NISTIR 5797

PIECS - A Software Program for Machine Tool Process-Intermittent Error Compensation

Herbert T. Bandy
David E. Gilsinn

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

NIST

NISTIR 5797

PIECS - A Software Program for Machine Tool Process-Intermittent Error Compensation

Herbert T. Bandy
David E. Gilsinn

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

March 1996

U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

Abstract

This report documents software, called PIECS, that performs process-intermittent error compensation for a turning center. The program is a part of a larger three loop control architecture that includes a real-time geometric-thermal error compensation loop and a post-process loop. In process-intermittent error compensation, a part is measured by on-machine gauging after a semifinish cut which uses the same cutting parameters (speed, feed, and depth of cut) as are used in the finish cut, to reproduce process-dependent errors such as cutting-force induced tool or part deflection. During gauging a touch-trigger probe signal indicates that the part surface has been contacted. The coordinates of the points are then transformed to the part coordinate system and compared to the corresponding nominal coordinates so that errors may be determined. The error vector is defined as having its head at the measured coordinates of the gauged point and its tail at the nominal coordinate for that point. Since the philosophy chosen in this program is to compensate process-intermittent errors by changing the position and orientation of features, least squares curve fitting through the ends of the error vectors is used to determine the adjusted tool path curve. The compensation curve becomes the tool path for the corresponding feature for the finish cut. The adjusted position and orientation of the feature are thus determined. Based on this information, errors may be compensated using either of two options: (1) an error specification file may be used to compensate the detected errors in real time by adjusting the machine-tool servo commands or (2) an adjustment of the programmed tool path can be made by changing the part program. The report includes a description of the program algorithm, the input and output data sets as well as descriptions of each of the C-programming language functions that compose PIECS. A listing of the program is included in the appendix.

Key Words: error; error compensation; machine tool; part; part program; process-intermittent inspection; quality control; real-time; tool path

Table of Contents

1.0 Introduction

Over the past several years the Quality in Automation (QIA) project at the National Institute of Standards and Technology (NIST) has developed a quality control and assurance system that exploits deterministic manufacturing principles in small-batch automated manufacturing using commercially available and affordable equipment. The premise behind deterministic manufacturing principles is that most errors in the manufacturing process are repeatable and predictable, and, therefore, can be compensated. The foundation of the project is a quality-control architecture that uses multiple feed-back loops to control the process. The three contol loops that are involved in this architecture are: real-time control, process-intermittent control, and post-process control.

This report documents the software that is the principal component of the process-intermittent control loop. The program is called the **P**rocess-**I**ntermittent **E**rror **C**ompensation **S**oftware (PIECS) and is part of a larger architecture that encompasses the real-time control loop and the post-process control loop. PIECS seeks to perform corrections for complex part dimension and form errors introduced during the semifinish machining of a part. The errors corrected are process-related errors, such as inaccurate tool dimensions, tool wear, tool and/or part deflection, etc. PIECS supports rapid process-intermittent measurements of the machined part by replacing the tool with a touch trigger probe, thereby, transforming the machine tool into a coordinate measuring machine. Compensation is achieved by part program modification or real-time compensation.

The function of the real-time control loop is to monitor the machine tool and the cutting process and to modify the tool path in real-time to achieve higher dimensional and form accuracy, and surface quality. The real-time loop modifies the tool-path through the Real-Time Error Corrector (RTEC), explained in Section 2.0, compensating errors predicted by Geometric-Thermal (G-T) error models of the machine-tool. The models are derived from the structure of the machine and pre-process characterization of the machine. These characterization procedures measure such motion related errors as axis positional errors, axis straightness errors, roll, pitch and yaw errors of the carriages along the axis slideways, and the squareness and parallelism errors. These errors are machine axes position and temperature dependent.

The post-process control loop is used primarily to verify and optimize the performance of the other two control loops. A finished batch of parts is measured on, for example, a coordinate measuring machine (CMM) to determine residual errors in part features. Systematic errors in specific features are used to modify the algorithms used in the process-intermittent loop, or, in the long term, to modify the parameters in the G-T model. This control loop can also be used to adjust the pre-process characterization of the machine to take into account the effects of process related machine functions such as the use of coolant.

The PIECS software described in this report compensates for the process-related errors by automatically modifying the tool-path specifications (such as coordinates, tool offsets, etc.). These modifications are made to a file of tool-path adjustment data that is used by the RTEC

during the machining of the finish cuts.The software also provides a modified NC part program for the finish cuts for those machining centers that do not have the capability of the RTEC. The procedures supporting the modifications depend on establishing the nominal geometry of the desired part, usually through a computer aided design (CAD) of the part. Design feature information, on-machine inspection points, inspection probe information are all automatically stored in files that are passed to the architecture of PIECS. The differences between the part dimensions gauged by a touch-trigger probe between machining passes, and the corresponding nominal dimensions, are analyzed and weighted to produce tool-path adjustments for the subsequent machining pass. The NC controller uses coordinates that are modified on the basis of error patterns observed from on-machine probing in order to make the tool traverse a path significantly closer to the ideal path.

Although the methodology detailed in this report is described relative to turning processes, nearly all of the principles apply as well to milling. The prototype software documented here has been developed for a turning center and has been designed to accomodate machine-gauge-machine iterations. However, the principles employed in the software are quite general. The program is written in the C programming language.

This report is structured in the following manner. Sections 2 through 5 present the general background necessary to understand the PIECS strategy. The program input files are described in APPENDIX A and the output files in APPENDIX B. The individual program functions are described in APPENDIX C and program listings are given there. A brief description of the software NCMAKE that is used to develop the supporting NC part-probing programs for PIECS is given in APPENDIX D. Finally, the general procedures for developing part data from CAD output are provided in APPENDIX E.

2.0 The Real-Time Error Corrector

Since the RTEC plays an integral part in both the real-time and process-intermittent control loops, this section briefly describes its features. A more detailed description can be found in references 3 and 4.

The RTEC is a generic smart interface that is inserted between the position feedback element of the axes of a machine tool, and the machine-tool controller. The device counts the signals from the feedback element to determine the machine position and sends this position data to a controlling PC computer, called the Quality Controller in the rest of this report, which records the data. There is a quadrature decoder and microprocessor for each axis.

When performing corrections, the RTEC alters the tool-path by modifying the square-wave signals from "encoder-type" feedback devices before they go to the machine tool controller. Additional pulses are added causing the machine axis to go to a slightly different position to compensate for the predicted errors. The correction counts to be added are determined by the Quality Controller using the G-T model along with the current machine temperatures and axis positions. For the machine tool in use, a two-axis turning center (Figure 1), the position is

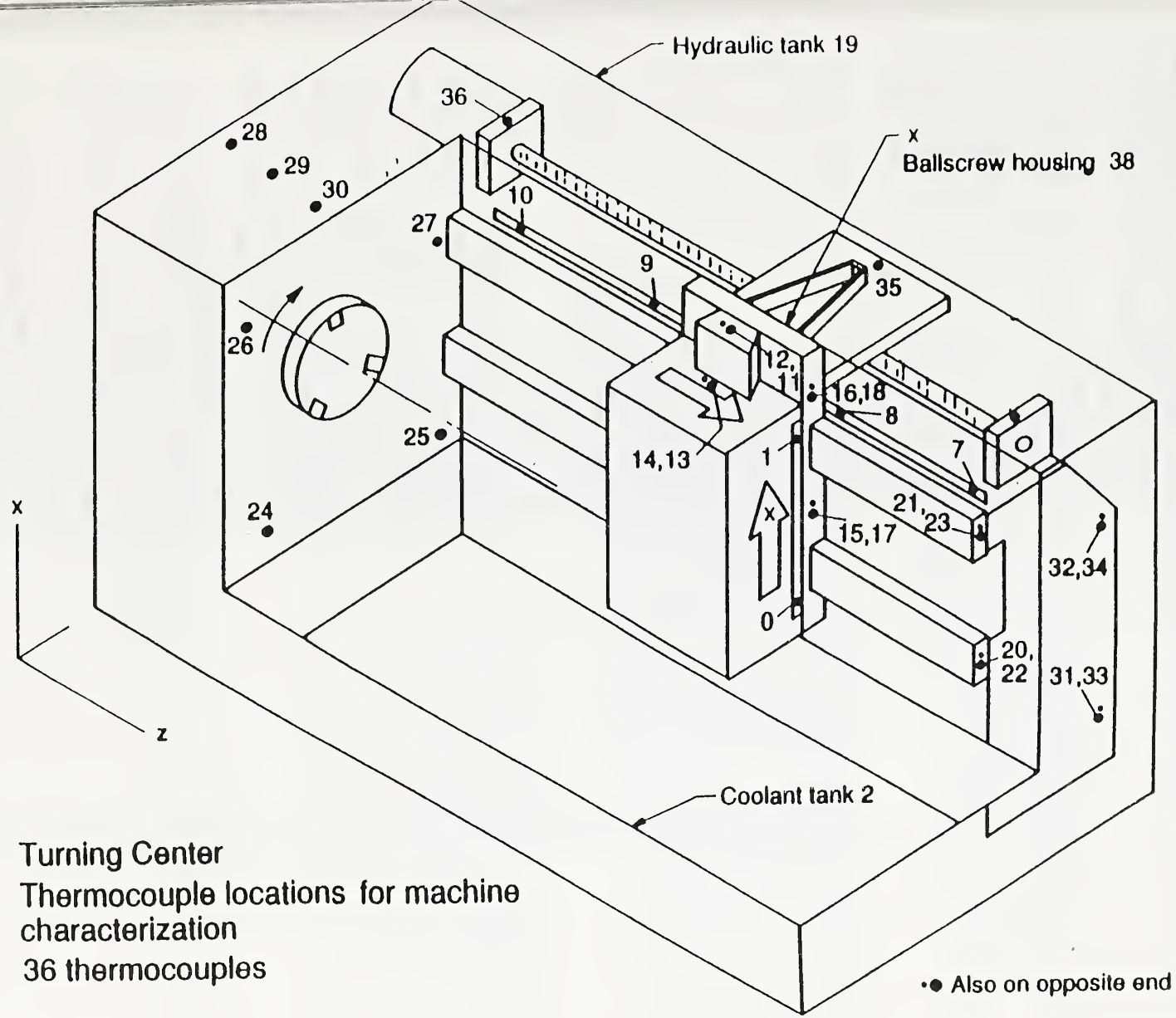

Figure 1 - A schematic of the turning center with thermocouple locations

altered by 2 μm for each correction count. The figure also shows thermocouple locations used in developing the G-T model. These are identified more precisely in Table 1.

Table 1
Thermocouple Locations on the Turning Center

Number	Location
0	Bottom of X-(Glass) Scale
1	Top of X-Scale
2	Coolant Tank
3	Not Used
4	Not Used
5	Not Used
6	Not Used
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 Left 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	Top Right of Bed
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

3.0 On-Machine Process-Intermittent Gauging

Two types of part inspection are used in the overall quality control strategy. The first uses on-machine gauging and the second uses a coordinate measuring machine (CMM). Since the inspection technique used in the process-intermittent loop is on-machine gauging, this section will only describe this method.

Most geometric and probably thermal errors that occur during machining will be reproduced during the probing operation and will not be "seen" (i.e., the same incorrect path is traced by the tool and the measuring probe). Therefore, only process-induced errors, caused by tool dimension, tool wear, tool and/or part deflection, etc., can normally be detected. The part probing operation takes place after a semifinish cut, which uses the same cutting parameters (speed, feed, and depth of cut) as are used in the finish cut.

The on-machine gauging used in the process-intermittent error compensation is called fast-probing. Fast-probing is rapid on-machine gauging using a touch-trigger probe. In fast-probing, the RTEC captures the axis positions in response to the probe trip and sends them to the PC. The coordinates of the point of contact are then calculated with respect to the part coordinate system so that they may be compared to the corresponding nominal part geometry. At the 2.5 m/min feed rate used for probing, the position must be captured within about 25 μs to avoid introducing a position error greater than the position resolution of the turning machine used in this study, which is 1 μm. Most machine tool controllers are not designed to respond to probe trips rapidly enough to allow fast probing, but it is possible using the RTEC.

The RTEC is capable of two modes of operation. It is not operated in its correction mode during process-intermittent gauging, since the RTEC microcomputer is not capable of doing both error compensation and fast probing simultaneously. At each trip of the probe, the RTEC reports the current machine position, but unlike the operation during machining, that position is not corrected by the RTEC according to the G-T model. Therefore, in order to compare the actual location of a surface on the part with the location that was intended as a result of the last machining pass, each position that is registered during gauging must first be adjusted by the model. The *error* is the difference between the ideal dimension and the dimension measurement obtained with the probe.

Touch-trigger probing is not the only process-intermittent gauging method investigated. On-machine gauging by ultrasonic probing has also been used for surface roughness measurement [5]. For the purposes of this report, however, touch-trigger probing for error compensation is the only process-intermittent gauging method discussed.

4.0 Implementation of Process-Intermittent Error Compensation

The implementation of process-intermittent error compensation requires several steps. The PIECS software requires adaptation of part programs, collection of data during dimensional inspection, and advance preparation of process data. Once this data is prepared the PIECS software analyzes the gauging data, modifies part programs and prepares the tool path adjustment file for the RTEC. These steps will be outlined here and expanded on in the sections below.

The part data preparation begins with the development of an archival part program. An initial NC part-program is segmented. An *archival* part program is the version of the NC part program with which the QIA system starts each time a part is made. It functions as a template and consists of special adaptations of the NC part-program segments that are generated by replacing certain tool-path coordinates with variables, and by adding tool-offset commands near the beginning of the finish cut program segment. They are used by PIECS, in conjunction with special data files, to generate usable NC part programs.

In the next step, probe and part information are prepared. The X - and Z -lengths and probe diameter are calibrated. A schematic of the turning center used is shown in Figure 1 where the axes directions are given. Table 1 gives the locations of thermal sensors. A *cut* is defined as the portion of part geometry which is produced by a machining pass of the tool from a position specified in a block of NC code to the next position, specified in the next block. Those blocks are called the "initial block" and "final block", respectively, for that cut. Cuts, as well as the nominal coordinates of the gauging points located along the cut, are defined using a CAD system. The data generated also includes such information as the order in which the gauging points will be measured, the point whose Z-coordinate is to be used as a reference location on the part, the tool-path specifications that are to be compensated, and the nominal shape of each cut and its important relationships to other cuts.

After a process-intermittent gauging session is finished the probed data points are converted to part coordinates. Error vectors are then created from the nominal coordinates of the gauged points to the measured data in the part coordinate system. Interpolating polynomials are fit through these error vectors by least-squares and the polynomials are then used to compensate the errors for each cut feature during the finish cut of the part.

4.1 NC Part-Program Segmentation

In order to insert on-machine inspection routines after both of the semifinish and finish machining, a typical NC part-program is divided into segments. For a sample of a segmented part-program see Section A.3.

The NC part-program is divided into segments in order to provide opportunities for certain logistical requirements of error compensation. The segmentation simplifies such tasks as setup

procedures, running gauging segments multiple times to study error repeatability or for experimental purposes to cut multiple copies of semifinish parts. Within a segment, sequential NC part-program blocks are executed without interruption, but between segments execution is temporarily suspended. Since roughing a part is not of critical interest, the groups of consecutive blocks of NC code pertaining to roughing comprise a single segment. In some cases, semifinish code is contained in the same segment. Otherwise, code for semi-finish and finish machining is contained in independent segments.

If part gauging were not performed, semi-finish and finish maching could be performed in the same segment. However, to insert a part gauging segment, a part programmer locates an instruction in the part program after which gauging is performed. The segment boundary is selected to make sure that this instruction occurs at the end of the semi-finish portion of the NC part-program.

The program NCMAKE, described in APPENDIX D, is a software routine that assists in part-program segmentation. The first step in applying NCMAKE to the segmentation process is to identify convenient places in the part program where execution may be safely interrupted. Since segments are executed one at a time, header and footer blocks are added to each segment as appropriate. The end of each segment is marked by an M30 NC command, which will cause the spindle, coolant and feed to stop. However, the segmentation decisions remain tentative until the requirements of the dimensional and surface inspection sequences, to be inserted as independent segments, are finally defined.

4.2 Development of Dimensional Probing Procedures

In preparation for developing gauging instructions to be sequenced with other segments of the part program, each part surface needing dimensional measurements must be identified as a cut. Figure 2 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-points and end-points of tool paths.

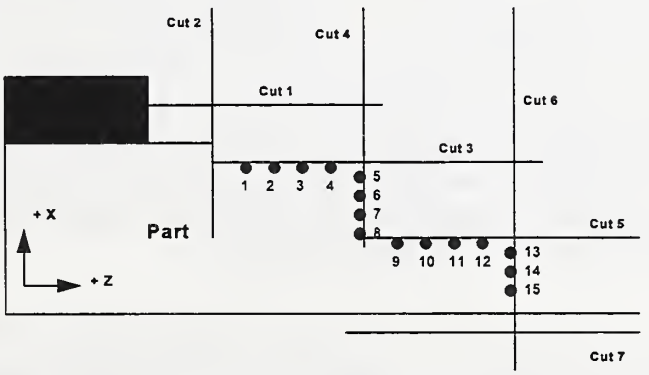

Figure 2 - Sample Part

The following outline describes the steps required to prepare NC part-probing routine segments. CAD-interactive programs facilitate this task.

1. On the basis of the part geometry, one determines the measurement points that will be taken on the specified surfaces and develops a file with the nominal coordinates of each of these points.

2. One next defines the NC part-program segments for making the necessary measurements. The program NCMAKE performs this automatically based on files created with the CAD program. Although the target coordinates of the probe paths are based on the nominal coordinates, one must take into consideration the machine-tool acceleration distance, overtravel limits of the touch-trigger probe stylus and the workholding devices.

3. To complete the part-probing segments, the part programmer adds header and footer blocks to each part-probing segment in order to form independent part-program segments. NCMAKE simplifies this task.

4. Finally, each complete part-probing segment is inserted into its designated place in the sequence of part-program segments. However, instead of actually being concatenated, the sequence is implied by the part program ID numbers assigned to the independent segments.

4.3 Preparation of Part-Probing Data Files

After the machining and gauging NC part-programs have been segmented, part-probing data files are prepared as input to PIECS. This section gives an overview of the tasks involved. A more detailed description of the files is given in APPENDIX A.

The file names are linked to machined parts by a part identification number. Each part is identified by a four digit number NNVV, called the part ID. The first two digits, NN, represent the part number and the last two digits, VV, represent different versions of the part. In particular, the first V of the version number represents the material and inspection point patterns and the last V denotes a multiple of the part. The digit 0 is used in place of a version digit for references which are independent of the pertinent type of version. Therefore, if a file name is specified to contain the four consecutive digits *nn*00, the file applies to any and every version of part *nn*. If the digits are specified as *nnv*0, the file applies to every nominally identical copy of part *nn* which is made with the particular material and inspection point pattern implied by the value of the first version digit. This convention allows producing as many as nine copies (having the second version digit numbered 1 through 9) of a part having the same first three ID digits. This limit has not been a problem for this experimental project.

Once the NC part program for part *nnvv* is tentatively segmented, tool-offset update commands are inserted to create a file PT*nn*0021.NC, where 21 is a code referring to the finish machining process. (Further elaboration on the part and process nonmenclature would exceed the scope of

this report.) Preserving the ".nc" versions of the machining segments as master versions, the QIA archival versions, having symbols for variables in place of tool path specifications, are produced next as input files ptNN0021.in. Then segments containing dimensional gauging instructions are developed and inserted as explained in sections 4.1 and 4.2.

The part-probing data file is then prepared. The X and Z-lengths and diameter of the probing tool are measured and the nominal coordinates of the gauging points are located along cuts. The order in which the gauging points will be measured is established. Finally, the point whose Z-coordinate is to be used as a reference location on the part is identified. These are entered into a file called pramNNV0.ref. Only the first three digits of the part number NNV are significant in this file name. 0 only marks a character position in the file name. The radius of the probe stylus is also a parameter that is entered into the file pramNNV0.ref. It is used , along with the surface normal vector at each gauging point, for an accurate determination of the location where the stylus contacts the part (See Figure 3).

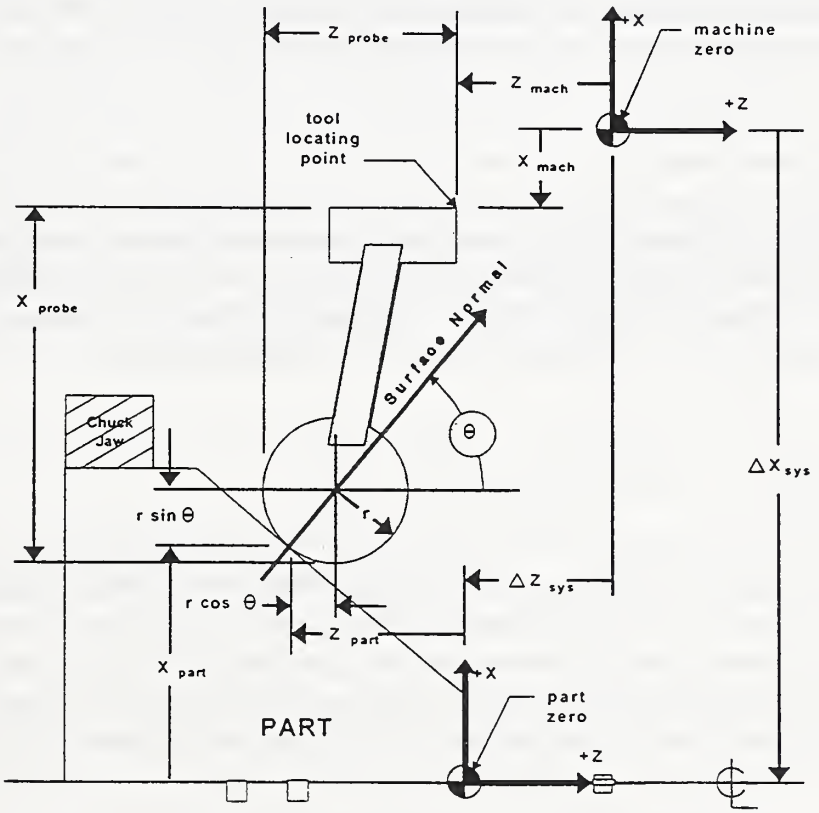

Figure 3 - Determining the Coordinates of a Point Gauged

The tool-path file is next generated. The tool-path specifications that are to be variable are designated in a file pathNN00.2, where the extension 2 refers to the finish cut. The ideal shape of each surface is recorded and the position and orientation of each surface to the other surfaces are also stored as surface attribute data.

