INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Development of a CAM Package for Robotic Deburring & Polishing of Components with Intricate Geometry

Jin Li

A Thesis

in

The Department

of

Mechanical Engineering

Presented in partial fulfillment of the requirements

for the Degree of Master of Applied Science

at

Concordia University

Montreal, Quebec, Canada

September 1999

Abstract

Development of a CAM Package for Robotic Deburring & Polishing of Components with Intricate Geometry

Jin Li

This thesis deals with the development of a CAM package based on AutoCAD for refurbished and new components with intricate geometry. By using a deburring robot such as the Yamaha Zeta-1 Robot, deburring and polishing of a refurbished workpiece involves analyzing the geometry of the workpiece, determining and designing tool path, generating tool path data, animating the path traced by the robot tool and verifying the tool path on the Yamaha Zeta-1 Robot.

The most popular PC-based CAD package, AutoCAD, is chosen as the developing platform. A software package mainly programmed in AutoLISP, which is an underlying software application of AutoCAD, functions as a processor of the solid models of the workpieces and generates tool path data for the robot. The Yamaha Zeta-1 Robot, which is designed specifically for precision deburring, is chosen for executing the generated tool path. The development is coined by exploiting the powers of AutoLISP, C, C++ and DCL (Dialogue Control Language). This CAM package functions as an interface between the workpiece and the user, and a postprocessor for the tool location for the robot.

A kinematic model of the Yamaha Zeta-1 has been established and a commercial virtual reality application software package, WorldToolKit, is employed in animating the real robot based on the Yamaha Zeta-1's geometric parameters and kinematics. This

virtual robot serves as a virtual reality simulator to test and verify the machining path generated.

Experimental verification of the post-processed data generated by the developed software is presented for both deburring and polishing features. The results show the successful implementation of this CAM software package.

Acknowledgements

I would like to extend my sincerest gratitude and appreciation to Dr. R. M. H. Cheng and my thesis supervisors, Drs. H. Hong and R. Rajagopalan, for their insightful guidance, encouragement, moral and financial support throughout the course of this research. Especially, my thanks are due to Dr. R.M.H. Cheng, who initiated this topic and defined the methodology, and with whom I had a great deal of fruitful discussions. His professional direction and research work has greatly helped in implementing this work. My thanks are also due to Mr. Karun Thanjavur, who shared his invaluable experiences and opinions helping me to start with this work. I also appreciate the time spent by Mr. Gilles Huard for helping me with the experiment setup and technical support. I wish to thank my colleague, Ms. Ye Su, with whom I had the fruitful discussions and cooperation in the experiments of this work.

I am grateful to all my colleagues and staff at the Center for Industrial Control for their help and friendship.

This work was supported by Strategic Grant STRO134360 from NSERC awarded to Drs. R.M.H. Cheng and R. Rajagopalan.

TABLE OF CONTENTS

List of Figures

List of Tables

Nomenclature

(θ, R, Z)	Three joints of Yamaha Robot Zeta-1 analogous to a cylindrical coordinate system
(α, β)	Wrist joints of Yamaha Robot Zeta-1
$(\theta, R, Z\ \alpha, \beta)$	Machine Coordinates in Machine Coordinate System of Yamaha Robot Zeta-1
δ	Angle between the tool and the robot x-axis before turning of α
ξ	Angle between the tool and the robot arm before turning of α
a, b, c	Three sides of the triangle formed by the wrist-tool from the top view of the robot
$\{x_1, y_1, z_1\}$	Coordinates of the Machining Tool Tip
$\{x_2, y_2, z_2\}$	Coordinates of the Machining Tool End
$\{x_k, y_k, z_k, A, B\}$	World Coordinates in World Coordinate System of Yamaha Robot Zeta-1
L	Measured Machining Tool Length
${}^{A}BP$	Tool vector in frame A
${}^{A}P$	Vector P in frame A
${}^{B}P$	Vector P in frame
${}^{A}P_{BORG}$	Vector of the origin of frame B in frame A
X_B, Y_B, Z_B	Axes vectors of local coordinate system B

Acronyms

AME	Advanced Modeling Extension
B-Rep	Boundary Representation
CAD	Computer Aided Design
CAM	Computer Aided Machining
CIC	Center for Industrial Control
CSG	Constructive Solid Geometry
LCS	Local Coordinate System
MCAD	Mechanical Computer Aided Design
TPD	Tool Path Data
VR	Virtual Reality
WCS	World Coordinate System

Chapter 1

Introduction

1.1 Robotic Deburring and Polishing

Deburring and polishing are secondary machining processes that remove the burrs and finish the surfaces of a workpiece generated by primary machining processes. These processes are essential from a safety point of view and are critical for mechanical parts in order to satisfy their performance in cases of their mating with each other, when large stresses result. Smooth surfaces and clean edges are always necessary for the workpiece to operate efficiently, such as the case of turbine surfaces.

The use of a robot for deburring and polishing cuts down the cost and improves the quality of the finished products. A robot can reproduce the same motions precisely and exactly. In processing of very valuable components, such as an impeller blade used in an aircraft engine, it takes a human operator very long time to remove the burrs and polish the metal surfaces because of the geometrical complexity of the component. Further, as humans are prone to errors, cost is far too expensive to allow a mistake. In addition, working environments, where dust and noise are harmful, is not conductive for the operator. In such cases a robot plays an important role in carrying out this time consuming work while providing consistent finish all the time.

Deburring can be classified into 38 basic deburring operations [1]; these include mechanical cutting, grinding, brushing and blasting. The term "robotic deburring" concerns the automation of all 38 processes. Robotic deburring is typically used for long-running parts and is not economical for one-time runs of very low quantities because the

engineering and programming time exceed the robotic savings. However, robotic deburring is not the only solution to many deburring processes but a single answer to some needs.

A robot chosen for deburring purpose should have characteristics such as repeatability, accuracy, continuous path capability and self-calibration ability. Bearing theses features, of many industrial robots used for machining, the Yamaha Deburring Zeta-1 Robot [2] is a dedicated machine to carry out automatically deburring and polishing processes. It is a teaching type robot and was developed exclusively for deburring and polishing operations. It has 5 degrees of freedom with rigidity to withstand the deburring or polishing load, which is comparatively much smaller than the primary machining. It also provides a high repeatability accuracy and the consistency required for deburring and polishing.

1.2 Previous Work in Tool Path Generation and Robotic Deburring

Basically, very little difference exists between the tool paths applied for NC machining, which has been widely studied and literatures have been published in this field, and robotic machining. The tool path for NC machining is classified into continuous path and point-to-point path [3]. Continuous path is typically used for milling, cutting, turning and grinding, while point-to-point path are usually employed in drilling, spot welding, punching, tapping and so on. No matter what kind of tool path is needed, the automatic generation of machining tool paths for computer based mechanical 3D modeling is quite challenging. Numerous methods have been developed to solve the problem of automatic tool path generation. Usually, a mathematical model generates the

Tool Posture or Cutter Location data directly and the data is then post-processed into a form that is suitable for the machine (Robot or NC) to execute. This approach requires an integration of CAM activity with CAD system and it shortens the product lead-time and reduces the cost of production [4]. Indeed, many commercial CAD/CAM systems do provide tool path generation algorithms. Various tool path generation algorithms are described in many research literatures.

Loney and Ozsoy [5] have developed an interactive CAD system called CISPA (Computer Interactive Surfaces Pre-APT). It uses a menu driven front end with graphical feedback to guide a user through curve and free form surface definition resulting in a mathematical model. It allows the user to create geometry, modify geometrical component, define a surface, generate tool path and create tool location source files needed to mill the surface. Another workcell developed by Selleck and Loucks [6] processes the workpiece based on the CAD database of the workpiece. It comprises a vision system to determine the orientation of the cylindrical component and inspect the formed edge.

While assuming the dimensions of the workpiece match with the CAD database, it does not hold true in many cases, especially in the case of the refurbished components, which have usually been geometrically reconditioned. For instance, the impeller blades and gas turbines used in aircraft industries may have as large as 2.0 mm of geometrical variations from the original manufactured part errors [7]. Cheng et al [8] proposed an approach that uses a series of probing and mathematical splining techniques to reconstruct the distorted surfaces and edges of refurbished components.

The research work discussed in this thesis is based on the assumption that either the CAD database matches the real workpiece, or the distortion of the workpiece has been detected and its CAD database has been reconstructed so that the CAD database describes the workpiece completely.

Although the current CAD/CAM systems help to program tool path, the user needs to specify the geometry of the component or the boundary of the area to be machined and the cutting parameters. Especially for the CAM purposes, users always need to specify a constant lead angle or a tilt angle in order to define the tool orientation [9]. Therefore, a user-interactive interface and an operative working environment are usually required.

Many robots have been chosen to execute the generated tool path other than the Yamaha Zeta-1 and NC machines. General-purpose industrial robots such as the PUMA 560 [10] and the GE P-50 [11] have been applied to robotic deburring. They are not dedicated deburring robots and they lack flexibility and accuracy for the sake of their multi-task abilities. Other robots used for deburring include a five axis SCARA type of robot, Adept One robot [12], Panasonic HZL four-axis robot [13], GE S-700 six axis robot [14] and a four-freedom-degree IBM SCARA 7576 robot [15]. Their structures and performances differ from one another. However, they do need something in common, which is the tool path for machining.

1.3 3D Mechanical Drawing – CAD Database

Computer science today has made a major impact on the method of 3-dimensional mechanical drafting and design. Designing methods have been drastically improved with

CAD, and three-dimensional modeling and analysis are widely used in mechanical engineering. Based on computer aided design, virtual reality, which will be utilized in the development of this work, emerges as a design and simulation tool.

This thesis is developed based on the technology provided by Spatial Technology's ACIS modeling strategy and the most widely used CAD tool, AutoCAD, which is an affordable CAD software compared to many expensive commercial CAD systems, such as CATIA, Pro-Engineering, CAD-Key and others. Although 3D modeling is not the objective here, for our unique purpose, certain 3D modeling techniques are used to help understand and to develop the CAM deburring/polishing system.

Mechanical parts could be drafted, designed and modeled by a variety of CAD software. There are three basic types of 3D (three-dimensional) models created by CAD systems and used to represent actual objects. They are:

- Wireframe models
- Surface models
- Solid models

These three types of 3D models range from a simple description to a very complete description of an actual object.

"Wireframe model" is a good descriptor and it represents every edge of an object. The surfaces of the object are not defined. No wires exist where edges do not exist. This kind of model is see-through since it has no surfaces to obscure the back edges. Although 3D objects are not very often modeled in wireframe for visualization purposes because of "transparency", wireframe model is extremely useful for the unique purpose of robotic

deburring, because it allows the access to the database of each edge. Figure 1.1 shows an object modeled in wireframe.

"Surface model" provides a better description of an object than a wireframe in the sense of computer aided drafting and designing. It gives a much better visualized view of the object. Surfaces defined in such a model have complicated databases and complexity is involved in the modeling. Also, edges in one surface can not be easily accessed compared to the wireframe models. Surface models are not used in the research of this thesis.

"Solid modeling" is the most complete and descriptive type of 3D modeling. It is much simpler and faster to model a 3D object into a solid model than into a surface one. Construction techniques in AutoCAD combine CSG (Constructive Solid Geometry) and B-Rep (Boundary Representation) together and provides a simple and straightforward construction method. That is, primitive shapes (boxes, cylinders, wedges, etc.) are combined by utilizing Boolean operations (Union, Subtraction, and Intersection, etc.), and the model is then bounded by the edges and surfaces. This approach is so-called AME (Advanced Modeling Extension) in AutoCAD terminology that is mostly used in 3D object modeling. The database of a solid model is even more complicated because it has a complete description of an actual object and it contains more geometric information. As with a surface model, a solid model bears the same characteristic that it does not allow the access to the constructing wires. This is because in AutoCAD the solid model is considered as one entity. Figure 1.2 shows a rendered solid 3D model.

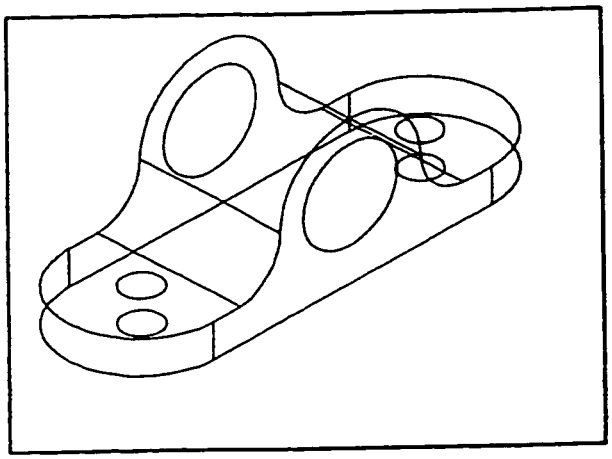

Fig. 1.1 3D Wireframe Model

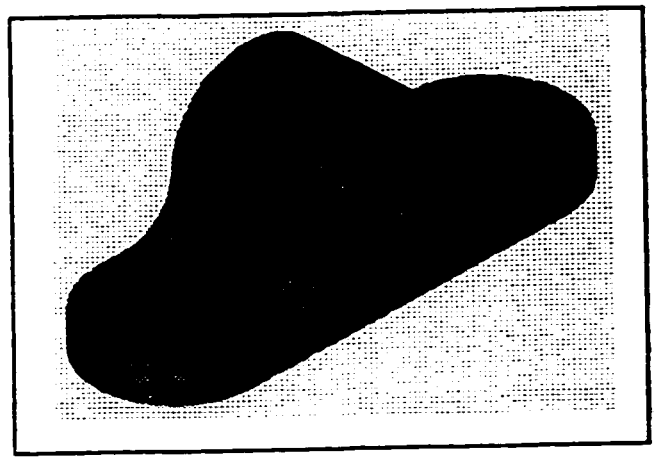

Fig. 1.2 Solid 3D Model

The 3D models are meaningful not only in the sense of mechanical drafting, its database could also be managed for the purpose of numerical machining. There are two main approaches [16] of manipulating the AutoCAD 3D model database. The first is with AutoLISP, a programming language within AutoCAD that enable users to write programs that manage and manipulate graphic and non-graphic data. The second approach is to extract the desired data from the drawing file and organize it outside of AutoCAD with external data management programs, such as Windows WordPad, Windows Excel, dBases and etc. A CAM software based on the second approach is made by Su Ye [17], who extracts geometric information from AutoCAD's 3D model database and reconstructs the geometry outside of AutoCAD platform. As in the course of this work, both approaches are employed so that the advantages of AutoCAD features could be used to develop the CAM package.

1.4 Scope of the Thesis

A common requirement of all manufacturing systems is to generate coordinated movement of the separately driven axes of motion in order to achieve the desired path of the tool relative to the workpiece [18]. In the case of the Yamaha Zeta-1 Robot, in order to process an actual mechanical part, the tool path data must be post-processed before being fed to the five driven axes of the robot as command data. A communication between the Yamaha Zeta-1 and an external workcell has been achieved by mimicking the motion of the joystick of the teaching unit of the robot, by Ayyadevara [7]. To provide the tool path, however, an approach was proposed and initiated by Cheng et al [19], that the machining data could be generated based on the 3D model database of the workpiece, which is constructed in the AutoCAD environment, taking advantage of an underlay programming language of AutoCAD, AutoLISP.

This thesis focuses on detailing the development of the CAM software package. The work includes:

i) Development of a filter and re-constructor for all kinds of 3D "solid" geometric objects drawn by AutoCAD, which have been translated into ACIS file type, to extract pertinent information. Information includes every geometric property of the 3D model, and is used to rebuild a wireframed model.

ii) Development of a "user interactive" interface between the 3D model and the user in the AutoCAD environment.

iii) Development of software using AutoLISP for tool path designing and generation.

iv) Development of a virtual reality model of the Yamaha Deburring Zeta-1 Robot to simulate the kinematics of the deburring and polishing processes.

v) Experimenting by feeding the post-processed data to the Yamaha robot and verifying the generated tool path.

For some specific mechanical jobs, say robotic deburring or NC polishing, in the small to medium sized manufacturing industry, the current commercial CAM software, such as CATIA, Pro-Engineering, are way too perfect and too expensive. They are made to be far more powerful that users hardly have a chance to meet and use those many powerful but irrelevant functions and most of these functions are neglected. A toolbox created in AutoCAD containing software tools specifically made for certain tasks could achieve some savings and users could easily expand the toolbox by adding in customized tools to meet their unique requirements. A main contribution of this research work lies on the development of this CAM software based on AutoCAD.

The hardware chosen for verification of the generated tool path is the Zeta-1 Robot developed by Yamaha Corporation specifically for edge deburring and surface machining tasks. This software tool has been developed based on Mechanical Desktop 2.0, which is basically an AutoCAD R13 with some add-on 3D modeling features, by Autodesk Inc. However, the whole software package could be installed in the regular AutoCAD R13 with some minor modifications, which are stipulated in Appendix 1.

The main contribution of this thesis is the generation of TPD (Tool Path Data) for regular geometries based on an affordable, widely used CAD tool, AutoCAD. A CAM toolbox is created and embedded inside AutoCAD so that the interactive user interface between users and 3D models eases the communication between engineers and mechanical drawings. A kinematic model of the Yamaha Deburring Zeta-1 Robot is also built in a virtual reality environment so that a trial run of a real machining process is

made possible. Virtual Reality application for mechanical engineering is also illustrated in this thesis.

Although this CAM software package is made for robotic deburring and polishing, the work presented in this thesis illustrates the potential of using the AutoCAD database. Therefore, it should not to be concluded that this approach is limited to the deburring and polishing processes only. Similar processing like brushing, fettling and etc. could also be considered. Also, it should not be concluded that this package could be utilised on robot deburring only. Tool path generated by this package could also be used for a NC machine.

1.5 Thesis Outline

Chapter 2 describes an overview of the methodology of this development and features that have been developed in this CAM software package. Chapter 3 presents the organization of the CAM software and the algorithm of each section of the software. Chapter 4 deals with the kinematic simulation of tool path executing in Virtual Reality environment. An experiment made on the Yamaha Zeta-1 Robot to verify the generated tool path is stipulated in Chapter 5. Conclusions and recommendations for future work are presented in Chapter 6.

Chapter 2

3D Database and Reconstruction

2.1 Introduction

Since the current CAD tools present very precise geometry, a CAM approach based on a database of AutoCAD is proposed. One of the typical and simplified workpieces that is chosen for the development of this CAM package is presented in Figure 2.1. A 3D model of the same workpiece is presented in Figure 2.2. It consists of most common geometrical components such as, straight edges, cylindrical edges, intersectional edges, flat surfaces and cylindrical surfaces. These are all the components that this thesis deals with. In this chapter, an investigation has been done on the CAD database of these geometrical components and interesting geometrical information has been extracted from the database.

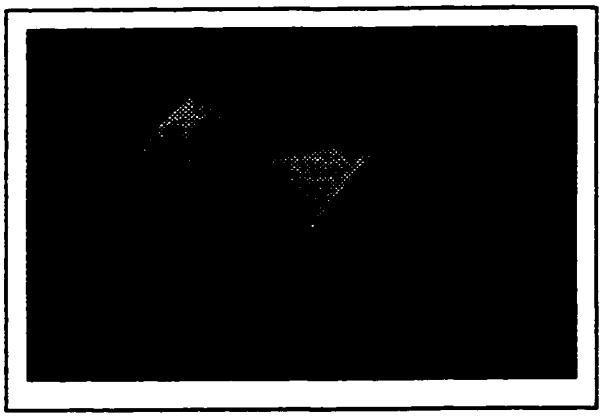

Fig. 2.1 A Typical Workpiece [17]

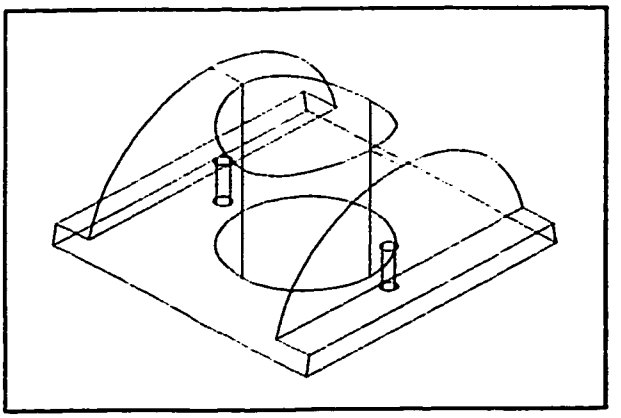

Fig. 2.2 3D Solid Model in AutoCAD

2.2 AutoLISP Language

AutoLISP is a unique language specifically designed for AutoCAD. It is one of the first high-level programming languages invented for AutoCAD users to develop programs using English-like expressions [20]. It lets users write customized programs to control every aspect of the drawing, the database and third party applications [21]. It also lets users draw in 2D or 3D and construct a 3D model, providing high efficiency and productivity. In this research work, since it is desired to generate TPD based on the database of AutoCAD drawings and take advantage of AutoCAD's CAD facilities, AutoLISP is the only choice to enable easy programming in AutoCAD's platform.

2.3 Examining AutoCAD Database

The workpiece shown in Figure 2.2 is modeled in AutoCAD R13 with the most common method, solid modeling. It is one of the easiest and fastest ways to draw a 3D object and usually, workpieces are modeled into such a solid model. However, for such a "solid" object that AutoCAD presents as one entity, the edges, which are presented as line, arcs, circles and spline curves, can not be easily accessed. Edges are formed by intersections and boundaries of surfaces. The databases of the edges are hidden behind the logic of these boundaries instead of appearing in the solid model database.

In surface modeling, surfaces are presented as minimum units, which are entities similar to a solid model, and one also can not access only one edge of a certain surface. The database of the drawing in AutoCAD could be listed either by simply inputting an AutoCAD command "list" or by an AutoLISP command "(entget (car (entsel)))". The former command gives a simplified and clearer list of data while the latter a much more detailed data with AutoLISP's internal code for the properties of the component. The data

coming out of the latter command could be stored in a user-defined variable and be manipulated by specifying the internal codes of the properties. This is the data source from the AutoCAD drawing.

2.3.1 Solid and Surface Database

The database of a solid workpiece is shown in Table 2.1, which does not show any pertinent geometric information for the CAM package under development.

3DSOLID Layer: 0
Space: Model space
Handle = 42
Bounding Box: Lower Bound X = -99.8749 , Y = -100.0000, Z = 5.0000 Upper Bound X = 99.8749 , Y = 100.0000 , Z = 100.0010

Table 2.1 Database of a Solid Model

To access the database of every component for such a solid 3D model in AutoCAD, a wireframed model with the same size and shape must be used to substitute the solid one. Indeed, AutoCAD does provide a technique to transfer a solid model into a wireframed one by "exploding" the solid model twice. "Exploding" the solid one time gives a surfaced model that one can specify the surfaces that AutoCAD describes as "regions" and "bodies". Alternatively, the AutoLISP's command "(entget (car (entsel)))" presents a totally different format of the database and in case of solids, regions and bodies, the output database by this command are hardly readable. Table 2.2 and 2.3 show the database of a typical "region" and "body" respectively. Again, the data in the table does not contain any pertinent geometric information for the CAM package under development.

REGION Layer: 0
Space: Model space
Handle = 51
Area: 31938.9074
Perimeter: 1176.4908
Bounding Box: Lower Bound X = -99.8749, Y = -100.0000, Z = 5.0000 Upper Bound X = 99.8749, Y = 100.0000 , Z = 5.0000

Table 2.2 Database of a "Region"

BODY Layer: 0
Space: Model space
Handle = 53
Bounding Box: Lower Bound X = -99.8749, Y = -100.0000, Z = 5.0000 Upper Bound X = 99.8749, Y = 100.0000 , Z = 100.0010

Table 2.3 Database of a "Body"

2.3.2 Wireframe Databases

While "exploding" the "region" and "body", lines with two end points, circles with radius and center location, and spline curves with control points come up. Figure 2.3 shows a "double exploded" 3D workpiece. Table 2.4 lists the database of the bottom large circle. Table 2.5 is the database of the same circle given by AutoLISP's command "(entget (car (entsel)))". The database of all other components could be found by the same method and they all appear similar to the ones shown in Table 2.4 and 2.5.

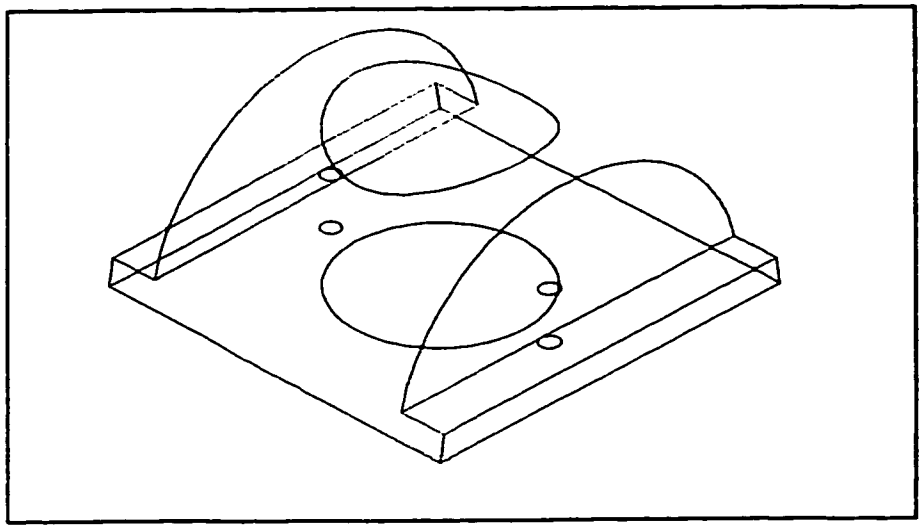

Fig. 2.3 "Double Exploded" 3D model (Wireframe)

CIRCLE Layer: 0
Space: Model space
Handle = 6F
center point, X= 10.0887 Y= -18.0897 Z= 58.7473
radius 50.0000
Extrusion direction relative to UCS: X= -0.2863 Y= 0.5133 Z= -0.8090
Circumference 314.1593
area 7853.9816

Table 2.4 Database of a Circle from AutoCAD

((-1 . <Entity name: 21e0738>) (0 . "CIRCLE") (5 . "6F") (100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10 1.42109e-014 17.7738 -59.7022) (40 . 50.0) (210 -0.286296 0.513348 -0.809017))

Table 2.5 Database of the same Circle Given by AutoLISP Command

The pertinent geometric information shown in Table 2.4 are "center point", "radius" and "Extrusion direction". In Table 2.5, the important information are (10 1.42109e-014 17.7738 -59.7022), (40 . 50.0) and (210 -0.286296 0.513348 -0.809017). Carefully examining the circle's database, it is seen that data presented in Table 2.5 could not be clearly mapped into the data in Table 2.4. For example, the coordinates of the center point in Table 2.4 is "X= 10.0887 Y= -18.0897 Z= 58.7473" while in Table 2.5, it is presented as (10 1.42109e-014 17.7738 -59.7022). Note the "10" is an internal code in AutoCAD database, representing a center point and the data following it are coordinates of that center point, x, y, z respectively. To understand the meanings of all the internal codes used by AutoCAD, the reader is referred to the AutoCAD Reference Manual [22] and AutoCAD database Book [16]. The reason for this matchless data set is that AutoLISP presents and constructs circle database from a UCS (User's Coordinates System) point of view so that the circle could be drawn and its properties could be modified in two dimensional space. And the UCS is defined in respect to the tilting angle of the normal direction of the plane formed by the circle. From the modeling point of view, it is much easier to draw a circle in 2D space. However, difficulties are introduced to find how the UCS is oriented. The logic determining the UCS is embedded in the AutoCAD modeling system and could not be easily found.

