for a contouring machine. Other interpolators, such as parabolic or spline fit interpolators, could also be included because the Sampled Data Technique was used for the machine control system. (Recall that the Reference Pulse Technique provides only limited interpolation capabilities.) The interpolators do not directly drive the motors but only provide a stream of reference speeds for each axis to the Control Loops Unit. The Control Loops Unit employs these speeds to calculate the reference positions of the axes. For each axis, the difference between the actual and reference positions is used to drive the

corresponding motor.

The process of interpolation involves the approximation of a desired curve by linear segments. This segmentization is caused by the sampled data nature of the control scheme whereby the speed of the axis motors can be adjusted only at the sampling frequency. Thus, a circular arc is approximated as a series of segments, each segment length being the distance that the toolbit travels in a single sample period. The segment length is proportional to the feedrate and inversely proportional to the sampling frequency. This segmentization will be explained in more detail later, but first it is useful to introduce the concept of a servo speed profile.

3.2.2.2 Speed Profile

Chapter II indicated that the drive response time and interrelated servo mismatch was the major source of contour errors. To minimize the possible contour error due to response time mismatch, it is desirable not to request instantaneous speed changes from the machine drive. For this reason, accelerations and decelerations were used at the beginning and end of each move.

It is useful to consider that each interpolator

produces a moving feed vector to better understand an interpolation function. The 'tail' of the vector moves along the desired curve and the vector always 'points' in a direction tangent to the curve. The speed profile provides the magnitude of this vector. For both the linear and circular interpolators, trapezoidal speed profiles were used as illustrated in Figure 3.2.

To determine the speed profile, the absolute value of the toolbit travel, delta, must be calculated for a given segment of motion. For linear interpolation, delta is given by the incremental distance between the initial and final points of the toolbit motion for any given linear segment. On the other hand, delta is the incremental arc distance between the initial and final points for circular interpolation. After calculating the toolbit's total travel, a series of pre-motion calculations are made to determine the speed profile of the toolbit based upon the incremental distance, acceleration and deceleration. The area under the trapezoidal speed profile represents the distance travelled and, for the purpose of subsequent discussion, it will be subdivided into the areas one, two and three defined in Figure 3.2.

For a given motion, the feedrate and toolbit's

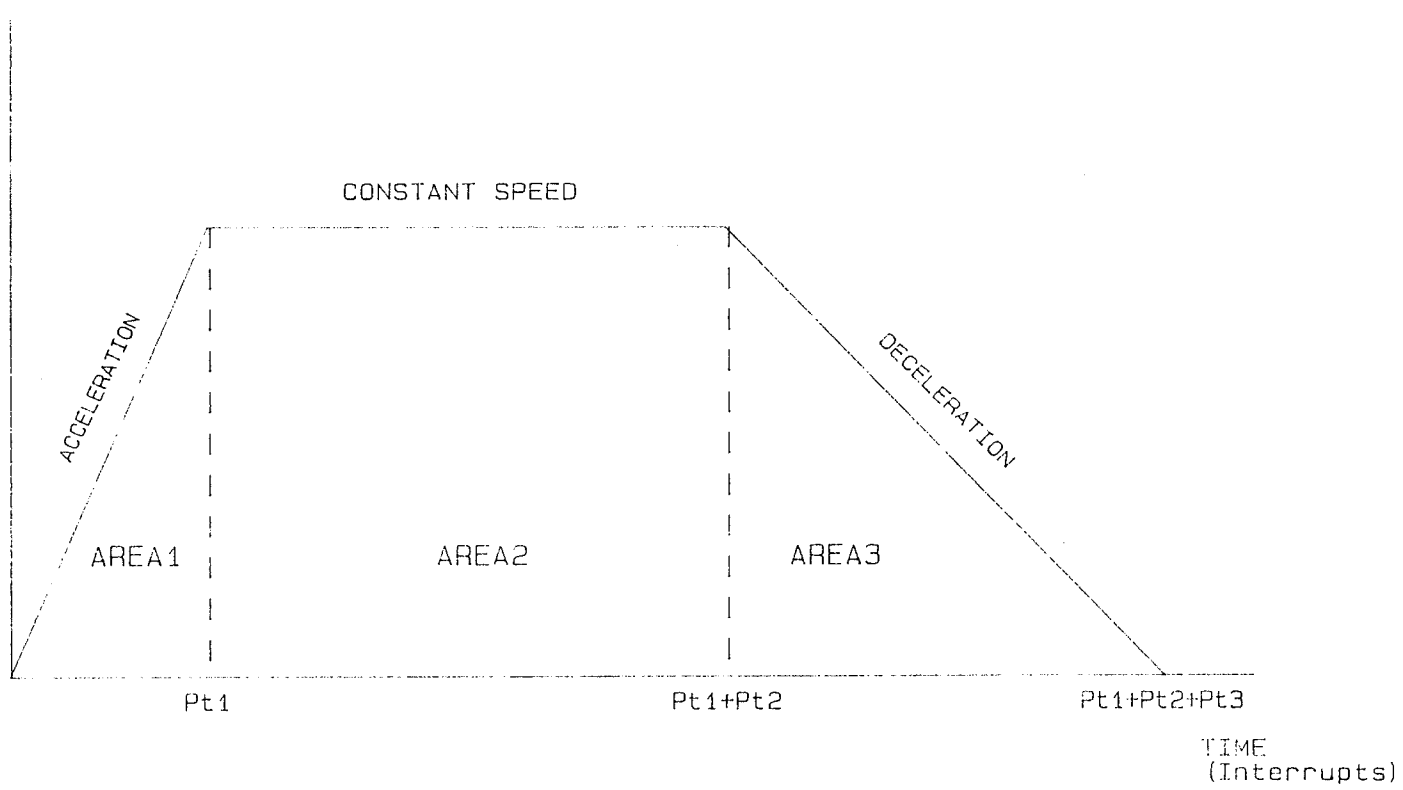


FIGURE 3.2 DEFINITION OF THE SPEED CURVE SUB-AREAS

final absolute position are specified by the part program. The present position of the toolbit is known so that the incremental path distance, delta, which the toolbit must travel for the next segment can be calculated easily. After determining delta, the interpolators must determine the number of sample periods, Pt1, Pt2 and Pt3, that the toolbit must accelerate, travel at the prescibed feedrate, and then decelerate. However, a problem can arise because the part programmer specifies only the feedrate and endpoint. It is quite possible, for instance, that a feedrate could be supplied which makes it impossible for the toolbit to accelerate to the prescribed feedrate and decelerate to a stop without overshooting a given move's endpoint. This possibility must be considered by the speed profile algorithm. A calculation procedure will be detailed in the following paragraphs which considers this possibility.

With reference to Figure 3.2, the acceleration and deceleration times are calculated as:

$$Pt1 = feedrate/acl$$
$$Pt3 = feedrate/decel \qquad (3.1)$$

where:

feedrate = prescribed feedrate (BLU's/interrupt)
acl = acceleration rate (BLU's/int/int)
decel = deceleration rate (BLU's/int/int)
Pt1 = time to accelerate (ints)

Pt3 = time to decelerate (ints)

Note that acceleration acl and deceleration rate, decel, are not specified by the part program so that they can be 'hard coded' as particular values in the Data Processing Unit. The acceleration rate, acl, was chosen experimentally to produce a fast speed change which did not severely saturate the motor amplifiers. Then the deceleration rate, decel, was selected as half this acceleration rate. Although the factor of a half is somewhat arbitrary, it does give a deceleration rate which is less than the acceleration rate. This, in turn, tends to alleviate toolbit overshoot by reducing the speed error, and subsequent position error, during the deceleration ramp.

Knowledge of pt1 and pt2 from equation (3.1) and the feedrate (from the part program) allows the determination of the total distance travelled while accelerating, Area1, and decelerating, Area3. These distances are given by:

$$\begin{aligned} \text{Area1} &= 1/2 * \text{pt1} * \text{feedrate} \quad (\text{BLUs}) \\ \text{Area3} &= 1/2 * \text{pt3} * \text{feedrate} \quad (\text{BLUs}) \end{aligned} \tag{3.2}$$

Consequently, Area2 can be calculated from:

$$\text{Area2} = \text{delta} - \text{Area1} - \text{Area3} \tag{3.3}$$

where:

delta = total incremental path travel (BLUs)
Area1 = toolbit travel while accelerating (BLUs)
Area2 = toolbit travel while at feedrate (BLUs)
Area3 = toolbit travel while decelerating (BLUs)

A negative Area2 implies that the prescribed feedrate is too high for the given delta. Under these circumstances, the program automatically chooses a new feedrate at 90% of the present feedrate and recalculates the three areas. This procedure continues until the feedrate can be reached in at least a single interrupt period.

It was mentioned previously that the deceleration period, pt3, is precalculated. Consider the typical reference and actual speed profiles shown in Figure 3.3. Recall the behaviour of a type '1' system introduced in Section 2.3.2 in which the steady state behaviour is known for both ramp and steady speed references. Consider the speed overshoot at the beginning of the deceleration ramp. Such an overshoot would cause the actual distance travelled by the cutter to be larger than the prescribed distance. However, the consistent and, hence, predictable nature of the actual speed profile can be employed advantageously to compensate for the response time of the axis drive system. (A computer literature search on the topic of machine tool control

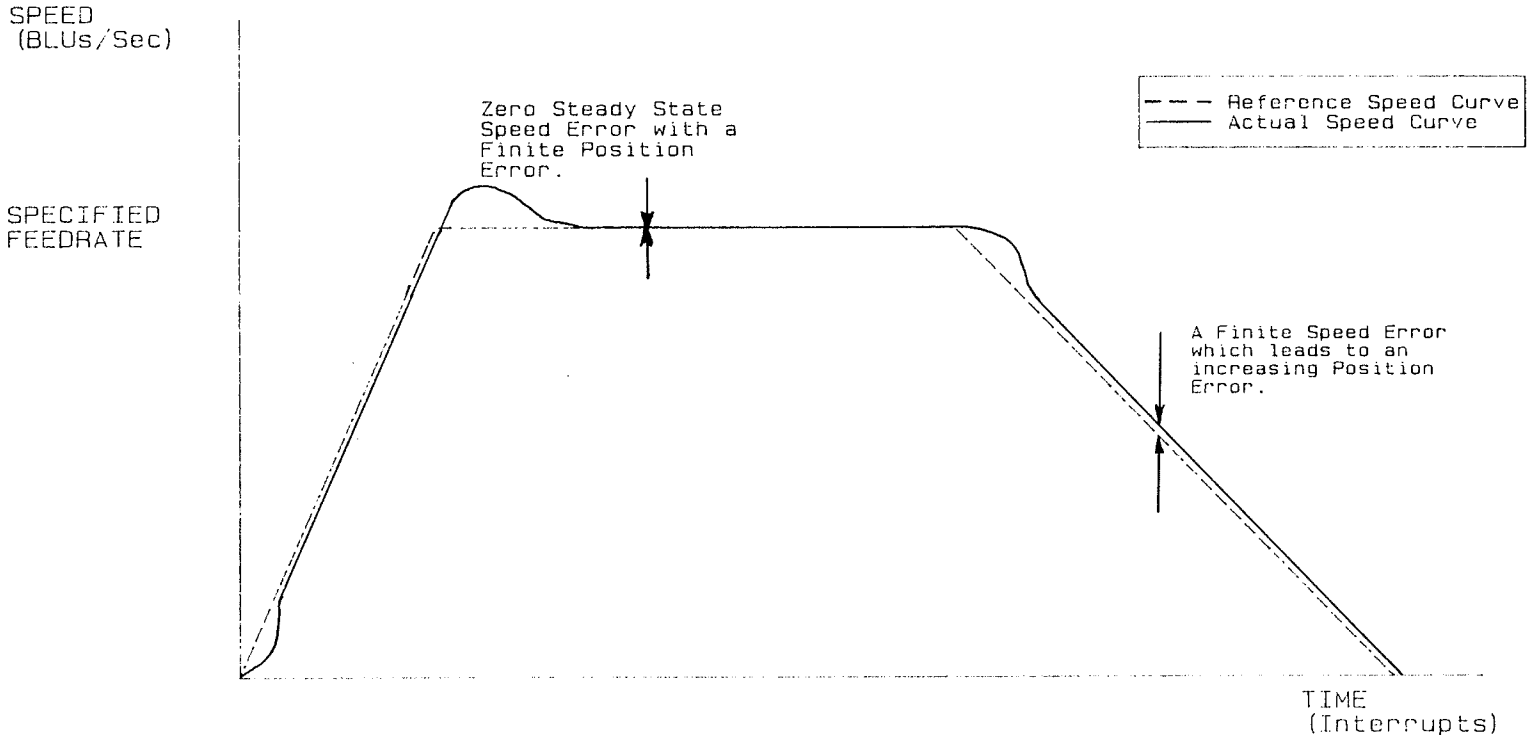


FIGURE 3.3 A COMPARISON OF ACTUAL AND REFERENCE SPEED CURVES

did not produce any description or treatment of motion speed profiles.) To compensate for both this overshoot tendency and digital quantisization or noise errors which may have accumulated at the end of the full speed portion of the curve, the deceleration rate is modified slightly and area3 is recalculated based on the actual distance left to travel. This technique greatly improves the control program's ability to compensate for the drive's actual speed curve. Chapter 4 will provide a detailed explanation of this deceleration adjustment and will present experimental results demonstrating its usefulness.

3.2.2.3 The Linear Interpolator

The linear interpolation subroutine receives the final desired position of the toolbit as well as the cutting feedrate from the decoding subroutine. The toolbit's present position is known because it is readily available from the position common block in the microcomputer. The linear interpolator uses the moving feed vector technique which was introduced earlier. This feed vector changes in magnitude and position and the vector 'points' directly from the toolbit's present, instantaneous reference position to the desired final

toolbit position. The magnitude of the feed vector is provided by the previously described trapezoidal speed algorithm. At each sample instant the 'tail' of the feed vector advances to the previous reference position and the Cartesian components of the feed vector are utilized as the speed inputs to the Control Loops Unit. These inputs are used by the Control Loops Unit to calculate the new reference position for each axis. Thus, at each sample, the feed vector advances along the desired curve by the amount specified by the feed vector of the previous sample. This procedure continues with the feed vector moving along the curve until the end of the motion segment is reached.

3.2.2.4 The Circular Interpolator

As mentioned in Section 3.2.2.1, the process of interpolation involves approximating a curve by a series of linear segments. At each sampling instant, the interpolator specifies the next point to which the toolbit should move during the next sample period. The Control Loops Unit drives the toolbit directly from the present point to the next specified point and, thus, the motion between the two interpolated points on the curve is represented by a linear segment. When considering

linear interpolation, this segmentization is not of concern because each reference position and segment lies exactly upon the desired line. However, during circular interpolation, these segments do not lie exactly on the desired arc. Fortunately, it is possible to easily calculate the effects of the linearization. For the experimental apparatus, the maximum segment length was 0.004 inches which corresponds to the maximum feedrate of 0.400 inches/second with the chosen 100 Hz. sampling frequency. Figure 3.4 shows a single linear segment of a circular interpolation in which the endpoint of the segment corresponds to the reference position for the sample. The circular arc gradually separates from the tangential line segment and E represents the maximum contour error during the sample. Straightforward calculations indicate that the contour error, E, will not be greater than 1 BLU or 0.0001 inches for arc radii of more than 0.080 inches at the maximum feedrate. This value of E indicates that, under normal operating ranges of the mill, line segments which represent circular arcs will not contribute noticeable contour errors. The details of the interpolator will now be presented.

The developed circular interpolator was designed to accept standard EIA codes. These codes require that

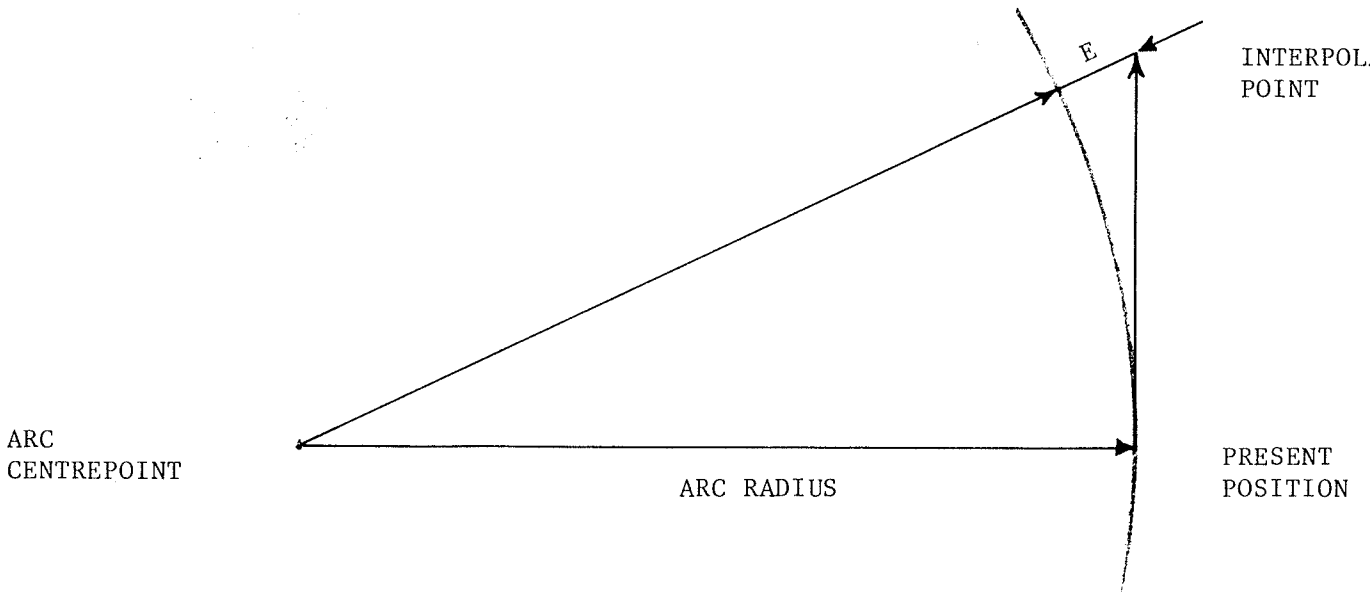


FIGURE 3.4 LINEARIZATION OF CIRCULAR ARC

all circular arcs must start and finish in the same quadrant. The codes include:

i) coordinates defining the final toolbit position,

ii) a pair of numbers, i and j, which define the absolute offsets of the centrepoint of the arc from the present toolbit position,

iii) a code defining whether the toolbit movement should be clockwise or counterclockwise, and

iv) the desired feedrate along the path.

When used in conjunction with the toolbit's (known) present position, the information provided in the part program provides enough information to perform the circular interpolation. The basic information that is required to begin the interpolation process is:

i) the arc's start point,

ii) the arc's final point,

iii) the arc's centrepoint, and

iv) the desired feedrate.

The present toolbit position is available within the microcomputer and the desired final toolbit position and feedrate are given by the part program. The arc's centrepoint may be calculated from the i and j codes. These codes represent the absolute values of the X and Y component offsets of the arc's centrepoint from the

present toolbit position. The interpolation subroutine must determine the signs of i and j. This is accomplished by finding the i and j sign combination which produces an arc's centrepoint that is equidistant from the toolbit's present position and the required final position of the toolbit. After determining the above mentioned information, the circular interpolator determines the start angle and the final angle. These angles are determined easily by using the arc's centrepoint, the toolbit's initial point and the toolbit's final position. The knowledge of these angles and the arc's radius enables the interpolator to calculate the 'delta' used in speed profile calculations described in Section 3.2.2.2. After establishing the speed profile, the interpolator has enough information to begin to drive the motors.

As mentioned previously, to better understand the circular interpolator, it is useful to consider the circular motion of the cutter as being represented by a feed vector. This vector moves tangentially around the circular arc, such that its 'tail' remains on the arc and at each sampling instant, its 'tip' establishes the new reference position for the axes. (Actually, it is the components of this vector which provide the axes

speeds for the Control Loops Unit which, in turn, establishes the new reference point.) At each sampling instant, the feed vector changes position, direction, and magnitude. The magnitude of the vector is provided by the trapezoidal speed profile. The direction and position of the vector is given by the interpolator such that, at each sampling instant, the vector is established tangential to the desired reference arc. Figure 3.5 provides a conceptual diagram which shows the movement of the feed vector, at equal time increments, during a circular interpolation. The magnitude of the vector is greatly enlarged for clarity. It is provided by the trapezoidal speed profile. The vector's magnitude can be seen to increase gradually over the initial portion of the arc and then decrease during the final portion of the arc. These portions correspond to the toolbit's acceleration and deceleration phases, respectively. The procedure for determining the direction of the feed vector will be detailed next.

The interpolator performs the feed vector's movement by establishing a position vector from the arc's centrepoint to the toolbit's present position. This position vector is updated at every sample and moves around the arc radii to its final position at

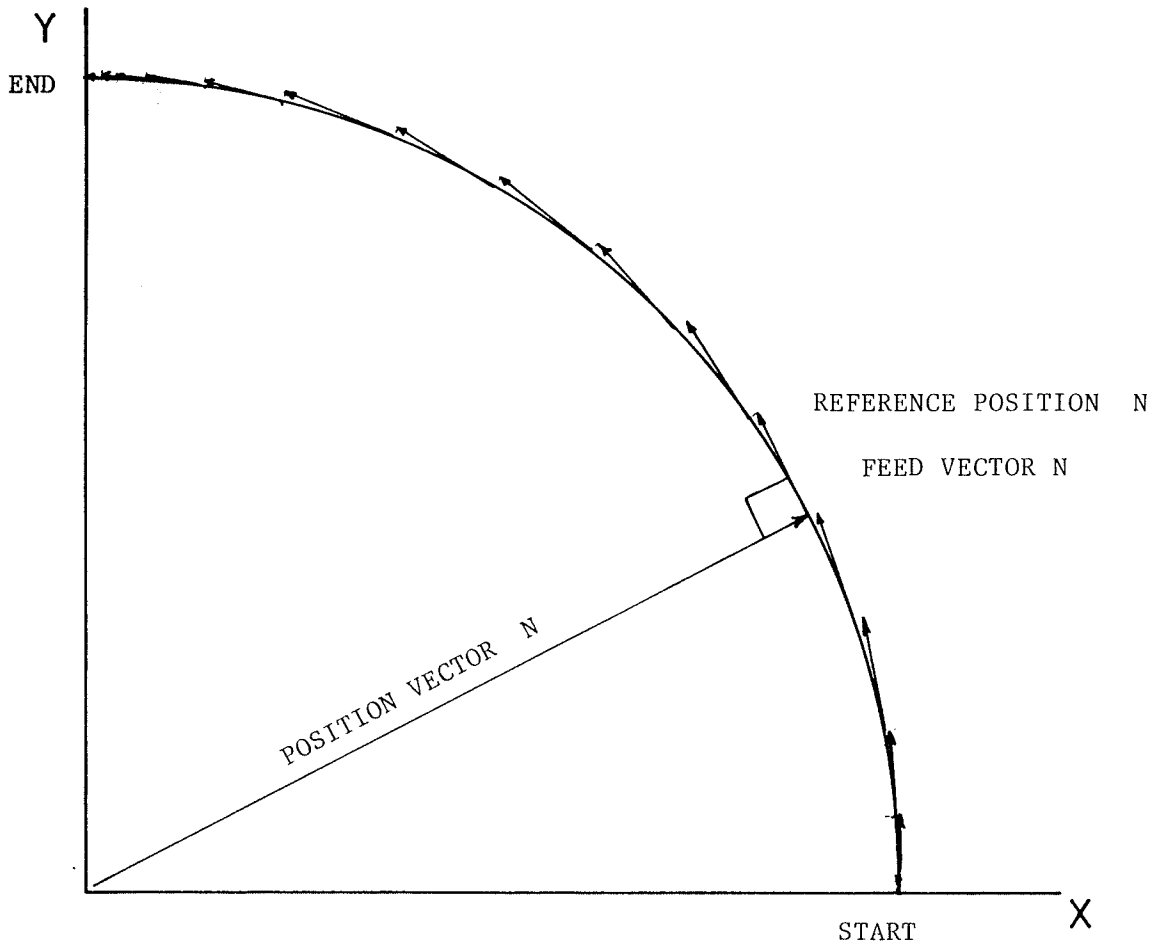


FIGURE 3.5 CONCEPTUAL DIAGRAM OF **CIRCULAR INTER**POLATION

θ_{tan}. The subroutine establishes the feed vector at right angles to the position vector in the direction which will drive the toolbit to the desired final position. The time varying Cartesian components of the feed vector represent the desired X and Y speed components for the axis drives. These components are used as axis speed references by the Control Loops Unit which, in turn, updates the reference position for the axes.

3.2.3 The Control Loops Unit

The Control Loops Unit (CLU) drives the axes motors in response to the speed reference signals generated by the interpolation routines. The CLU is implemented as a subroutine which, with the aid of ASSEMBLER subprograms, has access to the digital to analogue converter and encoder counter. The subroutine is driven directly by the interrupt clock at the sampling rate of 100 Hz in the following manner. When invoked, the subroutine receives the required axes speeds. It then waits for an interrupt from a background assembler subroutine to set a variable which serves as an interrupt occurrence indicator. After the interrupt has occurred, the computer accepts the incremental position change

from the external counter and adds it to the position accumulator. A new reference position is determined from the passed speeds so that the current position error can be calculated. This error may be scaled and used as the output to the digital to analoque converters. However, as explained in Section 2.2.3, it would be beneficial from the viewpoint of the machine drive's performance to implement a lead-lag filter. A direct digital realization of a sampled data compensator was utilized to improve the machine drive's performance. The compensator example, represented by equation (2.8) in Section 2.2.3, was implemented by employing direct digital programming.