Among data to be entered into each file attrNNV0.I (where I = 1 or 2 for the semifinish and finish part respectively) are information concerning each cut containing gauging points, including (1) the direction of the last pass; (2) the surface normal vector at each gauging point; and (3) codes associating part program instructions with particular gauging results.

4.4 Analysis of Gauging Data

After each process-intermittent gauging routine, error terms for all gauging points are calculated. The vector from the nominal coordinates of a gauging point to the measured coordinates of the same point is defined as the error vector. This vector is decomposed into its components along the surface normal and the surface tangent. The term *error* is defined as the signed magnitude (positive or negative) of the component of the error vector that lies in the direction of the surface normal of the gauging point. It is the only component used in the calculations for analyzing the gauging data and for compensation. Although the parallel component of the error vector is also calculated, it is only used as an indicator of gauging problems if it is found to be of significant magnitude.

Error calculations are carried out in several steps. First, using the machine-tool temperatures at the time of the gauging process, the measured coordinates are adjusted by the G-T model. These adjusted gauging coordinates are then converted from the machine-tool coordinate system to the part coordinate system. Figure 3 illustrates the determination of the part coordinates for the probe points. Referring to Figure 3, X_{part} and Z_{part} are coordinates of a gauging point in the part coordinate system which are calculated using the following equations:

$$\begin{aligned} X_{part} &= X_{mach} + r(1 - \sin(\theta)) - X_{probe} - \Delta X_{sys} \\ Z_{part} &= Z_{mach} + r(1 - \cos(\theta)) - Z_{probe} - \Delta Z_{sys} \end{aligned} \tag{4.1}$$

where in the current program $-\Delta X_{sys}$ is called MACHINE_HOME_X (see section C.3.5). It represents the distance in the X-coordinate direction from the part origin to the machine home origin. Since the current turning center controller, upon which PIECS has been implemented, requires that the diameter rather than radius be given, the actual part coordinate is twice the radial value and is designated here by a superscript d. Therefore, for the current turning center:

$$X_{part}^{d} = 2.0\,(X_{mach} + r(1 - \sin(\theta)) - X_{probe} + MACHINE\_HOME\_X) \tag{4.2}$$

A known z-coordinate on the part is used to locate the part coordinate system with respect to the machine system. This enables Z_{probe} and ΔZ_{sys} to be combined into one constant, z_{diff}. From

(4.1) one can write:

$$Z_{diff} = \Delta Z_{sys} + Z_{probe} = Z_{mach} - Z_{part} + r(1 - \cos\theta) \tag{4.3}$$

Note that Z_{diff} is calculated for a specific point but the resulting constant is applied generally.

For each gauging point, the error is next calculated as the difference between an adjusted measured coordinate and the target (nominal) coordinate. The magnitude (positive or negative) of the component of the error vector which lies in the direction of the surface normal of the point of interest is computed. The error vector is equal to the gauged coordinates minus the nominal coordinates (of the semifinish segment).

The compensation vector at a gauging point for the finish part is then generated. This value is computed as the nominal coordinates of a gauge point for the finish machining segment plus the product of the unit normal vector, the negative of the error scalar, and the percent correction.

A correction curve is then fit to the adjusted coordinates for the finish machining segment which creates an adjusted tool path for the finish cut. The equation form is given below as equation (4.4). The fit is accomplished by a least-squares technique and ensures smoothness. Thus, the actual position and the orientation of the measured feature are determined relative to those of the ideal feature.

According to the nominal shape of the surface of interest appropriate coefficients are assigned to the following general polynomial. The correction curve represented by the polynomial becomes the tool path for the corresponding cut in the finish machining segment. The general form for the correction curve polynomial used in PIECS is:

$$C_1 z^3 + C_2 z^2 + C_3 zx + C_4 x^2 + C_5 z + C_6 x - C_0 = 0 \tag{4.4}$$

In the table below an explicit quadratic or cubic function means a quadratic or cubic function with Z only. The general quadratic includes X terms.

NOMINAL CURVE COEFFICIENTS * = VALUES TO BE DETERMINED FOR SPECIFIC CASES							
	C_0	C_1	C_2	C_3	C_4	C_5	C_6
Line	*	0	0	0	0	*	1
Circle	*	0	1	0	1	*	*
Explicit Quadratic	*	0	*	0	0	*	1
General Quadratic	*	*	*	*	*	*	*
Explicit Cubic	*	*	*	0	0	*	1

4.5 Modification of Part Programs

The tool-path adjustment, as calculated above, is used as input to correction functions to calculate new tool-path specifications. There are three cases to consider.

If all features are found to be translated with respect to the ideal one by the same amount, which is indicated by a nearly constant offset error along an axis, a tool offset change in the part program is adequate. 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.

If the measured feature is found to be translated, rotated, and/or deformed without changing its rigid body characteristics (as in a length change, for example) compensation is done by modifying selected end coordinates related to these features. For cases in which the endpoint of one cut also serves as the start point of the next cut, the adjusted coordinates for that point are calculated as the intersection of the two correction curves for the cuts. The new coordinates are substituted for the nominal coordinates in the pertinent part-program segment.

If circles, arcs or contours are not true because of tool wear or deflections, then the contour or circular interpolation codes in the part program could be converted into explicit linear point-to-point paths. This process would involve breaking the features into smaller linear sections and modifying the end coordinates of these linear sections accordingly. The end result would be a large output file for running real-time compensation on the finish part. This feature is not included in the current version of PIECS given in APPENDIX C since more efficient algorithms based on splines are being investigated.

4.6 Creating A Tool-Path Adjustment File For The RTEC

In addition to an adjusted NC program for the finish cut in the machining process, a file containing error profiles for different coordinate ranges is prepared. For a more precise description of the meaning of coordinate ranges see the description of the output file ERRSPEC.DAT in Section B.3. This file is used by the QIA control program, running on the Quality Monitor, to interpolate the errors along each cut, during machining, and pass the resulting correction counts to the RTEC. The RTEC converts these to compensation counts and adds these counts to the feedback position and sends the result to the machine tool controller in order to produce a current machine position adjusted to compensate feature error.

5.0 The PIECS Logical Flow

The following outline describes the flow of program control.

1. When it is called PIECS initially gets the following data items from the QIA control program:
   Gauging file sequence number
   Gauging time machine temperature state vector
   Machine home-to-part origin distance

2. PIECS then tests for improperly read gauge points, which are skipped. It then iterates through the input files each time a gauge point is skipped so that cut attributes can be reinterpreted to match only the points which did register. Only three skipped points are allowed otherwise PIECS terminates. A gauge point is selected to be skipped if the component of the probed value parallel to the part surface is larger than some preselected tolerance. This precaution is taken in order to select only those gauge points for which the probe approaches the surface in a normal direction.

3. In the process of reading input files PIECS first reads in attributes of each cut to be affected by the gauging session for the current part. The data read include nominal coordinates for the gauging points, and nominal "go-to" coordinates for the finish cut as well as the curvature of the cuts (i.e. whether the cuts are linear or arcs). Two cut attribute files are read, one for the semifinish cut and one for the finish cut.

4. Semifinish probing data are read, adjusted for G-T errors and transformed to the part coordinate frame. Since the G-T model functions are not included in the program given in APPENDIX C the function calls to the G-T model have been commented out.

5. The PIECS function next calculates deviations of actual from nominal gauging coordinates in the part coordinate frame. During the first pass through the input files, no skipped points are

assumed. However, after all skipped points, if any, have been accounted for, the function computes the scalar errors in the direction of the normal vectors at the gauge points of the semifinish cut.

6. A report file is written displaying the computed errors at the nominal gauging points for all of the cuts. The computed errors are also stored for further examination. This terminates the processing of the semifinish data. The next steps develop the appropriate corrections for the finished part.

7. The nominal cut endpoints for the finish part are read from an externally prepared file. This file is usually generated with the assistance of a CAD system.

8. Each cut of the finish part is adjusted one at a time. Since the adjustments to the finish part cuts depend on the form of the cut, the errors computed from the semifinish part are classified as errors leading to untrue curves, constant offsets or untrue planar cuts, where untrue means deviating from normal.

9. An analysis of the semifinish part errors determines whether a constant offset is sufficient to compensate the errors for either axis. If so, then PIECS inserts the appropriate values in the tool offset UPDate command in the NC program segment for the finish cut, and leaves all coordinates at their nominal values. As an alternative means of compensating the errors, PIECS also writes the axis offsets to a file which is used for real-time compensation using the RTEC.

10. Otherwise, PIECS continues to adjust the finish part cuts for untrue curves or untrue planar cuts.

11. The current version of PIECS does not correct for curved cuts as such and provides an error message for this. However, with the current version of PIECS curved cuts could be corrected by applying a large number of untrue planar adjustments as described in the next step. More efficient techniques that involve spline interpolation are being developed.

12. Untrue planar cuts are adjusted by fitting linear functions through the error vectors computed at the gauge points for each cut of the finish part. The intersection of these linear equations with the comparable linear error equations for the two neighboring cuts gives the errors at the endpoints of the cuts. These endpoint errors are used to adjust the "go-to" endpoints for the finish part. These adjusted points are then entered into the NC program for the finish cut and are written to a file that is used to provide data to the RTEC to generate real-time cut adjustments. PIECS then returns to the calling function.

6.0 Summary

This report has documented the software that is the principal component of the process-intermittent control loop of a quality-control architecture used to control manufacturing processes on a turning center. Although the specific implementation is for a turning center the principles involved are generalizeable.

The software compensates the process-related errors by modifying tool-path specifications. This adjusted data can be used by either a Real-Time Error Corrector (RTEC) to machine finish parts or an NC part program for the finish cuts when an RTEC is not available. The program first uses on-machine probing data and previously developed nominal part specification files to estimate machining errors along each feature of a part. These feature errors are then converted to compensation vectors applied to the nominal finish cut specifications. Curves are fit through the ends of these vectors and the intersection points of these curves for adjacent features are finally used to produce modified tool-path specifications for the machining of the finish part.

This software program is made available so that those wishing to implement process-intermittent error compensation have model software with which to begin their development. The authors understand that modifications will have to be made for specific implementations. They feel, however, that having working source code available for at least one implementation makes future software development for other implementations an easier task.

REFERENCES

1. Donmez, M.A. (Ed.) Progress Report of the Quality in Automation Project for FY90. NISTIR 4536, National Institute of Standards and Technology, Gaithersburg, MD, March, 1991.

2. Lovett, C.D. (Ed.) Progress Report of the Quality in Automation Project for FY88. NISTIR 89-4045, National Institute of Standards and Technology, Gaithersburg, MD, April, 1989.

3. Vorburger, T.V., Scace, B., (Ed.)Progress Report of the Quality in Automation Project for FY89. NISTIR 4322, National Institute of Standards and Technology, Gaithersburg, MD, May,1990.

4. Yee, K.W.; Gavin, R.J. Implementing Fast Part Probing and Error Compensation on Machine Tools. NISTIR 4447, National Institute of Standards and Technology, Gaithersburg, MD, October, 1990.

5. Blessing, G.V.; Slotwinski, J.A.; Eitzen, D.G.; Ryan, H.M. Ultrasonic Measurements of Surface Roughness, Appl. Op. 32(19): 3433-3437; July 1993.

APPENDIX A

Input Data Files

There are a number of files involved with PIECS both as input and output. Figure A.1 shows the sources of information for each file and the destination of the data in the PIECS scheme. This section describes in detail the data they must contain. Some file names contain numbers referring to part identification, the pertinent part-program segment, or the pertinent machining or gauging session. Except as noted, data fields in a record are delimited by one or more spaces. Each file is terminated by a record containing only a ^Z character.

A part is first designed on a CAD system (see Figure A.1). There must also be an NC machine available that can machine the part and a kinematic model of the machine. From the CAD design certain part-program files are developed, tool path files are prepared and finally the feature attributes of the part are precisely defined in files for semi-finish and finish machining. Apart from the CAD related files two other sets of files are prepared, one set specific to the NC machine and the other to the definition of the kinematic model of the NC machine. Of these files the kinematic model coordinate offset file, the initial machine state file and the cutting speed file are primarily used by the G-T model when PIECS adjusts the contact probe coordinates before converting them to part coordinates. The other files are individual machine specific and contain, for example, machine structural offsets. The part gauging file is created each time a part is probed on-line. Finally the probe parameter file gives PIECS the necessary information about the probe length and width offsets and probe stylus radius of the on-machine probe.

The current turning center uses English units. Therefore some of the units used in the input and output files of PIECS as well as within the PIECS program are constrained to English units. The conversion of the turning center controller and the PIECS program to SI units is prohibitively expensive and requires a major conversion effort. Since the object of this program documentation is to provide the reader with the essential methodology and sample code for implementing process-intermittent error compensation, the authors felt that it was acceptable to leave the units in a form consistent with the current controller. Those wishing to implement similar procedures on new turning center controllers using SI units can make the appropriate unit conversions.

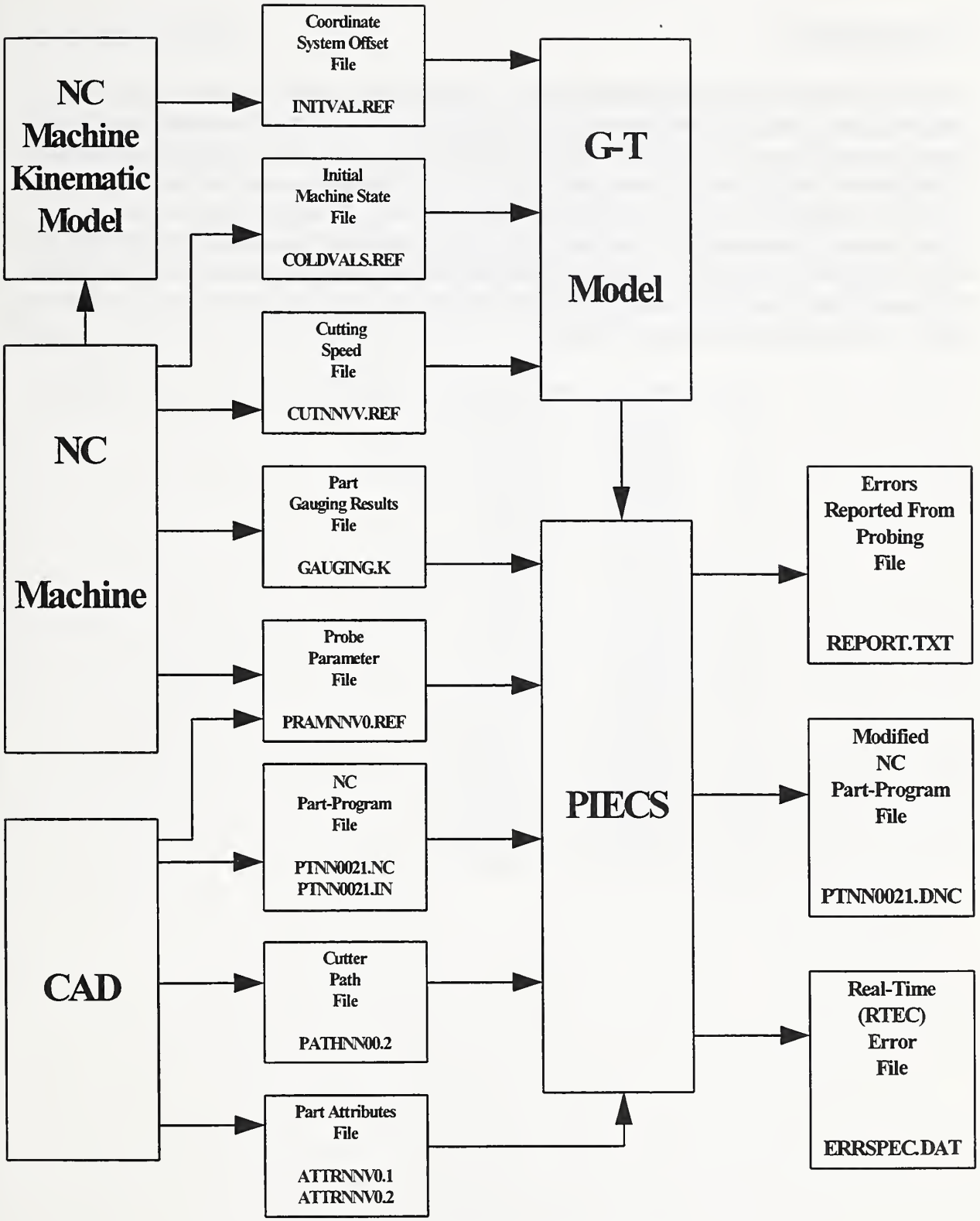

Figure A.1 - Data Flow for PIECS

A.1 GAUGING.K

This file contains dimensional gauging results. The value *k* is a sequence number for the file, and is used by a supporting database as a pointer. The sequence number is updated every time there is a gauging session and represents an index number for a gauging session. The first record in the file may be up to eighty characters in length. All fields in this record are interpreted as comments, except the second field, which contains the part ID, a decimal integer. The entire second record is interpreted as a comment. The remainder of the file contains three fields per record. The first is a decimal integer indicating the data record count (starting with 1). Next is an X coordinate, then a Z coordinate. Each coordinate is a decimal integer value (a micrometer count), exactly as reported from the RTEC.

```
Part 0611   METALIST   Wed Mar 15 10:06:24 1995
PT   X_RTEC   Z_RTEC
1  -182640   662609
2  -172479   662632
3  -162320   662661
4  -156717   657686
5  -156722   644139
6  -156727   630600
7  -156731   617049
8  -148349   611959
9  -136497   611924
10  -124642   611888
11  -112790   611851
12  -107203   606886
13  -107207   593339
14  -107213   579799
15  -107218   566249
^Z
```

A.2 PRAMNNV0.REF

This file contains dimensional probing parameters for a part ID having *nnv* as the first three digits. It contains one data value per record. The individual values are: (1) the decimal integer identifying the gauging point whose measured Z-coordinate is defined to have zero error; (2) the floating point absolute value of the X-length of the probing tool, in inches; (3) the floating point absolute value of the Z-length of the probing tool, in inches; (4) a character— either "s" or "r"— indicating whether the sequence in which the gauging points will be measured is to be the same as or the reverse of the order of points in the attributes file; (5) the floating point threshold, in inches, with respect to the part coordinate system, which the probe stylus must cross before the program reports probe trips; and (6) the floating point radius of the probe, in inches.

```
3
8.03
10.8
r
4.
.0985
^Z
```

A.3 PTNN0021.NC

The machining and probing of a part is broken into several parts. There are three sessions indexed by 0 for roughing, 1 for semifinish and 2 for finish. Each session can have up to two NC program segments, one for machining and one for probing. Roughing usually has only one segment for machining.The file described in this section is the master version of the NC machining segment for the finish cut (i.e., Session 2 Segment 1) for a part ID having *nn* as the first two digits. It is not actually an input file. However, it is the master file of which a modified copy is converted into the actual input file pt*nn*0021.in. The part program ID number in the first block of the file is set to *nn*0021 to identify it with part number *nn*.

Before it is modified for use with PIECS two adjustments are made. First of all, UPDate commands— one for X, then one for Z— for incrementing the tool offsets by zero are inserted before any machining instructions. The zeros in these commands will be replaced by values calculated during the processing of gauging data. For example, the following command specifies a zero increment for the z-offset for tool number 5, and assumes that a block number of 0160 is an appropriate insertion location in the part program.

```
N0160  (UPD, 5, TOV(Z), 0.0)
```

The second adjustment made to this segment involves blocks containing coordinates. Any machining block containing a coordinate that specifies only an X-coordinate or only a Z-coordinate is changed to specify both, even if this is redundant to the controller.

For testing purposes the NC segment below has two parameters in blocks 250 and 270, called P44 and P45 respectively. These have been included in order to allow the operator to introduce significant taper errors in the part as a means of testing PIECS's capability of correcting large errors. The results of the testing will be published elsewhere.

```
N0010  (ID,PROG, 060021, Finish Cutting of FACETST)
N0020  ! PIECS test part -- Aluminum 6061-T6
N0030  ! January 21, 1995, Neil Wilkin , mod H.B.1/26/95, mod K.Y.2/17/95
N0040  ! TOOL 1 K68, 35 DEG., LENGTH X-5.96875.,Z-1.21875
N0050  ! ...Using Center of tool, and OFFSET X+0.128, Z0
N0060  ! START MACHINE AT X0, Z10.
N0080  ! Finish Feed Rate=0.005 IN/REV
N0090  ! SEMI CUTS ARE 0.01 (0.02D) INCH OVERSIZE
N0100  ! RUFF CUTS ARE 0.02 (0.04D) INCH OVERSIZE
N0110  G70              ! Inches mode
N0120  T0100            ! Select Finishing Tool at start position of machine
N0130  T0               ! Remove tool data from controller
N0140  G94 G97 T0 M53   ! In./min., dir. rpm, no tool offsets, hi hdstk range
N0150  M03 S1600        ! Spindle on forward, set speed to 1600 rpm
N0160  G00 X0 Z(P42)    ! Move to  Zstart-semi-fin
N0170  G92 X30.4 Z2.0 S3000  ! Set Part zero and max spindle speed
N0180  T0100            ! Include Finishing tool length in registers
N0190  G00 X6.2625 Z-3.96875 T0101 M08  ! Add tool offsets, and Coolant ON
N0200  (UPD, 1, TOV(X), 0.00000)
N0205  (UPD, 1, TOV(Z), 0.00000)
N0210  G04 X5.0         ! Delay For Coolant And Spindle
N0220  G95              ! Inch per rev mode
N0230  G01 F.005 X5.9625 Z-3.96875
N0240  G01 X05.9625 Z-1.96875
N0250  G01 X02.0625 Z(-1.96875+P44) !5.9 inch "face" with taper
N0260  G01 X02.0625 Z00.03125
N0270  G01 X-0.1000 Z(00.03125+P45) !2 inch face with taper
N0280  G01 Z00.13125    ! To move away from the part safely
N0290  G00 X6.2625      !
N0300  G94              ! Inch per minute mode
N0310  G01 F100.0 X30.4 Z2.0 T00 M05 M09  ! Return to register position
N0320  G92 X0 Z(P42)    ! Reset register to Zstart
N0330  G00 X0 Z10       ! Return to machine start position
N0340   P42=(P42-0.02)  !Decrement Zstart-semi-fin for next cut
N0350  M30              ! Rewind
N9999  (END, PROG)
```

A.4 PTNN0021.IN

This is the input file for Session 2, Segment 1 (finish machining) for a part ID having *nn* as the first two digits. It is essentially identical to pt*nn*0021.nc except that symbols have replaced each value (e.g. coordinate offsets or coordinate values of a machining paths) and the second line has been changed. These symbols represent values that will be refined to reduce errors detected by the process-intermittent gauging. Each symbol consists of a dollar sign immediately followed by an integer. The symbols $1 and $2 replace the zero values in the X and Z tool offset UPDate commands, respectively. The remaining symbols are numbered in order as the file proceeds from the beginning to the end, starting with $3. Currently this file is prepared by hand after deciding on the key coordinates that will be refined for the finish cut.

```
N0010  (ID,PROG, 060021, Finish Cutting of FACETST)
N0020  ! PIECS test part -- Aluminum 6061-T6
N0030  ! February 19, 1995, Herbert T. Bandy
N0040  ! TOOL 1 K68, 35 DEG., LENGTH X-5.96875.,Z-1.21875
N0050  ! ...Using Center of tool, and OFFSET X+0.128, Z0
N0060  ! START MACHINE AT X0, Z10.
N0080  ! Finish Feed Rate=0.005 IN/REV
N0090  ! SEMI CUTS ARE 0.01 (0.02D) INCH OVERSIZE
N0100  ! RUFF CUTS ARE 0.02 (0.04D) INCH OVERSIZE
N0110  G70              ! Inches mode
N0120  T0100            ! Select Finishing Tool at start position of machine
N0130  T0               ! Remove tool data from controller
N0140  G94 G97 T0 M53   ! In./min., dir. rpm, no tool offsets, hi hdstk range
N0150  M03 S1600        ! Spindle on forward, set speed to 1600 rpm
N0160  G00 X0 Z(P42)    ! Move to  Zstart-semi-fin
N0170  G92 X30.4 Z2.0 S3000  ! Set Part zero and max spindle speed
N0180  T0100            ! Include Finishing tool length in registers
N0190  G00 X6.2625 Z-3.96875 T0101 M08  ! Add tool offsets, and Coolant ON
N0200  (UPD, 1, TOV(X), $1)
N0205  (UPD, 1, TOV(Z), $2)
N0210  G04 X5.0         ! Delay For Coolant And Spindle
N0220  G95              ! Inch per rev mode
N0230  G01 F.005 X$3 Z$4
N0240  G01 X$5 Z$6
N0250  G01 X$7 Z$8  !5.9 inch "face" with taper
N0260  G01 X$9 Z$10
N0270  G01 X$11 Z$12  !2 inch face with taper
N0280  G01 Z00.13125    ! To move away from the part safely
N0290  G00 X6.2625      !
N0300  G94              ! Inch per minute mode
N0310  G01 F100.0 X30.4 Z2.0 T00 M05 M09  ! Return to register position
N0320  G92 X0 Z(P42)    ! Reset register to Zstart
N0330  G00 X0 Z10       ! Return to machine start position
N0340   P42=(P42-0.02)  !Decrement Zstart-semi-fin for next cut
N0350  M30              ! Rewind
N9999  (END, PROG)
^Z
```

A.5 PATHNN00.2

This file contains the nominal tool path specifications (or coordinates) which have been replaced by symbols in pt*nn*0021.in and the corresponding cut numbers. A "cut" has been defined previously as the portion of part geometry which is produced by a machining pass of the tool from a position specified in the next block of NC code to the next position, specified in another block. Those blocks are called the "initial block" and "final block", respectively, for that cut. This file is required for only finish machining (Session 2 Segment 1) for a part ID having *nn* as the first two digits. It must be created with a text editor after pt*nn*0021.in is developed.