AutoCAD provides an alternative to draw a circle by constructing the circle as an ellipse with a radius ratio of 1.0. The ellipse could be drawn in the WCS (World Coordinate System). From a geometry point of view, this ellipse possesses the same geometrical properties in the space as a circle does under such circumstance that they

have the same radius, identical normal vector of the plane they form, and the same location of the center point.

Since the database of the circles and arcs taken by "double exploding" the solid model are not so simple and easy to use, instead of tracing the UCS of those circles and arcs, reconstructing the circles and arcs into full ellipses and ellipse arcs becomes necessary. Table 2.6 and 2.7 show an example database of an ellipse generated by AutoCAD and AutoLISP respectively. The pertinent geometric information shown in Table 2.6 are "Center", "Major Axis", "Minor Axis" and "Radius Ratio", while in Table 2.5, instead of giving the same information, Center (10 5.61425 -19.0544 58.0015), Major Axis (11 2.45146 41.6799 27.5096), Extrusion direction (210 0.16211 -0.550191 0.819152), Radius ratio (40 . 1.0) are given sequentially. It is seen that data from the two sources match each other.

ELLIPSE Layer: 0
Space: Model space
Handle = 90
Center: X = 5.6142 , Y = -19.0544 , Z = 58.0015
Major Axis: X = 2.4515 , Y = 41.6799 , Z = 27.5096
Minor Axis: X = -49.2777 , Y = -2.4515 , Z = 8.1055
Radius Ratio: 1.0000

Table 2.6 Database of an Ellipse from AutoCAD

((-1 . <Entity name: 21e0880>) (0 . "ELLIPSE") (5 . "90") (100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbEllipse") (10 5.61425 -19.0544 58.0015) (11 2.45146 41.6799 27.5096) (210 0.16211 -0.550191 0.819152) (40 . 1.0) (41 . -2.50111e-016) (42 . 6.28319))

Table 2.7 Database of the same Ellipse Given by AutoLISP Command

As to the straight edges of the solid model, they are reconstructed as straight lines in the "double exploded" wireframe model. Since the database of the straight lines are very simple (two end points are listed in the database and no UCS involved), there is no need to reconstruct the straight lines. On the other hand, the database for a spline curve is very complicated. Although AutoCAD's command "spline" provides a great tool for the user to create 3D spline though a set of vertices [23], the spline yielded from the "double explosion" does not provide those vertices or fitting points. It only includes the constructing information of the spline, which are the control points of the spline, and can not be directly used. A methodology is devised to extract the spline data and is detailed in Chapter 3.

2.4 Examining ACIS Database

3D solid models in AutoCAD could be "exported" to 3D model types of other 3D modeling engines, such as 3D Studio, ACIS, and so on. And the "exported" file types includes DXF, 3DS and SAT. Most of the types of files are unreadable except for DXF and SAT files, which stores the geometry information in ASCII code.

ACIS is an object-oriented geometric modeling toolkit designed for use as a geometry engine within 3D modeling applications [24]. Taking advantage of the ACIS file format, which stores a 3D-geometry model in ASCII codes, enables one to get every detail of the 3D solid model. From 3D modeling point of view, it provides an open architecture framework for wireframe, surface and solid modeling from a common unified data structure. As an example, the following list is a part of an ACIS file.

```
105 219 2 0
body $2 $3 $-1 $-1 #
body $4 $5 $-1 $-1 #
f_body-lwd-attrib $-1 $6 $-1 $0 #
lump $7 $-1 $8 $0 #
f_body-lwd-attrib $-1 $9 $-1 $1 #
lump $10 $-1 $11 $1 #
ref_vt-lwd-attrib $-1 $-1 $2 $0 $12 $13 #
ref_vt-lwd-attrib $-1 $-1 $-1 $3 $12 $13 #
shell $14 $-1 $-1 $15 $3 #
ref_vt-lwd-attrib $-1 $-1 $4 $1 $12 $13 #
.....

coedge $266 $154 $226 $253 $267 0 $129 $-1 #
epar-lwd-attrib $-1 $268 $-1 $155 #
vertex $-1 $267 $269 #
vertex $-1 $203 $270 #
ellipse-curve $-1 0 -75 0 0 1 0 100 0 0 1 #
.....
copar-lwd-attrib $-1 $-1 $-1 $158 #
coedge $271 $158 $182 $167 $221 1 $129 $-1 #
color-adesk-attrib $-1 $-1 $159 $120 256 #
point $-1 97.979589711327122 75 20 #
epar-lwd-attrib $-1 $272 $-1 $163 #
straight-curve $-1 97.979589711327122 0 20 0 -1 0 #

.....
```

Table 2.8 Text Format in ACIS file

Table 2.8 is a 3D-model database stored in ACIS file format, known as SAT file. Each line represents a component or property of a component and ends with a "#" sign. The first string in a line is an identifier denoting the class of the component, and the signs and numbers that follow are the data associated with it. All of the lines are bounded by a unique logic of ACIS. In a single line, the number followed by a "$" sign denotes a line number where a certain property is described. The "–1" follows the "$" sign means that there is no lines related to the component or property described by the line itself.

2.4.1 Geometric information in ACIS file format

The geometric entities are represented by different identifiers in the ACIS file. They are body, lump, shell, subshell, face, loop, coedge, edge and vertex. Within each of

these entities, geometric components such as Points, Straight lines, Ellipse curves and Intersection curves are defined. Since only the wireframe is of interest, geometric components that construct the wireframe have been studied and extracted. Other data such as body, lump, coedge, edge and so on are not relevant to the development of the CAM package.

2.4.2 Geometry Components

The ACIS file defines some basic geometry as follows:

1) Points: In SAT file, points are presented in the form of :

   point $-1 x y z #

   where, x, y, z are the coordinates in the world coordinate system.

2) Straight lines, in SAT file, are written as:

   straight-curve $-1 x0 y0 z0 xn yn zn #

   where, x0 y0 z0 are the coordinates of a point on the line and xn yn zn are the vector elements of the line.

3) Ellipse and arcs are all in a ellipse-curve form as follows:

   ellipse-curve $-1 x0 y0 z0 xn yn zn xm ym zm r #

   where x0 y0 z0 are the coordinates of the center. xn yn zn are the normal vector elements of plane that the ellipse forms. xm ym zm are the major radius vector elements. r is the ratio of the major radius to the minor radius.

4) Intersection curves (by a default 4^{th} order spline) has long list of coordinates of the control points. Here is an example.

   intcurve-curve $-1 0 { surfintcur nubs 3 periodic 17

   0 3 0.37724420556371729 2 0.75448841112743459 2 1.131727215958044 2 1.5089660207886535 2

1.8862048256192623 2 2.2634436304498711 2 2.6406878360135817 2 3.0179320415772923 2

3.3951762471410154 2

3.7724204527047385 2 4.1496592575353457 2 4.5268980623659534 2 4.9041368671965611 2

5.2813756720271678 2 5.6586198775908771 2 6.0358640831545864 3

* 6.4999999999999991 -2.5980762113533178 3

6.4999999999999991 -2.5980762113533178 2.8742519314787609

6.4906880451419964 -2.6217592102399325 2.7402414419277692

6.4514072357916072 -2.7169831266064688 2.4939140648628841

• • • • • •

6.348177005947683 -2.9510329220558593 3.7957945373038515

6.42126838025567 -2.7884151810958322 3.618417755372433

6.4514072357916099 -2.7169831266064635 3.5060859351371061

6.4906880451419973 -2.6217592102399307 3.259758558072225

* 6.4999999999999991 -2.5980762113533178 3.1257480685212364

6.4999999999999991 -2.5980762113533178 3

0.001

cone 4.3301270189221928 -2.5000000000000009 3 0.8660254037844386 -

0.50000000000000022 0 0.50000000000000022 0.8660254037844386 0 1 0 -1 0

cone 0 0 3 0 0 1 7 0 0 1 0 -1 0

nullbs

nullbs

} #

The line with * is the start point of the intersection curve and the second * line is the end point. ("*" is not a part of the SAT file.) Notice that following the end point, there are two lines that start with the word "cone". They imply that the intersection curve is the intersection of two cones.

The topology of the data of this intersection curve, however, is not straightforward. Comparing and investigating a variety of spline data, it is found that:

1) Intersection control points always end with a single data 0.001, followed by the names of the constructing components.

2) In the beginning of the coordinate area, integer 2 is always inserted in between some data, which are the slope values between two adjacent control points. When integer 2 changes to integer 3, the coordinates of the spline begin.

Note that the control points are not the fitting points on the spline curve. They are just intersecting points of the tangential lines at certain fitting point such that the ACIS spline builder could find the fitting points by using those control points. A full reconstruction of a spline curve will certainly make use of the control points, degree of the spline and so on, and give a complete description of the spline, including all the fitting points, which are of most interest. Logically, a conclusion can be drawn that AutoCAD and ACIS 3D modeling engine must have the construction algorithm built into their systems so that they can build a spline curve by taking certain control points.

Since AutoCAD presents a unique data form for circles and arcs, which are not convenient to use, parameters of circles and arcs are to be abstracted from the ACIS file and reconstructed by AutoCAD's "ellipse" command. Data concerning spline curves will be discussed in Chapter 3. The two points itemized above concerning spline curves might be meaningful for future work on ACIS database. However, they do not have particular usage in this thesis. For further detailed information in ACIS database, the reader is referred to the ACIS Geometric Modeler Format Manual [24].

2.5 Reconstructing Database

As far as the AutoCAD drawing is concerned, a workpiece is, within the scope of this work, a combination of regular geometry such as straight lines, circles, arcs, sphere surfaces, cylindrical surfaces and so on. Based on the assumption that the workpiece is modeled by using AutoCAD 3D solid modeling techniques, the workpiece is always presented as a solid model. By solid model, AutoCAD database defines it as one object entity and does not provide the details of that object such as edges and surfaces. In other words, the edges and surfaces are not accessible in the case of a solid model while those of a model in wireframe are. In the development of this CAM package, it is assumed that a solid model is given and the tool path must be generated based upon the given solid model drawing.

It is desired, therefore, to rebuild the solid model into a wireframe one so that just a mouse click could access each edge and surface. As mentioned before, by "double exploding", the solid model could be transformed into a wireframed one. The database of a transformed drawing in wireframe has been examined. However, the database for circles and arcs are not ideal for use. As previously mentioned, an alternative file type of drawing, SAT file by ACIS, provides geometry details of each component of a solid model in ASCII code, which is readable. Now, two types of database are available and each type has its own advantages and disadvantages for our purpose, because the style that AutoCAD and ACIS write their database are for the purpose of their unique 3D modeling approaches. Therefore, it is desired to combine these two types of database for reconstructing a 3D model.

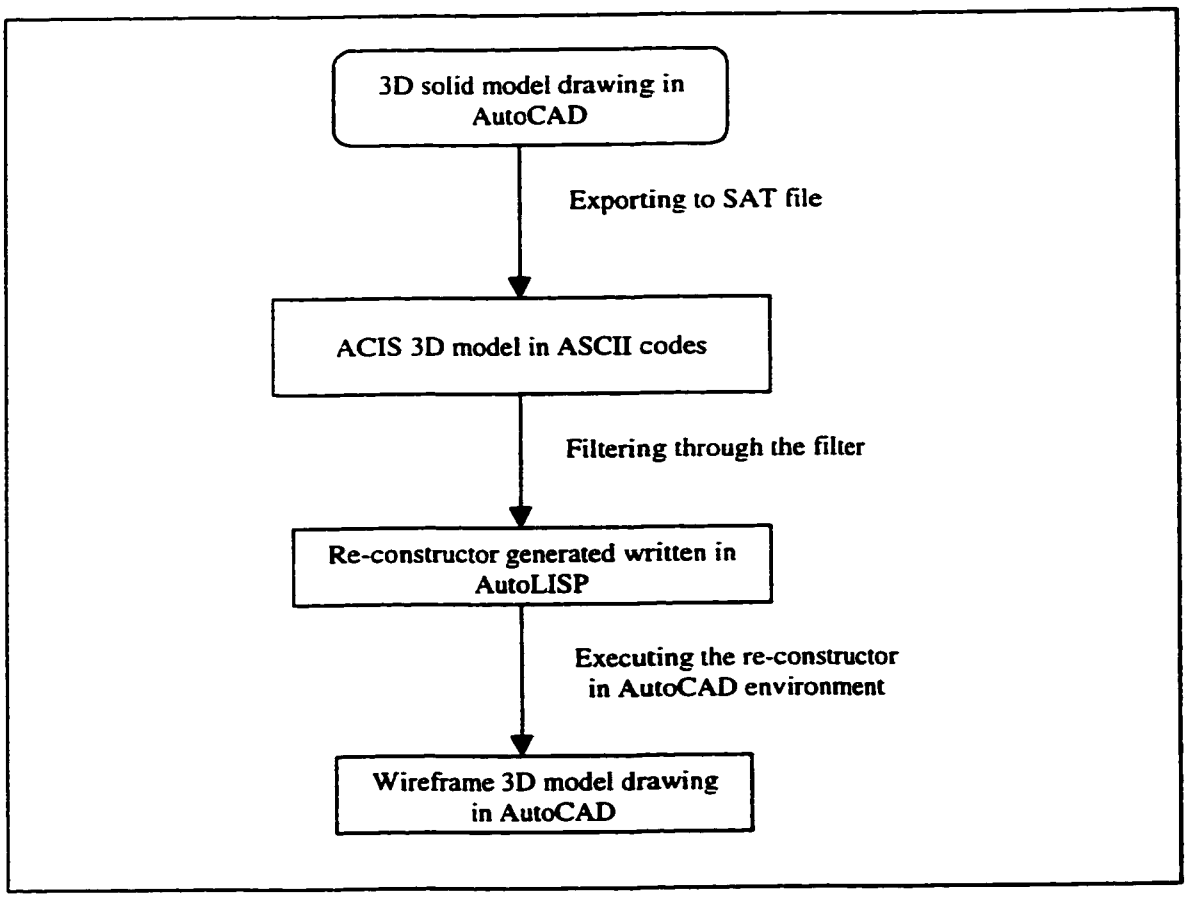

Fig. 2.4 Processing a 3D model

Given an object, the possible edges are straight edges, circular edges, and intersection edges of cylinders, spheres and cylinders, cylinder and torus and so on. These are the edges that have been so far studied. In AutoCAD terminology, a straight edge is referred to as a LINE. A Circular edge is an ARC or CIRCLE, and the edge formed by the intersection of circular geometry is referred to as a SPLINE because AutoCAD uses a 4th order spline function to generate the intersection curve by default.

A filter is programmed in C language and also functions as a re-constructor builder. It scans the SAT file and takes the parameters of lines and ellipses out of the SAT file and adds them into AutoLISP constructing commands. As a result, a constructor

written in AutoLISP will be generated. Those straight lines, circles and arcs of the "double exploded" wireframe model will be replaced by simply executing the re-constructor within the AutoCAD platform. A schematic chart that describes the processing of a 3D-model database is presented in Figure 2.4. Note that the filter is run under the DOS shell instead of AutoCAD environment. The flow chart for the filter is presented in Figure 2.5.

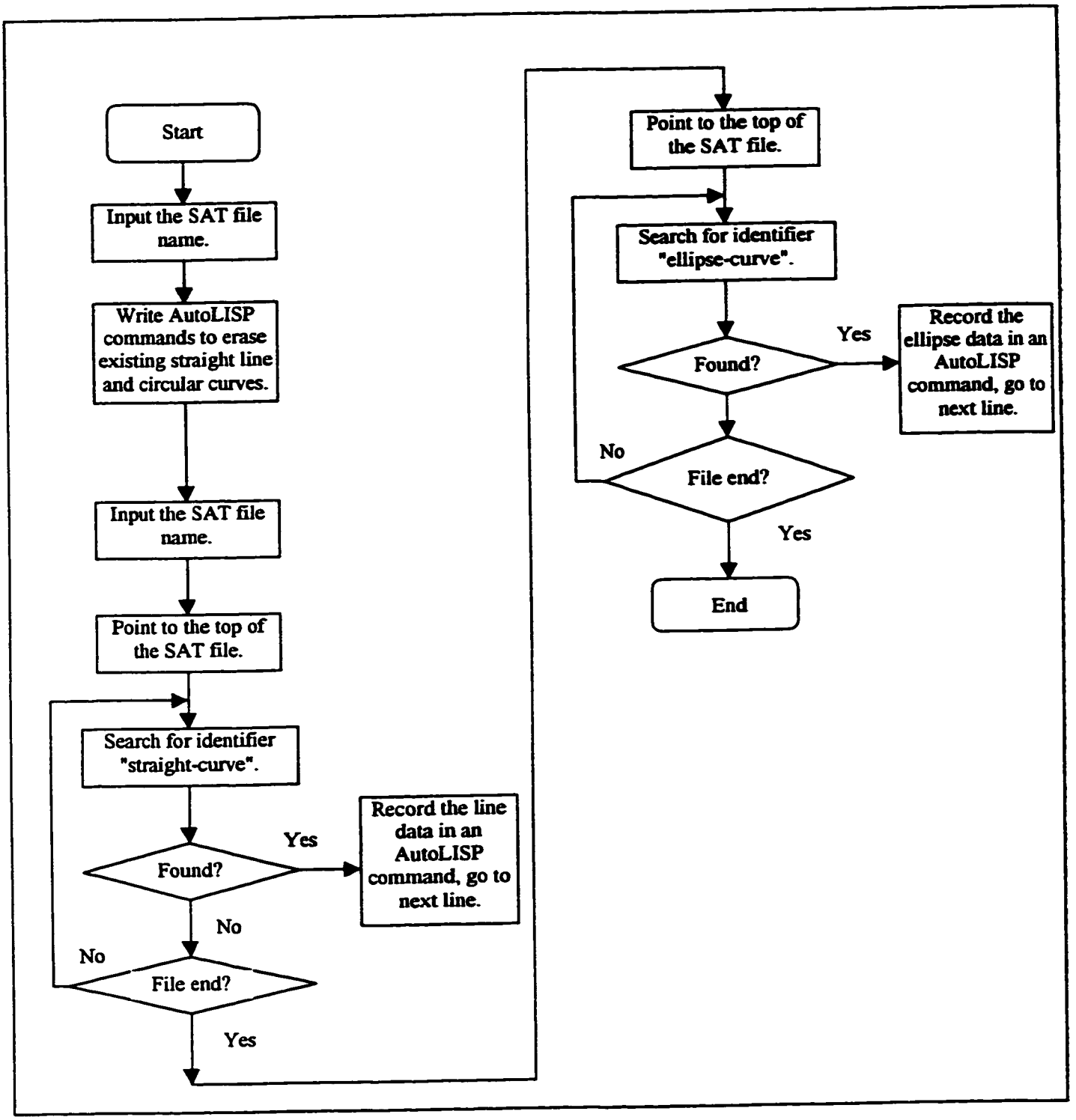

Fig. 2.5 Flowchart of the Filter Program

A wireframe model with reconstructed database is presented in Figure 2.6. It looks no different from the one in Figure 2.3. However, the AutoCAD database of the circular curves is described by "ELLIPSE" instead of the former "CIRCLE" or "ARC".

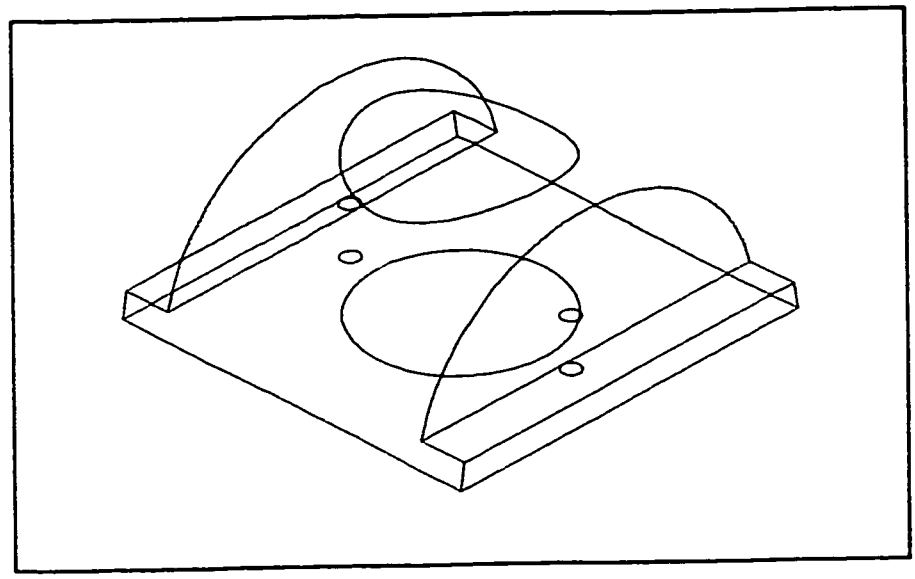

Fig. 2.6 Reconstructed 3D-model (Wireframe)

The reconstructed database of the circular edges is found to be similar to the formats shown in Table 2.6 and Table 2.7, which are desirable. Note that the database for the intersection edge (spline curve) has not been altered while re-constructing the 3D model. This will be further described in Section 3.3.3.5.

2.6 Summary

Two kinds of 3D model database have been investigated, ACIS model and AutoCAD model. The ACIS model, which is written in ASCII code, is processed outside of the AutoCAD environment so that useful geometrical information of the 3D model are extracted and stored. Investigation on the AutoCAD 3D-model database draws the

conclusion that a reconstructed wireframe model is suitable for the purpose of processing edge database. A 3D-model re-constructor is then made to rebuild the workpiece into a wireframe model, which was originally modeled as a solid model.

Chapter 3

Structure of the CAM Software Tool

3.1 Introduction

The major objective of this work is to generate tool path for workpieces containing "regular" geometric entities, based on their AutoCAD database. By "regular" geometry, as mentioned before, straight lines, circular curves and the intersection curves of objects that AutoCAD constructs as 4^{th} order spline curves are included. In order to provide good utilities, flexibility and user-interactivity, which are emphasized in designing this CAM package, a user-friendly interface has been built into the AutoCAD platform so that AutoCAD facilities could be taken advantage of.

A customized AutoCAD working platform is created and is briefly introduced in section 3.2. Feature designs of the CAM package are detailed in section 3.3. The results yielded from those designed features are graphically presented in these sections.

3.2 User Interface – Working Platform for CAM

3.2.1 Menus and Toolbars

AutoCAD is known as an open architecture platform that enables users to customize their own applications. Customized menus and toolbars could be added in by modifying certain ASCII text files that are provided by AutoCAD. Command aliases could also be included into AutoCAD by loading AutoLISP files, where the new commands are defined, generated by the AutoCAD user.

In AutoCAD R13 itself, pull-down menus, drawing window, command window and over 45 named tool bars are available in its user interface [25]. AutoCAD allows the user much more control over the command interface than most CAD tools. The AutoCAD menu file is an ASCII file containing AutoCAD command strings and macros. It also defines the type and appearance of the menu. For further details, the reader is referred to the AutoCAD Customization Guide [26] and AutoCAD13 Secrets [21].

The user interface of this CAM package is embedded into the AutoCAD environment, as shown in Figure 3.1. A pull-down menu named "ToolPath" is inserted in front of AutoCAD's "Help" menu. It includes database reconstruction and TPD generation for edging and polishing. Also, two toolbars are created corresponding to the added menu and are named as "Edging" and "Polishing". Text files of ASCII codes that are required to be modified are as follows:

1) ACAD.MNS

This is the AutoCAD default menu source file. The contents inserted into this file are listed in Appendix 1. In Mechanical Desktop, the main menu source file is named as MCAD.MNS. Basically, these two menus have the same text structures and the customized content is to be inserted into the corresponding locations respectively in ACAD.MNS and MCAD.MNS. Once the new menu source file ACAD.MNS replaces the old one, it has to be re-complied in the AutoCAD environment so that the new menu and toolbars can begin to function. In the AutoCAD command prompt, giving the "menu" command and by selecting the ACAD.MNS, AutoCAD will compile the new menu source file. Also, two toolbar

names, "EToolPath" and "PToolPath", should be created in AutoCAD's command "_tbconfig".

2) ACAD.LSP

This AutoLISP file is automatically loaded every time when the AutoCAD application is started. The contents inserted into this file are listed in Appendix 1. Text functions are added to generate two layers named "Tool" and "Arrow". The AutoLISP file named "CIC.LSP", which is the mainstream of the developed CAM package, is included to be loaded into the AutoCAD environment. By loading "CIC.LSP", the newly created commands for robotic deburring and polishing become valid.

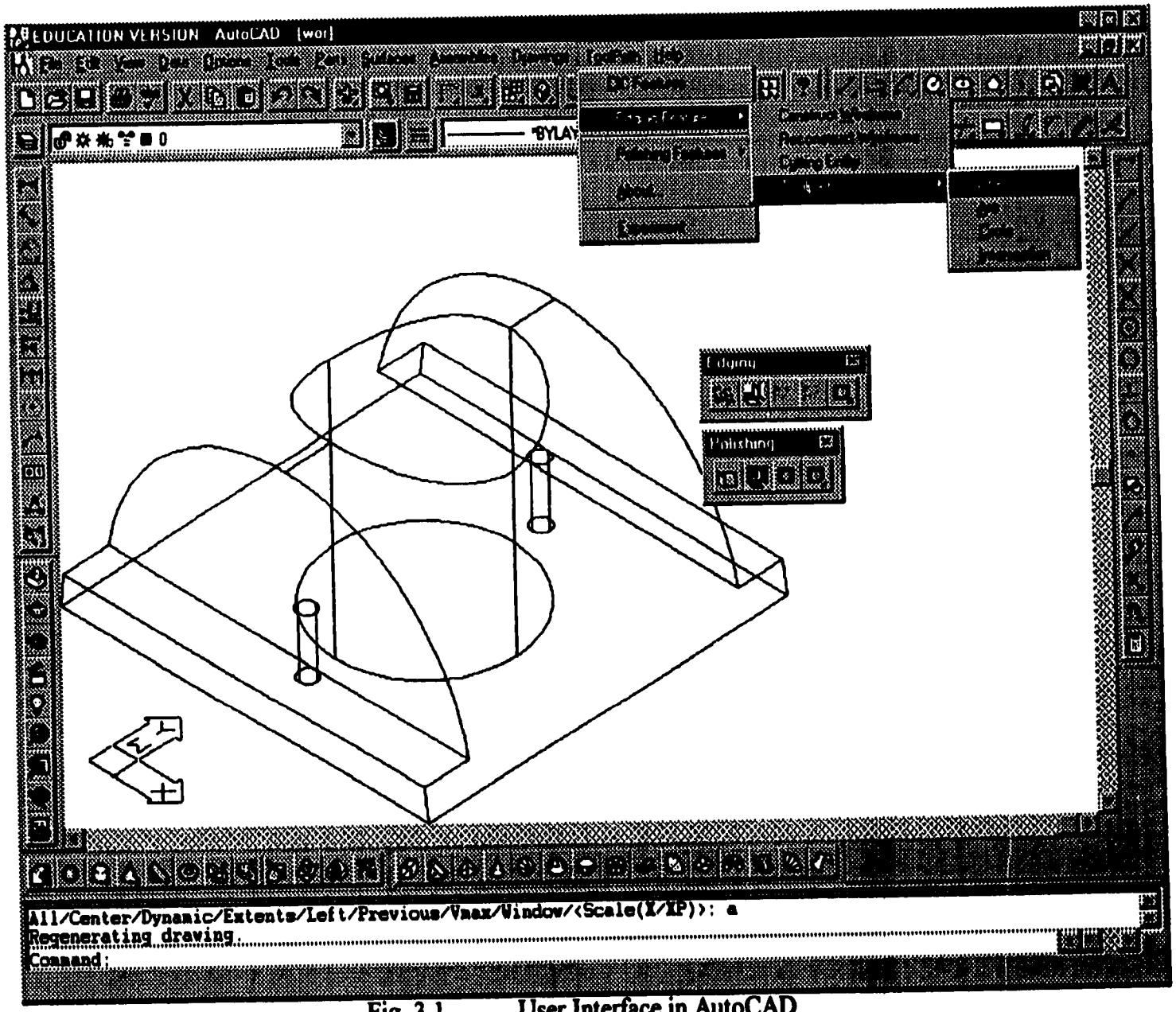

Fig. 3.1 User Interface in AutoCAD

3.2.2 New Commands

There are eight new commands that have been designed and created for the robotic deburring and polishing. The AutoLISP programs that define those eight commands are listed in Table 3.1 and they are loaded as a bunch when "CIC.LSP" is loading. The eight commands could be executed by typing the command names to the AutoCAD prompt or simply clicking on the icons appearing on the new toolbars.

