Q U A L I T Y ▪ I N ▪ A U T O M A T I O N

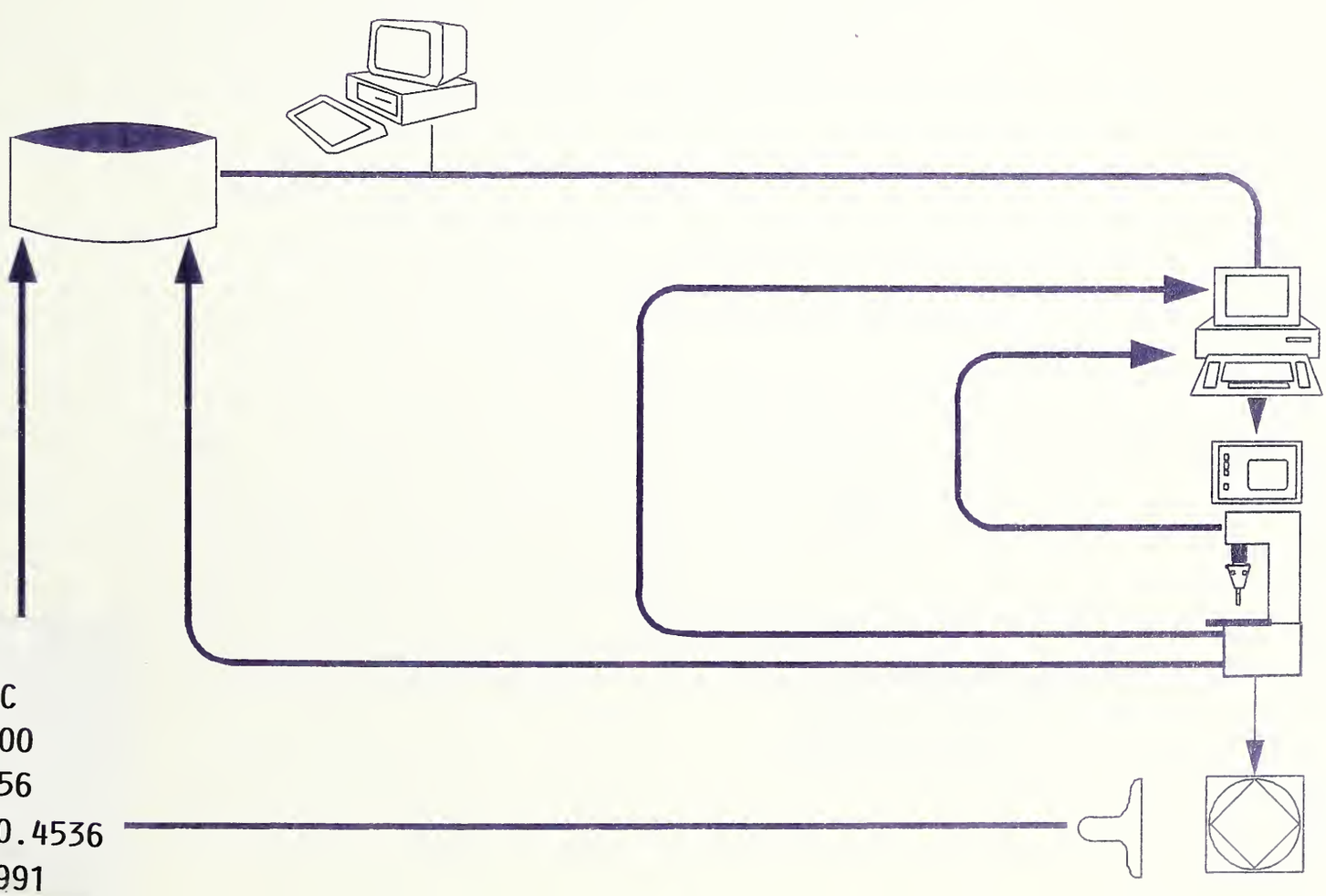
Progress Report of the Quality in Automation Project for FY90

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards
and Technology
Gaithersburg, MD 20899

NISTIR 4536

March 1991

Edited by:
M.A. Donmez



NISTIR 4536

PROGRESS REPORT OF THE QUALITY IN AUTOMATION PROJECT FOR FY90

Edited by
M. A. Donmez

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Gaithersburg, MD 20899

March 1991

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

DISCLAIMER

Commercial equipment or materials are identified in this report in order to specify adequately certain experimental procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the material or equipment identified is necessarily the best available for the purpose.

ACKNOWLEDGMENTS

We are grateful to the following industrial collaborators for their interest and technical contributions to the project:

Automation Software
CADKEY, Inc.
CMX Systems, Inc.
ICAMP, Inc.
Monarch Cortland
Monarch Sidney
Renishaw, Inc.
Sheffield Measurement
Zenith Data Systems

We acknowledge technical contributions from Dr. J. Filliben, R. Gavin, W. Guthrie, K. Harper, J. Riganati, B. Scace, and Dr. T. Vorburger.

We would like to thank K. Allen, P. Wetzel, and W. Wyatt for their assistance in the preparation of this report.

We also acknowledge the U.S. Navy's Manufacturing Technology Program, whose funding and support has made the reported progress in this project possible.

ABSTRACT

This document describes the progress of the Quality In Automation (QIA) program of the Manufacturing Engineering Laboratory (formerly the Center for Manufacturing Engineering) at the National Institute of Standards and Technology (NIST) for fiscal year 1990. The purpose of the QIA program is to develop a quality control and assurance system that exploits deterministic manufacturing principles in small-batch automated manufacturing using commercially available and affordable equipment. The foundation of the program is a quality-control architecture that uses multiple feed-back loops to control the process. Currently, we are implementing three control loops: real-time control, process-intermittent control, and post-process control loops.

During fiscal year 1990 work concentrated on the real-time and the process-intermittent control loops. The Real-Time Error Corrector (RTEC) was integrated into the real-time control loop. Its ability to modify the tool path--without any intrusion into the machine tool's CNC controller--during cutting was demonstrated for the first time. High-speed, on-machine part inspection using the fast-probing capability of the RTEC was also demonstrated as an integral part of the process-intermittent control loop. Preliminary testing has been done to evaluate the methodology for using errors determined by process-intermittent gauging, i.e., fast probing, to alter the tool path of the finishing cut by modification of NC part-program coordinates. Coordinated operation of both the real-time and the process-intermittent control loops has been demonstrated under the supervision of the turning center's Quality Controller (QC) running in a high-level language environment. As a part of the post-process control loop, an automated inspection system using the Dimensional Measurement Interface Specification (DMIS) and the Initial Graphics Exchange Specification (IGES) national interface standards to integrate Computer Aided Design/Computer Aided Manufacturing (CAD/CAM) software with a Coordinate Measuring Machine (CMM) has become operational.

This program is funded jointly by the U.S. Navy's Manufacturing Technology Program in support of the research at the Automated Manufacturing Research Facility (AMRF) at NIST, by private industry in the form of equipment and necessary software support, and by NIST.

TABLE OF CONTENTS

1. INTRODUCTION

M.A. Donmez and K.W. Yee

This report describes the progress of the Quality in Automation (QIA) program of the Manufacturing Engineering Laboratory (formerly the Center for Manufacturing Engineering) of the National Institute for Standards and Technology (NIST) for fiscal year 1990. The purpose of the QIA program is to develop a quality control/quality assurance system that exploits deterministic manufacturing principles in small-batch automated manufacturing using commercially available and affordable equipment. Deterministic manufacturing is based on the premise that most errors in the manufacturing process are repeatable and predictable and therefore can be compensated. Thus, quality can be assured by controlling both the manufacturing process and the equipment used in this process. The program combines statistical process control methods with on-machine sensing and gauging, real-time error compensation and distributed processing to produce quality parts.

Prior work has been described in the progress reports for the QIA program for fiscal years 1988 and 1989 [1,2]. The foundation of the program is a quality-control architecture that uses multiple feed-back loops to control the process as shown in Figure 1.1. There is a real-time loop, a process-intermittent loop and a post-process loop.

The function of the **real-time** control loop is to monitor the machine tool and the metal-cutting process and to modify the tool path, feed rate, and the spindle speed during cutting in order to achieve higher accuracy and surface quality of the workpiece. Monitoring can be done using various sensors incorporated into the machine tool such as position, temperature, vibration, audio and ultrasonic sensors. Real-time error compensation is achieved by implementing tool-path modification of the machine using a combination of kinematic and geometric-thermal (G-T) models of the machine-tool errors. The G-T model is derived from the **pre-process machine characterization** measurements. The kinematic model is constructed, based on the machine structure, using the theory of rigid body kinematics to describe the relative relationships between machine elements. The required correction to compensate for the resultant error, which is calculated using these models, is implemented by the Real-Time Error Corrector (RTEC). The RTEC is a microcomputer-controlled device

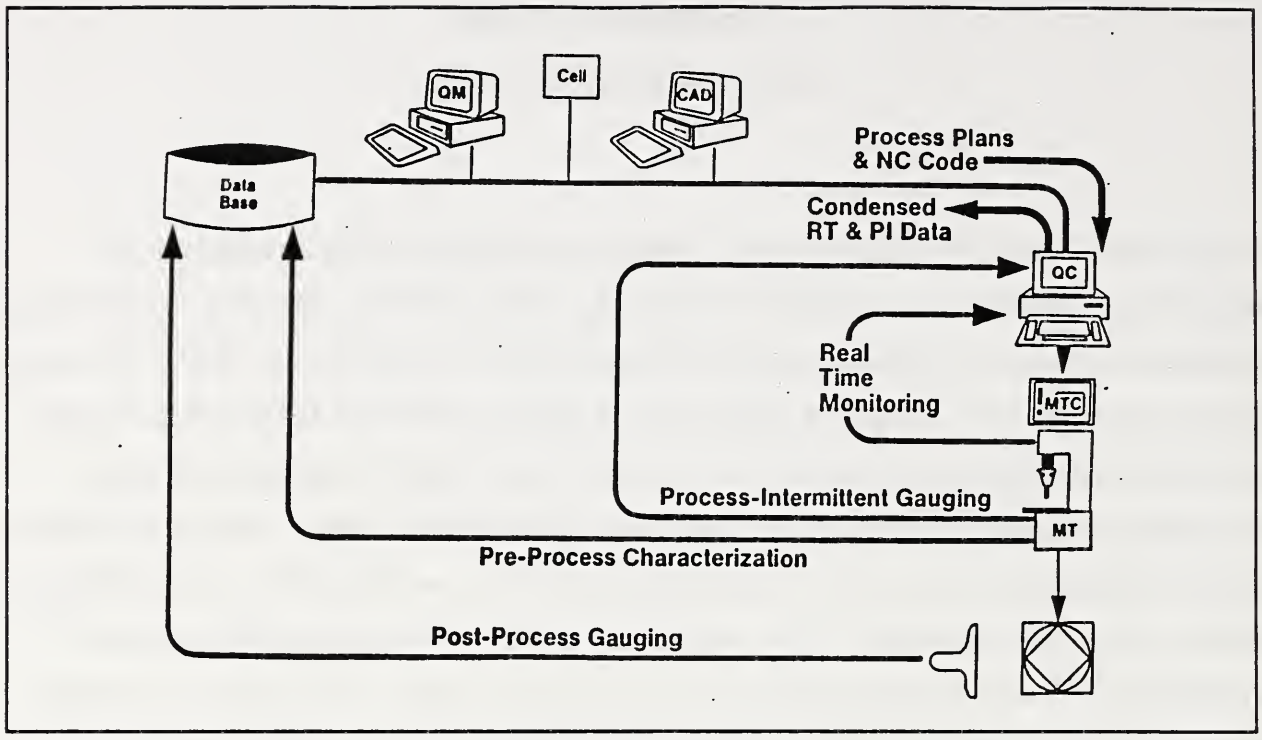


Figure 1.1

which is inserted between the position feedback device for each axis and the machine tool controller. It alters the feedback signal to cause the machine to go to a slightly different position to compensate for the predicted errors. The feed-rate and the spindle-speed modifications will be made either to minimize the vibration and chatter during cutting, or to optimize surface finish.

The functions of the **process-intermittent** control loop are 1) to determine workpiece errors introduced by the machining process which cannot be compensated by the real-time control loop and 2) to correct for them by generating modified NC part program and/or tool dimension offsets for the finishing cut. On-machine dimensional and shape measurements of the workpiece are performed (between semifinishing and finishing cuts) using fast probing. Fast probing is a gauging method which uses a touch-trigger probe at feed rates 10 to 20 times higher than is currently used.

The functions of the **post-process** control loop are to verify the cutting process and to tune the two other control loops by detecting residual systematic errors in the process and generating corrective actions. Process verification, over a period of time, is done by

inspecting the features of finished parts independently using a coordinate measuring machine (CMM). The errors measured on similar features, e.g., circles, are then correlated back to machine tool errors and the process parameters. Systematic residual errors will be used to determine modifications to the algorithm used in the process-intermittent control loop to alter the finishing cut based on the errors detected by on-machine gauging. In the long term, systematic errors of geometric features measured in the post-process control loop will be used 1) to modify the geometric-thermal model used in the real-time control loop or 2) to indicate that the pre-process characterization of the machine tool geometric errors needs to be revised.

The QIA architecture is being tested on a Monarch VMC-75 Vertical Machining Center with a GE-2000M CNC controller and a Monarch Metalist Turning Center with a GE-2000T CNC controller. The post-process control loop uses a Sheffield Apollo Series Cordax CMM. Each machine tool and the CMM has a PC-compatible computer called the Quality Controller (QC). At each machine tool, the QC is interfaced to the RTEC. Using the geometric-thermal model and a kinematic error model of the machine structure, the QC calculates the resultant error vector. This is converted to corrections required for each axis in units of correction counts and sent to the RTEC. The RTEC implements these corrections as fast as possible and reports the current correction status back to the QC. In the process-intermittent control loop, the QC receives the trip-point axes positions from the RTEC and stores this data. When probing is complete, the QC determines the shape and the dimensions of the gauged part and calculates the modification to the tool offsets or NC-part program coordinates to be used for the finishing cut. For the post-process control loop, the PC at the CMM is used off line to generate the inspection program in a Dimensional Measuring Interface Standard (DMIS) format. The inspection results are reported back to the PC in DMIS format for analysis. Complete understanding of the system requires a review of previous work reported in [1, 2].

The following sections of this report describe the progress of the project during FY90. For the real-time control loop, the geometric-thermal machine characterization measurements for each axis have been completed for both machine tools. Errors associated with thermally induced spindle growth of the machines remain to be determined. Development of the G-T models is underway for both machines. The RTEC was integrated into the real-time control

loop of the QIA architecture. Its capability to modify the tool path - without any intrusion into the machine tool's CNC controller - during cutting was demonstrated for the first time in the past year. High-speed, on-machine part inspection using the fast probing capability of the RTEC was also demonstrated as an integral part of the process-intermittent control loop. Preliminary testing has been done to evaluate the methodology for using errors determined by process-intermittent gauging to alter the tool path of the finishing cut by modification of NC part-program coordinates. Again for the first time, a coordinated operation of both the real-time and the process-intermittent control loops has been demonstrated under the supervision of the turning center's QC running in a high-level language environment, AMPLE. As a part of the post-process control loop, an automated inspection system using DMIS and the Initial Graphics Exchange Specification (IGES) national interface standards to integrate Computer Aided Design/Computer Aided Manufacturing (CAD/CAM) software with a CMM has become operational.

After completing the final integration of the real-time and the process-intermittent control loops for the turning center in the fiscal year 1991, the effort will be focused on the post-process control loop of the QIA architecture. The specific tasks of this effort are development of a feature-based Quality Database, development of a Quality Monitor, and integration of the Quality Database with the Quality Monitor. The Quality Database will be used to store all the information about the parts produced in the system based on their specific features. Such information will likely consist of: the errors found in these features, the machine axes, cutting tools, and cutting parameters used in producing these features, as well as other sensory information obtained during cutting. Examples of sensory information include temperature profiles, vibration signatures, and on-machine probing data. The Quality Monitor will be designed to sort this information stored in the Quality Database to identify the residual systematic errors in the process and generate corrective actions.

This program is funded by the U.S. Navy in support of the research at the Automated Manufacturing Research Facility (AMRF) at NIST, by private industry in the form of equipment and necessary software support, and by NIST.

2. CHARACTERIZATION OF THE VERTICAL MACHINING CENTER

F.F. Rudder, Jr.

2.1. INTRODUCTION

The two previous progress reports of the Quality In Automation (QIA) project described the initial phases of the pre-process machine tool characterization effort carried out for the vertical machining center [1,2]. The critical steps were the thermocouple installation on the machine structure; the development of the data acquisition system and related software; the development of the error measurement procedure for the x- and the y-axes; and the measurements. During the past year we have developed measurement procedures to characterize the geometric and thermal errors of the z-axis. Following these procedures, we have completed the acquisition of the z-axis error data. Initial statistical analyses for generating the geometric-thermal error (G-T) model of the machine were also carried out in the past year. As a part of machine characterization, errors related to exchanging tools and the effect of drift on updating tool-length offsets were measured.

2.2. MEASUREMENTS

As described in the earlier QIA reports, the primary objective of the machine characterization measurements is to acquire geometric error data at various thermal states of the machine tool representative of machining conditions. All the characterization measurements must be referenced to the initial thermal state of the machine tool.

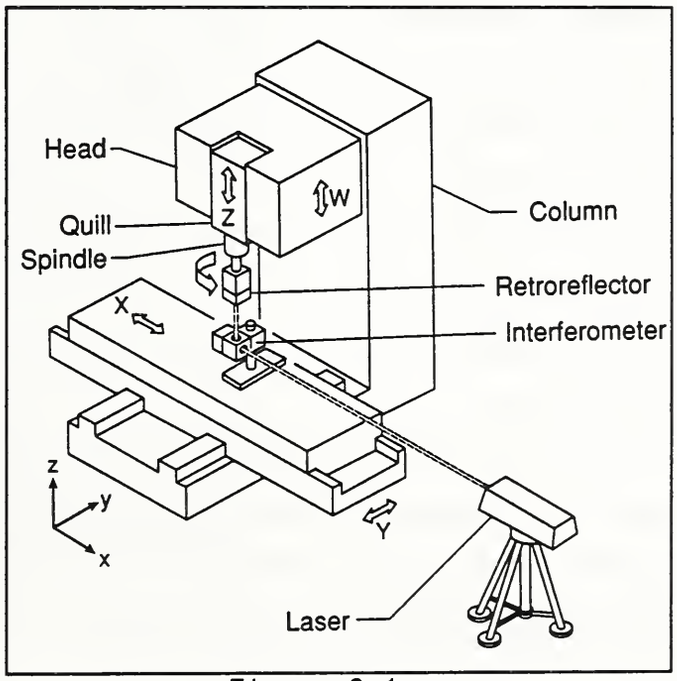


Figure 2.1

Figure 2.1 illustrates the axis configuration of the vertical machining center and identifies the major components. Since we

utilize laser interferometry as the primary metrology tool, we must maintain an uninterrupted path for the laser beam in order to reference all measurements to the initial "zero" position. Using spindle-mounted optics, this requirement is not too restrictive for the x-axis and the y-axis measurements. However, for the z-axis measurements, careful attention is required to obtain meaningful data as described in the next sections.

2.2.1. Machine Thermal State

Our previous measurements of the thermal state of the machining center identified high-speed spindle rotation as the most significant operation affecting the temperature distribution of the z-axis quill and the machine-tool head and column. For spindle-mounted optics, the laser beam cannot be maintained for any of the angular or straightness measurements. However, if the optics are properly aligned using a spindle-mounted corner cube, an unbroken laser path can be maintained for the z-axis linear-displacement error measurements. Hence, for the z-axis linear-displacement error measurements, we followed the procedure similar to the one we used in the x- and y-axes measurements. However, for

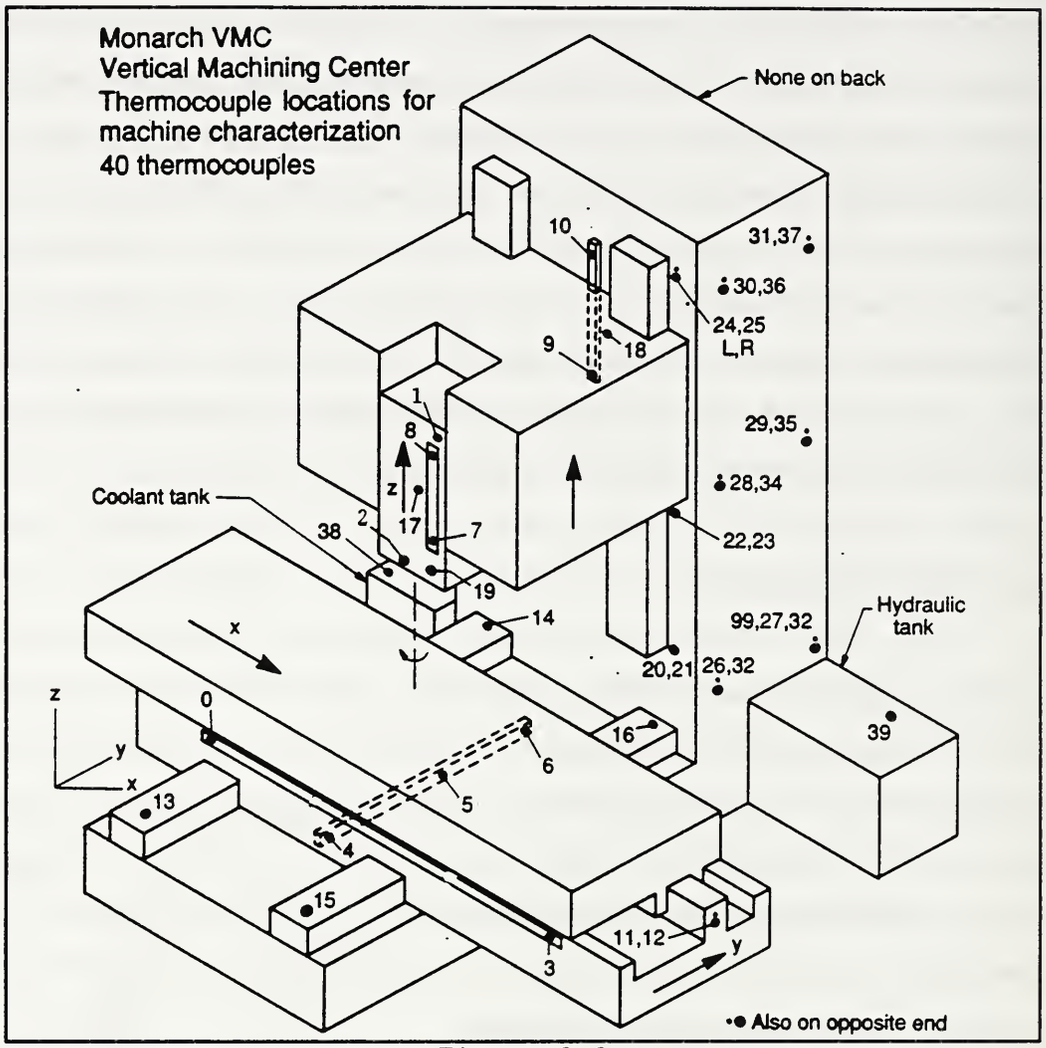


Figure 2.2

TABLE 2.1

Thermocouple Locations on Monarch VMC Machining Center

NO.	LOCATION
0	Left (Viewed from operator position) Side of X-(Glass) Scale
1	Upper Spindle Bearing on the Quill
2	Lower Spindle Bearing on the Quill
3	Right of X-Scale
4	Front of Y-Scale
5	Middle of Y-Scale
6	Rear of Y-Scale
7	Bottom of Z-Scale
8	Top of Z-Scale
9	Bottom of W-Scale
10	Top of W-Scale
11	Left End of X-Way
12	Right End of X-Way
13	Front of Left Y-Way
14	Rear of Left Y-Way
15	Front of Right Y-Way
16	Rear of Right Y-Way
17	Near Mount for Z-Scale Reader
18	Near Mount for W-Scale Reader
19	Near Coolant Nozzle
20	Outboard of Bottom of Left W-Way
21	Outboard of Bottom of Right W-Way
22	Outboard of Middle of Left W-Way
23	Outboard of Middle of Right W-Way
24	Outboard of Top of Left W-Way
25	Outboard of Top of Right W-Way
26	Bottom Front of Left Side of Column
27	Bottom Rear of Left Side of Column
28	Middle Front of Left Side of Column
29	Middle Rear of Left Side of Column
30	Top Front of Left Side of Column
31	Top Rear of Left Side of Column
32	Bottom Front of Right Side of Column
33	Bottom Rear of Right Side of Column
34	Middle Front of Right Side of Column
35	Middle Rear of Right Side of Column
36	Top Front of Right Side of Column
37	Top Rear of Right Side of Column
38	Coolant Tank
39	Hydraulic Tank

the angular and straightness measurements, we could only measure data for the "cold" machine state and for the "cool-down" machine state. Between these two states, we conducted a warm-up cycle of several hours' duration to reach the "hot" machine state. Since the warm-up cycle required high-speed spindle rotation, the laser optics were removed from the spindle during warm up. Hence, the laser zero was lost between data collected for the "cold" machine state and the "cool-down" machine state.

In the following discussion, we present temperature data identified by specific thermocouple locations. Figure 2.2 illustrates the general location of the thermocouples. Table 2.1 presents a description of each thermocouple location.

Figures 2.3 and 2.4 represent the output of sample thermocouples during the geometric error measurements for the z-axis. Figure 2.3 shows the thermal response during z-axis linear-displacement error measurements.

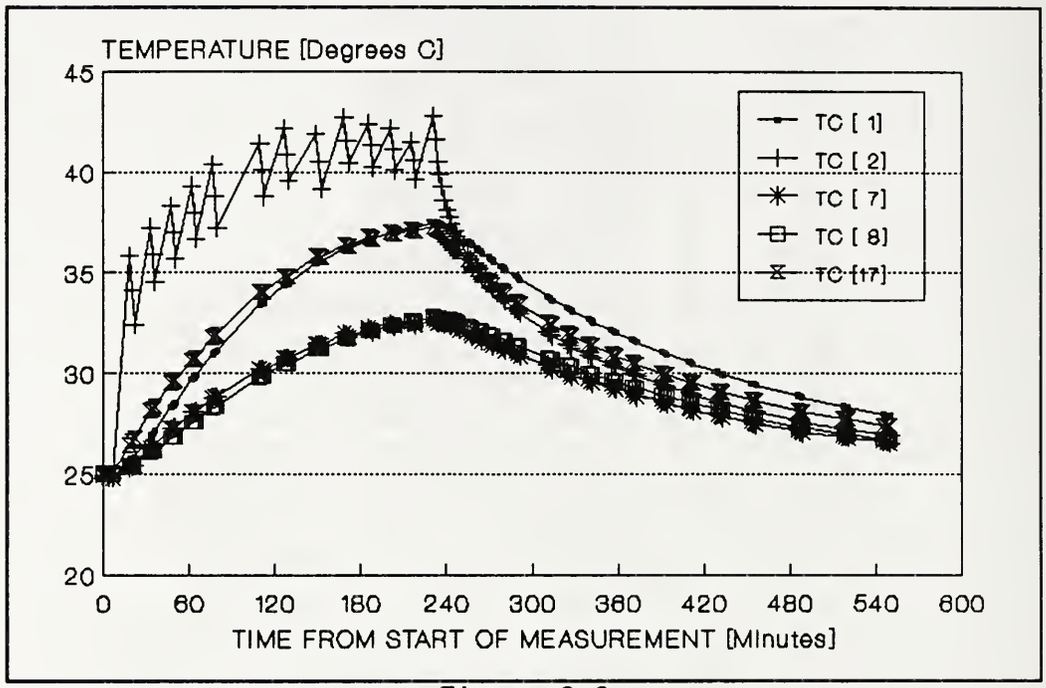


Figure 2.3

The "saw tooth" shape observed in quill temperature is caused by cool-down during the time spent for the laser measurements between the periods of warm-up.