Each record in path*nn*00.2 contains four fields. The first two columns contain the nominal X and Z-coordinates (or other tool path specifications, if applicable) for a block. Each pair of decimal floating point coordinates is taken from a block of NC code, which may, for example, be the initial block for a cut, say Cut P; and the final block for another, Cut Q. The third and fourth fields in each record contain the integer cut numbers, P and Q, respectively, which correspond to the coordinate pair appearing in the same record. If a cut number is not applicable, zero is entered instead. The x-coordinates are in diameter mode.

The first record is special. It represents both blocks for the tool offset increments. All four values in the first record are zeros, since they refer to the dummy increments of tool offsets, which are not associated with cuts.

In addition to being used by PIECS, this file is used by the CAD Part Definition Module to determine the initial and final blocks for each cut, and enter them in the PARTFEAT table of the database (see APPENDIX D). The blocks are identified by the numbers of the symbols they contain, which are implied by the sequence in path*nn*00.2.

```
0.00000     0.00000     0     0
5.96250    -3.96875     3     2
5.96250    -1.96875     4     3
2.06250    -1.96875     5     4
2.06250     0.03125     6     5
-0.10000    0.03125     7     6
^Z
```

A.6 ATTRNNV0.I

These files contain attributes of each cut concerned with the I-th session for a part ID having *nnv* as the first three digits, where I = 1 for semifinish and I = 2 for finish. Each of the files contains a series of records, associated with each cut, in the order of cut number. The format of the series of records for the L-th cut is as follows.

The first record in the series for the L-th cut consists of fourteen fields, delimited by spaces. The first field contains a single decimal integer specifying *how many points* that are to be gauged along the surface produced by the L-th cut. If this value is zero then the series of records associated with the L-th cut contains only one record.

The second and third fields each contain a single character to indicate the *direction of tool motion* for the X and Z-axes, respectively, during the L-th cut. The valid characters are "+", "0", and "-". For example, if the first field indicated that there were no gauging points for the L-th cut, then it is assumed that this may be a hypothetical cut. Hypothetical cuts are parallel to axes. The purpose of this imaginary cut is explained below. If this is an actual cut instead, but with no gauging points, it is treated the same as a hypothetical cut.If the imaginary cut is parallel to the X axis, then the second and third fields contain "+" (or "-") and "0" respectively to indicate vertical motion, or for the Z axis, "0" and "+" (or "-") respectively to indicate horizontal motion. In these cases the fourteenth field in the record then contains a coordinate — a Z-coordinate in the case of an imaginary cut parallel to the X axis, or an X-coordinate (diameter mode) in the case of an imaginary axis parallel to the Z axis. See the first two records of the example files below.

The fourth field contains an integer code representing the *intersecting cut,* the cut which is starting just as the L-th cut concludes (a clockwise direction is assumed from the machine Z axis). For the case in which no other actual cut intersects the endpoint of the L-th cut, a hypothetical cut is defined to serve the purpose, except for the final cut (necessarily hypothetical), for which a code of zero is used because there actually is no cut intersecting its endpoint.

The fifth and sixth fields indicate which *tool path coordinate* in the next machining segment is to be affected by adjusting the coordinates of the endpoint (not to be confused with the start-point) of the L-th cut. The fifth field contains a single character to denote the type of specification. Valid characters are "X" (for X-coordinate), "Z" (for Z-coordinate), "O" (i.e., the letter, not zero, for a type of specification "other" than an X- or Z-specification), and "N" (for no specification). The sixth field contains a decimal integer corresponding to the pertinent coordinate.

The seventh and eighth fields are similar to the fifth and sixth, allowing for an additional tool path specification (such as a "go-to" coordinate). In most cases, these four fields will specify X and Z coordinates.

The ninth field contains a floating point decimal value from which the *type of the curve* produced

by the L-th cut is known. Any (arbitrary) negative value indicates a straight line. Otherwise, the curve is a circular arc, and the value is the nominal X-coordinate of its center point with respect to the coordinate system of the part. It is possible for an arc to have a negative X coordinate for a center, but a positive X coordinate for the center covers most arcs of significance.

If the curve is an arc, the tenth field is similar to the ninth, except that it contains the Z-coordinate of the arc center. In the case of a straight line, the tenth field contains an arbitrary number.

The eleventh field contains an integer that identifies a cut (the same as the cut number for that surface) which lies at some angle (often two hundred seventy degrees) with respect to the surface produced by the L-th cut. The angle will be specified in the next field. Note that the plane referred to can be a theoretical tangent at the endpoint of an actual surface.

The twelfth field contains the decimal floating point angle, in degrees, which exists between the plane of the L-th cut and the plane indicated in the eleventh field. By convention the angle between normal vectors is assumed to lie between the exposed sides of the surfaces, measured in a counterclockwise direction from the positive Z axis to the positive X axis in the part coordinate system. The vertex is located between the feature boundaries. For example, in the third record below there is an angle between the third cut and the second cut of 90 degrees, which means the second cut is vertically down and the third cut is horizontal.

The thirteenth and fourteenth fields are for an additional planar relationship, such as a previous cut specification, just as for the eleventh and twelfth fields. If the first field in this record contains a zero, then the thirteenth field contains a dummy value, and the fourteenth field contains the plane coordinate mentioned in the explanation of the second and third fields. In the case of the third record below there is an angle of 270 degrees with the fourth cut. This implies that cut three, which is horizontal, then goes into cut four, which is vertically down. Note that the exposed angle between cut four and cut three is 270 degrees counterclockwise.

If the first field contains a positive integer, say N, then there are N more records in the series, each associated with a gauging point. These records consist of four fields, each containing a number. The first two numbers are the *nominal X and Z-coordinates* of a gauging point along the surface produced by the L-th cut. The X-coordinate is a diameter mode value. The next two numbers are the unit vector i and k components of a *surface normal vector* originating at that gauging point. The X (in diameter mode) and Z specifications are floating point decimal values representing inches, and are relative to the coordinate system of the part.

The remaining records in the series for the L-th cut are similar to the second record in the series, each associated sequentially with a gauging point along the surface produced by the cut.

```
0 0 + 2 X 0 Z 0 -1 0 0 0.000000 0 6.000000

0 - 0 3 X 3 Z 4 -1 0 0 0.000000 0 -3.990000

4 0 + 4 X 5 Z 6 -1 0 2 90.000000 4 270.000000
5.920000 -3.700000 1.000000 0.000000
5.920000 -3.166667 1.000000 0.000000
5.920000 -2.633333 1.000000 0.000000
5.920000 -2.100000 1.000000 0.000000

4 - 0 5 X 7 Z 8 -1 0 3 270.000000 5 90.000000
5.700000 -1.990000 0.000000 1.000000
4.766666 -1.990000 0.000000 1.000000
3.833334 -1.990000 0.000000 1.000000
2.900000 -1.990000 0.000000 1.000000

4 0 + 6 X 9 Z 10 -1 0 4 90.000000 6 270.000000
2.020000 -1.700000 1.000000 0.000000
2.020000 -1.166667 1.000000 0.000000
2.020000 -0.633333 1.000000 0.000000
2.020000 -0.100000 1.000000 0.000000

3 - 0 7 X 11 Z 12 -1 0 5 270.000000 0 0.000000
1.800000 0.010000 0.000000 1.000000
1.000000 0.010000 0.000000 1.000000
0.200000 0.010000 0.000000 1.000000

0 0 + 0 N 0 N 0 -1 0 0 0.0 0 -0.100000
^Z
```

The next sample file is for the finish cut and would have an extension of I = 2 in the file name.

```
0 0 + 2 X 0 Z 0 -1 0 0 0.000000 0 6.000000

0 - 0 3 X 3 Z 4 -1 0 0 0.000000 0 -4.000000

4 0 + 4 X 5 Z 6 -1 0 2 90.000000 4 270.000000
5.900000 -3.700000 1.000000 0.000000
5.900000 -3.166667 1.000000 0.000000
5.900000 -2.633333 1.000000 0.000000
5.900000 -2.100000 1.000000 0.000000

4 - 0 5 X 7 Z 8 -1 0 3 270.000000 5 90.000000
5.700000 -2.000000 0.000000 1.000000
4.766666 -2.000000 0.000000 1.000000
3.833334 -2.000000 0.000000 1.000000
2.900000 -2.000000 0.000000 1.000000

4 0 + 6 X 9 Z 10 -1 0 4 90.000000 6 270.000000
2.000000 -1.700000 1.000000 0.000000
2.000000 -1.166667 1.000000 0.000000
2.000000 -0.633333 1.000000 0.000000
2.000000 -0.100000 1.000000 0.000000

3 - 0 7 X 11 Z 12 -1 0 5 270.000000 0 0.000000
1.800000 0.000000 0.000000 1.000000
1.000000 0.000000 0.000000 1.000000
0.200000 0.000000 0.000000 1.000000

0 0 + 0 N 0 N 0 -1 0 0 0.0 0 -0.100000
^Z
```

A.7 CUTNNVV.REF

This file contains machining data required for tool setting, process-intermittent compensation, and real-time compensation, for part ID *nnvv*. The first record contains a decimal floating point value, in inches, which is the signed Z-distance from the part origin to the machine origin. This is determined using a tool locating point. That value is equal to Z_{part} minus $Z_{machine}$. Following that value in the first record is the decimal integer tool specification (no tool changes are allowed in the current version of PIECS). The second record contains two decimal floating point values that represent the X and Z-coordinates of the tool nose center at the dwell point specified in the finish cut NC program, before the first machining pass. The coordinates for the semifinish NC program are close enough not to need to be specified separately. The coordinates are in the part coordinate system and in diameter mode. Next, there is a record for each spindle speed specified in the NC program. Each record contains a decimal integer spindle speed, followed by the minimum Z-coordinate (a decimal floating point value in inches, with respect to the part coordinate system) of the range within which that speed is in effect.

```
-2.76364  4
6.2625 -3.96875
1000 -1.8
2000 -4.55
3000 -5.5
4000 -6.5
^Z
```

A.8 COLDVALS.REF

This file contains reference values used in a tool setting operation and in the G-T model. The values obtained using the tool setting station are taken on a cold (room temperature) machine. The first record contains a comment which may be up to eighty characters long. The second and third records each contain a single decimal integer value, in micrometers, for the X and Z-coordinates of the turning center carriage relative to the machine home. These coordinates are obtained when the tool setting station is touched with the "golden tool". This tool is special in that its length and width are measured on a CMM. The fourth and fifth records each contain a single decimal floating point value, in millimeters, for the X and Z-displacement error at the time of contact with the tool setting station. The sixth record contains forty decimal floating point values for the machine temperatures, in degrees celsius, in the same order in which the thermocouples are numbered, at the time of contact with the tool setting station.

```
METALIST   Thu Oct 14 15:31:19 1993
-28234
39798
-0.002307
-0.001130
23.84  24.19  22.38  24.52  27.39  27.96  23.54  ...
^Z
```

A.9 INITVALS.REF

This file contains nominal locations of the kinematic components of the turning center. The location of a component is expressed as an offset of its coordinate system from that of another component. The first record contains a comment which may be up to eighty characters long. The second and third records each contain a single decimal floating point value, in millimeters, for the X and Z-offsets of the tool setting station from the absolute reference system of the machine tool. Each remaining data record contains a single decimal integer value. Each pair of these records refer to an X and Z-offset, respectively, in micrometers. The fourth and fifth records are the offsets of the spindle from the absolute reference; the sixth and seventh the turret from the cross slide; the eighth and ninth the cross slide from the carriage; and the tenth and eleventh the carriage from the absolute reference. The offsets are used in the G-T model.

```
METALIST   Fri Oct 29 13:42:30 1993
124.081243
71.479497
-384407
-150112
0
0
88900
-36510
-198450
-31750
^Z
```

APPENDIX B

Output Files

B.1 REPORT.TXT

This file is a summary of error history for a completed part. The units are in inches. Below is a sample output.

```
                Record of Measured Errors
              From Manufacture of Part 0611 on
                  Wed Mar 15 10:07:02 1995

*****************************************

                      Gauging
              Session ->          1

            Point 1        -0.002047

            Point 2        -0.001142

            Point 3         0.000000

            Point 4        -0.009961

            Point 5        -0.010157

            Point 6        -0.010354

            Point 7        -0.010512

            Point 8         0.003858

            Point 9         0.002480

            Point 10        0.001063

            Point 11       -0.000394

            Point 12       -0.010591

            Point 13       -0.010748

            Point 14       -0.010984

            Point 15       -0.011181
```

B.2 PTNN0021.DNC

This is the output NC program file for Session 2, Segment 1 (finish machining) for a part ID having NN as the first two digits. This file is identical to PTNN0021.IN except now the symbols have been replaced by refined values in order to reduce error detected by the process-intermittent gauging.

```
N0010  (ID,PROG, 060021, Finish Cutting of FACETST)
N0020  ! PIECS test part -- Aluminum 6061-T6
N0030  ! February 19, 1995, Herbert T. Bandy
N0040  ! TOOL 1 K68, 35 DEG., LENGTH X-5.96875.,Z-1.21875
N0050  ! ...Using Center of tool, and OFFSET X+0.128, Z0
N0060  ! START MACHINE AT X0, Z10.
N0080  ! Finish Feed Rate=0.005 IN/REV
N0090  ! SEMI CUTS ARE 0.01 (0.02D) INCH OVERSIZE
N0100  ! RUFF CUTS ARE 0.02 (0.04D) INCH OVERSIZE
N0110  G70              ! Inches mode
N0120  T0100            ! Select Finishing Tool at start position of machine
N0130  T0               ! Remove tool data from controller
N0140  G94 G97 T0 M53   ! In./min., dir. rpm, no tool offsets, hi hdstk range
N0150  M03 S1600        ! Spindle on forward, set speed to 1600 rpm
N0160  G00 X0 Z(P42)    ! Move to  Zstart-semi-fin
N0170  G92 X30.4 Z2.0 S3000  ! Set Part zero and max spindle speed
N0180  T0100            ! Include Finishing tool length in registers
N0190  G00 X6.2625 Z-3.96875 T0101 M08  ! Add tool offsets, and Coolant ON
N0200  (UPD, 1, TOV(X), 0.00000)
N0205  (UPD, 1, TOV(Z), 0.00000)
N0210  G04 X5.0         ! Delay For Coolant And Spindle
N0220  G95              ! Inch per rev mode
N0230  G01 F.005 5.93063 -4.00000
N0240  G01 5.93092 -2.00002
N0250  G01 2.03120 -2.00515  !5.9 inch "face" with taper
N0260  G01 2.03141 -0.00030
N0270  G01 -0.05000 0.00274  !2 inch face with taper
N0280  G01 Z00.13125    ! To move away from the part safely
N0290  G00 X6.2625      !
N0300  G94              ! Inch per minute mode
N0310  G01 F100.0 X30.4 Z2.0 T00 M05 M09  ! Return to register position
N0320  G92 X0 Z(P42)    ! Reset register to Zstart
N0330  G00 X0 Z10       ! Return to machine start position
/N0340   P42=(P42-0.02)  !Decrement Zstart-semi-fin for next cut
N0350  M30              ! Rewind
N9999  (END, PROG)
```

B.3 ERRSPEC.DAT

For the process-intermittent compensation this file is an error profile for a part. The first field in the file contains the floating point version number of the PIECS software which was used to produce the file. The remainder of that same record is interpreted as a comment field of up to eighty characters in length. The second record consists of two fields containing the decimal floating point constant errors for the X and Z-axes, respectively. If the constant error for an axis is nonzero, it supersedes any subsequent errors the file may specify for that axis. If nonzero constant errors are specified for both axes, there will be no more data records in the file. Otherwise, for the rest of the file, each record represents a coordinate range specified in the part coordinate system for a given axis. This range is used by the RTEC controller program that reads the errspec.dat file. The first field in the record contains a character (either "X" or "Z") indicating whether the range is in terms of X-coordinates or Z-coordinates. The controller program monitors this axis from the beginning of the range to the end of the range. Next are six fields, each containing a decimal floating point value: the beginning (in terms of cut direction) coordinate in the range, followed by the corresponding errors, first X, then Z; then the ending coordinate and its X and Z errors. The X and Z errors at the extremities of the range are used by the controller program to linearly interpolate the errors in the range. These interpolated values are passed to the RTEC which automatically converts the interpolated error value to a compensation value by attaching a negative sign. All units are inches

```
1.00 (version); axis, start_coord, x_err, z_err, end_coord, x_err, z_err
0.00000  0.00000
X   15.20000    0.00000    0.00000    2.98125    0.00000    0.00000
Z   -3.96875   -0.01531    0.00000   -1.96875   -0.01546   -0.00007
X    2.98125   -0.01546   -0.00007    1.03125   -0.01560    0.00507
Z   -1.96875   -0.01560    0.00507    0.03125   -0.01571    0.00039
X    1.03125   -0.01571    0.00039   -0.05000    0.00000   -0.00281
^Z
```

APPENDIX C

Program Function Description and Listings

C.1 Program Function Calling Sequence

The hierarchy of function calls within the program is outlined below.

```
piecs
    get_part_no
    read_cut_attributes
        normalize_vector
    read_rtec
        thermal_adjust
        rtec_to_part
    calculate_errors
        skipped_points_test
    write_report
    read_toolpath_data
    classify_errors
    untrue_curve
    constant_error
        offset_change
            max_value
            min_value
            smallest_magnitude
    untrue_planar
        calculate_coefficients
            straight_line
        solve_equations
    write_segment
    errspec
```

C.2 Program Initializations

The program initialization portion declares the standard C program headers used along with defined constants, declared structures and global variables. These will be described in the next sections.

C.2.1 Defined Constants

Several constants are defined that are used throughout the program. These can be changed and the program recompiled. The units for some of the constants are inches since the NC programs for the turning center were written in the English units.

A program VERSION number is first given. This is to be changed when updates are made. A section is provided at the beginning of the program to record the date and by whom changes are

made. Notes can also be supplied. The machine tool servo resolution (*servo_res*) is 0.000039 in (1μm). The next constants are percentage tolerances used for program control. The first, called *tolerance1*, is the greatest value assumed to be approximately equal to zero in the calculations. The next, *tolerance2*, set to 0.2, is the fractional error variation acceptable for offset compensation. The third, *correction*, set to 1.0, is the percent correction to apply to the total error. This constant is provided so that a user can choose to apply only a selected percentage of the corrections. The final set of constants are limit values used in defining arrays. The overall maximum number of gauging points (*max_points*) is set to 25. The maximum number of cuts per part (*max_cuts*) is set to 10. The maximum number of gauge points per cut (*pts_per_cut*) is set to 10 but could be set higher if desired. Finally the maximum number of machining + gauging sessions (*max_sessions*) is set to 2. These settings of course could be changed, but for the current program these have been found adequate.