3.3 Interface

Section 2.3 introduced the basic requirements for a drive interface for each axis of a CNC controller. However, a more noise immune interface was designed and constructed for this project. The major advantage of this interface was that it utilized a third signal from the optical encoder. This signal was an index pulse which occurred only once every rotation of the encoder for unidirectional rotation. It was used to ensure that the contents of the position accumulators did not drift

from the correct values. Each time the index pulse occurred, circuitry on the interface stored the counter value at that instant. When the next sampling period was reached, a flag notified the software that an index count was available. The index count was used to align the counter and then the 'regular' (ie. the entire sample period count) was loaded to calculate the location of the axes. This technique realigned each axis every 100 encoder pulses, essentially confirming the machine position every 0.01 inches of toolbit motion. Appendix IV details the required drive interface hardware.

3.4 Chapter Summary

This chapter has introduced the actual environment under which the control algorithms developed in Chapter 2 were tested. A detailed description of the actual milling machine was not provided because the thrust of this thesis was towards controller flexibility through software sophistication.

Algorithms for the various interpolators were introduced and the 'moving feed vector' mathematical approach was described. In this latter approach, a speed profile provides the magnitude of a feed vector

and a direction algorithm establishes the position and angle of the vector. The approach allows the interpolator designer to specify any curve in two or three dimensions. Specifications are simplified because only a unit tangent vector need be calculated because the magnitude of the curve is provided by the speed profile. The speed profile calculation requires only knowledge of the total curve length.

Chapter 4 will provide the details of tests which were performed to determine the accuracy of the milling machine's control system.

CHAPTER 4

PERFORMANCE

4.1 Introduction

Accuracy is the main criterion used in the evaluation of a machine drive system. Chapter 2 indicated that there are two toolbit motion errors to be considered, path errors and position errors. In this Chapter, the expected performance of individual machine axes will be considered based upon the mathematical model developed earlier. The performance predicted by the model will be compared to data from experimental tests in order to assess the drive axes' accuracy.

An examination of path errors which result from the interrelation of axes will be considered. Previously, a detailed examination of the trapezoidal speed profile was introduced. In this Chapter the overshoot correction scheme, implemented by compensation of the trapezoidal speed profile will be detailed. Note that this reference signal compensation is not the classical lead-lag compensator which was implemented in the Control Loops Unit. It is a further level of compensation which is necessary due to overshoot of the

toolbit's speed during deceleration. This overshoot must be corrected to prevent an error in the toolbit's final position. As an introduction, tests which were performed on the CLU will be detailed next.

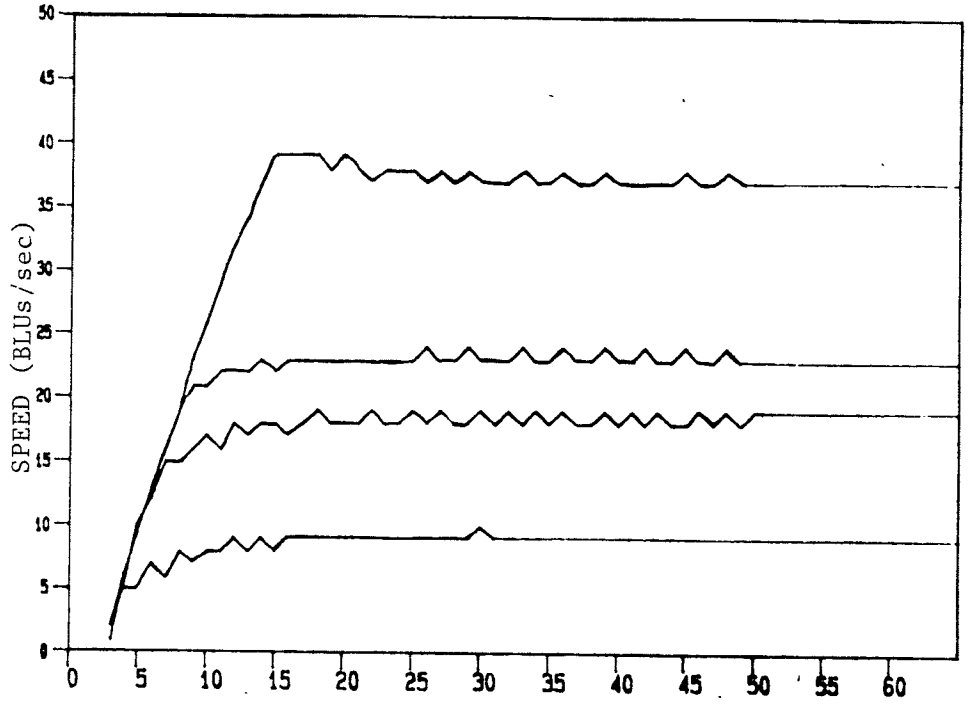
4.2 Performance of the Control Loops Unit

The mathematical analysis of Section 2.3.2 indicated that the drive system of the motors is a type '1' control system. It was indicated further that, for a constant feedrate speed, there will be zero speed error, $e_{ss\langle step\rangle} = 0$, with a finite position error (equal to 1/Kt x speed) in the steady state. Thus, if the toolbit travels in a line, for instance, the reference position is expected to lead the actual position by a constant distance which is inversely proportional to the control system's gain.

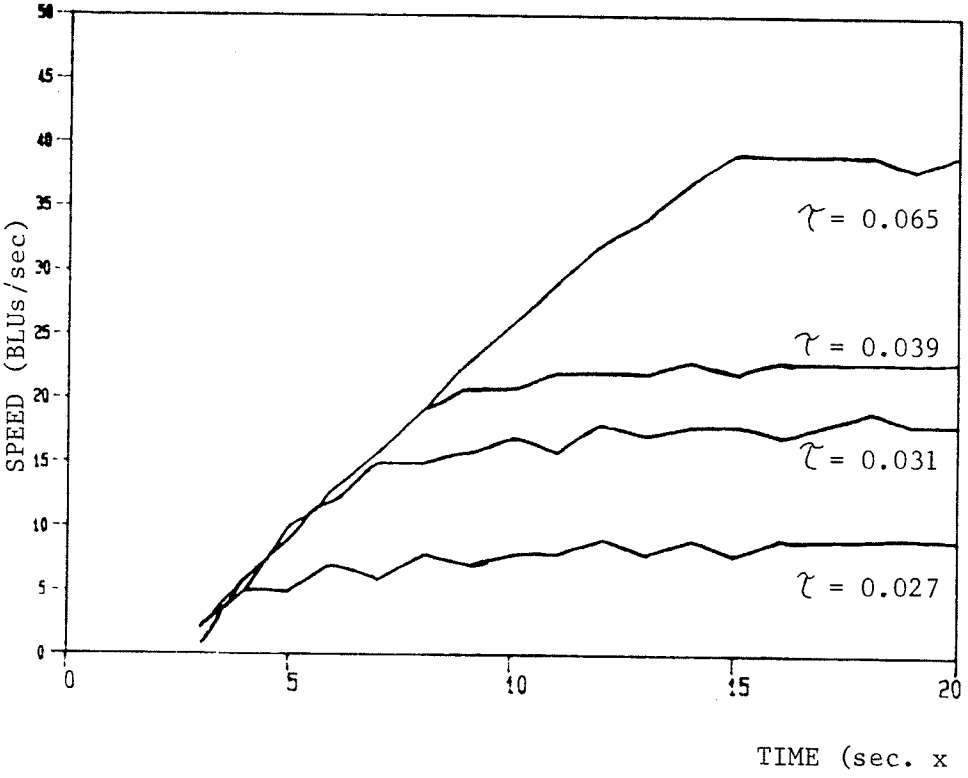
A type '1' system also has a known behavior to a ramp speed signal (which forms part of the trapezoidal speed profile). The speed error will be a constant, $ess_{\langle ramp\rangle}$ = 1/Kt x (constant) acceleration in the steady state. Thus, the position and the speed errors which arise from a given constant or linearly varying speed are no worse than inverse functions of the system's gain, Kt. Consequently, it is clearly advantageous to

employ high gain machine drive control systems.

Performance experiments began with a verification of the mathematical model and subsequently determined a qualitative measure of the motor-amplifier's non-linearities. Except for the loaded drive system's time constant, all the system parameters were found easily from hardware specifications of the manufacturers. To find the unknown time constant, the motor's amplifier drive was isolated by disconnecting the control system so that the encoder was used merely to measure, and not correct, the speed of the axis motor mounted on the mill. Various input step voltages were introduced independently to the reference speed terminal of the motor's amplifier and the resulting speeds were recorded. Overall results are presented in Figure 4.1a whilst the important details of the initial portions of the plots are given in Figure 4.1b. The time constants were found by using the method described in Section 2.3.1. There the time required for the motor to respond to 63.2% of a step value represents the loaded time constant of the motor. The various time constants are also shown in Figure 4.1b. All the initially varying speed portions of the plots have approximately the same slope which indicates that the motor's acceleration



(a)



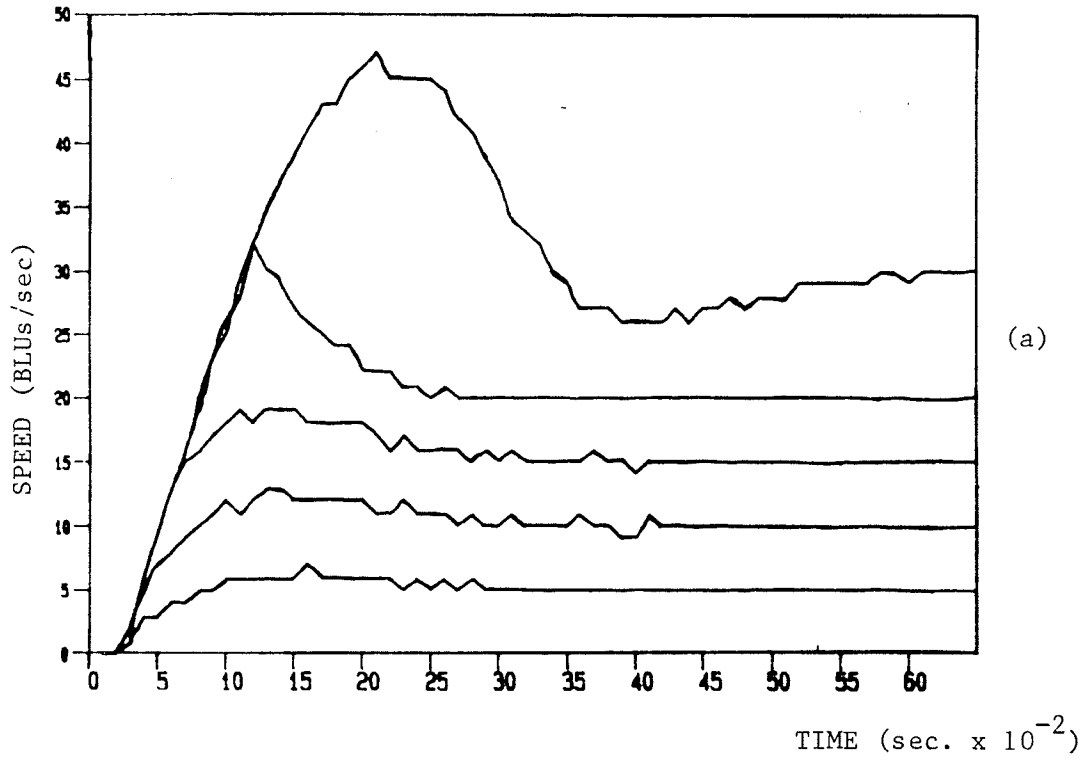
(b)

FIGURE 4.1 OPEN LOOP STEP RESPONSE OF MACHINE DRIVE SYSTEM

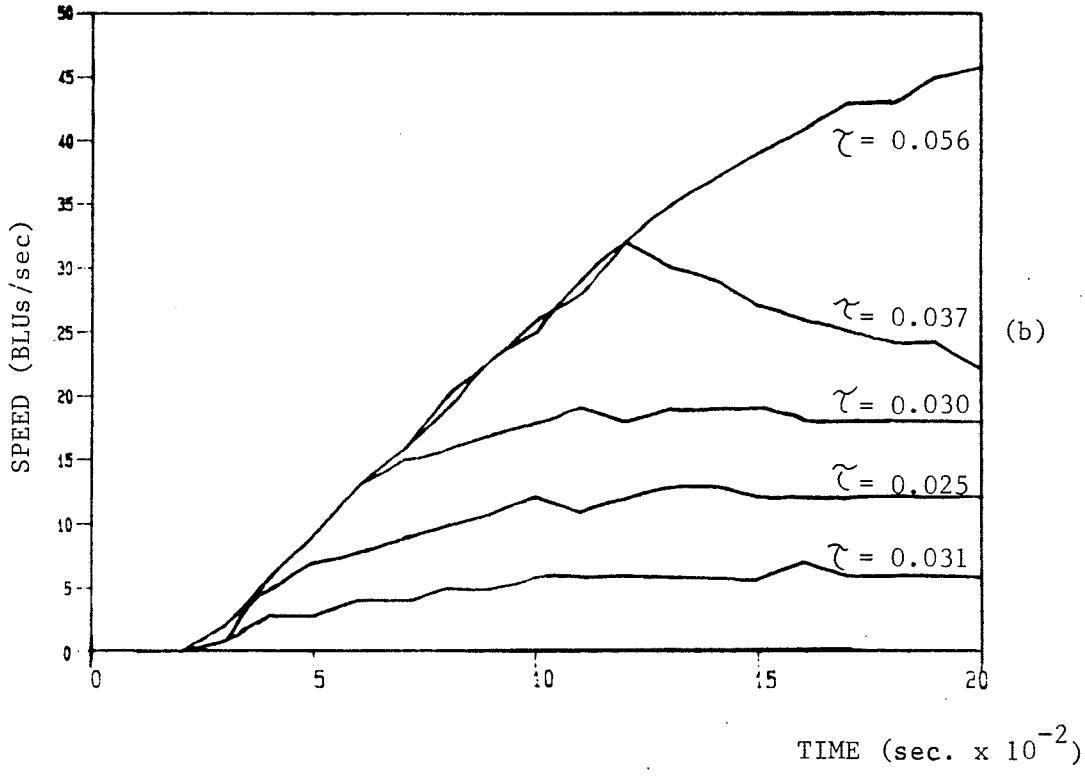
remained approximately constant for all input voltages. A truly linear behavior, on the other hand, would be reflected in accelerations becoming higher with larger inputs. This feature, however, is not present in Figure 4.1b which suggests that the current limitation of the amplifier prevented the greater surge of current required to further accelerate the motors. Now a machine drive would not be subjected to a step change in speed under normal use. Consequently, a time constant of 0.03 seconds was chosen for the loaded motor time constant because not only did it correspond to a low step but it also matched the time constant specified by the motor-amplifer's manufacturer. It should be also noted in passing that the demonstrated nonlinearity highlights the inability of any linear control idealization (no matter how many poles and zeros are used) to accurately model such a system. Indeed, the nonlinearity indicates that a linearized mathematical model alone is insufficient for design and the linear model is best utilized in conjunction with experimental methods.

After verifying the time constant, the feedback loop was reconnected and the control program was run with direct digital compensation by using step inputs

with various magnitudes. The overall results are shown in Figure 4.2a with initial details amplified in Figure 4.2b. Closing the loop of the control system reduced slightly the time constants of the drive. This reduction represents an improvement in the response speed of the control system. It was predicted by the compensated Bode plot, $G_{\ominus}$, whereby the compensated drive system had a slightly higher crossover frequency, Wc, in Figure 2.10. Unfortunately, a comparison of Figures 4.1 and 4.2 indicates that the closed loop system was less stable (higher overshoots) for large step inputs. Fortunately, the machine tool will not be required to instantaneously change speed because the interpolators use trapezoidal speed profiles. Indeed, for step speed changes up to 10 BLUs/int the system appears well behaved because the actual motor speed reaches the set speed quickly with little overshoot or oscillation. Tests were also performed in which the toolbit was moved in a line for a move and the steady state values of the motor speed and position were recorded. Approximately 1/3 second after the start, the motor's speed became steady and followed the reference speed without error. Corresponding position errors (ie. the lag of the cutter's actual location behind its reference position)



(a)



(b)

FIGURE 4.2 CLOSED LOOP STEP RESPONSE OF MACHINE DRIVE SYSTEM

are tabulated in Table 4.1 along with the position errors predicted by the mathematical model in which a gain,Kt, of 200 sec^{-1} was employed.

Speed (% of maximum)	Theoretical Error (BLUs) Gain=200 sec^{-1}	Actual Error BLUs Gain=200 sec-1
100	20	20
87.5	18	18
75	15	16
62.5	13	13
50	10	11
37.5	8	8
25	5	6
12.5	3	3
5	1	2
2.5	1	1

TABLE 4.1 STEADY STATE POSITION ERRORS FOR DIFFERENT CONSTANT SPEEDS OF THE MOTOR DRIVE

The invariably close agreement between the theoretical and actual position errors in Table 4.1 indicates that the compensator was performing as expected. Furthermore, it is interesting to compare the theoretical errors for a reduction in gain from 200 sec^{-1} to 16.7 sec^{-1} in order to underscore the importance of a high gain for the machine drive system. (Recall that 16.7 sec^{-1} was the gain for the uncompensated system having approximately the same damping factor as the compensated drive system.) This comparison is given in Table 4.2. It can be seen there

that an uncompensated position would lag behind the reference position by 240 BLUs or 0.024 inches at 100% of full speed. A compensated system, on the other hand, would lag by only 20 BLUs or 0.002 inches.

Speed (% of maximum)	Theoretical Error (BLUs) Gain=200 sec^{-1}	Theoretical Error (BLUs) Gain=16.7 sec^{-1}
100	20	240
87.5	18	216
75	15	180
62.5	13	156
50	10	120
37.5	8	96
25	5	60
12.5	3	36
5	1	12
2.5	1	6

TABLE 4.2 A COMPARISON OF THE STEADY STATE POSITION ERRORS FOR DIFFERENT CONSTANT SPEEDS OF AN UNCOMPENSATED AND COMPENSATED MOTOR DRIVE

The results of the tests performed on the Control Loops Unit may be summarized as:

a) The time constant of the servo drive system was found to vary due to the non-linearity of the amplifier-motor. The time constant specified by the manufacturer was adopted because it approximately matched the experimental values for the small signal changes likely in practice.

b) Closing the loop of the drive slightly improved

its response speed although the effects of the non-linearities were exaggerated.

c) The designed lead-lag compensator produced the predicted system gain of 200 sec^{-1}.

Verification of the behaviour of the Control Loops Unit established a basis for evaluating the performance of the Data Processing Unit.

4.3 Compensation and Performance of the Data Processing Unit

The Data Processing Unit is required to provide accurate reference speeds to the Control Loops Unit. Errors in such reference speeds would invalidate all efforts to maximize the performance of the Control Loops Unit because the Unit would then follow an erroneous signal. Fortunately, the sophisticated calculating power and floating point precision offered by FORTRAN allowed the Data Processing Unit to provide accurate reference speeds. In addition, the Data Processing Unit provided a further level of compensation to the Machine Control Unit. This compensation will be detailed next.

Section 3.2.2.2 indicated that the position error is predictable during a steady state motion. Thus, the position error is known before the controller begins to

decelerate the toolbit near the end of a cutting motion. During the deceleration portion of the move, however, the actual position of the cutter overshoots the reference position. Furthermore, the position error consistently increases and, hence, is greatest at the end of the move. Thus, the cutter will overshoot its desired final location and this is generally unacceptable. The overshoot was shown experimentally to be repeatable and, hence, it could be tabulated straightforwardly for various feedrates. Table 4.3 provides a sample list of the average overshoots at various speeds. It can be seen from this table that overshoots can be quite significant, especially for speeds higher than 12.5% of maximum speed.

Speed (% of maximum)	Overshoot (BLUs)
100	45
87.5	42
75	42
62.5	41
50	41
37.5	32
25	22
12.5	10
5	1
2.5	0

TABLE 4.3 TOOLBIT OVERSHOOT (BLUs) FOR VARIOUS SPEEDS

To correct for the overshoot, each feedrate was checked experimentally and its corresponding overshoot value was entered into a software data table. This table was used by the program to modify the deceleration phase of the interpolation to automatically compensate for the overshoot at the endpoint. Specifically, the deceleration loop timer counter, Pt3, was recalculated to cause the cutter to travel a distance which was exactly 'short' of the required distance by the tabulated overshoot value. This technique was successful in greatly reducing or eliminating the overshoot. Indeed, this real-time technique reduced all overshoots to less than 10 BLUs, which gave an accuracy better than 0.001 inches of toolbit motion.

A series of overshoot tests were performed in order to determine whether the cutter load would undermine the overshoot compensation. An aluminum block was cut with a 1/4 inch endmill, a spindle speed of 1400 RPM, and a depth of cut of 0.050 inches. The feedrate was varied between 0.05 and 0.300 inches per second. The mill was programmed to cut at angles of 0, 45, and 90 degrees, with the bit remaining in the block during the entire test. The overshoot of the endmill was found not to vary more than 0.0003 inches from the no-load to the full-load test. The maximum overshoot error during the

test was 0.0009 inches and, thus, the desired 0.001 inch accuracy was not violated. Now that the integrity of the mill's end position has been determined, the contour tests will be detailed.

4.4 Contour Tests

4.4.1 Contour Errors

Contour tests provide the definitive measure of a controller's accuracy. These tests examine the motion of the toolbit with all the controller's inaccuracies included.

Section 2.2.1 introduced the concepts of path and position errors. Path errors were described as position errors in which the cutter's actual location was offset from the desired reference curve. The tests described previously in this Chapter focussed upon the behaviour of a single axis to evaluate individual axis errors. The tests which will be described next were directed towards the errors which result when interrelated axes are mismatched.

Section 2.2.1 indicated that path errors can occur due to the mismatch of the time constants of two interrelated axes motors. These errors are noticeable

whenever the motor is in a transient state - for example when it is accelerating initially. On the other hand, during a steady state motion when the cutter is at the prescribed feedrate, a time constant mismatch will not cause significant path errors [3]. However, it can be shown [3] that, in a steady state motion, servo gain mismatch is the major contributor to contour errors. Ideally, it would be beneficial if all related axes had perfectly matched gains and time constants. Unfortunately, this is not generally possible in practice due to effects such as amplifier drift. The effects of gain mismatch will be described for both linear and circular interpolation.

Figure 4.3 shows how a gain mismatch causes path errors during linear interpolation. Figure 4.3 illustrates the corresponding motions along the X and Y axes, as a function of time, for a steady state, constant speed. The Y axis servo has a lower gain than that of the X axis servo so that the toolbit's projected toolbit motion has a path error. Note that if the Y axis servo had the same gain as that of the X axis, each axis would have individual position errors but the projection would still lie exactly upon the desired path. Poo et al [3] indicate that the maximum path error will occur at a 45 degree contour for linear motion.

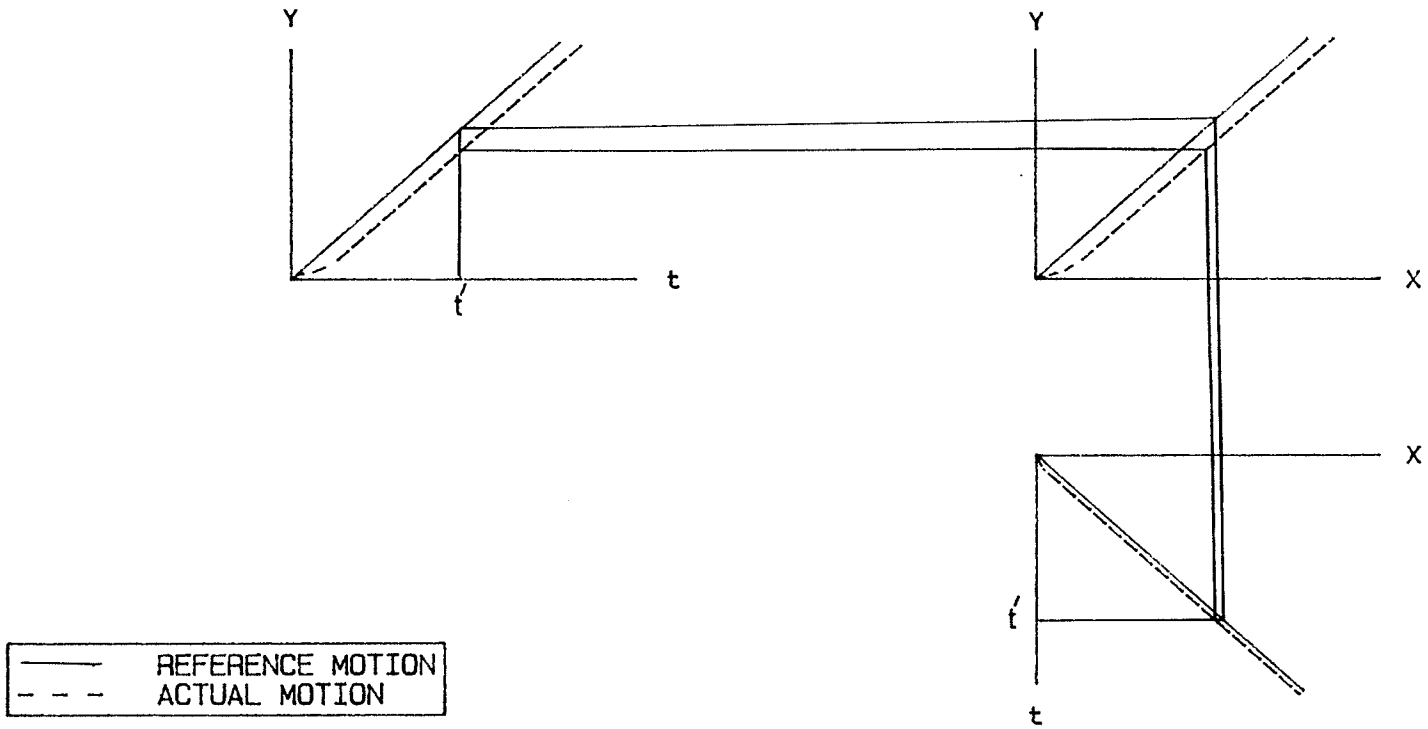


FIGURE 4.3 THE EFFECT OF GAIN MISMATCH ON CONTOUR ACCURACY DURING LINEAR INTERPOLATION

Figure 4.4 shows the effect of gain mismatch during circular motion. Each axis has a finite position error which is inversely proportional to the axis speed. Furthermore, the error grows with an increasing angular speed around the contour which implies that high feedrates and small arc radii will lead to the highest path errors. The gain mismatch causes the position errors to be different along each axis and this, in turn, results in an X-Y contour which is roughly elliptical. The maximum path errors occur at the midpoints of motion for each quadrant of motion.