Figure 2.4 presents data taken during the z-axis straightness measurements. The first few points represent the cold machine state. The straight-line portion of the graph represents the machine warm-up period during which the optics were removed from the spindle. The remainder of the graph corresponds to data measured during the machine cool-down period.

2.2.2. Geometric Error Data

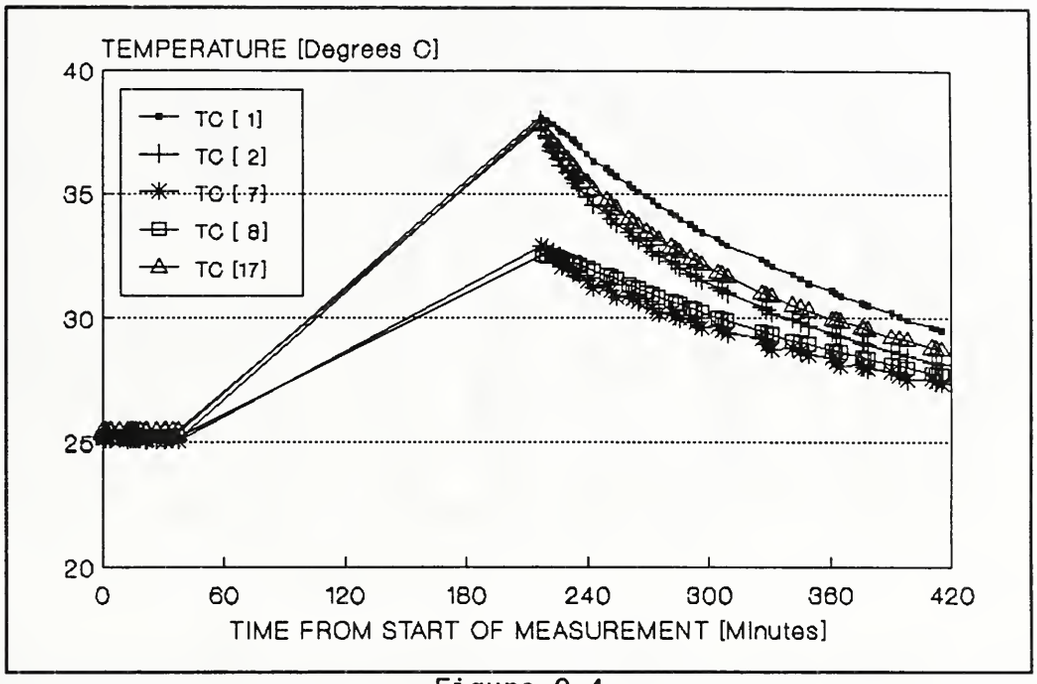


Figure 2.4

Figure 2.5 presents a summary of the geometric error data for the three axes of the machining center. This figure is presented to place the order of magnitude of each error in perspective. For these plots, the horizontal scale is the position of the respective machine tool axis. The vertical scale is the measured error. We use the fixed-range format in order to emphasize the relative magnitudes of the linear and the angular data. The glass scale temperature ranges of the respective axes are printed on each plot to indicate the similarity of machine-tool thermal state for all the measurements. The following conclusions may be reached by observing Figure 2.5:

- For all axes, linear-displacement errors are dominant and are temperature-dependent.
- Excepting the y-straightness of the x-axis, the magnitude of all other straightness errors are on the order of 10% or less than the magnitude of the linear-displacement errors.
- For the angular data, x-axis yaw and z-axis pitch exhibit the most variation.

The full size plots of errors correspond to the x- and the y-axes were given in the FY89 progress report [2]. The ones corresponding to the z-axis are included in Appendix A.

MACHINE TOOL CHARACTERIZATION

Temperature Effects

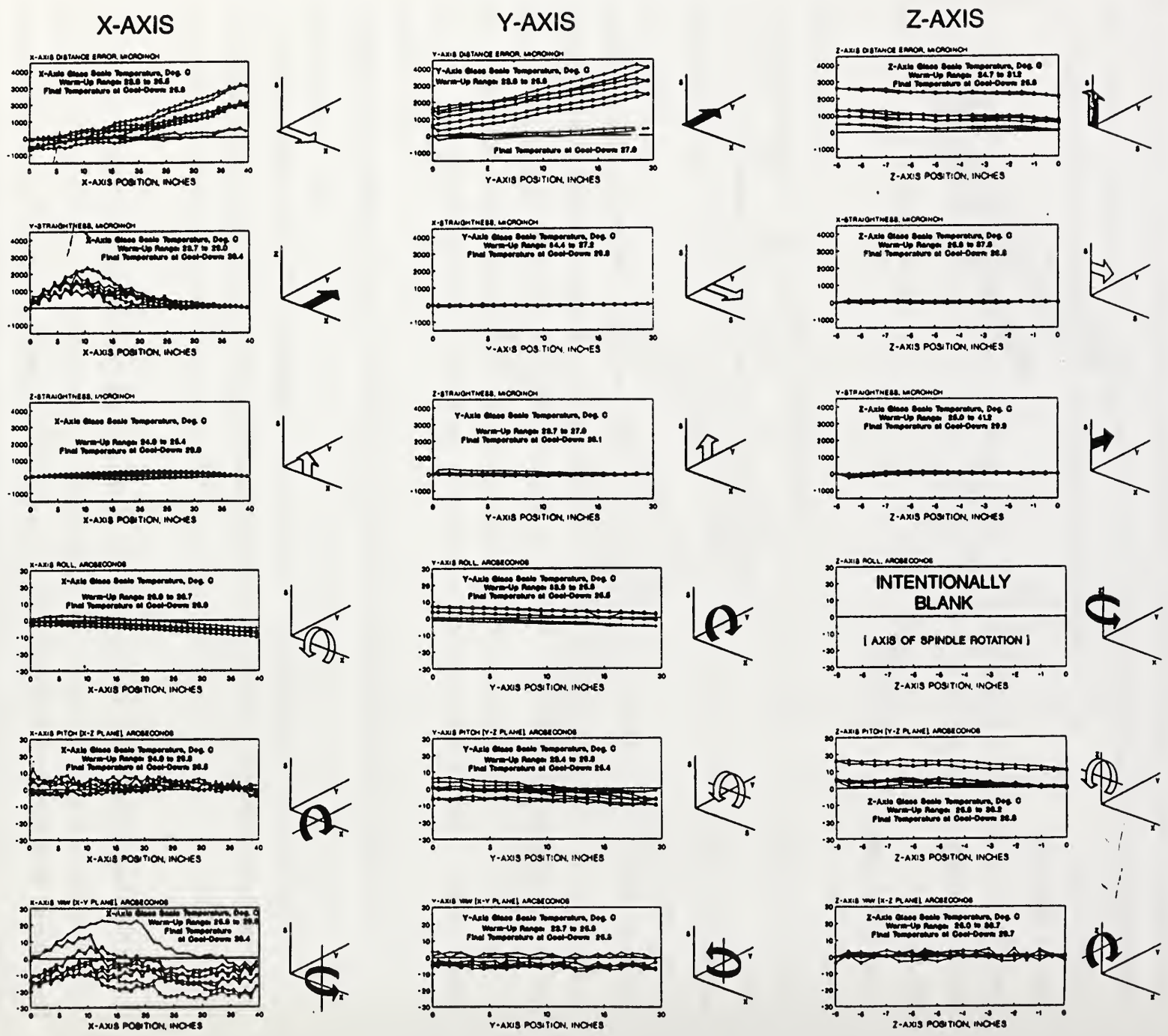


FIGURE 2.5

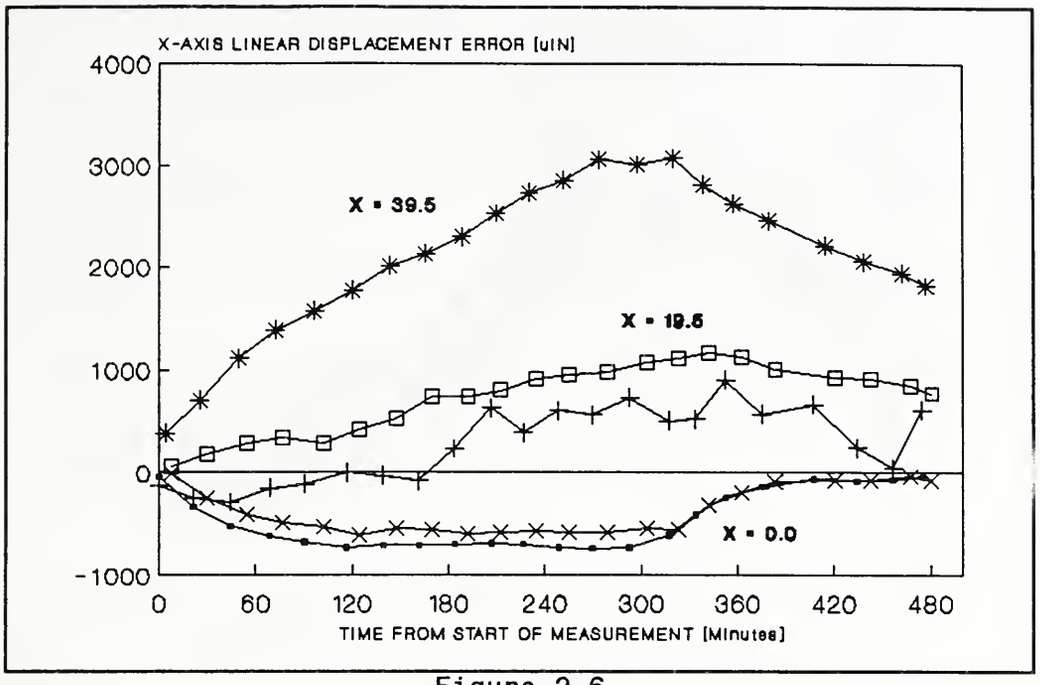


Figure 2.6

Figures 2.6 through 2.8 are plots of linear-displacement error for the x-, y-, and z-axes, respectively, versus the time. In each figure, four curves are presented: one curve for each axis's extreme position and two curves for a mid-axis position. It is seen that the error change is gradual for x- and y-axes warm-up and more pronounced for the cool-down state.

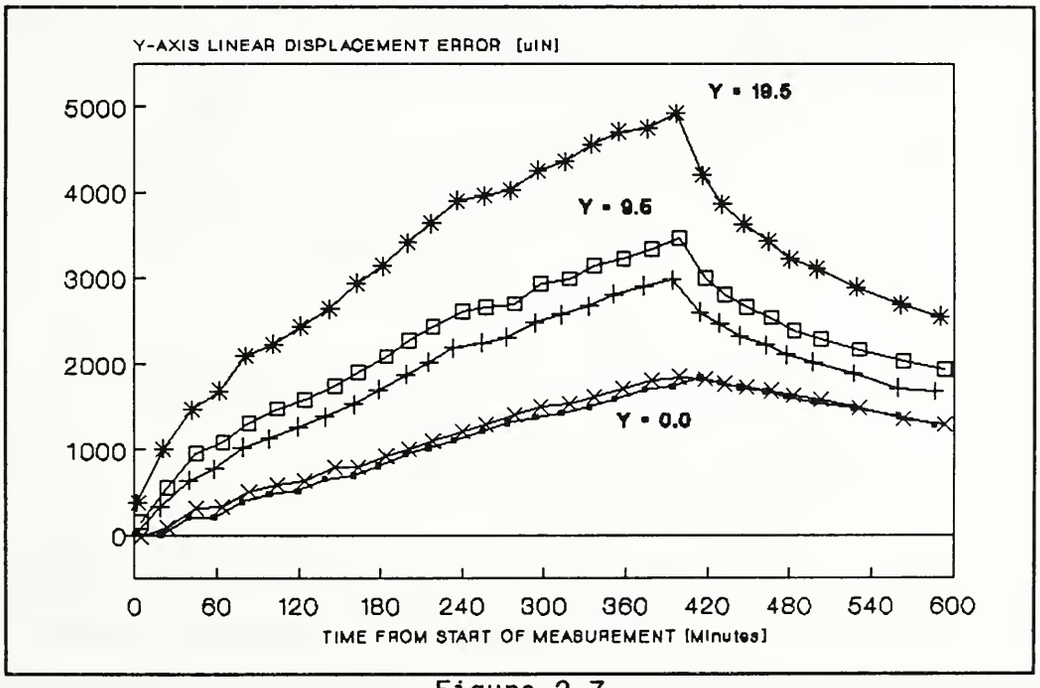


Figure 2.7

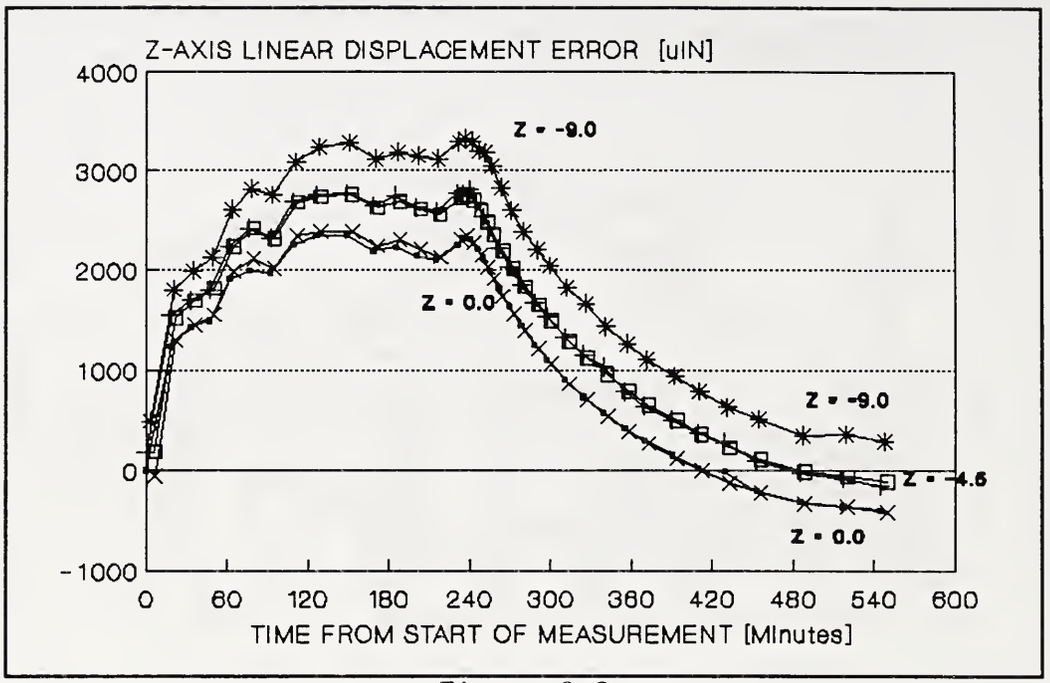


Figure 2.8

Figures 2.9 through 2.11 are plots of linear-displacement error for the x-, y-, and z-axes, respectively, versus the response of a selected thermocouple. Again, four curves are presented for each axis corresponding to the two extreme positions and two for a mid-axis position.

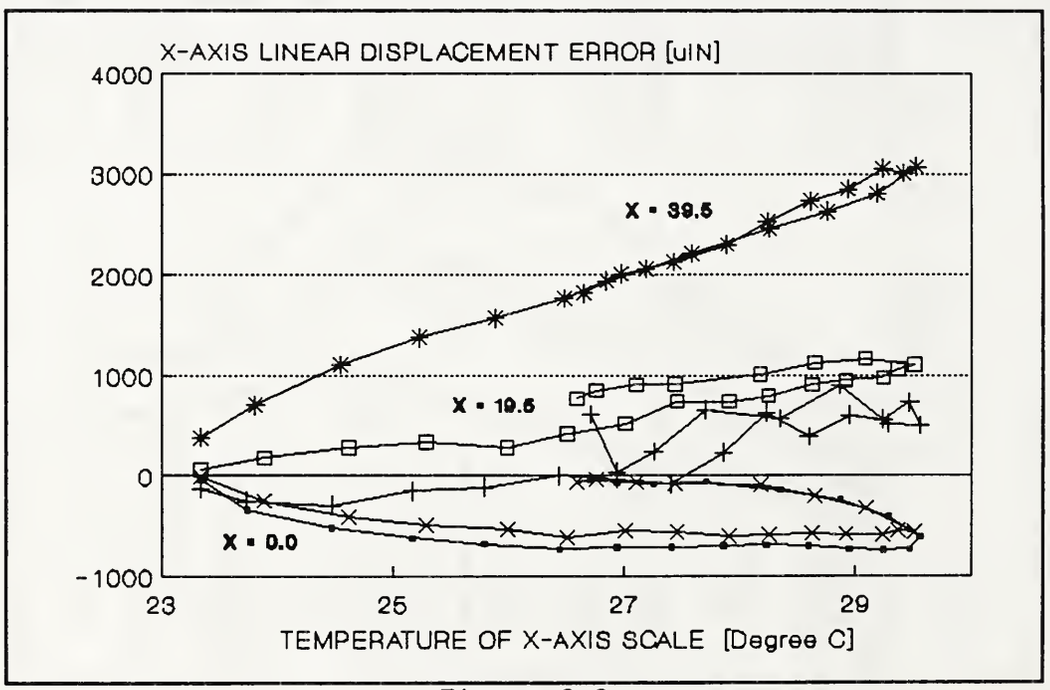


Figure 2.9

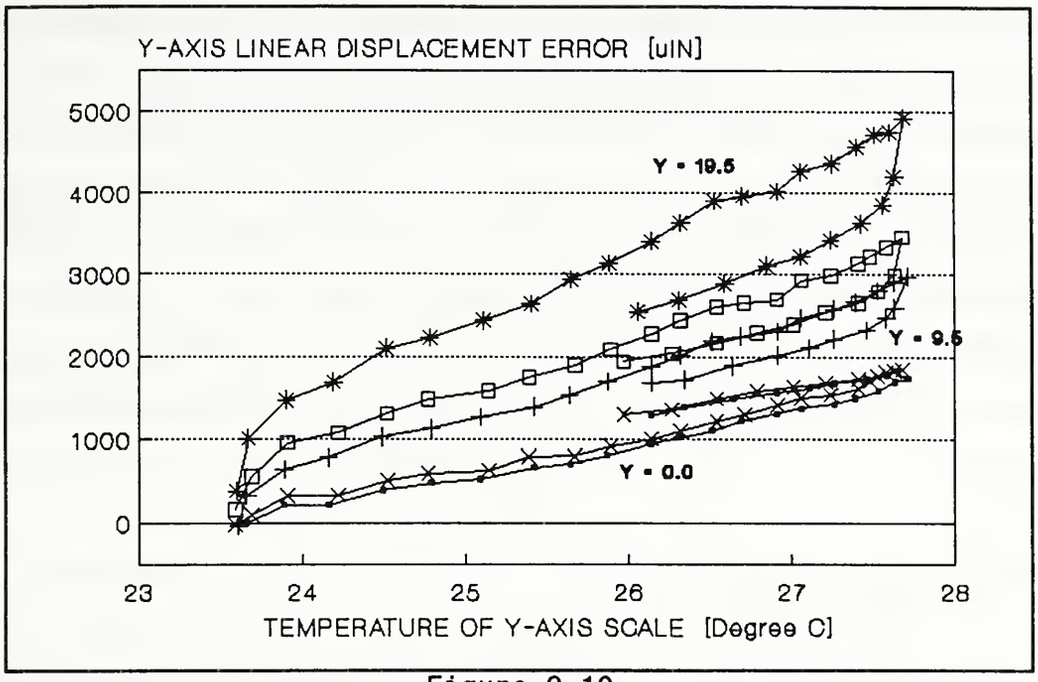


Figure 2.10

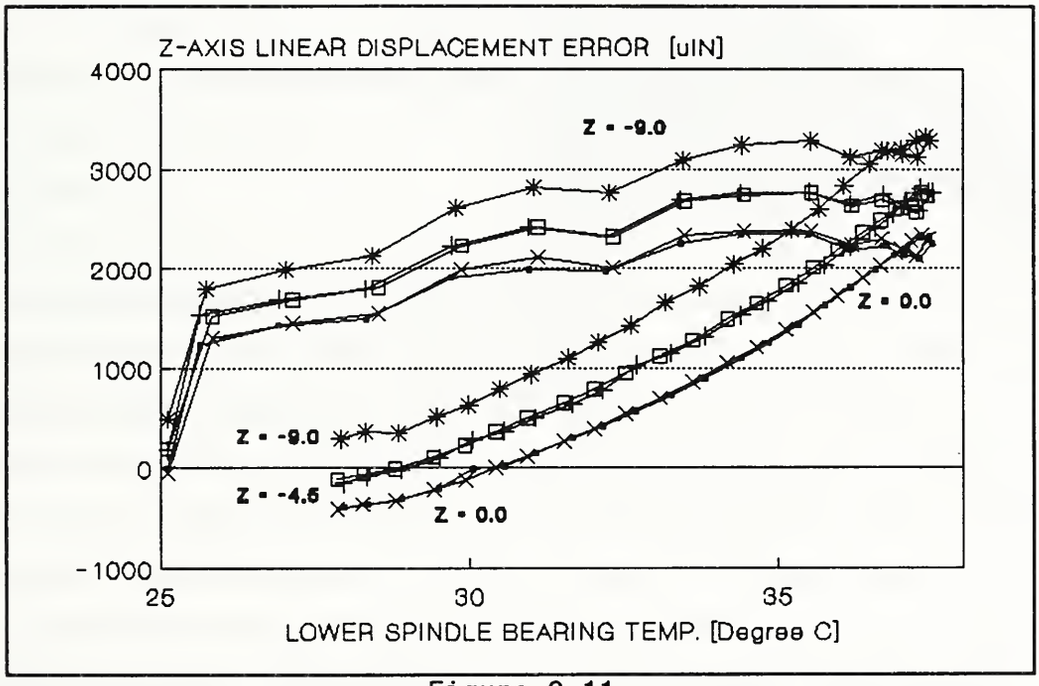


Figure 2.11

Figure 2.12 shows the difference in the error behavior during the warm-up and the cool-down cycles. This difference is an important consideration for any error compensation strategy that requires an interruption of the machining process. If the machine tool is idle for a period of time, the error behavior will follow a different pattern than the one during the warm-up cycle.

If the magnitude of the difference between the two modes of operation is significant relative to the desired precision of the machining process, then the compensation must include this effect. The possibility that the error may depend upon the rate of change of temperature as well as the magnitude of the temperature should be investigated through the statistical analyses.

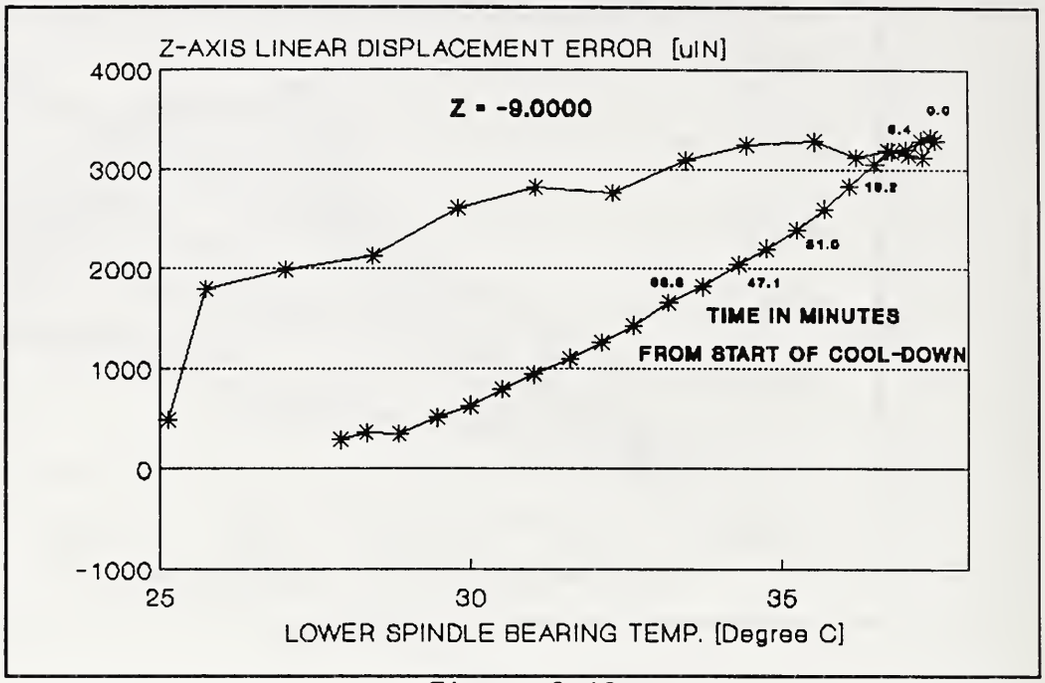


Figure 2.12

2.3. STATISTICAL DATA ANALYSIS

The objective here is to develop a geometric-thermal (G-T) model for the measured errors of the machining center. Using this empirical model, one determines the compensation based upon measurement of selected thermocouples and the nominal axes' positions during the machining process. To develop the G-T model two main steps must be taken. The first step is determining the thermocouples that are best correlated to the measured error. The second step is fitting a mathematical model to the data using the selected thermocouples. Our effort in developing the model has indicated that to identify significant thermocouples a consideration of the range of the temperature values as well as the direct correlation calculation are required.

In attempting to determine statistically significant correlations between a geometric error, axis position, and temperature, we have implemented the following analysis procedure. We first determine the best statistic relating the error to the axis position and then correlate this

statistic to the temperature data. As a result of this correlation we order the temperature locations by rank, based on the computed correlation coefficients. In correlating position error data to temperature data that spanned a narrow temperature range, the correlations may be very high although the result is not statistically significant for the particular error estimate. Therefore, we determine the temperature range of the data for the most highly correlated thermocouples and reject thermocouples that exhibit a high correlation but are associated with a narrow temperature range. As a final step, we conduct a multivariable regression between the error data, the axis position, and the temperature data that are highly correlated and exhibit the widest range of temperature.

Figure 2.13 illustrates a plot of rank-ordered thermocouples with the upper half of the vertical scale indicating the correlation coefficient and the lower half indicating the range of the temperature data. The number above each bar is the thermocouple ID number. Hence, we would consider thermocouple numbers 0, 3, 11, 12 and 4 to include in the data fit and reject thermocouple numbers 39, 18, 31, and 36. The data in Figure 2.13 are for x-axis linear-displacement error measurements and thermocouples 0 and 3 are at opposite ends of the x-axis glass scale and 11 and 12 are at opposite ends of the x-axis slideways. Thermocouple 4 is at the end of the y-axis glass scale nearest the x-axis scale. Thermocouples 39, 18, 31, and 36 are for locations on the machine-tool column and, in addition to the narrow temperature range, there is no physical reason to expect that such small temperature variations could result in significant position errors for this measurement.