C.2.2 Defined Data Structures

Two principal data structures are defined. The first is a structure that identifies attributes of cuts. The following is the list of attributes that are stored in the cut attribute data structure.

how_many_points: The actual number of points to be gauged along a surface produced by the cut of interest.

cut_direction: This indicates the direction of tool motion during a cut. The symbols are "+", "0" or "-" for each axis (x and z).

intersect_cut: This is a code indicating what other cut is starting just as the cut of current interest concludes.

endpoint_spec: A code indicating whether a variable used in the next NC block is an x or z "go-to" coordinate or another kind of specification; and a code indicating which particular variable. This variable will be refined for in the program for better machining accuracy on the basis of the analysis of current gauging data.

type_of_curve: Codes indicating whether the pertinent feature is an arc or a line. If the first value is negative, then the feature is a line. Otherwise, the two values are the x and z-coordinates, respectively, in terms of the part coordinate system, of the center of an arc. This data comes from the attribute files.

other_planes: The code for a plane (identified by the number of the cut having produced the surface), and the angle the plane makes with the surface produced by the cut of current interest. A plane can be a hypothetical tangent plane.

nominal_coord: The nominal x and z-coordinates of a gauging point, in terms of the part coordinate system, in the diameter mode used by the controller. In any particular attributes file or data structure, nominal_coord pertains to coordinates specified in the part coordinate frame of

the current session; i.e. the machining segment before the current gauging segment.

normal_vector: The i and k vector components of a surface normal vector originating at a gauging point.

gauged_coord: The measured x and z coordinates of a gauging point, in terms of the coordinate system of the part, in diameter mode for the NC controller.

error_scalar: The magnitude (positive or negative) of the component of the error vector which lies in the direction of the surface normal of the point of interest. The error vector is equal to the gauged_coord minus the nominal_coord (of the last machining segment; i.e. of the current session). This is a computed value.

adjusted_coord: This value is computed as the nominal_coord of a gauge point for the finish machining segment plus the product of the unit normal vector, the negative of the error scalar, and the percent correction. This is a computed value and generates the compensation vector at a gauging point for the finish part..

poly_coeff: The coefficients in the correction curve polynomial. The correction curve fits the adjusted_coord's for the finish machining segment and thus creates an adjusted tool path for the finish cut. The equation form is given in Section 5.4 of this report. These are computed values.

The second principal data structure is called toolpath_data and is used to point to cut coordinates.

C.2.3 Global Variables

This section defines those program variables that are accessible to all function modules of the program.

segment_no: The next segment of the NC program that will be executed. There is a machining segment (1), probing segment (2) and scanning segment (3) for each session.

part_no: The part number of the current part.

total_cuts[]: The number of cuts for each session, including hypothetical cuts defined to locate intersections with actual cuts.

sessions: The total number of machining plus gauging sessions for the current part. Session 0 is for rough cutting with no analysis, session 1 is for semifinish cutting and includes machining plus gauging, session 2 is for finished cutting and includes machining plus gauging. However, sessions is always equal to 1 or 2 in the current program version since it applies only to the semifinish and finish sessions.

z_difference: This is equal to the machine z-coordinate of a point, minus the z-coordinate in terms of the part coordinate system.

x_error: The x-direction component of the error at a point.

z_error: The z-direction component of the error at a point.

toolpath_spec[i]: Are specifications for the part program, where i = 0 and 1 for the z and x tool offset increments. Subscripts greater than 1 refer to specifications such as z and x "go-to" coordinates that are explicitly numbered accordingly (except that each number is one higher than the corresponding number in toolpath_spec) in the file attrnnv0.i.

probe_rad: Is the radius of the gauging probe.

nn, vv: Is temporary storage for part number components. These represent portions of the PARTID. The nn represents the two digit part number and the vv represents the two digit version number of the part.

mach_offset_x, mach_offset_y: These are offsets of the machine axis system caused by thermal distortions. In the current program version these are considered small and set to 0.

spindle_sp: Spindle speed used during cuts. This is used in the G-T model and is read from the cutnnvv.ref file.

direction: This is the direction of approach by the probe for each gauging point. It is used by the G-T model to select the appropriate adjustment for a gauged point. Adjustments can be directionally dependent due, for example, to backlash.

x_coord_offset, z_coord_offset: These are offsets between the various coordinate sytems used in the kinematic model that computes the G-T error correction equations. Since these offsets appear in the equations they must be supplied to the G-T model by PIECS.

C.2.4 Program Initialization Listing

```
/*************************************************************************
 *      PIECS: The QIA Process-Intermittent Error Compensation Software     *
 *         --A program to produce two means of compensating errors          *
 *               measured by on-machine dimensional gauging:                *
 *    a modified part program and an RTEC-oriented toolpath error profile   *
 *                                                                           *
 *    Designer:  Herbert T. Bandy          Programmer:  Herbert T. Bandy     *
 *                                                                           *
 *************************************************************************

       The official version number of this software is recorded here as
       a global parameter whose value is returned by the function piecs     */

#define                          VERSION    1.00           /*   18 Jun 95   */

/*************************************************************************
 *                                                                         *
 *                           MODIFICATION HISTORY                          *
 *                                                                         *
 *    DATE      PROGRAMMER                        COMMENTS                 *
 *                                                                         *
 *                                                                         *
 *************************************************************************

  INPUT FILE NAMES:
        \qia\probing\gauging.k
        \qia\probing\attrNNV0.N
        \qia\probing\pramNNV0.ref
        \qia\cutting\pathNN00.2
        \qia\cutting\ptNN0021.in
        \qia\toolset\coldvals.ref
        \qia\toolset\initvals.ref
        \qia\cutting\cutNNNN.ref

  OUTPUT FILE NAMES:
        \qia\piecs\report.txt
        \qia\cutting\ptNN0021.dnc
        \qia\cutting\errspec.dat

  LINKED FILES:
        \qia\thermal\gtmodel.cpp

  PREPROCESSOR DATA                                                      */

#include   <dos.h>
#include   <math.h>
#include   <time.h>
#include   <stdio.h>
#include   <conio.h>
```

```
#include   <stdlib.h>
#include   <string.h>


#define   servo_res    .000039 /*  Machine tool servo resolution             */
#define   tolerance1 1.0E-10  /*  Max value considered approximately zero.  */
#define   tolerance2  .2      /*  % error variation correctable by offset   */
#define   correction 1.0      /*  % correction to apply to total error      */
#define   max_points  25      /*  Overall maximum number of gauging points  */
#define   max_cuts    10      /*  Max no. of "cuts" (see below) per part    */
#define   pts_per_cut 10      /*  Maximum number of points per cut          */
#define   max_sessions 2      /*  Two machining/gauging sessions:
                                                    semifinish and finish */


    /*  TERMINOLOGY AND GLOBAL VARIABLES

        A "cut" is defined as the portion of part geometry which is produced
        by a machining pass of the tool from a position specified in a block
        of NC code to the next position, specified in another block

        "part" is composed of one or more "cuts"

        The attributes of each cut and the resulting surface are defined as
        follows:

        how_many_points:  The number of points to be gauged along the
            surface produced by the cut of interest. This number can be 0.
            Input from attributes file.

        cut_direction:  "+", "0", or "-" for each axis (X and Z),
            indicating the direction of tool motion during a cut.  Input
            from attributes file.

        intersect_cut:  Numeric code indicating what other cut is starting
            just as the cut of current interest concludes. Input from
            attributes file.

        endpoint_spec:  A code indicating whether a variable used in the
            next part program segment is an X or Z "go-to" coordinate or
            another kind of specification; and a code indicating which
            particular variable.  Such a variable is to be refined for
            better machining accuracy on the basis of the analysis of
            current gauging data. Input from attributes file.

        type_of_curve:  Codes indicating whether the pertinent feature is
            an arc or a line.  If the first value is negative, then the
            feature is a line.  Otherwise, the two values are the X and Z
            coordinates, respectively, in terms of the part coordinate
            system, of the center of an arc.  Input from attributes file.

        other_planes:  The code for a plane (identified by the number of
            the cut having produced the surface), and the angle the plane
```

makes with the surface produced by the cut of current interest. Then, a second code and angle, referring to a second plane. A plane can be a hypothetical tangent plane. Input from attributes file.

nominal_coord: The nominal X and Z coordinates of a gauging point, in terms of the part coordinate system, but not in diameter mode. In any particular attributes file or data structure, nominal_coord pertains to specifications attemped in the machining segment of the current session; i.e., the machining segment before the current gauging segment. Input from attributes file.

normal_vector: The X and Z (actually, i and k) components of a surface normal vector originating at a gauging point. Input from attributes file.

gauged_coord: The measured X and Z coordinates of a gauging point, in terms of the coordinate system of the part, and in diameter mode. Input from RTEC data file.

error_scalar: The magnitude (positive or negative) of the component of the error vector which is in the direction of the surface normal of the point of interest. The error vector is equal to the gauged_coord minus the nominal_coord (of the last machining segment; i.e., of the current session). Computed value.

adjusted_coord: The nominal_coord of the machining segment of the NEXT session, plus the product of the unit normal vector, the negative of the error scalar, and the percent correction. Computed value.

poly_coeff: The coefficients in the correction curve polynomial. The correction curve fits the adjusted_coord's, and is the tool path for the corresponding cut in the next machining segment. The equation of the curve is

```
C1z**3 + C2z**2 + C3zx + C4x**2 + C5z + C6x = C0.
```

The following coefficients are given for the curves indicated:

	C1	C2	C3	C4	C6
Line	0	0	0	0	1
Circle	0	1	0	1	
Explicit Quadratic	0		0	0	1
General Quadratic	0				
Explicit Cubic			0	0	1

```
/*  FUNCTION PROTOTYPES                                          */
```

```
void  prnwin(char *, int, int);


    /*  GLOBAL VARIABLES

        **** Define the cut attributes data structure *****

        First, define simple dependent data structures and dummy data-
        storage assignments.  Then build to form the more complex
        attributes structure.                                          */

    struct  sign
      {
        char  x;
        char  z;
      };

    struct  part_prog_spec
      {
        char  coord;
        int  spec;
      };
    struct part_prog_spec  dummy_endpoint_spec[max_sessions][max_cuts][2];

    struct  plane_angle
      {
        int  plane;
        double  angle;
      };
    struct  plane_angle  dummy_other_planes[max_sessions][max_cuts][2];

    struct  coord_values
      {
        double  x;
        double  z;
      };
    struct  coord_values
        dummy_nominal_coord[max_sessions][max_cuts][pts_per_cut],
        dummy_normal_vector[max_sessions][max_cuts][pts_per_cut],
        dummy_gauged_coord[max_sessions][max_cuts][pts_per_cut],
        dummy_adjusted_coord[max_sessions][max_cuts][pts_per_cut];

    struct  cut_attributes
      {
        int  how_many_points;
        struct sign  cut_direction;
        int intersect_cut;
        struct part_prog_spec  *endpoint_spec;
        double  *type_of_curve;
        struct plane_angle  *other_planes;
        struct coord_values  *nominal_coord;
        struct coord_values  *normal_vector;
        struct coord_values  *gauged_coord;
```

```
    double  *error_scalar;
    struct coord_values  *adjusted_coord;
    double  *poly_coeff;
  };
struct cut_attributes  attrib[max_sessions][max_cuts], *cut_no;

                /*  Establish a structure to contain cut coordinates.   */

struct  toolpath_data
  {
    double  *startpoint;
    double  *endpoint;
  };

/* Define some dummy arrays                                              */

double  dummy_error_scalar[max_sessions][max_cuts][pts_per_cut],
    dummy_type_of_curve[max_sessions][max_cuts][2],
    dummy_poly_coeff[max_sessions][max_cuts][7];


/*  OTHER GLOBAL VARIABLES

    segment_no:  refers to the next segment of the NC program that is
        to be executed.  Set in main.

    part_no:  is the number of the current part. Read in from the
        command line.

    total_cuts:  is the number of "cut"s for each session, including
        hypothetical cuts defined to locate intersections with actual
        cuts.  Computed in read_cut_attributes.

    total_points:  is the total number of gauge points for all cuts.
        Computed in read_cut_attributes.

    sessions:  is the total number of machining/gauging sessions for
        the current part.  Computed in read_cut_attributes.

    z_difference:  is equal to the RTEC decimal value corresponding to
        the  Z-coordinate of a point, minus the Z-coordinate in terms
        of the part coordinate system.  Computed in read_rtec.

    x_error, z_error:  are X- and Z-direction components of the error
        at a point.  Computed in constant_error.

    toolpath_spec[i]:  are specifications for the part program, where
        i = 0 and 1 for the Z and X tool offset increments,
        respectively; and higher subscripts refer to specifications
        such as Z and X "go-to" coordinates which are explicitly
        numbered accordingly (except that each number is one higher
        than  the corresponding number in "toolpath_spec") in the file
        pathNN00.2, the version of the part program used by this
```

```
            program.                                                    */

    int  spindle_sp, direction, nn[2], vv[2], segment_no,
        total_cuts[max_sessions], sessions, total_points;
    char  gauge_file[13], part_no[5];
    long  x_coord_offset[6], z_coord_offset[6];
    double  z_difference, x_error[max_points], z_error[max_points],
        toolpath_spec[max_cuts*2], probe_rad, mach_offset_x, mach_offset_z;

//  extern unsigned _stklen = 65519;    // Stack size increase


/*****************************************************************************/
```

C.3 PIECS Support Functions and Listings

This section includes a brief description of the functions and calling sequences for the various functions that comprise the PIECS code. Not all of the functions have been implemented in this version but calling sequences have been included to mark the places for future enhancements. The interested reader is urged to follow the code listings below while reading the general descriptions.

C.3.2 get_part_no

This function extracts the part number from the gauging file and is the first function in the program listing. All the other functions below now follow in sequence in the listing.

Function Prototype Declaration:

int get_part_no (char seq[])

Inputs:

seq - character array representing the extension sequence number for the gauging data file.

Outputs:

The program determines the current part number from the gauging file.

Function Procedures:

The program opens the gauging file with the extension given by the seq array and extracts the current part number from the first record. The part number is the second field of that record.

```
/*****************************************************************************
  Function to extract the part number from the gauging file
*****************************************************************************/

int  get_part_no (char seq[])

  {
    char  gauge_file[35];
    FILE  *in_file;
    static char  gauge[]="\\qia\\probing\\gauging.";

    strcpy(gauge_file, gauge);
    strncat(gauge_file, seq, 7);
    in_file = fopen(gauge_file, "r");
    fscanf(in_file, "%*s %s", part_no);
    fclose(in_file);

    return 0;
  }
    /* end of get_part_no */
```

C.3.3 normalize_vector

This function normalizes surface normal vectors and produces a unit normal_vector at the current gauge point for a specified cut.

Function Prototype Declaration:

void normalize_vector (int point)

Inputs:

point - The index of the current gauge point to be normalized.

Outputs:

The normalized normal vector indexed by point.

Function Procedures:

The function computes the magnitude of the normal vector as the square root of the sum of the squares of the normal vector components. It then divides the components by this magnitude.

```
/*****************************************************************************
  Function to normalize surface normal vectors to unit value. This function
    produces a unit normal_vector at the current gauge point for a
    specified cut_no.
*****************************************************************************/

void  normalize_vector (int point)

/*****************************************************************************
*
* Global variables:
*       normal_vector - component of cut attributes structure
*
* Parameter input:
*       point - index of current gauge point
*
* Parameter output:
*       none
*
* Non-System Function calls:
*       none
*
*****************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.                          */

    double  i_squared, k_squared, magnitude;

                    /*  Find the magnitude of the vector, and then divide
                        each, the i and k components, by the magnitude.  */

    i_squared = cut_no->normal_vector[point].x
              * cut_no->normal_vector[point].x;
    k_squared = cut_no->normal_vector[point].z
              * cut_no->normal_vector[point].z;

    magnitude = sqrt (i_squared + k_squared);

    cut_no->normal_vector[point].x /= magnitude;
    cut_no->normal_vector[point].z /= magnitude;

    return;
  }
    /* end of normalize_vector */
```

C.3.4 read_cut_attributes

This function stores the attributes of individual cut instructions read from files attrNNV0.1 for semifinish machining and attrNNV0.2 for finish machining.

Function Prototype Declaration:

void read_cut_attributes (int skipped[])

Inputs:

skipped - An array of indices for those gauge points that have been skipped.

Outputs:

Data from the semifinish and finish attributes files are entered into the appropriate data structures.

Function Procedures:

The function first opens the file attrNNV0.1 where NNV are the first three characters of the part number prepared by the function get_part_no. It then reads data from this semifinish attributes file. The values read in from each attribute file are: The number of gauge points on a cut, the cut direction in x, the cut direction in z, the next intersecting cut number, the end point specification coordinate (usually X), the end point coordinate number (usually an odd number), end point specification coordinate (usually Z), the end point coordinate number (usually even), the type of curve, the number of the prior cut, the angle (counterclockwise) it makes with current cut, the number of the next cut and the angle it makes with the current cut. In the case of no gauge points on a cut these last entries are special. In that cas the previous cut and angle are set to 0 and the next cut is set to 0 but the angle is set to the coordinate of the plane. That is, if it is a cut parallel to x-axis it is a z-coordinate, if it is parallel to the z-axis it is an x-coordinate. The only cuts that can have 0 or 1 gauge points are vertical or horizontal cuts. Therefore each cut is tested for this. If there are gauge points for a given cut then the coordinates and normal vectors at these gauge points are read. Once all of the data in attrNNV0.1 is read the function opens attrNNV0.2, the attributes for the finished part, and repeats the previous data entry.

```
/*****************************************************************************
  Function to record attributes of individual cut instructions.
*****************************************************************************/

void  read_cut_attributes (int skipped[])

/*****************************************************************************
*
* Global variables:
*     attrib - cut attributes structure
*
* Parameter input:
*     skipped - array of skipped point indices (max 3)
*
* Parameter output:
*       none
*
* Non-system Function calls:
*       normalize_vector
*
*****************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.

                        The variable "overall_point_no" counts all the
                        points expected to be gauged.  The variable
                       "total_points" counts only the points actually
                        gauged, skipping those missed by the probe.        */

    int  i, j, point, overall_point_no;
    char  dummy[81];
    FILE  *in_file;
    static char  digit[11] = "0123456789";
    static char  attrib_file[] = "\\qia\\probing\\attrNNV0.N";

                    /*  Set up loop to read entire attrNNV0.N file series.  */

    attrib_file[22] = digit[1];
    attrib_file[17] = part_no[0];
    attrib_file[18] = part_no[1];
    attrib_file[19] = part_no[2];

    if ( (in_file = fopen(attrib_file, "r")) == NULL )
      {
        sprintf(dummy,
                    "\nNo attributes file exists for Part %s.\n", part_no);
        prnwin(dummy, LIGHTRED, 1);
        delay(5000);
        exit (2);
      }
```

```
                /*  Loop over semifinish and finish sessions.  A file
                    required for each session.                          */

i = 0;
do
  {
                /*  Associate pointers with data storage assignments.   */

    for ( j = 0;  j < max_cuts; ++j )
      {
        attrib[i][j].endpoint_spec = dummy_endpoint_spec[i][j];
        attrib[i][j].type_of_curve = dummy_type_of_curve[i][j];
        attrib[i][j].other_planes = dummy_other_planes[i][j];
        attrib[i][j].nominal_coord = dummy_nominal_coord[i][j];
        attrib[i][j].normal_vector = dummy_normal_vector[i][j];
        attrib[i][j].gauged_coord = dummy_gauged_coord[i][j];
        attrib[i][j].error_scalar = dummy_error_scalar[i][j];
        attrib[i][j].adjusted_coord = dummy_adjusted_coord[i][j];
        attrib[i][j].poly_coeff = dummy_poly_coeff[i][j];
      }

                /*  Read data into structures from file.  Note that the
                    cut number is being incremented within each
                    session.  The values read in from each attributes
                    file are: The number of gauge points on a cut, the
                    cut direction in x, the cut direction in z, the
                    next intersecting cut number, the end point spec
                    coord (usually X), the end point coord number (odd
                    number), end point spec coord (usually Z), the
                    endpoint coord number (usually even), degree of x
                    curve, degree of z curve, number of prior cut,
                    angle (counterclockwise) it makes with current cut,
                    number of next cut, angle it makes with current
                    cut.   NOTE: In the case of no gauge points on a
                    cut these last entries are special. The previous
                    cut and angle are set to 0 and the next cut is set
                    to 0 but the angle is set to the coordinate of the
                    plane; i.e., if it is a cut parallel to x axis it
                    is a z coord, if it is parallel to the z axis it is
                    an x coord.                                         */

    total_points = 0;
    overall_point_no = 0;
    for ( cut_no = attrib[i];  !(feof (in_file));  ++cut_no )
      {
        fscanf (in_file,
                 "%d %1s %1s %d %1s %d %1s %d %lf %lf %d %lf %d %lf\n",
            &(cut_no->how_many_points),
            &(cut_no->cut_direction.x), &(cut_no->cut_direction.z),
            &(cut_no->intersect_cut),
            &(cut_no->endpoint_spec[0].coord),
                                  &(cut_no->endpoint_spec[0].spec),
            &(cut_no->endpoint_spec[1].coord),
```

```
                                &(cut_no->endpoint_spec[1].spec),
    &(cut_no->type_of_curve[0]), &(cut_no->type_of_curve[1]),
    &(cut_no->other_planes[0].plane),
                                 &(cut_no->other_planes[0].angle),
    &(cut_no->other_planes[1].plane),
                                &(cut_no->other_planes[1].angle));

        /*  For cuts with no gauge points and a z cut direction
            the x coordinate must be converted to radius mode   */

if ( cut_no->how_many_points == 0  &&
                                cut_no->cut_direction.x == '0'  &&
                                cut_no->cut_direction.z != '0' )
cut_no->other_planes[1].angle /= 2.0;

        /*  The only cuts that can have 0 or 1 gauge points are
            vertical or horizontal cuts. All others must have
            two or more.                                        */

if ( (cut_no->how_many_points == 0  ||
      cut_no->how_many_points == 1)  &&
                               (cut_no->cut_direction.x != '0'  &&
                                cut_no->cut_direction.z != '0') )
  {
    sprintf(dummy, "There must be 2 or more gauge points "
                                    "on a taper or dome cut.");
    prnwin(dummy, LIGHTRED, 1);
    delay(5000);
    exit(1);
  }
        /*  If there are gauge points then the coordinates and
            components of the normal vector at the points are
            read in. The x coordinate is a diameter value
            to suit the NC program.  Note that the nominal
            coordinates are for the part surface, not the tool
            nose center.                                        */

for ( point = 0;  point < cut_no->how_many_points;  ++point )
  {
    fscanf (in_file, "%lf %lf %lf %lf",
        &(cut_no->nominal_coord[point].x),
        &(cut_no->nominal_coord[point].z),
        &(cut_no->normal_vector[point].x),
        &(cut_no->normal_vector[point].z));

        /*  Make radius mode                                    */

    cut_no->nominal_coord[point].x /= 2;

        /*  Make unit normal vectors                            */

    normalize_vector (point);
```

```
                    /*  Adjust counts for skipped points                    */

                for ( j = 0;  j < 3;  ++j )
                  {
                    if ( overall_point_no == skipped[j] )
                      {
                        --(cut_no->how_many_points);
                        --point;
                      }
                  }
                ++overall_point_no;
              }

            total_points += cut_no->how_many_points;
          }

        fclose (in_file);
        total_cuts[i] = cut_no - attrib[i];

                    /*  Set up file name for the next file to be read       */

        attrib_file[22] = digit[(++i)+1];
        sessions = i;
      }
    while ( (in_file = fopen (attrib_file, "r")) != NULL );

    return;
  }
    /* end of read_cut_attributes */
```

C.3.5 rtec_to_part

This function converts RTEC output to part program coordinates. The coordinates are adjusted to take into account the probe stylus radius.