Considering the source of error, it is easily seen that increasing the individual axis gains would reduce the path error because each axis position error would be correspondingly smaller. Also, circular arc interpolations at small radii should be made by using slow feedrates in order to lower the angular speed, and resulting contour errors, associated with the motion.

4.4.2 Linear Contour Tests

In the performance of the linear contour tests, the microcomputer was programmed to detect path errors

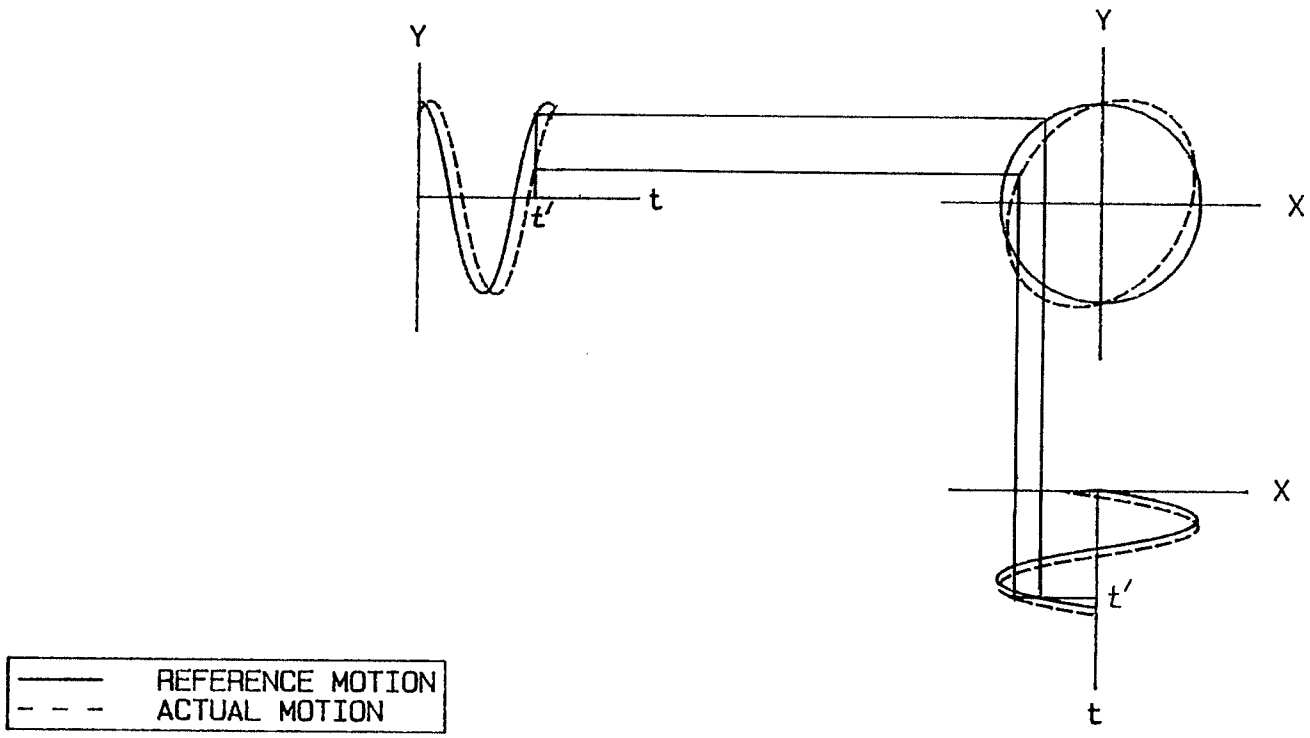


FIGURE 4.4 THE EFFECT OF GAIN MISMATCH ON CONTOUR ACCURACY DURING CIRCULAR INTERPOLATION

during any movement. These path errors represent the perpendicular distance of the cutter's actual position from the desired line segment. The toolbit was programmed to move at angles of 0, 22.5, 45, 67.5, and 90 degrees to the milling machine's X axis. This choice of angles produced a mix of contributions from the X and Y axis movements. Also, speeds were chosen to correspond to 12.5%, 25%, 37.5%, 50%, 75%, and 100% of the machine's feedrate maximum for every angle listed. These 30 combinations provided a sufficiently wide spectrum to indicate the accuracy of a supposedly linear path. Table 4.3 provides the path errors for the various speed - angle combinations with the errors further subdivided into those which occurred during the acceleration, full speed, and deceleration portions of the motions.

Speed (% of Full Speed)	Angle (degrees)	0	22.5	45	67.5	90
12.5	Acceleration	1	1	1	1	0
	Full Speed	1	1	1	1	1
	Deceleration	1	1	1	1	1
25	Acceleration	1	1	1	1	0
	Full Speed	1	1	1	1	1
	Deceleration	1	1	1	0	1
37.5	Acceleration	1	1	1	1	1
	Full Speed	1	1	1	1	1
	Deceleration	1	1	1	1	1
50	Acceleration	1	1	1	1	0
	Full Speed	1	1	1	1	1
	Deceleration	1	1	1	1	1
75	Acceleration	1	1	1	2	1
	Full Speed	1	1	2	1	1
	Deceleration	1	1	2	1	1
100	Acceleration	1	1	1	1	1
	Full Speed	1	8	7	3	1
	Deceleration	1	1	2	6	1

TABLE 4.4 - PATH ERRORS FOR LINEAR MOVES (BLUs)

An examination of the data in Table 4.4 reveals that all the contour errors were less than 10 BLUs which corresponds to a toolbit accuracy of better than 0.001 inches. As expected, path errors are more apparent for movements along both axes, particularly at the highest speed. They arise primarily from the slight mismatch between the gain of the two drive system's time constants.

4.4.3 Circular Contour Tests

Circular contour tests were performed in a similar

manner to the linear contour tests. The error in the radial position was recorded for a combination of speeds and circle radii. Table 4.5 gives the errors which were measured as described previously.

Speed (% of Full Speed)	Radius (inches)	0.5	1.0	2.0
12.5	Acceleration	0	0	1
	Full Speed	1	1	1
	Deceleration	1	1	1
25	Acceleration	1	1	1
	Full Speed	2	1	1
	Deceleration	1	1	1
37.5	Acceleration	1	1	1
	Full Speed	3	2	2
	Deceleration	3	2	2
50	Acceleration	2	1	1
	Full Speed	5	2	2
	Deceleration	3	3	2
75	Acceleration	1	2	1
	Full Speed	9	5	3
	Deceleration	4	4	2
100	Acceleration	1	1	1
	Full Speed	20	10	5
	Deceleration	4	4	2

TABLE 4.5 - PATH ERRORS FOR CIRCULAR INTERPOLATION (BLUs)

A comparison of the results in Table 4.4 and 4.5 indicates that circular interpolation is much more prone to contour errors than linear interpolation. As expected, at high feedrates, (which correspond to high angular speeds), the contour error is quite severe particularly at the smallest radii of motion, 0.5 inches. Note that, for a given feedrate, the smaller

the radii, then the higher is the angular speed of the axes. Table 4.5 indicates that for an arc radius of 1.0 inch or lower, the chosen feedrates should be less than 75% of the maximum feedrate to provide accurate control of the cutter.

4.5 Discussion of Overall Controller Performance

Certain key conclusions about the performance of a machine drive's control system can be drawn from the test results. Firstly, the non-linearities of the drive system can not be truly represented by a linear model, no matter how complex. Hence, experiments are required to optimize a drive system's performance. Secondly, the error driven nature of a drive system suggests that high overall system gains will improve the overall performance of the drive. Compensation is a viable method for increasing the gain of the drive without sacrificing stability.

Interrelated axis gain mismatch is seen to cause path errors. Fortunately, a high gain servo system tends to reduce the path errors caused by gain mismatch. Accepting that a small servo mismatch is inevitable in practice, it is desirable to reduce the speed of the cutter whenever high accuracy is desired. Lower cutting

speeds should be used with cuts of small arc radius.

It was suggested in Chapter 1 that, although the designed controller was only used for two axes of motion, the hardware could be expanded easily to a total of four axes. To verify this capacity, it is important to understand how much of the computer's capacity is utilized. To obtain a quantitative knowledge of the computer's utilization, a test was made in which the computer's 'wait states' were monitored during interpolation. This process was accomplished easily by programming the computer to turn on an input-output (I/O) line whenever the control program was waiting for an interrupt pulse. The I/O line was monitored by an oscilloscope. It was determined that the computer was waiting approximately 0.005 seconds out of every 0.010 second interrupt period. This observation implies that the computer's utilization was about 0.050/0.010 or 50%. Such a simple test indicates that it would not be unreasonable to use this controller to drive three and possibly four servo motors. To ensure sufficient intersample time, it would probably be necessary to reduce the sampling frequency to, say, 75 Hz. This frequency change would not undermine the system's

stability because 75 Hz. would still be at least 5 times the bandwidth of the machine drive.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER WORK

5.1 Conclusions and Discussion

An inexpensive microcomputer based, sampled data machine control system has been introduced. The choice of off-the-shelf components and standard development software packages helped to achieve a low cost. The controller was developed by using a high level language, FORTRAN, and a low level language, ASSEMBLER. The use of a floating point coprocessor gave the microcomputer the computational throughput required to implement real time floating point calculations. This real time capability allowed the machine's control unit to be programmed straightforwardy by utilizing the computational sophistication associated with FORTRAN. The ASSEMBLER portion of the control software performed the low level input/output work between the machine drive system and the microcomputer.

The advantages offered by real time floating point calculations are significant. The ability to perform digital compensation was shown to have obvious

performance advantages by allowing the controller to have a significantly higher gain without undermining the system's stability. Also, floating point capabilities allowed actual speed curve compensation by using the system's known behaviour to prevent final position overshoots.

Simple tests have been detailed which indicated how the implemented compensation techniques improved the system's overall performance. It was demonstrated that, due to the inherent non-linearities of the machine drive, it is unwise to rely totally upon a linear mathematical model. (A procedure which is generally followed in the relevant literature.) While a mathematical analysis is extremely useful, it should be supplemented with experimental tests to enable the designer to allow for, or be aware of, the effects of transient responses and non-linear effects.

5.2 Design Viability in the Commercial Market

The developed controller was built with a certain market niche in mind. This market is the machine retrofit or custom machine tool market. The flexibility of the controller could possibly make it extremely attractive. All the interpolation and control routines

were written in readily modified code whilst programming expertise for the IBM compatible microcomputers is found easily. Thus, it would not be too difficult for the control software to be modified for just about any application within the capability of the controller.

The computer utilization test described in Chapter 4 led to the conclusion that the microcomputer had the computational power to drive four axes for a sampling frequency of 75 Hz. with not unduly complex interpolation procedures. This reduced sampling frequency is sufficient for machine drive systems with a bandwidth of 7.5 to 15 Hz. which would include most machine tools. Thus, it appears that the controller could be used as a retrofit device on most typical machine tools. Under certain situations, such as a non-orthogonal Coordinate system on a specialized machine tool, it is quite conceivable that the interpolation calculations might exceed the microcomputer's capabilities. Assuming that the sampling frequency was at the boundary of 5 times the system's bandwidth, it might be concluded that the IBM (Intel 8088 based) microcomputer controller was inadequate. Fortunately, the control programs are written in standard languages and, thus, they are largely portable to more powerful microcomputers. For

example an Intel 80286 microcomputer has at least three times the computational capability of the 8088 and an Intel 80386 has at least 10 times the capability! This indicates that microcomputer power should not be a concern for most applications because a more powerful microcomputer can be chosen when required. This concern will become correspondingly less important as technology advances.

Certain enhancements would be necessary if it was desired to advance the developed controller to a marketable condition. First and foremost, the software should be improved to be extremely user friendly. A menu driven system could be developed to allow the users of the equipment to customize the software to their own machine environment. Essentially, the developed controller could be modified and customized for almost any application because of the flexibility and power of the software control scheme.

5.3 Control of Non-Ideal Machine Tools

It was mentioned early in this thesis that all references to accuracy assume that the machine's drive system is ideal beyond the servo motors. This implies that there is no backlash or nonlinearity in the

mechanical components which comprise the machine axes. While this might be true for an expensive high quality machine in good repair, it is generally not true for machines of medium and low quality. For the described project, the feedback encoders were mounted directly on the servo motors which are used to drive the axes. If linear encoders were mounted directly on the machine table, it would be possible for the computer to be aware of the table's actual position with all the mechanical errors included. This modification would allow the computer to compensate for backlash and non-linearity in the machine axes. The purchased experimental mill was inexpensive and simple tests quickly confirmed the severity of the axes' backlash. A scanning microscope was used to measure the cutter backlash on simple, single axis repetitive back and forth linear motions. It was found that the mechanical backlash for the tested axis was 0.020 inches! This machine tool would have definitely benefited from the installation of linear encoders. However, it is necessary to consider the effects of this backlash on the machine's control model.

Considering the control model developed in Section 2.3.1, mechanical backlash would be reflected as hysteresis. This hysteresis would lower the overall

control system's stability, the degree of destabilization depending upon the amount of backlash. At first glance, it would appear that this backlash would undermine the controller's effectiveness. Fortunately a simple solution is possible.

By using the flexibility of the control software, it would be simple to include software subroutines which would 'take up' the axes' backlash. These subroutines would monitor the motion of the cutter and, whenever an axis reversed direction, the subroutines could advance the cutter by a known tabulated amount. Better yet, they would advance the cutter until the encoder began to register a table motion. This technique would not affect the stability of the controller and would significantly reduce the errors caused by the backlash.

5.4 Recommendations for Further Work

In Section 4.4.1, the importance of interrelated axis gain match was introduced. Considering the importance of matched gains, it would be useful to add the capability for the controller to automatically match the gains of the interrelated axes. This could be done by performing a calibration program. The computer would

apply a known error word to each axis and monitor the actual speed of each axis motor without attempting to correct the motor speed. These actual speeds could be normalized and used to adjust the software gains of each motor to guarantee that the gains of the motors were matched accurately.

Another area of possible development would involve the addition of adaptive or optimum control techniques to the control algorithms. The required mathematical calculations could be programmed easily and modified as required.

In conclusion, the flexibility of the control software offered by the use of a high level programming language ensures the controller's versatility.

References

1. Koren, Yoram., Computer Control of Manufacturing Systems., McGraw Hill, New York: 1983.

2. Paul, Richard., Robot Manipulators: Mathematics, Programming, and Control., The MIT Press, Cambridge: 1981.

3. Poo, Aun-Neow., Bollinger, John G., and Younkin, George W., "Dynamic Errors in Type 1 Contouring Systems", IEEE Trans. Ind. App., vol. IA-8, no. 4, pp. 477-484, July/August, 1972.

4. Koren, Yoram., "Design Concepts of a CNC System", Proc. IEEE Ind. App. Soc., 10th Ann. Meeting, Atlanta, pp. 275-282, October, 1975.

5. Koren, Yoram., "Interpolator for a Computer Numerical Control System", IEEE Trans. Comp., vol. C-25, no. 1, pp. 32-37, January, 1976.

6. Danielson, Per E., "Incremental Curve Generation", IEEE Trans. Comp., vol C-19, no. 9, pp. 783-793, September, 1970.

7. Middleditch, A. E., "Design Criteria for Multi-Axis Closed Loop Computer Numerical Control Systems", Trans. ASME, J. Dyn. Sys., Meas. Contr., vol. 96, no. 1, pp. 36-40, March, 1974.

8. Electro-Craft Corporation, "DC Motors, Speed Controls, Servo Systems", Fifth Edition, 1980.

9. Koren, Yoram., "Design of Computer Control for Manufacturing Systems", Trans. ASME, J. Eng. Ind., vol. 101, no. 3, pp. 326-332, August, 1979.

10. Newland, D. E., An Introduction to Random Vibration and Spectral Analysis, 1st Edition, Longman Group Ltd., Essex: 1975.

11. Dorf, Richard., Modern Control Systems., 3rd Edition, Addison-Wesley, Reading: 1980.

12. DiStefano, Joseph J., Stubberud, Allen R., and

William, Ivan J., Feedback and Control Systems., SI (Metric) Edition, Schaum Outline Series, McGraw-Hill, New York: 1976.

13. VanLandingham, H. F., Introduction to Digital Control Systems., Macmillan Publishing Company, New York: 1985.

14. Kuo, Benjamin C., Digital Control Systems., Holt, Rinehart and Winston, Inc., New York: 1980.

15. Snyder, W. E., and Schott, G., "Using optical shaft encoders", Robotics Age, Fall 1980.

16. Snyder, W. E., Industrial Robots: Computer Interfacing and Control., Prentice-Hall, Inc., New Jersey: 1985.

Bibliography

Bergren, C., "A Simple Algorithm For Circular Interpolation", Contr. Eng., pp 57-59, September 1971.

Danielson, Per E., "Incremental Curve Generation", IEEE Trans. Comp., vol C-19, no. 9, pp. 783-793, September, 1970.

DiStefano, Joseph J., Stubberud, Allen R., and William, Ivan J., Feedback and Control Systems., SI (Metric) Edition, Schaum Outline Series, McGraw-Hill, New York: 1976.

Dorf, Richard., Modern Control Systems., 3rd Edition, Addison-Wesley, Reading: 1980.

EIA Standard RS-431, September 1976; Standard RS-274-D, February 1979; Standard RS-358-B, February 1979; Standard RS-227-A, October 1971; Standard RS-267-B, June 1983.

Electro-Craft Corporation, "DC Motors, Speed Controls, Servo Systems", Fifth Edition, 1980.

Evans, J. T., Kelling, L. U. C., "Inside the Mark Century Numerical Controls", Contr. Eng., vol. 10, pp. 112-118, May 1963.

Hariharan R., Sastry, P. A., "Position and Feed-Rate Control for Contouring Numerical-Control Systems", IEEE Trans. Ind. Elect. & Cont. Inst., pp 79-82, vol. IECI-26, No. 2, May 1979.

Kaiwa, Toshimasa., & Inaba, Seiuemon., "Latest Japanese Numerical Control Features", Contr. Eng., pp. 88-91 October 1961.

Koren, Yoram., "Computer-based machine-tool control", pp. 81-83, IEEE Spectrum, March 1977.

Koren, Yoram., Computer Control of Manufacturing Systems., McGraw Hill, New York: 1983.

Koren, Yoram., "Design Concepts of a CNC System", Proc.

IEEE Ind. App. Soc., 10th Ann. Meeting, Atlanta, pp. 275-282, October, 1975.

Koren, Yoram., "Design of a Digital Loop for Numerical Control", IEEE Trans., Ind. Elect. & Cont. Inst., pp 212-217, Vol. 25, No. 3, August 1978.

Koren, Yoram., "Design of Computer Control for Manufacturing Systems", Trans. ASME, J. Eng. Ind., vol 101, no. 3, pp. 326-332, August, 1979.

Koren, Yoram., "Interpolator for a Computer Numerical Control System", IEEE Trans. Comp., vol. C-25, no. 1, pp. 32-37, January, 1976.

Koren, Yoram., Bollinger, John G., "Design Parameters for Sampled-Data Drives for CNC Machine Tools", IEEE Trans. Ind. App., vol. 1A-14, no. 3, pp. 255-264, May, 1978.

Koren, Y., Masory, O., Reference-Pulse Circular Interpolators for CNC Systems", Trans. ASME, J. Eng. Ind., pp 131-136, Vol. 103, February 1981.

Koren, Y., Shani, A., & Ben-Uri, J., "Numerical Control of a Lathe", IEEE Trans. Ind. Gen. Appl., vol. 6, no. 2, pp. 175-179 March/April 1970.

Koren, Y., Shani, A., & Ben-Uri, J., "Overshoot correction in digital-control system" Control, pp 204-205, March 1969.

Kuo, Benjamin C., Digital Control Systems., Holt, Rinehart and Winston, Inc., New York: 1980.

Kusic, George L., "Feedback Design of Closed Contour Servomechanisms", IEEE Trans. Ind. App., pp 547-551, Vol. 1A-16, No. 4, July/August 1980.

Losic, N. A., Sedra, A. S., and Dewan, S. B., "A Phase-Locked DC Servo System for Contouring Numerical Control", IEEE Proc. Ind. App. Soc. 1981.

Mesniaeff, P. G., "The Technical Ins and Outs of Computerized Numerical Control - A Special Report", Contr. Eng. pp 65-84, March 1971.

Middleditch, A. E., "Design Criteria for Multi-Axis

Closed Loop Computer Numerical Control Systems", Trans. ASME, J. Dyn. Sys., Meas. Contr., vol. 96, no. 1, pp. 36-40, March, 1974.

Milner, D. A., "Some Aspects of Computer Numerical Control With Reference to Interpolation", Trans. ASME. J. Eng. Ind., pp 883-889, August 1976.

Newland, D. E., An Introduction to Random Vibration and Spectral Analysis, 1st Edition, Longman Group Ltd., Essex: 1975.

Paul, Richard., Robot Manipulators: Mathematics, Programming, and Control., The MIT Press, Cambridge: 1981.

Poo, Aun-Neow., Bollinger, John G., and Younkin, George W., "Dynamic Errors in Type 1 Contouring Systems", IEEE Trans. Ind. App., vol. IA-8, no. 4, pp. 477-484, July/August, 1972.

Schlueter, R. A., Retford, E. T., and Van Wieren, S., "The Optimal Machine Tool Control Problem", Trans. ASME J. Dyn. Sys., Meas., and Cont., pp 170-176, Vol. 100, Sept. 1978.

Schmitt, Lee E., " PC Versus CNC - Which Do You Choose?", IEEE Transactions on Industry Applications, Vol. 1A-20, No. 5. September/October 1984.

Seim, Thomas A., "Applying Microprocessors to Machine Tool Controller Design, Part 1", Computer Design, pp. 86-97, March 1980.

Seim, Thomas A., "Applying Microprocessors to Machine Tool Controller Design, Part 2", Computer Design, pp. 90-96, April 1980.
Snyder, W. E., and Schott, G., "Using optical shaft encoders", Robotics Age, Fall 1980.

Snyder, W. E., Industrial Robots: Computer Interfacing and Control., Prentice-Hall, Inc., New Jersey: 1985.

Tlusty, J., Elbestawi, M. A. A., "Analysis of Transients in an Adaptive Control Servomechanism for Milling With Constant Force"., Trans. ASME J. Eng. Ind., vol. 99, no. 3, pp. 766-772, August 1977.

VanLandingham, H. F., Introduction to Digital Control Systems., Macmillan Publishing Company, New York: 1985.

Zwitter, T., Hom, V., "Current Direction of Computerized Control"., IEEE Machine Tools Conference, Cincinnati, Ohio, USA. 1981, October 20-22.

APPENDIX I

Details of Digital Compensation

Pole Zero Mapping

The exact relationship between the continuous time s-plane and the discrete time z-plane is well known [13] as

$$z = e^{st} \qquad (A.1)$$

where:

t = sample period
e = 2.718282
s = Laplace Operator

The pole-zero technique simply translates critical frequencies from the s-plane to the z-plane. The corner frequencies of the transfer function are chosen as the critical frequencies. Also, the dc gain of the digital filter is matched to the dc gain of the continuous system.

Given a rational function G(s), the pole-zero mapping technique is performed as follows:

1) Each pole and finite zero of G(s) is calculated and transformed into the z-plane according to Equation 1,
2) Each zero of G(s) at s = infinity is mapped into z = -1,
3) The filter gains are matched at dc, so that G(0) = G*(1).

Example: Consider the following lag-lead compensator:

$$Gc(s) = \frac{(1 + s/8)\ (1 + s/33.3)}{(1 + s/1)\ \ (1 + s/235)} \qquad (A.2)$$

This can be mapped for a sample period of 0.01 seconds as follows,

The zero at s = -8 is mapped into

$$z = e^{-8.0(0.01)} = 0.923116$$

The zero at s = -33.3 is mapped into

$$z = e^{-33.3(0.01)} = 0.716770$$

The pole at s = -1 is mapped into

$$z = e^{-1.0(0.01)} = 0.990050$$

The pole at s = -235 is mapped into

$$z = e^{-235(0.01)} = 0.095369$$

Thus,

$$Gc^*(z) = \frac{K\,(z - 0.923116)\,(z - 0.716770)}{(z - 0.990050)\,(z - 0.095369)} \qquad (A.3)$$

but,

$$Gc^*(1) = 2.41925 * K = Gc(0) = 1 \qquad (A.4)$$

Hence,

$$K = 1 / 2.41925 = 0.413351$$

Therefore, the lag-lead compensator may be represented by:

$$Gc^*(z) = \frac{0.413351\,(z - 0.923116)\,(z - 0.716770)}{(z - 0.990050)\,(z - 0.095369)} \qquad (A.5)$$

Direct Digital Programming

A physically realizable discrete-data transfer function Gc(z) of a digital controller may be written as [14]:

$$Gc(z) = \frac{E_2(z)}{E_1(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_m z^{-m}}{a_0 + a_1 z^{-1} + a_1 z^{-2} + \ldots + a_n z^{-n}} \qquad (A.6)$$

where a_0 and b_0 must not be zero, m and n are positive integers, and $E_1(z)$ and $E_2(z)$ represent the z-transforms of the input and the output of the controller, respectively.