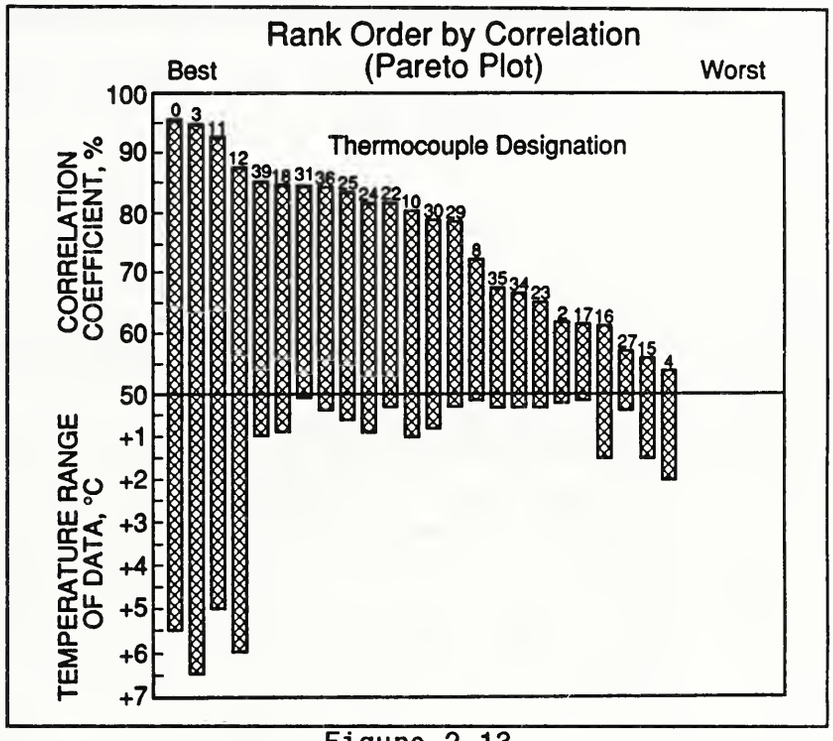


Figure 2.13

Based upon a multiple variable least-square data fit, the following preliminary model was established for the x-axis linear-displacement error[1]:

$$\epsilon(x,T_3,T_4) = A + Bx + CT_3 + DT_4 + Ex \bullet T_3 + Fx \bullet T_4, \ \mu m \quad \text{..... (2.1)}$$

where x is the x-axis nominal position

Range: $0.0 \leq x \leq 1003$ mm (39.5 inch)

T_3 is the temperature from Thermocouple 3

(end of x-axis glass scale)

Range: $23°C \leq T_3 \leq 30°C$

T_4 is the temperature from Thermocouple 4

(end of y-axis glass scale closest to x-axis scale)

Range: $23°C \leq T_4 \leq 26°C$

TABLE 2.2

Regression Results for the X Displacement Error Data

Parameter	Estimate	Std. Error	Units
A	-163.004	8.45	µm
B	-41.7385	15	µm/m
D	-5.21057	0.225	µm/°C
E	11.84595	0.5	µm/°C
E	16.1326	1	µm/m/°C
F	-13.604	1	µm/m/°C

[1]This analysis was conducted with the cooperation of NIST Statistical Engineering Division

2.4. COMPARISON OF PREDICTIONS TO MEASUREMENTS

In order to evaluate the ability of the above model to predict the x-axis linear-displacement error, we compare the output of the model to our measurements. The temperature values for thermocouples 3 and 4, as well as the nominal x position are used as input parameters to the model. The comparison is carried out for three sets of error data as follows:

1) The data set used to develop the above model. (These data were measured for a value of y= 254 mm (10 in) which is a mid-table location.)
2) A data set taken at a distinctly different ambient temperatures.
3) A data set of measurements taken at three different y-axis locations for the table.

First, we compare our model to the measurements used to develop the model. Figure 2.14 shows a plot of the temperatures measured by thermocouples 3 and 4 during the x-axis linear-displacement error measurements. In this figure, we have marked four sets of measurements corresponding to: beginning, middle, and end of the warm-up period as well as the middle of the cool-down period. Each set of three data points on this plot shows the beginning and end temperatures for individual measurement passes, as well as the average of these two temperatures.

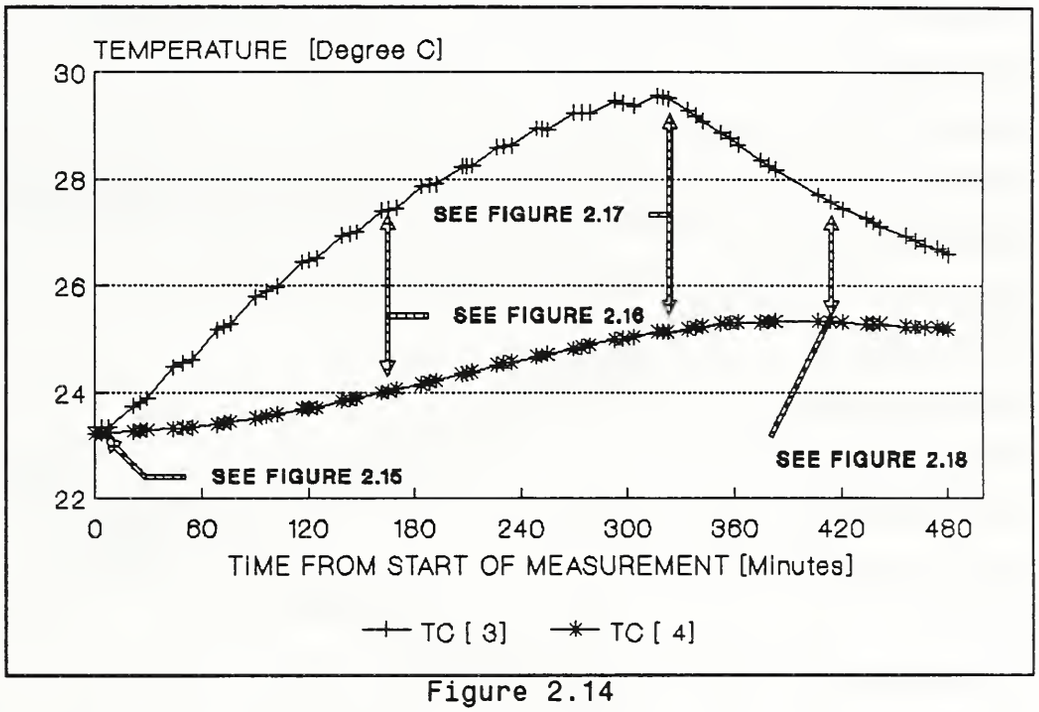


Figure 2.14

Figures 2.15 through 2.18 are plots of x-axis linear-displacement error versus x-axis position

for the above-mentioned temperature conditions. Each of these figures presents the measured error values with two values presented for each x-axis position. These two error values correspond to forward and reverse directions of x-axis motion. Two solid lines are plotted corresponding to those two directions. The solid lines are linear least-square curve fits obtained using only the data plotted in the figure. The dashed lines in these figures are derived from the above-presented model using the average temperatures of thermocouples 3 and 4 as logged at the start and the end of the measurement.

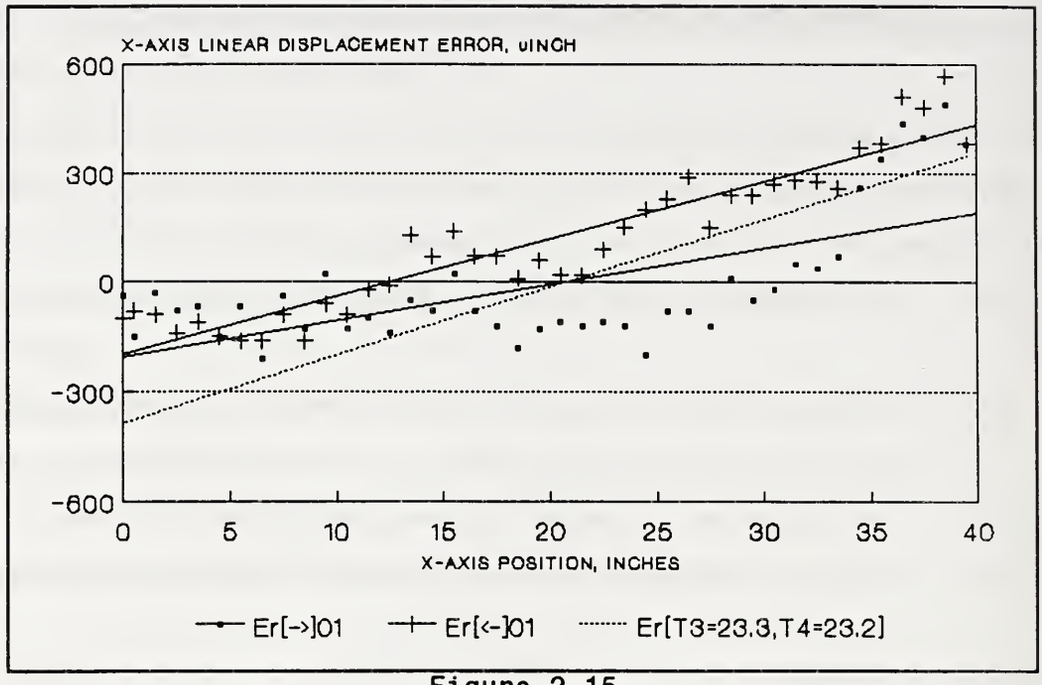


Figure 2.15

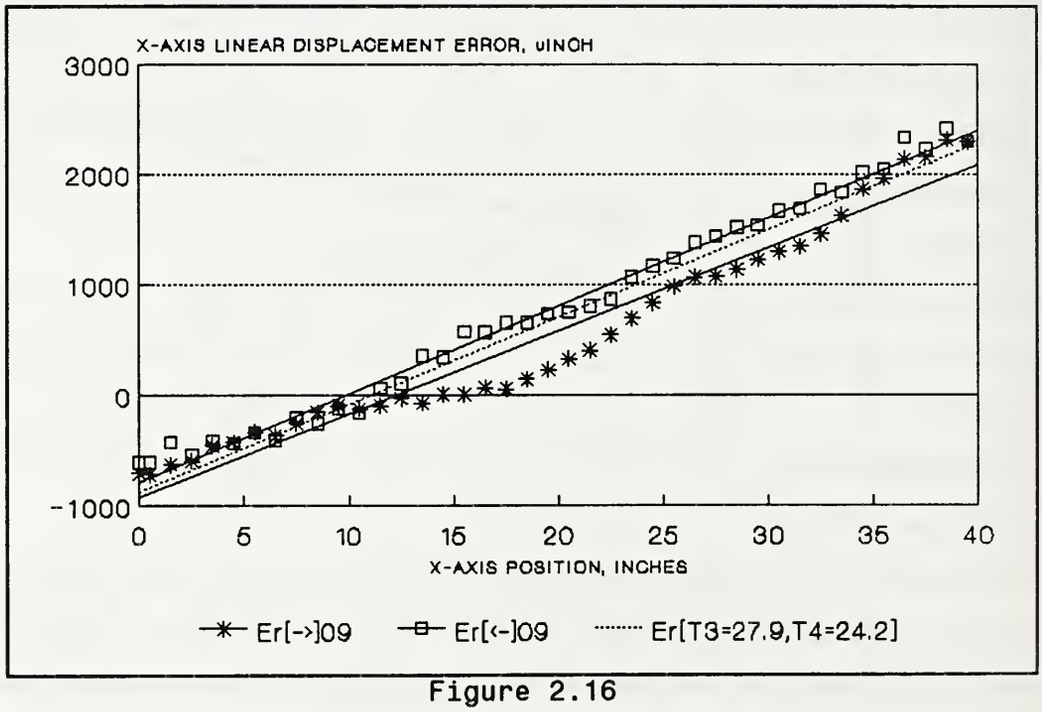


Figure 2.16

The model does not take the direction of motion into account, it averages the effects attributable to the backlash. This model is at least as good a predictor for all data as a linear

model based upon each individual data set.

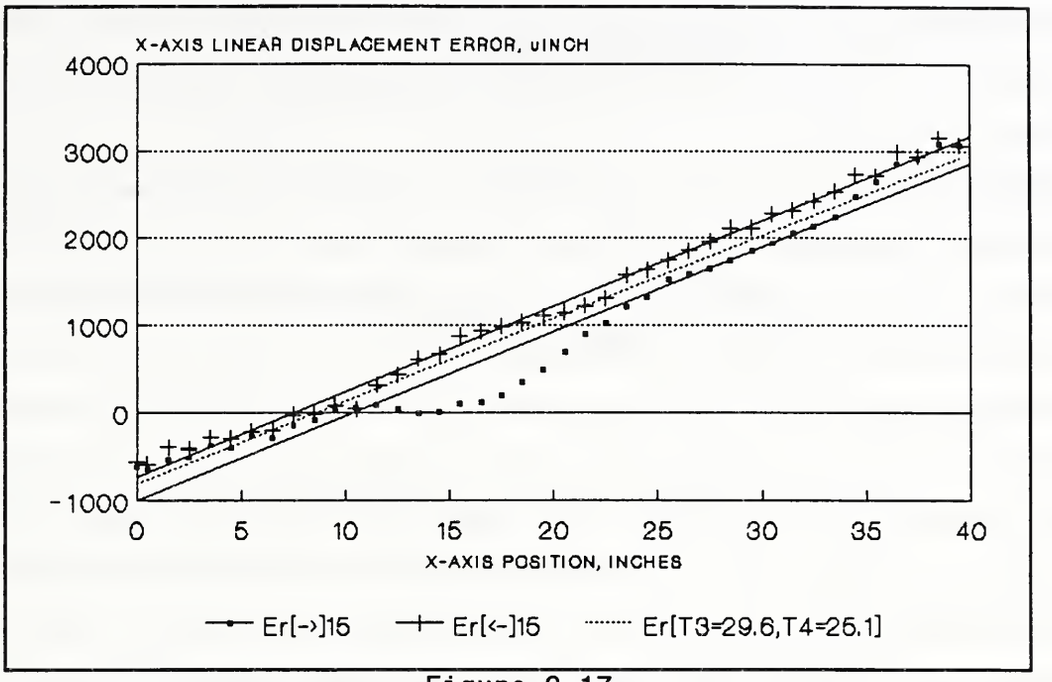


Figure 2.17

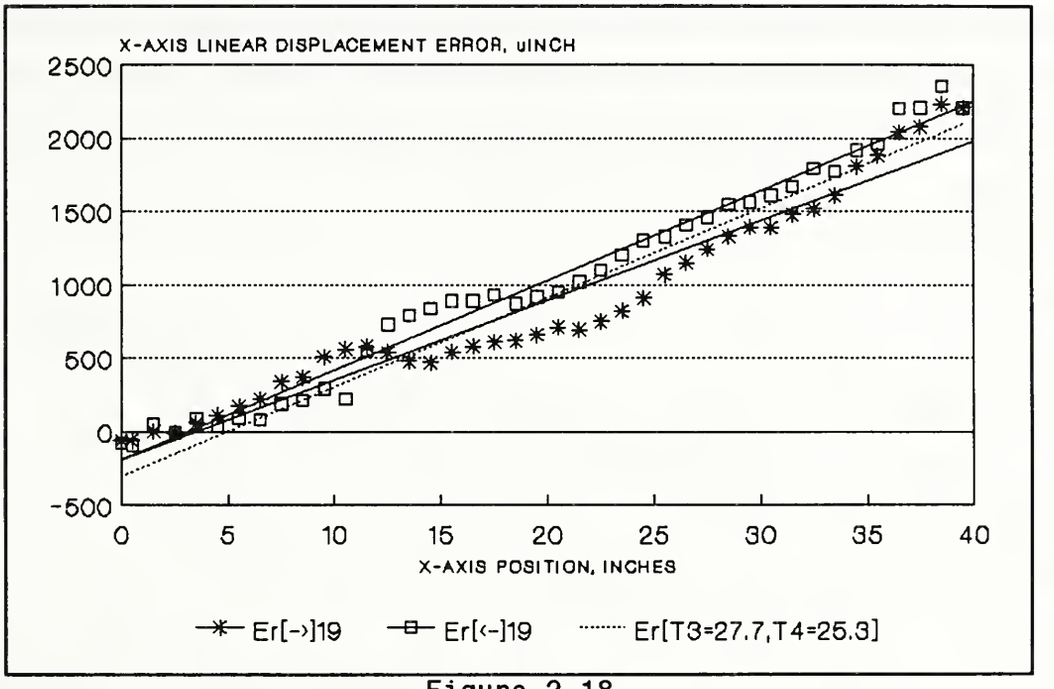


Figure 2.18

The above comparison was between the model and the data used to develop the model. We have also compared the model to a totally independent set of measurements. The measurements were conducted seven months subsequent to the data presented in Figures

2.15 through 2.18 and represent approximately a one degree Celsius increase in the ambient environment of the machine tool. Figure 2.19 presents the temperature response of thermocouples 3 and 4 during the measurements. Comparing this plot with Figure 2.14, we see that the curves in Figure 2.19 are initially one degree Celsius above the curves in Figure 2.14. Figure 2.20 is the plot of x-axis linear-displacement error versus x-axis position for the cold machine condition indicated in Figure 2.19 and we see that the model described in Equation 2.1 (dashed line) fits the measured data quite well. (Figure 2.20 corresponds to Figure 2.15.) Comparing the model to subsequent data sets within the measurements indicated in Figure 2.19 yields results similar to those presented in Figures 2.15 through 2.18.

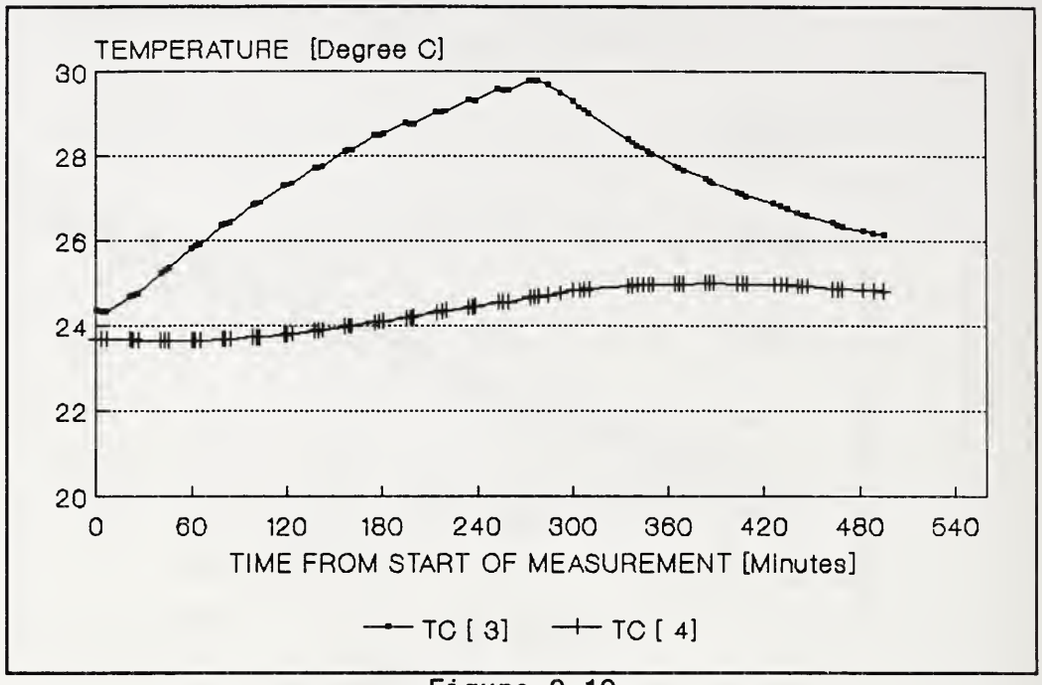


Figure 2.19

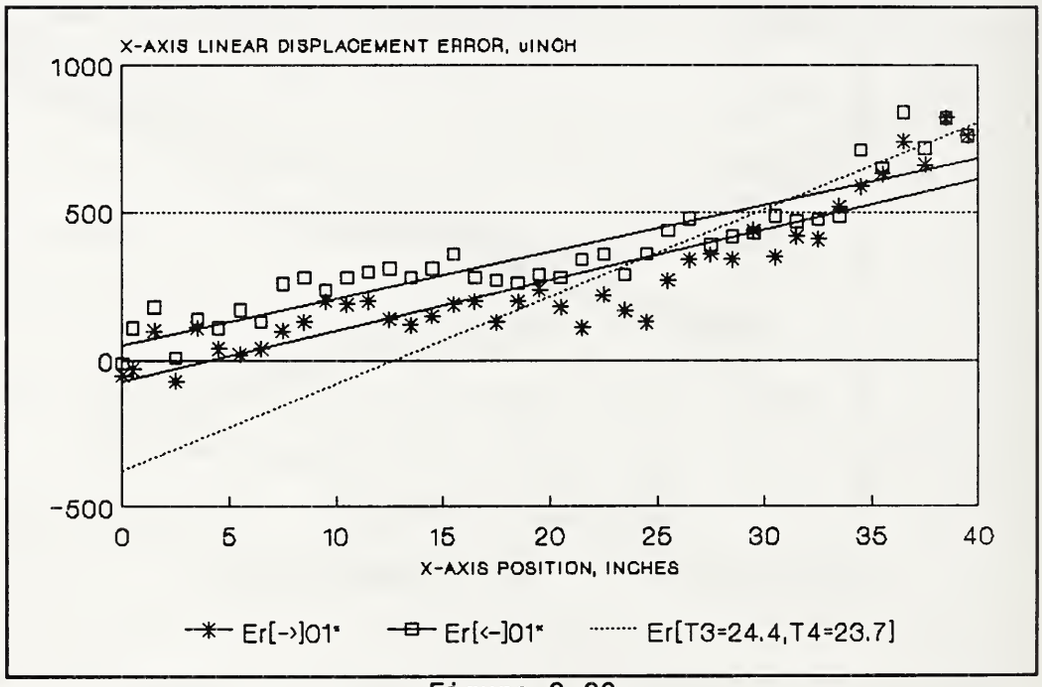


Figure 2.20

The conclusion is that the current model provides estimates of the position error under typical changes in the ambient environment within 25 μm (0.001 in). Future improvements in this model will take into account the direction of motion of the axes.

We now compare the model to data taken along the x-axis at three different table positions for the y-axis. This experiment comprised three sets of x-axis linear-displacement error data. Each set corresponds to a y-axis position of the machine tool table of y=127 mm (5 in), y=254 mm (10 in), and y=381 mm (15 in). These data were taken by independent measurements during machine warm-up and cool-down cycles using identical procedures. The model described above was developed using data obtained at y=254 mm (10 in) table position. It was used to compare with the data taken at different y-axis positions -- the form of the comparison is to calculate the difference between the model prediction and the measured values. This difference or residual may then be used to characterize the predictive ability of the model. One such characteristic is the median value of the difference between the prediction and the ten data points measured at different temperatures at each (x,y) location of the table. Figure 2.21 presents a three-dimensional plot of this calculation. Here we see that the median of the residuals for all (x,y) locations and temperatures is within a band of approximately -13 μm (-0.0005 in) to +25 μm (+0.0010 in).

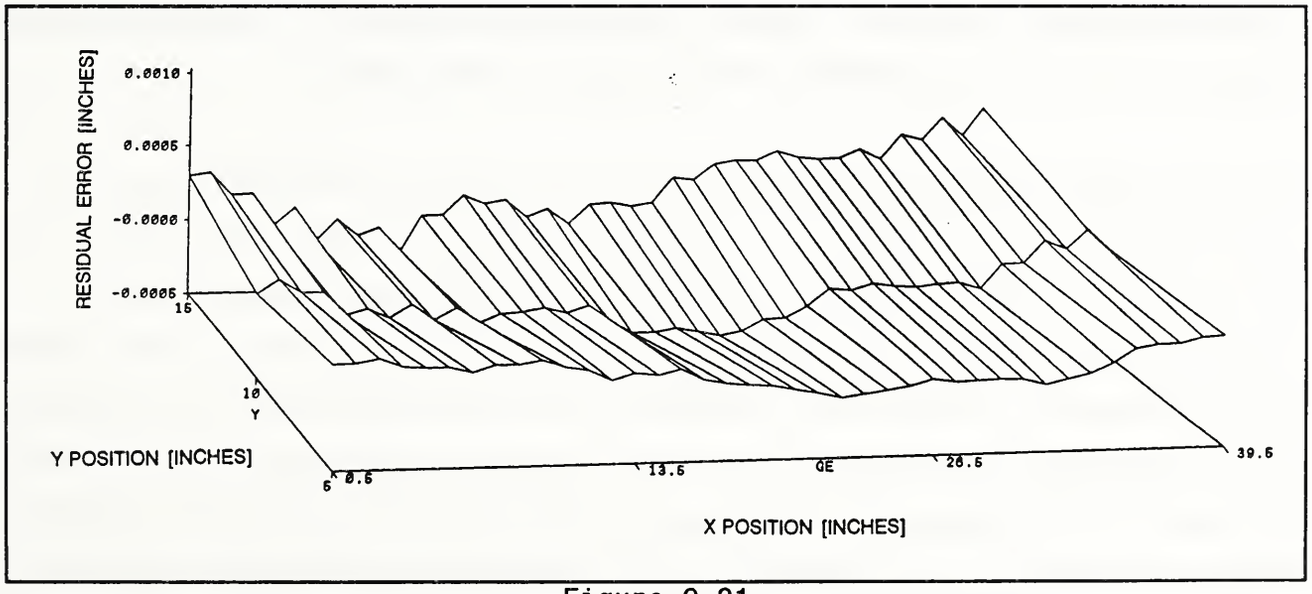


Figure 2.21

2.5. OTHER OPERATIONAL ERRORS

We have investigated the variation of tool-length offsets that may be due to nonthermal effects [3]. First, we compared manually measured master-gauge length with the magnitude of z-axis drift measured using laser interferometry. Since the tool-length offset is measured using a read-out of the z-axis position, we should observe a change in the master-gauge tool length comparable to the z-axis drift during a warm-up cycle. As illustrated in Figure 2.12, the z-axis linear-displacement error changes almost 25 μm (1000 μin) during the first 30 minutes of machine cool-down. Following a two hour warm-up with the spindle rotating at 3500 rpm, we measured the master-gauge tool length 32 times during a 25 minute period. The master gauge remained in the spindle during warm-up and during the measurements. The measured gauge length varied approximately 30 μm (1300 μin).

Next, we used the tool-length offset program resident in the machine tool controller to observe the variation of tool-length offset of an end mill. This resident program is machine-specific and utilizes a table-mounted tool setting station. The length offset of the end mill is relative to the master gauge length which is also measured each time the program is executed. However, the master gauge and the end mill are mechanically exchanged between a tool carousel and the spindle. Hence, our observations include any variation attributable to both thermal drift and repeatability of the tool adaptor in the spindle.

We conducted both master-gauge measurements and end-mill measurements for three thermal states of the machining center: cold machine (idle over night), warm machine (two hour high-speed spindle warm-up), and a cool machine (90 minutes following the warm machine measurements). From these measurements, we concluded that the mean value of the master-gauge tool length was not different for the cold machine and the cool machine, but was different for the warm machine. For the end-mill tool-length measurements (which are relative to the master-gauge length), we could not establish any statistically significant difference between the mean values of the end mill-length for the three different thermal states of the machining center. However, the standard deviation of the measured end mill lengths was varied from 14 μm (550 μin) to 19 μm (750 μin) and most likely was caused by

the tool exchange operation.

In summary, we observe similar variation of the z-axis position both with the laser measurements and the tool-length measurements. This variation represents a shift in the mean value and may be compensated by an empirical error model or, possibly, in-process gauging. The variation that may be attributable to tool exchange appears to be on the order of 14 μm to 19 μm. This variation represents a random error component that may be realized during machining operations that cannot be corrected solely by introducing z-axis compensation for thermal drift.