Function Prototype Declaration:

double rtec_to_part (double rtec_value, char axis, double
theta,double probe_x_length, double
MACHINE_HOME_X)

Inputs:

rtec_value - gauged value of the coordinate specified in axis
passed through in inches.

axis - character identifying axis of gauged point, either
x or z.

theta - angle measured counterclockwise in radians of the surface normal at the gauged point with respect to the z-axis on the NC lathe. Note: z-axis parallel to the spindle axis.

probe_x_length - vertical length of the probe in inches.

MACHINE_HOME_X - x-distance from the machine "home" to the part origin.

Outputs:

The function returns a value for the current part-program coordinate in inches for the specified axis in the calling parameters.

Function procedures:

If the axis selected is the "x" axis then the value returned is computed as

$$X_{part}^{d} = 2.0\,(X_{mach} + r(1 - \sin\theta) - X_{probe} + MACHINE\_HOME\_X) \quad (8.1)$$

If the axis selected is the "z" axis then the value returned is computed as

$$Z_{part} = Z_{mach} + r(1 - \cos\theta) - Z_{diff} \quad (8.2)$$

where Z_{diff} is represented in the program by the global variable z_difference and is computed in the function read_rtec.

```
/*************************************************************************
  Function to convert RTEC output to part program coordinates. The
    coordinates are adjusted to take into account the probe radius.
*************************************************************************/

double  rtec_to_part (double rtec_value, char axis, double theta,
                              double probe_x_length, double MACHINE_HOME_X)
/*************************************************************************
*   For each coordinate at a time, this function first
*   adjusts the RTEC value to the actual point of contact
*   between the part surface and the probe stylus.  (This
*   adjustment depends on theta, the angle of the surface
*   normal vector.)
*
*   Global variable:
*       z_difference -  is equal to the RTEC
*                       value corresponding to the z-coordinate of a point,
*                       minus the z-coordinate in terms of the part
*                       coordinate system.  The value of z_difference was
*                       calculated the function read_rtec.
*
*   Parameter Input:
*       rtec_value -  gauged value of the coordinate specified in axis
*                     passed through in inches.
*
*       axis       -  character identifying axis of gauged point, either
*                     X or Z.
*       theta      -  angle measured counterclockwise in radians of the
*                     surface normal at the gauged point with respect to
*                     the Z axis on the NC lathe. Note: Z-axis parallel to
*                     the spindle axis.
*
*       probe_x_length - vertical length of the probe in inches.
*
*   Parameter output:
*       rtec_to_part - value of RTEC gauged point in machine coordinates to
*                      parts coordinates.
*
* Non-system Function calls:
*       none
*
*************************************************************************/
  {
    double  part_prog_coord;

    if ( axis == 'x'  ||  axis == 'X' )

                    /*  Double coordinates because of diameter mode. That
                        is the RTEC uses x coordinates in the radius mode
                        relative to the machine coordinate system. They
                        need to be converted to diameter mode when sending
                        them to the CNC controller.                    */
```

```
    {
      rtec_value += ( probe_rad * (1. - sin (theta)) );
      part_prog_coord =
                    2. * (rtec_value - probe_x_length + MACHINE_HOME_X);
    }
  else /* ( axis == 'Z' )                                              */
    {
      rtec_value += ( probe_rad * (1. - cos (theta)) );
      part_prog_coord = rtec_value - z_difference;
    }
  return part_prog_coord;
}
  /* end of rtec_to_part */
```

C.3.6 read_rtec

This function reads the file of RTEC data and an associated parameter file. The RTEC file contains x and z-coordinates of gauged points in the machine coordinate system in units of micrometers. It produces the value for z_difference used in rtec_to_part as well as the gauged coordinate vector in the part coordinate frame for each gauged point on each cut.

Function Prototype Declaration:

char read_rtec (char seq[],double temp[], double MACHINE_HOME_X, int *tool)

Inputs:

seq[] - A string representing the gauging sequence number.

temp[] - array of temperatures, in celsius, representing the temperature state of the machine at the time of gauging.

MACHINE_HOME_X - distance from machine "home" to part origin.

*tool - pointer to the tool number.

Outputs:

The function directly returns a character for the order in which the gauging points are to be measured. An "s" is returned if the order is the same as the order of the cut direction and an "r" is returned if in reverse order. Indirectly, the function converts the gauged points from machine coordinates to part coordinates.

Function Procedures:

The function first reads the RTEC values from the file GAUGING.K, where K is the numeric value of seq[], into a two dimensional array, xz_rtec[max_points][2], after converting from micrometers to inches. A total count of all coordinate points read is computed andstored in the variable total_single_coords. The parameter file PRAMNNV0.REF is then opened where NNV are the first three characters of the part ID. The following values are read for the probed points: *ref_point* - an integer indicating which point in the list of gauged points is to be used as the reference point, *probe_x_length* - x length of probe in inches, *order* - a character r or s indicating the order in which the gauged points were taken relative to the cuts made, *r* for reverse, *s* for same, and finally *probe_rad* - the probe radius in inches. For an example of this file see Sections 6.1 and 6.2.

For whatever segment number, if the order is "r" then the points in xz_rtec are reversed. A

reference subscript, ref_subscript, is set for the reference point, ref_point. For example, if the reference point was the first point, ref_point = 1, and ref_point subscript = 0. Suppose 23 pairs of points had been read, then if the order is "r" the reference point subscript would be 22 and if the points are read in "s" the reference point subscript would be 0.

The program loops through each cut and points-per-cut to find the index of the point designated as the part reference point. Once found the x and z part coordinate values for this point are stored and the angle of the normal vector at that point is calculated relative to the z-axis.

As soon as the part reference z is known the quantity z_difference, used in the rtec_to_part, is computed. It is computed as the difference between the machine z-coordinate for the reference point and the part coordinate z value for the reference point plus an adjustment for the probe radius.

If the probe length is entered as 0 in the parameter file then the program computes the x-length of the probe. Otherwise the program continues to the next step.

The program next makes G-T adjustments to the x and z machine coordinates at the current gauge point. It then computes the angle of the normal vector at that point, which is needed to convert a machine coordinate to a part coordinate. Then the program calls the function rtec_to_part to compute the x and z part coordinates adjusted for geometric-thermal error. These are stored in the gauged coordinates portion of the cut attributes structure for the cut of interest.

```
/*************************************************************************
  Function to read the file of RTEC data and the associated parameter file.
    The RTEC file contains x, z coordinates of gauged points in the machine
    coordinate system in units of micrometers. The x values are in radius
    mode.  This function produces the value for z_difference used in
    rtec_to_part as well as the gauged_coord vector for each gauged point
    on each cut_no.
*************************************************************************/

char  read_rtec (char seq[], double temp[], double MACHINE_HOME_X, int *tool)

/*************************************************************************
*
* Global variables:
*       attrib      -  global attributes structure for each cut
*       cut_no      -  pointer used to point to each apprpriate
*                      attributes structure.
*       probe_rad   -  probe radius in inches. See DEFINE statements.
*
* Parameter input:
*       none
*
* Parameter output:
*       read_rtec   -  character indicating order in which points were probed
*                      relative to the order in which cuts were made
*
* Non-system Function calls:
*       thermal_adjust, rtec_to_part
*
*************************************************************************/

  {
                    /*  Function prototype                              */

    float  thermal_adjust (long x_coord_offset[], long z_coord_offset[],
        long x_machine, long z_machine, long x_previous, long z_previous,
        long therm_error[], double temperature[], double temp_init[],
        double mach_offset_x, double mach_offset_z, int spindle_sp);

                    /*  Descriptions of global variables used are at the
                        beginning of this file.

                        RTEC data contained in xz_rtec are machine
                        coordinates, but with the offset used by the RTEC.
                        Subscripts are:  [point no.][coordinate axis].      */

    int  ref_subscript, i, j, total_single_coords, k_int;
    char  dummy[81], gauge_file[35];
    long  therm_error[2], curr_x, curr_z, old_x, old_z, xz[2];
    FILE  *in_file;
    double  xz_rtec[max_points][2], dummy_xz_rtec[max_points][2], *pointr,
        theta, part_ref_x, part_ref_z, tempinit[40], mach_x_offset,
```

```
    mach_z_offset;
static int  ref_point;
static char  order;
static char  gauge[] = "\\qia\\probing\\gauging.";
static char  param_file[] = "\\qia\\probing\\pramNNV0.ref";
static double  probe_x_length, probe_z_length;

                /*  Read in all the coordinate data from the file
                    gauging.K, where K is a sequence number.  The
                    number of coordinate pairs which should be found in
                    the file is half of "total_single_coords", the
                    integer passed to this function as a parameter.  In
                    the case of gauging data, there should be as many
                    coordinate pairs in the file as there were gauging
                    points.                                          */

strcpy(gauge_file, gauge);
strncat(gauge_file, seq, 7);
in_file = fopen (gauge_file, "r");
total_single_coords = 0;

                /*  Skip the two leading lines from the gauging file.
                    Note that there will be a blank line before them.
                    Part_no was picked off earlier.                  */

fgets(dummy, 80, in_file);
fgets(dummy, 80, in_file);
fgets(dummy, 80, in_file);

                /*  Read coordinate index and coordinates for each
                    probe point in micrometers. Convert to inches.  The
                    coordinates are in absolute machine coordinates.    */

i = 0;
k_int = 0;
while ( !feof(in_file) )
  {
    fscanf(in_file, "%d %ld %ld", &j, &xz[0], &xz[1]);
    if ( j != k_int+1 )
        break;      /* This is a test to make sure the loop terminates. */
    k_int = j;
    xz_rtec[i][0] = xz[0];
    xz_rtec[i][1] = xz[1];
    xz_rtec[i][0] /= 25400;
    xz_rtec[i][1] /= 25400;
    total_single_coords += 2;
    ++i;
  }
fclose (in_file);

                /*  Read in the parameter file for the probed points:
                    ref_point - integer indicating which point in the
                    list of gauged points is to be used as the
```

```
                    reference point probe_x_length - X length of probe
                    in inches order - character r or s indicating the
                    order in which the gauged points were taken
                    relative to the cuts made. r for reverse, s for
                    same.                                              */

param_file[17] = part_no[0];
param_file[18] = part_no[1];
param_file[19] = part_no[2];

in_file = fopen (param_file, "r");
fscanf (in_file, "%d %lf %lf %1s %*f %lf",
     &ref_point, &probe_x_length, &probe_z_length, &order, &probe_rad);
fclose (in_file);

                /*  If the order of measurements is the reverse of the
                    point number sequence used here then reverse order.
                    That is if the points probed are opposite to the
                    their order in the direction of cut then they are
                    reversed.                                          */

if ( order == 'r'  ||  order == 'R' )
  {
    direction = -1;
    ref_subscript = (total_single_coords / 2) - ref_point;
    for ( i = 0;  i < (total_single_coords / 2);  ++i )
      {
        dummy_xz_rtec[i][0] = xz_rtec[i][0];
        dummy_xz_rtec[i][1] = xz_rtec[i][1];
      }
    for ( i = 0;  i < (total_single_coords / 2);  ++i )
      {
        xz_rtec[i][0] =
                        dummy_xz_rtec[total_single_coords/2 - 1 - i][0];
        xz_rtec[i][1] =
                        dummy_xz_rtec[total_single_coords/2 - 1 - i][1];
      }
  }
else
  {
    direction = 1;
    ref_subscript = ref_point - 1;
  }

                /*  Calculate z_difference for later use.  Also
                    calculate the X-length of the probe if it was not
                    provided in the file pramNNV.ref.                  */

j = 0;
for ( cut_no = attrib[0];  j <= ref_subscript;  ++cut_no )
    for ( i = 0;  i < cut_no->how_many_points;  ++i )
        if ( j++ == ref_subscript )
          {
```

```
            part_ref_x = cut_no->nominal_coord[i].x;
            part_ref_z = cut_no->nominal_coord[i].z;
            theta = atan2 (cut_no->normal_vector[i].x,
                                        cut_no->normal_vector[i].z);
            break;
          };
z_difference = xz_rtec[ref_subscript][1] - part_ref_z +
                                    ( probe_rad * (1. - cos (theta)) );

                /*  Calculate X-length of probe if not read in from
                    file.                                              */

if ( fabs(probe_x_length) < .001 )
    probe_x_length = MACHINE_HOME_X + xz_rtec[ref_subscript][0] +
                        ( probe_rad * (1. - sin (theta)) ) - part_ref_x;

                /*  Read in the temperatures for a cold machine.       */

if ( (in_file = fopen("\\qia\\toolset\\coldvals.ref", "r")) == NULL )
  {
    sprintf(dummy, "Cold machine temperatures not found in file "
                                        "\\QIA\\TOOLSET\\COLDVALS.REF");
    prnwin(dummy, LIGHTRED, 1);
    delay(5000);
    return 1;
  }

fgets(dummy, 80, in_file);
for ( j = 0;  j < 4;  ++j ) fscanf(in_file, "%*s");
for ( j = 0;  j < 40;  ++j ) fscanf(in_file, "%lf", &tempinit[j]);
fclose(in_file);

                /*  Read in coordinate system offsets                  */

if ( (in_file = fopen("\\qia\\toolset\\initvals.ref", "r")) == NULL )
  {
    sprintf(dummy, "Offset array may not be determined without file "
                                        "\\QIA\\TOOLSET\\INITVALS.REF");
    prnwin(dummy, LIGHTRED, 1);
    delay(5000);
    return 1;
  }

fgets(dummy, 80, in_file);
fscanf(in_file, "%*lf %*lf %ld %ld %ld %ld", &x_coord_offset[5],
          &z_coord_offset[5], &x_coord_offset[4], &z_coord_offset[4]);
for ( j = 2;  j > 0;  --j )
    fscanf(in_file, "%ld %ld", &x_coord_offset[j], &z_coord_offset[j]);
fclose(in_file);

                /*  Read in the spindle speeds.                        */

sprintf(dummy, "\\QIA\\CUTTING\\CUT%s.REF", part_no);
```

```
    if ( (in_file = fopen(dummy, "r")) == NULL )
      {
        sprintf(dummy, "Part coordinates may not be determined without "
                                "file \\QIA\\CUTTING\\CUT%s.REF", part_no);
        prnwin(dummy, LIGHTRED, 1);
        delay(5000);
        return 1;
      }

    fscanf(in_file, "%*lf %d", tool);
    fgets(dummy, 20, in_file);
    fscanf(in_file, "%*lf %d", &spindle_sp);
    fclose(in_file);

                    /*  Compute machine offsets                          */

    mach_x_offset = 0.0;
    mach_z_offset = 0.0;

                    /*  Now make geometric-thermal adjustments to gauged
                        coordinates, transform them to the part coordinate
                        system, and then load them into the cut_attributes
                        data structure, attrib for the semi-finished cut.

                        In the following loops, i counts points within a
                        specific cut, while j counts the points overall.
                        Theta is the angle that the surface normal of a
                        point makes with the +Z axis.

                        The current code does not correct for geometric-
                        thermal errors although lines are included to show
                        locations for future inclusion                   */

    j = 0;
    for ( cut_no = attrib[0];
                          (cut_no - attrib[0]) < total_cuts[0];  ++cut_no )
      {
        for ( i = 0;  i < cut_no->how_many_points;  ++i )
          {
            theta = atan2 (cut_no->normal_vector[i].x,
                           cut_no->normal_vector[i].z);
            curr_x = /*(long)*/(xz_rtec[j][0] * 25400);   /* inch to micron */
            old_x = curr_x + direction*3;     /* create a previous x in um */
            curr_z = /*(long)*/(xz_rtec[j][1] * 25400);  /* inch to micron */
            old_z = curr_z - direction*3;    /* create a previous z in um */

                    /*  Compute the thermal errors at the current probe
                        point.                                           */

            therm_error[0] = 0;
            therm_error[1] = 0;

//          thermal_adjust(x_coord_offset, z_coord_offset, curr_x, curr_z,
```

```
//              old_x, old_z, therm_error, temp, tempinit, mach_x_offset,
//              mach_z_offset, spindle_sp);

            xz_rtec[j][0] =
                          xz_rtec[j][0] + (double)(therm_error[0])/25400.;
            xz_rtec[j][1] =
                          xz_rtec[j][1] + (double)(therm_error[1])/25400.;

                    /*  Convert thermally adjusted gauge points to part
                        coord.

                        This call puts the x coordinate of a gauge point
                        into diameter mode.                                 */

            cut_no->gauged_coord[i].x = rtec_to_part (xz_rtec[j][0], 'x',
                                      theta, probe_x_length, MACHINE_HOME_X);

            cut_no->gauged_coord[i].z = rtec_to_part (xz_rtec[j][1], 'z',
                                      theta, probe_x_length, MACHINE_HOME_X);

            if ( ++j >= total_single_coords/2 ) break;
          }
      }                                     /* The above "break" is okay     */
                                            /* because the next execution of */
    return order;                           /* the function will be proper.  */
  }
    /* end of read_rtec */
```

C.3.7 skipped_points_test

This function tests whether any expected probe points were missed during the probing process of a session.

Function prototype declaration:

```
int skipped_points_test (double i_component, double k_component,
                         double normal_component, int skipped[],
                         int point)
```

Inputs:

i_component - x component of the error at point.

k_component - z component of the error at point.

normal_component - projection of the error vector on the normal vector.

skipped - array of skipped points indices (max 3).

point - index of the current gauge point.

Outputs:

Subscript of the array skipped[] if the current gauge point was skipped.

Function procedures:

The function first adjusts the point count to correct for any prior adjustments to the skipped[] array. The calling function does not adjust this point count if a gauge point is found to be skipped. It is only done when skipped_points_test is called.

In the next step the function computes the magnitude of the error vector and uses this to compute the tangent component to the surface of the error vector at the gauge point.

The subscript of the skipped[] array is set. If no points are skipped it is left at 0. Otherwise it is incremented and if it becomes greater than 3 an error message is sent to the console that too many gauge points have been skipped and the program aborts.

```
/*************************************************************************
  Function to test whether any expected points were missed during probing.
*************************************************************************/

int  skipped_points_test (double i_component, double k_component,
             double normal_component, int skipped[], int point)
/*************************************************************************
*
* Global variables:
*       none
*
* Parameter inputs:
*       i_component  - x component of the error at point
*
*       k_component  - z component of the error at point
*
*       normal_component -
                       projection of the error vector on the normal vector
*
*       skipped      - array of skipped points indices (max 3)
*
*       point        - index of current gauge point
*
* Parameter output:
*       skipped_points_test - subscript of skipped point if any
*
* Non-system Function calls:
*       none
*
*************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.

                        The variable "count" will count how many gauging
                        points have apparently been skipped.  It is
                        initialized to 1, and the program will abort if
                        it is incremented beyond 3.                      */

    int  i, skip_subscript;
    double  error_magnitude, parallel_component;
    static int  count = 1;

                    /*  The value "point" coming into this function should
                        be the overall point number.  However, if a point
                        was found to be skipped before this particular call
                        to this function, the overall point count would be
                        one less than what it should actually be; two less
                        if there were two previously skipped points.  The
                        following loop will make appropriate adjustments
                        if neccessary.                                   */
```

```
  for ( i = 0;  i < 2;  ++i )
  if ( skipped[i] != -1 ) ++point;

                  /*  The main basis for assuming that the probe has
                      skipped a point which was intended to be measured
                      is that the error vector has a large component
                      parallel to the part surface.  Since the probe
                      should approach a point in a direction normal to
                      the part surface, the measured machining error
                      should all be in + or - the direction of a vector
                      normal to the surface at that point.

                      Any component parallel to the surface must be due
                      to a problem with the probing process.  If such a
                      component is large (say, greater than 1/16 inch,
                      assuming that any ordinary gauging problem will
                      result in a much smaller value, and that this is
                      the minimum spacing which will be programmed
                      between gauging points), then, it seems probable
                      that the point being gauged is not the one
                      expected.  In this case, the nominal coordinates
                      for the point apparently skipped will be eliminated
                      from the array so that the measured and nominal
                      coordinates match in sequence.  The new matching
                      coordinates will then be tested similarly.

                      The only role in this strategy played by this
                      particular function is that of testing.          */

  error_magnitude = sqrt (i_component * i_component +
                          k_component * k_component);

  parallel_component = sqrt (error_magnitude * error_magnitude -
                                     normal_component * normal_component);

  skip_subscript = count - 1;
  if ( parallel_component > .06 )
    {
      if ( count > 3 )
        {
          prnwin("More than 3 gauging points "
                                "were skipped by the probe.", LIGHTRED, 1);
          delay(5000);
          exit (8);
        }
      skipped[(count++)-1] = point;
    }
  return skip_subscript;
}
  /* end of skipped_points_test */
```

C.3.8 calculate_errors

This function calculates deviations of actual from nominal gauging coordinates in the direction of the normal vector. It computes the error vector at each gauged point and produces the error_scalar multiplier for the normal vector at each gauged point for each cut.

Function prototype declaration:

void calculate_errors (int skipped[])

Inputs:

skipped[] - Array of skipped points indices.

Outputs:

The function does not directly return any values. Indirectly the error vectors and the error scalars are computed for each of the gauge points and the attribute structure components are updated with these values. The skipped[] array is incremented for any identified skipped points. Note that the error scalar is just the dot product of the error vector with the surface normal since both are assumed to be unit vectors.