AutoLISP Files Developed for the Interface	Command Names	Feature Description
CIC.LSP	Not Applicable	Loading the following 8 commands
CIC_EXPLODE.LSP	CIC_EXPLODE	Convert solid model into wireframe
LISP.LSP	REWIRE	Reconstruct the database of the wireframe model
CIC_CUT.LSP	CIC_CUT	Break entities in the wireframe model, similar to AutoCAD's "_break" command
CIC_LINE.LSP	CIC_LINE	Generate TPD for straight edges
CIC_CIRCLE.LSP	CIC_ARC	Generate TPD for circular edges
CIC_INTERSEC.LSP	CIC_INTERSEC	Generate TPD for intersection edges
CIC_PLANE.LSP	CIC_PLANESURF	Generate TPD for plane surface
CIC_CYLSURF.LSP	CIC_CYLSURF	Generate TPD for cylindrical surface

Table 3.1 New Commands

"CIC.LSP" also loads another AutoLISP file named "mathbox.lsp" into application. As its name implies, "mathbox.lsp" is a math box that contains mathematical functions for 3D calculations and vector manipulations. Sixteen functions, which have been used in the implementation of the eight main commands, are found in this box as listed in Table 3.2.

Maths Commands	Command Description
sgn	Return the sign of a rational number, +1 for 0 or positive numbers, -1 represents negative numbers
rond	Round a real number to the closest integer
rond1	Round a real number to the larger closest integer
ronvec	Round a vector
mk_vect	Make a vector, given the end points
neg_vec	Make a negative vector of a given vector
mk_unit_vect	Make a unit vector from a given vector
mk_i_line	Make a unit vector, given the end points
shl	Shift a vector list left by one place
shr	Shift a vector list right by one place
dotproduct	Vector dot product (three dimension)
x_prod	Vector cross product (three dimension)
rotrans	Rotation transformation for coordinates of one point in UCS, given UCS directions, returning coordinates in WCS
invmat	Invert a 3X3 matrix
toolang	Calculate the angle in XY plane of a straight line in 3D
tool_rot	Rotate a straight line in 3D and return a vector of the line in current UCS
polish_dir	Rotate a straight line in 2D and return a vector of the line in current UCS

Table 3.2 Mathematical Functions in mathbox.lsp

Note that the commands that are listed in Table 3.1 and 3.2 could be run in AutoCAD as well as those in Table 3.1, as the CIC.LSP is being loaded.

3.3 Features of the CAM Package

Features designed for the CAM package are classified into three classes:

- Database reconstruction
- Edge cutting
- Tool path generation for deburring straight, circular and intersection edges respectively and tool path generation for polishing plane surface and cylindrical surface.

3.3.1 Re-constructing Wireframe 3D Model

This feature transforms a 3D solid model into a wireframe model and enables the access to the AutoCAD database of the constructing wires. In other words, it makes the database of a 3D model accessible for programming. As mentioned in Chapter 2, a solid 3D model represented by ACIS and AutoCAD has different descriptions for the 3D data. Basically, only those constructing wires such as lines, circles, arcs (include full circles) and intersection curves (spline) are of interest.

In ACIS database, coordinates of the end points of a line are given for a straight line. Circles and arcs are described as ellipse curves and the coordinates of the center, major axis vector, minor axis vector, normal vector and axis ratio are given. The information are sufficient to generate the tool path. However, for the intersection curve (spline curve), only the control points are given. The control points of a spline are defined as derivative factors of certain fit points on the curve. Using those control points, the equation of the spline could be built and coordinates of every point on the curve could be calculated by using certain spline approaches, such as parabolic spline, cubic spline and

even higher order spline. Implementing a spline technique to find the coordinates of desired points is not complicated but would require a lot of programming, and it is not necessary because that would be a repeat of what AutoCAD has done. AutoCAD itself uses the 4th order spline to build spline curves by default.

On the other hand, a solid model could be transformed by AutoCAD's explode function, into wireframe. And the database for line, circle, arc and spline are stored in AutoCAD database form. For lines, the database has its two end points, layers, handles and so on. For spline curves, same as ACIS, the database also provides control points only. For arcs and circles, there is a small abnormality as discussed in Chapter 2. AutoCAD was developed from 2D drawing to 3D drawing and it defines circular curves as 2D drawing except for the ellipse. The database of arcs and circles present a "strange" order of coordinate arrangement. For example, instead of giving the center coordinates, it provides an offset point whose coordinates are derived by some formula in AutoCAD, which is unknown.

The present objective is to rebuild a wireframe with the same edges as the solid model such that the database of each edge could be easily accessed and utilized. Taking the AutoCAD and ACIS database formats into consideration, it is seen that:

1) Database for straight lines are sufficient either in AutoCAD or in ACIS;

2) ACIS provides a better database for circular curves;

3) For spline curves, nothing can be done unless a spline function is to be made. However, for an open spline curve, coordinates of two fitting points, starting point and end point, are given by the AutoCAD database. This will be fully described in section 3.3.3.5.

Based on 1) and 2), equations for straight lines and circular curves could be established by using the AutoCAD or ACIS database. A filter has been designed to take all the information of lines and circular curves from the ACIS type file and automatically generate an AutoLISP file, where the reconstruction commands are defined and given. Note that this filter command is to be issued in the DOS shell to manipulate the ACIS text file. This is the first and the only step that is processed outside of the AutoCAD environment in this CAM package. Exporting the 3D model to an ACIS file and running the filter program, then the re-constructor, LISP.LSP file, will be generated.

Running the "rewire" command, which is defined in LISP.LSP, will explode the solid model twice (exploding once gives surface boundaries) and will erase all the lines and circles except the intersection curves. Lines, arcs and circles with the same parameters as the removed ones will be redrawn by LISP.LSP. A similar model with the same size and dimension but in wireframe is then built. The re-constructed wireframe model is shown in Figure 2.5. Figure 3.2 presents the flowchart of the LISP.LSP, which is automatically generated by the filter program.

As for the spline curves, coordinates of two end points are provided in the open spline database. Based on this, all the fitting points could be found theoretically by "breaking" the existing spline curve into a certain amount of pieces. Therefore, there is no need to use the existing database, which provides all the control points, to build a spline function and find the fitting points on the curve. There is another important and useful property that the database of the open spline curve provides the coordinates of the control points. The first two control points forms a tangential line of the spline curve at the

starting point and the last two at the end point. How to find the fitting points and make use of the control points will be discussed later in section 3.3.3.5 of this chapter.

Up to now, a wireframe 3D model with a "better" database is established. The wires that construct the model could be specified and the tool path along the wire could be generated. As to the polishing process, which deals with surfaces, the database of a surface is not involved in the tool path generation. The tool path for polishing could be simply considered as a repetition of a deburring process and indeed, implementation of the polishing tool path generation is done by repeating the straight edge tool path.

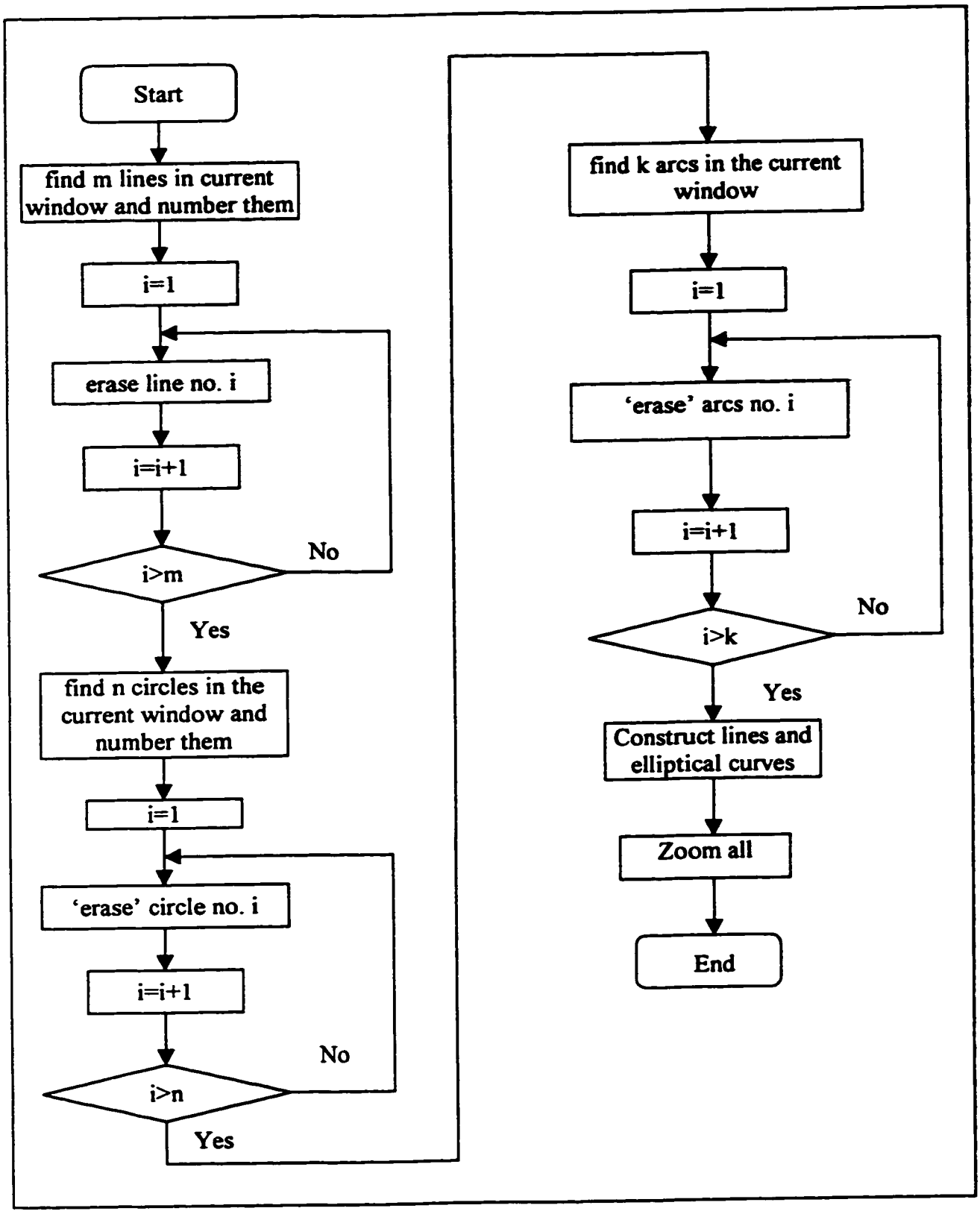

Fig. 3.2 Flowchart of the Re-constructor (in AutoLISP)

3.3.2 Edge Cutting

Although all the edges are ready for selection, one may want to machine only one part of the selected edge. Therefore, a breaking feature must be provided. One can use AutoCAD's "_break" function. There are two issues with the AutoCAD built-in break feature: (1) one broken part will be erased once the entity is broken; (2) additional commands must be issued to keep the broken part from being erased. Hence, a breaking feature with flexibility has been developed. The newly created "cutting" feature enables the users to break the wires more easily. The user could break the edge into two pieces by pointing out one desired point. (For the closed edge, two desired points are expected to break the edge.) A "broken" spline curve is shown in Figure 3.3 and a "broken" arc in Figure 3.4. The entity broken by AutoCAD's "_break" looks no different from that by the "cutting" feature. The advantage of this "cutting" feature lies on the flexibility and simplicity of the commands issued by the user.

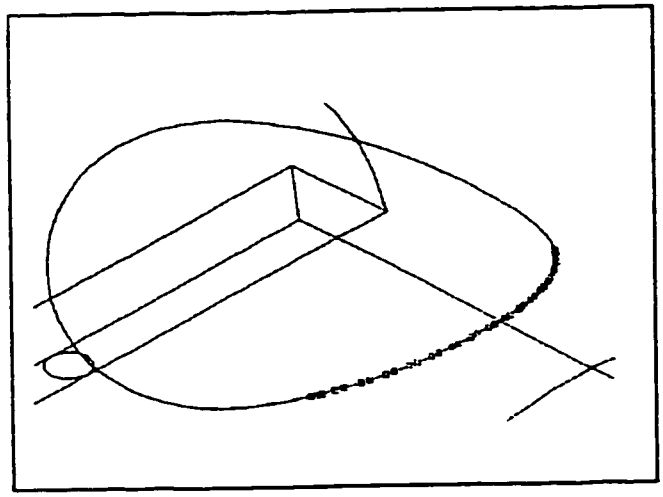

Fig. 3.3 A Part of a Spline Curve

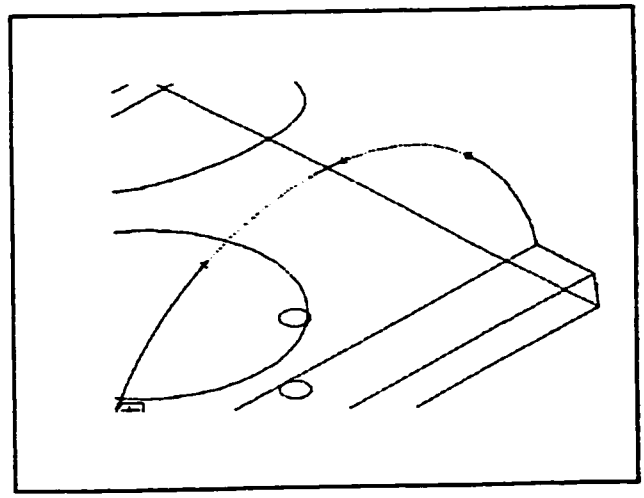

Fig. 3.4 A Part of an Arc

The flowchart that shows the implementation in AutoLISP of the "Cutting" feature is presented in Figure 3.5.

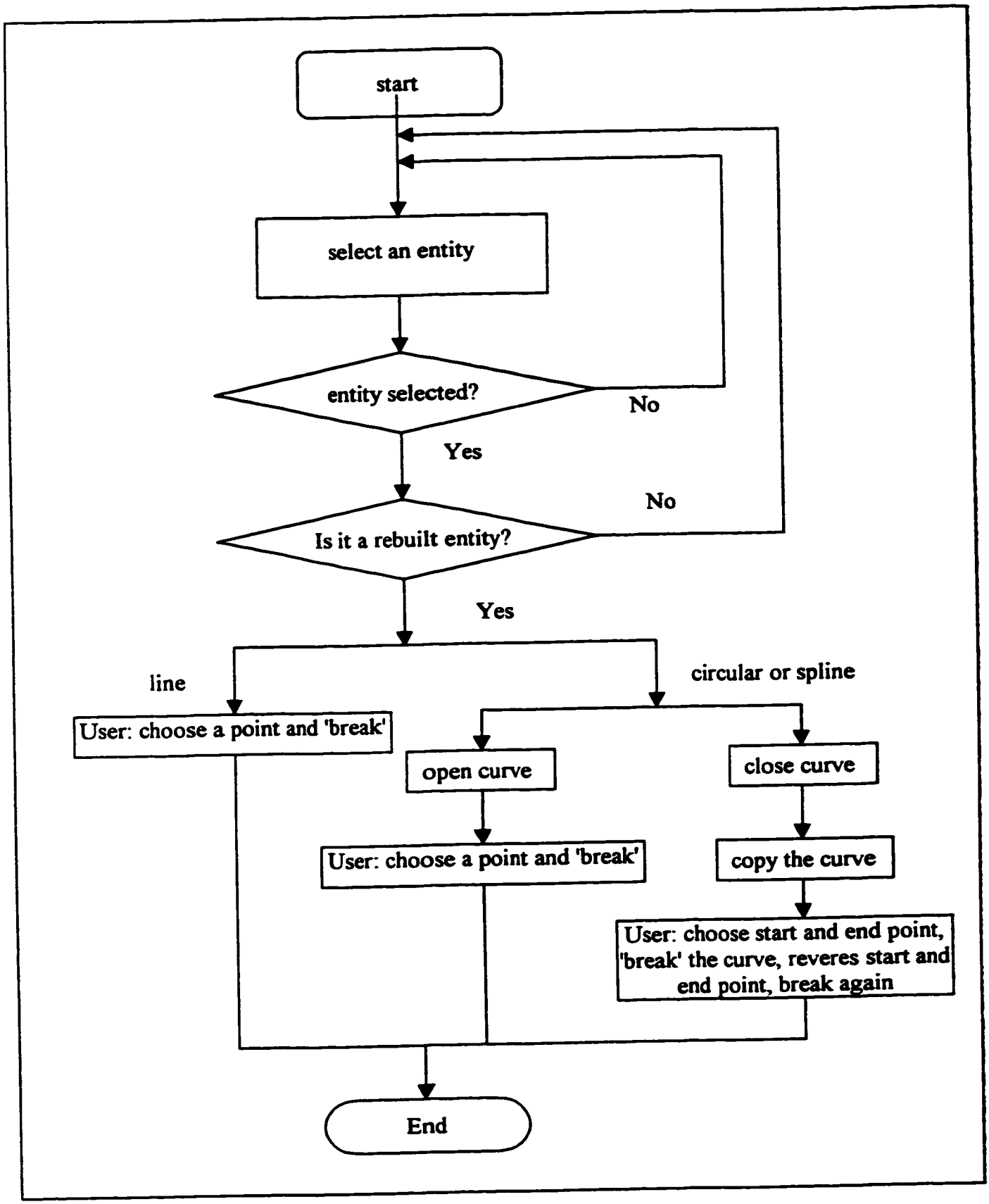

Fig. 3.5 Flowchart of the Cutting program (in AutoLISP)

3.3.3 Tool Path Generation

The tool path is generated for the Yamaha Zeta-1 Deburring Robot to move the tool along a pre-determined path. By feeding the coordinates of two set points to the controller workcell of the robot, the workcell is able to move the tool tip along a straight path, passing through a number of points calculated by linear interpolation. The linear interpolation that is used for the Yamaha Zeta-1 Robot runs in 3D space, where the combined motion of the five axes is required. However, in this work, only the generation of the desired path of the tool is discussed and is verified by experiment. Once the desired TPD is generated, the movement of the Yamaha Robot is then controlled by the workcell, executing the data by linear interpolation. The error between the desired path and the linear interpolated path by the Yamaha Robot Zeta-1 has been discussed by Ayyadevara [7] and is not included in the scope of this work.

3.3.3.1 Local Coordinate System

The tool path generation includes not only the tool position but also the tool direction. Dealing with edges, the position of the tool tip is the coordinates of the points on the desired geometry component, assuming zero machining tool diameter offset for simplicity. For polishing the surfaces, the tool is required to follow the desired polishing direction along the surfaces. The customer or user will initially define the tool angel based on their own request. Criteria have been established for deburring and polishing processes. In most cases, the tool is designed to be normal to the edge or the surface to be machined. And the user may somehow want to modify the tool angle to improve the finishing quality, based on these normal directions. Instead of setting the normal direction

as a default tool posture, the normal vector is designated as one of the coordinate axes such that this CAM package could provide more flexibility for the user to determine the tool posture easily. The details will be stipulated later in this section.

Once the required tool angle is given, the generated tool path must have the following property: The tool angles at different set point on the desired path should be identical, from the geometrical component point of view, no matter how the tool position changes. This is quite straightforward when dealing with a straight edge. Once the tool angle is defined, the tool path is simply a set of parallel lines along the edges. However, for circular edges and spline curves, a standard must be established for the tool angle to be defined. To illustrate this, assume a circular edge is to be processed. The tool trajectory will be a circular path but the tool postures would not be parallel lines any more. At each set point, whose coordinates are fed to the robot, the angle of the tool must be given as per the original user-defined tool posture. For example, suppose the angle is defined such that the tool is 30 degree to the circular plane and perpendicular to the tangential direction. Then, at every point on the circle, the tool angle will satisfy this condition.

Now it is necessary to establish a local coordinates system so that at each point on a certain component, this coordinates system looks "identical" from that component's point of view. Therefore, the following local coordinate systems are defined for different geometry components.

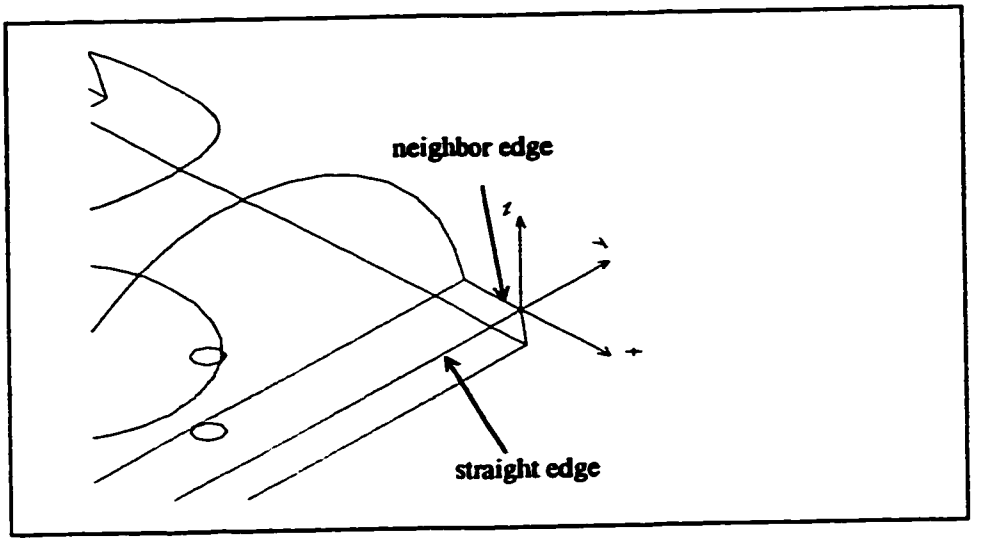

Fig. 3.6 Local Coordinate System for a Straight Edge

1) Local Coordinate System (LCS) for Straight Edge: Y-axis is defined by the two end points starting from the one that the user defines. It is also necessary for the user to select a neighbour edge so that the neighbour edge and the edge itself will form a plane. The Z-axis is defined as the normal direction of the plane. Applying the right hand rule, the direction of the X-axis can then be deduced.

2) Circular Edge: Circular edge includes arc edge and edge of circles. Since the circular edge forms a plane itself, the Z-axis is defined as the normal direction of the plane. The X-axis is defined as the tangential direction at each set point on the edge. The Y-axis can then be deduced by right hand rule.

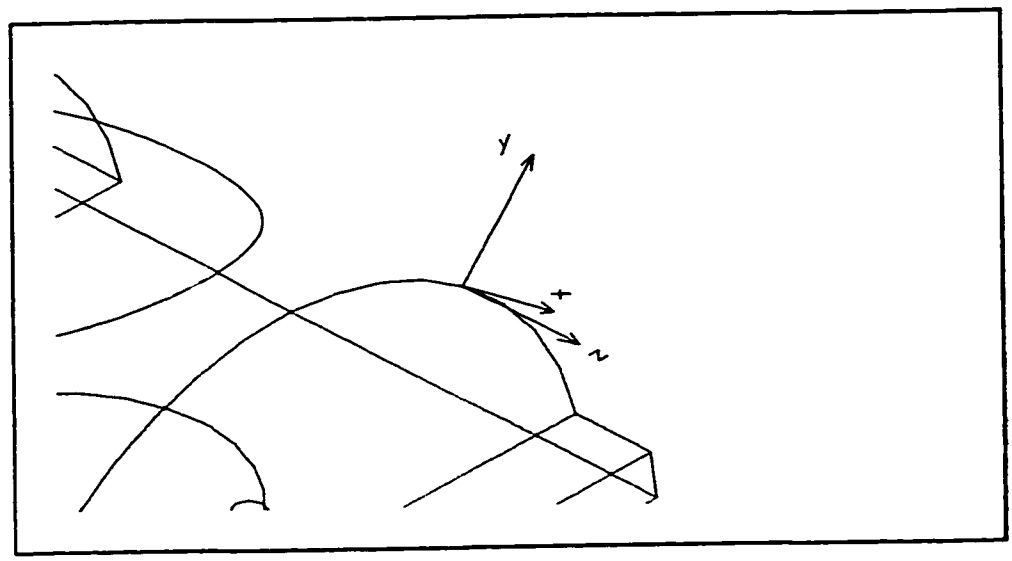

Fig. 3.7 Local Coordinate System for a Circular Edge

3) Spline Edge: Figure 3.8 shows the local coordinate system established for a spline curve at two different points on the curve. With reference to Figure 3.9, similar to the circular edge, the X-axis direction is defined as the tangential direction at each point. The Z-axis is defined as the normal of the tangential circle at fitting point A, while one neighbour fitting point B and the tangential line form a plane that the tangential circle of the spline at point A lies on. The Z-axis is usually called the minor normal or bi-normal of the spline, while the major normal or the principle normal is the one that lies within the circular plane and perpendicular to the tangential line of the spline curve at that point. The major normal is then defined as the Y-axis and its direction should follow the right hand rule.

One can see that, 1) and 2) are special cases of 3). Given certain limits in 3), the establishment of the LCS for a straight edge and a circular edge could be deduced. However, in this CAM package, straight edges and circular edges are treated as two other

kinds of geometric entities than the edge of spline curves because of the geometric complexity involved in the latter.