To conduct a direct digital programming of Gc(z), a cross-multiplication is performed and then the inverse z-transform of equation (A.6) is taken. Solving for

$e_2^*(t)$ will show that the present value of the output $e_2^*(t)$ depends upon the present and the past values of the input $e_1^*(t)$ as well as the past information of the output. This implies that data storage will be required.

Consider the z-transform shown in equation (A.5). Expanding the numerator and denominator and gathering terms, equation (A.5) may be rewritten as:

$$G_c(z) = \frac{E_2(z)}{E_1(z)} = \frac{0.413351\,z^2 - 0.67785\,z + 0.27350}{z^2 - 1.08542\,z + 0.09442}. \quad \text{(A.7)}$$

Dividing both the numerator and denominator by z^2 and rearranging yields:

$$G_c(z) = \frac{E_2(z)}{E_1(z)} = \frac{0.413351 - 0.67785\,z^{-1} + 0.27350\,z^{-2}}{1 - 1.08542\,z^{-1} + 0.09442\,z^{-2}}. \quad \text{(A.8)}$$

Taking the inverse transform and solving for E_2 yields:

$$\begin{aligned} E_2(t) = {} & 0.413351\,E_1 - 0.67785\,E_1(t-T) + \\ & + 0.27350\,E_1(t-2T) + 1.08542\,E_2(t-T) \\ & - 0.09442\,E_2(t-2T) \end{aligned} \quad \text{(A.9)}$$

This formula is easily programmed whereby:

E_1 = position error at the present sample.
$E_1(t-T)$ = position error at the previous sample.
$E_1(t-2T)$ = position error at the sample period two sample periods prior to the present sample period.

$E_2(t)$ = compensator output at the present sample.
$E_2(t-T)$ = compensator output at the previous error.
$E_2(t-2T)$ = compensator output at the sample period two sample periods prior to the present sample period.

APPENDIX II

Computer Hardware Specifications

The specifications for the computer system are as follows:

1) Microcomputer - The microcomputer must be an IBM Pc, or XT, or a true compatible. Any monitor with accompanying video card is acceptable. Two floppy diskettes and at least 256K of memory are recommended. If the computer is going to be used for control program modification, a fixed-disk is recommended. The computer must have an Intel 8087 math coprocessor installed.

2) Commercial Interface Board - The experimental apparatus used a commercial input output board. This TECMAR™ BASEBOARD consisted of four 8255A I/O chips with appropriate decoding circuits. Each axis of motion required 8 output lines for the DAC and 8 input lines from the external counter. In addition to these basic 16 I/O lines, 5 control lines were required. The 8255A provided 24 lines of completely programmable I/O and, thus, a single 8255A had the capability of providing the capability for an entire axis of motion.

3) Custom Encoder Interface Board - The experimental apparatus used two separate breadboards each of which contained an encoder interface. Ribbon cables from each board were connected to the BASEBOARD.

Software:

The software was developed by using Microsoft FORTRAN V3.3 and Microsoft MACRO ASSEMBLER V1.27. The programs were run under the PC-DOS 2.1 operating system.

APPENDIX III

PROGRAM LISTINGS

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
     1 $nofloatcalls
     2 $nodebug
     3 $storage:2
     4 $linesize:80
     5 $pagesize:54
     6 $title:'FORTRAN Driver for MILL Control'
     7 $subtitle:'COMMON BLOCK DEFINITIONS'
     8 $page
```

```
 9
10
11
12
13
14 ********************************************************************************
15 *                                                                              *
16 *              PROGRAM: M6                                                     *
17 *              AUTHOR: C. G. GADSBY                                            *
18 *              DATE: November 29, 1986                                         *
19 *                                                                              *
20 *        This program, in conjunction with the two assembler                   *
21 *        subroutines BASE6, and GETKY, provide a software                      *
22 *        environment for the path control of two axes of                       *
23 *        motion.  The program requires a two interface cards,                  *
24 *        a commercial I/O card, and a custom encoder interface.                *
25 *        These cards servo to marry the servo motor hardware to                *
26 *        the control software.                                                 *
27 *                                                                              *
28 *                                                                              *
29 *                                                                              *
30 *                                                                              *
31 *                                                                              *
32 ********************************************************************************
33 $page
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
34
35 ********************************************************************************
36 *                                                                              *
37 *   /PROG/    Part Program Common Block                                        *
38 *                                                                              *
39 *    tbnum  - total number of blocks in the part program.                      *
40 *    n() - N code array.                                                       *
41 *    g() - G code array.                                                       *
42 *    x() - X code array.                                                       *
43 *    y() - Y code array.                                                       *
44 *    z() - Z code array.                                                       *
45 *    i() - I code array.                                                       *
46 *    j() - J code array.                                                       *
47 *    k() - K code array.                                                       *
48 *    f() - F code (feedrate) array.                                            *
49 *    s() - S code (spindle speed) array.                                       *
50 *    t() - T code (tool) array.                                                *
51 *    m() - M code (miscellaneous) array.                                       *
52 *                                                                              *
53 *------------------------------------------------------------------------------*
54 *                                                                              *
55 *   /POS/   Position Common Block                                              *
56 *                                                                              *
57 *    px, py   Actual X-axis and Y-axis position respectfully.                  *
58 *    rx, ry   Reference X-axis and Y-axis position respectfully.               *
59 *                                                                              *
60 *                                                                              *
61 *------------------------------------------------------------------------------*
62 *                                                                              *
63 *   /ASSEMB/  Assembler Subroutine Variables.                                  *
64 *                                                                              *
65 *    intflag   This is the 'flag' which indicates that an interrupt            *
66 *              has occurred.  Under interrupt condition this integer           *
67 *              variable changes from a value of 0 to 1.                        *
68 *                                                                              *
69 *      flag0   This variable indicates that an X-axis encoder index            *
70 *              pulse has occurred.  If it is non-zero, which signals           *
71 *              that an index has occurred, it contains the                     *
72 *              incremental position change to the intersample index            *
73 *              occurance.                                                      *
74 *                                                                              *
75 *      flag1   This variable is similar to FLAG0 except that it                *
76 *              reports the status of the Y axis encoder.                       *
77 *                                                                              *
78 *       err0                                                                   *
79 *         &                                                                    *
80 *       err1   These are the error word values for the X and Y                 *
81 *              axis servo drives respectfully.                                 *
82 *                                                                              *
83 *     count0                                                                   *
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
     84 *       &                                                          *
     85 *     count1   These integer variables are the incremental count   *
     86 *              values.  They correspond to the change in position  *
     87 *              for each axis in a single interrupt period.         *
     88 *                                                                  *
     89 *------------------------------------------------------------------*
     90 *                                                                  *
     91 *     /DYN/    Dynamic Variables.                                  *
     92 *                                                                  *
     93 *         ke   This variable contains the encoder count in lines   *
     94 *              (or pulses) per revolution.                         *
     95 *                                                                  *
     96 *   ex1, ey1                                                       *
     97 *   ex2, ey2                                                       *
     98 *   ex3, ey3   These are variables required by the digital         *
     99 *              lead-lag filter.  They represent the filter         *
    100 *              outputs (X and Y) at the previous three samples     *
    101 *              ex1 and ey1 are the most recent filter outputs and  *
    102 *              ex3 and ey3 are the filter outputs three samples    *
    103 *              ago.                                                *
    104 *                                                                  *
    105 *   qx1, qy1                                                       *
    106 *   qx2, qy2                                                       *
    107 *   qx3, qy3   These are variables required by the digital         *
    108 *              lead-lag filter.  They represent the filter         *
    109 *              inputs (X and Y) at the previous three samples      *
    110 *              qx1 and qy1 are the most recent filter inputs and   *
    111 *              qx3 and qy3 are the filter inputs three samples     *
    112 *              ago.                                                *
    113 *                                                                  *
    114 *                                                                  *
    115 ********************************************************************
    116 $subtitle:'MODULE:  MAIN'
    117 $page
```

```
D Line# 1     7                    Microsoft FORTRAN77 V3.30 March 1985
    118
    119 ************************************************************************
    120 *                                                                      *
    121 *    MODULE:  MAIN                                                     *
    122 *                                                                      *
    123 *    This is the main module for the Mill controller. Its major        *
    124 *    function is to control program flow by invoking the correct       *
    125 *    subroutines to perform the requested tasks.  This subroutine      *
    126 *    also installs the assembler subprograms into memory and removes   *
    127 *    them at the end of the program to ensure a correct return to      *
    128 *    DOS.                                                              *
    129 *                                                                      *
    130 *    COMMON BLOCKS ACCESSIBLE:                                         *
    131 *                                  /pprog/                             *
    132 *                                  /pos/                               *
    133 *                                  /assemb/                            *
    134 *                                  /dyn/                               *
    135 *                                                                      *
    136 *                                                                      *
    137 *    LOCAL VARIABLES:                                                  *
    138 *                                                                      *
    139 *        cnt - Dummy index variable.                                   *
    140 *       acod - ASCII code of a pressed key.  This variable is          *
    141 *              returned by the GETKY assembler subroutine.             *
    142 *       scod - Scan code of a pressed key.  This variable is           *
    143 *              also returned by the GETKY assembler subroutine.        *
    144 *                                                                      *
    145 *   jogdelta - This variable contains the number of BLUs that          *
    146 *              the toolbit will move when a toolbit jog request        *
    147 *              is made by the user.                                    *
    148 *                                                                      *
    149 *                                                                      *
    150 ************************************************************************
    151
    152 $page
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    153       program CNCMILL
    154
    155       integer*2 tbnum, n(250), g(250), i(250), j(250), k(250), f(250),
    156      &s(250), t(250), m(250)
    157       integer*4 x(250), y(250), z(250)
    158       common /pprog/ tbnum, n, g, x, y, z, i, j, k, f, s, t, m
    159
    160       integer*4 px, py
    161       real rx, ry
    162       common /pos/ px, rx, py, ry
    163
    164       integer*2 intflag, flag0, flag1, err0, err1, count0, count1
    165       common /assemb/ intflag, flag0, flag1, err0, err1, count0, count1
    166
    167       real ex1, qx1, ex2, qx2, ex3, qx3
    168       real ey1, qy1, ey2, qy2, ey3, qy3
    169       integer*2 ke
    170       common /dyn/ ke, ex1, qx1, ex2, qx2, ex3, qx3,
    171      &ey1, qy1, ey2, qy2, ey3, qy3
    172
    173       integer*2 cnt, acod, scod
    174       real jogdelta
    175
    176 251   ke = 100
    177       rx=0.0
    178       ry=0.0
    179       px = 0
    180       py = 0
    181       errcnt = 1
    182       jogdelta = 1000.0
    183       call setup(intflag,flag0,flag1,err0,err1,count0,count1)
    184
    185       call video(0,5,1)
    186       write (*,100)
    187       write (*,101)
    188       write (*,100)
    189
    190 25    continue
    191       call video(1,16,38)
    192       write (*,102) jogdelta/10000.0
    193       call video(1,21,46)
    194       write (*,103) real(px/10000.0)
    195       call video(1,22,46)
    196       write (*,104) real(py/10000.0)
    197
    198 100   format (1x,'**************************************************')
    199 101   format (1x,/,
    200      &10x,'Use the CURSOR for jog control',//,10x,'Function keys:',//
    201      &10x,'F1 Sets Zeropoint       F2 =  0.001 inches',/
    202      &10x,'                        F4 =  0.010 inches',/
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    203      &10x,'F5 Quits Program       F6 =  0.100 inches',/
    204      &10x,'                       F8 =  1.000 inches',/
    205      &10x,'F9 Runs Program       F10 =  5.000 inches',/
    206      &10x,'       Jog Currently Set at: ',//)
    207 102   format (1x,F6.3,' inches')
    208 103   format (1x,'X position',F6.3,' inches')
    209 104   format (1x,'Y position',F6.3,' inches')
    210
    211       ex1 = 0.0
    212       qx1 = 0.0
    213       ex2 = 0.0
    214       qx2 = 0.0
    215       ex3 = 0.0
    216       qx3 = 0.0
    217       ey1 = 0.0
    218       qy1 = 0.0
    219       ey2 = 0.0
    220       qy2 = 0.0
    221       ey3 = 0.0
    222       qy3 = 0.0
    223       call getky(acod,scod)
    224       if (scod .eq. 60) then
    225            jogdelta = 10.0
    226       endif
    227       if (scod .eq. 62) then
    228            jogdelta = 100.0
    229       endif
    230       if (scod .eq. 64) then
    231            jogdelta = 1000.0
    232       endif
    233       if (scod .eq. 66) then
    234            jogdelta = 10000.0
    235       endif
    236       if (scod .eq. 68) then
    237            jogdelta =  50000.0
    238       endif
    239       if (scod .eq. 67) then
    240            goto 200
    241       endif
    242       if (scod .eq. 63) then
    243            goto 334
    244       endif
    245       if (scod .eq. 59) then
    246            px = 0
    247            py = 0
    248            rx = 0.0
    249            ry = 0.0
    250       endif
    251
    252       if (scod .eq. 75) then
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
    253           if (jogdelta .lt. 1000) then
    254                rx = rx - jogdelta
    255                call moton
    256                call tweak
    257                call motoff
    258           else
    259                call linear(int4(rx-jogdelta),int4(ry),40.0)
    260           endif
    261       endif
    262       if (scod .eq. 77) then
    263           if (jogdelta .lt. 1000) then
    264           rx = rx + jogdelta
    265           call moton
    266           call tweak
    267           call motoff
    268           else
    269                call linear(int4(rx+jogdelta),int4(ry),40.0)
    270           endif
    271       endif
    272       if (scod .eq. 72) then
    273           if (jogdelta .lt. 1000) then
    274           ry = ry + jogdelta
    275           call moton
    276           call tweak
    277           call motoff
    278           else
    279                call linear(int4(rx),int4(ry+jogdelta),40.0)
    280           endif
    281       endif
    282       if (scod .eq. 80) then
    283           if (jogdelta .lt. 1000) then
    284           ry = ry - jogdelta
    285           call moton
    286           call tweak
    287           call motoff
    288           else
    289                call linear(int4(rx),int4(ry-jogdelta),40.0)
    290           endif
    291       endif
    292       goto 25
    293
    294 200   continue
    295
    296       call video(0,1,1)
    297       call fileread
    298       call video(0,1,1)
    299
    300 c-----------------------------------------------------------------
    301 c Write out the file on the screen:
    302       write (*,106)
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    303       do 2, cnt = 1, tbnum
1   304       write (*, 105) n(cnt), g(cnt), x(cnt), y(cnt), z(cnt), i(cnt),
1   305      &j(cnt), k(cnt), f(cnt), s(cnt), t(cnt), m(cnt)
1   306 2     continue
    307 106   format(1x,/)
    308 105   format (1x, 'N', i3, 1x, 'G', i2, 1x, 'X', i6, 1x, 'Y', i6, 1x,
    309      &'Z', i6, 1x, 'I', i6, 1x, 'J', i6, 1x, 'K', i6, 1x, 'F', i4,
    310      &1x, 'S', i3, 1x, 'T', i2, 1x, 'M', i2)
    311
    312       pause 'Strike Enter to continue . . .'
    313 c
    314 c-------------------------------------------------------------------
    315
    316       call video(0,1,1)
    317       call decode
    318       call clsup
    319       goto 251
    320
    321 334   call clsup
    322       stop 'END of MAIN'
    323       end

Name    Type         Offset P Class

ACOD    INTEGER*2       690
CNT     INTEGER*2       694
COUNT0  INTEGER*2        10   /ASSEMB/
COUNT1  INTEGER*2        12   /ASSEMB/
ERR0    INTEGER*2         6   /ASSEMB/
ERR1    INTEGER*2         8   /ASSEMB/
ERRCNT  REAL             12
EX1     REAL              2   /DYN   /
EX2     REAL             10   /DYN   /
EX3     REAL             18   /DYN   /
EY1     REAL             26   /DYN   /
EY2     REAL             34   /DYN   /
EY3     REAL             42   /DYN   /
F       INTEGER*2      5502   /PPROG /
FLAG0   INTEGER*2         2   /ASSEMB/
FLAG1   INTEGER*2         4   /ASSEMB/
G       INTEGER*2       502   /PPROG /
I       INTEGER*2      4002   /PPROG /
INT4                          INTRINSIC
INTFLA  INTEGER*2         0   /ASSEMB/
J       INTEGER*2      4502   /PPROG /
JOGDEL  REAL             16
K       INTEGER*2      5002   /PPROG /
KE      INTEGER*2         0   /DYN   /
M       INTEGER*2      7002   /PPROG /
N       INTEGER*2         2   /PPROG /
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

PX	INTEGER*4	0	/POS /
PY	INTEGER*4	8	/POS /
QX1	REAL	6	/DYN /
QX2	REAL	14	/DYN /
QX3	REAL	22	/DYN /
QY1	REAL	30	/DYN /
QY2	REAL	38	/DYN /
QY3	REAL	46	/DYN /
REAL			INTRINSIC
RX	REAL	4	/POS /
RY	REAL	12	/POS /
S	INTEGER*2	6002	/PPROG /
SCOD	INTEGER*2	692	
T	INTEGER*2	6502	/PPROG /
TBNUM	INTEGER*2	0	/PPROG /
X	INTEGER*4	1002	/PPROG /
Y	INTEGER*4	2002	/PPROG /
Z	INTEGER*4	3002	/PPROG /

```
324
325 $subtitle:'MODULE:  SUBROUTINE DECODE'
326 $page
```

```
    327 ********************************************************************************
    328 *                                                                              *
    329 *  SUBROUTINE DECODE                                                           *
    330 *                                                                              *
    331 *    This subroutine accepts the part program as input and invokes             *
    332 *    the appropriate interpolation subroutines for a given block.              *
    333 *    The subroutine invokes one of the interpolators for each                  *
    334 *    block of the part program.                                                *
    335 *                                                                              *
    336 *    COMMON BLOCKS ACCESSED:                                                   *
    337 *                                /dyn/                                         *
    338 *                                /pos/                                         *
    339 *                                /pprog/                                       *
    340 *                                                                              *
    341 *    LOCAL VARIABLES                                                           *
    342 *                                                                              *
    343 *         bnum - index counter for part program block number.                  *
    344 *         junk - dummy input variable for manually moving the                  *
    345 *                z - axis of the milling machine.                              *
    346 *                                                                              *
    347 *    NOTES:                                                                    *
    348 *         This subroutine zeros the digital compensation                       *
    349 *         parameters at the beginning of each segment of                       *
    350 *         motion.                                                              *
    351 *                                                                              *
    352 *                                                                              *
    353 ********************************************************************************
    354       subroutine decode
    355
    356       integer*2 tbnum, n(250), g(250), i(250), j(250), k(250), f(250),
    357      &s(250), t(250), m(250)
    358       integer*4 x(250), y(250), z(250)
    359       common /pprog/ tbnum, n, g, x, y, z, i, j, k, f, s, t, m
    360
    361       integer*4 px, py
    362       real rx, ry
    363       common /pos/ px, rx, py, ry
    364
    365       real ex1, qx1, ex2, qx2, ex3, qx3
    366       real ey1, qy1, ey2, qy2, ey3, qy3
    367       integer*2 ke
    368       common /dyn/ ke, ex1, qx1, ex2, qx2, ex3, qx3,
    369      &ey1, qy1, ey2, qy2, ey3, qy3
    370
    371       integer*2 bnum
    372       data bnum/0/
    373
    374
    375       do 1, bnum = 1, tbnum
  1 376       ex1 = 0.0
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
1   377       qx1 = 0.0
1   378       ex2 = 0.0
1   379       qx2 = 0.0
1   380       ex3 = 0.0
1   381       qx3 = 0.0
1   382       ey1 = 0.0
1   383       qy1 = 0.0
1   384       ey2 = 0.0
1   385       qy2 = 0.0
1   386       ey3 = 0.0
1   387       qy3 = 0.0
1   388            call video(1,1,60)
1   389            write(*,'(1x,i3)') n(bnum)
1   390            call video(1,22,30)
1   391            write(*,'(1x,''X POSITION = '',i8,'' Y POSITION = '',i8)')
1   392       &    px, py
1   393
1   394            if(g(bnum) .eq. 00) then
1   395                 call linear(x(bnum),y(bnum),40.0)
1   396            endif
1   397            if(g(bnum) .eq. 01) then
1   398                 call linear(x(bnum),y(bnum),real(f(bnum)))
1   399            endif
1   400            if ((g(bnum) .eq. 02) .or. (g(bnum) .eq. 03)) then
1   401                 call circular(x(bnum), y(bnum), int4(i(bnum)),
1   402       &         int4(j(bnum)), real(f(bnum)), g(bnum))
1   403            endif
1   404            if(g(bnum) .eq. 20) then
1   405                 call video(1,1,20)
1   406                 write (*,'(1x,''Move Z - axis to '',I6,'' inches'')')
1   407       &         z(bnum)
1   408                 call video(1,3,20)
1   409                 write (*,'(1x,''Strike ENTER to continue.'')')
1   410                 read (*,'(BN,I5)') junk
1   411                 call video(1,1,20)
1   412                 write (*,'(1x,''                                  '')')
1   413                 call video(1,3,20)
1   414                 write (*,'(1x,''                            '')')
1   415            endif
1   416 1     continue
    417       return
    418       end

Name    Type        Offset P Class

BNUM    INTEGER*2      898
EX1     REAL             2   /DYN   /
EX2     REAL            10   /DYN   /
EX3     REAL            18   /DYN   /
EY1     REAL            26   /DYN   /
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

EY2	REAL	34	/DYN /
EY3	REAL	42	/DYN /
F	INTEGER*2	5502	/PPROG /
G	INTEGER*2	502	/PPROG /
I	INTEGER*2	4002	/PPROG /
INT4			INTRINSIC
J	INTEGER*2	4502	/PPROG /
JUNK	INTEGER*2	902	
K	INTEGER*2	5002	/PPROG /
KE	INTEGER*2	0	/DYN /
M	INTEGER*2	7002	/PPROG /
N	INTEGER*2	2	/PPROG /
PX	INTEGER*4	0	/POS /
PY	INTEGER*4	8	/POS /
QX1	REAL	6	/DYN /
QX2	REAL	14	/DYN /
QX3	REAL	22	/DYN /
QY1	REAL	30	/DYN /
QY2	REAL	38	/DYN /
QY3	REAL	46	/DYN /
REAL			INTRINSIC
RX	REAL	4	/POS /
RY	REAL	12	/POS /
S	INTEGER*2	6002	/PPROG /
T	INTEGER*2	6502	/PPROG /
TBNUM	INTEGER*2	0	/PPROG /
X	INTEGER*4	1002	/PPROG /
Y	INTEGER*4	2002	/PPROG /
Z	INTEGER*4	3002	/PPROG /