Function procedures:

An overall point count is initialized. The function then sequentially goes through each cut in the session and each point-per-cut until the overall point count exceeds the total number of points for the session. At each point the x and z-components of gauged point minus the nominal point are computed. Note that the x gauged coordinate, store in diameter mode, is divided by 2 to convert it to radial mode for use in this function. The magnitude of the error along the normal vector is computed as the dot product of the error vector and the normal vector. Finally a skipped point test is performed.

```
/*****************************************************************************
  Function to calculate deviations of actual from nominal gauging
    coordinates in the direction of the normal vector. This function
    computes the error vector at each gauged point and produces the
    error_scalar multiplier for the normal vector at each gauged point for
    each cut_no.
*****************************************************************************/

void  calculate_errors (int skipped[])

/*****************************************************************************
*
* Global variables:
*       attrib - cut attributes structure
*
*       cut_no - pointer to a cut
*
* Parameter input:
*       skipped    - Array of skipped points indices
*
* Parameter outputs:
*       none
*
* Non-system Function calls:
*       skipped_points_test
*
*****************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.                          */

    int  point, overall_point_no, count;
    double  error_vector_i, error_vector_k;

    overall_point_no = 0;

                    /*  Cycle through all of the cuts.                   */

    for ( cut_no = attrib[0];
                          (cut_no - attrib[0]) < total_cuts[0];  ++cut_no )

                    /*  Now cycle through all of the gauge points per cut.
                        First calculate the error vector.  Note that the
                        gauged x coordinate must be transformed from
                        diameter mode. It is stored in the diameter mode in
                        the attributes structure.                        */

      {
        for ( point = 0;  point < cut_no->how_many_points;  ++point )
          {
            if ( (overall_point_no) >= total_points ) continue;
```

```
                /*  Gauged_coord stored in diameter mode                */

        error_vector_i = cut_no->gauged_coord[point].x / 2 -
                                        cut_no->nominal_coord[point].x;
        error_vector_k = cut_no->gauged_coord[point].z -
                                        cut_no->nominal_coord[point].z;

                /*  The error scalar is equal to the magnitude
                    (positive or negative) of the component of the
                    error vector which is in the direction of the
                    surface normal.  This would be equal to the dot
                    product of the two vectors divided by the magnitude
                    of the surface normal, but since the surface normal
                    is equal to unity, the error scalar is equal to the
                    dot product of the two vectors.                     */

        cut_no->error_scalar[point] =
            error_vector_i * cut_no->normal_vector[point].x +
            error_vector_k * cut_no->normal_vector[point].z;

                /*  Check on skipped points                             */

        count = skipped_points_test (error_vector_i, error_vector_k,
             cut_no->error_scalar[point], skipped, overall_point_no++);
        if ( skipped[count] != -1 ) return;
      }

    }

                /*  Move error scalar data to the finished cut
                    attributes.                                         */

  for ( cut_no = attrib[0];
                      (cut_no - attrib[0]) < total_cuts[0];  ++cut_no )
     for ( point = 0;  point < cut_no->how_many_points;  ++point )
         (attrib[1] + (cut_no - attrib[0]))->error_scalar[point]
                                  = cut_no->error_scalar[point];

  return;
}
  /* end of calculate_errors */
```

C.3.9 write_report

This function writes a file report.txt that records the part number, time and the magnitudes of the error vectors normal to the surface at each of the gauge points.

Function prototype declaration:

void write_report (char point_order)

Inputs:

point_order - character r or s designating order of probed points relative to cuts

Outputs:

The file report.txt

Function Procedures:

The function first opens the file report.txt for writing. Next it writes a header text that includes the part number and session time. Then for the semifinish session the function writes magnitudes of the errors in the normal direction for each of the gauge points in the order in which they were taken. The text file is then closed.

```
/*****************************************************************************
  Function to write file comparing errors in the direction of the normal
    vector at each of the gauge points. This function writes out the
    error_scalar multiplier at each gauged point.
*****************************************************************************/

void  write_report (char point_order, int seq_no)

/*****************************************************************************
*
* Global variables:
*       attrib   - attributes structure
*
* Parameter input:
*       point_order  - character r or s designating order of probed points
*                      relative to cuts
*
* Parameter output:
*       none
*
* Non-system Function calls:
*       none
*
*****************************************************************************/


  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.                          */

    int  session_no, point, point_no, tot_points, i;
    char  buffer[75];
    long int  ltime;
    FILE  *out_file;
    double  error_scalar[max_points][max_sessions],
            dummy_scalar[max_points][max_sessions];
    static char  probing_results_file[] = "\\QIA\\PROBING\\REPORT.***";

                    /*  Assemble the output file name and prepare to start. */

    itoa(seq_no, buffer, 10);
    for ( i = 0;  i < 3;  i++ )
        probing_results_file[i+20] = buffer[i];

                    /*  Open file and write date, time and title.           */

    time (&ltime);
    out_file = fopen (probing_results_file, "w");
    fprintf (out_file,
       "\n        Record of Measured Errors\
        \n      From Manufacture of Part %s on\
        \n        %s\
```

```
      \n*************************************\
      \n\nGauging\nSession ->", part_no, ctime (&ltime));

                  /*  Loop for each point within each cut within each
                      session, storing the error values and writing the
                      session numbers in the column headings.            */

  session_no = 1;

  point_no = 0;
  for ( cut_no = attrib[session_no-1];
       (cut_no - attrib[session_no-1]) < total_cuts[session_no-1];
                                                          ++cut_no )
      for ( point = 0;  point < cut_no->how_many_points;  ++point )
          error_scalar[++point_no][session_no] =
                                          cut_no->error_scalar[point];
  fprintf (out_file, "          %d", session_no);

  tot_points = point_no;

                  /*  If measurements were taken in the reverse order of
                      the point number sequence used here --             */

  if ( point_order == 'r'  ||  point_order == 'R' )
    {
      for ( point_no = 1;  point_no <= tot_points;  ++point_no )
          dummy_scalar[point_no][session_no] =
                                    error_scalar[point_no][session_no];
      for ( point_no = 1;  point_no <= tot_points;  ++point_no )
          error_scalar[point_no][session_no] =
                        dummy_scalar[tot_points+1-point_no][session_no];
    }
                  /*  Write the errors to the file in rows pertaining to
                      each point.                                        */

  for ( point_no = 1;  point_no <= tot_points;  ++point_no )
    {
      fprintf (out_file, "\n\nPoint %d      ", point_no);
      fprintf (out_file, "%12.6f", error_scalar[point_no][session_no]);
    }
  fclose (out_file);

  return;
}
  /* end of write_report */
```

C.3.10 read_toolpath_data

This function reads in "go-to" coordinates and other tool path specifications for the part program segment for the NEXT session. The "go-to" coordinates are the nominal endpoints of cuts. It initializes an array toolpath_spec with the nominal endpoint vectors for each cut. If tool offsets are read then they are put into the first two elements of toolpath_spec.

Function Prototype Declarations:

```
void  read_toolpath_data (int next_segment, struct toolpath_data cut_coords[],
                                    int init_block_spec[], int initial_block_cut[],
                                     int final_block_cut[])
```

Inputs:

next_segment - NC code segment to be generated.

cut_coords - coordinates of start- and end-points of cuts.

init_block_spec - tool path spec numbers in terms of cut numbers.

initial_block_cut - cut for which a tool path spec is initial.

final_block_cut - cut for which a tool path spec is final.

Outputs:

Directly the function does not return any value. Indirectly the function returns the start and end points of each cut, expressed in terms of the cuts.

Function Procedures:

The function begins by opening the file PATHNN00.2 where NN represents the first two characters of the part number. An example of this file is given in Section 6.5. Each record read has four fields. In the first record the first two values are usually set to 0 and represent z and x offset. These two values in the file are dummy values, since the actual offset increments are calculated in PIECS. The numbered specifications (e.g., "go-to" coordinates) in the file ptNN0021.in, the part program segment template used by PIECS, begin with 3 (after the two tool offset specifications). Each of the next records have x and z-coordinates of a point that is the beginning point of the cut numbered in the third field and the end point of the cut numbered in the fourth field. The function then computes the start and end points of each cut.

```
/*************************************************************************
  Function to read in "go-to" coordinates and other tool path
    specifications for the part program segment for the NEXT session. The
   "go-to" coordinates are the nominal endpoints of cuts. This function
    initializes toolpath_spec with the nominal endpoint vectors for each
    cut.  If tool offsets are read then they are put into the first two
    elements of toolpath_spec.
*************************************************************************/

void  read_toolpath_data (int next_segment,
   struct toolpath_data cut_coords[], int init_block_spec[],
   int initial_block_cut[], int final_block_cut[])

/*************************************************************************
*
* Global variables:
*       part_no   -  part number
*
*       toolpath_spec  - go-to points for NC segment (These will be adjusted)
*
*       toolpath_spec_nom - go-to points for NC segment and for RTEC file
*
* Parameter inputs:
*       next_segment - NC code segment to be generated
*       cut_coords - coordinates of start- and end-points of cuts
*       init_block_spec - tool path spec numbers in terms of cut numbers
*       initial_block_cut - cut for which a tool path spec is initial
*       final_block_cut - cut for which a tool path spec is final
*
* Parameter outputs:
*       none
*
* Non-system Function calls:
*       none
*
*************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.                          */

    int  i, spec;
    FILE  *in_file;
    char dummy1[81];
    double  nominal_spec_x[max_cuts], nominal_spec_z[max_cuts];
    static char  toolpath_file[] = "\\qia\\cutting\\pathNN00.2";

                    /*  This function is called only when there will be at
                        least one more machining-gauging session.
                        Therefore it is fine for this function to consider
                        the next segment, which is the first in the next
                        session.                                         */
```

```
toolpath_file[17] = part_no[0];
toolpath_file[18] = part_no[1];

if ( (in_file = fopen (toolpath_file, "r")) == NULL )
  {
    sprintf (dummy1, "No tool path file exists "
                     "for Part %i Segment %i.\n", part_no, next_segment);
    prnwin(dummy1, LIGHTRED, 1);
    delay(5000);
    exit (4);
  }
                /*  The first two variables (i=0 and i=1) in the array
                    "toolpath_spec[i]" are the Z and X tool offset
                     increments.  The first two values in the tool path
                     file to be read are dummy values, since the actual
                     offset increments are calculated in this program.

                     The numbered specifications (e.g., "go-to"
                     coordinates) in the file pathNN00.2, the part
                     program segment template used by this program,
                     begin with 3 (after the 2 tool offset
                     specifications).  Therefore, the subscript 2 in
                    "toolpath_spec" corresponds to specification 3 in
                     pathNN00.2.                                          */

for ( spec = 0;  !(feof (in_file));  spec += 2 )
  {
    if ( spec >= 2*(total_cuts[1]-1) ) break;
    fscanf (in_file, "%lf %lf %d %d", &nominal_spec_x[spec],
        &nominal_spec_z[spec], &initial_block_cut[spec],
        &final_block_cut[spec]);
    toolpath_spec[spec] = nominal_spec_x[spec];
    toolpath_spec[spec+1] = nominal_spec_z[spec];

                /*  For later use, the spec numbers need to be known
                    from the cut numbers.                                 */

    init_block_spec[initial_block_cut[spec]] = spec;
  }
fclose (in_file);

                /*  The coordinates of the start- and end-points of the
                    cuts will now be expressed in terms of cuts.

                    The left-hand side of the following equations can
                    be easily understood if read backwards.  For the
                    start- and end-point subscripts, [0] refers to the
                    X coordinate; [1] refers to Z.  The other
                    subscripts are specified according to the way
                    values were read in from the tool path file by the
                    function read_toolpath_data.  The first equation
                    reads, for example: The X coordinate of the start-
                    point of cut[spec] equals nominal_spec_x[spec].     */
```

```
  for ( spec = 2;  spec < 2*(total_cuts[1]-1);  spec += 2 )
    {
      cut_coords[initial_block_cut[spec]].startpoint[0] =  /* radius mode */
                                                  nominal_spec_x[spec]/2.;
      cut_coords[initial_block_cut[spec]].startpoint[1] =
                                                     nominal_spec_z[spec];
      cut_coords[final_block_cut[spec]].endpoint[0] =      /* radius mode */
                                                  nominal_spec_x[spec]/2.;
      cut_coords[final_block_cut[spec]].endpoint[1] =
                                                     nominal_spec_z[spec];
    }

  return;
}
  /* end of read_toolpath_data */
```

C.3.11 classify_errors

This function determines the geometric category for which an error correction algorithm will be applied.

Function prototype declaration:

int classify_errors (void)

Inputs:

None.

Outputs:

This function returns the value 1 if at least one of the axis curve degrees is not quadratic and 2 otherwise.

Function procedures:

For each cut this function tests the degree-of-curve value for whether the first value is negative. If it is it returns 1, otherwise 2.

```
/*****************************************************************************
  Function to determine the geometric category according to which an error
    correction algorithm will be invoked.
*****************************************************************************/

int  classify_errors (void)

/*****************************************************************************
*
* Global variables:
*       attrib  - cut attributes structure
*
* Parameter inputs:
*       session_no  - There are two sessions: 1 for semifinish, 2 for finish.
*                     Only 1 should be used.
*
* Parameter outputs:
*       classify_errors - 1 if the degree of curve is not quadratic,
*                         otherwise it is 2.
*
* Non-system Function calls:
*       none
*
*****************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.

                        Test whether any cuts are a nonlinear curve-
                        interpolation type (i.e., machined in a curve-
                        interpolation mode).  (A better means than the
                        method used here for identifying such cuts will be
                        implemented later.)  If such is the case, then an
                        analysis for trueness of curves will be performed.
                        Otherwise that analysis will be skipped, allowing
                        other tests to be performed in succession.          */

    for ( cut_no = attrib[1];
                          (cut_no - attrib[1]) < total_cuts[1];  ++cut_no )
        if ( cut_no->type_of_curve[0] <  0 )
            return 1;

    return 2;
  }
    /* end of classify_errors */
```

C.3.12 untrue_curve

This function is a placemarker for a future function to calculate corrections for untrue curved features.

Function prototype declaration:

void untrue_curve (void)

Inputs:

None.

Outputs:

None.

Function procedures:

None.

```
/*****************************************************************************
  Function to calculate corrections for untrue curves.
*****************************************************************************/

void  untrue_curve (void)

/*****************************************************************************
*
* Global variables:
*       none
*
* Parameter inputs:
*       none
*
* Parameter outputs:
*       none
*
* Non-system Function calls:
*       none
*
*****************************************************************************/

  {
           /*  Descriptions of global variables used are at the
           beginning of this file.                                  */

    return;
  }
    /* end of untrue_curve */
```

C.3.13 max_value

This function calculates the maximum (double precision) floating point value of the array all_values.

Function Prototype Declaration:

double max_value (double all_values[], int how_many)

Inputs:

all_values - array of numbers whose maximum is sought

how_many - number of elements in the array

Outputs:

The function returns the maximum value in the array all_values.

Function Procedures:

The function keeps updating a temporary variable that represents the current maximum in the array. When the last element of the array has been examined it returns the last value in the temporary array.

```
/*****************************************************************************
  Function to calculate maximum (double precision) floating point value.
*****************************************************************************/

double  max_value (double all_values[], int how_many)

/*****************************************************************************
*
* Global variable:
*       none
*
* Parameter inputs:
*       all_values  - array of numbers whose maximum is sought
*
*       how_many    - number of elements in the array
*
* Parameter outputs:
*       max_value   - maximum value in the array
*
* Non-system Function calls:
*       none
*
*****************************************************************************/

  {
    int  n;
    double  max_so_far;

    max_so_far = all_values[0];
    for ( n = 1;  n < how_many;  ++n )
        if ( all_values[n]  >  max_so_far )  max_so_far = all_values[n];

    return max_so_far;
  }
    /* end of max_value */
```

C.3.14 min_value

This function calculates the minimum (double precision) floating point value of an array.

Function Prototype Declaration:

double min_value (double all_values[], int how_many)

Inputs:

all_values - array of numbers whose minimum is sought.

how_many - number of elements in the array.

Outputs:

The function returns the minimum value in an array.

Function Procedures:

The procedure is similar to that in max_value except in this case the temporary variable maintains the current minimum.

```
/*************************************************************************
  Function to calculate minimum (double precision) floating point value.
*************************************************************************/

double  min_value (double all_values[], int how_many)

/*************************************************************************
*
* Global variables:
*       none
*
* Parameter inputs:
*       all_values  -  array of numbers whose minimum is sought
*
*       how_many    -  number of elements in the array
*
* Parameter outputs:
*       min_value   -  minimum value in the array
*
* Non-system Function calls:
*       none
*
*************************************************************************/


  {
    int n;
    double  min_so_far;

    min_so_far = all_values[0];
    for ( n = 1;  n < how_many;  ++n )
        if ( all_values[n]  <  min_so_far )  min_so_far = all_values[n];

    return min_so_far;
  }
    /* end of min_value */
```

C.3.15 smallest_magnitude

This function identifies the subscript of the (double precision) floating point array that has the smallest magnitude.

Function Prototype Declaration:

int smallest_magnitude (double all_values[], int how_many)

Inputs:

all_values - array of numbers whose minimum absolute value is sought

how_many - number of elements in the array

Outputs:

This function returns the index of the element in the array all_values that has the smallest absolute value.

Function Procedures:

The function first takes the absolute values of all elements of the array and temporarily stores them. The it searches the temporary array for the smallest value, keeping track of the index each time.

```
/*****************************************************************************
  Function to identify the subscript of the (double precision) floating point
    value which has the smallest magnitude.
*****************************************************************************/

int  smallest_magnitude (double all_values[], int how_many)

/*****************************************************************************
*
* Global variables:
*       none
*
* Parameter inputs:
*       all_values    -  array of numbers whose minimum absolute value is
*                        sought
*
*       how_many      -  number of elements in the array
*
* Parameter outputs:
*       smallest_magnitude - subscript of element in array with smallest
*                            absolute value
*
* Non-system Function calls:
*       none
*
*****************************************************************************/


  {
    int  n, subscript;
    double  magnitudes[20], smallest_so_far;

    for ( n = 0;  n < how_many;  ++n )
        magnitudes[n] = fabs(all_values[n]);

    smallest_so_far = magnitudes[0];
    subscript = 0;
    for ( n = 1;  n < how_many;  ++n )
        if ( magnitudes[n]  <  smallest_so_far )
          {
            smallest_so_far = magnitudes[n];
            subscript = n;
          }
    return subscript;
  }
    /* end of smallest_magnitude */
```

C.3.16 offset_change

This function determines whether all errors for a single axis are reasonably constant, and if so, it calculates a compensating tool offset increment. If the error range in the array error_value is relatively constant then this function returns a fixed error offset value.

Function Prototype Declaration:

double offset_change (double error_value[], int how_many)

Inputs:

error_value - array of errors.

how_many - number of elements in the array.

Outputs:

This function returns the offset correction for a single axis.

Function Procedures:

Identify subscript of the error value with the smallest magnitude. If the range of error is small, a constant offset can produce a satisfactory correction. Since the current tool offset in the machine tool should be an integer multiple of the machine tool servo resolution, the increment calculated here is also converted to such a multiple. The multiple used will be slightly higher than the actual resolution since this program rounds down for the final figure, and since the machine tool controller will round down again to a multiple of the actual resolution. If the range of error is not small enough, the coordinates in the part program must be modified, so the offset is not changed.

```
/*************************************************************************
  Function to determine whether all errors for a single axis are reasonably
    constant, and if so, to calculate a compensating tool offset increment.
    If the error range in the array error_value is relatively constant then
    this function returns a fixed error offset value.
*************************************************************************/

double  offset_change (double error_value[], int how_many)

/*************************************************************************
*
* Global variables:
*       correction  -  error multiplier fraction to obtain desired tool offset
*                      See DEFINE statements.
*
*       servo_res   -  resolution of the machine tool servo system. See DEFINE
*                      statements.
*
*       tolerance2  -  percentage of the smallest absolute value of error that
*                      can be corrected. See DEFINE's.
*
* Parameter inputs:
*       error_value  -  array of errors
*
*       how_many     -  number of elements in the array
*
* Parameter outputs:
*       offset_change - offset correction
*
* Non-system function calls:
*       max_value, min_value, smallest_magnitude
*
*************************************************************************/


  {
    int  i, smallest;
    double  range, delta_offset;

                    /*  The parameter "tolerance2" represents the widest
                        range of errors that can be corrected by a tool
                        offset.  It is used as a percentage of the error of
                        smallest magnitude.

                        The parameter "correction" is the fraction by which
                        an error will be multiplied to obtain the quantity
                        by which the tool offset is to be changed.          */

    range = max_value (error_value, how_many) -
            min_value (error_value, how_many);

                    /*  Identify subscript of the error value with the
```

```
                    smallest magnitude.                                  */

  smallest = smallest_magnitude (error_value, how_many);

                /*  If the range of error is small, a constant offset
                    can produce a satisfactory correction.               */

  if ( range < fabs(tolerance2 * error_value[smallest]) )
    {
      delta_offset = error_value[smallest] * correction;

                /*  The currently active tool offset should already be
                    an integer multiple of the machine tool servo
                    resolution.  Therefore, the increment calculated
                    here will also be so converted.  The multiple used
                    will be slightly higher than the actual resolution
                    since this program rounds down for the final
                    figure, and since the machine tool controller will
                    round down again to a multiple of the actual
                    resolution.                                          */

      delta_offset -= fmod(delta_offset, servo_res);
    }
                /*  If the range of error is not small enough, the
                    coordinates in the part program must be modified,
                    so the offset will not be changed.                   */
  else
      delta_offset = 0;

  return delta_offset;
}
  /* end of offset_change */
```

C.3.17 Constant_error

This function assigns new tool offsets if it is found that they could significantly decrease errors. This is done by producing the error_scalar times the normal_vector at each gauged point and testing the components for relatively constant values. If so it stores the offset values in the first two elements of toolpath_spec array.

Function Prototype Declaration:

int constant_error (void)

Inputs:

None.

Outputs:

constant_error - a flag indicating whether both toolpath_specs have been set.

Function Procedures:

For all cuts except nonlinear curve-interpolation types, the function first tests whether the range of x-errors and the range of z-errors are both small. If this is the case, then errors may be reduced with x and z offsets and these values will be substituted into the tool offset update command in the next segment of NC code. They will often be zero, and will not matter. When either is (or both are) zero, further corrections will be calculated for the corresponding axis (or both axes) by the untrue planes function.

```
/*****************************************************************************
  Function to assign new tool offsets if it is found that they could
    significantly decrease errors. This is done by producing the
    error_scalar times the normal_vector at each gauged point and testing
    the components for relatively constant values. If so it stores the
    offset values in the first two elements of toolpath_spec.
*****************************************************************************/

int  constant_error (void)

/*****************************************************************************
*
* Global variables:
*       attrib   -  cut attributes structure
*
*       x_error  -  array of x errors for all cuts
*
*       z_error  -  array of z errors for all cuts
*
*       toolpath_spec - array of go-to end points for NC segment
*
* Parameter inputs:
*       session_no -  There are two sessions: 1 for semifinish, 2 for finish.
*                     Only 1 should be used.
*
* Parameter outputs:
*       constant_error - a flag indicating whether both toolpath_specs have
*                        been set.
*
* Non-system function calls:
*       offset_change
*
*****************************************************************************/


  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.                          */

    int  point, pt, both_done;

                    /*  For all cuts except nonlinear curve-interpolation
                        types, test whether the range of X errors and the
                        range of Z errors are both small.  If such is the
                        case, then errors may be reduced with X and Z
                        offsets.                                         */

    pt = 0;
    for ( cut_no = attrib[1];
                          (cut_no - attrib[1]) < total_cuts[1];  ++cut_no )
        if ( cut_no->type_of_curve[0] < 0 )
            for ( point = 0;  point < cut_no->how_many_points;  ++point )
```

```
            {
              x_error[++pt] = cut_no->error_scalar[point] *
                                        cut_no->normal_vector[point].x;
              z_error[pt]   =  cut_no->error_scalar[point] *
                                        cut_no->normal_vector[point].z;
            }

  both_done = 0;
  toolpath_spec[0] = offset_change (z_error, pt);
  toolpath_spec[1] = offset_change (x_error, pt);

                /*  These values will always be substituted into the
                    tool offset update command in the next segment of NC
                    code.  They will often be zero, and will not matter.
                    When either is (or both are) zero, further
                    corrections will be calculated for the
                    corresponding axis (or both axes) by the untrue
                    planes function.                                    */

  if ( toolpath_spec[0] != 0  &&  toolpath_spec[1] != 0 ) both_done = 1;

  return both_done;
}
  /* end of constant_error */
```

C.3.18 straight_line

This function fits a straight line to data points and fills in poly_coeff numbers 5, 6, 0.