2.6. FUTURE WORK

The measurement data and related records described above have been archived. Reports are in preparation describing the methodology used, the data obtained, and the analysis results achieved for characterization of the vertical machining center. These reports will be published separately during this reporting period.

3. IMPLEMENTING REAL-TIME CONTROL FOR TURNING CENTER

M.A. Donmez, K.W. Yee, D.H. Neumann, and L. Greenspan

3.1. INTRODUCTION

This chapter presents the work done during the past year on the turning center to implement the real-time control loop. The two main functions of the real-time control loop are 1) to monitor both the machine tool and the cutting process during the metal removal operation, and 2) to predict and compensate for the machine tool systematic errors. A critical task for the prediction of the systematic errors is to build the geometric and thermal (G-T) error model of the machine. The second task is to modify the cutting-tool's path, during the actual machining, based on the error prediction from the evaluation of the G-T model for the current tool position and temperature profile of the machine. This task will be implemented by the Real-Time Error Corrector (RTEC) as described in the FY89 QIA Progress Report [2].

In the following sections, we describe the final stage of the machine characterization effort, which is the analysis of the raw geometric and thermal data to build the G-T model. Then, we describe the software module being developed for the Quality Controller (QC) of the turning center. The purpose of this module is to combine the G-T model and the machine kinematic model in order to implement the error prediction in real time. Finally, we describe the tests designed and carried out during the year for the evaluation of the RTEC's real-time tool-path modification performance.

3.2. MACHINE CHARACTERIZATION

3.2.1. Overview

We continued the characterization of the turning center's quasistatic and thermally induced geometric errors. As a parallel effort to the work being done for the vertical machining center characterization, a statistical data analysis procedure was developed for the turning center and used to establish error functions with respect to machine positions and the temperature profile. Our premise is that the error functions are continuous, and the error values are slowly

changing with respect to positions and temperatures. In addition, from a previous investigation we found that the cause and effect relationship between structure temperature and the resultant error is relatively simple [4].

3.2.2. Data Analysis

The machine-characterization raw data consists of a series of error values taken at each measuring interval while a particular machine slide was moved back and forth along its axis of motion. The temperatures at 36 locations around the machine structure were taken before and after each bidirectional run.

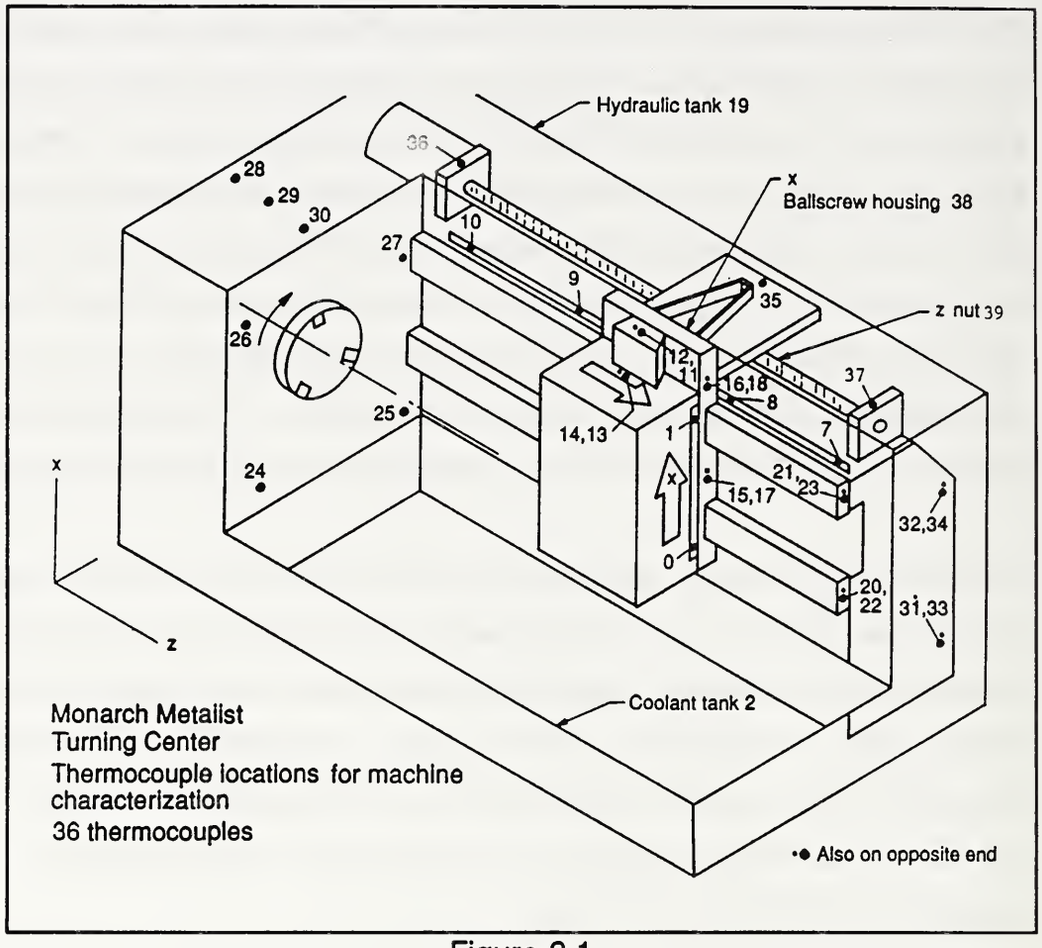


Figure 3.1

Figure 3.1 shows the temperature measurement locations on the machine. The descriptions of these locations are given in Table 3.1. The procedure for the data acquisition used for the machine characterization was explained in detail in the Progress Report of the Quality In Automation Project for FY89 [2]. A sample plot of the raw data obtained from the z displacement error measurements is shown in Figure 3.2. The apparent nonrepeatability of the data is due to the changes in the temperature profile of the machine.

TABLE 3.1

Thermocouple Locations on Monarch Metalist Turning Center

NO.	LOCATION
0	Bottom of X-(Glass) Scale
1	Top of X-Scale
2	Coolant Tank
3	Ambient
4	Not Used
5	Not Used
6	Top Right of Bed
7	Right of Z-Scale
8	Right Center of Z-Scale
9	Left Center of Z-Scale
10	Left of Z-Scale
11	Top of X-Way
12	Bottom of X-Way
13	Top of X-Head
14	Bottom of X-Head
15	Bottom of Z-Slide
16	Top Left of Z-Slide
17	Bottom Right of Z-Slide
18	Top Right of Z-Slide
19	Hydraulic Tank
20	Left End of Lower Z-Way
21	Left End of Upper Z-Way
22	Right End of Lower Z-Way
23	Right End of Upper Z-Way
24	Lower Front of Spindle Head
25	Lower Rear of Spindle Head
26	Upper Front of Spindle Head
27	Upper Rear of Spindle Head
28	Left of Top of Spindle Head
29	Middle of Top of Spindle Head
30	Right of Top of Spindle Head
31	Bottom Left of Bed
32	Top Left of Bed
33	Bottom Right of Bed
34	Not Used
35	Near X-Drive Motor Shaft Bearing
36	Left Z-Ballscrew Bearing
37	Right Z-Ballscrew Bearing
38	X-Ballscrew Housing
39	Z-Ballscrew Nut

We started the data analysis by normalizing the error values at each nominal position with respect to the measured error at the starting position of each bidirectional run. Figure 3.3 shows the normalized version of the same data shown in Figure 3.2. Since the error data is direction sensitive due to backlash and other reversal errors, we separated the data directionwise into two groups. Figures 3.4 and 3.5 show two sets of curves corresponding to the forward and the reverse directions respectively.

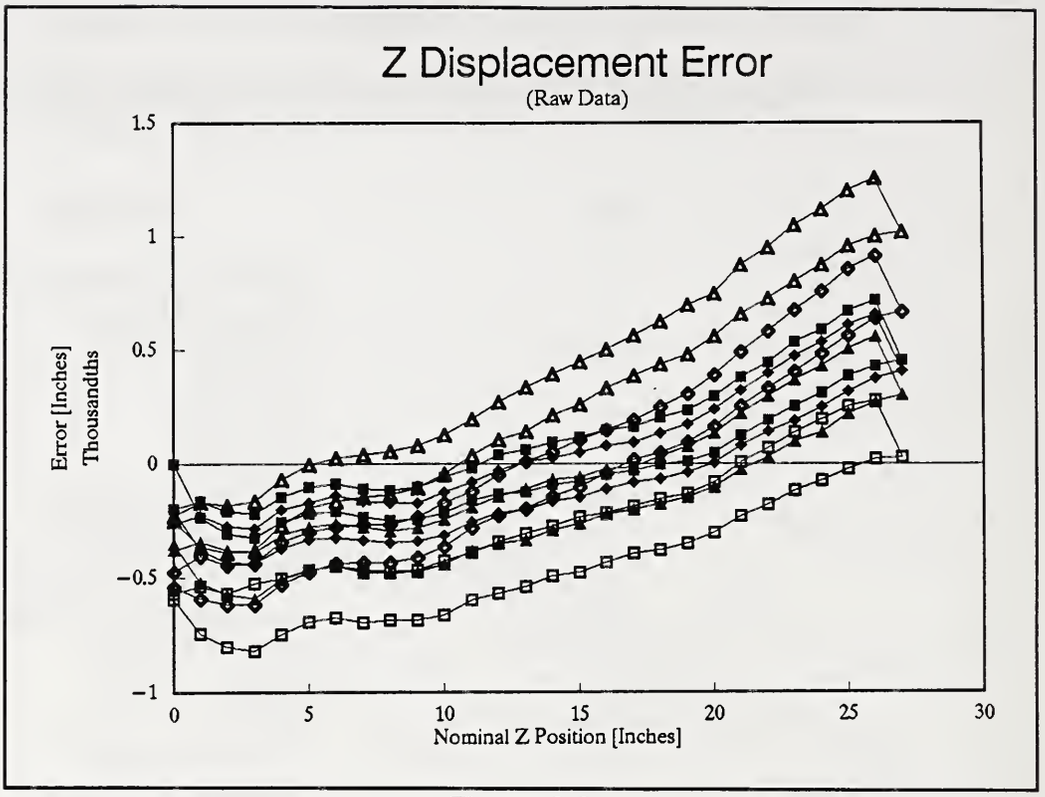


Figure 3.2

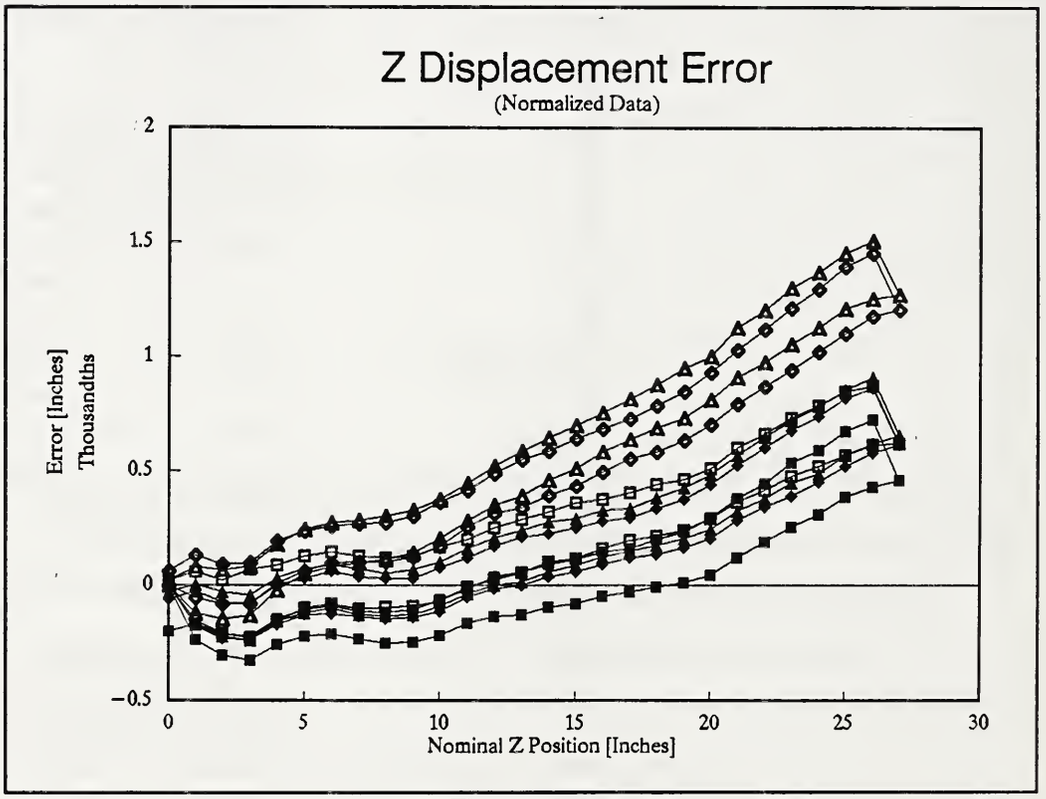


Figure 3.3

For each nominal position, we had one set of error data for the forward direction and one set for the reverse direction. By interpolating the temperature values before and after each bidirectional run, we calculated temperature values corresponding to each nominal position. Then, for each nominal position we carried out a linear fit of error with respect to temperature at each temperature measurement location. We then ordered these locations with respect to the goodness of the fit in order to select the best temperature

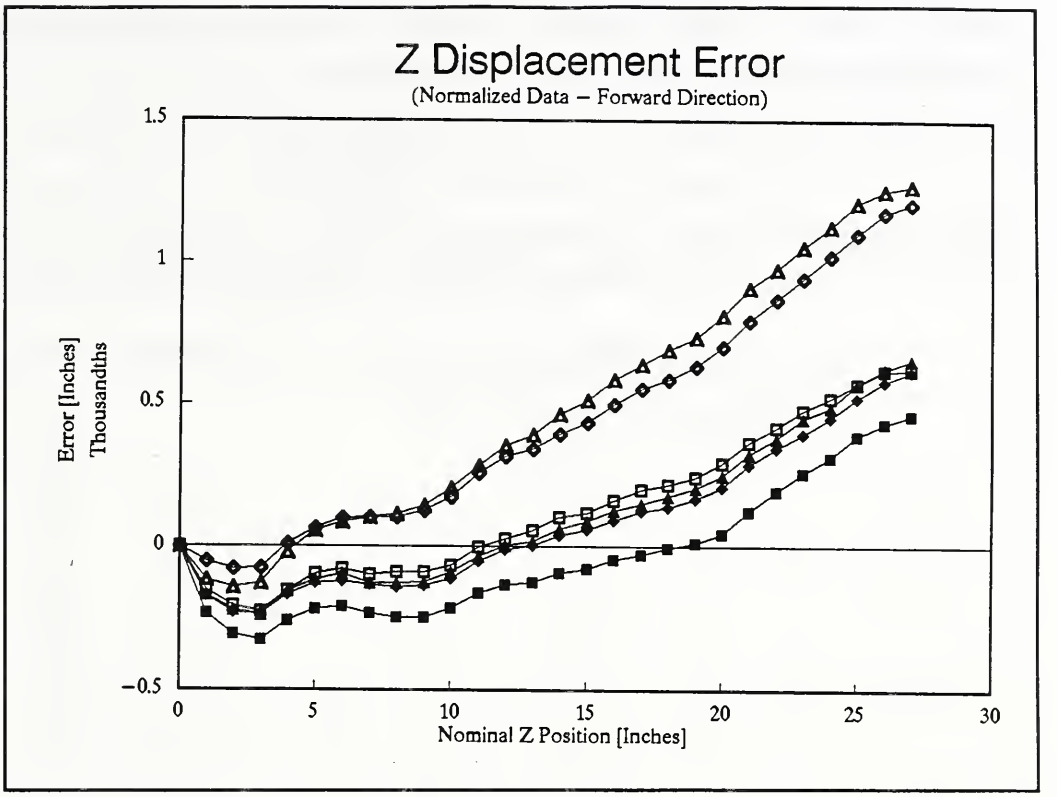


Figure 3.4

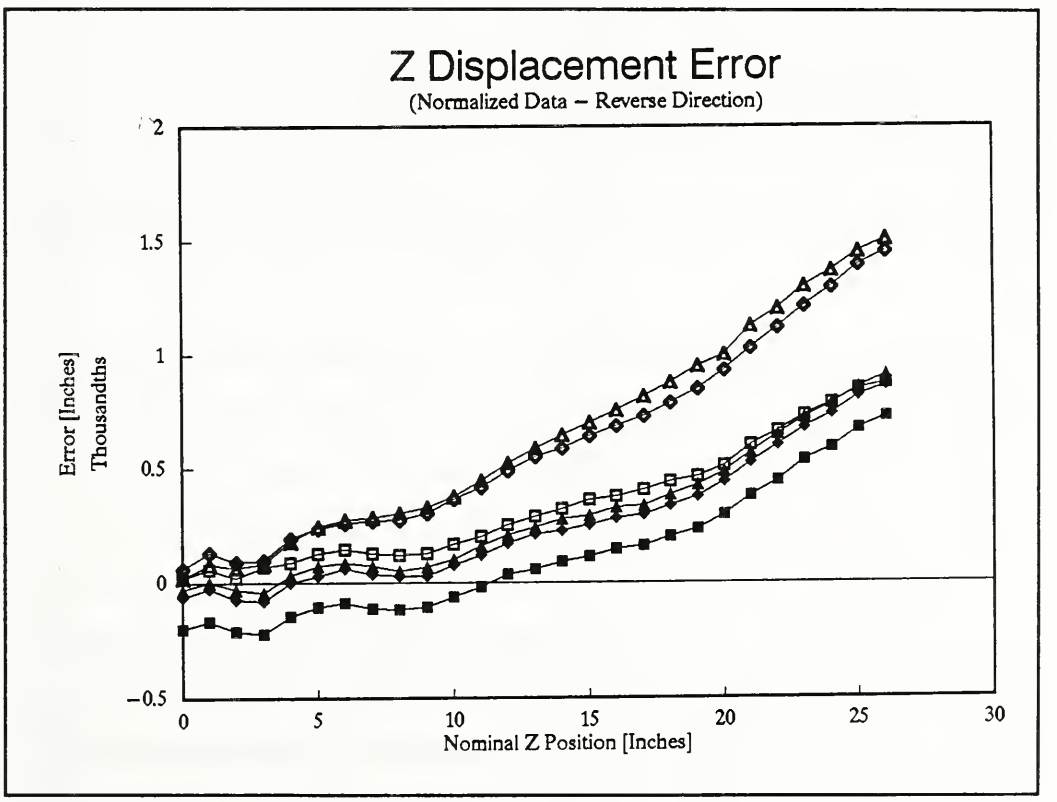


Figure 3.5

locations to be monitored for error prediction. Table 3.2 shows a sample result from such a fit carried out for the nominal position of 356 mm

TABLE 3.2

Linear Regression Fits of Z Displacement Error as a Function of Different Temperatures (at position Z=356 mm; forward direction of motion)

THERMOCOUPLE NO.	STANDARD DEVIATION (μm)
12	0.86
15	0.86
14	0.89
17	0.91
22	0.95
33	0.95
38	0.97
11	0.99
18	0.99
16	1.0
35	1.1
1	1.1
0	1.1
20	1.2
32	1.3
31	1.33
25	1.47
29	1.48
30	1.49
28	1.52
6	1.59
26	1.62
23	1.66
19	1.69
3	1.70
24	1.71
13	1.80
27	1.82
8	1.86
9	1.90
21	2.12
7	2.13
10	2.28
2	2.56
39	2.73
36	3.37
37	3.73

(14 in) along the z axis for the forward direction of motion. Figure 3.1 shows the approximate locations of the thermocouple channels listed in Table 3.1. A sample plot showing the temperature dependency of the displacement error at three different nominal positions (approx. 250 mm, 350 mm, and 460 mm) is presented in Figure 3.6.

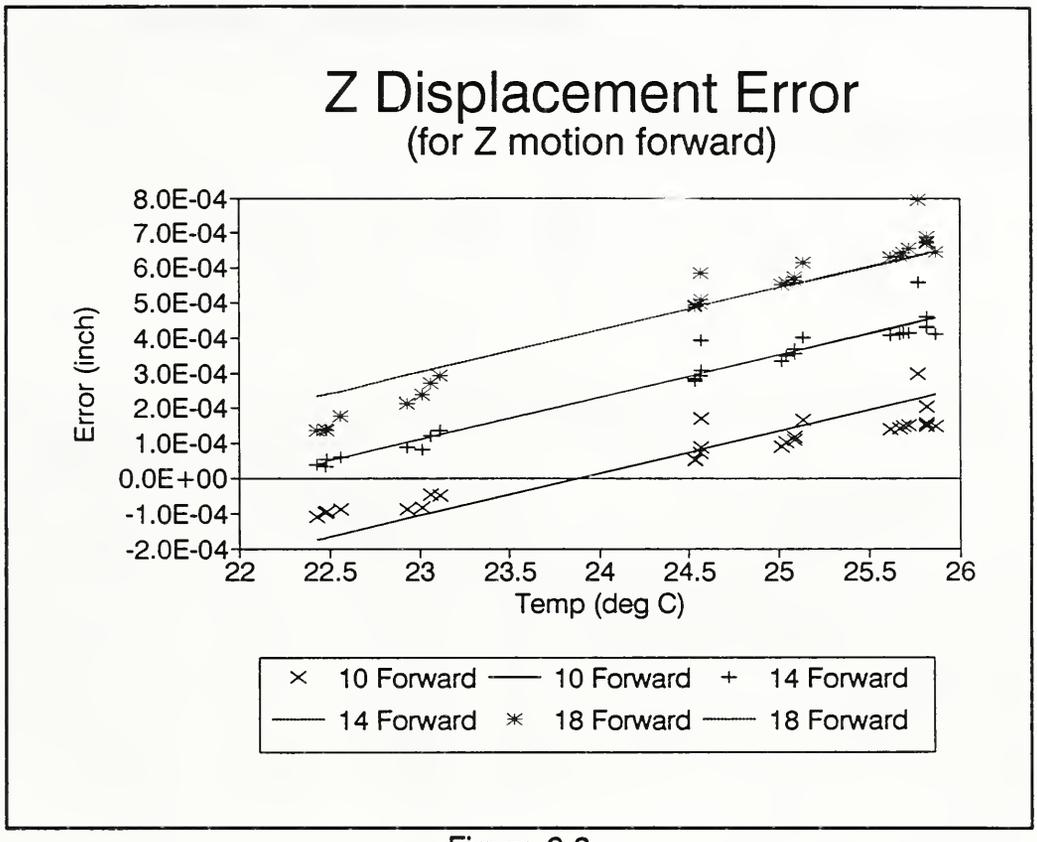


Figure 3.6

By looking at the generic error versus nominal-position function, we divided the data corresponding to the whole travel range into sections with similar characteristics. In the case of data shown in Figure 3.4 for example, we divided the data into three groups, which overlaps slightly. The first group of data corresponds to the first 250 mm (10 in) of travel. The second group of data corresponds to the 250 mm (10 in) of travel in the middle section of the travel range. The third group consists of the data corresponding to the last 230 mm (9 in) of travel. The data corresponding to each section are then curve-fitted with respect to nominal position and the temperature of the best representative location selected at the previous step. A sample result from these statistical analyses is shown in Table 3.3. The error surface generated by the fitting in this example has a linear relationship with respect to the temperature and a quadratic relationship with respect to the nominal axis position. Figure 3.7 shows a sample error surface generated by the model for the z displacement error of the turning center.

TABLE 3.3

Results of the Statistical Analysis of Z Displacement Error Data

(Forward Direction of Motion)

General form of the equation:

$$\epsilon(z,T_{12}) = A + BT_{12} + Cz + Dz^2 \quad \mu m$$

Regression Parameters		Position Segments		
		25 - 250 mm (1 - 10 in)	225 - 475 mm (9 - 19 in)	450 - 680 mm (18 - 27 in)
Std Error of Y Estimate (μm)		1.39	1.26	1.26
R Squared		0.77	0.95	0.98
Number of Observations		280	308	280
Degrees of Freedom		276	304	276
Constant Term (A) (μm)		-3.53	-8.863	-149.127
Coefficients	Temp.(B) (μm/C)	1.212	3.029	4.952
	Position (C) (μm/m)	32.703	73.05	96.356
	(Position)^2 (D) (μm/m^2)	-14.961	-30.58	-19.92
Max Positive Deviation (μm)		3.025	2.937	3.916
Max Negative Deviation (μm)		-4.442	-4.027	-3.797

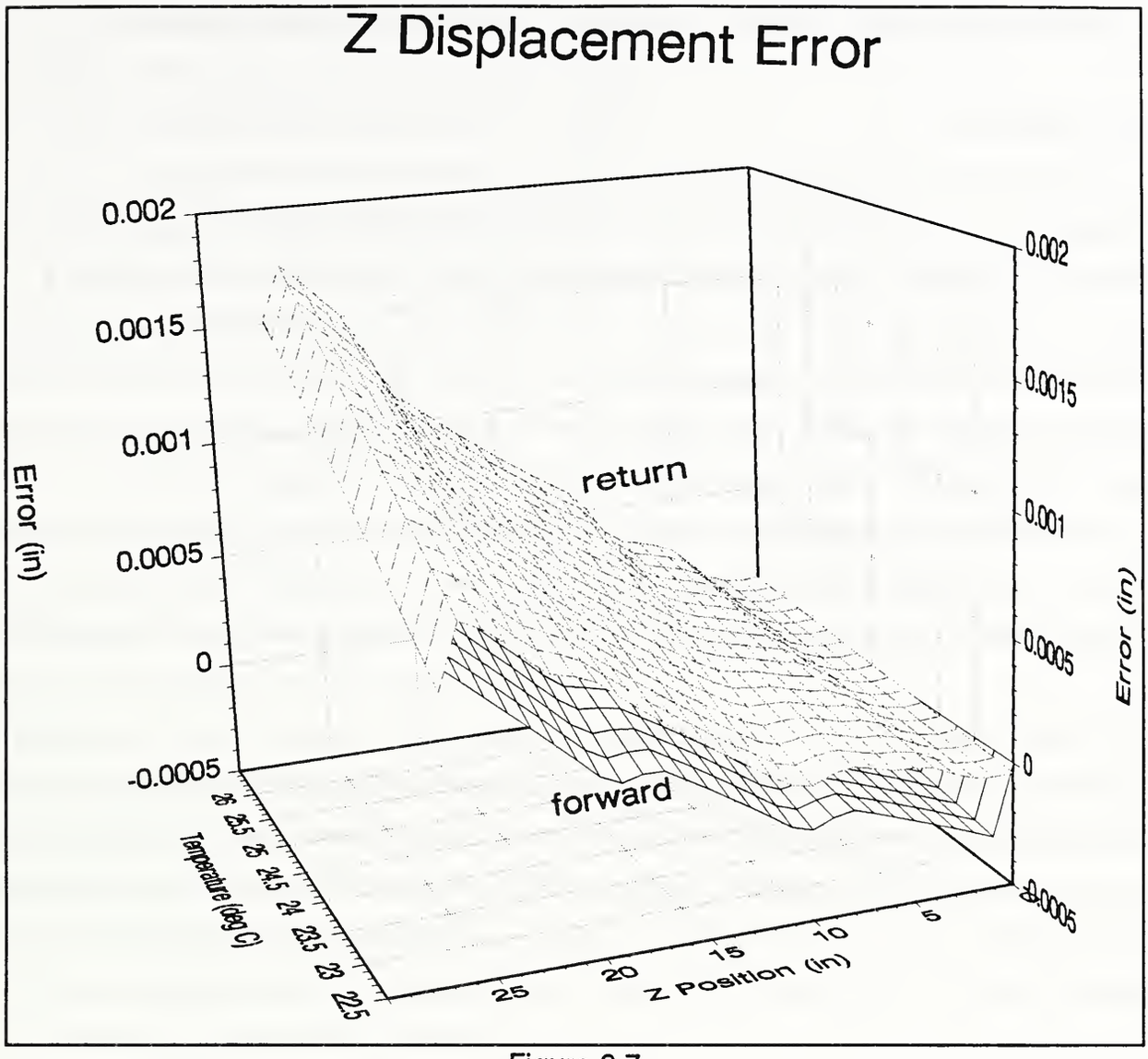


Figure 3.7

The error surfaces generated by the statistical techniques described above can be used to predict any particular geometric error component of the machine during the cutting operation regardless of the time and/or temperature history of the machine.