```
419
420 $subtitle:'MODULE:  SUBROUTINE LINEAR'
421 $page
```

```
D Line# 1    7                          Microsoft FORTRAN77 V3.30 March 1985
   422 ********************************************************************************
   423 *                                                                              *
   424 *  SUBROUTINE LINEAR                                                           *
   425 *                                                                              *
   426 *  This subroutine performs linear interpolation.                              *
   427 *  This subroutine is invoked by DECODE during automatic                       *
   428 *  program control and by module MAIN during jogging                           *
   429 *  operations.                                                                 *
   430 *                                                                              *
   431 *  COMMON BLOCKS ACCESSED:                                                     *
   432 *                                                                              *
   433 *                      /pos/                                                   *
   434 *                                                                              *
   435 *  RECEIVED VARIABLES:                                                         *
   436 *                                                                              *
   437 *            xfinal - this specifies the desired absolute                      *
   438 *                     final x value of the cutter for the                      *
   439 *                     motion segment at hand.                                  *
   440 *                                                                              *
   441 *            yfinal - this specifies the desired absolute                      *
   442 *                     final y value of the cutter for the                      *
   443 *                     motion segment at hand.                                  *
   444 *                                                                              *
   445 *              rate - this parameter specifies the desired                     *
   446 *                     maximum feedrate along the line.                         *
   447 *                                                                              *
   448 *                                                                              *
   449 *                                                                              *
   450 *  LOCAL VARIABLES:                                                            *
   451 *                                                                              *
   452 *          costheta - this variable represents the cosine                      *
   453 *                     value of the present theta angle.                        *
   454 *                     This variable is used to prevent                         *
   455 *                     repeated calculations of the same value.                 *
   456 *                                                                              *
   457 *          sintheta - this variable represents the sine                        *
   458 *                     value of the present theta angle.                        *
   459 *                     This variable is used to prevent                         *
   460 *                     repeated calculations of the same value.                 *
   461 *                                                                              *
   462 *               acl - this value represents the acceleration                   *
   463 *                     of the toolbit in units of BLUS/INT/INT                  *
   464 *                     where INT = period of sample.                            *
   465 *                                                                              *
   466 *             decel - this value represents the deceleration                   *
   467 *                     of the toolbit in units of BLUS/INT/INT                  *
   468 *                     where INT = period of sample.                            *
   469 *                                                                              *
   470 *             areal - represents the distance travelled while                  *
   471 *                     the toolbit is accelerating.                             *
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    472 *                                                                    *
    473 *              area2 - represents the distance travelled while       *
    474 *                      the toolbit is at the maximum feedrate.       *
    475 *                                                                    *
    476 *              area3 - represents the distance travelled while       *
    477 *                      the toolbit is decelerating.  This area       *
    478 *                      is modified by 'fudge' to allow for the       *
    479 *                      expected overshoot of the cutter during       *
    480 *                      deceleration.                                 *
    481 *                                                                    *
    482 *                pt1 - the number of interrupt periods at the        *
    483 *                      acceleration rate acl required to bring       *
    484 *                      the toolbit speed from zero to the            *
    485 *                      desired feedrate.                             *
    486 *                                                                    *
    487 *                pt2 - the number of interrupt periods at the        *
    488 *                      specified feedrate required to cause the      *
    489 *                      toolbit to travel a distance of area2.        *
    490 *                                                                    *
    491 *                pt3 - the number of interrupt periods at the        *
    492 *                      deceleration rate decel required to bring     *
    493 *                      the toolbit speed from the specified          *
    494 *                      feedrate to zero.                             *
    495 *                                                                    *
    496 *              fudge - tabulated value of expected toolbit           *
    497 *                      overshoot as a function of feedrate.          *
    498 *                                                                    *
    499 *              delta - total distance to travel                      *
    500 *                                                                    *
    501 *             deltax - x component of total distance to travel       *
    502 *                                                                    *
    503 *             deltay - y component of total distance to travel       *
    504 *                                                                    *
    505 *              theta - angle of line on the mill table with          *
    506 *                      reference to the mill's coordinate system     *
    507 *                                                                    *
    508 *                 pi - 3.141592 . . .                                *
    509 *                                                                    *
    510 *                vel - desired reference velocity for the toolbit    *
    511 *                                                                    *
    512 *                 vx - x component of vel                            *
    513 *                                                                    *
    514 *                 vy - y component of vel                            *
    515 *                                                                    *
    516 *                                                                    *
    517 **********************************************************************
    518       subroutine linear(xfinal, yfinal, rate)
    519       integer*4 xfinal, yfinal
    520       real rate, costheta, sintheta
    521       real acl, decel, areal, area2, area3, vx, vy, vel
```

```
D Line# 1     7
    522
    523       integer*4 pt1, pt2, pt3, i
    524       real delta, deltax, deltay
    525       real fudge
    526
    527       integer*4 px, py
    528       real rx, ry
    529       common /pos/ px, rx, py, ry
    530
    531       real pi, theta
    532       acl = 2.00
    533       decel = 1.00
    534       pi = real(3.1415926535)
    535
    536
    537 c-------------------------------------------------------------
    538 c Set up the initial parameters.
    539 c
    540 c Determine the angle of motion.
    541 c
    542       deltax = real(xfinal) - rx
    543       deltay = real(yfinal) - ry
    544
    545       if ((deltax .eq. 0) .and. (deltay .eq. 0)) then
    546           return
    547       endif
    548
    549       delta=sqrt(deltax**2+deltay**2)
    550       theta = atan2(deltay, deltax)
    551       costheta = cos(theta)
    552       sintheta = sin(theta)
    553       delta = abs(delta)
    554
    555 10    area1 = 0.5 * rate * rate / acl
    556       area3 = 0.5 * rate * rate / decel
    557       area2 = delta - area1 - area3
    558
    559       if (ifix(area2/rate) .le. 1) then
    560           rate = rate * 0.9
    561           goto 10
    562       endif
    563
    564       if(ifix(rate) .eq. 40) fudge = 45.0
    565       if(ifix(rate) .eq. 39) fudge = 44.5
    566       if(ifix(rate) .eq. 38) fudge = 44.5
    567       if(ifix(rate) .eq. 37) fudge = 43.0
    568       if(ifix(rate) .eq. 36) fudge = 44.0
    569       if(ifix(rate) .eq. 35) fudge = 42.5
    570       if(ifix(rate) .eq. 34) fudge = 43.0
    571       if(ifix(rate) .eq. 33) fudge = 43.0
```

```
     572        if(ifix(rate) .eq. 32) fudge = 43.0
     573        if(ifix(rate) .eq. 31) fudge = 42.5
     574        if(ifix(rate) .eq. 30) fudge = 42.0
     575        if(ifix(rate) .eq. 29) fudge = 43.0
     576        if(ifix(rate) .eq. 28) fudge = 43.5
     577        if(ifix(rate) .eq. 27) fudge = 42.0
     578        if(ifix(rate) .eq. 26) fudge = 44.0
     579        if(ifix(rate) .eq. 25) fudge = 41.5
     580        if(ifix(rate) .eq. 24) fudge = 41.5
     581        if(ifix(rate) .eq. 23) fudge = 40.0
     582        if(ifix(rate) .eq. 22) fudge = 42.5
     583        if(ifix(rate) .eq. 21) fudge = 38.0
     584        if(ifix(rate) .eq. 20) fudge = 41.0
     585        if(ifix(rate) .eq. 19) fudge = 38.5
     586        if(ifix(rate) .eq. 18) fudge = 39.0
     587        if(ifix(rate) .eq. 17) fudge = 35.5
     588        if(ifix(rate) .eq. 16) fudge = 36.5
     589        if(ifix(rate) .eq. 15) fudge = 32.5
     590        if(ifix(rate) .eq. 14) fudge = 31.5
     591        if(ifix(rate) .eq. 13) fudge = 28.5
     592        if(ifix(rate) .eq. 12) fudge = 27.5
     593        if(ifix(rate) .eq. 11) fudge = 27.0
     594        if(ifix(rate) .eq. 10) fudge = 22.0
     595        if(ifix(rate) .eq. 09) fudge = 20.5
     596        if(ifix(rate) .eq. 08) fudge = 18.5
     597        if(ifix(rate) .eq. 07) fudge = 16.0
     598        if(ifix(rate) .eq. 06) fudge = 12.0
     599        if(ifix(rate) .eq. 05) fudge = 09.5
     600        if(ifix(rate) .eq. 04) fudge = 07.0
     601        if(ifix(rate) .eq. 03) fudge = 05.0
     602        if(ifix(rate) .eq. 02) fudge = 00.5
     603        if(ifix(rate) .eq. 01) fudge = 00.0
     604
     605        pt1 = int4(rate/acl)
     606        pt2 = int4(area2/rate)
     607
     608        vx = 0.0
     609        vy = 0.0
     610        vel = 0.0
     611
     612        call moton
     613
     614        do 2, i=1, pt1
1    615            vel = vel + acl
1    616            vx = vel*costheta
1    617            vy = vel*sintheta
1    618            call wait(vx, vy, 0)
1    619 2      continue
     620        do 3, i=1, pt2
1    621            vel = rate
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
1   622           vx = vel*costheta
1   623           vy = vel*sintheta
1   624           call wait(vx, vy, 0)
1   625 3     continue
    626       area3 = sqrt(real((xfinal-int4(rx))**2+(yfinal-int4(ry))**2)) +
    627      &rate/2.0 - fudge
    628
    629       decel = 0.5 * rate * rate / area3
    630       pt3 = int4(rate/decel)
    631       do 4, i = 1, pt3
1   632           vel = vel - decel
1   633           vx = vel*costheta
1   634           vy = vel*sintheta
1   635           call wait(vx, vy, 0)
1   636 4     continue
    637       goto 5
    638
    639 5     continue
    640       vx = 0.0
    641       vy = 0.0
    642       rx = real(xfinal)
    643       ry = real(yfinal)
    644       call tweak
    645       call motoff
    646       return
    647       end
```

Name	Type	Offset	P	Class
ABS				INTRINSIC
ACL	REAL	904		
AREA1	REAL	940		
AREA2	REAL	948		
AREA3	REAL	944		
ATAN2				INTRINSIC
COS				INTRINSIC
COSTHE	REAL	932		
DECEL	REAL	908		
DELTA	REAL	924		
DELTAX	REAL	916		
DELTAY	REAL	920		
FUDGE	REAL	952		
I	INTEGER*4	976		
IFIX				INTRINSIC
INT4				INTRINSIC
PI	REAL	912		
PT1	INTEGER*4	956		
PT2	INTEGER*4	960		
PT3	INTEGER*4	988		
PX	INTEGER*4	0		/POS /

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
PY      INTEGER*4        8   /POS   /
RATE    REAL             8 *
REAL                         INTRINSIC
RX      REAL             4   /POS   /
RY      REAL            12   /POS   /
SIN                          INTRINSIC
SINTHE  REAL           936
SQRT                         INTRINSIC
THETA   REAL           928
VEL     REAL           972
VX      REAL           964
VY      REAL           968
XFINAL  INTEGER*4        0 *
YFINAL  INTEGER*4        4 *
```

```
    648
    649 $subtitle:'MODULE:  SUBROUTINE CIRCULAR'
    650 $page
```

```
D Line# 1    7                          Microsoft FORTRAN77 V3.30 March 1985
   651 ********************************************************************************
   652 *                                                                              *
   653 *  SUBROUTINE CIRCULAR                                                         *
   654 *                                                                              *
   655 *       This subroutine performs circular interpolation of the                 *
   656 *  toolbit.                                                                    *
   657 *                                                                              *
   658 *  RECEIVED VARIABLES:                                                         *
   659 *                                                                              *
   660 *     pt2i, pt2j - represent the X and Y coordinates which                     *
   661 *                  specify the desired coordinates of the                      *
   662 *                  arc endpoint.                                               *
   663 *                                                                              *
   664 *           i, j - represent the absolute values of the offset                 *
   665 *                  values of the arc centrepoint from the toolbit's            *
   666 *                  present position.                                           *
   667 *                                                                              *
   668 *           rate - specifies the desired feedrate along the contour.           *
   669 *                                                                              *
   670 *           code - represents the G code (G02 or G03) which                    *
   671 *                  specifies whether the direction of motion is                *
   672 *                  clockwise or counterclockwise.                              *
   673 *                                                                              *
   674 *                                                                              *
   675 *  LOCAL VARIABLES:                                                            *
   676 *                                                                              *
   677 *            vel - present feedrate.                                           *
   678 *                                                                              *
   679 *         vx, vy - component velocities of vel.  These velocities              *
   680 *                  change in a sinusoidal manner as the toolbit                *
   681 *                  travels around the desired arc.                             *
   682 *                                                                              *
   683 *          radi1 - distance from the arc centrepoint to the present            *
   684 *                  toolbit position.                                           *
   685 *                                                                              *
   686 *          radi2 - distance from the arc centrpoint to the toolbit's           *
   687 *                  desired final position. (radi1 should equal radi2)          *
   688 *                                                                              *
   689 *       thetamod - specifies the angle of the feedvector with respect          *
   690 *                  to the angle of the position vector, thetastart.            *
   691 *                                                                              *
   692 *     thetastart - this angle specifies the angle of the position              *
   693 *                  vector at the start of each sample period.                  *
   694 *                                                                              *
   695 *         thetab - this angle specifies the initial angle of the               *
   696 *                  circular arc segment.                                       *
   697 *                                                                              *
   698 *         thetaf - specifies the final angle of the circular arc               *
   699 *                  segment.                                                    *
   700 *                                                                              *
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    701 *             k - local counter.                                    *
    702 *                                                                   *
    703 *   top, bottom  - numerator and denominator for ARCTAN calculations. *
    704 *                                                                   *
    705 *********************************************************************
    706       subroutine circular(pt2i, pt2j, i, j, rate, code)
    707       integer*4 pt1i, pt1j, pt2i, pt2j, pt3i, pt3j, i, j
    708       real rate
    709       integer*2 code
    710
    711       integer*4 px, py
    712       real rx, ry
    713       common /pos/ px, rx, py, ry
    714       real vx, vy
    715
    716       real radi1, radi2, thetastart, thetamod, pi
    717       real acl, decel, vel
    718       real area1, area2, area3, delta
    719       integer*4 pt1, pt2, pt3, k
    720       real thetab, thetaf, fudge
    721       real top, bottom
    722 c-----------------------------------------------------------------------
    723 c Establish the initial parameters:
    724
    725       pi = real(3.1415926535)
    726       acl = 2.00
    727       decel = 1.00
    728       pt1i = int4(rx)
    729       pt1j = int4(ry)
    730       pt3i = pt1i + i
    731       pt3j = pt1j + j
    732
    733 c radi1 and radi2 should be equal if the I and J of the part program are
    734 c OK.
    735
    736       radi1 = sqrt(real(pt1i-pt3i)**2+real(pt1j-pt3j)**2)
    737       radi2 = sqrt(real(pt2i-pt3i)**2+real(pt2j-pt3j)**2)
    738
    739       if(abs(radi1 - radi2) .ge. 5) then
    740         pt3i = pt1i - i
    741         radi1 = sqrt(real(pt1i-pt3i)**2+real(pt1j-pt3j)**2)
    742         radi2 = sqrt(real(pt2i-pt3i)**2+real(pt2j-pt3j)**2)
    743         if(abs(radi1 - radi2) .ge. 5) then
    744           pt3j = pt1j - j
    745           radi1 = sqrt(real(pt1i-pt3i)**2+real(pt1j-pt3j)**2)
    746           radi2 = sqrt(real(pt2i-pt3i)**2+real(pt2j-pt3j)**2)
    747             if(abs(radi1 - radi2) .ge. 5) then
    748                 pt3i = pt1i + i
    749                 radi1 = sqrt(real(pt1i-pt3i)**2+real(pt1j-pt3j)**2)
    750                 radi2 = sqrt(real(pt2i-pt3i)**2+real(pt2j-pt3j)**2)
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    751                  if(abs(radi1 - radi2) .ge. 5) then
    752                     call video(1,10,10)
    753                     write (*,'(1x,a\)') 'Invalid I and J parameters '
    754                     goto 1
    755                  endif
    756               endif
    757            endif
    758         endif
    759
    760         if (code .eq. 2) then
    761              thetamod = -pi/2
    762         else
    763              thetamod = +pi/2
    764         endif
    765
    766         top = abs(real(pt2j - pt3j))
    767         bottom = abs(real(pt2i - pt3i))
    768
    769         if (bottom .le. 0.001) then
    770              thetaf = pi/2
    771         else
    772              thetaf = atan(top/bottom)
    773         endif
    774
    775         top = abs(real(pt1j - pt3j))
    776         bottom = abs(real(pt1i - pt3i))
    777
    778         if (bottom .le. 0.001) then
    779              thetab = pi/2
    780         else
    781              thetab = atan(top/bottom)
    782         endif
    783
    784         delta = abs((thetaf-thetab) * radi1)
    785         call video(1,10,10)
    786
    787 6       area1 = 0.5 * rate * rate / acl
    788         area3 = 0.5 * rate * rate / decel
    789         area2 = delta - area1 - area3
    790
    791         if (ifix(area2/rate) .le. 1) then
    792              rate = rate * 0.9
    793              goto 6
    794         endif
    795
    796         if(ifix(rate) .eq. 40) fudge = 45.48
    797         if(ifix(rate) .eq. 39) fudge = 44.23
    798         if(ifix(rate) .eq. 38) fudge = 44.65
    799         if(ifix(rate) .eq. 37) fudge = 43.75
    800         if(ifix(rate) .eq. 36) fudge = 45.64
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    801       if(ifix(rate) .eq. 35) fudge = 43.66
    802       if(ifix(rate) .eq. 34) fudge = 44.14
    803       if(ifix(rate) .eq. 33) fudge = 44.32
    804       if(ifix(rate) .eq. 32) fudge = 44.38
    805       if(ifix(rate) .eq. 31) fudge = 43.84
    806       if(ifix(rate) .eq. 30) fudge = 45.01
    807       if(ifix(rate) .eq. 29) fudge = 43.09
    808       if(ifix(rate) .eq. 28) fudge = 44.24
    809       if(ifix(rate) .eq. 27) fudge = 42.92
    810       if(ifix(rate) .eq. 26) fudge = 43.75
    811       if(ifix(rate) .eq. 25) fudge = 42.51
    812       if(ifix(rate) .eq. 24) fudge = 43.33
    813       if(ifix(rate) .eq. 23) fudge = 41.62
    814       if(ifix(rate) .eq. 22) fudge = 41.99
    815       if(ifix(rate) .eq. 21) fudge = 41.24
    816       if(ifix(rate) .eq. 20) fudge = 40.67
    817       if(ifix(rate) .eq. 19) fudge = 38.15
    818       if(ifix(rate) .eq. 18) fudge = 38.98
    819       if(ifix(rate) .eq. 17) fudge = 37.58
    820       if(ifix(rate) .eq. 16) fudge = 35.82
    821       if(ifix(rate) .eq. 15) fudge = 33.87
    822       if(ifix(rate) .eq. 14) fudge = 33.50
    823       if(ifix(rate) .eq. 13) fudge = 31.23
    824       if(ifix(rate) .eq. 12) fudge = 29.32
    825       if(ifix(rate) .eq. 11) fudge = 26.47
    826       if(ifix(rate) .eq. 10) fudge = 23.80
    827       if(ifix(rate) .eq. 09) fudge = 22.21
    828       if(ifix(rate) .eq. 08) fudge = 18.36
    829       if(ifix(rate) .eq. 07) fudge = 16.52
    830       if(ifix(rate) .eq. 06) fudge = 12.69
    831       if(ifix(rate) .eq. 05) fudge = 10.52
    832       if(ifix(rate) .eq. 04) fudge = 07.12
    833       if(ifix(rate) .eq. 03) fudge = 05.22
    834       if(ifix(rate) .eq. 02) fudge = 01.91
    835       if(ifix(rate) .eq. 01) fudge = 01.09
    836
    837
    838
    839       pt1 = int4(rate/acl)
    840       pt2 = int4(area2/rate)
    841
    842 c
    843 c-------------------------------------------------------------------------
    844
    845
    846       vx = 0.0
    847       vy = 0.0
    848       vel = 0.0
    849       call moton
    850
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    851       do 2, k = 1, pt1
1   852            top = real(ry - pt3j)
1   853            bottom = real(rx - pt3i)
1   854
1   855            thetastart = atan2(top,bottom)
1   856
1   857            rx = real(pt3i) + cos(thetastart)*radi1
1   858            ry = real(pt3j) + sin(thetastart)*radi1
1   859
1   860            vel = vel + acl
1   861            vx = vel * cos(thetastart + thetamod)
1   862            vy = vel * sin(thetastart + thetamod)
1   863
1   864            call wait(vx, vy, 0)
1   865 2     continue
    866
    867       do 3, k = 1, pt2
1   868            top = real(ry - pt3j)
1   869            bottom = real(rx - pt3i)
1   870
1   871            thetastart = atan2(top,bottom)
1   872
1   873            rx = real(pt3i) + cos(thetastart)*radi1
1   874            ry = real(pt3j) + sin(thetastart)*radi1
1   875
1   876            vel = rate
1   877            vx = vel * cos(thetastart + thetamod)
1   878            vy = vel * sin(thetastart + thetamod)
1   879
1   880            call wait(vx, vy, 0)
1   881 3     continue
    882
    883
    884       top = abs(real(ry - pt3j))
    885       bottom = abs(real(rx - pt3i))
    886
    887       if (bottom .le. 0.001) then
    888            thetab = pi/2
    889       else
    890            thetab = atan(top/bottom)
    891       endif
    892
    893       area3 = abs((thetaf-thetab) * radi1) + rate/2.0 - fudge
    894 c Have calculated the new area3.
    895       decel = 0.5*rate*rate/area3
    896       pt3 = int4(rate/decel)
    897
    898       do 4, k = 1, pt3
1   899            top = real(ry - pt3j)
1   900            bottom = real(rx - pt3i)
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
1   901
1   902           thetastart = atan2(top,bottom)
1   903           rx = real(pt3i) + cos(thetastart)*radii
1   904           ry = real(pt3j) + sin(thetastart)*radii
1   905
1   906           vel = vel - decel
1   907           vx = vel * cos(thetastart + thetamod)
1   908           vy = vel * sin(thetastart + thetamod)
1   909
1   910           call wait(vx, vy, 0)
1   911 4     continue
    912
    913       vx = 0.0
    914       vy = 0.0
    915       rx = real(pt2i)
    916       ry = real(pt2j)
    917       call tweak
    918       call motoff
    919 1     return
    920       end
```

Name	Type	Offset	P	Class
ABS				INTRINSIC
ACL	REAL	1000		
AREA1	REAL	1056		
AREA2	REAL	1064		
AREA3	REAL	1060		
ATAN				INTRINSIC
ATAN2				INTRINSIC
BOTTOM	REAL	1040		
CODE	INTEGER*2	20	*	
COS				INTRINSIC
DECEL	REAL	1004		
DELTA	REAL	1052		
FUDGE	REAL	1068		
I	INTEGER*4	8	*	
IFIX				INTRINSIC
INT4				INTRINSIC
J	INTEGER*4	12	*	
K	INTEGER*4	1092		
PI	REAL	996		
PT1	INTEGER*4	1072		
PT1I	INTEGER*4	1008		
PT1J	INTEGER*4	1012		
PT2	INTEGER*4	1076		
PT2I	INTEGER*4	0	*	
PT2J	INTEGER*4	4	*	
PT3	INTEGER*4	1108		
PT3I	INTEGER*4	1016		

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

PT3J	INTEGER*4	1020	
PX	INTEGER*4	0	/POS /
PY	INTEGER*4	8	/POS /
RADI1	REAL	1024	
RADI2	REAL	1028	
RATE	REAL	16	*
REAL			INTRINSIC
RX	REAL	4	/POS /
RY	REAL	12	/POS /
SIN			INTRINSIC
SQRT			INTRINSIC
THETAB	REAL	1048	
THETAF	REAL	1044	
THETAM	REAL	1032	
THETAS	REAL	1100	
TOP	REAL	1036	
VEL	REAL	1088	
VX	REAL	1080	
VY	REAL	1084	

```
921
922 $subtitle:'MODULE:  SUBROUTINE WAIT'
923 $page
```

```
    924 ********************************************************************************
    925 *                                                                              *
    926 *  Subroutine WAIT acts as the high level software interface between           *
    927 *  the interface hardware (DAC and ENCODER counter) and the FORTRAN            *
    928 *  interpolation subroutines.                                                  *
    929 *                                                                              *
    930 *  The loop is normally in mode 0 which is the velocity loop.  Mode 1          *
    931 *  restablishes the loop as a position loop whereby the loop becomes           *
    932 *  critically damped for position changes.  This mode is used by the           *
    933 *  subroutine TWEAK to ensure that the position of the toolbit is              *
    934 *  within 0.0005 inches of the desired segment endpoint.                       *
    935 *                                                                              *
    936 *  NOTES:                                                                      *
    937 *        The seemingly infinite loop driven by variable testflag               *
    938 *        will not cause the program to 'hang'.  Testflag is an                 *
    939 *        interrupt driven variable which provides the timing for               *
    940 *        the control loop.                                                     *
    941 *                                                                              *
    942 *  COMMON BLOCKS ACCESSIBLE:                                                   *
    943 *                                    /pos/                                     *
    944 *                                    /dyn/                                     *
    945 *                                    /assemb/                                  *
    946 *                                                                              *
    947 *                                                                              *
    948 *  LOCAL VARIABLES:                                                            *
    949 *                  mode - determines if loop is set as a velocity              *
    950 *                         or a position loop.                                  *
    951 *                                                                              *
    952 *                  vx - requested velocity in X direction (BLUs/int).          *
    953 *                                                                              *
    954 *                  vy - requested velocity in Y direction (BLUs/int).          *
    955 *                                                                              *
    956 *                  erx - actual X axis position error (BLUs).                  *
    957 *                                                                              *
    958 *                  ery - actual Y axis position error (BLUs).                  *
    959 *                                                                              *
    960 *                  qx - X axis compensated error.                              *
    961 *                                                                              *
    962 *                  qy - Y axis compensated error.                              *
    963 *                                                                              *
    964 ********************************************************************************
    965       subroutine wait(vx, vy, mode)
    966
    967       integer*4 px, py
    968       real rx, ry
    969       common /pos/ px, rx, py, ry
    970
    971       integer*2 intflag, flag0, flag1, err0, err1, count0, count1
    972       common /assemb/ intflag, flag0, flag1, err0, err1, count0, count1
    973
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
    974       real ex1, qx1, ex2, qx2, ex3, qx3
    975       real ey1, qy1, ey2, qy2, ey3, qy3
    976       integer*2 ke
    977       common /dyn/ ke, ex1, qx1, ex2, qx2, ex3, qx3,
    978      &ey1, qy1, ey2, qy2, ey3, qy3
    979
    980
    981       integer*2  mode
    982       real vx, vy, erx, ery, qx, qy
    983
    984       if (intflag .ne. 0) then
    985           goto 5
    986       endif
    987
    988 c Assuming an overrun hasn't occurred, wait for interrupt.
    989
    990       call showon
    991 1     if (intflag .eq. 0) then
    992           goto 1
    993       endif
    994       if (intflag .eq. 2) then
    995 c A key has been pressed.
    996           call motoff
    997 c         call EMERGENCY STOP    [FUTURE SUBROUTINE]
    998           return
    999       endif
   1000
   1001 5     intflag = 0
   1002
   1003 c Interrupt made - send out the new velocity reference.
   1004       call showoff
   1005
   1006
   1007       if (flag0 .ne. 0) then
   1008           px = px + int4(flag0)
   1009           px = nint(px/real(ke))*ke
   1010       endif
   1011
   1012       px = px + int4(count0)-int4(flag0)
   1013       rx = rx + vx
   1014
   1015       if (flag1 .ne. 0) then
   1016           py = py + int4(flag1)
   1017           py = nint(py/real(ke))*ke
   1018       endif
   1019
   1020       py = py + int4(count1)-int4(flag1)
   1021       ry = ry + vy
   1022
   1023       flag0 = 0
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
   1024       flag1 = 0
   1025
   1026       erx = rx - real(px)
   1027       ery = ry - real(py)
   1028
   1029 c     If the mode flag = 0, the loop is set as a velocity loop.
   1030 c     If the mode flag = 1, then the loop is set as a position
   1031 c     control loop.
   1032
   1033 c     The following code represents a Direct Digital Realization of
   1034 c     a pole-zero transformed lead-lag filter.
   1035
   1036       if (mode .eq. 0) then
   1037            qx = .413351*erx - .677849*ex1 + .273499*ex2 + 1.085419
   1038      &     *qx1 - .094420*qx2
   1039            ex2=ex1
   1040            ex1=erx
   1041            qx2=qx1
   1042            qx1 = qx
   1043
   1044            qy = .413351*ery - .677849*ey1 + .273499*ey2 + 1.085419
   1045      &     *qy1 - .094420*qy2
   1046            ey2=ey1
   1047            ey1=ery
   1048            qy2=qy1
   1049            qy1 = qy
   1050       else
   1051            qx = erx-0.990*ex3+qx3
   1052            ex3 = erx
   1053            qx3 = qx
   1054
   1055            qy = ery-0.990*ey3+qy3
   1056            ey3 = ery
   1057            qy3 = qy
   1058       endif
   1059
   1060       if (mode .eq. 0) then
   1061            qx = qx * 5.081
   1062            qy = qy * 5.081
   1063       else
   1064            qx = qx * 0.2174
   1065            qy = qy * 0.2174
   1066       endif
   1067
   1068       err0=int2(qx)
   1069       err1=int2(qy)
   1070
   1071 c   Must limit the error to within the capability of the DAC.
   1072       if(err0 .gt. 127) then
   1073            err0 = 127
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
   1074       elseif(err0 .lt. -127) then
   1075           err0 = -127
   1076       endif
   1077
   1078       if(err1 .gt. 127) then
   1079           err1 = 127
   1080       elseif(err1 .lt. -127) then
   1081           err1 = -127
   1082       endif
   1083       return
   1084       end
```