Function Prototype Declaration:

void straight_line (double nose_rad)

Inputs:

nose_rad - radius of the probe tip nose

Outputs:

none

Function Procedures:

The coefficients in poly_coeff numbered 1, 2, 3, 4 are first set to zero. The function then prepares for the case of a cut with no points. (It may or may not be a hypothetical cut.) Data are interpreted in an ususual way in this case. Furthermore, since these cuts will be used in solving for compensation curve intersections just as the other cuts will, they need to be offset to the location of the tool nose center. Next, the function addresses the case in which only a single probe point is provided. According to the rules pertaining to the specific part, a line parallel to a coordinate axis will pass through the point. Even though a hypothetical cut actually has no points, it will also be treated in this way since the function assigns it a point. The function then finally addresses the general case with the least-squares method of fitting a line to points. The coefficients are then loaded into the data structure variables.

```
/*****************************************************************************
  Function to fit a straight line to data points. This function fills in
    poly_coeff numbers 5, 6, 0.
*****************************************************************************/

void  straight_line (double nose_rad)

/*****************************************************************************
*
* Global variables:
*       attrib  -  cut attributes structure
*
* Parameter input:
*       none
*
* Parameter output:
*       none
*
* Non-system function calls:
*       none
*
*****************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.  The pointer cut_no has the
                        value last assigned by the calling function.        */

    int point;
    double  final_hyp_cut, switched_coeff_5, switched_coeff_0;
    double  sum_x, sum_z, sum_xz, sum_x_sq, sum_z_sq, x_bar, z_bar;


                    /*  Certain coefficients are zero for a straight line.  */

    cut_no->poly_coeff[1] = cut_no->poly_coeff[2] =
                         cut_no->poly_coeff[3] = cut_no->poly_coeff[4] = 0;

                    /*  First prepare for the case of a cut with no points.
                        (It may or may not be a hypothetical cut.)  Data
                        are interpreted in an ususual way in this case.
                        Also, since these cuts will be used in solving for
                        compensation curve intersections just as the other
                        cuts will, they need to be offset to the location
                        of the tool nose center (now) just as the other
                        cuts will (later).                                  */

    if ( (cut_no - attrib[1])+1 == total_cuts[1] )
        final_hyp_cut = 0;
    else                        //  These values are used in the next section.
        final_hyp_cut = 1;
```

```
if ( cut_no->how_many_points == 0 )
  {
    if ( cut_no->cut_direction.x != '0' )
        cut_no->adjusted_coord[0].z = cut_no->other_planes[1].angle +
                                      nose_rad;  /*  Must assume +k normal.  */
    else
        cut_no->adjusted_coord[0].x = cut_no->other_planes[1].angle +
                 final_hyp_cut * nose_rad;  /*  Must assume +i normal.
                        The tool nose center may actually coincide with
                        the surface of a hypothetical cut, while it
                        must be offset by nose_rad from other surfaces. */
  }

                /*  Now address the case in which only a single point
                    is provided.  According to rules pertaining to the
                    specific part, a line parallel to a coordinate axis
                    will pass through the point.  Even though a
                    hypothetical cut actually has no points, it will
                    also be treated in this way since the above section
                    has assigned it a point.                              */

if ( cut_no->how_many_points == 0  ||  cut_no->how_many_points == 1 )
  {
    if ( cut_no->cut_direction.x != '0' )
      {
        cut_no->poly_coeff[5] = 1;
        cut_no->poly_coeff[6] = 0;
        cut_no->poly_coeff[0] = cut_no->adjusted_coord[0].z;
      }
    else
      {
        cut_no->poly_coeff[5] = 0;
        cut_no->poly_coeff[6] = 1;
        cut_no->poly_coeff[0] = cut_no->adjusted_coord[0].x;
      }
    return;
  }

                /*  Now to address the general case with the least-
                    squares method of fitting a line to points.         */

sum_x = sum_z = sum_xz = sum_x_sq = sum_z_sq = x_bar = z_bar = 0;

for ( point = 0;  point < cut_no->how_many_points;  ++point )
  {
    sum_x += cut_no->adjusted_coord[point].x;
    sum_z += cut_no->adjusted_coord[point].z;
    sum_xz += cut_no->adjusted_coord[point].z *
              cut_no->adjusted_coord[point].x;
    sum_z_sq += cut_no->adjusted_coord[point].z *
                cut_no->adjusted_coord[point].z;
    sum_x_sq += cut_no->adjusted_coord[point].x *
        cut_no->adjusted_coord[point].x;
```

```
    }

  x_bar = sum_x / cut_no->how_many_points;
  z_bar = sum_z / cut_no->how_many_points;

                  /*  Load coefficients into data structure variables.

                      When the slope of the fitted line will be steep, a
                      singularity will arise in the derivation of the
                      standard equation for x(z).  This problem can be
                      averted by deriving the equation for z(x) instead.
                      Then, by dividing through by the coefficient of x,
                      the equation will be in the desired form of x(z).
                      After this is done in the "switched" case below,
                      the values of the coefficients are assigned to the
                      variable names used in the standard equations.  The
                      slope is forced to be perfectly vertical, which is
                      virtually identical to that for cases within the
                      specified tolerance.                                */

  if ( fabs(sum_z_sq - z_bar*sum_z) > tolerance1 ) // Standard case
    {                                                 // (acceptable denominator)
      cut_no->poly_coeff[5] = ( sum_xz - x_bar * sum_z ) /
                                                  ( sum_z_sq - z_bar * sum_z );
      cut_no->poly_coeff[0] = -( x_bar - cut_no->poly_coeff[5] * z_bar );
      cut_no->poly_coeff[6] = -1;
    }
  else                                              // Switched case
    {
      switched_coeff_5 = ( sum_xz - z_bar * sum_x ) /
                                                  ( sum_x_sq - x_bar * sum_x );
      switched_coeff_0 = -( z_bar - switched_coeff_5 * x_bar );
      cut_no->poly_coeff[5] = -1;
      cut_no->poly_coeff[0] = switched_coeff_0;
      cut_no->poly_coeff[6] = 0;
    }

  return;
}
  /* end of straight_line */
```

C.3.19 calculate_coefficients

This function calculates the coefficients of the correction curve polynomial. It computes the compensation adjusted coordinates at each of the gauged points and calls the straight line function to fit a line through the adjusted points.

Function Prototype Declaration:

void calculate_coefficients (int constant_offset, int tool)

Inputs:

constant_offset - An integer flag: If != 0 the normal errors are
compensated by negative offsets otherwise by
negative scalar_error multiples of normal
vectors.

tool - tool number

Outputs:

None.

Function Procedures:

The function first reads the tool nose radius from the file CUTTER.K where K is the tool number entered through the parameter list. Next it computes the adjusted coordinates by adding to the nominal gauging coordinates of the finishing session the product of (1) the negative of the error determined in the current session, (2) the percentage correction to be applied, and (3) the unit vector normal to the surface. The result will be target coordinates for the finish machining session. These coordinate adjustments are for gauge points. Each type of curve is considered separately to determine the coordinates.

```
/*************************************************************************
  Function to calculate the coefficients of the correction curve polynomial.
  It computes the compensation adjusted coordinates at each of the gauged
  points and calls the straight line function to fit a line through the
  adjusted points
*************************************************************************/

void  calculate_coefficients (int constant_offset, int tool)

/*************************************************************************
*
* Global variables:
*       attrib  -  cut attributes structure
*       toolpath_spec - array of toolpath endpoints. First two are offsets.
*
* Parameter input:
*       constant_offset - An integer flag: If != 0 the normal errors are
*                         compensated by negative offsets otherwise by
*                         negative scalar_error multiples of normal vectors.
*
* Parameter output:
*       none
*
* Non-system function calls:
*       straight_line
*
*************************************************************************/

  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.                          */

    int  pointer_index, point, j;
    char  buffer[80], comment[80];
    FILE  *in_file;
    double  compensation, compensation_x, compensation_z,
                                              dividend, divisor, nose_rad;
    struct cut_attributes  *next_session_cut;

                    /*  Read the tool nose radius.                       */

    sprintf(buffer, "\\QIA\\DATABASE\\TOOLS\\CUTTER.%d", tool);
    if ( (in_file = fopen(buffer, "r")) == NULL )
      {
        sprintf(buffer, "Tool data not available in file "
                             "\\QIA\\DATABASE\\TOOLS\\CUTTER.%d", tool);
        prnwin(buffer, LIGHTRED, 1);
        delay(5000);
        return;
      }

    for ( j = 0;  j < 30;  ++j )
```

```
  {
    fgets(buffer, 80, in_file);
    sscanf(buffer, "%s", comment);
    if ( strcmp(comment, "RADIUS:") == 0 )
      {
        sscanf(buffer, "%*s %lf/%lf", &dividend, &divisor);
        nose_rad = dividend / divisor;
        break;
      }
  }
fclose(in_file);

                /*  Compute the adjusted coordinates: add to the
                    nominal gauging coordinates of the next session the
                    product of (1) the negative of the error determined
                    in the current session, (2) the percentage
                    correction to be applied, and (3) the unit vector
                    normal to the surface.  The result will be target
                    coordinates for the next machining session. These
                    coordinate adjustments are for gauge points.

                    Add compensation to each gauge point.              */

for ( cut_no = attrib[1];
                      (cut_no - attrib[1]) < total_cuts[1];  ++cut_no )
  {
    for ( point = 0;  point < cut_no->how_many_points;  ++point )
      {
        if ( constant_offset == 0 )
          {
                /*  The compensation curve is being adjusted to be the
                    path of the tool nose center instead of the path
                    along the surface.                                 */

            compensation = -(cut_no->error_scalar[point]) *
                                                  correction + nose_rad;
            compensation_x = compensation;
            compensation_z = compensation;
          }
        else
          {
            compensation_x = -toolpath_spec[1];
            compensation_z = -toolpath_spec[0];
          }

        cut_no->adjusted_coord[point].x =
            cut_no->nominal_coord[point].x +
                    ( compensation_x * cut_no->normal_vector[point].x );
        cut_no->adjusted_coord[point].z =
            cut_no->nominal_coord[point].z +
                    ( compensation_z * cut_no->normal_vector[point].z );
      }
```

```
                    /*  Each type of curve must be considered separately to
                        determine the coordinates.                          */

        if ( cut_no->type_of_curve[0] < 0 )
            straight_line (nose_rad);
        else
          {
            prnwin ("\nOnly first degree curves are implemented.",
            LIGHTRED, 1);
            delay(5000);
            exit (3);
          }

      }
    return;
}
  /* end of calculate_coefficients */
```

C.3.20 solve_equations

This function solves simultaneous linear algebraic equations by Gaussian Elimination.

Function Prototype Declaration:

void solve_equations (int how_many, double coeff[][7], double constant[], double root[])

Inputs:

how_many - number of equations to be solved.

coeff - matrix of coefficients of the equations.

constant - array of right hand sides of the equations.

Outputs:

root - array of solutions for the linear equations.

Function Procedures:

The function first tests whether any diagonals are equal to zero. It then switches equations to avoid a zero pivot element. It continues in this fashion by Gauss elimination to triangularize the coefficient matrix. Finally it solves for the roots by back substitution.

```
/*************************************************************************
  Function to solve simultaneous linear algebraic equations by Gaussian
    Elimination.
*************************************************************************/

void  solve_equations
        (int how_many, double coeff[][7], double constant[], double root[])

/*************************************************************************
*
* Global variables:
*       none
*
* Parameter inputs:
*       how_many  -  number of equations to be solved.
*
*       coeff     -  matrix of coefficients of the equations
*
*       constant  -  array of right hand sides of the equations
*
*
* Parametric outputs:
*       root      -  array of solutions for the linear equations
*
* Non-system function calls:
*       none
*
*************************************************************************/

                    /*  The form of equation which this function solves is
                        as follows:

 coeff[0][0]*root[0] +...+ coeff[0][how_many-1]*root[how_many-1] = constant[0]
 coeff[1][0]*root[0] +...+ coeff[1][how_many-1]*root[how_many-1] = constant[1]
 etc.
                        The parameter "tolerance1" should be defined
                        elsewhere as a small positive value (.00004, for
                        example).  It allows numbers very close to zero to
                        be treated as if they are exactly zero.

                        In the calling function, each dimension of all
                        array arguments should be "6", regardless of how
                        many equations are to be solved (up to a maximum of
                        six).

                        As an argument for this function, the solution
                       "root" is available to the calling function as a
                        (double precision) floating point array.          */
  {

    int  i, j, k, l, m, n;
    double  temp_storage, frozen_coeff, sum;
```

```
                /*  Test whether any diagonals are equal to zero.       */

  for ( k = 0;  k < (how_many - 1);  ++k )
    {
      if ( fabs(coeff[k][k]) < tolerance1 )
        {
          for ( m = k + 2;  fabs(coeff[m-1][k]) < tolerance1;  ++m )
              if ( (m - (how_many - 1)) > tolerance1 ) /* All diagonals 0 */
                {
                  prnwin ("*** Equations not solvable ***", LIGHTRED, 1);
                  delay(5000);
                  return;
                }

                  /*  Switch equation with next to avoid diagonal zero.   */
          --m;
          temp_storage = constant[m];
          constant[m] = constant[k];
          constant[k] = temp_storage;
          for ( l = 0;  l < how_many;  ++l )
            {
              temp_storage = coeff[m][l];
              coeff[m][l] = coeff[k][l];
              coeff[k][l] = temp_storage;
            }
        }
                  /*  Triangularize coefficient matrix.                   */

      for ( i = k + 1;  i < how_many;  ++i )
        {
          constant[i] += -constant[k] * coeff[i][k] / coeff[k][k];
          frozen_coeff = coeff[i][k];   /*  Will otherwise change below.  */
          for ( j = k;  j < how_many;  ++j )
              coeff[i][j] += -coeff[k][j] * frozen_coeff / coeff[k][k];
        }
    }
                  /*  Solve for roots by back substitution.               */

  for ( n = 0;  n < how_many;  ++n )
    {
      l = how_many - n - 1;
      sum = 0;
      if ( (l - (how_many - 1)) < -tolerance1 )
          for ( j = (l + 1);  j < how_many;  ++j )
              sum += coeff[l][j] * root[j];
      root[l] = (constant[l] - sum) / coeff[l][l];
    }

  return;
}
  /* end of solve_equations */
```

C.3.21 untrue_planar

Function to calculate corrections for untrue planar relationships.

Function Prototype Declaration:

void untrue_planar (int constant_offset, int tool)

Inputs:

constant_offset - An integer flag: If it is not equal to 0 the normal errors are compensated by negative offsets, otherwise by negative scalr_error multiples of normal vectors.

tool - tool number.

Outputs:

none.

Function Procedures:

This function calculates coordinate values to be inserted into part program instructions to correct all cases of errors of straightness, parallelism and perpendicularity -- i.e., all errors except for those on nonlinear curve-interpolation cuts, and those which can be corrected with tool offsets. It first calls calculate_coefficients to determine the equations of the tool paths for the next machining session for each cut. Then it considers each cut one at a time. At this point the coefficients of the equations for each cut have already been determined. The last cut is hypothetical and therefore not included in the total cuts. Next it finds the coordinates of the point common to both, the current cut and the "intersecting cut" next in the direction of cutting. If a tool offset increment was calculated for an axis (stored in either toolpath_spec[0] or [1]), then it supercedes the need for the new specification calculated here for the same axis, and so the nominal value is retained.

```
/*****************************************************************************
  Function to calculate corrections for untrue planar relationships.
*****************************************************************************/

void  untrue_planar (int constant_offset, int tool)

/*****************************************************************************
*
* Global variables:
*       attrib  -  cut attributes structure
*
*       toolpath_spec  - array of go-to points for generating NC segment
*
* Parameter inputs:
*       constant_offset - An integer flag: If !0 the normal errors are
*                         compensated by negative offsets otherwise by
*                         negative scalr_error multiples of normal
*                         vectors.
*
* Parameter outputs:
*       none
*
* Non-system function calls:
*       calculate_coefficients, solve_equations
*
*****************************************************************************/

  {
                    /*  This function calculates coordinate values to be
                        inserted into part program instructions to correct
                        all cases of errors of straightness, parallelism
                        and perpendicularity -- i.e., all errors except for
                        those on nonlinear curve-interpolation cuts, and
                        those which can be corrected with tool offsets.

                        Descriptions of global variables used are at the
                        beginning of this file.                          */

    int  i, j;
    double  x_coord, z_coord, coeff[7][7], constant[7], root[7];

                    /*  Determine the equations of the tool paths for the
                        next machining session for each cut.             */

    calculate_coefficients (constant_offset, tool);

                    /*  Now consider each cut one at a time.  The
                        coefficients of the equations for each cut have
                        already been determined.The last cut is
                        hypothetical and therefore not included in the
                        total cuts.                                      */
```

```
  for ( cut_no = attrib[1];
                    (cut_no - attrib[1]) < total_cuts[1]-1 ;  ++cut_no )
    {
                /*  Find the coordinates of the point common to both,
                    the current cut and the "intersecting cut" next
                    in the direction of cutting.(See definitions at the
                    beginning of the file.)                              */

      coeff[0][0] = cut_no->poly_coeff[5];
      coeff[0][1] = cut_no->poly_coeff[6];
      constant[0] = cut_no->poly_coeff[0];
      coeff[1][0] =
                ( attrib[1] + cut_no->intersect_cut - 1 )->poly_coeff[5];
      coeff[1][1] =
                ( attrib[1] + cut_no->intersect_cut - 1 )->poly_coeff[6];
      constant[1] =
                ( attrib[1] + cut_no->intersect_cut - 1 )->poly_coeff[0];
      solve_equations (2, coeff, constant, root);

                /*  If a tool offset increment was calculated for an
                    axis (stored in either toolpath_spec[0] or [1]),
                    then it supercedes the need for the new
                    specification calculated here for the same axis,
                    and so the nominal value will be retained.          */

      for ( i = 0;  i < 2;  ++i )
        {
          switch ( cut_no->endpoint_spec[i].coord )
            {
              case 'Z':
              case 'z':
                  toolpath_spec[(cut_no->endpoint_spec[i].spec) - 1] =
                                                          root[0];
                  break;
              case 'X':
              case 'x':
                  toolpath_spec[(cut_no->endpoint_spec[i].spec) - 1] =
                                                      2 * root[1];
                  break;
            }
        }

    }
  return;
}
  /* end of untrue_planar */
```

C.3.22 write_segment

This function writes the NC part program segment with modified tool path specifications.

Function Prototype Declaration:

void write_segment (int segment_no)

Inputs:

segment_no - NC code segment number

Outputs:

None.

Function Procedures:

The function reads all characters from the file PTNN0021.IN, one at a time and writes them out to the new segment file called PTNN0021.DNC. When it reaches a "$" sign the next character read is the path specification number. This is used as an index to extract from the the array of modified toolpath specification points the adjusted value to be entered into the segment file in place of the $path_spec_no combination. The function continues on to the end of the input file.

```
/*************************************************************************
  Function to write NC part program segment with modified tool path
    specifications.
*************************************************************************/

void  write_segment (int segment_no)

/*************************************************************************
*
* Global variables:
*       part_no  -  part number
*
*       toolpath_spec - go-to points for NC code segment
*
* Parameter inputs:
*       segment_no  -  NC code segment number
*
* Parameter outputs:
*       none
*
* Non-system function calls:
*       none
*
*************************************************************************/

  {
    int  c, path_spec_no;
    char  dummy[81];
    FILE  *in_file, *out_file;
    static char  segment_template[] = "\\QIA\\CUTTING\\PTNN0021.IN";
    static char  new_segment[] = "\\METALIST\\PTNN0021.DNC";

    segment_template[15] = new_segment[12] = part_no[0];
    segment_template[16] = new_segment[13] = part_no[1];

    if ( (in_file = fopen (segment_template, "r")) == NULL )
      {
        sprintf (dummy, "No template program exists "
                        "for Part %s Segment %i.\n", part_no, segment_no);
        prnwin(dummy, LIGHTRED, 1);
        delay(5000);
        exit (2);
      }

    out_file = fopen (new_segment, "w");

    while ( ( c = fgetc (in_file) ) != EOF )
      {
        if ( c != '$' )
            fputc (c, out_file);
        else
          {
```

```
            fscanf (in_file, "%d", &path_spec_no);
            fprintf (out_file, "%-.5f", toolpath_spec[path_spec_no-1]);
          }
      }

    fprintf (out_file, "%c", '\x1A');   /*  Adding the ^Z marking EOF.
*/
    fclose (out_file);
    fclose (in_file);

    return;
  }
    /* end of write_segment */
```

C.3.23 errspec

This function writes the file errspec.dat of RTEC cut specification start and end points for each of the cuts.

Function Prototype Declaration:

```
void errspec (struct toolpath_data cut_coords[],
                    int init_block_spec[],int initial_block_cut[],
                     int final_block_cut[], double MACHINE_HOME_X)
```

Inputs:

cut_coords - coordinates of start- and end-points of cuts

init_block_spec - tool path spec numbers in terms of cut numbers

final_block_cut - cut for which a tool path spec is final

MACHINE_HOME_X - x-distance from machine home to part origin

Outputs:

None.