3.2.3. Future Work

Currently, the spindle thermal-drift measurements are being taken on the turning center. The purpose of these measurements is to identify spindle axial, radial and tilt drift motions as functions of spindle bearing temperatures. Once these relationships are constructed, they will be incorporated into the G-T model.

3.3. SOFTWARE DEVELOPMENT FOR REAL-TIME ERROR COMPENSATION

3.3.1. Overview

The empirical G-T model being developed is used to estimate the various components of the machine's systematic errors at a given cutting-tool position and for a given machine temperature profile. In order to calculate the resultant error vector of the cutting tool based on these error components, one has to use a kinematic error model of the machine structure. This type of model describes the actual relationships of the machine elements to each other, taking individual error components into account. Thus, it is possible to describe mathematically the actual cutting-tool position with respect to the workpiece. A kinematic model for a two-axis turning center has already been developed in an earlier study [5]. In this project we are using a similar kinematic model to calculate the resultant error vector.

The simplified version of the kinematic model developed in the previous study expresses the resultant error vector of the cutting tool in terms of its two components, p_{Ex} and p_{Ez}, along the machine's x and z axes of motion. Since a two-axis turning center does not have any ability to move the cutting tool in the third orthogonal direction, i.e., the "y-axis", the third component of the resultant vector is of no significance to the current error compensation effort. Therefore, only two components of the error vector are given by the following equations:

$$p_{Ex} = \epsilon_y(s)*z(w) - [\epsilon_y(z)+\epsilon_y(x)]*Z_T - \delta_x(z) - \delta'_x(z) - \alpha_p*\Delta z + X_1 \quad \text{..... (3.1)}$$

$$p_{Ez} = -\epsilon_y(s)*x(w) + [\epsilon_y(z)+\epsilon_y(x)]*X_T - \epsilon_y(z)*x - \delta_z(z) - \delta'_z(x) - \alpha_o*\Delta x + Z_1 \quad \text{..... (3.2)}$$

where

$\epsilon_y(s)$ tilt error of spindle about y-axis,

$\epsilon_y(z)$ yaw error due to carriage z-motion,

$\epsilon_y(x)$ yaw error due to cross slide x-motion,

$\delta_x(x)$ displacement error of cross slide x-motion,

$\delta'_x(z)$ x straightness of carriage z-motion,

α_p parallelism error between z-motion and axis average line of spindle,

Δx, Δz incremental x and z motion,

$\delta_z(z)$ displacement error of carriage z-motion,

$\delta'_z(x)$ z straightness of cross slide x-motion,

α_o orthogonality error between x-motion and axis average line of the spindle,

x(w),z(w) ideal cutting point coordinates on work piece,

X_1, Z_1 machine offset,

X_T, Z_T tool dimensions.

The error components in the above equation are functions of the values of x and z, and of the machine temperatures as defined by the empirical G-T model. The G-T model for the previous study consisted of polynomial functions of temperatures corresponding to various positions along the two axes. In order to represent these polynomial relationships, tables of coefficients were constructed. This scheme is slightly different from the approach in the present modeling effort discussed in Section 3.2.2.

The software described in the following sections was developed using the previous model as the basis. This software is not necessarily the final version to be used in this project. Therefore, it is expected that this software will be modified according to the results of the ongoing G-T model development effort.

3.3.2. The Error Compensation Program

The recently developed error-compensation program implements the model described above to calculate the geometric and thermally-induced machine errors. It was developed for the PC/AT- compatible QC running at 12 Mhz with a math coprocessor. It is written in "C" and has about 450 lines of code and requires about 47K bytes. One design objective was to minimize the execution time, since correction calculations must run in real time. Another objective was to determine the complexity of the calculations involved by measuring the actual run time, and drawing conclusions with respect to other more complex machine tools, where three error vector components with more terms will be computed.

3.3.2.1. Interpolation

Since interpolation is involved in the computation of every error term, it came under particular scrutiny, and was optimized for minimal execution time. If a function to be interpolated is a polynomial, as in our G-T model, time is saved by interpolating the coefficients instead of the function itself. This saving becomes more important for machine tools with complicated structures.

Generally, the value of a variable y corresponding to an independent variable x between x_n and x_{n+1} can be calculated by the following linear interpolation equation:

$$y = y_n + [(y_{n+1} - y_n) / (x_{n+1} - x_n)] * (x - x_n) \quad \text{..... (3.3)}$$

where y_n and y_{n+1} are the values of the variable y corresponding to independent variables x_n and x_{n+1} respectively.

This equation can be rewritten as follows:

$$y = C_n + V_n * x \quad \text{..... (3.4)}$$

where

$$V_n = (y_{n+1} - y_n) / (x_{n+1} - x_n) \quad \text{..... (3.5)}$$

$$C_n = y_n - V_n * x_n \quad \text{..... (3.6)}$$

The values of V_n and C_n can be computed once and for all and outside the real-time loop. At interpolation time, only one multiplication and one addition have to be performed.

3.3.2.2. Error Calculations

Prior to error-compensation calculations, auxiliary tables required for interpolation are created. The error calculation process consists of two phases. In the first phase the error terms, which are functions of temperature only, are computed for a set of input machine temperatures.

These are the errors such as spindle tilt, orthogonality, and the yaw drift of the home position. In the next phase, error terms which are functions of both positions and temperatures are calculated. These are the errors such as linear displacement errors, angular and straightness errors along both axes of the machine tool. The calculations in the second phase as well as others required to evaluate Equations 3.1 and 3.2 are performed during cutting in real time.

In order to carry out the calculations, the program uses the current nominal position x and z from the RTEC, the tool dimensions, - X_T, Z_T,- and the temperatures of six locations determined in the earlier study [5] to be representing the temperature profile of the machine structure. The components of the resultant error vector p_{Ex}, p_{Ez} are the only outputs of the program.

3.3.3. Performance Tests and Results

The run time of the two-axes error compensation calculations was measured and found to be 6 ms. Considering the additional error terms to be calculated, it is estimated that the calculations for a three-axes machine will take approximately 10 ms. The current plan for the error-compensation cycle time is 100 ms. Even though there are more computations to be included into the current version of the software such as the calculations for the machine offsets, tool errors, etc., it is clear that the timing will not be any problem.

3.4. EVALUATION OF THE REAL-TIME ERROR CORRECTOR PERFORMANCE

3.4.1. Overview

Selected testing has been performed on the Real-Time Error Corrector (RTEC) to quantify its repeatability in the fast-probing mode (which is used in the process-intermittent control loop) and to demonstrate its ability to modify the tool path of the machine tool (which is used in the real-time control loop). Fast probing is an on-machine part measurement using a touch-trigger probe with a feed rate of 2500 mm/min (100 in/min) which is 10- to 20-times faster than has been commonly used. The tool path can be modified by the RTEC which is inserted between a position feed-back device producing "encoder-type" signals and the

machine-tool controller (MTC). Pulses are added or subtracted from the signals which alters the count in the MTC and thus alters the tool path from the NC-programmed positions. The number of pulses added or subtracted compensates for the resultant error vector computed by the QC from the G-T and kinematic models. The design of the RTEC has been described previously [6]. The initial application of the RTEC is on the two-axis 18.6 Kw (25 hp) Monarch Metalist turning center. This machine has linear glass scales with electronic conditioning that produces encoder-type signals with a position resolution in the MTC of one micrometer.

3.4.2. Repeatability of Fast Probing

To determine the measurement repeatability of fast probing, a point on the face of a part was probed repeatedly. The stand-off distance was 5 mm (0.2 in) and the over-travel distance 2.5 mm (0.1 in) to ensure a constant velocity of 2500 mm/min (100 in/min) at the trip point . A point was probed and the trip-point axes-positions stored in a PC-type computer, interfaced to the RTEC, every 520 ms. The standard deviation of 35 measurements was calculated to be 0.45 micrometer. Hence, the two standard-deviation repeatability is 0.9 micrometer which is essentially the same as the repeatability specification of the probe [1 micrometer at a velocity of 480 mm/min (18 in/min)] and the resolution of the position feedback after decoding (1 micrometer). Therefore, we conclude that the fast probing does not reduce repeatability.

3.4.3. Tool Path Modification

An experiment was designed to demonstrate the ability of the RTEC to predictably and precisely modify the tool path of the machine tool. A part 152 mm (6 in) in diameter was faced with a very slight deviation from flat over the radius. The NC part program was written to produce an 0.20-mm (0.008-in) deviation over a 66-mm (2.6-in) measuring range portion of the part radius. A part was cut and measured using fast probing and a mechanical dial gauge to verify the NC program and machine tool operation. The identical NC program was then run again while the RTEC inserted "correction" counts to modify the tool path to cut the face flat. The QC computer receives real-time axis-position data from the RTEC. The QC commanded the RTEC to insert a count for every increment of travel in x (along the radius)

which would result in a 4-micrometer (the correction count resolution) deviation in z for the programmed tool path. Over the 66-mm (2.6-in) measuring range, 50 correction counts were required. A correspondingly larger number of counts were inserted over the total tool-path length of the cut. The part was again measured using fast probing. A representative test had the following deviations in micrometers from the ideal modified flat-face at 8 uniformly spaced probing points over the measuring range: 0, 1, 1, 2, 5, 4, 4, and 3. This is as good as can be expected since the correction resolution is 4 micrometers. On-machine probing may not show the true profile of the part since systematic errors in the machine-tool geometry will not be observed. However, the accuracy of the tool-path modification by the RTEC will be correctly measured.

3.4.4. Future Work

As soon as the G-T model and kinematic model have been completed and programmed to run on the QC, a test of real-time error compensation will be conducted. Initial testing will be with a simple test part, selected to have as large dimensions as practical and with a flat face and turned diameters. A step in the diameter will allow incremental and absolute diameter measurements, and an axial incremental length measurement. This part will allow assessment of the machine-tool nominal and error- compensated performance for x and z linear displacement, x and z straightness, orthogonality between x and z, and parallelism between z motion and the spindle axis. When satisfactory improvements have been demonstrated with this part over the range of machine operating temperatures, other test parts with tapers and circular contours will be used.

4. PROCESS-INTERMITTENT ERROR COMPENSATION

H.T. Bandy

4.1. INTRODUCTION

In the process-intermittent (PI) control loop, part-program modifications compensate for certain classes of process-related errors by modifying variables that determine the tool-path (such as coordinates, tool offsets, etc.) in an NC part program for the finishing cuts. The differences between the part dimensions gauged during pauses between machining passes, and the corresponding nominal dimensions, are analyzed to determine tool-path adjustments required for subsequent machining passes. The adjustments will be in the form of tool offsets in simple cases; otherwise, the coordinates in the NC part program will be modified. By programming coordinates which were modified on the basis of observed error patterns, the tool will traverse a path closer to the nominal path than it would if the part program remained unchanged.

Software for process-intermittent error compensation has been designed as part of an overall strategy which includes real-time error compensation. Even though the real-time components of the system are not complete, explanations of the process-intermittent software will be in the context of the completed system.

One goal of the real-time control strategy of QIA is compensate for all the systematic machine tool errors using the geometric-thermal (G-T) model and the Real-Time Error Corrector (RTEC). However, some machining errors, such as those due to varying tool length, deflections and wear, will occur in addition to random errors, despite the use of the G-T model and the RTEC. These are the errors targeted by the process-intermittent error-compensation strategy.

4.2. METHODOLOGY FOR PROCESS-INTERMITTENT ERROR COMPENSATION

The methodology described in this report has general applicability to turning processes, and nearly all of the principles are extensible to milling. However, the prototype software has

been developed for the turning center. The software is designed to accommodate several machine-then-gauge iterations. It is written in the C programming language and runs on the Quality Controller (QC), the personal computer used to control QIA processes on the turning center. A high-level programming language environment, *AMPLE*, is used on the QC to supervise the execution of process-intermittent error-compensation functions (See Chapter 5).

4.2.1. Process-Intermittent Inspection

Process-intermittent dimensional inspection is performed with a touch-trigger probe. The numerically-controlled machine tool can be programmed to use such a device to measure part dimensions and profiles, hole diameters, and distances between hole centers. As a gauging routine is executed, the RTEC reports the machine-axis positions to the QC for each probe-trip. The coordinates of the point of contact are then calculated by the QC in terms of the part coordinate system so that they may be compared to the corresponding nominal dimensions.

4.2.2. Error Compensation for PI Inspection

Once the G-T model is implemented , the QC and the RTEC will continuously correct the tool path during machining. But during gauging, the QC switches off this correction function. Therefore, to determine the true probe-trip locations, the gauging data has to be adjusted off line using the G-T model.

The *error*, or difference between the nominal dimension and the measurement obtained with the probe, is due to differences between the characteristics of the tool and the probe and between the dynamics of the machining and the probing process. Other errors are not detected. The process-intermittent error-compensation strategy described here assumes the G-T model to be accurate. The probe and gauging process are also considered to be accurate for the process-intermittent error data analysis. Inaccuracies in those areas will be detected in the post-process control loop. Because of these idealistic assumptions, the measured error is multiplied by a weighting factor of less than 100% (the actual value to be refined with experience), before being used to adjust the tool path.

4.3. IMPLEMENTATION OF PROCESS-INTERMITTENT ERROR COMPENSATION

To implement process-intermittent error compensation some critical steps have to be performed in sequence. These steps are segmentation of NC part programs, development of on-machine inspection routines, generation of archival part programs, generation of data files associated with the parts, collection and analysis of on-machine inspection data, and modification of part programs.

4.3.1. NC Part-Program Segmentation

In order to insert on-machine inspection routines before and after various semifinishing and finishing cuts, original NC part programs must be divided into segments. Segmentation is based on subdividing the part geometry into features according to the processes for machining different sections. Places in the part program where execution may be safely interrupted must be identified. They typically occur after blocks of code corresponding to the machining of certain sets of features. Since the dimensions after roughing cuts are not of critical interest, NC code pertaining to roughing cuts may comprise large segments, or even a single segment. NC code for semifinishing and finishing cuts, on the other hand, are contained in small segments to permit gauging after only few cuts.

These independent segments are downloaded to the machine-tool controller sequentially, and they are executed one at a time. Therefore, header and footer blocks are added to each segment as appropriate. The end of each segment is marked by an "M30" NC command, which causes the spindle, coolant and feed to stop. Between the segments the execution will be suspended to perform certain logistical tasks for error compensation such as acquiring temperature data, transferring measurement data, etc.

Eventually, a software routine may assist in part-program segmentation. Until it is completed, segmentation will be accomplished manually. The segmentation boundaries may change when the requirements of the dimensional gauging sequences, to be inserted as independent segments, are considered.

4.3.2. Development of Dimensional Gauging Routines

In this report, the word *cut* refers to: a machining pass of the tool along a single curve (or straight line), or the surface produced by that pass, as a geometric subfeature of the part. In preparation for developing gauging instructions to be sequenced with other segments of the part program, each part surface which will need dimensional measurements must be identified as a cut. Figure 4.1 shows a sample part with cuts and gauging points identified on it. In this figure, cuts 1 and 7 are "hypothetical cuts". They do not represent actual surfaces on the part, but are defined such that their intersections with real cuts are start- and end-points of tool paths.

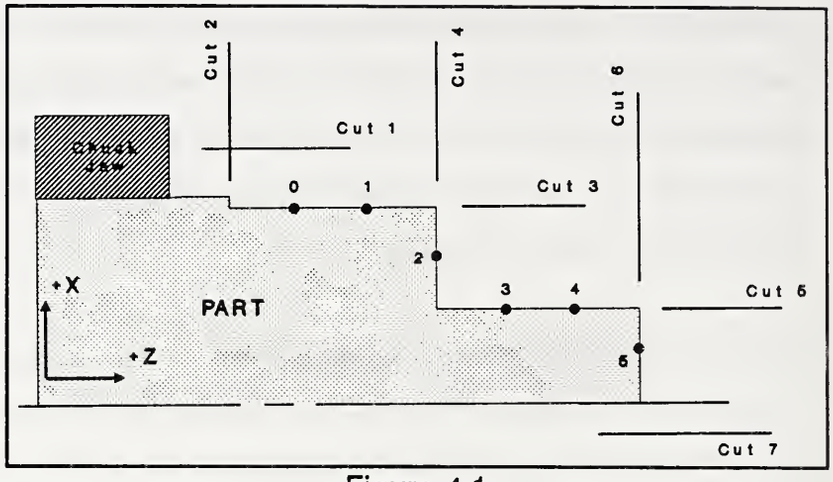


Figure 4.1

Future software will assist with the following procedures, presently done manually:

1. On the basis of the part geometry, determine all the measurements that will need to be taken on the surfaces considered. Develop a file of the nominal dimensions at each measurement point.

2. Define the gauging steps for making the necessary measurements. Taking the machine-tool acceleration distance, and overtravel limits of the touch-probe stylus into consideration, the coordinates of the probe paths will be based on the nominal dimensions.

3. After the verification of the probing steps, convert the steps into routines——i.e., entire sequences of formal instructions. Add header and footer blocks to each gauging routine as necessary to form independent part-program segments. Each routine will be executed during a single interval between machining passes.

4. Insert each gauging routine segment into its appropriate place in the sequence of part-program segments.

4.3.3. Generation of Archival Programs and Related Data Files

Archival part programs are special adaptations of original NC part-program segments. They are generated by replacing tool-path coordinates with variables and by adding tool-offset commands near the beginning of the program segment. Archival programs function as templates. They are used in conjunction with special data files to generate usable NC part programs. Process-intermittent modifications are made only to temporary copies of the archival part program.

In order to generate the above mentioned special data files, the following determinations must be made prior to the operation. The x-length of the probe and its stylus diameter must be calibrated. Each part surface to be identified as a *cut*, as well as the nominal coordinates of the gauging points to be located along the cut, must be decided. The order in which the gauging points will be measured must be known. The point whose z-coordinate is to be used as a reference location on the part must be identified. The tool-path specifications which are to be variable must be decided. The nominal shape of each surface and its important planar relationships to others must be specified.

Among other data to be available in files is information concerning each cut containing gauging points, including (1) a cut-identification number to identify each surface to be gauged; (2) the particular tool-path specifications responsible for the last machining pass that produced the cut; (3) the direction of the last pass; (4) the surface-normal vector at each gauging point; and (5) codes associating part-program instructions with particular gauging results.

4.3.4. Collection and Analysis of Gauging Data

While the machine tool controller is executing a gauging routine, inspection data are acquired by the QC from the RTEC under the supervision of *AMPLE* and are stored in a file. The

details of the communication between the QC and the RTEC are explained in FY89 Annual Progress Report of the project [2]. The process-intermittent error compensation software module reads the data from the file and processes it to generate NC-program modifications. The procedure for the analysis is given below.

The error at each measurement point is defined as the vector between the measured coordinates, and the nominal coordinates corresponding to that point. This vector is decomposed into its components along the surface normal and the surface tangent. Only the component along the surface normal is used for the compensation purposes. Excessive magnitude of the component along the surface tangent indicates a gauging problem. In this case, a warning should be generated and data corresponding to this point disregarded.

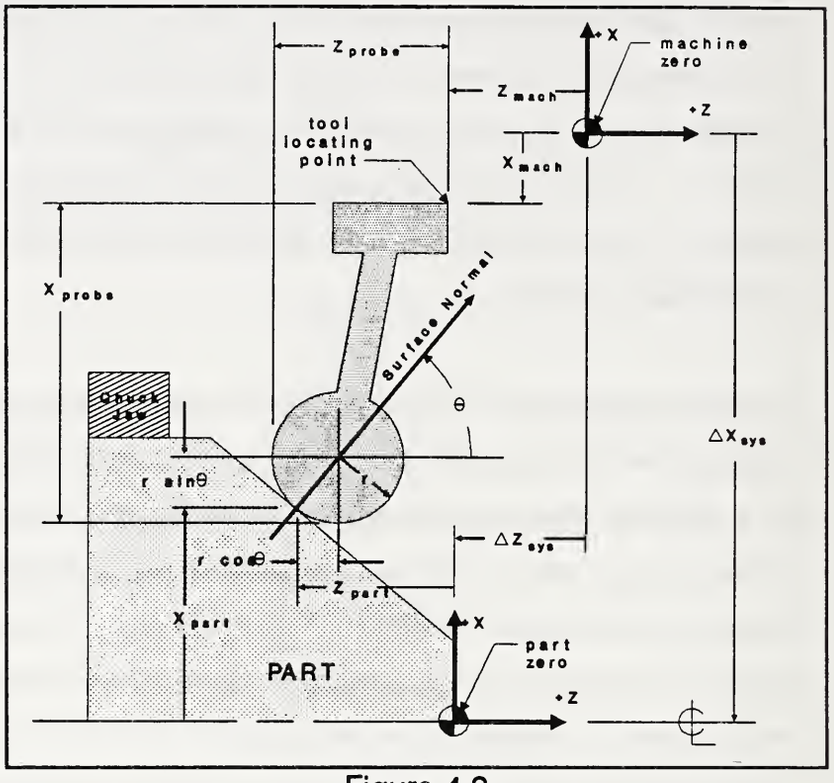


Figure 4.2

At the end of execution of each process-intermittent gauging routine, the following steps will be taken:

1. Using the machine-tool temperatures that existed at the time of gauging, the machine-tool coordinates that are registered at each probe-trip will be adjusted using the G-T model. These adjusted gauging coordinates are then converted from the machine-tool coordinate system to the part coordinate system. The surface normal vector for each gauging point is used for this conversion. Figure 4.2 illustrates the determination of the part coordinates. Referring to Figure 4.2, X_{part} and Z_{part} can be calculated using the

following equations:

$$X_{part} = X_{mach} + r(1-\sin\theta) - X_{probe} - \Delta X_{sys} \quad \text{..... (4.1)}$$

$$Z_{part} = Z_{mach} + r(1-\cos\theta) - Z_{probe} - \Delta Z_{sys} \quad \text{..... (4.2)}$$

The adjusted part coordinates would be identical to the target coordinates in the case of zero error. Otherwise, each error is calculated as the difference between an adjusted coordinate and the target coordinate corresponding to it.

For the next machining cut, the tool-path error expected on the basis of this analysis will be compensated by an adjustment in the tool offset or part-program coordinates:

$$\text{adjustment} = -(\text{error}) \times (\text{weighting factor}) \quad \text{..... (4.3)}$$

There may be a need for a separate adjustment equation for each axis, or even for different coordinate ranges for each axis. The equations will differ only in the values of weighting factors, which should be less than 100% in every case, to avoid overcompensation. (The Monarch Metalist Programming Manual suggests a weighting factor of 20%.) Coordinates in tool-path specifications in the next machining segment may be adjusted by adding the calculated adjustment to the nominal coordinates in that segment.

2. Modified tool paths should retain the general shape of the original tool paths. Therefore, the set of coordinate adjustments calculated in the above step has to be processed to ensure the smoothness. The adjusted coordinates are fitted to the nominal shape of the measured part feature using least-squares curve fitting techniques. Thus, the actual position and the orientation of the measured feature are determined with respect to those of the ideal feature. NC-program modifications are performed based on this information. The general polynomial equation used to determine the correction curve is given below.

$$C_1z^3 + C_2z^2 + C_3zx + C_4x^2 + C_5z + C_6x + C_7 = 0 \quad \text{..... (4.4)}$$

With the use of appropriate of coefficients, Equation 4.4 can be used to describe various geometries such as lines, circles, quadratic and cubic curves. The coefficients are selected based on the nominal shape of the measured feature. The correction curve, derived by fitting the adjusted coordinates to the polynomial, becomes the tool path for the corresponding cut in the next machining segment.

4.3.5. Modification of Part Programs

The adjusted data described in the previous section are used as input to correction functions to calculate new tool-path specifications. Figure 4.3 shows the functional block diagram of the overall part-program modification system. The procedure for the modification is described below.

1. If the measured feature is found to be only translated with respect to the ideal one, which is indicated by a nearly constant offset error along an axis, a tool offset change would be adequate.

2. If the measured feature is found to be translated, rotated, and/or deformed without changing the planar characteristics (called a plane angle error), compensation can be done by modifying selected end coordinates related to these features.

3. If the shape of the measured feature is found to be deformed with respect to that of the ideal one, the compensation requires more elaborate modifications in the NC programs. These modifications include breaking the features into smaller linear sections and modifying the end coordinates of these linear sections accordingly.

If tool offsets are to be adjusted, the calculated x and z increments are substituted into the update commands in the pertinent part-program segment.

For cases in which the endpoint of one smooth tool path also serves as the start point of the next tool path, the adjusted coordinates for that point are calculated as the intersection of the

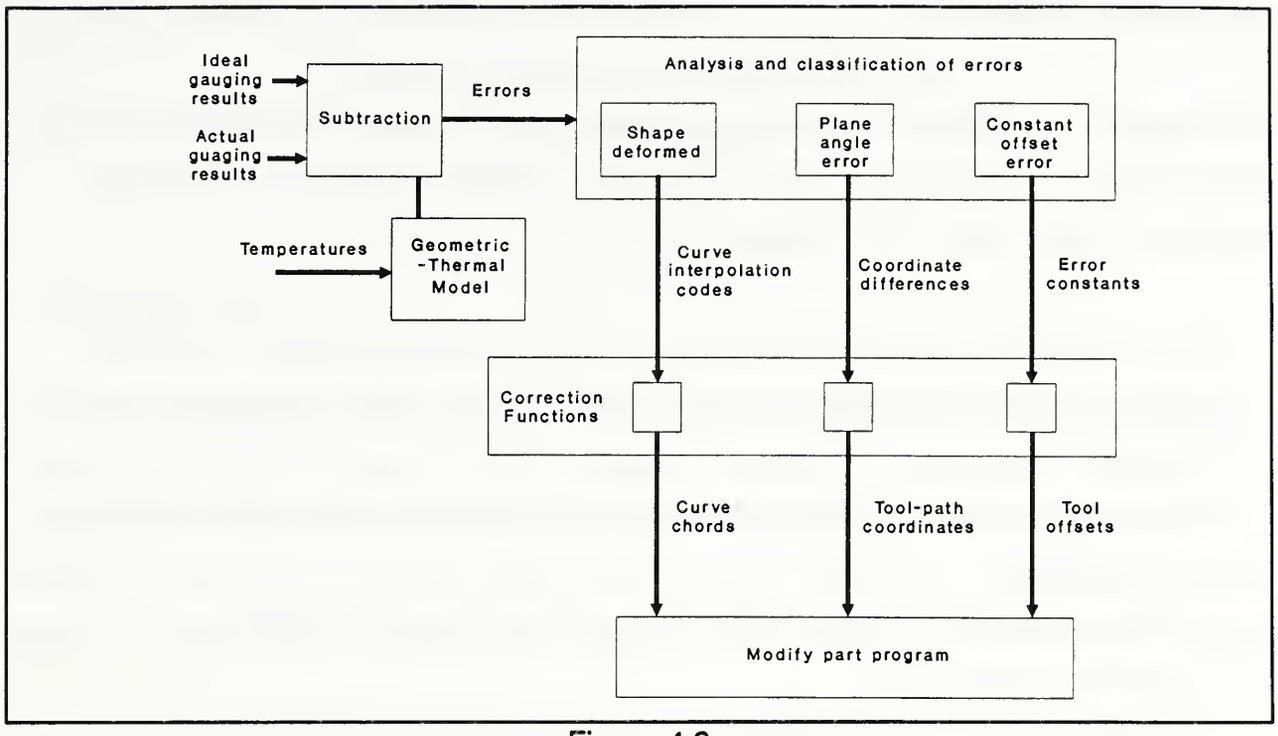


Figure 4.3

two correction curves for the paths. The new coordinates are substituted for the nominal coordinates in the pertinent part-program segment.