Name	Type	Offset	P	Class
COUNT0	INTEGER*2	10		/ASSEMB/
COUNT1	INTEGER*2	12		/ASSEMB/
ERR0	INTEGER*2	6		/ASSEMB/
ERR1	INTEGER*2	8		/ASSEMB/
ERX	REAL	1116		
ERY	REAL	1120		
EX1	REAL	2		/DYN /
EX2	REAL	10		/DYN /
EX3	REAL	18		/DYN /
EY1	REAL	26		/DYN /
EY2	REAL	34		/DYN /
EY3	REAL	42		/DYN /
FLAG0	INTEGER*2	2		/ASSEMB/
FLAG1	INTEGER*2	4		/ASSEMB/
INT2				INTRINSIC
INT4				INTRINSIC
INTFLA	INTEGER*2	0		/ASSEMB/
KE	INTEGER*2	0		/DYN /
MODE	INTEGER*2	8	*	
NINT				INTRINSIC
PX	INTEGER*4	0		/POS /
PY	INTEGER*4	8		/POS /
QX	REAL	1124		
QX1	REAL	6		/DYN /
QX2	REAL	14		/DYN /
QX3	REAL	22		/DYN /
QY	REAL	1128		
QY1	REAL	30		/DYN /
QY2	REAL	38		/DYN /
QY3	REAL	46		/DYN /
REAL				INTRINSIC
RX	REAL	4		/POS /
RY	REAL	12		/POS /
VX	REAL	0	*	
VY	REAL	4	*	

```
   1085
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
 1086
 1087
 1088 $subtitle:'MODULE:  SUBROUTINE VIDEO'
 1089 $page
```

```
1090 ********************************************************************************
1091 *                                                                              *
1092 *  Subroutine VIDEO performs rudimentary video functions for the               *
1093 *  CNCMILL program. The available functions will clear the screen              *
1094 *  or position the cursor.                                                     *
1095 *                                                                              *
1096 *                                                                              *
1097 *  NOTES:                                                                      *
1098 *  This subroutine makes use of DOS's ANSI terminal emulation                  *
1099 *  video driver.  This driver, ANSI.SYS, is a DOS file which                   *
1100 *  must be loaded on the root boot directory of the system                     *
1101 *  In addition, the driver must be declared in the CONFIG.SYS                  *
1102 *  parameter file, also found in the root boot directory.                      *
1103 *                                                                              *
1104 *  COMMON BLOCKS ACCESSIBLE:                                                   *
1105 *                                  NIL                                         *
1106 *                                                                              *
1107 *  LOCAL VARIABLES:                                                            *
1108 *                  flag - determines if screen should be cleared,              *
1109 *                         a 0 clears the screen.                               *
1110 *                                                                              *
1111 *                  row - determines the screen row to be positioned            *
1112 *                        to (1 - 24).                                          *
1113 *                                                                              *
1114 *                  column - determines the screen column to be                 *
1115 *                           positioned to (1 - 80).                            *
1116 *                                                                              *
1117 ********************************************************************************
1118
1119       subroutine video(flag, row, column)
1120       integer*2 flag, row, column
1121            if(flag .eq. 0) then
1122                 write(*,'(1x,a,a\)') char(27), '[2J'
1123            endif
1124            if(row .lt. 10 .and. column .lt. 10) then
1125                 write(*,'(1x,a,a,i1,a,i1,a\)') char(27),
1126      &          '[',row,';',column,'H'
1127            elseif(row .ge. 10 .and. column .lt. 10) then
1128                 write(*,'(1x,a,a,i2,a,i1,a\)') char(27),
1129      &          '[',row,';',column,'H'
1130            elseif(row .lt. 10 .and. column .ge. 10) then
1131                 write(*,'(1x,a,a,i1,a,i2,a\)') char(27),
1132      &          '[',row,';',column,'H'
1133            elseif(row .ge. 10 .and. column .ge. 10) then
1134                 write(*,'(1x,a,a,i2,a,i2,a\)') char(27),
1135      &          '[',row,';',column,'H'
1136            endif
1137       return
1138       end
```

```
Name    Type        Offset P Class

CHAR                       INTRINSIC
COLUMN  INTEGER*2       8 *
FLAG    INTEGER*2       0 *
ROW     INTEGER*2       4 *


  1139 $subtitle:'MODULE:  SUBROUTINE FILEREAD'
  1140 $page
```

```
D Line# 1    7                          Microsoft FORTRAN77 V3.30 March 1985
  1141 ********************************************************************************
  1142 *                                                                              *
  1143 *  Subroutine FILEREAD loads in a part program for use by the                  *
  1144 *  DECODE subroutine.                                                          *
  1145 *                                                                              *
  1146 *  NOTES:                                                                      *
  1147 *  Due to FORTRAN's limited character string manipulation capability           *
  1148 *  this subroutine makes use of core-to-core I/O techniques to                 *
  1149 *  reduce the loaded ASCII character strings to numerical arrays.              *
  1150 *                                                                              *
  1151 *  This subroutine requires that every word of a block line must               *
  1152 *  appear in a specific format.                                                *
  1153 *                                                                              *
  1154 *                                                                              *
  1155 *  COMMON BLOCKS ACCESSIBLE:                                                   *
  1156 *                                    /pprog/                                   *
  1157 *                                                                              *
  1158 *  LOCAL VARIABLES:                                                            *
  1159 *                  chrpos - character position within the 80                   *
  1160 *                           element BLOCK character string.                    *
  1161 *                                                                              *
  1162 *                  filename - desired ASCII part program file.                 *
  1163 *                                                                              *
  1164 *                  block - this buffer holds a single line (block)             *
  1165 *                          of the part program.                                *
  1166 *                                                                              *
  1167 *                  buff1 - core-to-core I/O temporary buffer.                  *
  1168 *                                                                              *
  1169 ********************************************************************************
  1170       subroutine fileread
  1171
  1172       integer*2 tbnum, n(250), g(250), i(250), j(250), k(250), f(250),
  1173      &s(250), t(250), m(250)
  1174       integer*4 x(250), y(250), z(250)
  1175       common /pprog/ tbnum, n, g, x, y, z, i, j, k, f, s, t, m
  1176
  1177       integer*2 chrpos
  1178       character fname*20, block*80, buff1*10
  1179
  1180       call video(1,20,1)
  1181       write (*,'(1x,a\)') '[ENTER] the filename: '
  1182       read (*,'(a\)') fname
  1183
  1184       open (1, file = fname, status = 'old', access = 'sequential',
  1185      &form = 'formatted')
  1186
  1187       tbnum = 1
  1188
  1189 1     continue
  1190
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
   1191       read (1, '(a80)', end=10, err=11) block
   1192
   1193       do 2, chrpos = 1, 78
1  1194
1  1195       if (block(chrpos:chrpos) .eq. 'N') then
1  1196           write (buff1,'(a3)') block(chrpos+1:chrpos+4)
1  1197           read (buff1,'(bn,i3)') n(tbnum)
1  1198       elseif (block(chrpos:chrpos) .eq. 'G') then
1  1199           write (buff1,'(a2)') block(chrpos+1:chrpos+3)
1  1200           read (buff1,'(bn,i2)') g(tbnum)
1  1201       elseif (block(chrpos:chrpos) .eq. 'X') then
1  1202           write (buff1,'(a6)') block(chrpos+1:chrpos+6)
1  1203           read (buff1,'(bn,i6)') x(tbnum)
1  1204       elseif (block(chrpos:chrpos) .eq. 'Y') then
1  1205           write (buff1,'(a6)') block(chrpos+1:chrpos+6)
1  1206           read (buff1,'(bn,i6)') y(tbnum)
1  1207       elseif (block(chrpos:chrpos) .eq. 'Z') then
1  1208           write (buff1,'(a6)') block(chrpos+1:chrpos+6)
1  1209           read (buff1,'(bn,i6)') z(tbnum)
1  1210       elseif (block(chrpos:chrpos) .eq. 'I') then
1  1211           write (buff1,'(a6)') block(chrpos+1:chrpos+6)
1  1212           read (buff1,'(bn,i6)') i(tbnum)
1  1213       elseif (block(chrpos:chrpos) .eq. 'J') then
1  1214           write (buff1,'(a6)') block(chrpos+1:chrpos+6)
1  1215           read (buff1,'(bn,i6)') j(tbnum)
1  1216       elseif (block(chrpos:chrpos) .eq. 'K') then
1  1217           write (buff1,'(a6)') block(chrpos+1:chrpos+6)
1  1218           read (buff1,'(bn,i6)') k(tbnum)
1  1219       elseif (block(chrpos:chrpos) .eq. 'F') then
1  1220           write (buff1,'(a4)') block(chrpos+1:chrpos+4)
1  1221           read (buff1,'(bn,i4)') f(tbnum)
1  1222       elseif (block(chrpos:chrpos) .eq. 'S') then
1  1223           write (buff1,'(a3)') block(chrpos+1:chrpos+3)
1  1224           read (buff1,'(bn,i3)') s(tbnum)
1  1225       elseif (block(chrpos:chrpos) .eq. 'T') then
1  1226           write (buff1,'(a2)') block(chrpos+1:chrpos+2)
1  1227           read (buff1,'(bn,i2)') t(tbnum)
1  1228       elseif (block(chrpos:chrpos) .eq. 'M') then
1  1229           write (buff1,'(a2)') block(chrpos+1:chrpos+2)
1  1230           read (buff1,'(bn,i2)') m(tbnum)
1  1231       endif
1  1232 2     continue
   1233
   1234       tbnum = tbnum + 1
   1235       n(tbnum) = n(tbnum-1)
   1236       g(tbnum) = g(tbnum-1)
   1237       x(tbnum) = x(tbnum-1)
   1238       y(tbnum) = y(tbnum-1)
   1239       z(tbnum) = z(tbnum-1)
   1240       i(tbnum) = i(tbnum-1)
```

```
D Line# 1     7                          Microsoft FORTRAN77 V3.30 March 1985
   1241       j(tbnum) = j(tbnum-1)
   1242       k(tbnum) = k(tbnum-1)
   1243       f(tbnum) = f(tbnum-1)
   1244       s(tbnum) = s(tbnum-1)
   1245       t(tbnum) = t(tbnum-1)
   1246       m(tbnum) = m(tbnum-1)
   1247       goto 1
   1248
   1249 10    continue
   1250       tbnum = tbnum - 1
   1251       close (1, status='keep')
   1252       write (*,'(1x,a)') 'File successfully read '
   1253
   1254       return
   1255
   1256 11    stop 'File reading error in unit FILEREAD'
   1257       end
```

Name	Type	Offset	P	Class
BLOCK	CHAR*80	1152		
BUFF1	CHAR*10	1234		
CHRPOS	INTEGER*2	1232		
F	INTEGER*2	5502		/PPROG /
FNAME	CHAR*20	1132		
G	INTEGER*2	502		/PPROG /
I	INTEGER*2	4002		/PPROG /
J	INTEGER*2	4502		/PPROG /
K	INTEGER*2	5002		/PPROG /
M	INTEGER*2	7002		/PPROG /
N	INTEGER*2	2		/PPROG /
S	INTEGER*2	6002		/PPROG /
T	INTEGER*2	6502		/PPROG /
TBNUM	INTEGER*2	0		/PPROG /
X	INTEGER*4	1002		/PPROG /
Y	INTEGER*4	2002		/PPROG /
Z	INTEGER*4	3002		/PPROG /

```
   1258
   1259
   1260
   1261 $subtitle:'MODULE:  SUBROUTINE TWEAK'
   1262 $page
```

D Line# 1 7 Microsoft FORTRAN77 V3.30 March 1985

```
1263 ********************************************************************************
1264 *                                                                              *
1265 *  Subroutine TWEAK ensures that the error between the actual                  *
1266 *  and reference positions at the end of a move are within a                   *
1267 *  specified tolerance.                                                        *
1268 *                                                                              *
1269 *  NOTES:                                                                      *
1270 *  This subroutine makes use of the WAIT subroutine in mode 1.                 *
1271 *  Mode 1 adds an extra integrator to the control loop to ensure               *
1272 *  a theoretical zero steady state step position error.                        *
1273 *                                                                              *
1274 *  COMMON BLOCKS ACCESSIBLE:                                                   *
1275 *                                    /pos/                                     *
1276 *                                                                              *
1277 *  LOCAL VARIABLES:                                                            *
1278 *                   okcnt - number of consecutive interrupt                    *
1279 *                           periods within the specified tolerance.            *
1280 *                                                                              *
1281 *                   spectol - specified tolerance in BLUs.                     *
1282 *                                                                              *
1283 *                   specper - specified period in interrupt periods            *
1284 *                             that the system must hold the axes               *
1285 *                             within the spectol to ensure that the            *
1286 *                             axes have stopped precisely.                     *
1287 *                                                                              *
1288 ********************************************************************************
1289       subroutine tweak
1290       integer*4 px, py
1291       real rx, ry
1292       common /pos/ px, rx, py, ry
1293       integer*2 okcnt, spectol, specper
1294
1295       spectol = 5
1296       specper = 2
1297       okcnt = 0
1298
1299 62    if ((abs(px-int4(rx)) .gt.  spectol).or.(abs(py-int4(ry)) .gt.
1300      &spectol)) then
1301            okcnt = 0
1302       endif
1303
1304       call wait(0.0, 0.0, 1)
1305       okcnt = okcnt+1
1306       if (okcnt .le. specper) then
1307            goto 62
1308       endif
1309       return
1310       end
```

Name	Type	Offset	P	Class
ABS				INTRINSIC
INT4				INTRINSIC
OKCNT	INTEGER*2	1272		
PX	INTEGER*4	0		/POS /
PY	INTEGER*4	8		/POS /
RX	REAL	4		/POS /
RY	REAL	12		/POS /
SPECPE	INTEGER*2	1270		
SPECTO	INTEGER*2	1268		

Name	Type	Size	Class
ASSEMB		14	COMMON
CIRCUL			SUBROUTINE
CLSUP			SUBROUTINE
CNCMIL			PROGRAM
DECODE			SUBROUTINE
DYN		50	COMMON
FILERE			SUBROUTINE
GETKY			SUBROUTINE
LINEAR			SUBROUTINE
MOTOFF			SUBROUTINE
MOTON			SUBROUTINE
POS		16	COMMON
PPROG		7502	COMMON
SETUP			SUBROUTINE
SHOWOF			SUBROUTINE
SHOWON			SUBROUTINE
TWEAK			SUBROUTINE
VIDEO			SUBROUTINE
WAIT			SUBROUTINE

Pass One No Errors Detected
1310 Source Lines

```
                page 54, 132
                TITLE  FORTRAN subroutine - interrupt 0 (hard) soft 8.
                ;
```

```
page
comment ~

*************************************************************************
*                                                                       *
*                                                                       *
*                                                                       *
*           PROGRAM:   BASE6                                            *
*            AUTHOR:   C. G. Gadsby                                     *
*                                                                       *
*                                                                       *
*      These assembly language subroutines are used                     *
*      by the FORTRAN mainline program called M6.                       *
*      A description of the routines follows on the                     *
*      next few pages.                                                  *
*                                                                       *
*                                                                       *
*                                                                       *
*                                                                       *
*************************************************************************
~
```

```
page
comment *

        This code is the second generation code used
        to drive the TECMAR BASE BOARD and the
        custom encoder interface.

        This code must perform several functions:

        SETUP:
        1)  The interrupt servicing routine must be installed,
            the system clock must be reprogrammed to interrupt
            at the desired frequency and the TECMAR BASE BOARD
            must be initialized.  [At a future date, these
            initialization tasks will also include the clearing
            of all the registers and the clearing of a failsafe
            mechanism.]

        CLSUP:
        2)  Provide closing services, ie. after the program has
            run, a subroutine must restore the operating system
            DOS to its original state.  In this instance the
            system clock must be reprogrammed to its original
            speed and the original BIOS interrupt vector must
            be reinstalled.

        The interrupt service routine ISR0 is the heart of the
        servo system.  It must provide the low level servicing
        of the velocity reference for the servo system.  The
        system consists of two distinct sections, the velocity
        reference section and the encoder feedback section.

        i)  The velocity reference section must provide an analog
        output to the velocity reference of the servo.  The error
        number, passed down from the FORTRAN interpolation routine
        is converted to an analog signal by an 8 bit A-D converter.
        This 8 bit A-D converter is addressed through the TECMAR I/O
        board.

        ii) The encoder count must be transferred back to the FORTRAN
        mainline so that the position counter can be updated.  This
        transfer is fairly sophisticated in that a transparent latch
        must be triggered and the results read in via the BASEBOARD.
        A special flag exists which indicates that the count to be read
        is an INDEX count.  An index count indicates that the shaft's
        ABSOLUTE position should be an INTEGER multiple of the
        encoder's number of counts per revolution.

        The flow of the routine is as follows:
```