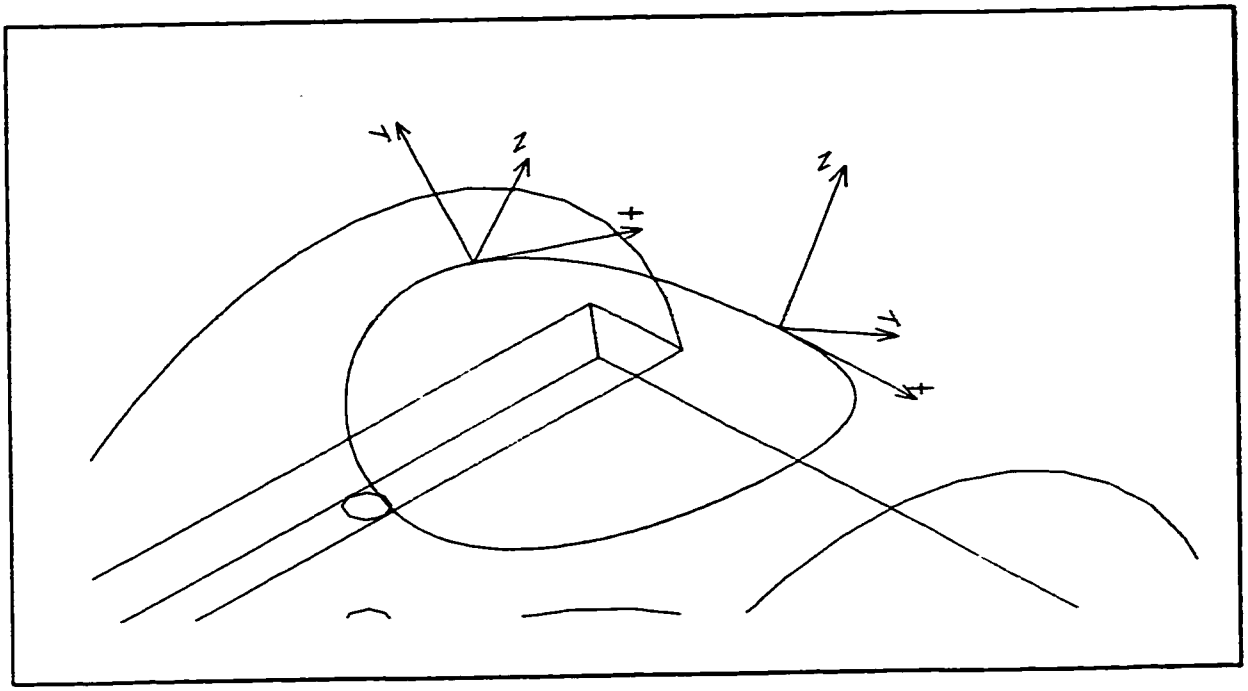

Fig. 3.8 Local Coordinate System for a Spline Curve at Two Fitting Points

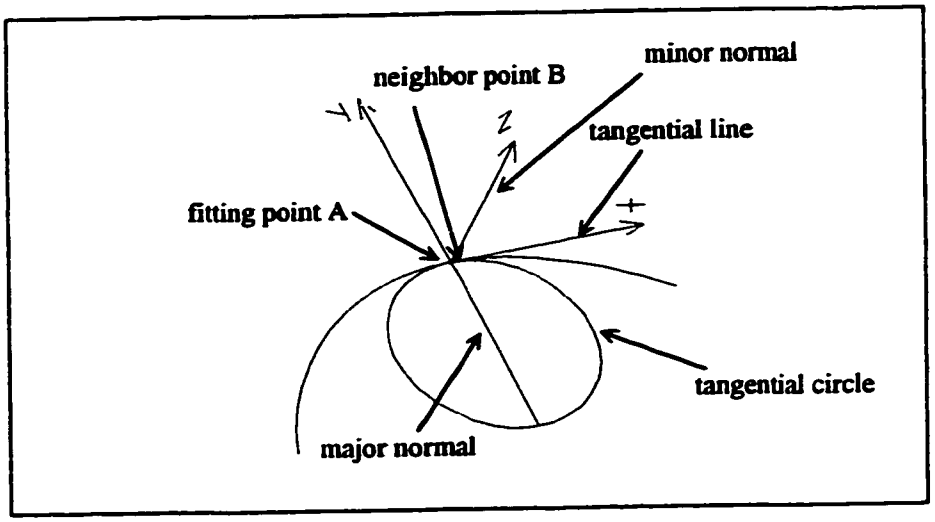

Fig. 3.9 Establishment of the LCS for Spline Curve

Establishment of the coordinate system is embedded in the tool path generation program and will be initiated immediately after a certain edge is selected so that the tool path can be generated readily. Figures 3.6 to 3.8 show the definitions of LCS for three different kinds of entity.

3.3.3.2 Homogenous Transformation from LCS to WCS

The tool path could be generated in various ways, such as using the parametric method and cutting surface method. Since the AutoCAD database is available, the trace of the tool tip is simply the curve that is to be machined. (Note that the tool diameter offset is not considered here.) Taking advantage of the AutoCAD database, the tangential direction, minor normal and major normal direction of each point on the curve could be easily found and are taken as the local x, y, z axes respectively. By establishing such a LCS at all selected machining points, only at one point should the user determine the direction of the tool in the coordinate system and this direction is then kept "identical" for all the other points. Knowing the coordinates of the machining points and each vector of the local x, y, z, the tool vector is then calculated by a homogenous transformation.

Figure 3.10 shows the homogenous transformation for the tool vector $\boldsymbol{BP}$. Local coordinate system is given by {B} and the WCS is represented as {A} as shown. The objective is to calculate the vector $\boldsymbol{BP}$ in {A}, knowing the tool vector defined in {B}, ${}^{B}\boldsymbol{P}$, and the position and orientation of frame {B} with respect to {A}, $\boldsymbol{X_B}$, $\boldsymbol{Y_B}$, $\boldsymbol{Z_B}$.

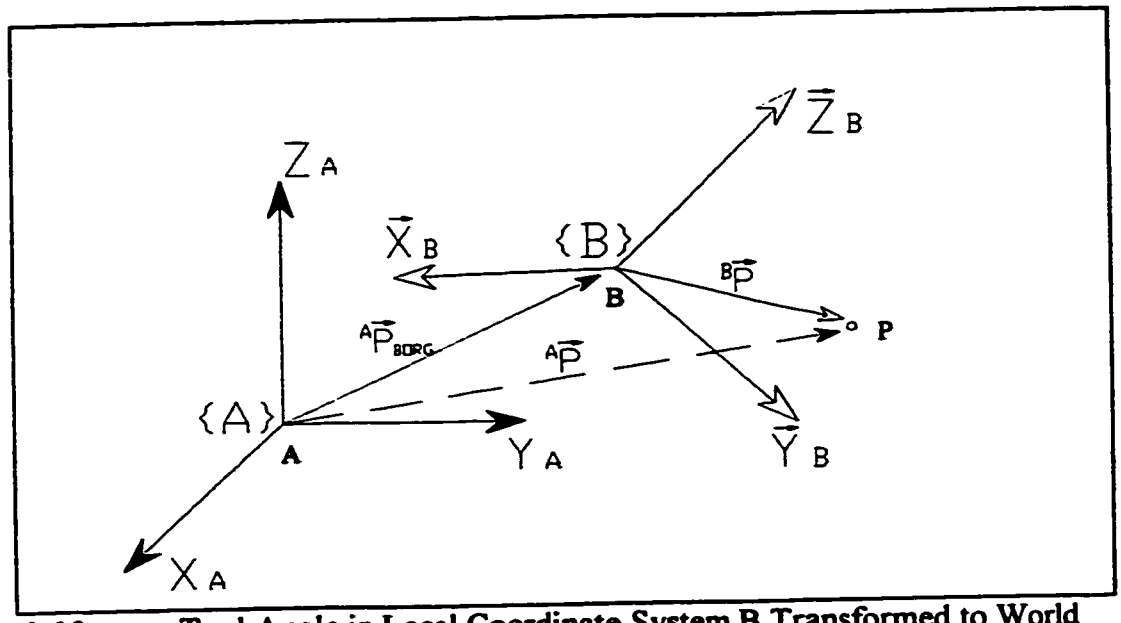

Fig.3.10 Tool Angle in Local Coordinate System B Transformed to World Coordinate System A [27]

The tool end vector ${}^{A}\vec{P}$ is calculated as follows [25]:

$$\begin{bmatrix} {}^{A}\vec{P} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{A}_{B}R & {}^{A}\vec{P}_{BORG} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{B}\vec{P} \\ 1 \end{bmatrix} \tag{3.1}$$

where ${}^{A}P_{BORG}$ represents the position of frame {B} with respect to {A}, and the rotation transformation matrix

$${}^{A}_{B}R = \begin{bmatrix} {}^{A}\vec{X}_{B} & {}^{A}\vec{Y}_{B} & {}^{A}\vec{Z}_{B} \end{bmatrix} \tag{3.2}$$

represents the orientation of frame {B} with respect to {A}.

Then, the tool vector can be calculated by

$${}^{A}\vec{BP} = {}^{A}\vec{P} - {}^{A}\vec{P}_{BORG} \tag{3.3}$$

The tool path is calculated by using Equation (3.1) to (3.3) at every machining point. It should be remembered that the 4x4 matrix in Equation (3.1) has been determined during the establishment of the LCS as described in Section 3.3.3.1.

The most common edges that are to be machined are straight, circular and intersection curves. As to the first two types of edges, determining the tool angle is quite straightforward. As to the third type, except for those straight, circular and elliptical intersection curves, AutoCAD constructs all the other types of intersection curves using a 4^{th} order spline by default. And the AutoCAD database always provides its spline function results, which consists of coordinates at every point on the curve. By choice of the user, the spline curve could be modified to a higher order.

Once the tool path generation feature is initiated, the LCS is immediately established at a default point on the curve and a default tool angle is given. The user is allowed to determine the tool angle at the default starting point by inputting a vector in the LCS, or can be modified interactively by rotating the tool to a desired position. The tool path is then calculated and drawn on the screen. Figure 3.11 illustrates a tool rotating in the LCS.

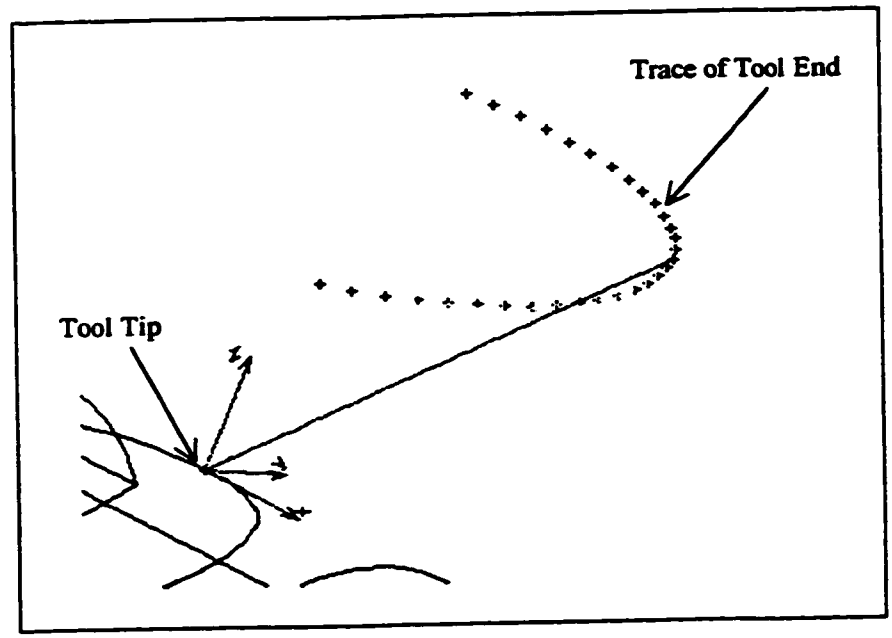

Fig. 3.11 Tool Vector Rotating in the LCS

Tool path generation for deburring straight edges, circular edges and spline edges, and polishing planes and cylindrical surfaces are respectively designed and similarly programmed by following the above-described algorithm. The generated tool path is saved in ASCII code as off-line data to be fed to the robot later on.

3.3.3.3 Tool Path for Straight Edges

For straight edges, it is not necessary to break the edge in advance. Before the tool path is generated for a designated straight edge, an option is provided to the user. When a straight line is selected as the desired edge, two end points of the selected edge will be provided as default processing end points. Different end points can be defined at this time by inputting coordinates of start and end points. The user can also "break" this edge in advance to specify the portion of the edge to be machined.

Then, within a LCS described in Section 3.3.3.1, a default tool shows up with a default tool angle. Users are able to use the number pad on the keyboard to rotate the tool about the local X, Y, Z axes. Also, users are allowed to use a vector in the LCS to define the tool angle. Once the tool angle is set, the tool path will be generated and written into a text file. Users have a final option to view the tool path on the screen.

To trace a straight-line path using the Yamaha Zeta-1 Robot, the coordinates of the starting point and end point need to be specified. Linear interpolation is carried out to calculate the intermediate points.

In the case of straight edges, the resulting errors due to this process depend on the linear interpolation carried out by the robot controller only, provided that the coordinates

of the two end points that are provided by AutoCAD are accurate, and the robot is accurate enough to reach these end points precisely.

Figure 3.12 shows the flow chart for straight edge tool path generation. Not only does the chart present the programming steps, but also it indicates the operating procedures of this feature. The tool path generated for a portion of a straight edge is illustrated in Figure 3.13.

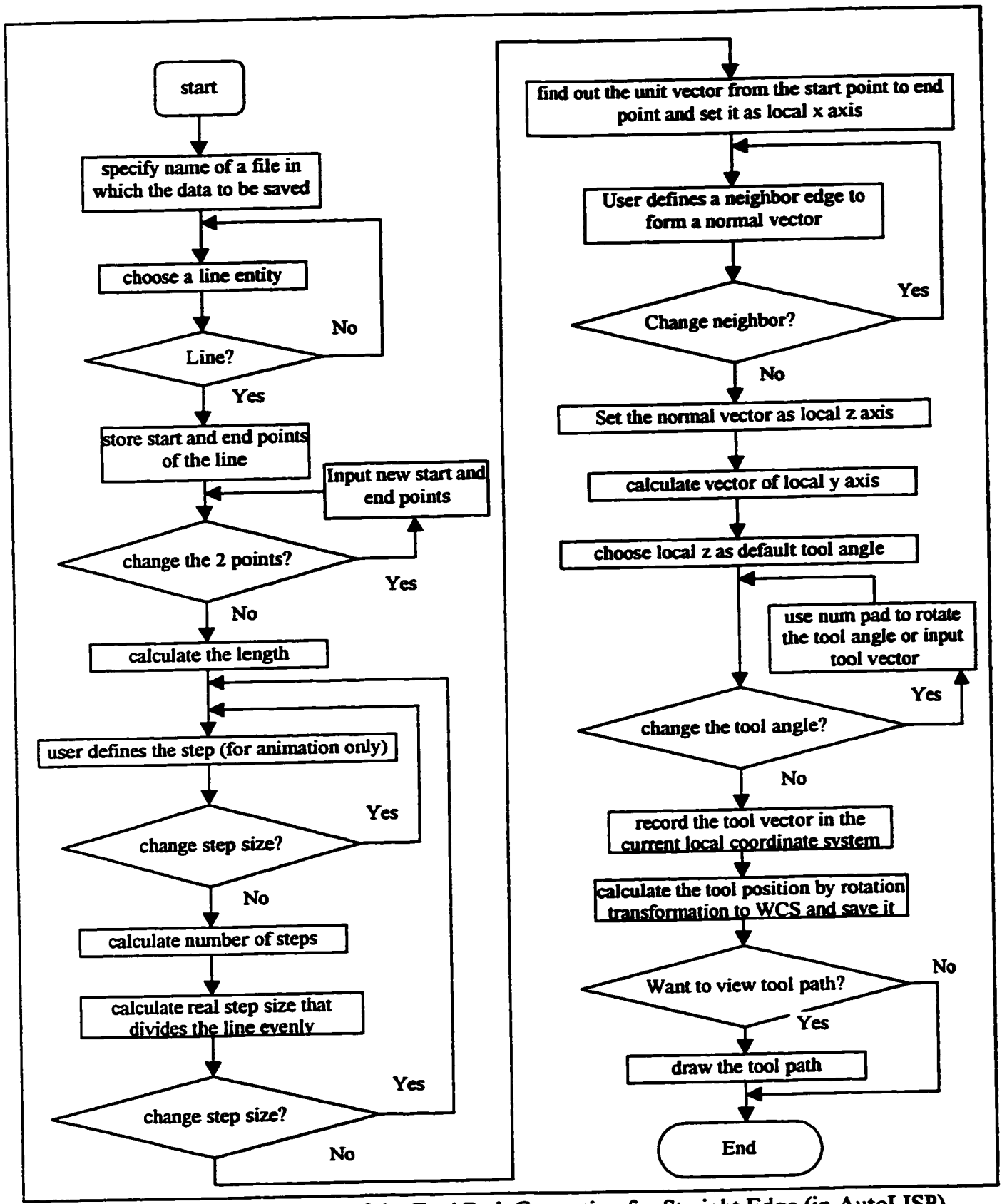

Fig. 3.12 Flowchart of the Tool Path Generation for Straight Edge (in AutoLISP)

3.3.3.4 Tool Path for Circular Edges

There are slight differences between the tool path generated for straight edges and circular edges. Once the circular tool path generation is initiated, by default, the starting point and end point are chosen upon the database of the selected entity. Comparing to the command for straight edges, the option of choosing end points is not allowed here because usually it is hard for the user to specify a precise point on a circular curve. If only one part of the edge is desired, users have to break it before initiating this function, using either the AutoCAD's "_break" or edge cutting mentioned in section 3.3.2.

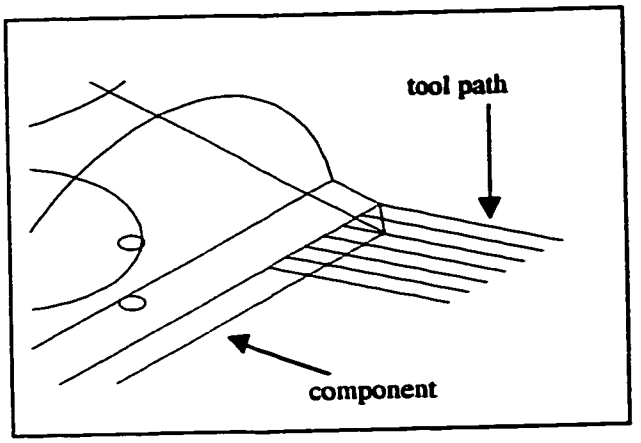

Fig. 3.13 Tool Path for Straight Edge

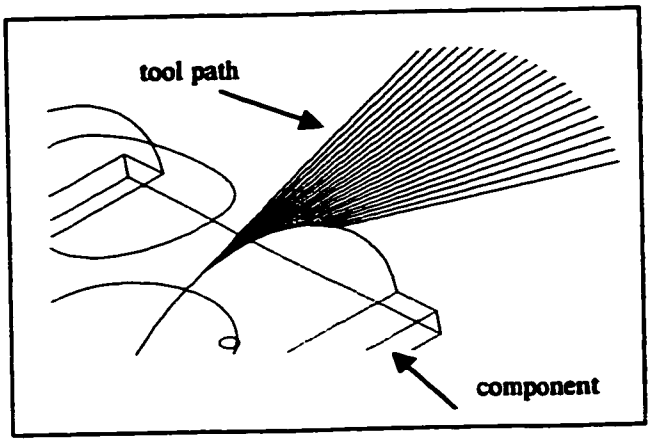

Fig. 3.14 Tool Path for Circular Edge

The LCS for a circular edge is different from that of a straight edge. It is necessary to specify the error limits that can be tolerated without specifying the machining accuracy. This is necessary as the robot is moved along a straight segment (resulting from linear interpolation) between adjacent set points thereby approximating the curved path. This error determines how many set points or sets of data will be calculated. The smaller the error, the more sets of data. There is another error produced by the linear interpolation of the robot controller when real cutting is being done.

Therefore, the resulting error would be the later one superimposed on the former one. A tool angle defined similarly to that in the straight edge processing is also provided. A schematic for a circular tool path is illustrated in Figure 3.14 and a flowchart for circular edge processing is presented in Figure 3.15.

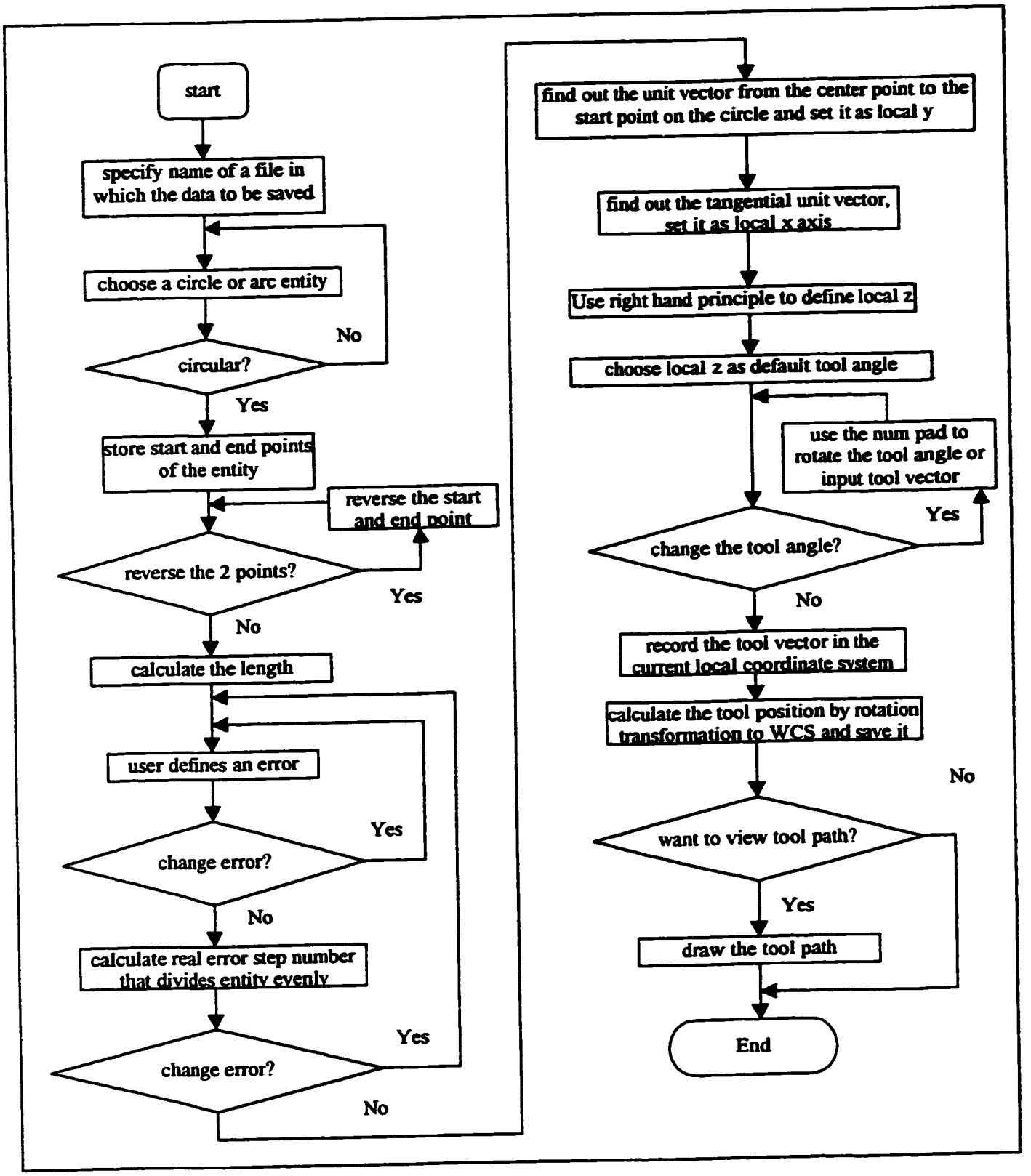

Fig. 3.15 Flowchart of the Tool Path Generation Utility for Circular Edge

3.3.3.5 Tool Path for Intersection Edges

Two types of spline curves are constructed in AutoCAD: open spline and closed spline. As mentioned before, database of spline curves remain untouched in the re-construction process and AutoCAD itself constructs the spline curves when the 3D model is being built. Using AutoCAD's command "divide" divides the spline curve by an integer number n. The division does not break down the curve into pieces but draws n points on the curve. Those n points are the fitting points or vertices on the curve and they divide the curve in equal 3D lengths. By this approach, every fitting point on the spline curve could be determined by dividing the curve with a large number. However, the larger the number n, the longer the computing time. If the two adjacent points are close enough (depending on the required tolerance), the tangential line and the tangential circle at a certain fitting point could be approximated. Once the tangential circle is found, a LCS is then established by assigning the tangential vector along the x-axis, major normal vector along the y-axis, and minor normal vector along the z-axis. Figure 3.16 shows the points that divides a spline curve by 100. The original curve is shown on the left side of the figure.

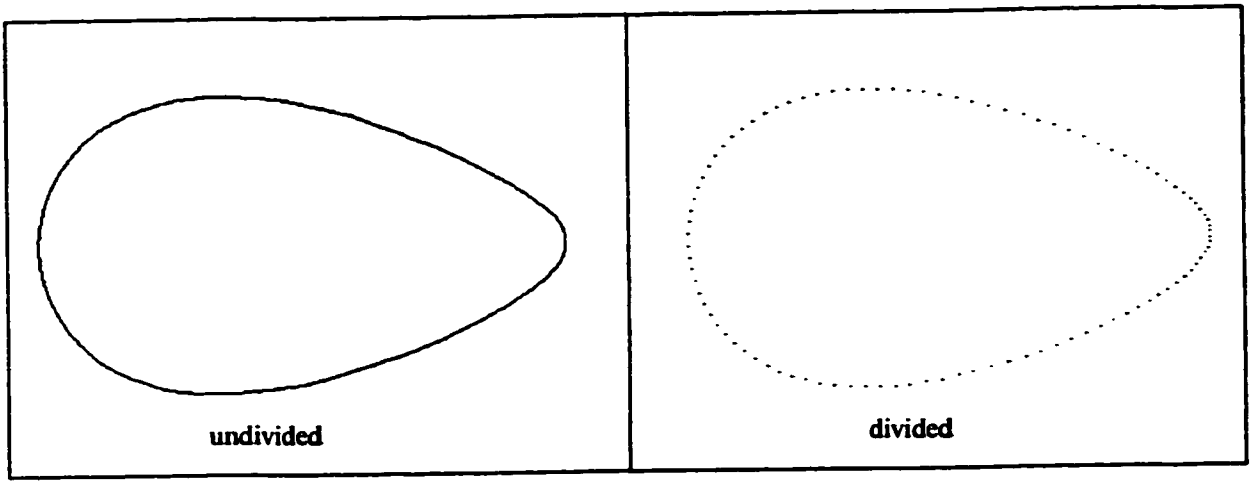

Fig. 3.16 Divide a Spline Curve by 100 in AutoCAD

An alternative approach is employed to improve the accuracy of the tangential vector and the tangential circle. Based on the conclusion drawn in Section 3.2.1 that the database of an open spline curve provides two fitting points, which are the starting and the ending points, and the first and last two control points form a tangential line at those points, "breaking" the spline curve into a number of pieces can also give every fitting point on the curve. As well, the tangential lines on those fitting point are also found. The tangential line at one fitting point on the curve and one neighbor fitting point form a plane where the tangential circle lies on. The closer the two points, the more precise is the location of the tangential circle. Taking the tangential line as the local x-axis and the vector perpendicular to the plane, which is formed by the neighbor fitting point and the tangential line, as the z-axis, a LCS is then built. Figure 3.9 shows the establishment of the LCS on an open spline curve. Establishing such a LCS at every fitting point and defining a tool vector within it, knowing the vectors of each coordinate axis in WCS, the tool position and tool angle could be generated by a homogenous transformation.

There are two kinds of errors that are involved in this approach. First, given two adjacent fitting points, the robot is supposed to move the tool along a straight line. The maximum error between this straight line and the spline curve is still unknown. Secondly, since the normal vector of the plane formed by the tangential line and a neighbor fitting point approximates the local z-axis, this approximation could affect the accuracy of the tool angle. In other words, the question is, at one fitting point, how close should the neighbor fitting point be chosen such that the local z-axis is close enough to the bi-normal? Actually, this error results when the LCS is being established for the spline

curve. Taking advantage of AutoCAD, this question can be answered by using some AutoCAD techniques.

Consider a spline curve as shown in Figure 3.16. It is divided into 100 segments by 100 fitting points. Taking one segment into consideration, as shown in Figure 3.17, it is equally divided again into a number of sections. Note the term "divide" refers to AutoCAD's command "divide" and after such a "divide" process, AutoCAD draws the dividing points in the 3D space. From those dividing points, lines are drawn perpendicular to a straight line, which has been constructed by connecting the two end points of the segment.

The maximum length of those perpendicular lines represents the maximum error between the spline curve and the tool path. The lengths of the errors could be read from the databases of those lines by simply inputting an AutoCAD command "list". Usually this process needs to be done several times at those fitting points where the largest curvature occurs on the curve. If the user is not satisfied with the maximum error, he/she may want to divide the whole spline curve again by a larger number and then redo the error lines. This is a trial-and-error process and it needs to be done before the tool path is generated so that the user could have a segment number to divide the spline and get the number of fitting points.