4.4. PROGRAM INPUT AND OUTPUT

Descriptions of the required input files and the data they must contain would involve much more detail than would be appropriate in this report. Thus references to input data have only stated that the data would be available in files. However, descriptions of the only three output files, namely report.txt, seg0N0K.dnc, and status.now, are easily simplified. These files are created at various stages of execution of the software.

The file "report.txt" contains the summary of error history for a completed part. The file "seg0N0K.dnc" contains the version of Segment K of the NC program for Part N, which is to be downloaded to the NC controller. Tool-path specifications in this version have been refined to reduce errors. The file "status.now" contains the status information to be used by the other software modules in the system.

4.5. FUTURE PLANS

The software for process-intermittent error compensation has been described here in the context of the completed QIA system, even though certain modules are not yet fully functional. Further work on this software will include:

- More precise and systematic treatment of part feature segmentation, relating the features used here to those used elsewhere in the QIA system, particularly the post-process loop;
- Incorporation of the G-T model of the Monarch Metalist into the process-intermittent data analysis;
- Software to assist in the generation of dimensional inspection routines;
- Analysis of circular arcs;
- Extension of curve-fitting capability to correct for the shape errors of the parts.

5. IMPLEMENTING THE PROGRAMMING LANGUAGE ENVIRONMENT FOR THE QUALITY CONTROLLER

J.C. Boudreaux

5.1. INTRODUCTION

The Quality in Automation (QIA) project requires 1) the real-time acquisition and processing of sensory data, and 2) the development of adaptive machining algorithms for modifying the tool path as well as the cutting parameters in order to compensate for changes in the machining environment. The Automated Manufacturing Programming Language Environment (*AMPLE*) is being enhanced to provide the functional and computational capabilities needed in this project.

AMPLE was conceived as an interpretive programming language environment which provided off-line programming services to permit the construction of control interfaces to industrial manufacturing systems [7,8]. It consists of a core and associated software modules. The *AMPLE* Core is a collection of representable values, called **objects** (see Appendix B, Section B.1). In its original version, Version 0.1, *AMPLE* Core Interpreter (**amcore**) was based upon a commercially available dialect of Lisp called Franzlisp [9,10]. This version was implemented on a Silicon Graphics IRIS workstation under the Unix[1] operating system.

The QIA project has two major requirements from the software system which can not be supported by the original version of *AMPLE*. These requirements are (1) the capability of running on a AT-class personal computer called the Quality Controller (QC) under an MS-DOS[2] operating system and (2) the capability of providing control, data acquisition, and processing services in real time. These requirements created the need for a new *AMPLE* version to be implemented in the QIA project.

The requirement of running under MS-DOS implied that several important Version 0.1

---

[1]Unix is a trademark of Bell Laboratories

[2]MS-DOS is a registered trademark of Microsoft Corporation

modules would be difficult or impossible to transport to the new platform. In addition, Unix multiprocessing commands used in version 0.1 [7, Appendix B], would no longer be available.

The second requirement implied that **amcore** should use a Lisp interpreter with exotic capabilities such as an internal real-time processor, **rtp**. The **rtp** must support two design goals: (1) in order to return the correct value in a timely manner, it must be able to execute at nearly compiled-language speeds; but (2) in order to be adequately programmable, it must provide direct access to **amcore** objects. These goals can be achieved by using special off-line programming services to build **amcore** objects which may then be accessed and updated using the special access functions within **rtp**. All Lisp dialects and all other interpretive systems need to reclaim memory, a process called *garbage collection*. Garbage collection for MS-DOS AT-class computers can take 500 ms or more. To prevent garbage collection, the access and update functions must never cause the allocation of any free memory cells.

After reviewing public domain Lisp interpreters for the AT-class MS-DOS PCs, the XLISP 1.7 [11,12] was selected as the starting point for the development of the new *AMPLE* version, **amcore** Version 1.0. The User's Guide [13] documents the revisions and additions made to XLISP 1.7 and also provides a complete inventory of the primitive **amcore** functions. Note that the User's Guide does not describe the functions which provide interface services to the **rtp**. A partial list of this family of functions, still under development, is given in Section 5.3 below.

In the remainder of this chapter, Section 5.2 describes the real-time processing component in the context of the turning center operations. Section 5.3 defines several methods for increasing the functional capabilities of *AMPLE* to accommodate various QIA tasks. Section 5.4 summarizes the future work planned for the current fiscal year. To make this report more or less self-contained, a technical review of **amcore** is given in Appendix B.

5.2. REAL-TIME PROCESSING

As noted earlier, the QIA project requires that *AMPLE* should support not only an off-line programming mode, but also a real-time processing mode. In this section, a general overview of an early prototype of the **rtp** will be given. This prototype was designed to support the implementation of the real-time tool path modification and the process-intermittent control algorithms on the Metalist Turning Center during the past year. This overview will include descriptions of both the primitive **amcore** functions and the functions programmed directly in *AMPLE* for execution by **amcore** itself.

Generally speaking, *AMPLE* programs may be resolved into two different kinds of components: (1) functional components, or operations, which correspond to stereotypic motions of the system being controlled, and (2) flow-of-control components, or scripts, used to control the order in which operations are performed. During the past year, the prototype was configured to provide four operations:

1. **correction**, which reads and displays Real-Time Error Corrector (RTEC) data, calculates the number of correction pulses required, and then writes this value to the RTEC; this mode of operation is used to modify the tool path during cutting to compensate for the expected errors. In the current version of the script, the correction values are generated internally. In the future, correction values will be generated by an external module described in Chapter 3.

2. **no-correction**, which reads and displays data from RTEC but which introduces no corrections; this mode of operation is used when the machine axes are moved at rapid traverse rates indicating no cutting therefore no correction necessity.

3. **fast-probing**, which collects, stores, and displays positional data from probed points [14]; this mode of operation is a part of the process-intermittent control function of the QC. It is used to collect data to determine the part shape and dimensions before and after semifinishing and finishing cuts.

4. **process-intermittent-analysis**, which modifies NC part programs to correct form errors in the part that are detected by comparing the empirically derived probed surface with the associated nominal surface. The software module to carry out this operation is written in C language and used as an external DOS extension to *AMPLE*. The functional specifications and operational details of this module were described in Chapter 4.

5.2.1. A Prototype *AMPLE* Script

An *AMPLE* script was written to carry out the above mentioned functions to demonstrate the real-time and the process-intermittent control capability of the QC. The script begins with the selection of a simple part geometry to be produced. For initial demonstration purposes this geometric feature was designed to be a flat face of a cylindrical part with a specific slope built in with respect to its centerline. Therefore, the geometry selection is done by keying in the desired slope. The requested slope is written to a file and an external process is started using the **amcore** dos function, as explained in the Section 5.3.2 below. This process, which is a part of the software module described in Chapter 4, generates an NC part program to produce a part with the requested slope and downloads it to the machine tool CNC via a serial RS-232 interface. The script initiates the first machining operation in no-correction mode to generate the original part geometry. Next, a fast-probing and the process-intermittent analysis operations are performed by sequential execution calls from the script to verify the shape generated in this cut.

In order to show the real-time tool-path modification capability, the next cut was designed to generate a different geometry using the original NC program. For simplicity, the new geometry is selected to be a flat-face which is perpendicular to the cylinder centerline. Generation of this part using the original NC program required the modification of the tool path during cutting by enabling the correction function of the QC. The script calls for the generation correction tables, discussed in the Section 5.2.2 below, to create the effect of canceling the requested slope based on initially keyed-in information. Tool path modification was achieved by the interaction between an internal **amcore** function and the RTEC.

A sequence of fast-probing and the process-intermittent analysis operations was called again

by the script. To demonstrate the process-intermittent error compensation capability, the resultant shape from the second cut was compared with the original desired geometry and the shape error was determined as described in Chapter 4. NC program was modified accordingly and redownloaded to the machine tool's CNC. In the final cut, the script calls for enabling the correction function while the machine is executing the modified NC program. Thus, the final cut generates the originally selected part geometry under significantly different operating conditions. By performing another fast-probing and analysis sequence, the QC verifies the part as programmed by this *AMPLE* script.

5.2.2. *AMPLE* - RTEC Interface

The current version of *AMPLE* is interfaced to the RTEC by means of a parallel digital IO board. The communication protocol for this interface, fully defined in [14], may be summarized as follows:

```
begin_record
   X.axis.status    :   4 bits;        Input
   X.axis.position  :   20 bits;        Input
   Z.axis.status    :   4 bits;        Input
   Z.axis.position  :   20 bits;        Input
   correction.C.x   :   1 byte;        Input
   correction.C.z   :   1 byte;        Input
   correction.R.x   :   1 byte;        Output
   correction.R.z   :   1 byte;        Output
   program.status   : 1 byte;          Input
   correction.status:  1 byte;         Output
end_record;
```

This protocol was determined in part by the characteristics of the digital IO board selected for this project. This board has seven Intel 8255A Programmed Peripheral Interface (PPI) chips, each of which contains three addressable IO ports and an addressable register. The width of each port is one byte. The IO direction is described from the point of view of the QC: if the

direction is Input, then **amcore** treats the corresponding ports as *read-only*, and if the direction is Output, then **amcore** treats the corresponding ports as *write-only*.

The **rtp** is serviced by primitive **amcore** functions. There are several interface functions described earlier [14] to facilitate the communications with the RTEC.

initialize_digital_io_board() initializes the ports of each of the 8255A PPI chips for either input or output. It is executed once during start-up.

initialize_diores () creates an **amcore** array object and associates it with the symbolic name DIORES. The components of this array object are initialized to zero.

get_digital_io () uses a function called **inp** to read all of the input ports of the digital IO board. In the case of the turning center, only the **X.axis**, the **Z.axis**, the **correction.C** data, and **program.status** are used.

put_digital_io () gets the **correction.R** and **correction.status** values from the internal *AMPLE* representation DIORES, and writes them to the appropriate port.

display_digital_io () is used to display the current values of the shared resource DIORES on the QC screen.

alarm () gets an initial value of **X.axis.position** and **Z.axis.position** and then returns either **true** or **nil**. **Alarm** is sensitive to three events: the NC part program has concluded, the operator has hit any key on the keyboard, or the current value of either the **X.axis.position** or **Z.axis.position** is not equal to their initial values. In the first two cases, **alarm** returns **nil**; in the third case, **alarm** returns **true**; otherwise, **alarm** loops.

5.2.3. New Additions to the Interface

During FY90, several functions were added to the existing **rtp**. The first group of four

functions is used to prevent the QC and the RTEC from performing concurrent IO operations.

write-hold() sets the high-order bit of correction.status to 0. This operation prevents RTEC from attempting to read the requested correction fields.

write-release() resets the high-order bit of correction.status to 1, which enables RTEC to read the correction fields.

The **read-hold()** function notifies RTEC that a read operation is to be done on the DIORES board.

read-release() indicates that no read operation is expected on the DIORES board.

correction-cycle(scale) implements an early version of the machine tool correction algorithm. It has a single floating-point parameter scale which is used to control the amount of time that will be spent in a delay cycle, which is used to guarantee that the correction-cycle will be repeated after a fixed, programmed wait. The following pseudocode illustrates the overall design of this function:

```
correction-cycle(scale);
 begin
  until kbhit() do
      get x-axis-position, z-axis-position;
      calculate x-axis-correction;
      calculate z-axis-correction;
      put x-axis-correction to correction.R.x;
      put z-axis-correction to correction.R.z;
      display_digital_io();
      delay-cycle(scale);
 end_do;
end;
```

In the current script, the correction algorithm was implemented by storing corrections for each axis in dynamic arrays. The general method is to create limit-value arrays for each axis containing the following integer values:

min = axis minimum position value for correction region
max = axis maximum position value for correction region
grain = resolution interval between min and max for correction region

These three values are selected based upon certain rules. First, min is not greater than max. Second, there is one and only one integer, say size, which is the largest integer such that

```
max >=  (min + (size * grain))
```

Third, every axis position less than min or greater than max has zero corrections.

Axis correction tables may be created by making arrays of that size, and then entering the correction count to be implemented at that position. For example, suppose that xpos is value along the x axis, and that XCOR is the x-axis correction table, then the calculation of the index of XCOR corresponding to xpos is determined by evaluating the following expression:

```
( xpos / grain) - ( min / grain)
```

Since all of the variables are integers, and thus both divisions are integer divisions, this expression will return an integer.

The array XCOR contains x-axis corrections based on x-axis position. But this is not the only possible case. In the current version, an array XZCOR was also used. This array contained z-axis corrections based on an x-axis position, which was required by the demonstration described in Section 5.2.1. In general, there needs to be a specific rule for combining the correction counts in several tables, the simplest of which is to add the partial corrections together.

5.3. METHODS FOR EXTENDING *AMPLE* FUNCTIONALITY

One of the things that any programming language environment must do is provide uniform mechanisms for accessing external operations. Three methods for extending the functional capability of *AMPLE* are described below.

5.3.1. Extending amcore

The most direct method to extend the functional capability provided to end users is to add functions as new **amcore** primitives. This method was used to develop the **rtp** interface discussed in the preceding section. This method has several advantages. First, the implementation of the functions is likely to run very efficiently. Second, if proper care is exercised, the results of the execution are immediately visible at the **amcore** level.

This method has one serious defect: in order to add new primitive functions to **amcore**, the programmer has to relink the system from scratch.

5.3.2. Using the DOS function

The second method is to use the primitive **amcore** function dos. Any command that can be legally entered to the MS-DOS prompt can be passed as an argument string to the dos function, including those commands that invoke executable programs. Thus, if there is an existing program, say PROC.EXE, which has the needed functionality, then the simplest way to invoke it is by keying the following command to **amcore**:

```
> (dos "PROC")
```

which has the effect of opening a MS-DOS shell, and then executing the program PROC.EXE. When the shell process terminates, the DOS shell is closed, the memory allocated to it is reclaimed, and control is returned to the parent **amcore** process. This method was used in the current version of the script to access the process-intermittent analysis and compensation software described in Chapter 4.

This method solves the most important defect of the first method, but has several problems of its own. Since **amcore** remains in memory, the upper bound on the size of all processes which can be can be started is severely restricted by the MS-DOS 640 Kbyte limit on accessible memory.

5.3.3. Remote Access

This method partially solves the MS-DOS size limit by allowing the existing **amcore** environment to be saved between sessions. A snapshot of the current state of **amcore**, which is the state of the environment, is taken and then written to a file. Given the fact that the entire state of amcore is the state of its symbolic values, what is actually being saved is the list of all user-defined symbols, called the *savelist*, and the values assigned to each of them. Only user-defined symbols need to be saved since the meaning of all primitive **amcore** symbols is fixed in advance by the interpreter itself.

Once *savelist* has been created, the defined **amcore** function save, which must be given a filename as an argument, writes to that file all of the symbols in *savelist*. If filename is LOADed then the effect is the same as if all of the functions had been re-keyed. In other words, the state of the system is the same as it was just prior to the execution of save. This method is especially useful when the external software to be accessed is too large to fit in memory when **amcore** is resident. For example, by using remote access, it is possible to define a practical interface with the CADKEY system which will eventually be used in the QC.

5.4. FUTURE WORK

There are two tasks to be accomplished in the next fiscal year. The first task will be the incorporation of ultrasonic sensing capabilities within *AMPLE*. Initially, this effort will require the integration of analog-to-digital conversion of a voltage signal in order to be able to measure the surface roughness of machined parts from the echo amplitude of ultrasonic pulses. The second task will be the incorporation of a commercially available database system on the QC for the storage and retrieval of both archival and transient data.

6. AUTOMATED INSPECTION USING DMIS AND IGES TO INTEGRATE CAD/CAM SOFTWARE PRODUCTS WITH CMMS

D.C. Stieren and S.D. Phillips

6.1. INTRODUCTION

The post-process inspection of a manufactured part has come to serve as the "metrological anchor" of the manufacturing process. The inspection procedure performed in this loop ultimately determines whether a finished part meets dimensional specifications --if the part is accepted or if it is rejected. Also, post-process inspection monitors and provides the data to measure the performance of, and improve, other parts of the QIA system, such as process-intermittent (PI) part probing and the real-time error compensation algorithm provided by the machine tool's geometric-thermal (G-T) error model.

Advances in production methods such as CNC machine tools and integrated CAD/CAM products have increased the speed of the production run. These increases in production speed have dictated the need for advances in part inspection techniques. This problem has been addressed by the use of coordinate measuring machines (CMMs). Not only have CMMs greatly reduced inspection times, but they have also increased the flexibility, reliability, and accuracy of measurements. CMMs that have different levels of sophistication are available and in use in the manufacturing arena: there are manual machines, machines that operate under remote joy stick control, and machines that operate under direct computer control (DCC).

6.2. TRADITIONAL PART INSPECTION USING THE DCC CMM

All of the CMM inspection work performed within this project has been carried out on a DCC machine. DCC CMMs provide the highest levels of inspection accuracy, throughput, and automation. Until recently DCC CMMs have received their inspection instructions through a labor-intensive process. Instructions are either produced in a teach mode, or written in the specific language utilized by the particular CMM's operating system. The first method requires that at least one part of a batch be produced before the inspection routine for the

part can be created. The second method requires the operator to be familiar with the CMM's specific operating language. Both these methods require considerable time to produce a part program, and they both require a certain amount of time for operator-CMM interfacing, which is essentially nonproductive down-time for the CMM. This down-time can be minimized by using a standard interface language.

6.3. THE DIMENSIONAL MEASURING INTERFACE SPECIFICATION

In February, 1990, the American National Standards Institute (ANSI) adopted the Dimensional Measuring Interface Specification (DMIS) as a national standard [15]. DMIS is designed to prescribe a standard format in terms of a common language to allow interfacing between CAD/CAM software products and dimensional measuring equipment hardware products, such as CMMs.

DMIS serves as a bi-directional communication standard for computer systems and inspection equipment. In other words, inspection programs and the resulting inspection data can be expressed in a generic vocabulary that is available to both CAD/CAM software and CMMs and serves as the common link between the two. This allows the generation of part programs within CAM software that utilize data bases produced within a CAD package. These part programs can be generated off-line through graphical computer interfacing and the use of pre- and post-processors. This method, detailed in this text, greatly reduces the amount of time required to produce a part program.

While it is designed for use in a computer-controlled environment, DMIS is both man-readable and man-writable. If required, DMIS part programs can be edited manually. In addition, DMIS facilitates the development of a data base to store inspection results from various types of dimensional measuring equipment by providing a common format that allows easy access to information. In other words, with the aid of translators, DMIS information can flow to and from different inspection machines, regardless of their specific operating languages.

6.4. SYSTEM OVERVIEW

6.4.1. NIST Inspection System

An inspection system has been implemented at NIST that integrates CAD/CAM software products with a CMM using the DMIS standard. This system also utilizes the Initial Graphics Exchange Specification (IGES) [16] to allow for interfacing between the different CAD and CAM software products that are present. IGES provides for the exchange of graphical data among CAD systems from different manufacturers. The NIST system has been implemented with various goals in mind: (1) demonstrating the operation of DMIS using products commercially available to the industrial public; (2) incorporating off-line automated inspection path generation directly from a CAD data base to the CMM using DMIS; (3) demonstrating the utility and flexibility of ANSI standards by integrating into the system products from many different manufacturers; and (4) addressing the needs of the small-to-moderate size machine shops in terms of cost and system simplicity by integrating the entire post-process inspection system into one enhanced PC-class computer with an 80386 processor.

There are other systems existing in manufacturing environments today that perform basically the same functions as the NIST system. These other systems, however, may eliminate the integration of DMIS and/or IGES. Therefore, they are machine-specific to the CMM used, the computer system used as the controller, or both. Also, these systems are generally more complex and expensive. By using ANSI standards and PC-class computer control, NIST is attempting to address the needs of a wide audience with this system.

The schematic flow diagram for the NIST automated inspection system is shown in Figure 6.1. This diagram illustrates graphically what the system does and how the system works. The realization of the strategy for this system is the direct result of a cooperative research effort among NIST and six U.S. industrial manufacturers, in conjunction with NIST's research associate (RA) program [17]. It is stressed that each module in Figure 6.1 represents a class of products that can be used in this type of system configuration; specific products are mentioned in the discussion of NIST's system with the intention of conveying general information regarding the integration and implementation of this type of system.

6.4.2. Inspection Process

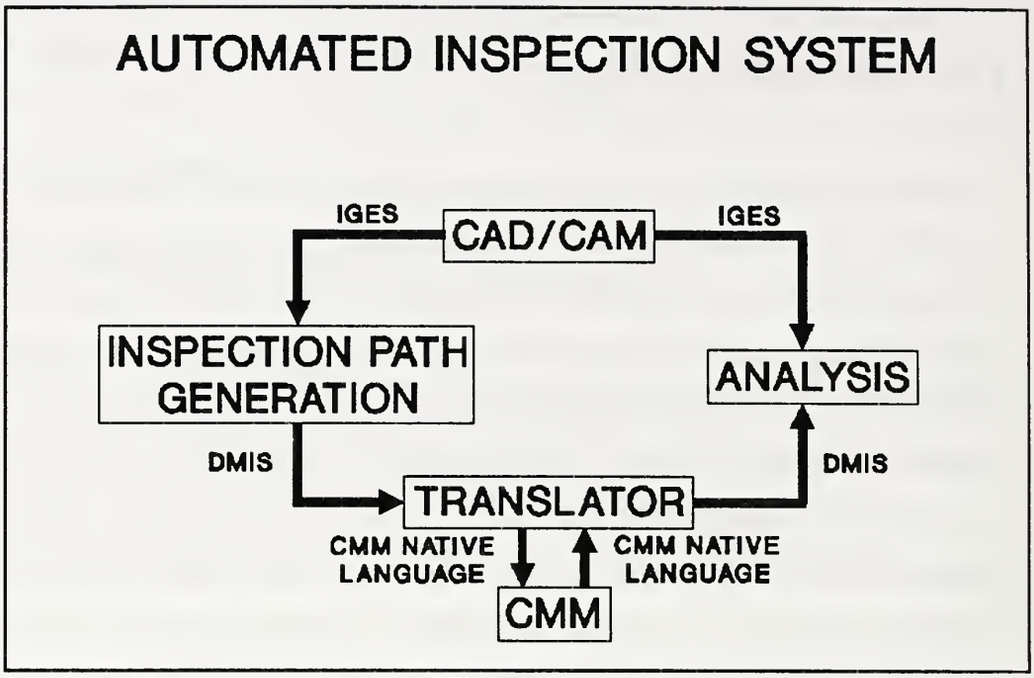


Figure 6.1

The inspection process commences with the generation of a CAD representation of the manufactured part. This CAD representation is in the form of a three-dimensional wire-frame model and is produced within our system using a Cadkey CAD package. The CAD data base is then translated into the IGES format using a Cadkey-to-IGES translator, to allow the CAD geometry to be utilized elsewhere in the system. The IGES standard provides a common graphics format that allows part geometry information to be used by various software products. This IGES geometry file is imported into a CAM software package called PCDMIS, where the CMM inspection path is created in DMIS format.

The path is created through a graphical interface on the computer screen, off-line and independent from the CMM. The user produces the inspection path by directing a mouse around the wire-frame image (IGES graphical representation) of the part geometry to designate the various points and features to be included in the part inspection. Once all the points, features, and other inspection information such as tolerance requirements and probe(s) data have been identified, the inspection routine, which is generated by the system, can be simulated on the computer. The simulation consists of an animation of the probe maneuvering about the part exactly as the actual hardware would when conducting the CMM inspection. This is basically an editing feature of the system, and when all input and editing is completed, the software produces a DMIS part program.

6.4.5. The NIST CMM

The CMM used in this system, a Sheffield Apollo Series Cordax[1] machine, has been retrofitted with a laser interferometer system for scale positioning and utilizes a Renishaw probing system. Since the operating system of the CMM is based on Sheffield's own software (not DMIS) a translation of the DMIS program must be performed using a pre-processor. The Sheffield translator turns the DMIS part program into a part program that the CMM can execute. This CMM-executable program (FLB3_D) is structured in the Hewlett-Packard (HP) Basic language and calls commands from Sheffield's function library (FLB3), as well as from the DMIS standard. The FLB3_D program is then loaded into the CMM operating system, and the part inspection is conducted on the CMM. All of the software for the system to this point operates within DOS. The operating system for the CMM which will execute the part program, however, is designed to run under the HP Basic language. This language operates using a separate processor located on a separate card installed in an expansion slot of the PC. This card is invoked in order to load and execute the Sheffield part program, and this system runs independently of DOS. Also, since this system operates independently of DOS, HP Basic can be operated in the background while running other DOS applications. Hence, the system can have multitasking inspection capability.

6.4.6. Inspection Results

The inspection results from the CMM are structured in two formats. The first is the inspection report that is produced by the Sheffield system. This report contains the specific information that is requested during the inspection path generation: for example, the specific report format, system information such as program name and execution date and time, feature numbers and types, actual and nominal point locations or feature sizes, and tolerance data. The second is the DMIS output file. This file is a DOS-formatted file created by a Sheffield application program within the Basic system. It is produced from the ASCII data file of the inspection results, and contains the inspection results in the DMIS output structure that is tailored by the DMIS part program. The specific DMIS structure here varies with each specific

[1]For disclaimer on the use of commercial equipment, see page iii.

inspection program, but it can contain the x, y, and z coordinate values of each individual data point collected for contoured surface parts; or it can contain the geometric feature information associated with prismatic parts.

6.4.7. Analysis of Results

For most prismatic parts either of the two inspection result formats described above provide adequate inspection information. However, for sculptured or contoured surfaces, DMIS results provide only x, y, and z probe coordinate data. Analysis of this type of data is performed within a commercially available software package called Qualstar. This software runs from an Interactive Systems UNIX 386/ix operating system that occupies a partition of the hard disk of the PC. There are software utility programs in UNIX that emulate DOS and allow DOS files to be read from floppy drives. These programs, however, do not allow one partition of the hard disk to be read from within another, but they do facilitate reading the DMIS output files in Qualstar.

Qualstar provides for the analysis of inspection results by comparing nominal part geometry with the actual, inspected part geometry. If the nominal part geometry is defined by the IGES parametric spline surface entities, the IGES file created earlier in the process may be used by this software. Otherwise, the nominal geometry must be re-created within the Qualstar software. The analysis here is conducted on a point-by-point and feature-by-feature basis for a part, and mainly consists of graphically overlaying inspection data onto nominal part data. Numerous fitting routines and geometric algorithms are available in this software that allow data manipulation and analysis capabilities. In addition to indicating the out-of-tolerance parts, the analysis provides information about the possible reason for the part rejection by examining every data point in an effort to depict trends in the manufacturing process.

6.5. DMIS SYSTEM PERFORMANCE

6.5.1. CAD-to-CMM

This portion of the text will concentrate on how the CAD-directed inspection system has

performed with regard to the various tasks that have been attempted. Most of the inspection work performed during FY90 focused on the NAS979 test part for milling machines [18].