a)Turn the STROBE line on and off.
b)Read the counter in through port B
c)Check the FLAG line to see if it is an INDEX count.
Modify the FLAG returned to FORTRAN accordingly.
[The correction of the absolute position counter
will be done at the FORTRAN level.]
d)Turn the CLEAR line on and off if FLAG was set.
This code resets the index trapping circuitry.
e)Send out through port A the appropriate DAC error
signal.

```
                        ;
= 0040                  timer           equ     40h         ;IBM clock port.
                        ;
= 0210                  base            equ     0210h       ;Base address of baseboard.
=                       bbctrl0         equ     base
= 0211                  bb0c            equ     base + 1
= 0212                  bb0b            equ     base + 2
= 0213                  bb0a            equ     base + 3
= 0214                  bbctrl1         equ     base + 4
= 0215                  bb1c            equ     base + 5
= 0216                  bb1b            equ     base + 6
= 0217                  bb1a            equ     base + 7
= 0218                  bbctrl2         equ     base + 8
= 0219                  bb2c            equ     base + 9
= 021A                  bb2b            equ     base + 10
= 021B                  bb2a            equ     base + 11
= 021C                  bbctrl3         equ     base + 12
= 021D                  bb3c            equ     base + 13
= 021E                  bb3b            equ     base + 14
= 021F                  bb3a            equ     base + 15

                        ;The following words, when sent to the control ports
                        ;will set or reset the appropriate output bits of port C.
                        ;Hence, bbc5h will set bit C5 high, and bbc5l will set
                        ;bit C5 low.

= 0009                  bbc4h           equ     00001001b
= 0008                  bbc4l           equ     00001000b

= 000B                  bbc5h           equ     00001011b
= 000A                  bbc5l           equ     00001010b

= 000D                  bbc6h           equ     00001101b
= 000C                  bbc6l           equ     00001100b

= 000F                  bbc7h           equ     00001111b
= 000E                  bbc7l           equ     00001110b
```

```
= 0083                  ctrl3           equ     10000011b
```

```
                        page
                        comment ~
                                ***********************************************
                               * The data segment consists entirely of FORTRAN *
                               * data variable addresses.  These addresses are *
                               * stored by the SETUP subroutine for use by the *
                               * other subroutines, notably ISR0.              *
                                ***********************************************
                               ~

0000                    data    segment public 'data'
                                even
0000  0000                     intflag_addr dw 0
0002  0000                     intflag_seg  dw 0
0004  0000                     flag0_addr   dw 0
0006  0000                     flag0_seg    dw 0
0008  0000                     flag1_addr   dw 0
000A  0000                     flag1_seg    dw 0
000C  0000                     da0_addr     dw 0
000E  0000                     da0_seg      dw 0
0010  0000                     da1_addr     dw 0
0012  0000                     da1_seg      dw 0
0014  0000                     count0_addr dw 0
0016  0000                     count0_seg  dw 0
0018  0000                     count1_addr dw 0
001A  0000                     count1_seg  dw 0
001C                    data    ends

                        dgroup  group data
```

```
                                        page
                                        comment ~
                                              *****************************************************************
                                              *The subroutine code is entirely contained in the following    *
                                              *segment.  The code consists of four subroutines:              *
                                              *       1) SETUP                                               *
                                              *       2) CLSUP                                               *
                                              *       3) MOTON                                               *
                                              *       4) MOTOFF                                              *
                                              *       5) ISR0                                                *
                                              *       6) ISR1                                                *
                                              *                                                              *
                                              *SETUP establishes the environment for the other subroutines. *
                                              *CLSUP terminates the environment for the other subroutines.  *
                                              *MOTON ensures that ISR0 is operating and connects the motor  *
                                              *      amplifier speed reference terminals to the DAC outputs.*
                                              *MOTOFF installs ISR1 and disconnects the motor amplifier     *
                                              *       speed reference terminals from the DAC and connects   *
                                              *       together the terminals.  This stops the motor from    *
                                              *       drifting in idle states.                              *
                                              *ISR0   Main interrupt procedure.  This procedure outputs the *
                                              *       error words to the DACS via the TECMAR BASEBOARD and  *
                                              *       reads the external counter and passes the value back  *
                                              *       to FORTRAN to update the position accumulators.       *
                                              *****************************************************************
                                                    ~

0000                                    code    segment 'code'
                                                assume cs:code, ds:dgroup, ss:dgroup
                                        public  setup          ;Must declare subroutines public.
                                        public  clsup
                                        public  motoff
                                        public  moton
```

```
                        page
                        comment ~
                                ***********************************************************
                               * The SETUP subroutine is used to intall the interrupt *
                               * handling subroutines, and reprogram the timer which  *
                               * interrupts the computer.                             *
                                ***********************************************************
                                ~
0000                    setup   proc far

                                frame struc
0000  0000              savebp  dw
0002  00 00 00 00       saveret dd
0006  00 00 00 00       count1 dd
000A  00 00 00 00       count0 dd
000E  00 00 00 00       da1     dd
0012  00 00 00 00       da0     dd
0016  00 00 00 00       flag1   dd
001A  00 00 00 00       flag0   dd
001E  00 00 00 00       intflag dd
0022                    frame ends

0000  55                        push bp                 ;Save framepointer.
0001  8B EC                     mov bp, sp              ;Get new framepointer.
0003  06                        push es                 ;Save extra - seg.

0004  C4 5E 1E                  les bx, [bp].intflag    ;Get INTFLAG address.
0007  89 1E 0000 R              mov intflag_addr, bx    ;Save INTFLAG offset.
000B  8C 06 0002 R              mov intflag_seg, es     ;Save INTFLAG segment.

000F  C4 5E 1A                  les bx, [bp].flag0      ;Get FLAG0 address.
0012  89 1E 0004 R              mov flag0_addr, bx      ;Save FLAG0 offset.
0016  8C 06 0006 R              mov flag0_seg, es       ;Save FLAG0 segment.

001A  C4 5E 16                  les bx, [bp].flag1      ;Get FLAG1 address.
001D  89 1E 0008 R              mov flag1_addr, bx      ;Save FLAG1 offset.
0021  8C 06 000A R              mov flag1_seg, es       ;Save FLAG1 segment.

0025  C4 5E 12                  les bx, [bp].da0        ;Get DA0 address.
0028  89 1E 000C R              mov da0_addr, bx        ;Save DA0 offset.
002C  8C 06 000E R              mov da0_seg, es         ;Save DA0 segment.

0030  C4 5E 0E                  les bx, [bp].da1        ;Get DA1 address.
0033  89 1E 0010 R              mov da1_addr, bx        ;Save DA1 offset.
0037  8C 06 0012 R              mov da1_seg, es         ;Save DA1 segment.

003B  C4 5E 0A                  les bx, [bp].count0     ;Get count0 address.
003E  89 1E 0014 R              mov count0_addr, bx     ;Save count0 offset.
0042  8C 06 0016 R              mov count0_seg, es      ;Save count0 segment.
```

```
0046  C4 5E 06                    les bx, [bp].count1     ;Get count1 address.
0049  89 1E 0018 R                 mov count1_addr, bx     ;Save count1 offset.
004D  8C 06 001A R                 mov count1_seg, es      ;Save count1 segment.

                          ;-----------------------------------------------------------------
                          ;  Set up the BASE BOARD for the work required:
                          ;
                          ;           MODE 0, CONTROL WORD 3
                          ;
                          ;           A is set for output (latched).
                          ;           B is set for input.
                          ;           C4 - C7 is set for output.
                          ;           C0 - C3 is set for input.
                          ;
                          ;           A is DAC signal.
                          ;           B is ENCODER count.
                          ;           C5 is the strobe.
                          ;           C7 is the clear.
                          ;           C3 is the flag.
                          ;
                          ;           Bit 7 = 1          Mode Setup.
                          ;           Bit 6 = 0          Port A is mode 0.
                          ;           Bit 5 = 0          Port A is mode 0.
                          ;           Bit 4 = 0          Port A is output.
                          ;           Bit 3 = 0          C4 - C7 is output.
                          ;           Bit 2 = 0          Port B = mode 0.
                          ;           Bit 1 = 1          Port B = input.
                          ;           Bit 0 = 1          C0 - C3 = input.

0051  BA 0210                      mov dx, bbctrl0     ;Get the setup port.
0054  B0 83                        mov al, ctrl3       ;Get the setup byte.
0056  EE                           out dx, al          ;Send it out.

0057  BA 0218                      mov dx, bbctrl2     ;Get the setup port.
005A  B0 83                        mov al, ctrl3       ;Get the setup byte.
005C  EE                           out dx, al          ;Send it out.
                          ;
005D  BA 0213                      mov dx, bb0a        ;Get the DAC port.
0060  B0 7F                        mov al, 127         ;Zero the DAC.
0062  EE                           out dx, al

0063  BA 021B                      mov dx, bb2a        ;Get the DAC port.
0066  B0 7F                        mov al, 127         ;Zero the DAC.
0068  EE                           out dx, al
                          ;
                          ; The next few lines turn off the relay and ensure that the servos
                          ; are disconnected from the system.
```

```
0069  BA 0210                        mov dx, bbctrl0
006C  B0 08                         mov al, bbc4l       ;Turn off bit C4.
006E  EE                            out dx, al          ;Turn off relay.

006F  BA 0218                       mov dx, bbctrl2
0072  B0 08                         mov al, bbc4l       ;Turn off bit C4.
0074  EE                            out dx, al          ;Turn off relay.
                              ;
                              ;-------------------------------------------------------------


                              ;-------------------------------------------------------------------
                              ;
                              ;  Set up the 8253-5 Timer to interrupt at my frequency:
                              ;
                              ;           Clock frequency to 8253   f = 1.193182 MHz.
                              ;                                 T = 1/f = 0.838095 microseconds.
                              ;
                              ;           Counter values: Cntval = f/desired f.
                              ;                           0000h (FFFFh) = 18.2 Hz.
                              ;                           04A9h = 1000 Hz.
                              ;                           091Ah =  512 Hz.
                              ;                           1235h =  256 Hz.
                              ;                           246Ah =  128 Hz.
                              ;                           2E9Bh =  100 Hz.
                              ;                           4BD3h =   64 Hz.
                              ;                           5D36h =   50 Hz.
                              ;
                              ;
0075  B0 36                           mov al,00110110b      ;Timer 0, lsb,msb,mode 3 binary.
0077  E6 43                           out timer+3, al

0079  B0 9B                           mov al,9Bh           ;LSB.
007B  E6 40                           out timer,al

007D  90                              nop
007E  90                              nop
007F  90                              nop

0080  B0 2E                           mov al,2Eh           ;MSB.
0082  E6 40                           out timer,al

0084  B9 4E20                         mov cx, 20000
0087                          qwert:
0087  E2 FE                           loop qwert           ;Wait 70 msecx. for chip to accept.

                              ;
                              ;At this point the clock is running at 100 Hz.
                              ;-------------------------------------------------------------------
```

```
                        ;-----------------------------------------------------------------
                        ;Install the interrupt.
                        ;
0089  FA                            cli
                                    push ds             ;Save FORTRAN's ds.
008A  1E
008B  BA 021A R                     mov dx, offset isr1 ;Get interrupt routine.
008E  0E                            push cs             ;Push a copy of the current cs.
008F  1F                            pop ds              ;Pop it into ds.
0090  B0 1C                         mov al, 1Ch         ;Will make it interrupt 1Ch.
0092  B4 25                         mov ah, 25h         ;Set up interrupt at ds:dx.
0094  CD 21                         int 21h             ;DOSCALL.
0096  1F                            pop ds              ;Restore FORTRAN's ds.
0097  FB                            sti
                        ;
                        ; At this point the interrupt is installed, the timer is ticking at
                        ; the selected frequency and the baseboard is ready to go.
                        ;
                        ;-----------------------------------------------------------------

0098  07                            pop es
0099  5D                            pop bp
009A  CA 0018                       ret 24              ;Dump six parms & back to FORTRAN.
009D                    setup   endp
```

```
                        page
                        comment ~
                                 ************************************************************
                                * The CLSUP subroutine is used to restore the computer's *
                                * environment to its original condition.  This involves  *
                                * removing the interrupt handling subroutines and        *
                                * reprogramming the clock of the computer to its usual   *
                                * 18.2 Hz.                                               *
                                 ************************************************************
                                 ~
009D                    clsup   proc far

009D  55                                push bp                 ;Save framepointer.
009E  8B EC                             mov bp, sp              ;Get new framepointer.
00A0  06                                push es                 ;Save extra - seg.

00A1  FA                                cli
00A2  1E                                    push ds             ;Save FORTRAN's ds.
00A3  BA FF53                               mov dx, 0FF53h      ;Restore interrupt routine.
00A6  B8 F000                               mov ax, 0F000h      ;Get segment.
00A9  50                                    push ax             ;Put it on the stack.
00AA  1F                                    pop ds              ;Pop it into ds.
00AB  B0 1C                                 mov al, 1Ch         ;Will make it interrupt 1Ch.
00AD  B4 25                                 mov ah, 25h         ;Set up interrupt at ds:dx.
00AF  CD 21                                 int 21h             ;DOSCALL.
00B1  1F                                    pop ds              ;Restore FORTRAN's ds.
00B2  FB                                sti

                        ;---------------------------------------------------------------------
                        ; Restore clock to original interrupt frequency of 18.2 hz.
                        ;
00B3  B0 36                             mov al,00110110b        ;Timer 0, lsb,msb,mode 3,binary.
00B5  E6 43                             out timer+3,al

00B7  B0 00                             mov al,0h               ;Get a zero.
00B9  E6 40                             out timer,al            ;Send LSB.

00BB  90                                nop
00BC  90                                nop
00BD  90                                nop

00BE  E6 40                             out timer,al            ;Send MSB.

00C0  B9 4E20                           mov cx,20000
00C3                    qwert2:
00C3  E2 FE                             loop qwert2             ;Wait 70 msecs. for chip to accept.
                        ;
                        ;---------------------------------------------------------------------
```

```
                        ;---------------------------------------------------------------
                        ;The next bit of code turns off and grounds together the servo inputs.
                        ;
00C5  BA 0210                       mov dx, bbctrl0
00C8  B0 08                         mov al, bbc4l       ;Turn on bit C4.
00CA  EE                            out dx, al          ;Turn off relay.
                        ;
00CB  BA 0218                       mov dx, bbctrl2
00CE  B0 08                         mov al, bbc4l       ;Turn on bit C4.
00D0  EE                            out dx, al          ;Turn off relay.
                        ;
                        ;---------------------------------------------------------------

00D1  07                            pop es
00D2  5D                            pop bp
00D3  CB                            ret                 ;Back to FORTRAN.
00D4                    clsup       endp
```

```
                        page
                        comment ^
                                 ***************************************************************
                                * Subroutine MOTON connects the speed reference terminals      *
                                * of the servo amplifiers to the DAC outputs of the interface  *
                                * board.  The subroutine also installs the main interrupt      *
                                * handler ISR0.                                                *
                                 ***************************************************************
                          ^

00D4                    moton   proc far

00D4  55                                push bp                 ;Save framepointer.
00D5  8B EC                             mov bp, sp              ;Get new framepointer.
00D7  06                                push es                 ;Save extra - seg.

                        ;---------------------------------------------------------------
                        ;Install ISR0.
                        ;
00D8  FA                                cli
00D9  1E                                    push ds             ;Save FORTRAN's ds.
00DA  BA 0118 R                             mov dx, offset isr0 ;Get interrupt routine.
00DD  0E                                    push cs             ;Push a copy of the current cs.
00DE  1F                                    pop ds              ;Pop it into ds.
00DF  B0 1C                                 mov al, 1Ch         ;Will make it interrupt 1Ch.
00E1  B4 25                                 mov ah, 25h         ;Set up interrupt at ds:dx.
00E3  CD 21                                 int 21h             ;DOSCALL.
00E5  1F                                    pop ds              ;Restore FORTRAN's ds.
00E6  FB                                sti
                        ;
                        ; At this point the main interrupt, ISR0 is installed.
                        ;---------------------------------------------------------------

                        ;---------------------------------------------------------------
                        ;The next bit of code turns on the servo inputs.
                        ;
00E7  BA 0210                           mov dx, bbctrl0
00EA  B0 09                             mov al, bbc4h       ;Turn on bit C4.
00EC  EE                                out dx, al          ;Turn on relay.
                        ;
                        ;
00ED  BA 0218                           mov dx, bbctrl2
00F0  B0 09                             mov al, bbc4h       ;Turn on bit C4.
00F2  EE                                out dx, al          ;Turn on relay.
                        ;---------------------------------------------------------------


00F3  07                                pop es
00F4  5D                                pop bp
```

```
00F5  CB                              ret                 ;Back to FORTRAN.
00F6                          motor   endp
```

```
                        page
                        comment ^
                                 ****************************************************************
                                * Subroutine MOTOFF disconnects the speed reference terminals *
                                * of the servo amplifiers to the DAC outputs of the interface *
                                * board.  The subroutine also installs the dummy interrupt    *
                                * handler ISR1.                                               *
                                 ****************************************************************
                               ^

00F6                    motoff  proc far

00F6  55                                push bp                 ;Save framepointer.
00F7  8B EC                             mov bp, sp              ;Get new framepointer.
00F9  06                                push es                 ;Save extra - seg.


                        ;---------------------------------------------------------------
                        ;Install ISR1.
                        ;
00FA  FA                                cli
00FB  1E                                    push ds             ;Save FORTRAN's ds.
00FC  BA 021A R                             mov dx, offset isr1 ;Get interrupt routine.
00FF  0E                                    push cs             ;Push a copy of the current cs.
0100  1F                                    pop ds              ;Pop it into ds.
0101  B0 1C                                 mov al, 1Ch         ;Will make it interrupt 1Ch.
0103  B4 25                                 mov ah, 25h         ;Set up interrupt at ds:dx.
0105  CD 21                                 int 21h             ;DOSCALL.
0107  1F                                    pop ds              ;Restore FORTRAN's ds.
0108  FB                                sti
                        ;
                        ; At this point the maintenance interrupt, ISR1 is installed.
                        ;
                        ;---------------------------------------------------------------

                        ;---------------------------------------------------------------
                        ;The next bit of code turns off and grounds together the servo inputs.
                        ;
0109  BA 0210                           mov dx, bbctrl0
010C  B0 08                             mov al, bbc4l       ;Turn off bit C4.
010E  EE                                out dx, al          ;Turn off relay.
                        ;
                        ;
010F  BA 0218                           mov dx, bbctrl2
0112  B0 08                             mov al, bbc4l       ;Turn off bit C4.
0114  EE                                out dx, al          ;Turn off relay.
                        ;---------------------------------------------------------------
```

```
0115  07                              pop es
0116  5D                              pop bp
0117  CB                              ret                     ;Back to FORTRAN.
0118                        motoff      endp
```

```
                        page
                        comment ~
                                 ****************************************************************
                                * Subroutine ISR0 is the main interrupt routine and represents *
                                * the 'heart' of the Control Loops Unit.  The routine          *
                                * must perform all of the I/O functions to the external       *
                                * encoder interface board through the custom encoder interface.*
                                 ****************************************************************
                               ~

0118                    isr0            proc near           ;Interrupt routine.
0118  FB                               sti
0119  50                               push ax
011A  1E                               push ds
011B  53                               push bx             ;Save registers used.
011C  06                               push es
011D  51                               push cx
011E  52                               push dx

                        ;-------------------------------------------------------------
                        ;Set up the known data segment.

011F  B8 ---- R                        mov ax, dgroup      ;Get FORTRAN's data seg.
0122  8E D8                            mov ds,ax           ;Stuff it in ds.
                        ;
                        ;-------------------------------------------------------------


                        ;-------------------------------------------------------------
                        ;The next few lines of code are useful to add
                        ;to ensure that the interrupts are really occurring.
                        ;This code increments a counter at 0000:0182.
                        ;       mov ax,ds:[bx+2]    ;Get the next address.
                        ;       add ax,1            ;Add one.
                        ;       mov ds:[bx+2],ax    ;Inc interrupt counter.
                        ;-------------------------------------------------------------


                        ;-------------------------------------------------------------
                        ;Set the INTFLAG.
                        ;
0124  A1 0000 R                        mov ax,ds:intflag_addr  ;Get INTFLAG address.
0127  8B D8                            mov bx,ax           ;Move it into bx.
0129  A1 0002 R                        mov ax,ds:intflag_seg   ;Get INTFLAG segment.
012C  8E C0                            mov es,ax           ;Move it into es.
012E  26: C7 07 0001                   mov es:[bx], word ptr 1 ;Move 1 to INTFLAG.

0133  B2 FF                            mov dl, 0FFh        ;Direct Console input.
0135  B4 06                            mov ah, 06h         ;Check standard direct input status.
0137  CD 21                            int 21h
```

```
0139  74 05                              jz no_key_pressed
013B  26: C7 07 0002                    mov es:[bx], word ptr 2  ;Move a 2 if key pressed.
                                ;
0140  90                        no_key_pressed: nop
                                ;-------------------------------------------------------

                                ;-------------------------------------------------------
                                ;Set the INHIBIT lines high for the duration of the reads.
                                ;
0141  BA 0210                           mov dx, bbctrl0         ;Get the 8255 0 Control port.
0144  B0 0F                             mov al, bbc7h           ;Set the INHIBIT high.
0146  EE                               out dx, al              ;Do it.

0147  BA 0218                           mov dx, bbctrl2         ;Get the 8255 2 Control port.
014A  B0 0F                             mov al, bbc7h           ;Set the INHIBIT high.
014C  EE                               out dx, al              ;Do it.
                                ;
                                ;-------------------------------------------------------

                                ;-------------------------------------------------------
                                ;Read the INDEX FLAG to see if an INDEX has occured, and if so
                                ;read the latch which will contain the count at index.  Pass this
                                ;value back in FLAG0.
                                ;
014D  BA 0211                           mov dx, bb0c            ;Get the 8255 0 C port.
0150  EC                                in al, dx               ;Read the byte.
0151  24 02                            and al, 2               ;Check bit C1.
0153  74 1B                             jz no_index             ;Go ahead if no index.

                                ;Read the counter value and pass it back to FORTRAN.
                                ;Correct the counter value for the correct sign.

0155  B5 00                             mov ch,0                ;Clear upper half of register.
0157  BA 0212                           mov dx, bb0b            ;Get A data for 8255 0.
015A  EC                                in al, dx               ;Got counter value.
015B  A8 80                             test al, 80h            ;Test sign bit.

015D  79 02                             jns posnum              ;It's pos. go ahead.
015F  B5 FF                             mov ch, 0FFh            ;Set upper bits esp. sign.
0161                            posnum:
0161  8A C8                             mov cl, al              ;Store counter value.
0163  A1 0004 R                         mov ax,ds:flag0_addr    ;Get FLAG0 address.
0166  8B D8                             mov bx,ax               ;Move it into bx.
0168  A1 0006 R                         mov ax,ds:flag0_seg     ;Get FLAG0 segment.
016B  8E C0                             mov es,ax               ;Move it into es.
016D  26: 89 0F                         mov es:[bx], cx         ;Pass it back to FORTRAN.
                                ;
```

```
                        ;-------------------------------------------------------------


0170                    no_index:

                        ;-------------------------------------------------------------
                        ;Strobe the transparent latch to capture and clear the counter.
                        ;
0170  BA 0210                       mov dx, bbctrl0     ;Get the control for 8255 0.
0173  B0 0B                        mov al, bbc5h       ;Set bit high.
0175  EE                           out dx, al          ;Do it.
0176  B0 0A                        mov al, bbc5l       ;Set bit low.
0178  EE                           out dx, al          ;Do it.
                        ;-------------------------------------------------------------


                        ;-------------------------------------------------------------
                        ;Read the counter value and pass it back to FORTRAN.
                        ;Correct the counter value for the correct sign.

0179  B5 00                        mov ch,0            ;Clear upper half of register.
017B  BA 0212                      mov dx, bb0b        ;Get A data for 8255 0.
017E  EC                           in al, dx           ;Got counter value.
017F  A8 80                        test al, 80h        ;Test sign bit.

0181  79 02                        jns posnum2         ;It's pos. go ahead.
0183  B5 FF                        mov ch, 0FFh        ;Set upper bits esp. sign.
0185                    posnum2:
0185  8A C8                        mov cl, al          ;Store counter value.
0187  A1 0014 R                    mov ax,ds:count0_addr ;Get ENCODER address.
018A  8B D8                        mov bx,ax           ;Move it into bx.
018C  A1 0016 R                    mov ax,ds:count0_seg ;Get ENCODER segment.
018F  8E C0                        mov es,ax           ;Move it into es.
0191  26: 89 0F                    mov es:[bx], cx     ;Pass it back to FORTRAN.
                        ;
                        ;-------------------------------------------------------------

                        ;-------------------------------------------------------------
                        ;The following code will write out the passed FORTRAN ERR0 value to
                        ;the DA converter.
                        ;

0194  A1 000C R                     mov ax,ds:DA0_addr  ;Get the DA number offset.
0197  8B D8                        mov bx,ax           ;Move it into bx.
0199  A1 000E R                    mov ax,ds:da0_seg   ;Get the DA number segment.
019C  8E C0                        mov es,ax           ;Move it into es.
019E  26: 8B 0F                    mov cx,es:[bx]      ;Get the DA number.
                        ;
                        ;CX should now contain the FORTRAN's chosen error number.
```

```
                        ;
                        ;It must be decoded from +/- binary to offset binary.
                        ;This decoding will be done by adding decimal 127.
                        ;
                        ;
01A1  83 D1 7F                  adc cx, 127             ;Add offset.
01A4  BA 0213                   mov dx, bb0a            ;Get the port.
01A7  8A C1                     mov al, cl              ;Get the byte.
01A9  EE                        out dx, al              ;Send it out.

                        ;
                        ;-----------------------------------------------------------------



                        ;---------------------------------------------------------
                        ;
                        ;  SECOND AXIS
                        ;
                        ;Read the INDEX FLAG to see if an INDEX has occured, and if so
                        ;read the latch which will contain the count at index.  Pass this
                        ;value back in FLAG1.
                        ;

01AA  BA 0219                   mov dx, bb2c            ;Get the 8255 2 C port.
01AD  EC                        in al, dx               ;Read the byte.
01AE  24 02                     and al, 2               ;Check bit C1.
01B0  74 1B                     jz no_index2            ;Go ahead if no index.

                        ;Read the counter value and pass it back to FORTRAN.
                        ;Correct the counter value for the correct sign.

01B2  B5 00                     mov ch,0                ;Clear upper half of register.
01B4  BA 021A                   mov dx, bb2b            ;Get A data for 8255 2.
01B7  EC                        in al, dx               ;Got counter value.
01B8  A8 80                     test al, 80h            ;Test sign bit.

01BA  79 02                     jns posnum3             ;It's pos. go ahead.
01BC  B5 FF                     mov ch, 0FFh            ;Set upper bits esp. sign.
01BE                    posnum3:
01BE  8A C8                     mov cl, al              ;Store counter value.
01C0  A1 0008 R                 mov ax,ds:flag1_addr    ;Get FLAG1 address.
01C3  8B D8                     mov bx,ax               ;Move it into bx.
01C5  A1 000A R                 mov ax,ds:flag1_seg     ;Get FLAG1 segment.
01C8  8E C0                     mov es,ax               ;Move it into es.
01CA  26: 89 0F                 mov es:[bx], cx         ;Pass it back to FORTRAN.
                        ;
                        ;---------------------------------------------------------
```

```
01CD                    no_index2:

                        ;-------------------------------------------------------------
                        ;Strobe the transparent latch to capture and clear the counter.
                        ;
01CD  BA 0218                       mov dx, bbctrl2     ;Get the control for 8255 2.
01D0  B0 0B                         mov al, bbc5h       ;Set bit high.
01D2  EE                            out dx, al          ;Do it.
01D3  B0 0A                         mov al, bbc5l       ;Set bit low.
01D5  EE                            out dx, al          ;Do it.
                        ;-------------------------------------------------------------


                        ;-------------------------------------------------------------
                        ;Read the counter value and pass it back to FORTRAN.
                        ;Correct the counter value for the correct sign.

01D6  B5 00                         mov ch,0            ;Clear upper half of register.
01D8  BA 021A                       mov dx, bb2b        ;Get A data for 8255 2.
01DB  EC                            in al, dx           ;Got counter value.
01DC  A8 80                         test al, 80h        ;Test sign bit.

01DE  79 02                         jns posnum4         ;It's pos. go ahead.
01E0  B5 FF                         mov ch, 0FFh        ;Set upper bits esp. sign.
01E2                    posnum4:
01E2  8A C8                         mov cl, al          ;Store counter value.
01E4  A1 0018 R                     mov ax,ds:count1_addr ;Get ENCODER address.
01E7  8B D8                         mov bx,ax           ;Move it into bx.
01E9  A1 001A R                     mov ax,ds:count1_seg ;Get ENCODER segment.
01EC  8E C0                         mov es,ax           ;Move it into es.
01EE  26: 89 0F                     mov es:[bx], cx     ;Pass it back to FORTRAN.
                        ;
                        ;-------------------------------------------------------------

                        ;-------------------------------------------------------------
                        ;The following code will write out the passed FORTRAN ERR1 value to
                        ;the DA converter.
                        ;

01F1  A1 0010 R                     mov ax,ds:DA1_addr  ;Get the DA number offset.
01F4  8B D8                         mov bx,ax           ;Move it into bx.
01F6  A1 0012 R                     mov ax,ds:DA1_seg   ;Get the DA number segment.
01F9  8E C0                         mov es,ax           ;Move it into es.
01FB  26: 8B 0F                     mov cx,es:[bx]      ;Get the DA number.
                        ;
                        ;CX should now contain the FORTRAN's chosen error number.
                        ;
                        ;It must be decoded from +/- binary to offset binary.
```

```
                        ;This decoding will be done by adding decimal 127.
                        ;
                        ;
01FE  83 D1 7F                  adc cx, 127             ;Add offset.
0201  BA 021B                   mov dx, bb2a            ;Get the port.
0204  8A C1                     mov al, cl              ;Get the byte.
0206  EE                        out dx, al              ;Send it out.

                        ;
                        ;---------------------------------------------------------------

                        ;-------------------------------------------------------
                        ;Lower the INHIBIT lines.
                        ;
0207  BA 0210                   mov dx, bbctrl0         ;Get the 8255 0 Control port.
020A  B0 0E                     mov al, bbc7l           ;Set the INHIBIT low.
020C  EE                        out dx, al              ;Do it.

020D  BA 0218                   mov dx, bbctrl2         ;Get the 8255 2 Control port.
0210  B0 0E                     mov al, bbc7l           ;Set the INHIBIT low.
0212  EE                        out dx, al              ;Do it.

                        ;
                        ;-------------------------------------------------------
0213  5A                pop dx
0214  59                pop cx
0215  07                pop es
0216  5B                pop bx                ;Restore registers used.
0217  1F                pop ds
0218  58                pop ax

0219  CF                iret
021A              isr0    endp

021A              isr1          proc near             ;Interrupt routine.
021A  FB                  sti
021B  CF                  iret
021C              isr1    endp

021C              code    ends
                  end
```

Structures and records:

Name	Width / Shift	# fields / Width	Mask	Initial
FRAME.	0022	0009		
SAVEBP	0000			
SAVERET.	0002			
COUNT1	0006			
COUNT0	000A			
DA1.	000E			
DA0.	0012			
FLAG1.	0016			
FLAG0.	001A			
INTFLAG.	001E			

Segments and groups:

Name	Size	align	combine	class
CODE	021C	PARA	NONE	'CODE'
DGROUP	GROUP			
DATA	001C	PARA	PUBLIC	'DATA'

Symbols:

Name	Type	Value	Attr
BASE	Number	0210	
BB0A	Number	0213	
BB0B	Number	0212	
BB0C	Number	0211	
BB1A	Number	0217	
BB1B	Number	0216	
BB1C	Number	0215	
BB2A	Number	021B	
BB2B	Number	021A	
BB2C	Number	0219	
BB3A	Number	021F	
BB3B	Number	021E	
BB3C	Number	021D	
BBC4H.	Number	0009	
BBC4L.	Number	0008	
BBC5H.	Number	000B	
BBC5L.	Number	000A	
BBC6H.	Number	000D	
BBC6L.	Number	000C	
BBC7H.	Number	000F	
BBC7L.	Number	000E	
BBCTRL0.	Alias	BASE	