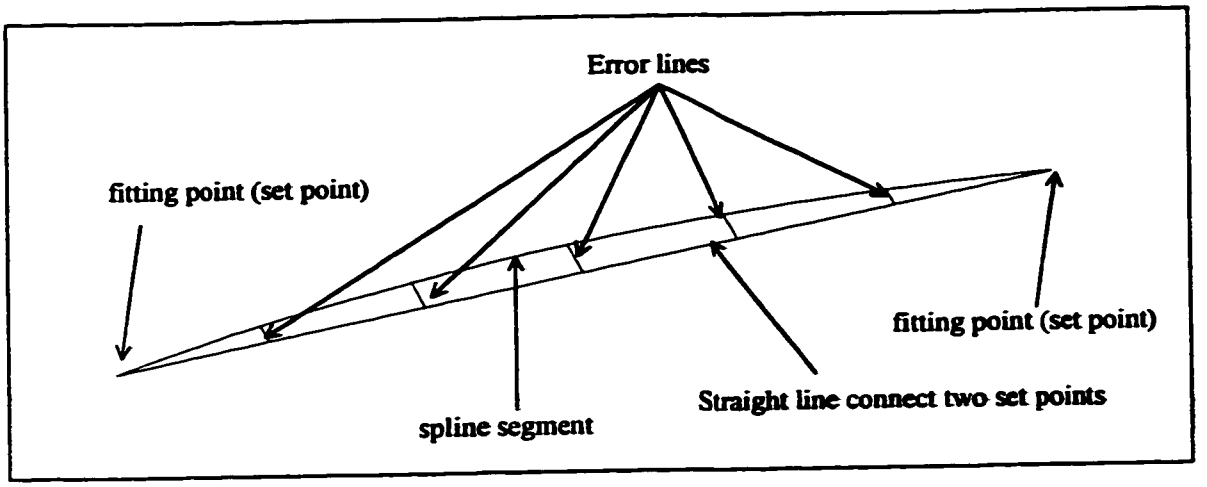

Fig. 3.17 Error Lines between the Spline and the Tool Path

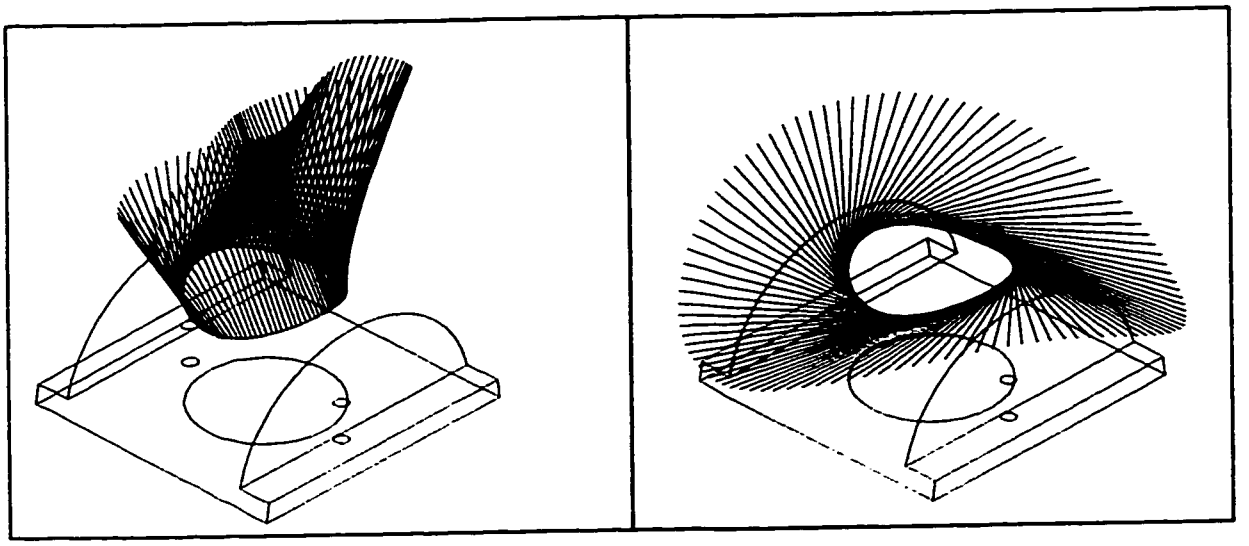

Fig. 3.18 Tool Path Generated for Spline Curve

Figure 3.18 shows the tool path generated by defining the angle of the tool as the minor normal direction (left) and as the tangential direction (right). Similar to the circular edge, coordinates of a number of points along the path are provided to the robot and the robot will move along these points in the 3D space by linear interpolation. The error due

to the linear interpolation will superimpose on the tool position error that comes from the tool path generation.

Since no spline equation is employed in the tool path generation process, AutoCAD techniques are needed to find the error of the LCS Z-axis, which affects the accuracy of the tool angle. Dividing the existing spline by a very large number, say 10,000, which is much larger than the segment number figured out from error analysis before, the resulted tool path could be far more precise than the one by dividing the curve by 100 in the sense of tool orientation. Comparing the tool angles generated at the same fitting point, one can read the angle difference between the two sets of tool path by different dividing numbers. Figure 3.19 shows the angular difference between two tool angles generated by dividing the whole spline curve into 10 segments and 10,000 segments. The actual angle is measured as 3.1675°. This angular difference decreases drastically when the dividing number increases. For example, the angle difference between tool angles by dividing number 100 and 10,000 is as small as 0.105°. This small angular difference could be observed by AutoCAD's zooming.

After such a comparison, the user could also figure out a dividing number that would be utilized to generate the tool path. Although the actual angle error is not possible to evaluate by this approach, one can always increase the dividing number to decrease the angle error so that a certain error requirement is being met.

As a conclusion, two dividing numbers are found for the requirement of two errors: one from the error lines and one from the tool orientation. The larger number is to be chosen to generate that amount of tool positions and the tool angles at those positions.

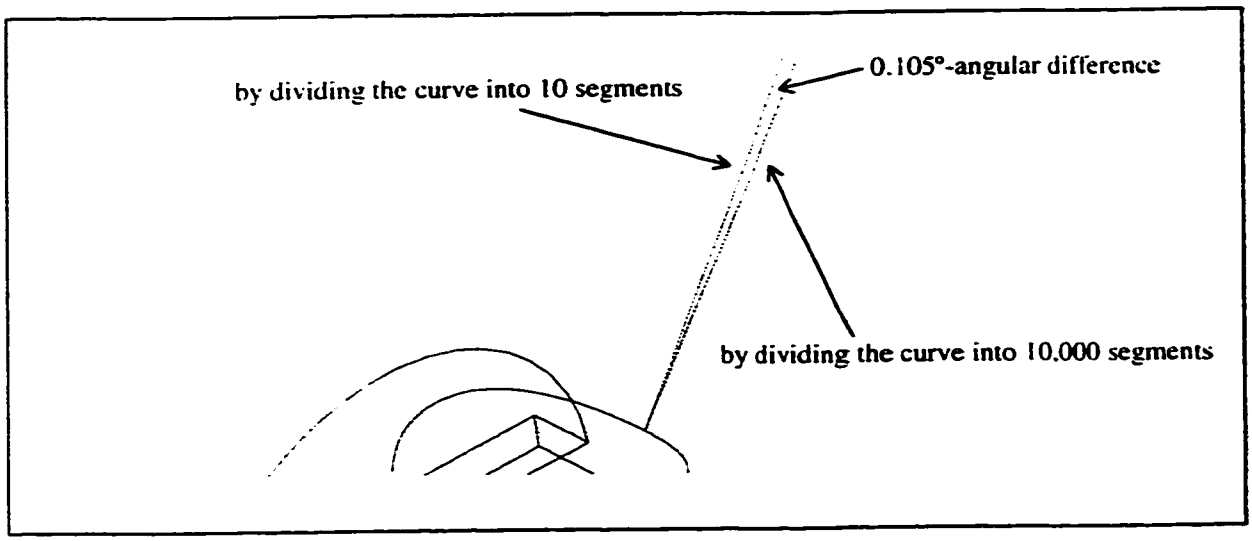

Fig. 3.19 Tool Angle Difference

These two error-finding processes are not incorporated into the tool path generation program because of the complexity of the processes. One may find that it is much more flexible to do this error-finding by AutoCAD commands instead of an integrated command because it involves a lot redoes and undoes. Figure 3.20 is the flowchart of the program that generates the tool path for spline curve.

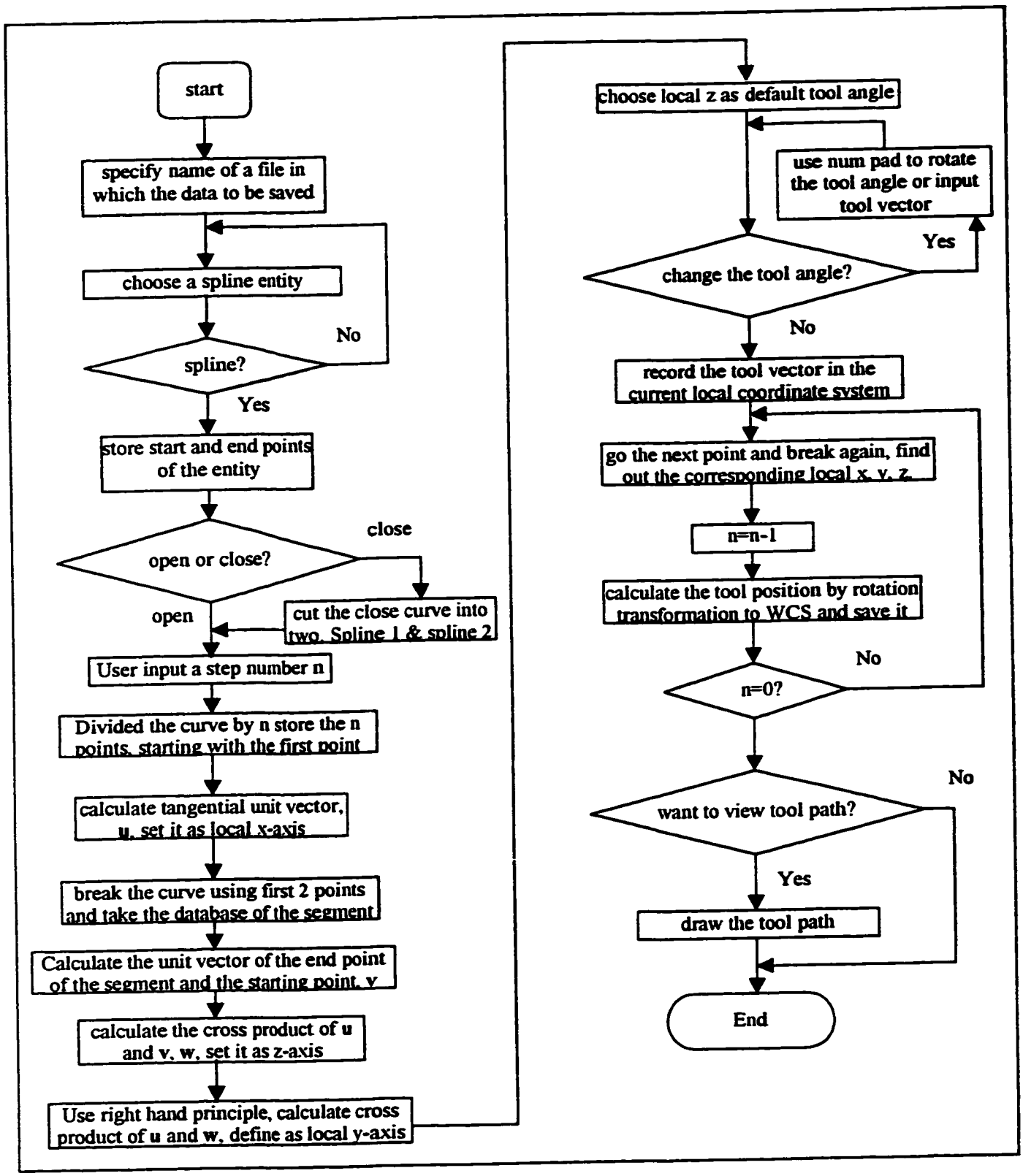

Fig. 3.20 Flowchart of Tool Path Generation for Intersection Edge (in AutoLISP)

3.3.3.6 Tool Path for Plane Surface

Polishing a plane surface is, basically, a repetitive process of deburring a straight edge in the sense of tool path generation, although it does not hold true in polishing complex surfaces. 3D solid models of workpieces are also rebuilt into wireframe models by going through the filtering, double exploding, and reconstructing processes as described earlier in this chapter. Therefore, the polishing feature deals with the same reconstructed 3D models as in deburring. A LCS similar to the one shown in Figure 3.6 is established. Figure 3.21 shows a workpiece that is used to illustrate the plane-polishing feature. On one plane surface of the workpiece, a LCS is established and is shown in Figure 3.22.

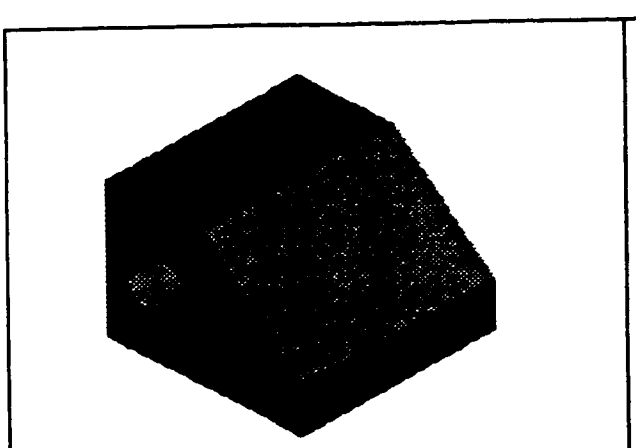

Fig. 3.21 A Workpiece for Polishing

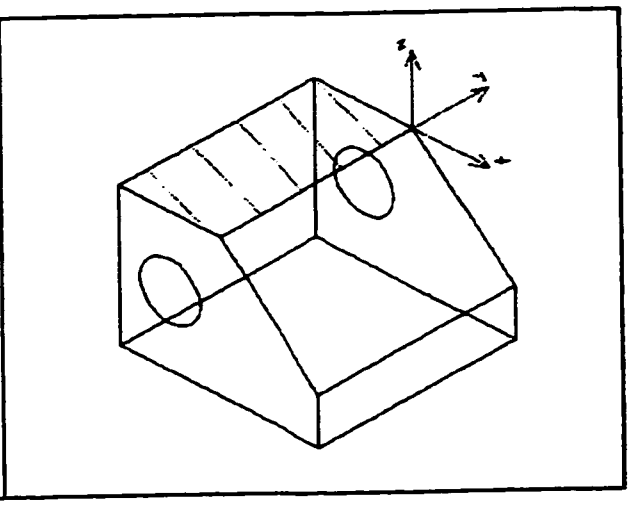

Fig. 3.22 LCS for a Plane Surface

In this feature, tool path is generated based on some conditions given by the user. First, the user has to define a starting edge and the starting point on the edge. Also, the step length between two polishing lines has to be specified. This length determines how many passes the polishing tool should process on the plane. Afterwards, similar to the

tool angle changing feature, a line is shown up within the plane representing the polishing direction. It could be rotated in this plane or changed by inputting a 2D vector to specify the polishing direction. Figure 3.23 shows the polishing direction changing in the plane.

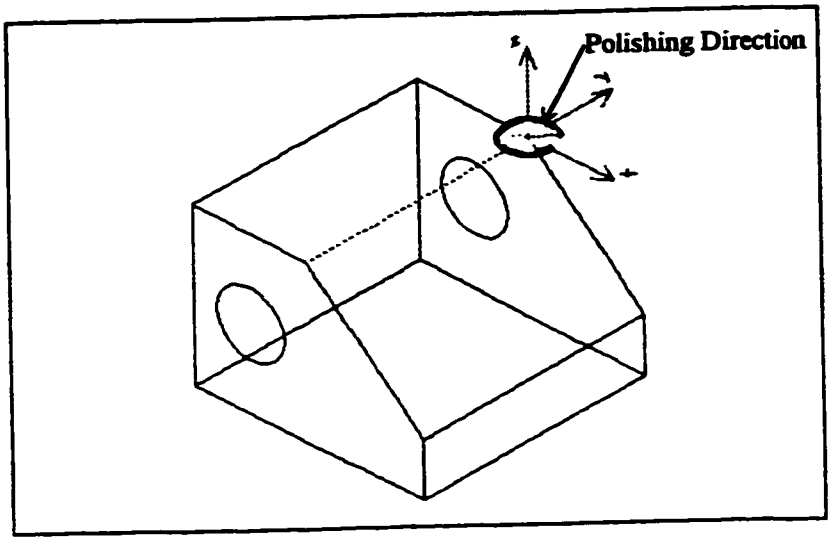

Fig. 3.23 Polishing Direction Changing in the Plane

Once the polishing direction is determined, a step-out point on the extended line of the polishing direction is calculated. The location of this point is outside of the plane defined by the selected two edges. This is a step for the polishing tool to step out of the surface in order to avoid scratching the surface during pass step-over. And the user then could customize the tool angle at the step-out point. Figure 3.24 shows the tool angle changing by the user.

Tool path generated for a plane is illustrated in Figure 3.25. The schematic chart shown in Fig. 3.26 presents the procedures to generate the polishing tool path.

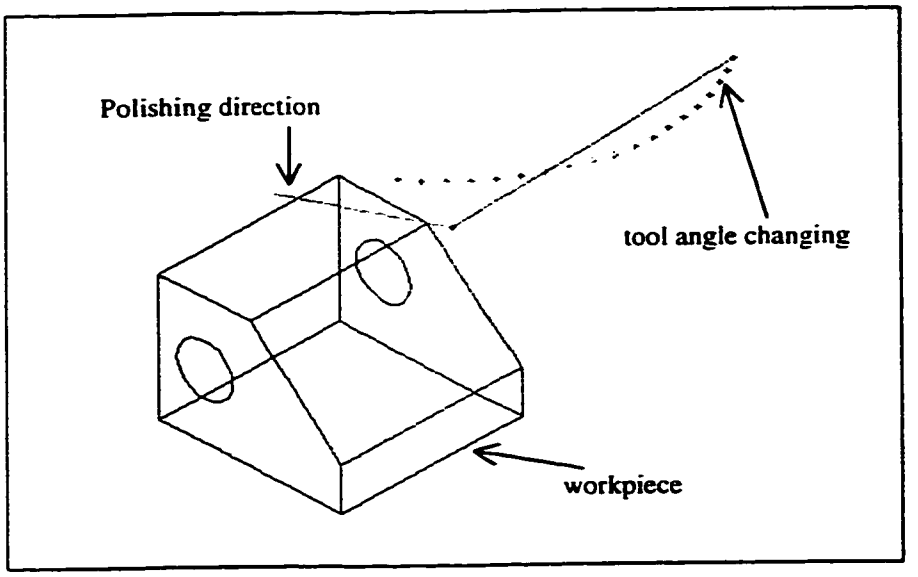

Fig. 3.24 Tool Direction Changing in the Plane

In the case of irregular plane surface where two straight edges are not easy to be specified, the user can create two straight lines within the plane upon his/her own request and then use this feature to generate the tool path.

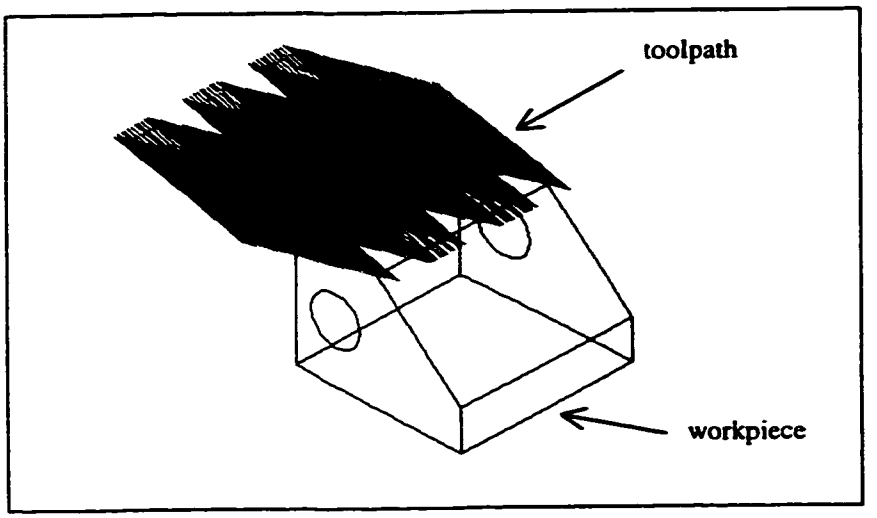

Fig. 3.25 Tool Path for Polishing a Plane Surface

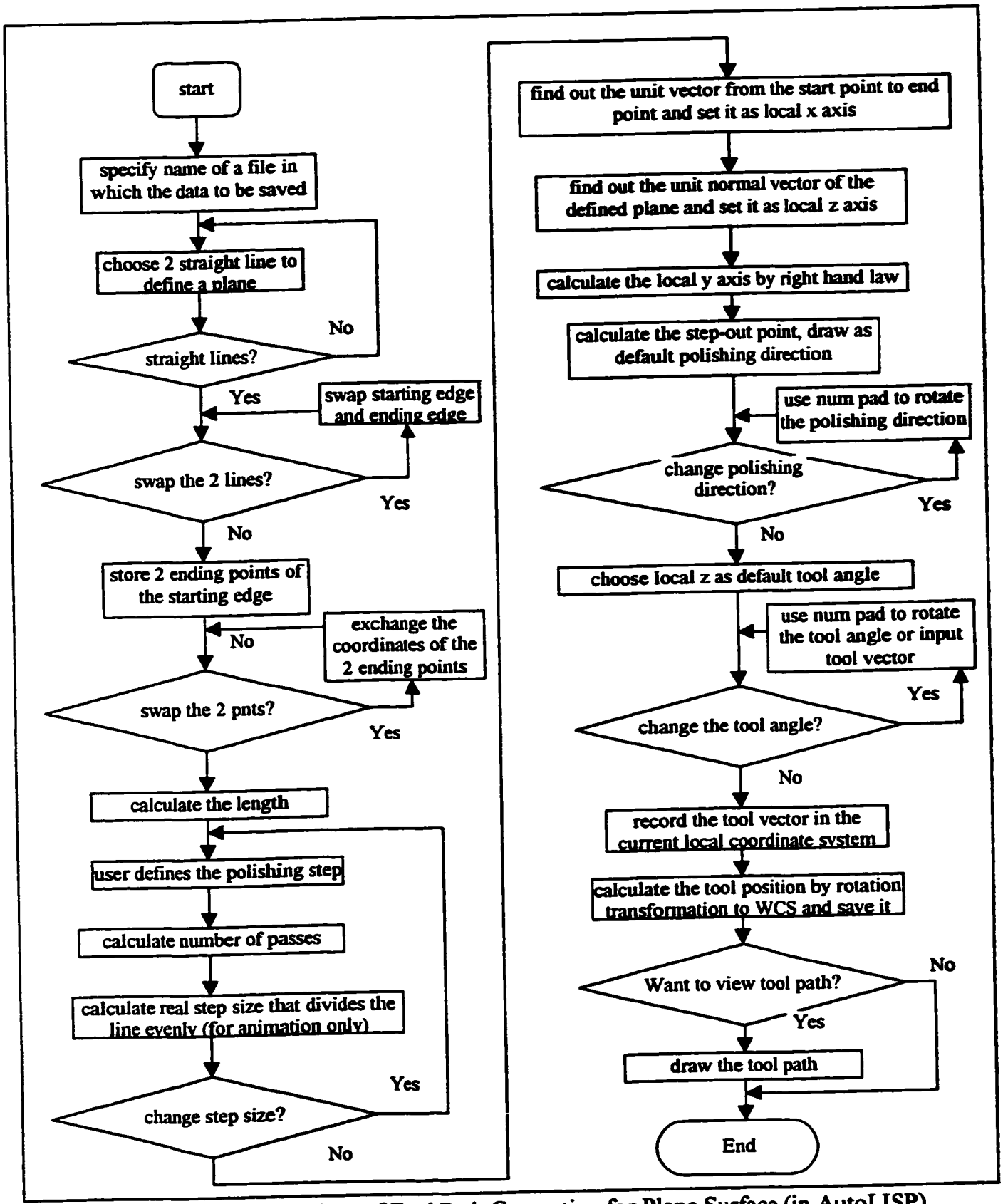

Fig. 3.26 Flowchart of Tool Path Generation for Plane Surface (in AutoLISP)

3.3.3.7 Tool Path for Cylindrical Surface

Similar to the tool path generated for circular curves, a LCS is established as the one shown in Figure 3.7. The tool path is generated along the cylindrical surface and parallel to the cylinder axis, as shown in Figure 3.27. Similar procedures as dealing with the plane surface are employed and shown in Figure 3.28.