Construction of a 3-D wireframe representation of the part geometry on the CAD system and the translation into IGES format are straightforward. All the IGES files we produced from the part geometries have been successfully accepted by the inspection path generation module. The inspection program for the NAS test part was constructed in approximately 40 minutes using this system, and the resulting output contained approximately 700 lines of DMIS statements. This is roughly an order of magnitude faster than can be achieved by manual programming without the assistance of CAD direction. Specifying inspection features is relatively easy, and they often have user-definable default values, e.g., six equally spaced points to measure a circle. Similarly, the computer simulation of the inspection procedure, which can be observed simultaneously in as many as four different views, is useful to debug the inspection routine.

6.5.2. Progress to Date

Referring to Figure 6.1, during the past year the system has implemented each of the major blocks in the diagram; however, it is not yet complete. For the inspections we have attempted, there is a certain amount of manual editing that must take place within the system. The main source of the manual editing necessary for a part inspection has been related to the translators within the system. These translators allow the different software products to be integrated with one another. Specifically, a translator is required to produce an IGES file from an associated CAD data base for use in both inspection path production and inspection result analysis. Another processor is needed to produce the CMM-specific language part program from the DMIS part program, and to produce the DMIS output file from the ASCII data file of the inspection results.

6.5.3. Inspection of Parts

As described above, there are two different strategies using DMIS for part inspections. For parts containing basic geometric features (circles, lines, etc.) the usual procedure would be to

use commands to evaluate these features. In this case the points measured on the part have been processed to yield meaningful entities such as a circle diameter. Alternatively, an inspection program could be written containing commands which produce raw point data, in the form of x,y, and z coordinates, as the final result. Contoured parts generally do not contain easily identifiable geometric features; therefore, these parts must be inspected in this manner, with the aim of collecting information from a number of individual points that constitute a certain surface or curve on the part.

Parts can be inspected using part programs that contain measurement functions from the vocabulary of DMIS commands for basic geometric features, or the part programs can utilize the DMIS commands for general curves and surfaces, "FEAT/GCURVE" and "FEAT/GSURF," respectively. The specific set of DMIS commands used in the part program dictates the format of the inspection results, as well as the result analysis. For example, the DMIS command to define a 2-D circle, "FEAT/CIRCLE," generates a measurement output that can include the coordinates of the circle center point and the diameter of the circle, among other entities. The DMIS command that defines a feature as a general curve, "FEAT/GCURVE," on the other hand, generates a measurement output that contains the 3-D coordinates of each point measured on the feature. Contoured surface parts must be inspected on a point-by-point basis using the DMIS command for individual point identification, "FEAT/POINT," or using the DMIS "FEAT/GCURVE" or "FEAT/GSURF" commands. All three of the DMIS commands just mentioned produce a DMIS output file of inspection results that contains the x, y, and z coordinate values of each point measured on the part.

6.5.4. DMIS Part Programs

The inspection system at NIST can produce DMIS part programs for prismatic parts that implement the commands for features from the DMIS vocabulary (see Table 6.1). For contoured surface parts our system has the capability to generate DMIS programs that produce individual data point output only using the "FEAT/POINT" function. Within our system we have the capability to conduct detailed analyses of such individual data point results, both graphically and numerically through algorithmic fitting routines, using the Qualstar metrology-based analysis software. At this time, splines representing contoured

surfaces can be directly imported into Qualstar from their IGES entities, while simple geometric entities can be constructed in the software using an on-board editor.

TABLE 6.1

DMIS Vocabulary of Feature Definitions

FEAT/ARC	Format 1
FEAT/ARC	Format 2
FEAT/CIRCLE	
FEAT/CONE	
FEAT/CYLNDR	
FEAT/ELLIPS	
FEAT/GCURVE	*
FEAT/GSURF	*
FEAT/LINE	
FEAT/PARPLN	
FEAT/PATERN	
FEAT/PLANE	
FEAT/POINT	
FEAT/RAWDAT	
FEAT/RCTNGL	
FEAT/SPHERE	

* NOTE: These feature definitions are intended for use with contoured surfaces, rather than prismatic features.

6.5.5. Further DMIS Command Development

We are working closely with our industrial partners to support the ability to generate individual point inspection results using the commands for general curves and general surfaces, in addition to the DMIS command for points. This capability is extremely useful for part inspection because it allows the operator to forego the procedure of digitizing every point to

be inspected on a feature and/or part; in other words, the DMIS commands "FEAT/GCURVE" and "FEAT/GSURF" facilitate individual data point results while optimizing the time required to produce an inspection path.

6.5.6. Limitations

The limitations of the inspection system can be divided into two categories: fundamental and application-dependent. A fundamental limitation of this system is set by the scope of the various national standards. IGES version 4.0 does not associate tolerancing information with geometric features. Thus, it is necessary to input the tolerances during the inspection path generation or analysis steps. The Standard for the Exchange of Product Model Data (STEP, and formerly known as PDES), which will eventually supersede IGES, will directly associate the tolerances with their geometric features [19]. This would allow the tolerancing information to come directly from the CAD file and increase the efficiency of the system.

The scope of the DMIS standard supported by the various application programs represents a more immediate limitation. Since the DMIS standard is intended to be applicable to a wide range of automated dimensional measuring equipment, few applications support all the statements within the standard. Consequently, a particular application program may not contain a desired command or definition. In our system, an example of this is the inability to calibrate or change the CMM probe directly from DMIS commands. While the system can make allowances for different positions of a probe by redefining the probe index within the part program, DMIS has no means of changing the physical type of probe sensor the CMM utilizes. In other words, DMIS has the capability to manipulate one probe head into many different probing angles within a part program, but no way to actually change, for example, that probe from a mechanical touch-trigger probe to a touch-trigger probe of different tip radius, or to a piezoelectric probe, should one of these be required. The NIST system does have hardware in the presence of a Renishaw automatic probe changer rack that allows the CMM to physically change the type of probe it uses. In practice, we execute a short calibration program written in the CMM's native language before starting the CMM inspection run. Since the probe calibration program is relatively short, and does not usually change

between part inspections, in contrast to part programs which presumably change often, this is only a minor inconvenience. Similarly a probe can be exchanged in the auto change rack, but this command is in the native CMM language.

6.5.7. Present Status of Applying DMIS

The usable DMIS statements which can be executed in a fully automated manner are those common to the various program vocabularies. Even if all the implemented DMIS applications contain common DMIS terms, there is still a possibility that the interpretation of these terms may be different. This is largely due to the fact that the DMIS standard is less than one year old, and like all new standards, is undergoing a stabilization period. We do not mean to overemphasize these, as we were able to initiate CAD-to-CMM inspections using DMIS within a week of installing the software. Furthermore, it can be expected that market forces will rapidly expand the scope and number of DMIS applications available.

6.5.8. Future Work

In FY91, and possibly beyond, NIST will continue to work with our industrial partners through the Research Associate (RA) program in an effort to continue to support and promote, as well as enhance, inspection techniques that utilize the DMIS standard. We will work towards developing new aspects of the DMIS standard to produce maximum flexibility. We will also be actively working with the DMIS standards committees and users' groups, providing a third party perspective to the direction of the standard, since NIST is neither a product vendor, nor a true product user.

6.6. SUMMARY

An inspection system for manufactured parts has been implemented at NIST that uses two national interface standards to integrate CAD/CAM software products with a CMM. This automated system integrates CAD data bases and software result analyses into part inspections to produce a closed-loop process. The system, which is controlled entirely from

one enhanced PC 386-class computer, demonstrates the use of the DMIS standard to facilitate part inspections. Also demonstrated is the integration of hardware and software products from several different manufacturers into one operational system. The different manufacturers have become involved with this system and research through NIST's RA program.

The system, while fully operational, is not complete. This is due to certain incompatibilities among the different commercial products. The system does, however, possess the capability to inspect and analyze the associated results of many manufactured parts, whether they are prismatic, cylindrical or contoured surfaces.

The DMIS standard effectively enables communication between CAD systems and dimensional measuring equipment. The inspection system at NIST utilizing this standard can produce part inspections integrating DMIS instructions an order of magnitude faster than is possible by traditional, manual programming. The part programs that contain these DMIS instructions are produced off-line, without the need for an actual part to be produced. This off-line generation of part programs can occur without taking the CMM out of productive service, which introduces a flexible, multitasking capability to the system. The graphical interface between the CAD system and the CMM greatly reduces the need for detailed knowledge of the DMIS vocabulary since these statements are generated automatically in the correct syntactical form. Although some temporary limitations exist regarding the scope of the DMIS standard that each particular application supports, these should rapidly disappear as DMIS becomes accepted throughout the inspection industry. Future developments in the representation of solid part geometry in IGES and STEP will expand the capability of DMIS-based products.

7. CONCLUSION

M.A. Donmez

7.1. PROGRAM DIRECTION

Development of a closed-loop quality control system for discrete part manufacturing is one of the most challenging research efforts in the manufacturing field. With our Quality In Automation (QIA) program, we have proposed a three-layered control architecture to address this need and have nearly completed the steps required to demonstrate the inner two control layers.

In the preceding sections we have described the work done during the past year in our QIA program. This program is a combined effort of two divisions in the Manufacturing Engineering Laboratory at NIST. It is funded primarily by the U.S. Navy and internal NIST resources. Even though the main goal of the program, to develop and implement a quality control architecture to obtain an order-of-magnitude increase in part accuracy from commercially available manufacturing equipment, is still the same as in the beginning, the operational plans have been modified over the past two years. These modifications are mostly the results of changing resources for the program. We started out with a plan to implement our QIA system to three machining stations--a turning center, a vertical machining center, and a horizontal machining center--in addition to an inspection station. But, due to manpower and equipment shortages we dropped the horizontal machining center from our plans at the end of the first year. The resource shortages are forcing us to concentrate on only one machining station in FY91, the turning center only. Implementing the three control loops for the turning operations is now our first goal. Once this goal is achieved, the methods we develop will be extended to include machining center operations as the second stage of the program. We will describe the current work plan to achieve our first goal in the following sections.

7.2. WORK PLAN FOR FY91

The goals of the activities planned for FY91 are to complete the details of the two innermost control loops (i.e., real-time and process-intermittent control loops) of the QIA architecture for the turning center, as well as to concentrate new efforts on the post-process control loop.

In the real-time (RT) control loop, we will be monitoring the cutting process using position, temperature and ultrasonic sensors, and modifying the tool path, and subsequently the feed rate and spindle speed during cutting to compensate for predicted geometric and thermal errors as well as to minimize the chatter and vibration or optimize surface finish. To achieve this, we will complete the geometric-thermal (G-T) and kinematic model of the turning center, develop a speed/feed regulation system, integrate a tool-setting station into the tool path correction algorithm, and integrate an ultrasonic surface roughness sensor system into the Quality Controller (QC).

In the process-intermittent (PI) control loop, we will develop feature segmentation criteria and develop algorithms to implement feature segmentation. Our goal is to automate the generation of on-machine inspection routines, and incorporate the GT model into the off-line PI data analysis. We will also develop more extensive and structured decision rules for PI-based NC program modifications.

The efforts for implementing the post-process control loop will be concentrated in three main areas: (1) performing closed-loop post-process part measurements on the Coordinate Measuring Machine (CMM), (2) developing a Quality Monitor (QM) to analyze the post-process measurement results and close the post-process control loop, (3) developing a relational Quality Database (QD) to store all the part and manufacturing related information to be used by the QM and the individual QCs.

The work on closed-loop post-process measurements on the CMM has already started during the past year. We will continue this work to implement fully automated post-process part measurements. To achieve this goal we will develop procedures to generate inspection paths based on features created by the segmentation algorithm and analyze the relationships

of datums described in ANSI Y14.5 standard [20] to our part feature analysis. We will perform CMM data analysis to identify feature errors on our test parts, which will be redesigned for initial simplified analysis. We will also continue working with software manufacturers to improve the fluency of translation between IGES and DMIS.

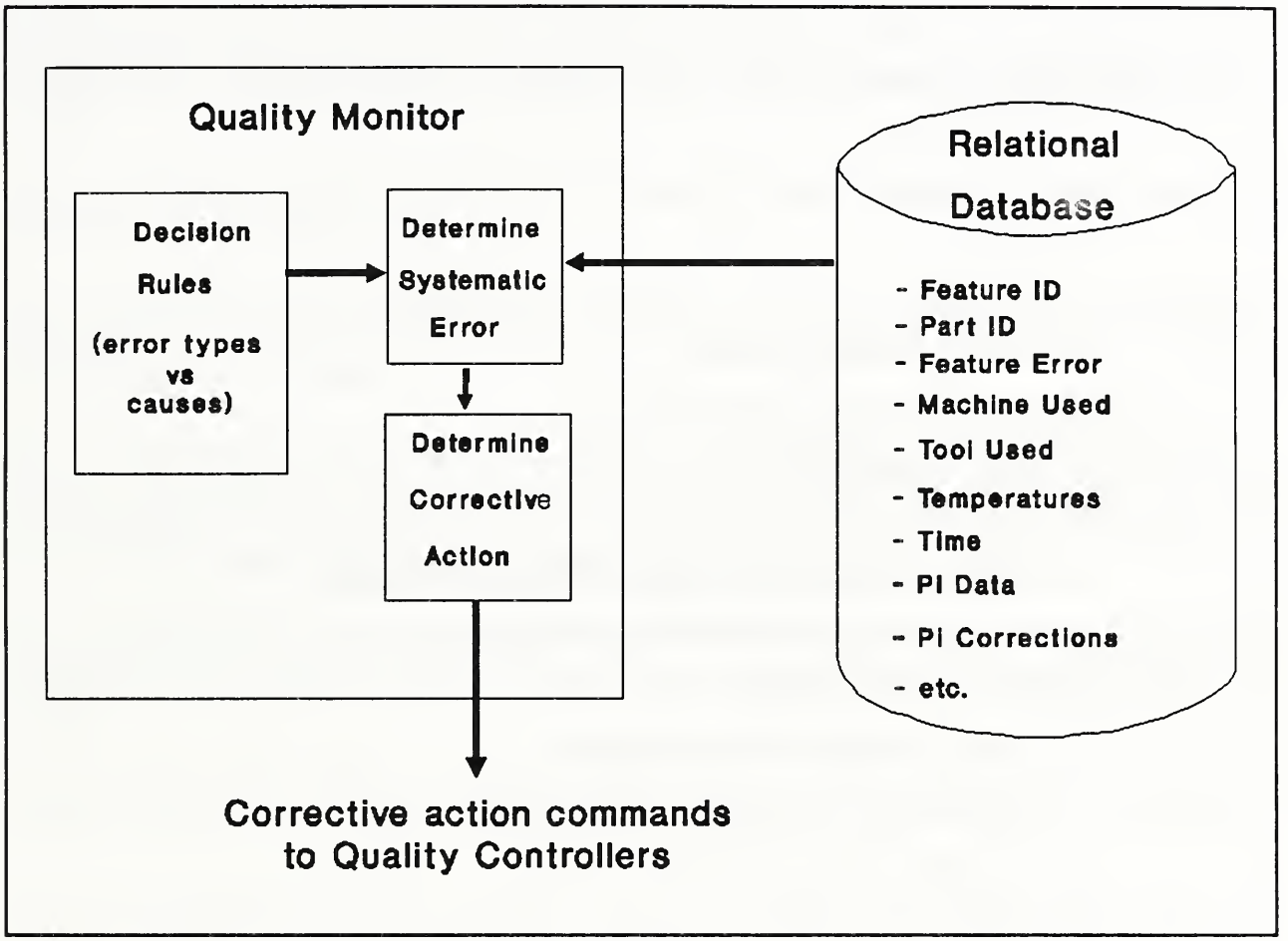


Figure 7.1

The development of the QM requires four major tasks to be completed. These four tasks are: (1) development of decision rules for machine tool diagnostics, (2) development of a Statistical Process Control (SPC) type of algorithms to determine residual systematic errors in the process, (3) development of algorithms to determine the corrective actions to be taken to eliminate these residual errors, and (4) development of a graphics-oriented user interface to extract process information upon request. The interactions between the functional elements of the QM and the QD are shown in Figure 7.1. These tasks will be initiated during FY91, but concluding all of them will not be possible until FY92.

The development of the QD requires the selection of a commercially-available relational database system in the first step. The capabilities of such systems will be investigated based on our needs within the scope of the QIA program. Three candidates are under consideration. Once the system is selected, the database structure will be implemented. Implementation of the database will include at least the following attributes:

I. part-related attributes
    a. feature ID
    b. feature type
    c. part ID
    d. material
    e. measured feature errors
II. time/temperature/history-related attributes
    a. time of machining
    b. elapsed time to complete
    c. machine temperature profile at the time of cutting
    d. ambient temperature at the time of cutting
III. equipment-related attributes
    a. machine used to make the feature
    b. axes involved in producing the geometry
    c. nominal positions along these axes
    d. cutting tools used
IV. process-related attributes
    a. cutting parameters used
    b. PI inspection data
    c. PI modification executed

New attributes may be added to this list as needed by the QM analysis algorithms. The software and the hardware integration of the QD with the QC of the turning center will be completed in FY 91. Further development of the QD will be coordinated with the development of the QM in order to ensure satisfying the data requirements of the post-process control loop.

8. REFERENCES

1. *Progress Report of the Quality In Automation Project for FY88*, C.D. Lovett, ed., National Institute of Standards and Technology (US), NISTIR 89-4045, April 1989.

2. *Progress Report of the Quality In Automation Project for FY89*, T.V. Vorburger and B. Scace, eds., National Institute of Standards and Technology (US), NISTIR 4322, May 1990.

3. F.F. Rudder, Jr. and J.P. Riganati, *Tool-Length Offset: The Machine Tool Environment*, (NISTIR in preparation), 1991.

4. M.A. Donmez, D.S. Blomquist, R.J. Hocken, C.R. Liu, and M.M. Barash, "A General Methodology for Machine Tool Accuracy Enhancement by Error Compensation", *Precision Engineering*, vol. 4, 1986.

5. M.A. Donmez, C.R. Liu, and M.M. Barash, "A Generalized Mathematical Model for Machine-Tool Errors", *Modeling, Sensing, and Control of Manufacturing Processes*, PED-Vol. 23/DSC-Vol. 4, Book No. H00370, K. Srinivasan, D.L.E. Hardt, and R. Komanduri, eds. (The American Society of Mechanical Engineers, New York, 1987).

6. K.W. Yee and R.J. Gavin, *Implementing Fast Part Probing and Error Compensation on Machine Tools*, National Institute of Standards and Technology (US), NISTIR 4447, Oct. 1990.

7. H.T. Bandy, V.E. Carew, Jr., and J.C. Boudreaux, *An AMPLE Version 0.1 Prototype: The HWS Implementation*, National Bureau of Standards (US), NBSIR 88-3770, April 1988.

8. J.C. Boudreaux, "AMPLE: A Programming Language Environment for Automated Manufacturing", *The Role of Language in Problem Solving - 2*, J.C. Boudreaux, B. Hamill, and R. Jernigan eds., (North Holland, Amsterdam, 1987).

9. J.C. Boudreaux, "Requirements for Global Programming Languages", *Proceedings of the Symposium on Manufacturing Application Languages*, MAPL 1988.

10. R. Wilensky, *LISPcraft*, (W.W. Norton, New York, 1984).

11. D.M. Betz, "An Xlisp Tutorial", *BYTE*, March 1985.

12. D.M. Betz, "XLISP: An Object-oriented Lisp, Version 1.7", unpublished manuscript, June 2, 1986.

13. J.C. Boudreaux, *AMPLE Core Interpreter: User's Guide*, National Institute of Standards and Technology (US), NISTIR 4388, September 1990.

14. J.C. Boudreaux, "A Prototype Programming Language Environment for Process-Intermittent Inspection", *Progress Report of the Quality in Automation Project for FY89*, T.V. Vorberger and B. Scace eds., NISTIR 4322, May 1990.

15. "DMIS 2.1," Specification CAM-I Standard 101 (CAMI, Inc., Arlington, TX, 1990).

16. *Initial Graphics Exchange Specification (IGES) Version 4.0*, National Bureau of Standards (US), NBSIR 88-3813, June 1988.

17. NIST Research Associate Program information folder (U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, 1988).

18. "NAS979", National Aerospace Standard 979 Specification, (Aerospace Industries of America, Inc., Washington, D.C., 1969).

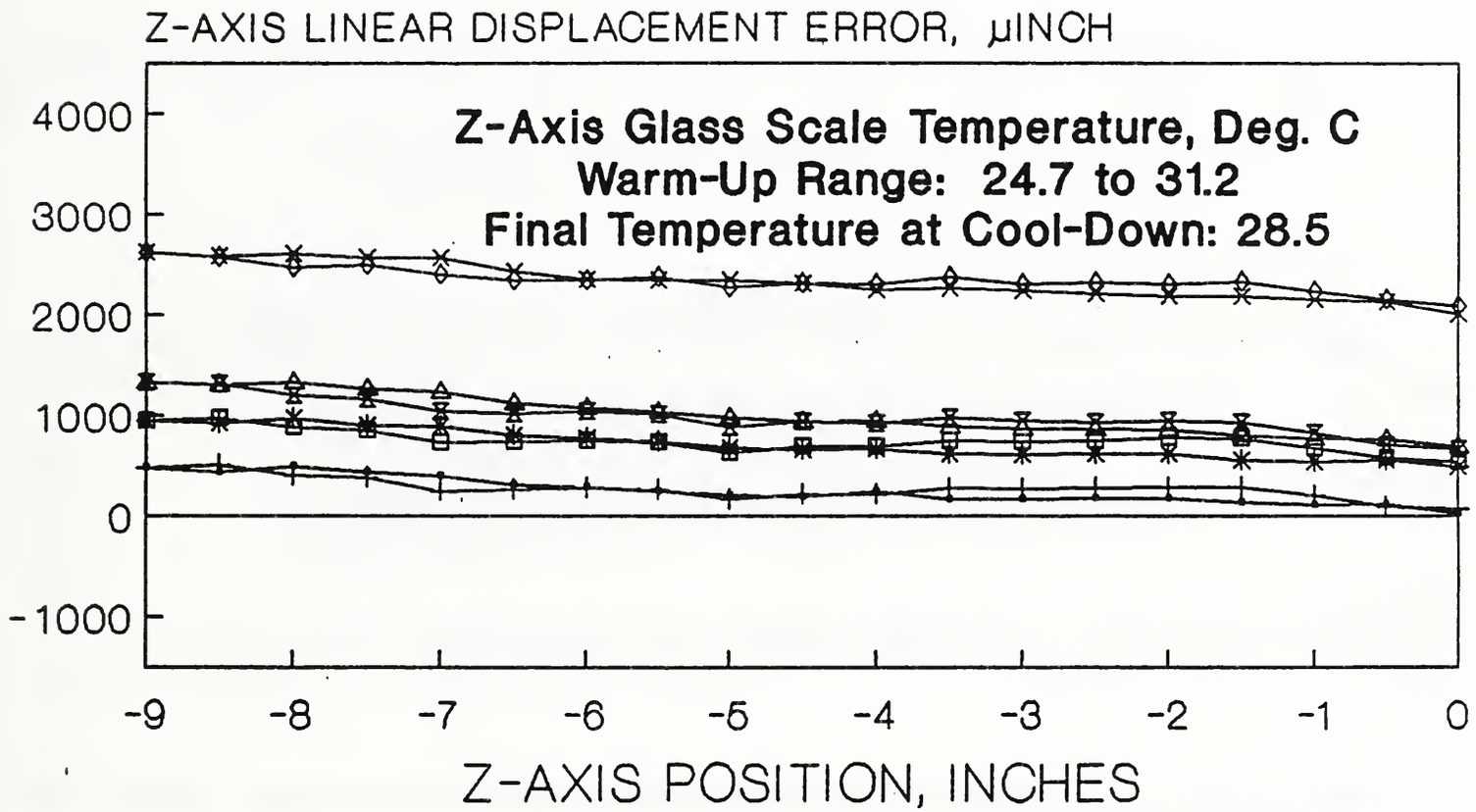
19. B. Smith, *Product Data Exchange: The PDES Project - Status and Objectives*, National Institute of Standards and Technology (US), NISTIR 89-4165, September 1989.

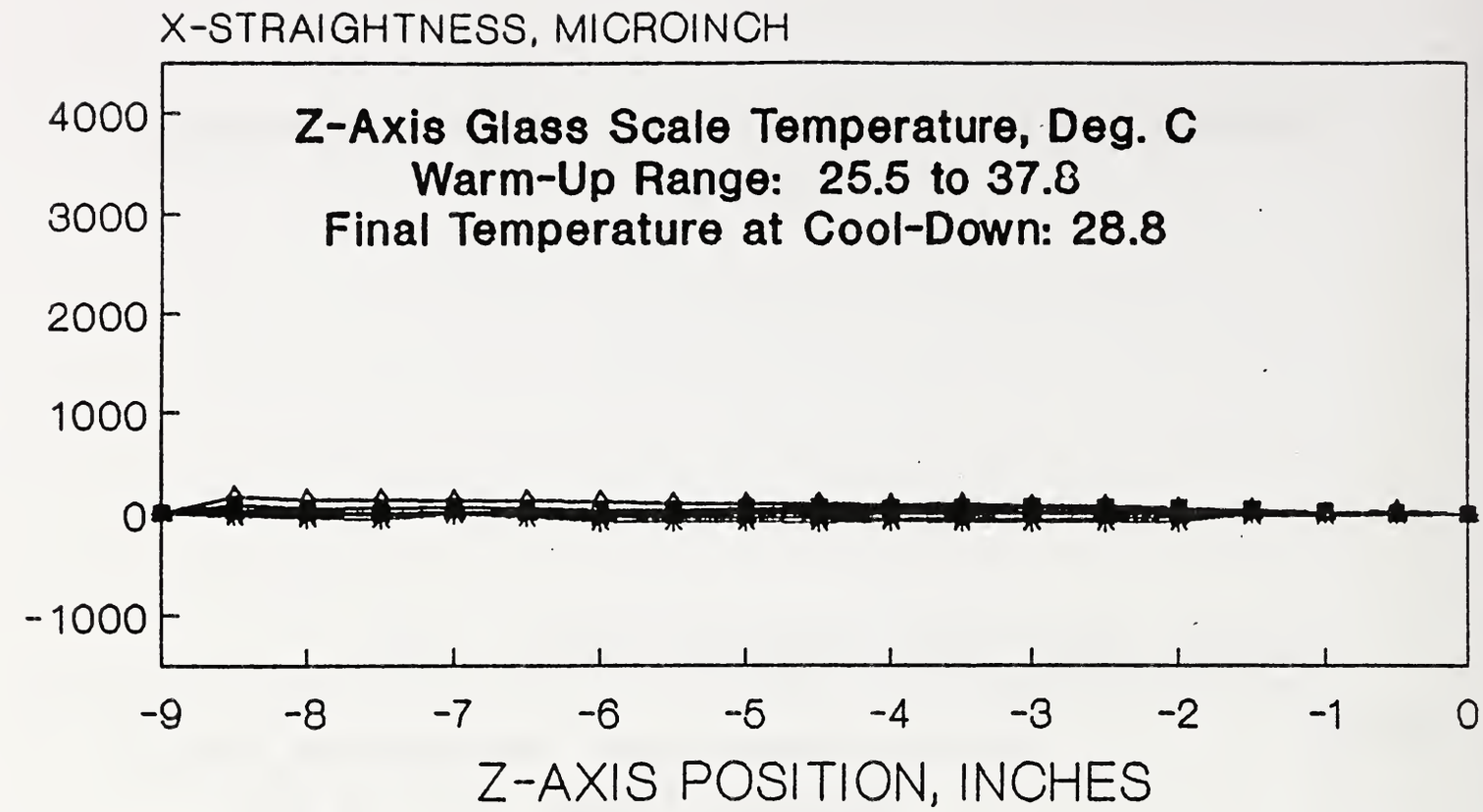
20. *Dimensioning and Tolerancing*, ANSI Y14.5M-1982, (The American Society of Mechanical Engineers, New York NY, 1982).