BBCTRL1.	Number	0214		
BBCTRL2.	Number	0218		
BBCTRL3.	Number	021C		
CLSUP.	F PROC	009D	CODE	Global Length =0037
COUNT0_ADDR.	L WORD	0014	DATA	
COUNT0_SEG	L WORD	0016	DATA	
COUNT1_ADDR.	L WORD	0018	DATA	
COUNT1_SEG	L WORD	001A	DATA	
CTRL3.	Number	0083		
DA0_ADDR	L WORD	000C	DATA	
DA0_SEG.	L WORD	000E	DATA	
DA1_ADDR	L WORD	0010	DATA	
DA1_SEG.	L WORD	0012	DATA	
FLAG0_ADDR	L WORD	0004	DATA	
FLAG0_SEG.	L WORD	0006	DATA	
FLAG1_ADDR	L WORD	0008	DATA	
FLAG1_SEG.	L WORD	000A	DATA	
INTFLAG_ADDR	L WORD	0000	DATA	
INTFLAG_SEG.	L WORD	0002	DATA	
ISR0	N PROC	0118	CODE	Length =0102
ISR1	N PROC	021A	CODE	Length =0002
MOTOFF	F PROC	00F6	CODE	Global Length =0022
MOTON.	F PROC	00D4	CODE	Global Length =0022
NO_INDEX	L NEAR	0170	CODE	
NO_INDEX2.	L NEAR	01CD	CODE	
NO_KEY_PRESSED	L NEAR	0140	CODE	
POSNUM	L NEAR	0161	CODE	
POSNUM2.	L NEAR	0185	CODE	
POSNUM3.	L NEAR	01BE	CODE	
POSNUM4.	L NEAR	01E2	CODE	
QWERT.	L NEAR	0087	CODE	
QWERT2	L NEAR	00C3	CODE	
SETUP.	F PROC	0000	CODE	Global Length =009D
TIMER.	Number	0040		

Warning Errors	Severe Errors
0	0

```
                        ; compiler directives for program listing
                        ;
                        TITLE "GETKY - FORTRAN CALLABLE ASM SUBROUTINE
                        - 85/02/19"
                        PAGE    54,132
                        COMMENT \

                        ********************************************************************************
                        *   SUBROUTINE NAME : (GETKY) Get keyboard scan code                           *
                        *   FILE NAME       : getky1.ASM                                               *
                        *   VERSION         : 1                                                        *
                        *   DATE REVISED    : 85/02/20                                                 *
                        *   AUTHOR(S)       : O.B. WOLFE                                               *
                        *   PURPOSE : This FORTRAN callable subroutine uses ROM BIOS interrupt         *
                        *   16H to read the the keyboard scan and exetended codes. Both codes          *
                        *   are set to default to 0 if no key has been pressed.                        *
                        *                                                                              *
                        *   CALLED BY:  CALL GETKY(ACOD,SCOD)                                          *
                        *   INPUT:  none                                                               *
                        *   OUTPUT: acod - standard IBM ascii key code (000 - 255)                     *
                        *           scod - IBM keyboard scan code      (000 - 255)                     *
                        *                                                                              *
                        ********************************************************************************

                        \
                        ;
                        ; place data segment definition here
                        ;
0000                    DATA            SEGMENT PUBLIC 'DATA'
                        ;
                        ; declare user defined variables used in assembler subroutine
                        ;
                                        EVEN            ; force to even word boundries
0000  00                acod_val        DB      0       ; key code value
0001  00                scod_val        DB      0       ; extended code value
                                        EVEN
0002  0000              acod_addr       DW      0       ; key code offset address
0004  0000              acod_segt       DW      0       ; key code segment address
0006  0000              scod_addr       DW      0       ; extended code offset address
0008  0000              scod_segt       DW      0       ; extended code segment address
                        ;
000A                    DATA            ENDS
                        ;
                        DGROUP          GROUP   DATA
0000                    CODE            SEGMENT 'CODE'
                                        ASSUME  CS:CODE,DS:DGROUP,SS:DGROUP
                        ;
                                        PUBLIC  getky
0000                    getky           PROC    FAR
```

```
                        ;
                        ; set up a data structure to access parameters passed to subroutine
                        ;
                        FRAME           STRUC
0000  0000              SAVEBP          DW
0002  00 00 00 00       SAVERET         DD
0006  00 00 00 00       PARM2           DD
000A  00 00 00 00       PARM1           DD
000E                    FRAME           ENDS
                        ;
                        ; initialize stack by saving FORTRAN'S BP and ES registers on the stack
                        ;
0000  55                                PUSH    BP                      ; save base pointer register
0001  8B EC                             MOV     BP,SP                   ; move stack ptr to base ptr
0003  06                                PUSH    ES                      ; save extra segment register
                        ;
                        ; transfer parameter memory segment and offset addresses to
                        ; user defined variables. (ie. acod, scod )
                        ;
0004  C4 5E 06                          LES     BX,[BP].PARM2           ; ADDR PARM2 -> scod
0007  89 1E 0006 R                      MOV     scod_addr,BX
000B  8C 06 0008 R                      MOV     scod_segt,ES
000F  C4 5E 0A                          LES     BX,[BP].PARM1           ; ADDR PARM1 -> acod
0012  89 1E 0002 R                      MOV     acod_addr,BX
0016  8C 06 0004 R                      MOV     acod_segt,ES
                        ;
                        ; initialize default values for returning parameters
                        ;
001A  C6 06 0000 R 00                   MOV     acod_val,0              ; default key code
001F  C6 06 0001 R 00                   MOV     scod_val,0              ; default extended code
                        ;
                        ; make ROM BIOS call to read keyboard
                        ;
0024  B4 00                             MOV     AH,0                    ; select read ascii function
0026  CD 16                             INT     16H                     ; call function
0028  A2 0000 R                         MOV     acod_val,AL             ; move key code into acod
002B  88 26 0001 R                      MOV     scod_val,AH             ; move extended code into scod
                        ;
                        ; end of executable assembler subroutine code.
                        ;
                        ; transfer returning data from user defined variables to the memory
                        ; locations of the passed parameters.
                        ;
002F  8B 1E 0006 R                      MOV     BX,scod_addr            ; setup scod offset address
0033  8E 06 0008 R                      MOV     ES,scod_segt            ; setup scod segment address
0037  8A 0E 0001 R                      MOV     CL,scod_val             ; load accum. with 1 byte val
003B  26: 88 0F                         MOV     ES:[BX],CL              ; move to FORTRAN KEYCOD addr.
003E  8B 1E 0002 R                      MOV     BX,acod_ADDR            ; setup acod offset address
0042  8E 06 0004 R                      MOV     ES,acod_segt            ; setup acod segment address
```

```
0046  8A 0E 0000 R                    MOV     CL,accd_val             ; load accum. with 1 byte val.
004A  26: 88 0F                       MOV     ES:[BX],CL              ; move to FORTRAN EXTCOD addr.
                                      ;
                                      ; restore FORTRAN'S BP and ES registers and stack, dispose of passed
                                      ; parameter addresses and return to calling program.
                                      ;
004D  07                              POP     ES
004E  5D                              POP     BP
004F  CA 0008                         RET     8                       ; 4 * 2 passed parameters
0052                                  ENDP                            ; end of subroutine
0052                                  ENDS                            ; end of code segment
                                      END
```

Structures and records:

Name	Width / Shift	# fields / Width	Mask	Initial
FRAME.	000E	0004		
SAVEBP	0000			
SAVERET.	0002			
PARM2.	0006			
PARM1.	000A			

Segments and groups:

Name	Size	align	combine	class
CODE	0052	PARA	NONE	'CODE'
DGROUP	GROUP			
DATA	000A	PARA	PUBLIC	'DATA'

Symbols:

Name	Type	Value	Attr		
ACOD_ADDR.	L WORD	0002	DATA		
ACOD_SEGT.	L WORD	0004	DATA		
ACOD_VAL	L BYTE	0000	DATA		
GETKY.	F PROC	0000	CODE	Global	Length =0052
SCOD_ADDR.	L WORD	0006	DATA		
SCOD_SEGT.	L WORD	0008	DATA		
SCOD_VAL	L BYTE	0001	DATA		

Warning Errors	Severe Errors
0	0

APPENDIX IV

Details of the Encoder Interface

The encoder interface consisted of two main parts, the commercial I/O board, and the custom interface. The commercial I/O board was a TECMAR BASEBOARD™ and consisted of four 8255 programmable peripheral interface adapters. These chips are extremely powerful and have many different modes of operation. Figure D1 provides a conceptual view of how the BASEBOARD was used. Only a single 8255 is needed for each axis drive system. The milling machine project only used two drive systems and, hence, only two of the 8255 chips were used.

Each 8255 has 24 fully programmable I/O lines. The 8255 was used in mode 0 with a control word of 83_{16}. (Refer to the Intel 8255 datasheet for more details.) Figure D2 provides a description of how the lines were utilized. Port A was used as a latched output and was used to contain the output word for the axis DAC. Port B was programmed as an input and was used to read the contents of the 193 counters via the 373 latch. Port C was split so that bits 0-3 were input lines and bits 4-7 were output lines. Bit C1 served to indicate that an INDEX pulse had occurred. This pulse marked a

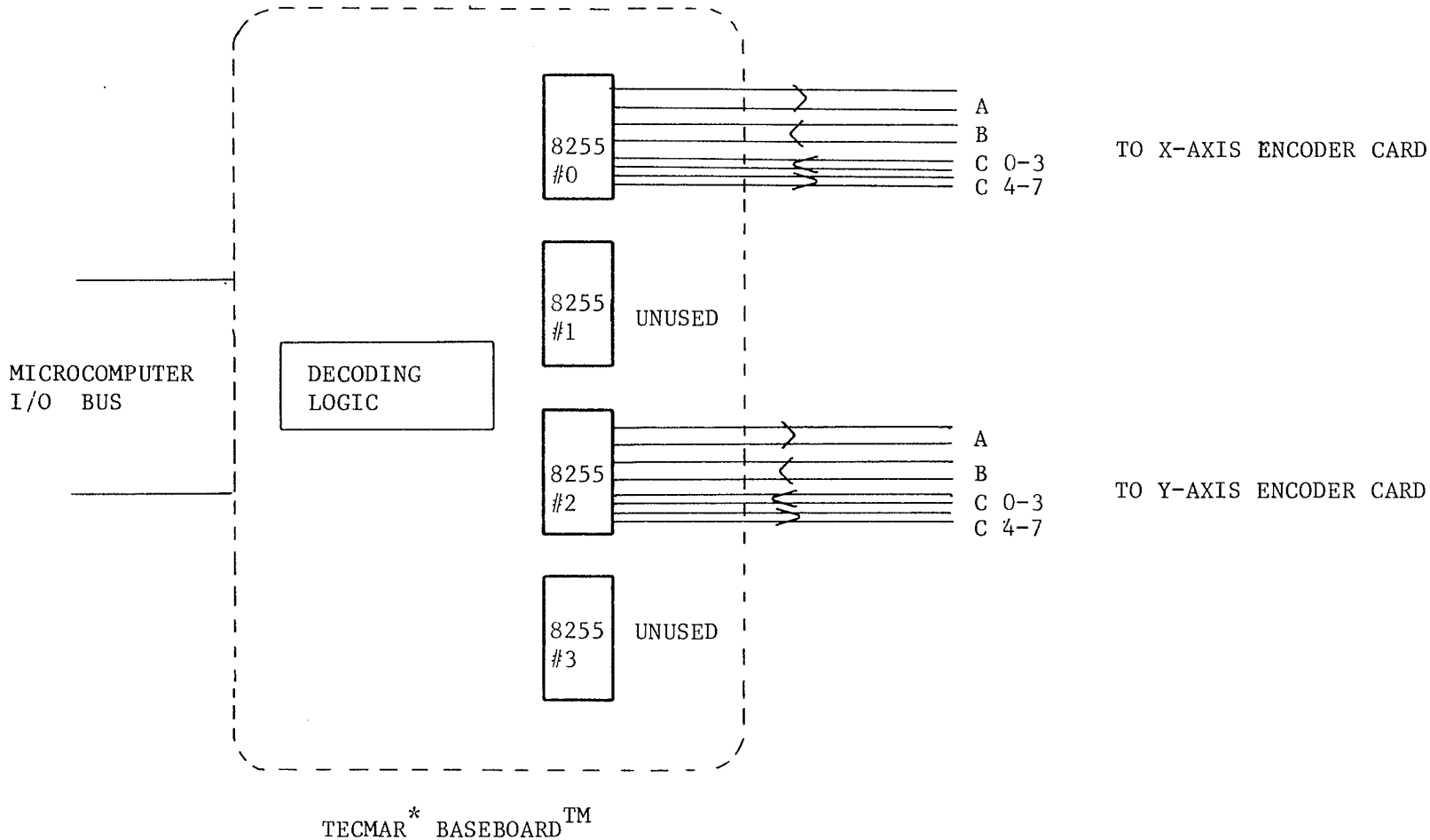
complete revolution of the encoder and indicated that the absolute position of the axes should be an integer multiple of the encoder's pulse count. Bit C4 was used to electrically connect and disconnect the servo motor amplifier speed reference terminal to and from the DAC outputs. Disconnecting the motor prevented it from drifting whenever the computer was not sampling the axes' positions. The STROBE line was triggered by the ASSEMBLER ISR0 subroutine and this initiated the loading of the 373 latch and the subsequent clearing of the 193 counters. The INHIBIT line was used to prevent an unexpected INDEX pulse from occurring while the computer was reading the latch's status.

Figure D3 shows the details of the control circuitry used to control the encoder interface shown in Figure D4. Consider a scenario whereby the ASSEMBLER subroutine ISR0 receives an interrupt and samples the encoder interface. The subroutine first sets the INHIBIT line high. This line prevents index pulses from affecting the circuitry by effectively blocking them at the AND gate in Figure D3. The subroutine then reads the FLAG to determine whether an INDEX pulse from the encoder has occurred during the previous sample. If an INDEX sample has not occurred, the subroutine then sets the STROBE line high and then low. On the leading edge of the ensuing pulse, the first 122 one-shot fires and

loads the 373 latch with the 193 counter's current contents. The trailing edge of the pulse from the first 122 one-shot also fires the second 122 one-shot which clears the counter's contents. If the index FLAG had been set (which has been assumed false for this example), the second 122 one-shot would have also cleared the 74 Flip-Flop which would have been indicating the presence of the INDEX count.

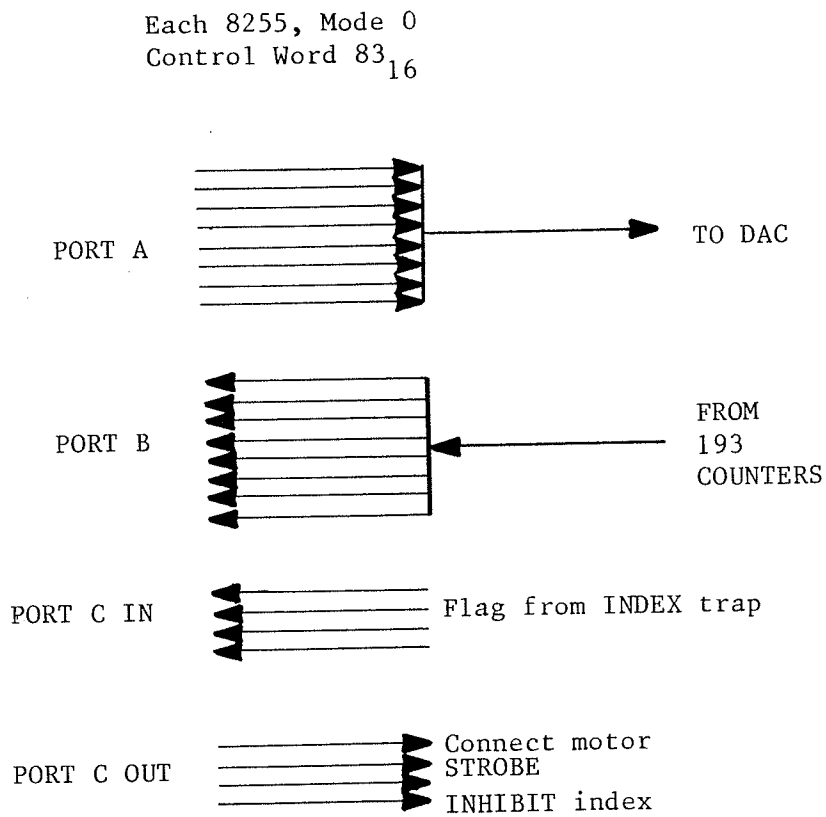
Now consider the more complex scenario whereby an INDEX count had occurred during the previous sampling period. The computer would have been busy elsewhere. Thus the INHIBIT line would not have prevented the INDEX pulse from triggering the circuitry. This INDEX pulse would have loaded the 373 latch with the 193 counter value's at the <u>moment of the INDEX pulse</u>. Simultaneously, the pulse from the 122 one shot would have set the 74 Flip-Flop. Note that the INDEX pulse would <u>not</u> have cleared the 193 counters. When the computer next samples the interface, it would find the FLAG set. This would indicate that an INDEX count was available in the 373 latch and the ASSEMBLER subroutine ISR0 would read the latch contents via 8255 port B <u>without</u> strobing the data. After reading this value, the subroutine would then STROBE the counter to get the 193 counter value at the sampling instant. This STROBE action would fire the one-shots as before and thus clear

the counters and the FLAG.



*TECMAR INC., 6225 Cochran Rd.
Cleveland Ohio 44139

FIGURE D1 CONCEPTUAL VIEW OF TECMAR™ BASEBOARD



Each 8255 parallel port is subdivided into three groups, A, B, and C.

For mode 0, control word 83_{16}

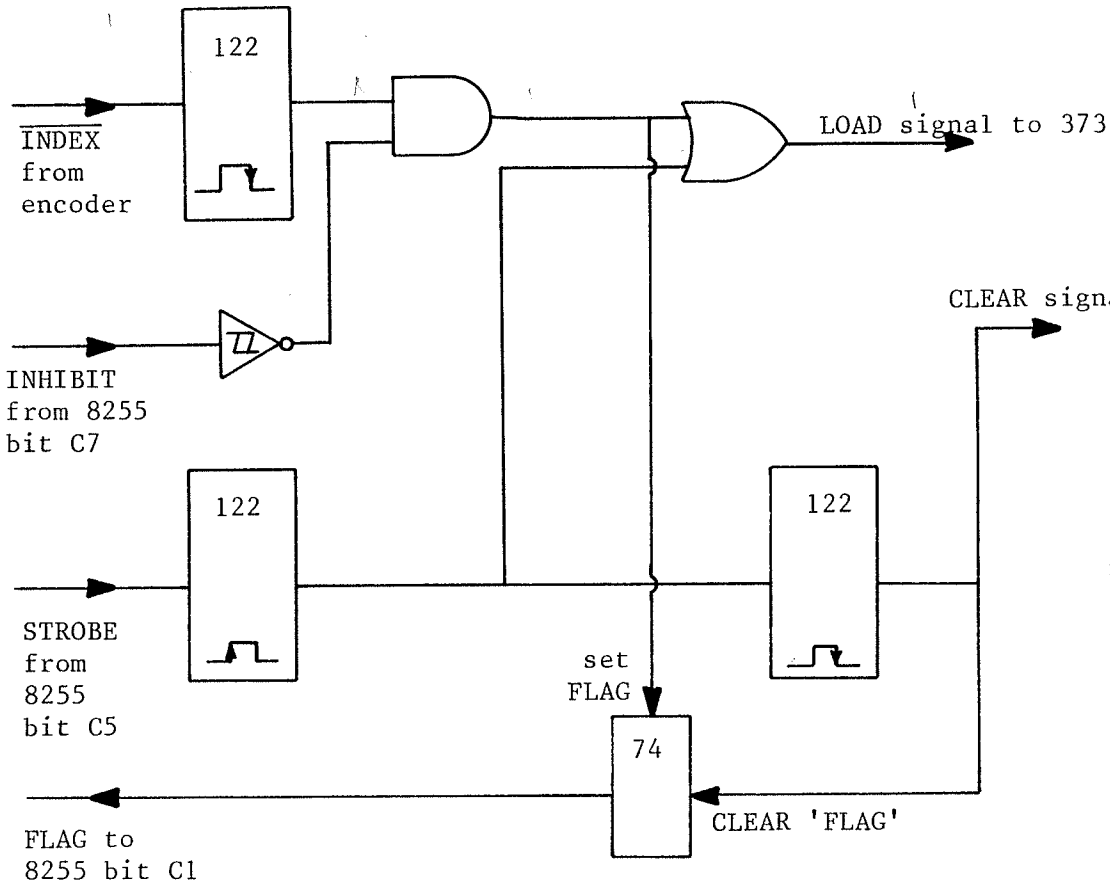
'A' is a latched output and is used to contain the output word for the DAC.

'B' is an input and is used to read the contents of the 193 counters via the 373 latch.

'C' is subdivided whereas bits 0-3 are input lines and bits 4-7 are output lines.

Bit C1 serves to indicate whenever an index pulse has occurred.
Bit C4 is used to connect the servo motor speed reference terminals to the DAC output.
Bit C5 is used to strobe the counter contents into the latch and to start the 122 one-shots which clear the counters.
Bit C7 is used to prevent index pulses while the computer is reading the latch's contents.

FIGURE D2 USAGE OF 8255 I/O LINES



$\overline{\text{INDEX}}$ - signal from encoder which goes low once/rev.

INHIBIT - signal from microcomputer which prevents INDEX pulse from triggering during a read.

FLAG - signal which indicates the presence of an index count in the latch.

FIGURE D3 CONTROL CIRCUITRY FOR ENCODER INTERFACE

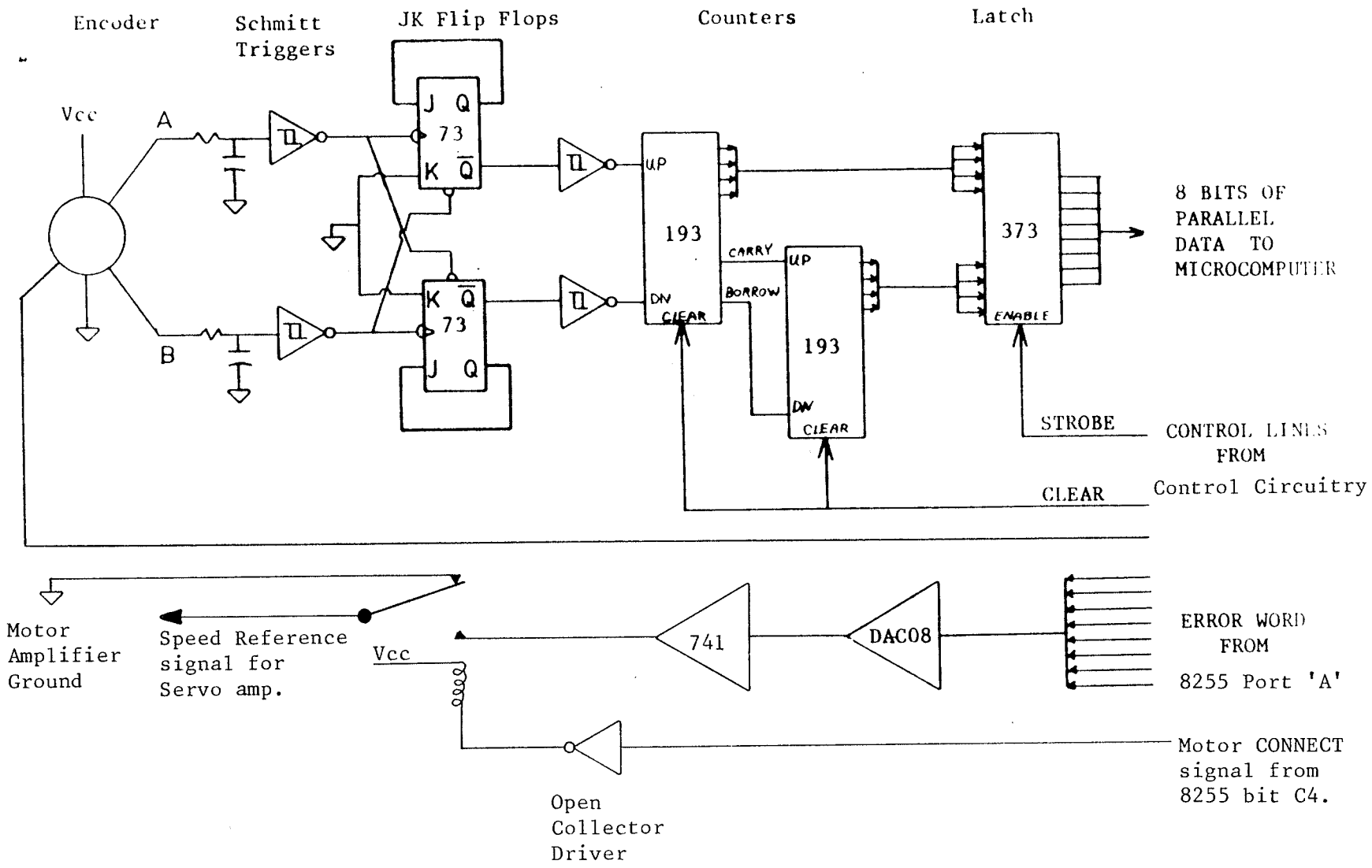


FIGURE D4 ENCODER INTERFACE