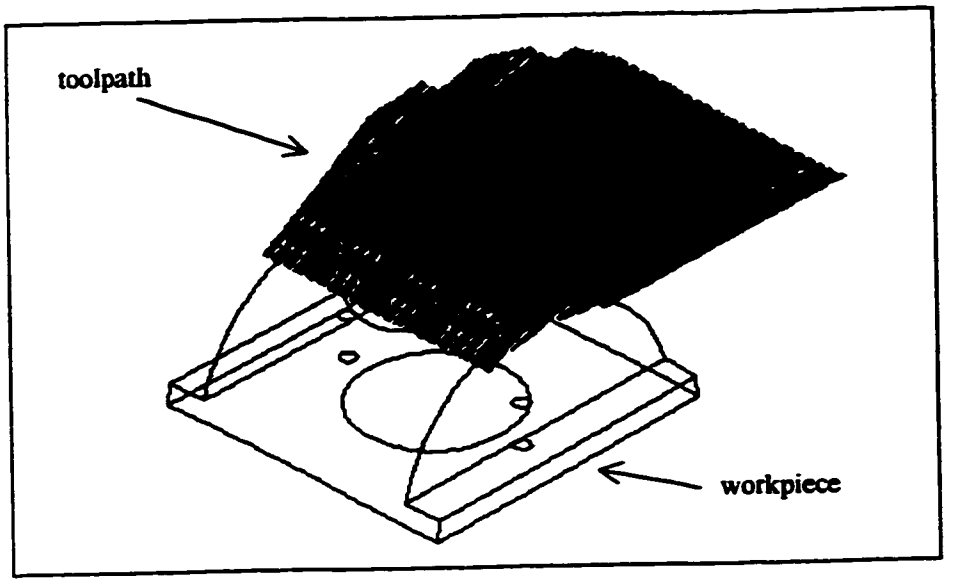

Fig. 3.27 Tool Path for Polishing a Cylindrical Surface

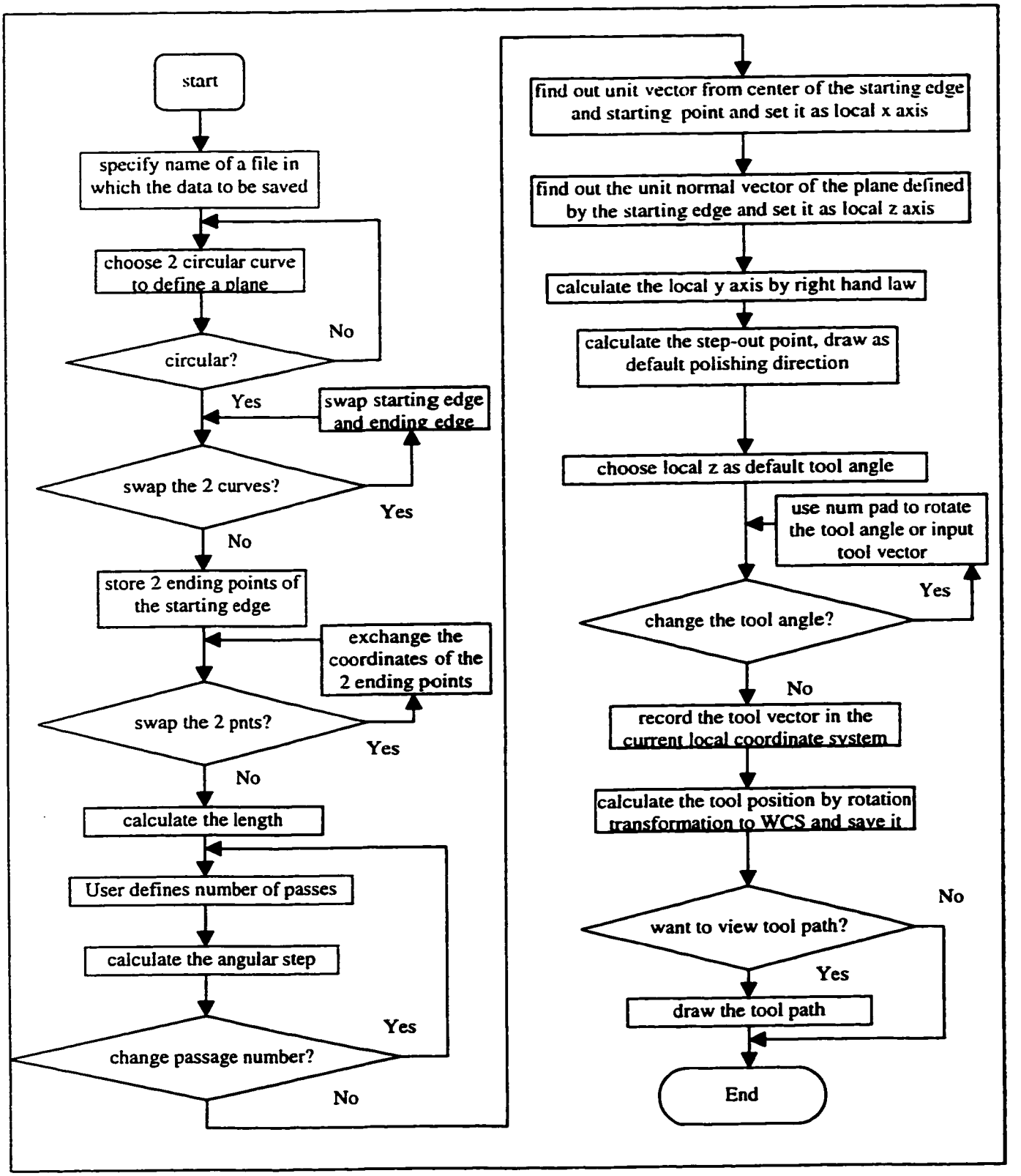

Fig. 3.28 Flowchart of Tool Path Generation for Cylindrical Surface (in AutoLISP)

3.4 Robot Coordinate System

The tool path has been generated and the coordinates of the tool tip (x_1, y_1, z_1) and tool end (x_2, y_2, z_2) have been calculated. They are absolute locations in the 3D space and they have to be formatted into the data format that the Yamaha Robot Zeta-1 needs. Table 3.3 shows the data that are fed to the Robot.

x_1 (mm)	y_1 (mm)	z_1 (mm)	A (°)	B(°)	V (mm/s)
43.9565	-85.4545	89.8211	325.064	60.4267	1.0
43.9565	-83.5654	89.8211	325.064	60.4267	1.0
43.9565	-81.5675	89.8211	325.064	60.4267	1.0
43.9565	-79.5654	89.8211	325.064	60.4267	1.0
43.9565	-77.5453	89.8211	325.064	60.4267	1.0
43.9565	-75.5754	89.8211	325.064	60.4267	1.0
43.9565	-73.5754	89.8211	325.064	60.4267	1.0
43.9565	-71.7876	89.8211	325.064	60.4267	1.0
43.9565	-69.2576	89.8211	325.064	60.4267	1.0
43.9565	-67.8455	89.8211	325.064	60.4267	1.0
43.9565	-65.8565	89.8211	325.064	60.4267	1.0
43.9565	-63.7555	89.8211	325.064	60.4267	1.0
43.9565	-61.1246	89.8211	325.064	60.4267	1.0
43.9565	-59.8665	89.8211	325.064	60.4267	1.0
43.9565	-57.8454	89.8211	325.064	60.4267	1.0
43.9565	-55.7645	89.8211	325.064	60.4267	1.0

Table 3.3 Robot Data Format

In the above table, x_1, y_1 and z_1 are the coordinates of the tool tip while A and B are two angles that defines the tool orientation. Angle B is defined as the angle between the tool and the vertical axis and it is always given as a positive number. (There are negative values of the angle B. In order to simplify the calculation of the tool orientation, B is kept to be positive.) Angle A is defined as the angle between the projection of the tool to the x-y plane and measured from the positive x-axis of the robot while B is positive. The last column of the table is the linear velocity of the tool tip, V, and it is always given a constant value 1.0 mm/s.

In order to transform the tool ends coordinates into such data format, the following calculation is incorporated into every tool path generation program.

$$A = \pi \pm \tan^{-1}\left(\frac{|y_2 - y_1|}{|x_2 - x_1|}\right) \tag{3.4}$$

$$B = \tan^{-1}\left(\frac{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}{|z_2 - z_1|}\right) \tag{3.5}$$

One can see that angle B is always positive by the calculation above. Angle A is to be determined by the location of the tool end relative to the tool tip. The data calculated by the above equations are listed in the format shown in Table 3.3 and saved as a file. By such a post-processing, the tool path is then revised into instructions for the Yamaha Robot Zeta-1. This post-processing is programmed within the tool path generation procedure so that the generated file could be directly used as command data for the robot.

3.5 Summary

The tool path generation algorithm is detailed in this chapter. All of the procedures to perform tool path generation are carried out under the AutoCAD environment. A user interface is incorporated into AutoCAD by adding customized commands, using AutoLISP as a programming language. The generated tool path is post-processed into the robot coordinate system and is to be fed to the robot. Although only regular geometry has been studied so far, new features and new commands could be added into the interface by the same programming strategy.

Chapter 4

Computer Simulation of Tool Path Execution

4.1 Introduction

Although most CAD/CAM systems can assist human operators to program tool paths for robots or NC machines, detailed geometry analysis is needed to make a "correct" decision that the tool path has been properly designed. However, the "correct" decisions are mostly made based on operator experience, which may not always be "correct" [28]. A trial cut or simulation is then a must to verify the cutter path and to correct the errors.

This CAM package provides a simulation feature that runs the machining process in a simulated, virtual reality mode, using an off-the-shelf graphical visualization package in Virtual Reality (VR). This VR environment makes simulation possible in a short lead-time.

In this VR environment, the user is able to "walk around" the machining setup, say the Yamaha Zeta-1 Robot, to inspect it virtually and visually during a machining production trial run. This simulation can also demonstrate the tool path generated by using AutoCAD database, assist jig and fixture design and setup, and enable the user to observe possible tool collisions with the workpiece and fixture [19]. It also demonstrates the potential for utilizing the increasing graphics capabilities of PC-based platforms in the field of mechanical engineering, especially in the visual display of information (generated by established simulation outputs).

4.2 Virtual Reality Tool — WorldToolKit (WTK)

The term "Virtual Reality" (VR) was initially coined by Jaron Lanier, founder of VPL Research (1989) [29]. Other related terms include 'Artificial Reality' (Myron Krueger, 1970s) [29], 'Cyberspace' (William Gibson, 1984) [29], and, more recently, 'Virtual Worlds' and 'Virtual Environments' (1990s) [29]. The basic goal of VR is to create the most realistic interaction between a human and an environment, which does not exist, or, from the environment point of view, to simulate the behavior of objects in the environment. VR creates interactive "worlds" which allow the user to explore and alter the objects within it in real-time. Then, after creating a world in which everything feels, smells, looks and sounds the same way as in real life, the programmers can change the variables, to create worlds that were unattainable before. It is the most advanced computer graphics technique and it is found to be applied in many fields. It provides the user with many tools to animate the scene and react on user input.

Although there is no unique definition for Virtual Reality at this time, basically, there are four fundamental and common concepts in Virtual Reality. They are:

1) Viewpoint - the calculations that the computer must make in order to determine where exactly in space the simulated person (the user) is at, where he/she is looking, and the proportion of all the simulated objects around that user, based on distance, height, and angle from his/her Point Of View (POV). These calculations are then used to "draw" graphics on the screen.

2) Navigation - the calculations that the computer must make, in real time, in order for someone (the user) to move about the simulated environment that he/she find him/herself in.

3) Manipulation - a process by which a person immersed in a virtual world can interact with the imaginary, or simulated objects around him.

4) Immersion - the quintessential ingredient of virtual reality. To achieve the illusion of immersion in the computer-generated environment requires some method of putting pictures in front of your eyes so that your total view, no matter which direction you turn, encompasses only the virtual world.

A virtual reality development toolbox, WorldToolKit (WTK) Release 6 [30], developed by Sense8 Corporation, helps to provide a virtual 3D environment in which the robot model is built. Such an environment allows the user to "walk through", "walk around" and "look" around the machining scene and examine the machining process.

Written in C++ language, WTK is basically a function base or subroutine base in which contains a large number of function calls for programmers to use. And those function calls are to be used in order to create a virtual reality environment, load in 2D or 3D models, and make them move. The advantage of such a "tool box" is to save a lot of programming on the calculations of screen updating, viewpoint switching, and environment changing. Figure 4.1 shows a virtual environment in which a robot model is built.

The advantage of using VR to simulate a manufacturing process lies on the fact that evaluation and verification can be done with a virtual trial run of that machining process. When evaluating the manufacturing process, if any part of the robot other than the tool collides with the workpiece or the fixture, it could not only be visualized but also be tested by a proper collision-detection function call from WTK. If the user is not satisfied with the machining process, he/she would have a chance to go back to the tool

generation program in AutoCAD or other CAD package, go through the tool path generation, and make another trial run with a redesigned tool path data.

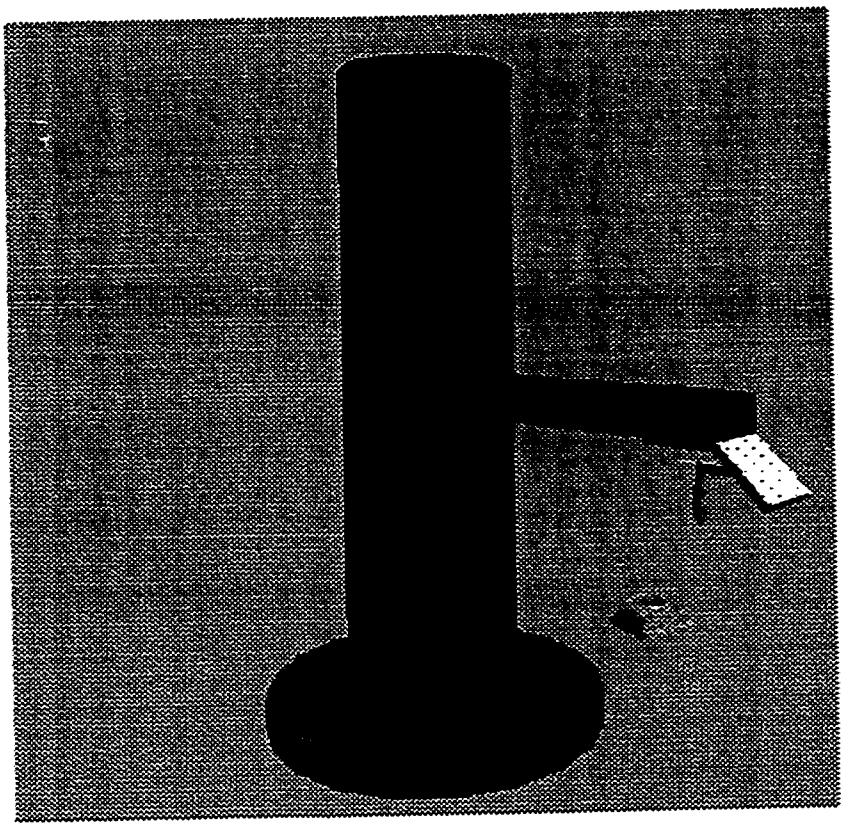

Fig. 4.1 Virtual Yamaha Robot Zeta-1

4.3 Kinematic Model of Yamaha Zeta-1 Robot

In order to perform the same motion in VR as the real robot, a virtual Yamaha Zeta-1 Robot, shown in Figure 4.1, has been modeled with the same size dimensions as the real one with all the degrees of freedom. Inverse kinematics is used to derive the movement of each joint of the robot from a certain tool posture. Figure 4.2 demonstrates each joint movement of the Yamaha Zeta-1 Robot.

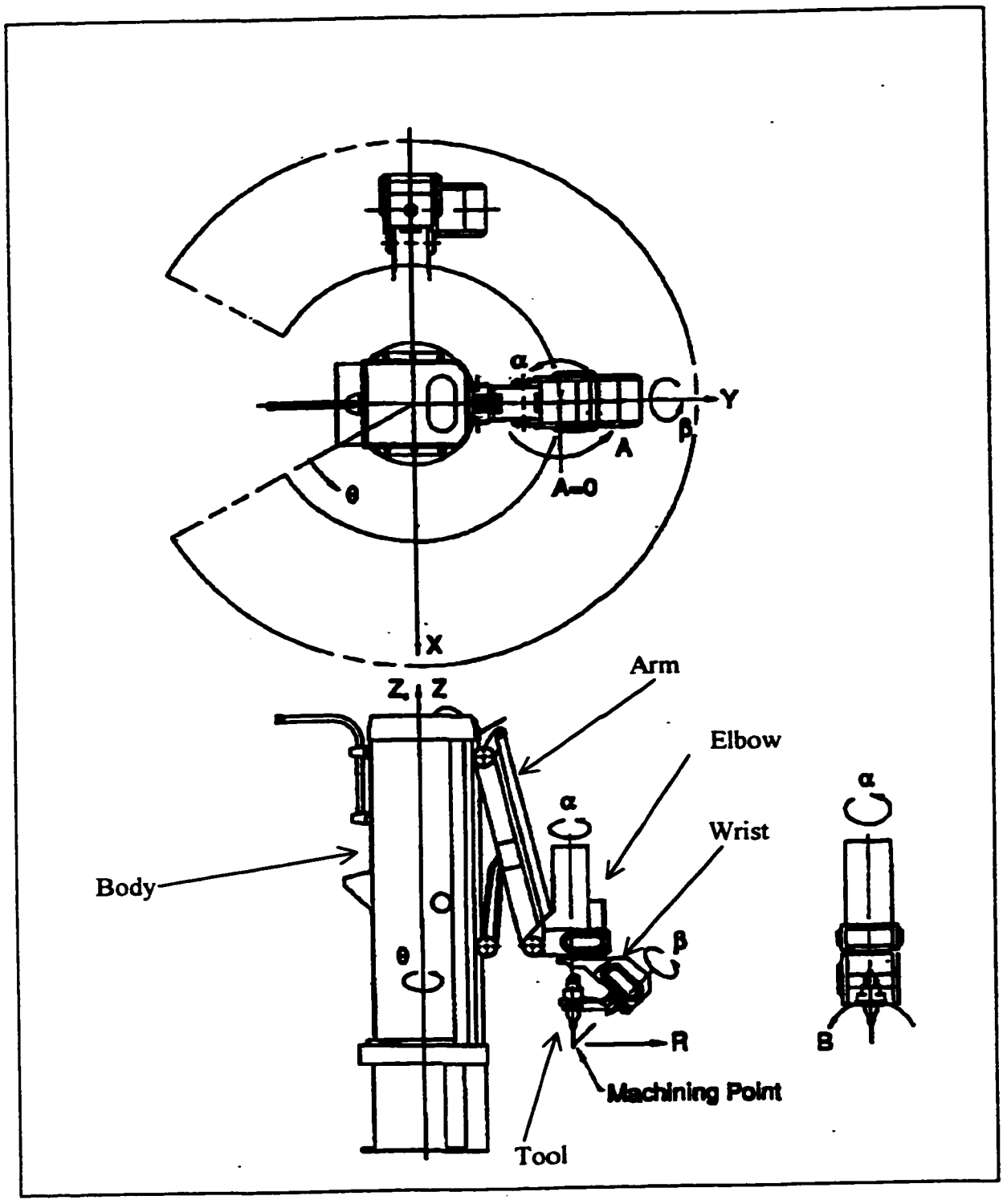

Fig. 4.2 Yamaha Zeta-1 Robot [2]

4.3.1 The Yamaha Zeta-1 Deburring Robot

The Yamaha Zeta-1 Deburring Robot (Figure 4.2) is a five axis manipulator with three joints (θ, R and Z) analogous to a cylindrical coordinate system and two wrist joints (α and β). The robot could rotate the whole body about the z-axis (θ), move its arm vertically along the z-axis (z) and extend its arm (R). The tool holder is designed in such a way that the tip of the tool (machining point) is held at the intersection of the two wrist axes: α and β. Owing to this unique feature, the position of the machining point is kept constant while the orientation of the tool can be changed by rotating the α and β axes (Figure 4.3). This robot has a repeatability precision of ± 0.1 mm and its payload is 30 kg [2].

Although it is structurally a cylindrical coordinates type robot, the commands issued to the robot are in a Cartesian coordinate system or WCS, as {X, Y, Z, A, B}. The Cartesian coordinate system is shown in Table 4.1 and Figure 4.2. The tool posture in the Cartesian coordinate system is presented in Table 4.2. Note that the term "tumbling surface" refers to the surface formed by the tool and the vertical axis going through the α joint.

One can see from these tables that the data for the robot are the machining point location and the tool orientation. The data are the input to the controller of the robot. These data are transformed through inverse kinematics to joint signals (θ, R, Z α and β), by the robot controller. The inverse kinematic transformation is discussed later in Section 4.3 in this chapter.

Axis Name	Minimum Setting Unites	0 Point (origin)	Positive Direction
X	0.01mm	Rotation center of θ axis	90° to the right of the y axis plus direction
Y	0.01mm	Rotation center of θ axis	Front of the robot
Z	0.01mm	Bottom end of range of movement	Upward direction
A	0.01°	The state where the tumbling surface* is matched with the x axis	Turning counterclockwise when viewed from above
B	0.01°	State where the tool is vertical	β axis plus direction

Table 4.1 Cartesian Coordinate System

* Tumbling surface is defined as the surface formed by the tool and the α axis.

Axis Name	Meaning	Description
X	x coordinate of machining point	Machining point position
Y	y coordinate of machining point	
Z	z coordinate of machining point	
A	Tool turning angle	Tool Posture
B	Tool collapse angle	

Table 4.2 Tool Location and Posture

The tool length is 150 mm as shown in Figure 4.3. The α axis intersects with the β axis at the machining point or processing point and they form an angle of 45°. As this figure shows, one can see clearly that the machining point stays constant no matter how α and β axes rotate.

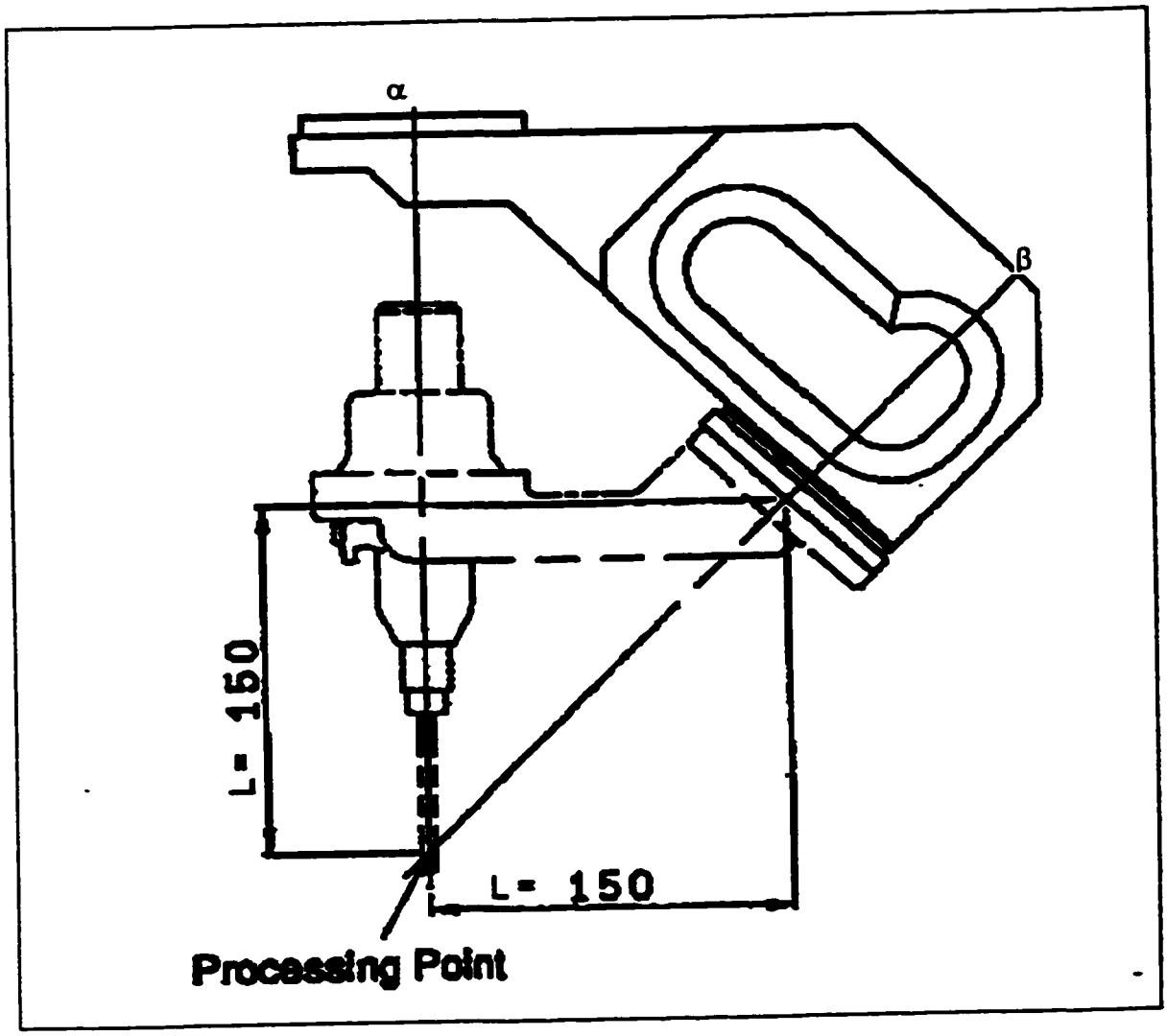

Fig. 4.3 α and β Axes of Yamaha Robot Zeta-1 [2]

The α and β axes could turn either clockwise or counterclockwise so that the sign of the resulted B is related to the sign of angle A, as shown in Figure 4.4. Two different A-B combinations can achieve a same tool posture. However, to simplify the tool path generation process, the angle B is always kept positive in this work.

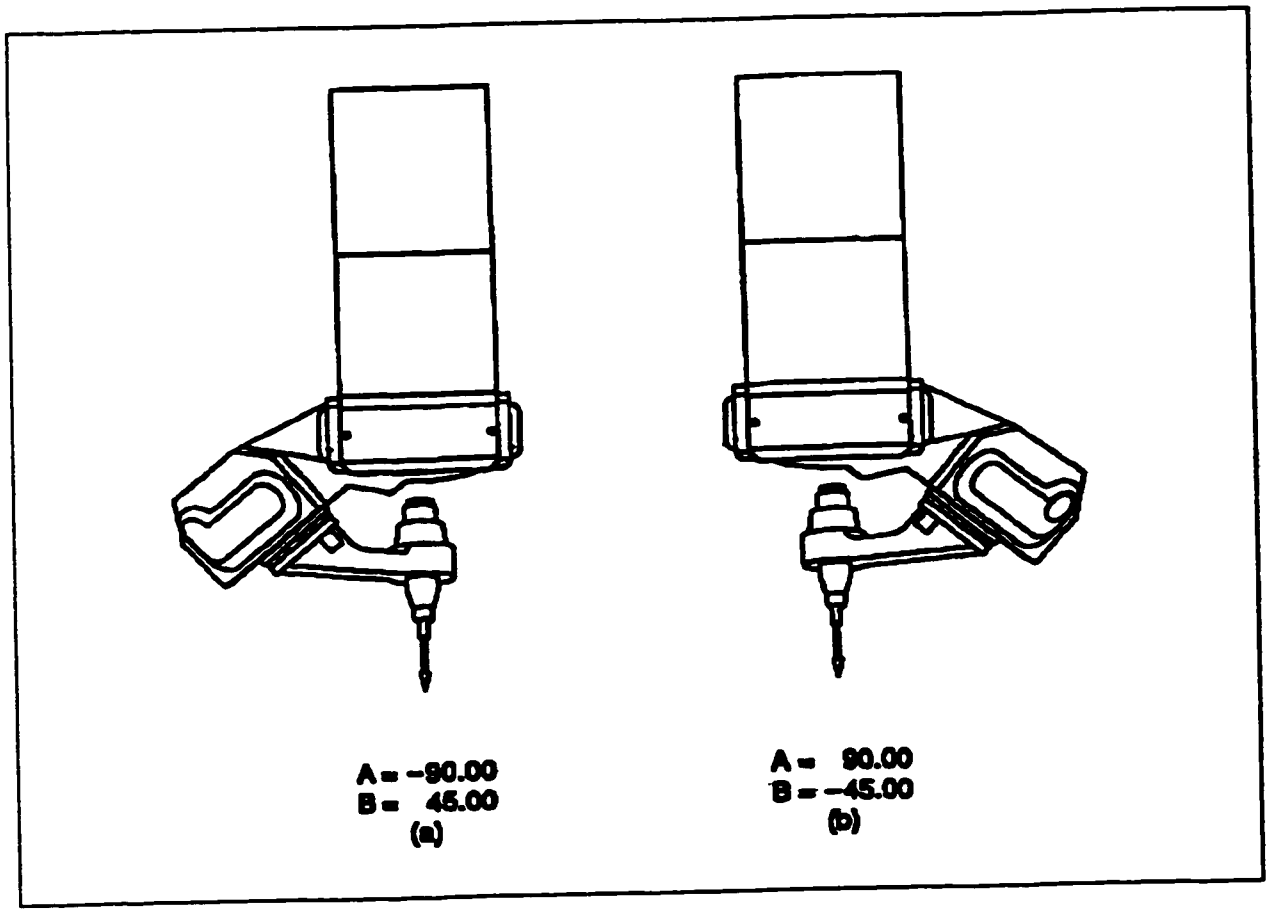

Fig. 4.4 Same Tool Posture by Different A and B [2]

4.3.2 Assemble Virtual Robot

There are five geometry parts that assemble the virtual robot and simulate the joints of the Yamaha Robot Zeta-1. They are called Base, Body, Arm, Elbow, Wrist, and Tool respectively. They are constructed either by programming in C++ language using WTK or by simply loading in AutoCAD 3D models. The parts such as the Base, Body and Arm, whose shapes and sizes are less important, are portrayed by some simple geometrical objects, which are created by C++ programming. Two cylinders represent the Base and the Body and a rectangular box represents the Arm. Figure 4.5 shows the sizes of the Base, Body and Arm. The Base is the only static object in the assembling and it

does not have any freedom of movement. The Body could rotate about its axis fully in 360 degrees. The arm has two freedoms of movement along the Body axis, which is the z-axis of the robot, and R direction, which intersects with the z-axis.