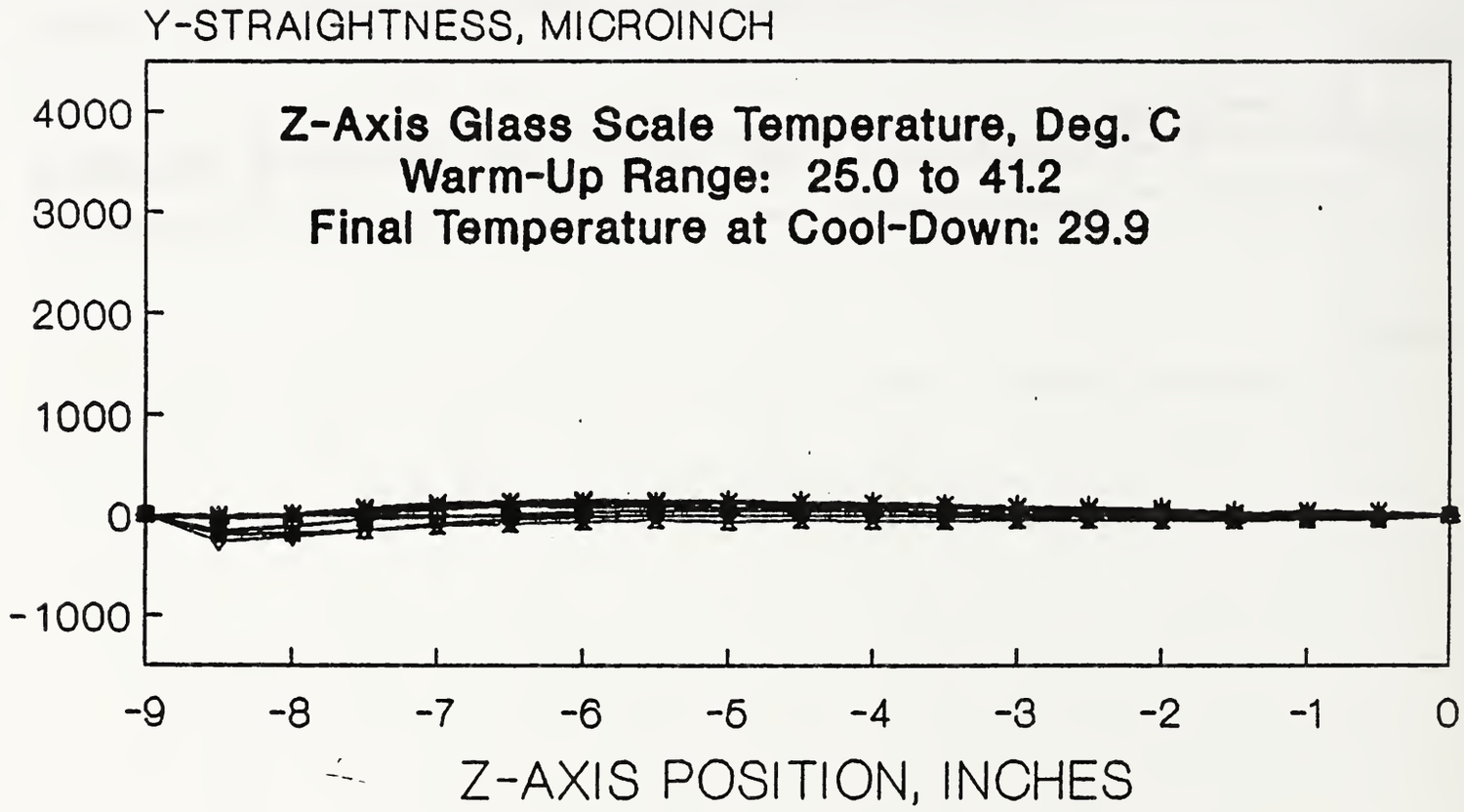
21. G.L. Steele, Jr., *Common Lisp: The Language*, (Digital Press, Billerica, MA, 1984).

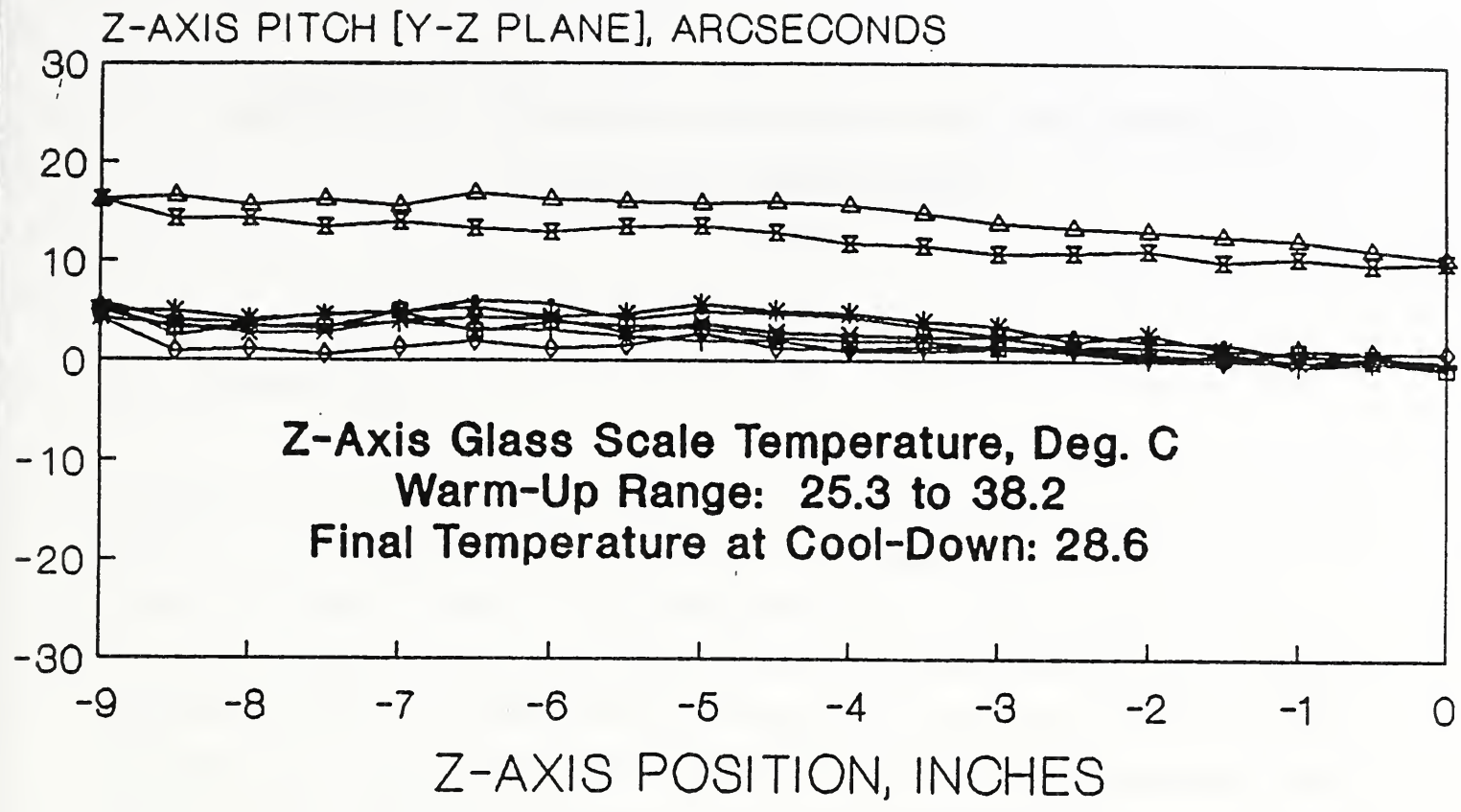
APPENDICES

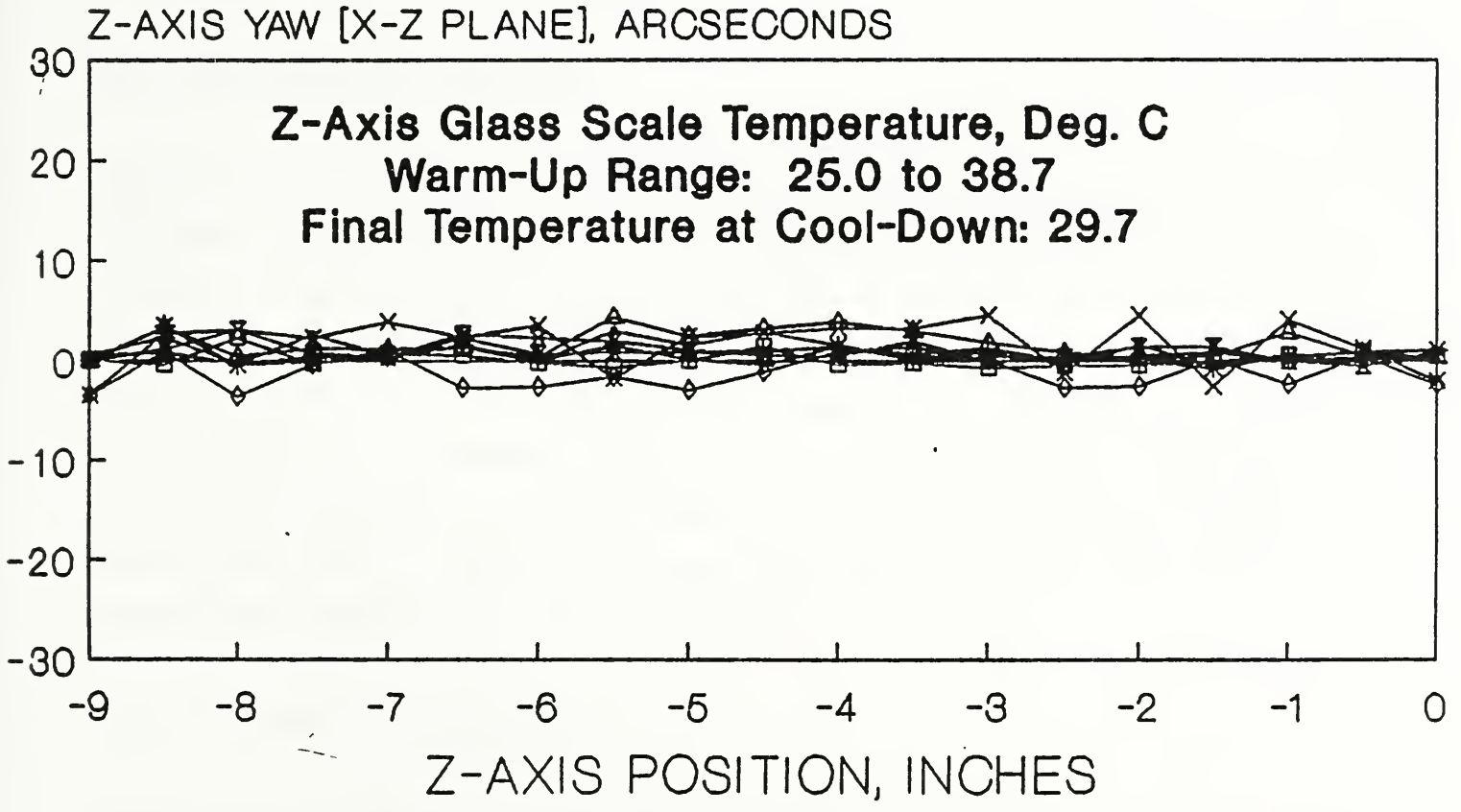
APPENDIX A. Z AXIS ERROR PLOTS OF THE VERTICAL MACHINE CENTER




X-STRAIGHTNESS, MICROINCH
Z-Axis Glass Scale Temperature, Deg. C
Warm-Up Range: 25.5 to 37.8
Final Temperature at Cool-Down: 28.8
4000
3000
2000
1000
0
-1000
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
Z-AXIS POSITION, INCHES


Y-STRAIGHTNESS, MICROINCH
Z-Axis Glass Scale Temperature, Deg. C
Warm-Up Range: 25.0 to 41.2
Final Temperature at Cool-Down: 29.9
4000
3000
2000
1000
0
-1000
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
Z-AXIS POSITION, INCHES


Z-AXIS PITCH [Y-Z PLANE], ARCSECONDS
30
20
10
0
-10
-20
-30
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
Z-Axis Glass Scale Temperature, Deg. C
Warm-Up Range: 25.3 to 38.2
Final Temperature at Cool-Down: 28.6
Z-AXIS POSITION, INCHES


Z-AXIS YAW [X-Z PLANE], ARCSECONDS
30
20
10
0
-10
-20
-30
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
Z-Axis Glass Scale Temperature, Deg. C
Warm-Up Range: 25.0 to 38.7
Final Temperature at Cool-Down: 29.7
Z-AXIS POSITION, INCHES

APPENDIX B. TECHNICAL REVIEW OF THE PROGRAMMING LANGUAGE ENVIRONMENT

J.C. Boudreaux

B.1. INTRODUCTION

This appendix is divided into three sections. The first section gives enough of an overview of **amcore** to allow readers, who are not familiar with Lisp dialects, to interpret the code presented in Chapter 5 above. The account given below is a brief selection of material which is presented more thoroughly in [12]. The second section describes the methods to be followed to add new primitive functions to **amcore**. By studying this section, the reader will acquire a good feel for the overall organization of the Microsoft C implementation itself. The third section is an exercise in **amcore** coding which defines the functions needed to take snapshots of the symbolic environment.

B.2. AN OVERVIEW OF AMCORE

Amcore is a small subset of Common Lisp [14], adapted to the specific requirements of automated manufacturing [10], [11]. This Lisp dialect and its underlying rationale has been described in [12]. All dialects of Lisp, including **amcore**, distinguish between lists and atomic objects. List are sequences of items, the first of which is called the *head* of the list, and the rest are called the *tail*. These components are accessed by two primitive functions: the car function, which returns the head, and the cdr function which returns the tail. Lists are constructed by applying the primitive function cons to two arguments, the first of which becomes the head of the returned list and the second becomes the tail. There is one headless list, called **nil**. Every other list has one and only one head.

B.2.1. The Model of Computation

Amcore is an interpreter whose behavior is defined by a **read-eval-print** loop. Each of these functions is a primitive function of **amcore**. The read function returns the next expression from the specified input stream. This expression is assembled character at a time under the

control of a data structure called the readtable. Once the read operation is properly terminated, the assembled expression is passed to the **eval** function.

- If the expression is a *list*, then **eval** applies the head of the expression, which must be a function, to the items in the tail of the expression, which are the arguments.

Functions are distinguished into two classes, depending upon the manner in which the argument expressions are handled. For the largest class of functions, called SUBRs, each of the arguments is fully **eval**uated before the function is applied. For other functions, called *special forms* or FSUBRs, the arguments are passed without prior **eval**uation.

- If the expression is an *atom*, then it **eval**uates to itself.

After the **eval** function finishes its work, the returned value is **print**ed to the designated output stream. The following transcript illustrates the behavior of this loop:

```
> (* 2 3)
6
```

Special characters, called **read macros**, may be inserted in the input stream to modify the operation of the read function. The **quote** function is used to prevent the evaluation of its argument, which means that the value of a **quote**d expression is the expression itself:

```
> (quote (+ 2 3)) ;using the QUOTE function
(+ 2 3)
> '(+ 2 3)        ;using the equivalent read macro
(+ 2 3)
```

This session also illustrates the use of the *semicolon* macro to indicate the start of a comment. The scope of this macro extends to the next end-of-line character.

B.2.2. Symbols

Symbols are atomic objects, but unlike such atoms as integers or floating point numbers, they may be used in different ways. At any time during an interactive session, a finite and clearly defined aggregation of symbols have each been assigned a concrete meaning. From an operational point of view, the meaning of a symbol is whatever non-error value is returned when that symbol is **eval**ed, or, equivalently, the non-error value **print**ed when the symbol is entered at the **amcore** prompt. If the symbol has no current meaning, then the result of **eval**ing it is an error message, saying that the symbol is unbound. At any time, the collection of all meaningful symbols is called the *(current) environment*. Throughout the progress of an interactive session, the environment will usually change many times. In Version 1.0, the environment is represented by the **amcore** object *obarray*, which is in effect a symbolic hash table.

Symbols may be used to designate *variables*. Variables point to a location in store and may be assigned a value. For example, **eval**ing the expression

```
> (setq X  #C(1.0 1.0))
```

causes the symbol X to be entered into *obarray* and assigns X the value #C(1.0 1.0)).

Symbols may also be used to designate *functions*. When the interpreter is presented a list whose head is a function symbol, then it applies that function to the tail of the list which consists of the arguments. The evaluation of such an expression will succeed only if the symbol is either a primitive function or has been assigned a value by a legal function definition. Function definitions in **amcore** are the work of the primitive special function defun:

```
> (defun cube (x) (* x))
CUBE
```

The first entry of this session defines cube which requires one and only one argument, here represented by the symbol x. The body of the function says that the value returned by

applying cube to x is the number obtained by multiplying x by itself three times. The symbol x is a *parameter*, that is, whenever the expression

```
> (cube 2)
```

is **eval**ed, the current environment is temporarily modified by entering the symbol x, as a so-called *lambda-bound* variable, and assigning it the value 2. Then the body of the function is **eval**ed in order and the final valued so obtained is returned as the value of the function. After this value is returned, the added parameters are dropped from the environment.

Symbols may be used within complex data structures called *property lists*. A property list is a list of attribute-value pairs. An attribute-value pair may be thought of as a list whose first member is a symbol, called the attribute, and whose second member is any bonafide object. For example, the following expressions:

```
(putprop 'a-sphere 'diameter 3.75)
(putprop 'a-sphere 'unit 'inch)
```

say that the diameter of a-sphere is 3.75 inches. The getprop function permits access to the value of an attribute:

```
> (getprop 'a-sphere 'unit)
INCH
> (getprop 'a-sphere 'diameter)
3.75
```

If the attribute specified by **getprop** is not among the attributes of the symbol, then **nil** is returned. The property-list construction provides a special environment within which symbols can be chained together to form very complicated patterns: attribute symbols can have property lists, whose attribute symbols can have property lists, and so on.

B.2.3. Special Forms

The cond special form is the primary branching construct of **amcore**. The arguments of the **cond** expression are called **cond** clauses, each of which is a list whose head is called the *test expression*, followed by a list of zero or more tail expressions. The **eval** function processes each clause in order, selecting for further evaluation the first clause whose test expression evaluates to a non-**nil** value.

```
(defun adjoin (item bag)
  (cond ((member item bag) bag)
        (T (cons item bag))
  )
)
```

That is, if item is a member of **bag**, then adjoin will return the value bag unchanged. Otherwise, adjoin returns a list whose head is item and whose tail is bag. Note that the test expression of the second clause is T, which is a constant symbol designating an arbitrary, but fixed,non-**nil** value.

Function definitions can also be constructed *recursively*, that is, legal definitions may contain occurrences of the function being defined. For example, the function member uses recursion to determine if item is a member of bag:

```
(defun member (item bag)
  (cond ((null bag) nil)
        ((equal item (car bag)) bag)
        (t (member item (cdr bag)))    ;the RECURSION step
))
```

This function is a simplified version of the primitive function member, which was used above.

There are several mechanisms in **amcore** for performing *iterative* calculations. One is the prog special form, which allows for the declaration of local variables and for the use of symbols as labels. Iteration may also be done using the do special form which consists of three components: an arbitrary number of loop control variables, a loop test clause, and then a sequence of zero or more expressions which is analogous to a prog body. Each loop control variable is declared by specifying an initial value and a next-value, or update, function. If the head of the test clause is non-**nil** ,then the loop test is said to be activated. In this case, the remaining expressions in the test clause are evaluated and the value of the last of these expressions is returned. If the loop test is not activated, then a value returned by the do loop only if it is the value of a return function in the loop body.

```
(defun reverse (lst)
   (do ((x lst (cdr x))
        (y nil (cons (car x) y)))
       ((null x) y))
)
```

Notice that this do loop manages to return the correct value, and in fact will always do so if applied to a list, even though the prog body is **nil**.

B.3. ADDING NEW PRIMITIVE FUNCTIONS TO AMCORE

The quality and reliability of an interpretive system depends entirely upon the quality of the underlying interpretation. Building an interpreter is a difficult task in the best of cases, but the structural properties of **amcore** demanded even higher standards of rigor: a strictly regimented set of fundamental design principles, each of which is clearly and distinctly articulated. The single most important principle is that every **amcore** object belongs to one and the same type, namely, the type NODE. The fact that every **amcore** object is a NODE means the addition of new primitive functions needs to be done with some care. The following function, written in Microsoft C, illustrates the kind of care that has to be taken.

```
/* block 1: the preamble */
NODE *xtimed_wait(args)
  NODE *args;
{
   union REGS;
   unsigned long us_delay;
   us_delay = (unsigned long)xlmatch(INT,&args)->n_int;
   xllastarg(args);

/* block 2: the body */
  regs.h.ah = (char)0x86;
  regs.x.dx = (unsigned short)(0x0000ffff & us_delay);
  regs.x.cx = (unsigned short)((0xffff0000 & us_delay) / 0x10000);
  int86(0x15,&regs,&regs);

/* block 3: the postscript */
  return(cvfixnum((FIXNUM)us_delay));
 }
```

The function xtimed_wait is the C implementation of the undocumented **amcore** function timed-wait. When this function is applied to an integer argument, it returns after a delay of that number of microseconds. Most C programmers will recognize that the intended effect of the body of xtimed_wait is to call ROM-BIOS Interrupt 15Hex, Service 86Hex. Other pieces of this code in what I have called the preamble and postscript make use of constructs which are defined in the implementation of **amcore**, and thus require some auxiliary explanations.

First, the function header says that xtimed_wait as well as the arguments to which it is applied are pointers to entities of this type. Second, the effect of the C functions xlmatch and xllastarg in the remainder of block 1 is to obtain the C integer value within the **amcore** integer NODE passed as an argument to timed-wait, and then to make sure that this argument is in fact the last argument provided. One of the most important functions of the preamble is to associate excavated C values with local C variables. In this instance, the single excavated

value is assigned to the variable us_delay. Third, the effect of the function cvfixnum in what I have called the postscript is to create a integer NODE whose integer value is set to us_delay, and then to return a pointer to the newly created NODE as the value of the function itself.

The function of the postscript block is to use the local variables introduced in the preamble to construct the **amcore** NODE which represents the returned value of the function.

After the function has been prepared, only one more step remains, namely, the addition of the function the **amcore** symbol table *obarray*. I call this step the *AMPLE promotion* of the function.

B.4. RE-ENTRANT AMCORE AND SNAPSHOTS

Being able to take a *snapshot* of **amcore** means being able to restart **amcore** in the same state even after exiting the system and returning to the MS-DOS level. The state of **amcore** may be entirely specified by a list of all user-defined symbols and the value assigned to them. This list, represented by the global variable *savelist*, is created by the following function:

```
(defun make-savelist ()
 (setq *savelist* nil)
 (do* ((i 0 (1+ i)))
     ((= i 199) (terpri) T)
     (setq *savelist*
       (append *savelist* (symbol-filter (aref *obarray* i))))
))
```

The next function drops from *savelist* all symbols known to be present whenever amcore is operational. These symbols are of several kinds: symbols like *unbound* which have exotic properties and thus need extra work; special symbols, including those used in **amcore** object processing and those used as type indicators; and primitive function symbols. Primitive symbols may be identified as the only symbols which have the types :SUBR or :FSUBR. The easiest way to deal with special symbols is place them in a special symbol table, and then to

define a function special-symbol? which returns T if and only if its argument is a member of that table. All symbols which do not fall in these categories are assumed to be user-defined and should be added to *savelist*.

```
(defun symbol-filter (l)
 (cond ((null l) nil)
       (T
        (do ((sym (car l)(car l))
             (ret nil))
            ((null l) ret)
            (cond
             ((or
               (equal (car l) '*UNBOUND*)
               (not (boundp sym))
               (equal (type-of (eval sym)) ':SUBR)
               (equal (type-of (eval sym)) ':FSUBR)
               (special-symbol? sym)
              ) T)
             (T
              (setq ret (cons sym ret))))
            (setq l (cdr l))
       )))
)
```

Once *savelist* has been created, the function save, which must be given a filename argument, writes to that file all of the symbols in *savelist*.

```
(defun save (filename)
 (setq *savelist* (cons '*SAVELIST* *savelist*))
 (do* ((fp (openo filename))
       (sym (car *savelist*)(car *savelist*))
       (plist (symbol-plist sym)(symbol-plist sym)))
```

```
    ()
    (princ (strcat "(setq " (symbol-name sym) " '") fp)
    (print (eval sym) fp)
    (princ ")" fp)
    (terpri fp)
    (cond (plist
          (princ (strcat
                "(setf (symbol-plist "
                (symbol-name sym) ") '")
               fp)
          (print plist fp)
          (princ ")" fp)
          (terpri fp)))
    (setq *savelist* (cdr *savelist*))
    (cond ((null *savelist*)(close fp)(return T)))
))
```

The symbol *SAVELIST* is consed onto *savelist* because that allows us to open the file, do one read operation, and see all of the symbols which the file contains. Having saved a snapshot, the subsequent restoration of that same symbolic environment is accomplished by LOADing the saved file into the new *AMPLE* session. For the user's point of view, the effect is the same as if all of the functions had been re-keyed.

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4536

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
MARCH 1991

4. TITLE AND SUBTITLE

Progress Report of the Quality in Automation Project for FY90

5. AUTHOR(S)

A. Donmez, Editor

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

U.S. Navy

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

This document describes the progress of the Quality In Automation (QIA) program of the Manufacturing Engineering Laboratory (formerly the Center for Manufacturing Engineering) of the National Institute of Standards and Technology (NIST) for fiscal year 1990. The purpose of the QIA program is to develop a quality control and assurance system that exploits deterministic manufacturing principles in small-batch automated manufacturing using commercially available and affordable equipment. The foundation of the program is a quality-control architecture that uses multiple feed-back loops to control the process. Currently, we are implementing three control loops: real-time control, process-intermittent control, and post-process control loops.

During fiscal year 1990 work concentrated on the real-time and the process-intermittent control loops. The Real-Time Error Corrector (RTEC) was integrated into the real-time control loop. Its ability to modify the tool path--without any intrusion into the machine tool's CNC controller--during cutting was demonstrated for the first time. High-speed, on-machine part inspection using the fast-probing capability of the RTEC was also demonstrated as an integral part of the process-intermittent control loop. Preliminary testing has been done to evaluate the methodology for using errors determined by process-intermittent gauging, i.e., fast probing, to alter the tool path of the finishing cut by modification of NC part-program coordinates. Coordinated operation of both the real-time and the process-intermittent control loops has been demonstrated under the supervision of the turning center's Quality Controller (QC) running in a high-level language environment. As a part of the post-process control loop, an automated inspection system using the Dimensional Measurement Interface Specification (DMIS) and the Initial Graphics Exchange Specification (IGES) national interface standards to integrate Computer Aided Design/Computer Aided Manufacturing (CAD/CAM) software with a Coordinate Measuring Machine (CMM) has become operational.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

Quality control; machine tools; automated manufacturing; probing; inspection; real-time control; error compensation; DMIS; IGES; CMM; CNC

13. AVAILABILITY

XX	UNLIMITED
	FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
	ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
X	ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
98

15. PRICE
A05

ELECTRONIC FORM