Function Procedures:

The function first opens the error specification file. It writes the header record into the file. Then it writes the axis offsets to the file even if they are both equal to zero. A nonzero offset indicates that all the errors are reasonably uniform for that axis, and may be compensated by an offset which is constant for that axis. Next, the function proceeds to compensate individual cuts. The subscripts of the variable "toolpath_spec" are in the order in which the cuts are machined. This is the same order required for the records in the error specification file. Note that each pair of tool path specs, beginning with the even-numbered subscript, refer to x and z-coordinates, respectively. The first pair of values in the toolpath spec file, being the tool offset increments, were written to the file as stated above. The first cut specified in the attributes file is never actually machined, regardless of whether it is hypothetical. (Note also that the order in which cuts are listed in the attributes file is not necessarily the order in which they are machined.) Therefore, the only useful information in the first data record in the error spec file is the x-threshold for determining when to start monitoring an axis for the cut specified next. It is assumed here that that cut will follow the one which initially moves the tool into the part, and that it is the cut whose initial block contains the toolpath spec with subscript 2. The first cut actually machined is always in the x-direction (into the part), and may or may not have gauging

points. In any case, the pair of tool path specs beginning with the subscript 4 are the coordinates of the end point of the first cut containing gauging points. The current cut is that for which the current spec is an end point--that is, the cut whose final block of NC code contains the current tool path spec. If the angle of a taper is steeper than 45 degrees, the x-axis will be monitored. Otherwise, the z-axis will be monitored. The compensation values are equal to the adjusted coordinates minus the nominal coordinates. The errors are the negative of the compensation values. (If there are no points on the cut, specify zero errors.) The first variable on the right-hand side of the first equation reads: the tool path spec corresponding to the initial block of NC code for the current cut. That is, the adjusted coordinates of the start point for the current cut. Note that the subscript of the tool path spec for a z-coordinate if one greater than for the x. The function then writes data to error spec file and terminates the file.

```
/*************************************************************************
  Function to write the file errspec.dat of RTEC cut specification start
    and end points for each of the cuts.
*************************************************************************/

void errspec (struct toolpath_data cut_coords[], int init_block_spec[],
   int initial_block_cut[], int final_block_cut[], double MACHINE_HOME_X)

/*************************************************************************
*
* Global variables:
*       VERSION - program version number, See DEFINE's
*
*       toolpath_spec  - Adjusted NC go-to points
*
*       toolpath_spec_nom - Nominal NC go-to points
*
* Parameter inputs:
*       cut_coords - coordinates of start- and end-points of cuts
*
*       init_block_spec - tool path spec numbers in terms of cut numbers
*
*       final_block_cut - cut for which a tool path spec is final
*
*
* Parameter outputs:
*       none
*
* Non-system function calls:
*       none
*
*************************************************************************/
  {
    int  j, current_cut, spec, axis;
    FILE  *out_file;
    double  delta_x, delta_z, start_error[2], end_error[2],
        PIECS_version_no = VERSION;
    static char  header[] = "(version); axis, "
                     "start_coord, x_err, z_err, end_coord, x_err, z_err";
    static char  err_file[] = "\\QIA\\CUTTING\\ERRSPEC.DAT";
    static char  axis_code[3] = "XZ";

                    /*  Open the error spec file and write the header
                        record.                                        */

    out_file = fopen(err_file, "w");
    fprintf(out_file, "%-.2f %s\n", PIECS_version_no, header);

                    /*  Write the axis offsets to the file.  If either is
                        nonzero, all errors are reasonably uniform for the
                        corresponding axis, and may be compensated by
                        offsets which are constant for the axis.       */
```

```
fprintf(out_file, "%-.5f %-.5f\n", toolpath_spec[0], toolpath_spec[1]);


                /*  The subscripts of the variable "toolpath_spec" are
                    in the order in which the cuts are machined.  This
                    is the same order required for the records in the
                    error spec file.  Note that each pair of tool path
                    specs, beginning with the even-numbered subscript,
                    refer to X and Z coordinates, respectively.  The
                    first pair of values in the toolpath spec file,
                    being the tool offset increments, do not apply to
                    the error spec file.

                    The first cut specified in the attributes file is
                    never actually machined, regardless of whether it
                    is hypothetical.  (Note also that the order in
                    which cuts are listed in the attributes file is not
                    necessarily the order in which they are machined.)
                    Therefore, the only useful information in the first
                    data record in the error spec file is the X
                    threshold for determining when to start monitoring
                    an axis for the cut specified next.  It is assumed
                    here that that cut will follow the one which
                    initially moves the tool into the part, and that it
                    is the cut whose initial block contains the
                    toolpath spec with subscript 2.                    */

fprintf(out_file,
    "X  %9.5f    0.00000    0.00000  %9.5f    0.00000    0.00000\n",
       MACHINE_HOME_X, cut_coords[initial_block_cut[2]].startpoint[0]);

                /*  The first cut actually machined is always in the -X
                    direction (into the part), and may or may not have
                    gauging points.  In any case, the pair of tool path
                    specs beginning with the subscript 4 are the
                    coordinates of the end point of the first cut
                    containing gauging points.                         */

for ( spec = 4;  spec < 2*(total_cuts[1]-1);  spec += 2 )
  {
                /*  The current cut is that for which the current spec
                    is an end point--that is, the cut whose final block
                    of NC code contains the current tool path spec.     */

    current_cut = final_block_cut[spec];

                /*  Determine which axis to monitor.  If the angle of a
                    taper is steeper than 45 degrees, the X axis will
                    be monitored.  Otherwise, the Z axis will be
                    monitored.                                         */

    delta_x = fabs( cut_coords[current_cut].endpoint[0] -
                                cut_coords[current_cut].startpoint[0] );
```

```
    delta_z = fabs( cut_coords[current_cut].endpoint[1] -
                              cut_coords[current_cut].startpoint[1] );

    if ( delta_x > delta_z )
        axis = 0;                          /*  0 subscript for X axis.  */
    else
        axis = 1;                          /*  1 subscript for Z axis.  */

                /*  Calculate the errors at each point.  The
                    compensation values are equal to the adjusted
                    coordinates minus the nominal coordinates.  The
                    errors are the negative of the compensation values.
                    (If there are no points on the cut, specify zero
                    errors.)  The first variable on the right-hand side
                    of the first equation reads: the tool path spec
                    corresponding to the initial block of NC code for
                    the current cut.  That is, the adjusted coordinates
                    of the start point for the current cut.  Note that
                    the subscript of the tool path spec for a Z
                    coordinate if one greater than for the X.          */

    if ( attrib[1][current_cut-1].how_many_points != 0 )
      {
        start_error[0] =
            -( toolpath_spec[init_block_spec[current_cut]]/2. -
                              cut_coords[current_cut].startpoint[0] );
        start_error[1] =
            -( toolpath_spec[init_block_spec[current_cut]+1] -
                              cut_coords[current_cut].startpoint[1] );
        end_error[0] = -( toolpath_spec[spec]/2. -
                                cut_coords[current_cut].endpoint[0] );
        end_error[1] = -( toolpath_spec[spec+1] -
                                cut_coords[current_cut].endpoint[1] );
      }
    else
        for ( j = 0;  j < 2;  ++j )
          {
            start_error[j] = 0;
            end_error[j] = 0;
          }

                /*  Write data to error spec file.                     */

    fprintf(out_file, "%c  %9.5f %10.5f %10.5f %10.5f %10.5f %10.5f\n",
        axis_code[axis],
        cut_coords[current_cut].startpoint[axis],
        start_error[0], start_error[1],
        cut_coords[current_cut].endpoint[axis],
        end_error[0], end_error[1]);
}

                /*  Terminate the file.                                */
```

```
  fprintf(out_file, "%c", '\x1A');       /*  Adding the ^Z marking EOF.   */
  fclose(out_file);

  return;
}
  /* end of errspec */
```

C.3.1 piecs

This is the main function of the PIECS software module and is the last function in the program listing. This placement is done only for compiler requirements. It takes nominal part specifications from CAD generated specification files along with probed part coordinates at selected points of a semifinished part and computes the necessary cut adjustments to make in the finish cut in order to bring the final finished workpiece in conformity with the nominal specifications. The function is called from the QIA Control program.

Function Prototype Declaration

float piecs(int seq_no, double temp[], double MACHINE_HOME_X)

Inputs:

seq_no - This is a sequence number used to index the gauging data by the QIA database. It is converted to a character string and appended as the "K" to the file GAUGING.K

temp[] - This is an array of 40 thermocouple readings, in celsius, passed to the PIECS function for use by the geometric-thermal error model. They would be the thermocouple readings taken by the Quality Controller at the time of the gauging operation. These would have been stored in a separate database.

MACHINE_HOME_X - This is the distance from the machine home to the part origin in inches.

Outputs:

The program prepares three files report.txt, ptNN0021.dnc and errspec.dat.

Function Procedures:

Using the sequence number the program first reads the current part number from the gauging file. It then forms a loop so that, in case a point to be gauged is skipped, cut attributes can be re-interpreted to match only the points which did register. It next reads in the attributes of each cut to be affected by the gauging session for the current part. The data include nominal coordinates for gauging points, and nominal "go-to" coordinates for post-gauging cuts. Two cut attribute files are read, one for the semifinish cut and the second for the finish cut. The semi-finish probing session RTEC data is then read and the program transforms gauging results to part coordinates from the probing segment format of the semi-finish session. It then calculates deviations of actual from nominal gauging coordinates by computing the scalar errors in the direction of the normal vectors at the gauge points of the semifinish cut. The program finally classifies the errors as to whether they are constant error offsets, untrue curve errors or untrue planar errors, applies the appropriate correction functions and prepares the output files.

```
/*****************************************************************************
  Piecs function call
*****************************************************************************/
float  piecs (int seq_no, double temp[], double MACHINE_HOME_X)
/*****************************************************************************/
  {
                    /*  Descriptions of global variables used are at the
                        beginning of this file.

                       "Skipped" will equal the point numbers of those
                        missed by the probe during gauging.                 */

    int  i, all_done, next_segment, skipped[3], tool,
      init_block_spec[max_cuts], initial_block_cut[max_cuts],
      final_block_cut[max_cuts];
    char  point_order, seq[7];
    float  PIECS_version_no = VERSION;

                    /*  Declare a structure to contain cut coordinates in
                        terms of the cut number.  This will clarify the
                        meanings of variables.  Create storage addresses
                        and point to them.                                  */

    struct toolpath_data  cut_coords[max_cuts];
    double  dummy_startpoint_coords[max_cuts][2],
              dummy_endpoint_coords[max_cuts][2];

    for ( i = 0;  i < max_cuts; ++i )
      {
        cut_coords[i].startpoint = dummy_startpoint_coords[i];
        cut_coords[i].endpoint = dummy_endpoint_coords[i];
      }

                    /*  Convert sequence number to a string.              */

    itoa(seq_no, seq, 10);

                    /*  Read in the attributes of each cut to be affected
                        by each gauging session for the current part.
                        There is a separate file for each post-gauging
                        machining session.  Data include nominal
                        coordinates for gauging points, and nominal "go-to"
                        coordinates for post-gauging cuts.                  */

    skipped[0] = skipped[1] = skipped[2] = -1; /* Set skip indicators to F. */

                    /*  Read the part number from the gauging file.       */

    get_part_no (seq);

                    /*  Form a loop so that, in case a point to be gauged
                        is skipped, cut attributes can be re-interpreted to
                        match only the points which did register.           */
```

```
i = 0;
do
  {
                /*  Two cut attributes files will be read in. The first
                    for the semifinished cut and the second for the
                    finished cut. The files are to be titles attrib1.n,
                    for the semifinished cut, and attrib2.n for the
                    finished cut, where n refers to a part number.      */

    read_cut_attributes (skipped);

                /*  Semi-finished probing session RTEC data will now be
                    entered. The next function reads, adjusts and
                    transforms gauging results from the probing segment
                    of the semi-finished session.                       */

    point_order = read_rtec (seq, temp, MACHINE_HOME_X, &tool);

                /*  There are two sessions. The first session (1) is
                    the semi-finished session that includes machining
                    and then gauging. The second session (2) is the
                    finished cut session that includes machining and
                    gauging. Piecs is executed between the semifinished
                    probe, called segment 2, and the finished machining
                    segment, called segment 3. Thus the current segment
                    should be segment 2.                                */

    segment_no = 2;

                /*  Calculate deviations of actual from nominal gauging
                    coordinates. During the first pass no skipped
                    points are assumed.  After all skipped points, if
                    any, have been accounted for, the next function
                    writes the computed scalar errors in the direction
                    of the normal vectors at the gauge points of the
                    semifinished cut to the finished cut attributes.    */

    calculate_errors (skipped);

                /*  The following statement will test whether a point
                    was skipped.  If not, execution of the subsequent
                    statements will proceed.  If so, the outermost loop
                    will start over with reading of the cut attributes,
                    this time matching the arrays in accord with the
                    missed point.                                       */

    if ( skipped[i] != -1 ) continue;

                /*  Write a file displaying errors for gauging session
                    1.                                                  */

    write_report (point_order, seq_no);
```

```
                /*  This ends processing of semifinished data.

                    Classify errors and apply the appropriate
                    correction functions.  First read in nominal tool
                    path specifications for the NEXT segment.  The next
                    segment is the machining segment of the finishing
                    session, which would be segment 3.                  */

    next_segment = 3;

                /*  Read any constant offsets and the nominal endpoints
                    for the finished cut.                               */

    read_toolpath_data (next_segment, cut_coords, init_block_spec,
                                  initial_block_cut, final_block_cut);

                /*  The following switch statement is structured to
                    make it possible for all cases except for any
                    before for the first one matched to be executed in
                    succession.                                         */

    switch ( classify_errors () )
      {
        case 1:
            untrue_curve ();

        case 2:
            all_done = constant_error ();

        case 3:
            untrue_planar (all_done, tool);
            break;
      }

                /*  Write the NC file with adjusted "go-to" points.     */

    write_segment (next_segment);

                /*  Write the RTEC file of adjusted cut endpoints       */

    errspec (cut_coords, init_block_spec, initial_block_cut,
                                    final_block_cut, MACHINE_HOME_X);
  }
while ( skipped[i] != -1  &&  i++ < 3 );

return PIECS_version_no;
}
/* end of piecs */
```

APPENDIX D

Introduction to NCMAKE

Since an NC probing code is required for each inspection session of a part processed through the QIA system, a software procedure has been developed to facilitate the generation of the required probing code. A CAD representation of the part is first used to develop a file of probe-path coordinates. Then a program called NCMAKE, to be described here, reads that file, prompts the user for certain NC program parameters, and writes the file of NC probing code. NCMAKE is written in the C programming language.

The process-intermittent loop is a QIA procedure for refining tool-path coordinates. A workpiece first undergoes rough and semifinish machining, and then is inspected with the probe. The tool-path coordinates for the finish machining are adjusted by PIECS to compensate for the dimensional error patterns detected. After the finish machining, the part is inspected again. The alternating machining and probing sessions are accomodated by segmenting the NC program. NCMAKE produces the probing segments such that they can be executed as independent NC programs.

Considering various requirements of process-intermittent dimensional inspection, it is clear that most segments of NC probing code should have certain types of notes and comments in common. NCMAKE provides for certain "standard" kinds of comments which will be included in every file of NC code produced. For example, a comment states whether the touch-trigger probe has an extension.

The routine nature of probe moves during inspection allowed a simple design for the file of probe-move coordinates required by the program NCMAKE. The file is named "prbpathi.dat", where "i" is the session number. (See Figure D1.) The file is made up of repeated sequences of coordinates to approach, touch, and retract from each successive gauging point. An exception to this pattern is the case of a "bypass" point, a point in space to be traversed by the probe after retracting from one gauging point, before proceeding to the approach point for the next. A bypass point is sometimes necessary to provide sufficient clearance where the probe would otherwise collide with the part. Comments are included in each record in the file, indicating whether the values in that record are coordinates of the approach, touch, retract, or bypass point. The gauging point number is also given for each touch point.

A separate probing-path definition program, an interactive computer program using a CAD system, has been developed to produce prbpathi.dat. This program creates the probing path on a CAD drawing of the part, and the resulting coordinates are written to prbpathi.dat with the following modifications. First, the coordinates for each gauging point are adjusted such that the probe stylus will attempt to touch a point slightly beneath the part surface. That is necessary because, for the probe to trigger with accurate timing, the target coordinates when it moves to touch the part must take into account the machine tool acceleration distance and the overtravel limits of the probe stylus. Furthermore, all coordinates are adjusted to the center point of the probe stylus. Tool offsets for the probe are determined such that the stylus center is the point which must move according to NC code instructions. With prbpathi.dat thus created, NCMAKE may use the coordinates directly.

NCMAKE is designed to create code which moves the machine first to the user-defined probing start location, and then moves only along the Z axis to the Z coordinate of the approach point for Gauging Point 1. Next, it moves in only the X direction to the approach point. Moving one axis at a time in this order ensures that the preliminary moves of the probe allow maximum clearance from the workpiece. The probe will touch Point 1, retract, approach Point 2, etc., according to the coordinates in prbpathi.dat, as explained above.

The program uses a file named template.prb as a template for the NC code segment. (See Figure D2.) The file is copied character-by-character, except that variable symbols are replaced by corresponding numerical values or text strings. A variable symbol consists of a special character followed immediately by an integer. There are two special characters used: $ and @. If the special character is $, then the integer is a number referring to one of the twenty parameters discussed earlier. The only exception is $21, which is replaced by the complete set of probe-move blocks comprising the main body of the NC code segment. If the special character is @, then the text to be substituted does not originate from the parameter list, but from within the main program. This approach is necessary, for example, when the substitution will be derived from a calculation which is based on a parameter value.

The current version of NCMAKE has been tailored to the requirements of the prototype QIA turning center. However, the program has been structured so that a few simple changes, mostly to data files rather than program modules, could adapt it to significantly different NC programming applications.

```
PARTLENGTH: 6.25
POINT start       5.148643, 1.793852, 0.000000
POINT touch 1,    5.123869, 1.494877, 0.000000
POINT finish      5.148643, 1.793852, 0.000000
POINT start       5.000000, 1.800000, 0.000000
POINT touch 2,    5.000000, 1.500000, 0.000000
POINT finish      5.000000, 1.800000, 0.000000
POINT start       4.850000, 1.800000, 0.000000
POINT touch 3,    4.850000, 1.500000, 0.000000
POINT finish      4.850000, 1.800000, 0.000000
POINT bypass      4.854984, 2.092784, 0.000000
POINT start       4.550000, 2.100000, 0.000000
POINT touch 4,    4.550000, 1.800000, 0.000000
POINT finish      4.550000, 2.100000, 0.000000
POINT start       4.243750, 2.100000, 0.000000
POINT touch 5,    4.243750, 1.800000, 0.000000
POINT finish      4.243750, 2.100000, 0.000000
```

FIGURE D1
Example "pathfin.cdl" File of Probe-Path Coordinates

```
N0010  (ID,PROG,$1@1$2,$3 SEG$2: GAUGING @2)
N0020  ! @3
N0030  ! Tool $4:  Touch-trigger probe$5 in $6" boring bar holder.
N0040  !   Length: Z=$7  X=$8    Offset $20: Z=$9  X=$10
N0050  ! Touch face of part, then MDI W$11 for PROBING START
N0060  ! P$15 =$13 PROBING START: Machine Z coordinate to which
N0070  !   probe moves from the "standard" initial machine coordinates
N0080  !   [X0, Z10].
N0090  ! Probe tip diameter = $14"
N0100  ! Feedrate = $12 (inches per minute)
N0110  ! $16
N0120  ! $17
N0130  ! $18
N0140  G70                              ! INCH MODE
N0150  G94 T00                          ! INCHES/MIN, CLEAR TOOL
N0160  G00 Z(P$15)                      ! RAPID TO PROBING START
N0170  G92 X30.4 Z$19                   ! PRESET REF COORDS
N0180  T0$4@4                           ! INDEX TURRET TO PROBE
N0190  M14                              ! TURN ON PROBE
N0200  M15
$21
N@5  G00 X30.4 Z$19 T00                 ! BACK TO REF COORDS, CLEAR TOOL
N@6  G92 X0 Z(P$15)                     ! BACK TO PROBING START
N@7  G00 Z10                            ! BACK TO INITIAL MACHINE POSITION
N@8  M02                                ! END OF SEGMENT
N9999  (END, PROG)
```

_FIGURE D2
The Template File for the NC Code, "template.prb"

APPENDIX E
Procedure for Generating Part Data

This appendix describes the general steps that are followed in order to develop data for PIECS beginning with a CAD definition of a part. Any user of the PIECS software will have to tailor the steps in this appendix to their particular CAD system.

Step 1: Start a new CAD drawing. Design the part, creating the original CAD profile. The profile should consist of a single geometric entity (line or arc) for each cut. The CAD program should be structures to prompt the user for the part number, part name and CAD file name. It should also prompt for the date and time of CAD file creation. Write all of this to a part feature file.

Step 2: Create CAD profiles of the semi-finished and finished part. The profile of the semi-finished part should differ from that of the finished part by a constant offset.

Step 3: Identify and number all the cuts in sequence, assigning "1" to the lowest right-most cut in the part profile, and proceed in a counterclockwise direction. Although the basic (root-level) features, considered to be the parents of the cuts, have not yet been defined at this stage, the cuts must be recognized before the NC code for machining is created.

Step 4: Create the NC program segments for machining the semi-finished and finished parts by using NCMAKE. It produces the probing segments such that they can be executed as independent NC programs. The NC code identification number, required in the first block of NC code, is assigned as six digits: the first three digits of the part ID, followed by a "0", the session number, and a "2" for finish. The file name for a probing segment consists of the letters "PT" followed by the NC code identification number, and with the file extension "DNC". For example the semifinish probing segment for part 6324 would have the file name "PT632012.DNC".

Step 5: Determine which tool path coordinates in the NC code for machining the finished part are to be variables. Create PATHnn00.2, the file containing (1) the nominal coordinate values for the tool path variables and (2) the associated cut numbers. Also create the NC code input file for the process-intermittent error analysis.

Step 6: Interactively define all features (including cuts) except gauging points. The features for the finished part should be created before those for the semi-finished part. Have the name of each feature on the semi-finished part automatically become the same as that of the corresponding feature on the finished part. Since, in use, names will always be associated with gauging session numbers, the identical names will not cause confusion. Other than to conform to this naming convention, duplication of names should be automatically disallowed.

Some values for feature characteristics are predictable. Examples are that the gauging session number is always 2 for the finished part, and that Attribute 1 is always "INITIAL BLOCK" if the feature is a cut. Some values are calculable, such as feature count. Geometric data such as coordinates of endpoints of lines, and degree of curve in x and x, are to be derived from the CAD system's own database. Have values in these categories entered automatically. Write the the part

feature file (PARTFEAT).

Step 7: Use the probing path program to create (1) the gauging points, one cut at a time, for only the finished part profile, then (2) the same gauging points applied to the semi-finished part profile, and finally (3) the probing path and corresponding NC probing program segment for each of the two profiles. Append this to the part feature file created after step 1.

Step 8: Write ATTRnnv0.j, the cut attributes file which is used in the process-intermittent error analysis, where "j" is the gauging session number.

Step 9: After setting up for machining and gauging, write the various tool data files.

Step 10: Have a database system read in the part data, the part features and the tool data files.

Step 11: Produce the DMIS file for the current part to be used for CMM inspection.