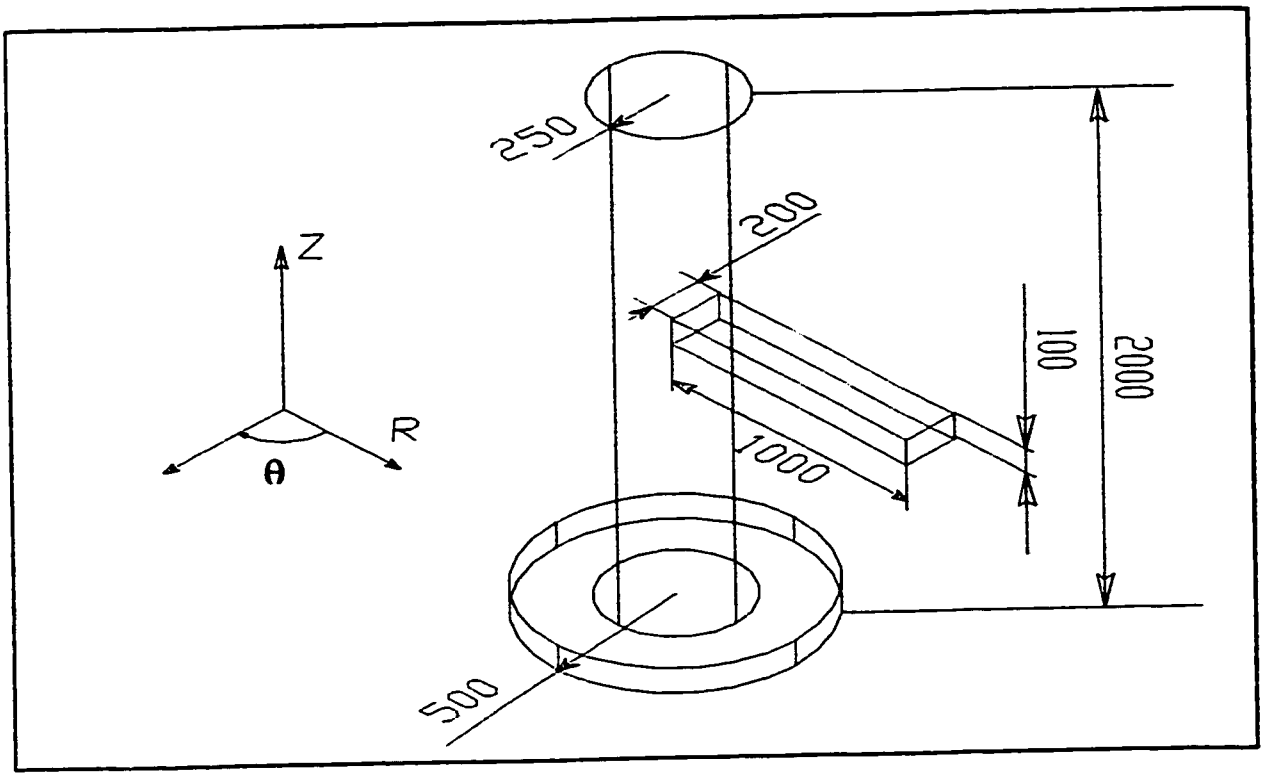

Fig. 4.5 Dimension of Robot Base, Body and Arm (in mm)

The dimensions are of less importance and are just approximates of the real robot. They only imply the range of movement of each part. The Arm is "attached" to the Body and there is a parent-child structure between all these geometric objects. For example, the Base is constructed based on the universe and is the root on which the Body is built. And the Arm is planted on the Body. By such "attaching", in the case of Body-Arm movement relationship, a motion link is established so that the Arm follows every rotation of the Body.

Using the modeling feature in AutoCAD, the Elbow, Wrist and Tool are created. The Wrist and Tool are integrated as one object, which is then called Wrist-Tool object. The Elbow and Wrist-Tool object have their unique shapes and sizes so that they form the joints of α and β. Figure 4.6 shows a rendered picture of the Elbow (left) and Wrist-Tool (right).

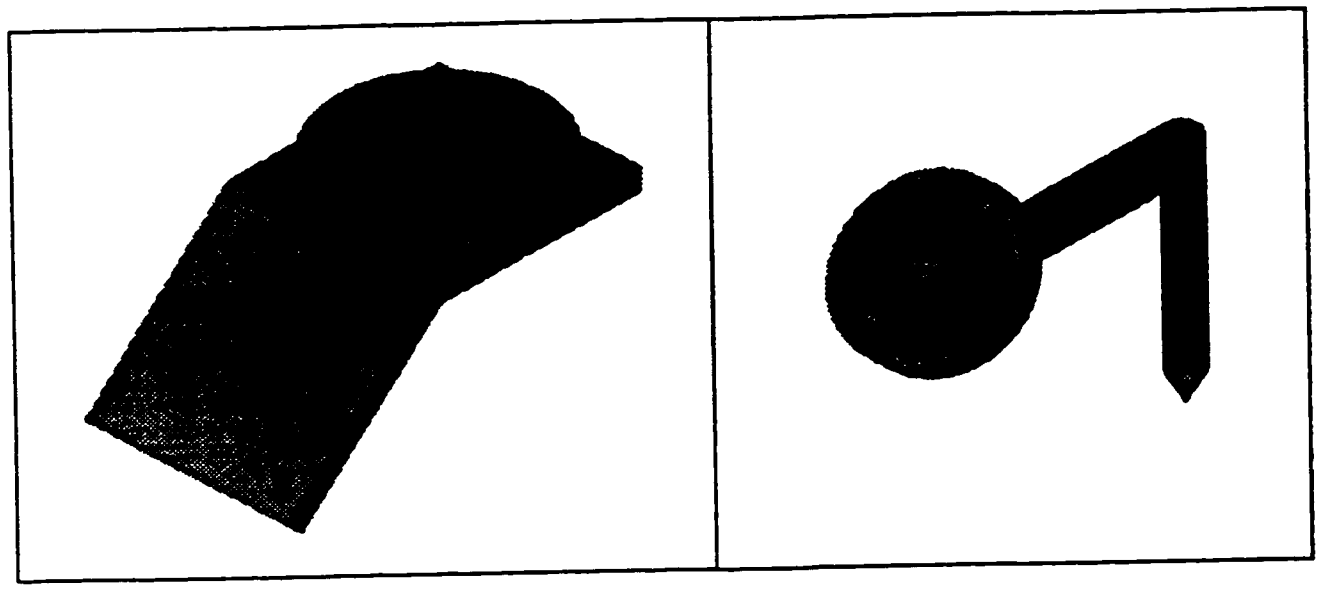

Fig. 4.6 Elbow and Wrist-Tool

The Elbow and Wrist-Tool are modeled into irregular geometrical shapes and it is difficult to designate the rotation axes for them. Hence, two cylinders are creates by C++ programming and they function as parents of these parts. Figure 4.7 shows how the Elbow and the Wrist-Tool are attached to their "parent" cylinders. This attachment is achieved by C++ programming, using WTK functions, and functions the same way as was explained for the Body-Arm attachment. By doing so, rotating of the joints α and β are simplified to the rotating of these parent cylinders about their axes. An assembly drawing of the Elbow and the Wrist-Tool is presented in Figure 4.8.

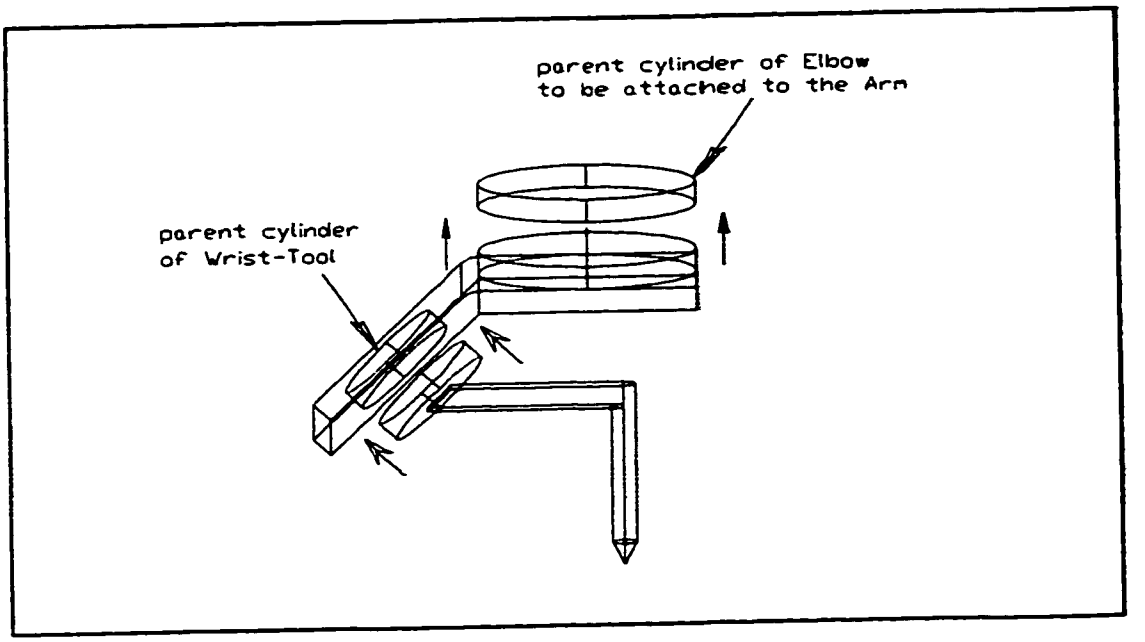

Fig. 4.7 Assembly of the Elbow and the Wrist-Tool

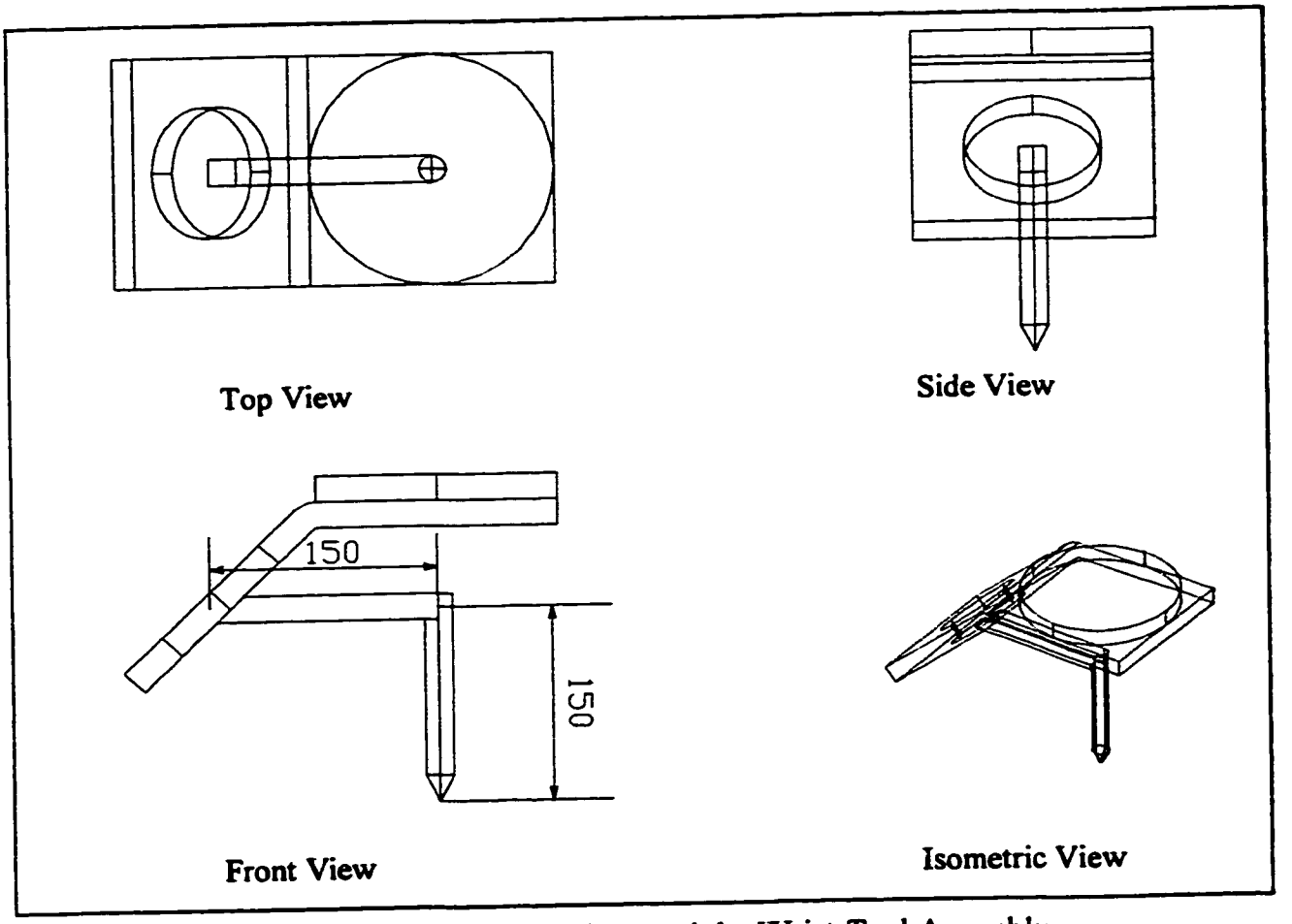

Fig.4.8 Draft of the Elbow and the Wrist-Tool Assembly

The robot is therefore constructed piece by piece. Given each joint certain degrees of freedom as the real one has, the freedom of movement of the real robot is then fully replicated.

A coordinate system is established in the virtual space and is coincident to the real robot. Also, the work piece is located at the same location as it is in the AutoCAD coordinate system, where the tool path is generated and also, the two coordinate systems are designated to be identical to the real robot. In other words, the reference systems for Yamaha robot, virtual robot and AutoCAD drawing are kept unified through all the procedures of 3D-model filtering, reconstruction, and tool path generation. Since the AutoCAD constructed workpiece (to be saved in the format of DXF or 3DS) is usually modeled under the reference at the origin of the AutoCAD coordinate system, the 3D-model of the workpiece is relocated into the simulation environment.

The workpiece could be loaded into the virtual space as a solid model either in 3DS or DXF type. In WTK, loading in such a 3DS or DXF object does not change its shape and size, which are modeled in AutoCAD or 3DSTUDIO. The workpiece is loaded into the virtual environment as a separate object independent to the robot and neither "parent" nor "child" object is attached to it. Therefore, movement commands will not alter the location of the workpiece.

In order to run the virtual robot, commands are to be issued to each joint based on the robot model. The commands are the joint coordinates, $\{\theta, R, z, \alpha, \beta\}$. However, the generated tool path is given by $\{X, Y, Z, A, B\}$. Inverse kinematic transformation is then applied to the generated tool path such that the world coordinates be transformed into joint coordinates.

4.4 Inverse Kinematics

In Chapter 3, the tool path including tool position and tool orientation is generated and is given in the form of coordinates in WCS {X, Y, Z, A, B}, which is derived from the coordinates of the two end points of the tool $\{x_1, y_1, z_1\}$ and $\{x_2, y_2, z_2\}$. For the robot model built in virtual reality, the movements of θ, R, z, α and β need to be derived so that the moving or rotating orders could be issued to each joint. In other words, knowing coordinates in WCS $\{x_1, y_1, z_1, A, B\}$, a transformation in machine joint coordinate space $\{\theta, R, z, \alpha, \beta\}$ needs to be found. This is called the inverse kinematic transformation of robot coordinates.

In order to determine the relationship between $\{x_1, y_1, z_1, A, B\}$ and $\{\theta, R, z, \alpha, \beta\}$, a very important and interesting feature of the Yamaha Robot Zeta-1 is then examined. The Elbow and Wrist-Tool shown in Figure 4.9 are used to demonstrate this feature. The α and β axes intersect with each other and form an angle of 45°. Suppose the Wrist has turned an angle β, the tool then tilts and forms an angle B with the vertical axis, as defined. Now, let the joint Elbow, which is the α axis, turn 360° and the trace of the tool is on a cone surface, keeping the same B angle. During all these turnings, the tool tip remains constant.

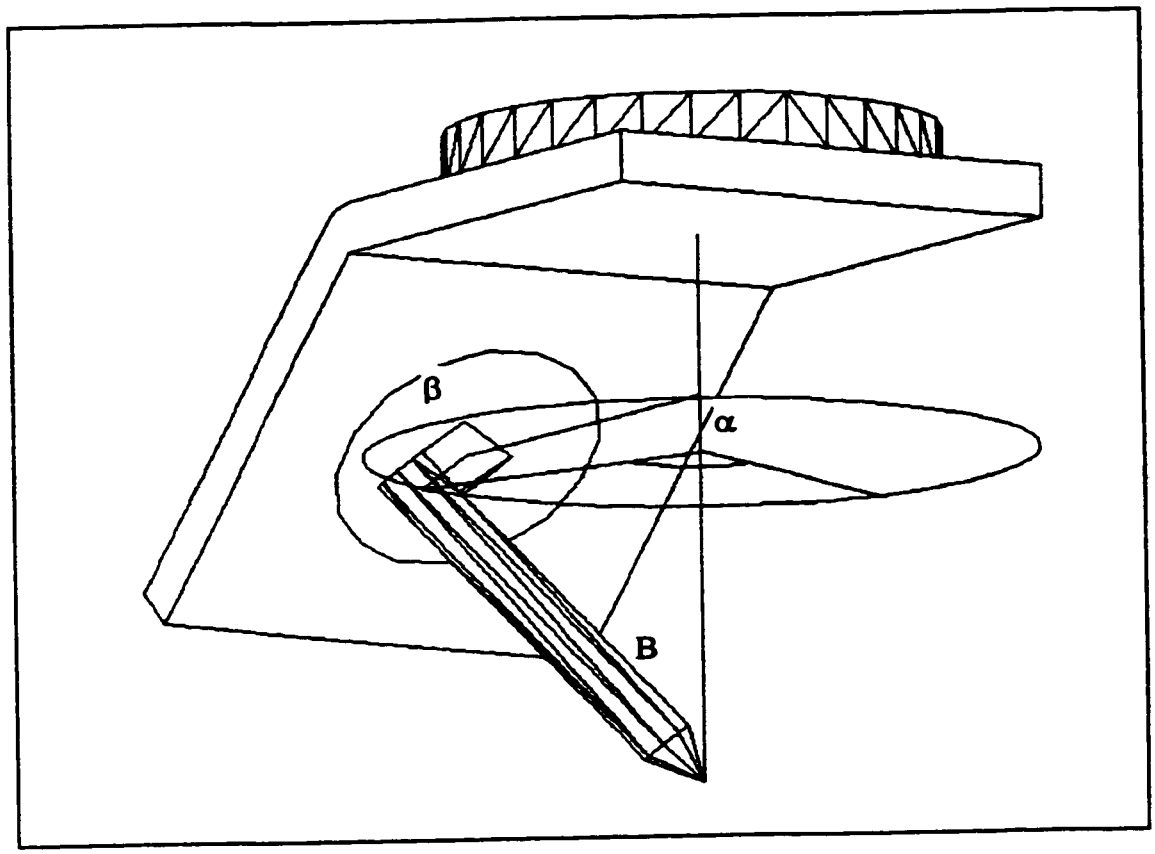

Fig. 4.9 Angle B, α and β

Assuming a tool with its location in WCS $\{x_1, y_1, z_1, A, B\}$ or $\{x_1, y_1, z_1\}$:$\{x_2, y_2, z_2\}$, the angle B, which is formed by the tool and the vertical axis with a maximum value of 90°, can always be determined by Equation (3.5) and it is only affected by the turning of the Wrist (β axis). A simplified drawing demonstrates the derivation of angle β, shown in Figure 4.10.

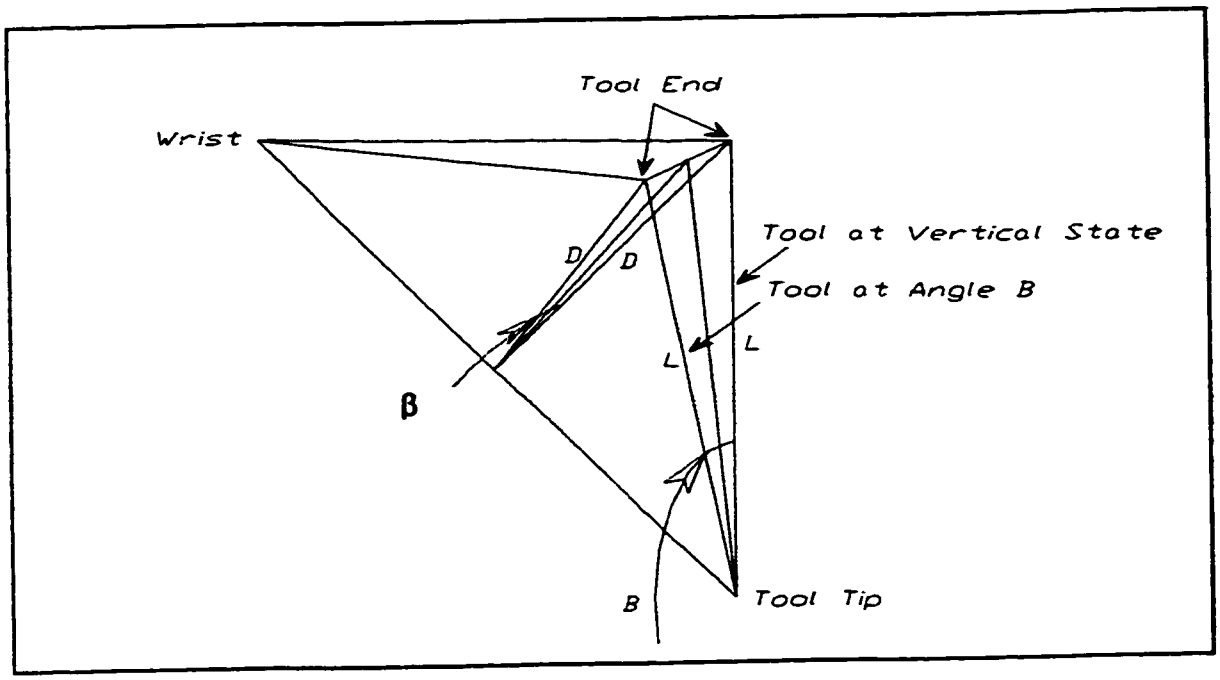

Fig. 4.10 Derivation of Angle β

Angle β can be seen and calculated as:

$$\beta = 2\sin^{-1}\left(\frac{L\sin(\frac{B}{2})}{D}\right) \tag{4.1}$$

where L is the length of the tool, which is 150mm, and $D = L\sin\frac{\pi}{4}$.

Observing the Yamaha Robot Zeta-1, one can see that the position of the machining point is only affected by the joints θ, R, z, because rotation about α and β axes does not change the position of the tool tip. This effect is then given by the transformation from a cylindrical coordinate system to a Cartesian coordinate system.

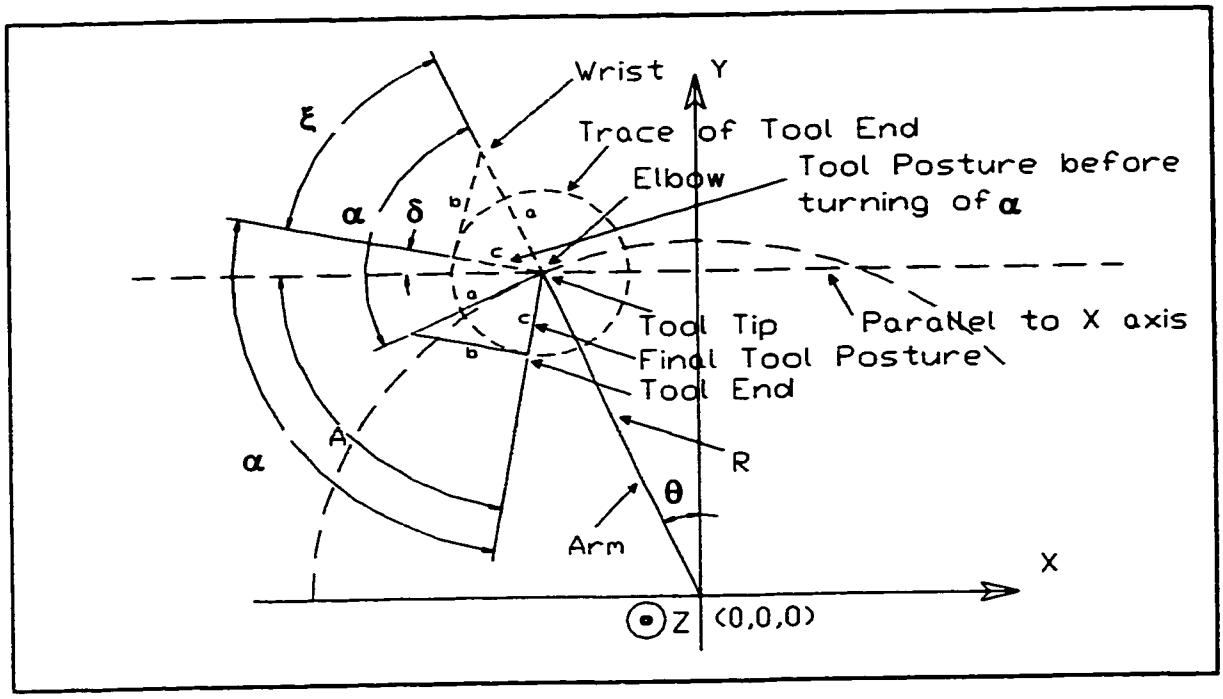

Fig. 4.11 Inverse Kinematics of Yamaha Zeta-1 Robot

Figure 4.11 shows a top view of the movements of the Yamaha Zeta-1 Robot. The Tool Tip and Tool End of a randomly chosen tool posture are pointed out in the figure as the "Final Tool Posture". For such a tool posture, note that the β axis had to be rotated otherwise the top view of the tool would only be seen as a point instead of a line. The movement of the robot has been divided into 5 processes in order: R, z, θ, β, α, knowing the coordinates of WCS $\{x_1, y_1, z_1, A, B\}$ or $\{x_1, y_1, z_1\}$:$\{x_2, y_2, z_2\}$. Assuming the robot completes each of these movements one by one, following the order of R, z, θ, β, α. The value of θ, R and z can then be easily derived as:

$$\theta = \tan^{-1}\left(\frac{y_1}{x_1}\right) \tag{4.2}$$

$$R = \frac{x_1}{\cos\theta} \text{ or } R = \frac{y_1}{\sin\theta} \tag{4.3}$$

$$z = z_1 \tag{4.4}$$

And β is calculated by Equation (4.1). Derivation of machine coordinate α is not straightforward because both turnings of the joints α and β are involved and contribute to the angle A value. Examining the top view in Figure 4.11, supposing the robot has completed the movements of R, z, θ, and β, the tool tip, tool end and the joint Wrist forms a triangle with each side shown as a, b, c respectively (shown with dashed-lines). During the final α turning, this triangle remains constant, as shown.

Let

$$\begin{cases} \Delta x = |x_2 - x_1| \\ \Delta y = |y_2 - y_1| \\ \Delta z = |z_2 - z_1| \end{cases} \tag{4.5}$$

and

$$\begin{cases} a = L \\ b = \sqrt{L^2 - (L - \Delta z)^2} = \sqrt{2L\Delta z - \Delta z^2} \\ c = \sqrt{\Delta x^2 + \Delta y^2} \end{cases} \tag{4.6}$$

Solving the triangle Δabc, angle ξ is found as

$$\xi = \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2ac}\right)$$
$$= \cos^{-1}\left(\frac{L^2 + \Delta x^2 + \Delta y^2 + \Delta z^2 - 2L\Delta z}{2L\sqrt{\Delta x^2 + \Delta y^2}}\right) \quad (4.7)$$
$$= \cos^{-1}\left(\frac{L - \Delta z}{\sqrt{\Delta x^2 + \Delta y^2}}\right)$$

where

$$\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} = L$$

And

$$\delta = (\frac{\pi}{2} - \theta) - \xi \quad (4.8)$$

Then α is found as

$$\alpha = \tan^{-1}\frac{\Delta y}{\Delta x} + \delta \quad (4.9)$$
$$= A + \delta$$

Finally, α is derived and depicted by Δx, Δy and Δz. The relationship between angle {A, B} and {Δx, Δy, Δz} are described by Equation (3.4) and (3.5). {Δx, Δy, Δz} can also be depicted by angle {A, B}as follows:

$$\Delta x = |L\cos A|$$
$$\Delta y = L\sqrt{1 - \cos^2 A - \cos^2 B} \quad (4.10)$$
$$\Delta z = |L\cos B|$$

Knowing $\{x_1, y_1, z_1, A, B\}$, the variables $\{\theta, R, z, \alpha, \beta\}$ is then given respectively by Equations (4.2), (4.3), (4.4), (4.9) and (4.1), and the inverse kinematics transformation of the robot is completed.

Deriving the inverse kinematics of the robot is totally based on the geometric structure of the robot. Different approaches to derive the forward and inverse kinematics have been achieved by Ayyadevara [7] and Y. Su [17].

4.5 Executing Tool Path in Virtual Reality

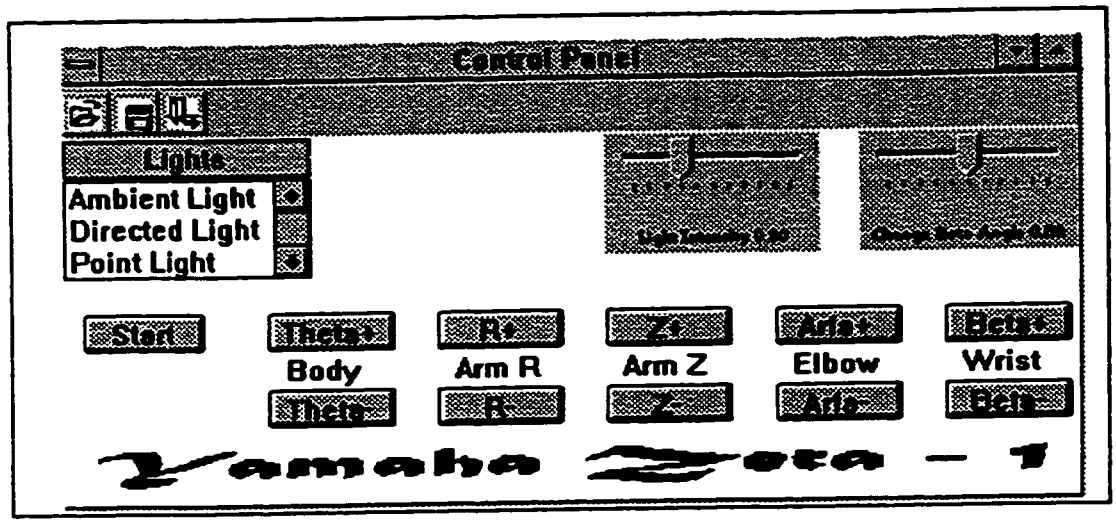

Fig. 4.12 Control Panel for Simulation

A Control Panel is created for animation purposes, as shown in Figure 4.12. This Panel is a simplified Teaching Unit, similar to that used in the real robot. There are 11 push-on buttons created for the tool path verification and robot manipulating. The tool path animation starts when the "Start" button is pushed. The other 10 buttons are created for the user to change the robot coordinates: θ, R, z, α,β respectively without the need to read a data file.

There are two data files that are generated by the tool path generation program in AutoCAD. One data file lists the coordinates of the two end point of the tool, $\{x_1, y_1, z_1\}$ and $\{x_2, y_2, z_2\}$, while the other one provides robot coordinates to run the robot, $\{x_1, y_1, z_1, A, B\}$. When the simulation starts to run, by using the first data file, a trace of the tool is created at the location according to the coordinates of $\{x_1, y_1, z_1\}$ and $\{x_2, y_2, z_2\}$ by putting small dots at those locations. By doing so, the tool postures achieved by rotating and moving the robot joints could be verified by observing the two ends of the tool following the tool trace. The second data file contains the commands $\{x_1, y_1, z_1, A, B\}$, which are to be fed to both the virtual robot and the real robot. The simulation program will transform the commands $\{x_1, y_1, z_1, A, B\}$ into $\{\theta, R, z, \alpha, \beta\}$, by which each variable actually moves or rotates the robot joints. Therefore, the two data files are independently used. Data provided by the second data file runs the robot and simulates the VR machining process while data from the first data file verifies the simulation.

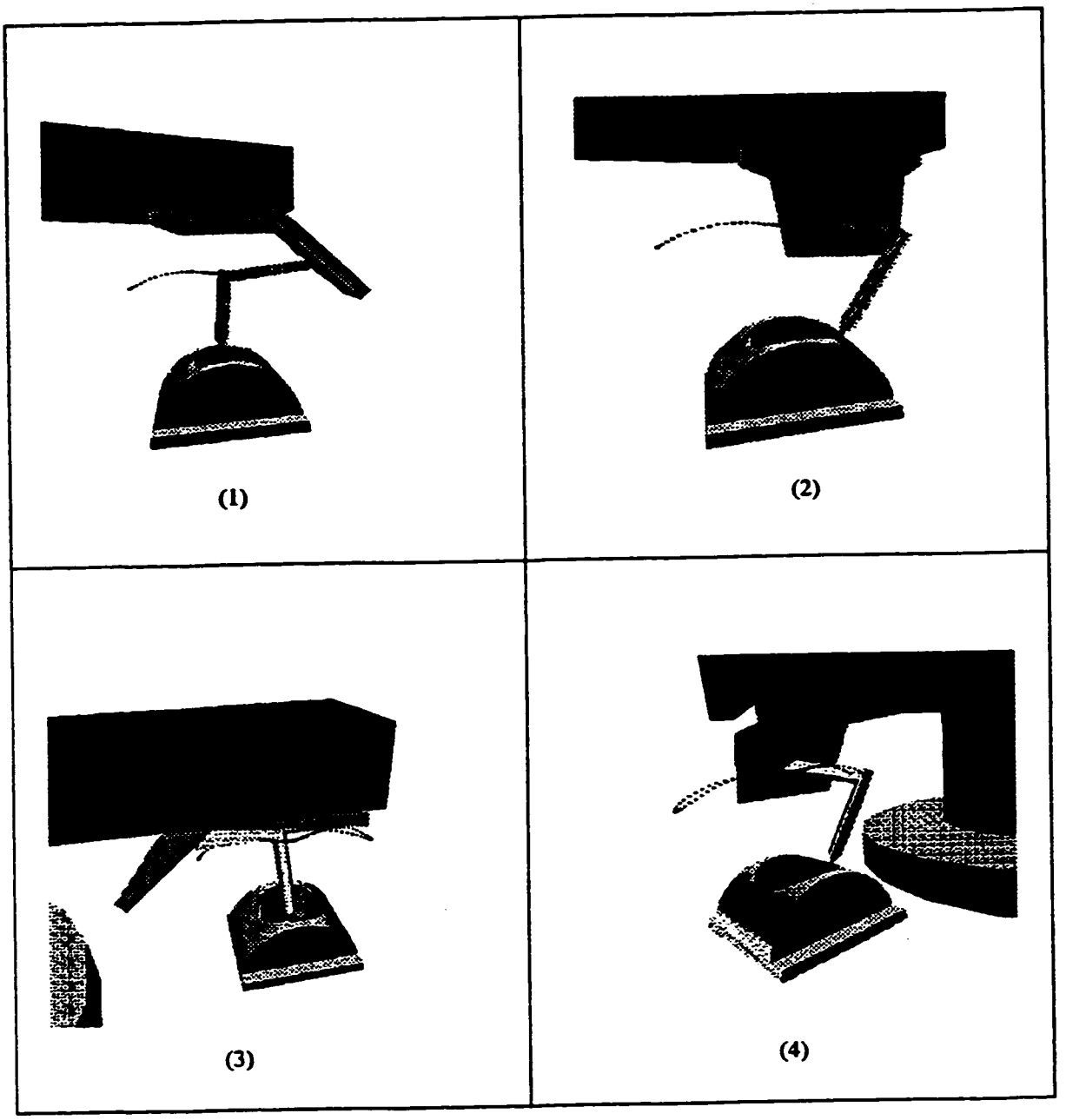

Fig. 4.13 Simulation of a Machining Process

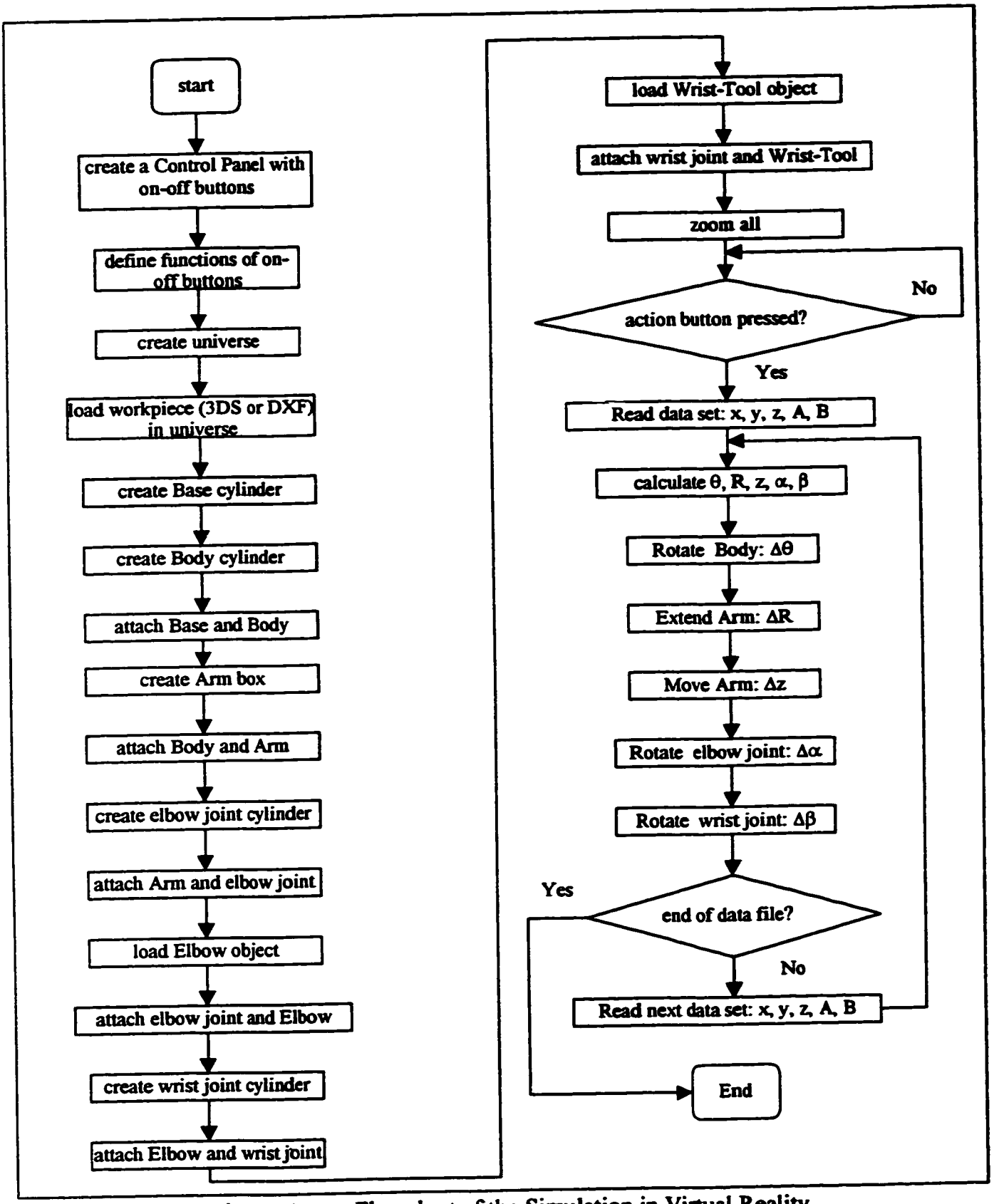

Fig. 4.14 Flowchart of the Simulation in Virtual Reality

The simulation process in Figure 4.13 shows the tool is machining the intersection edge of a workpiece, following the tool path generated by the CAM package, from 4 different points of view. A schematic flowchart of the simulation program is presented in Figure 4.14.

4.6 Summary

The kinematics of the Yamaha Zeta-1 Robot has been studied and explained in detail. With the help of WTK, VR is introduced as a simulation tool. A kinematic robot model has been developed, fully replicating the inverse kinematics of the Yamaha Zeta-1 Robot. The potential of applying of VR in the field of robotic simulation has been demonstrated. Although only kinematics of the robot has been studied, the current model can be further extended to simulate a dynamic robot by applying the equations of motion to the model.

Chapter 5

Tool Path Verification

5.1 Introduction

The previous three chapters discussed and focused on the development of the software that provides the post-processed data or command data for simulation in a VR environment and the real robot. The experiments described in this chapter show that the tool path designed by the user and generated by the CAM package is verified by running the Yamaha Zeta-1 Robot with the generated data. Since a zero radius cutter has been assumed when the tool path data is generated and only kinematic transformation is involved in this work, the experiments are carried out without any real machining or cutting being performed. The trajectory tracking performance of each axis of the robot is not within the scope of this thesis and has been discussed and conducted by Ayyadevara [7].

5.2 Locating Workpiece

The workpiece chosen for the tool path verification is shown in Figure 2.1 and the tool path data have been generated for different edges on it. The workpiece is to be placed on a horizontal surface of the working table. There are two position holes purposely designed for locating the workpiece on the working table of the robot. Figure 5.1 shows the locations of the locating holes. The centers of the locating pins on the working table are measured with respect to the robot WCS. In AutoCAD 3D modeling environment, the workpiece model has been moved and rotated until the two centers of the holes are located at those two points. An alternative approach to locate the workpiece

in AutoCAD is to model the workpiece starting at the location of the position holes. The tool path is then generated based on the location of this re-located 3D model.

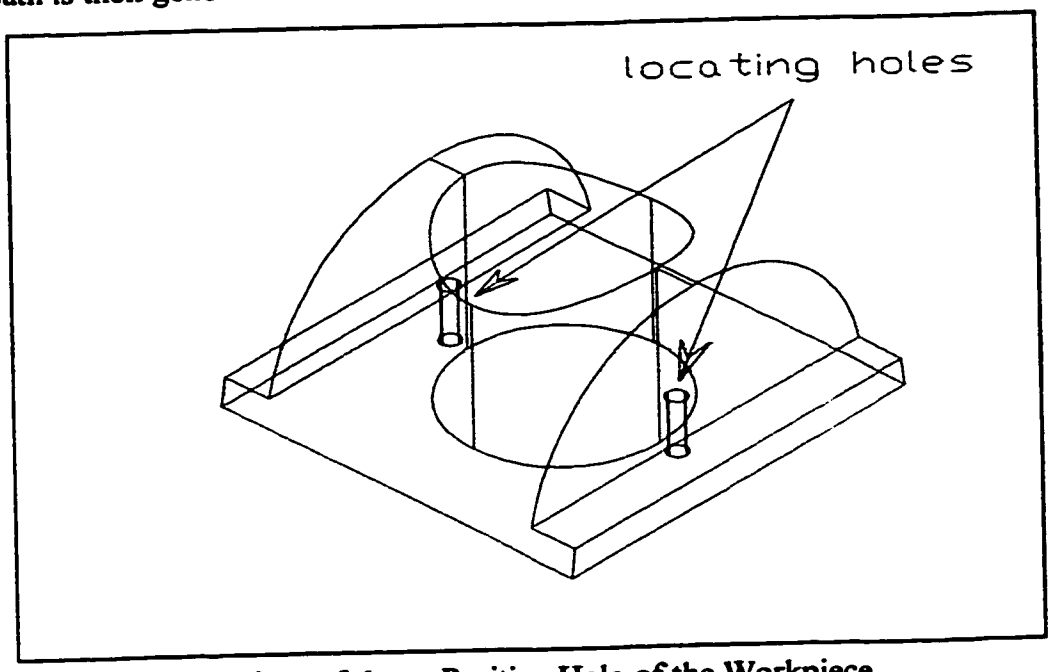

Figure 5.1 Position Hole of the Workpiece

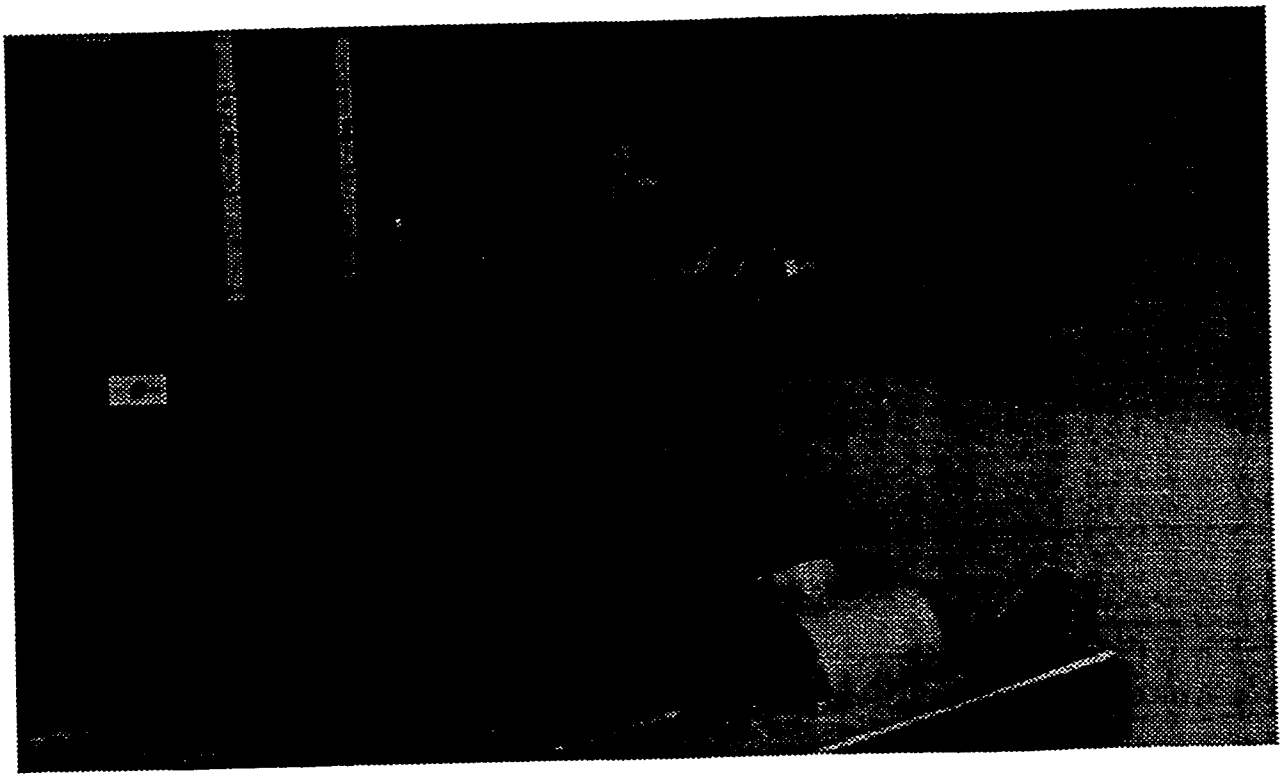

Figure 5.2 Workpiece ready to be machined

On the working table for the robot, the workpiece is located by fixing the workpiece on the table where the locating pins have already been placed. Additional clamps have been added in order to make sure that the workpiece has been fastened. Figure 5.2 shows the workpiece is fixed on the working table.

5.3 Tool Path Execution

In teaching mode, the Yamaha Zeta-1 Robot accepts instruction from the teaching station, shown in Figure 5.3. The teaching unit station consists of five joysticks. Each of them manipulates the motion of one axis in the Machining Coordinate System {X, Y, Z, α, β}. The teaching unit also has a keypad and LCD (Liquid Crystal Display) which not only can be used to manipulate the motion of each axis, but also allows the user to input a listing of point sequences.

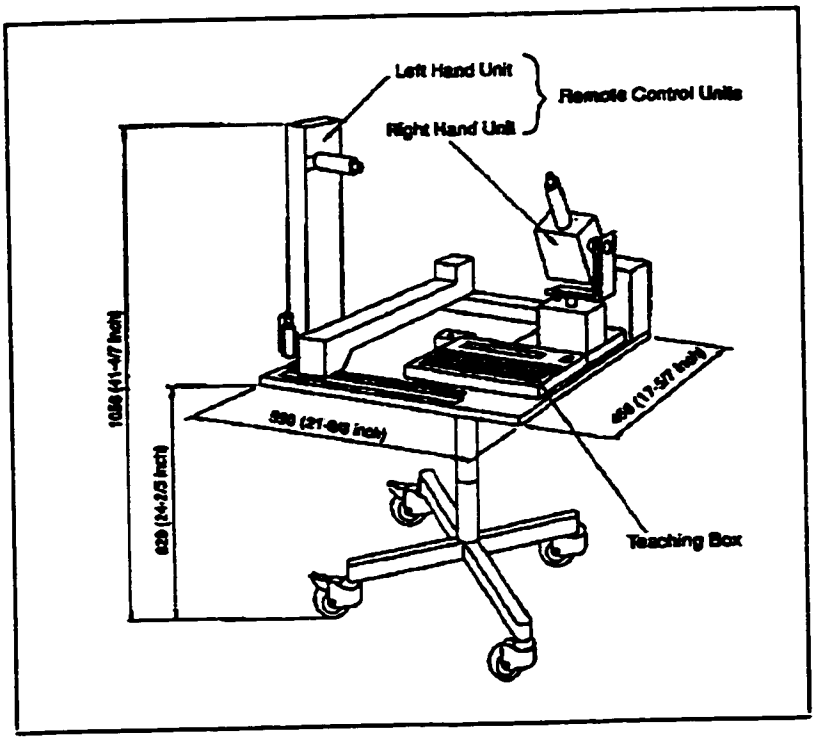

Fig. 5.3 Teaching Station of Yamaha Robot Zeta-1 [2]

Instead of using the real cutter, a digital displacement probe is mounted in place of the cutting tool so that the actual cutting error can be measured. It is a contact type sensor called Digimatic Indicator, shown in Figure 5.4. It converts the linear displacement to a digital signal by a capacitance type encoder [31].

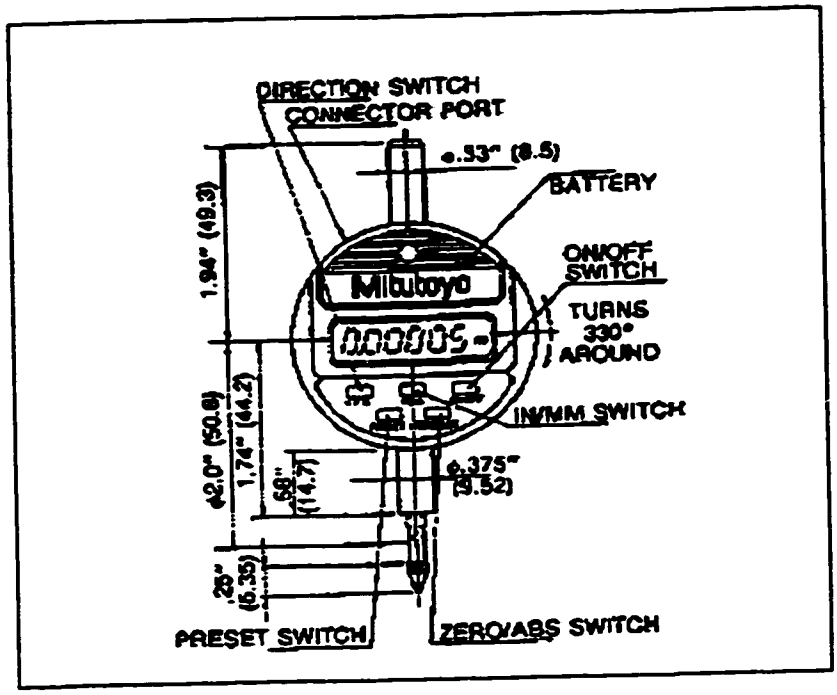

Fig. 5.4 Digimatic Indicator [31]

Experiment procedure:

1) The workpiece in AutoCAD is modeled based on the mechanical drawing of the workpiece. A reference point is chosen as the origin of the 3D model. The 3D model is "exported" to produce a "3DS" file of the object.

2) The AutoCAD 3D model of the workpiece is repositioned to the locating holes which are at coordinates (-833.45, 139.47, 56.40) and (-795.03, 263.76, 56.40). The change of the reference point is then found and recorded for step (7).

3) In AutoCAD environment, the 3D model is "exported" to produce a "SAT" file of the object.

4) In DOS shell, filter.exe is run to produce the re-constructor by inputting the exported SAT type file name. The re-constructor is automatically generated.

5) Back in AutoCAD environment, the 3D model is "double exploded". The re-constructor is loaded and run under AutoCAD. A wireframed 3D model is then constructed.

6) In AutoCAD environment, the tool path generation command, "cic_line" or "cic_intersec" and etc. could be run to generate tool path for a certain edge.

7) Run the simulation program. The change of the reference point from step (3) should be input as the location of the workpiece in the simulation environment. The 3D model in "3DS" file type is then located. Also, the generated data file name is input. While running the virtual robot, observe carefully the tool posture when the virtual robot is maneuvering during the machining.

8) If tool postures in the simulation is not as desired, step (6) and (7) should be repeated.

9) Experiment the tool path data with the Yamaha Robot Zeta-1.

   i) Turn on the Yamaha Robot Zeta-1 set it in Teaching Mode.

   ii) Use joysticks to move the machining point, angles A and B close to the coordinates and angles of the first set of tool path data.

   iii) Using the keypad on the Teaching Unit, input the first set of data so that the robot will move to this point. Observe the final tool posture.

   iv) Go to the next set of data, repeat ii) and iii).

v) After going through all the data manually, if all of the data give the correct tool posture, switch to the workstation, input the data file name and run the robot automatically and continuously.

In step (9) (v) of the experiment, to avoid the tool collision with the workpiece, the digital probe is preset to –5 mm by pushing in the probe upwards. In the manually running of the robot (step (9) (ii) and (iii)), the probe is released to measure the actual displacement point by point.

5.4 Observations of Yamaha Zeta-1 Robot Tool Motion

The tool path generated by the developed CAM package is tested on the Yamaha Zeta-1 Robot. By feeding the data script file to the Teaching Unit of the robot, it is observed that the trajectories of the machining points do coincide with the desired edges of the workpiece. Although only one workpiece has been tested and the location of the workpiece is not altered during the experiment, by verifying tool path on different edges of the workpiece, the robot successfully tracks the geometry, such as straight line, arc, and spline curve. It is difficult to measure the tool angle directly from the robot. However, the data of the tool angle is fed to the robot by the interface workcell and the Teaching Unit could display such data all the time, which means the actual angle coincides with the data.

Chapter 6

Conclusions and Recommendations for Future Work

6.1 Conclusions

An automated robotic deburring & polishing CAM software package has been successfully developed. It allows the user to interactively specify desired edges and surfaces, design tool posture, generate tool posture data and post-process the generated data for the Yamaha Zeta-1 Robot. It provides an interactive designing environment built on the most popular CAD software, AutoCAD, which eases the communication between the user and the 3D models of workpieces. It has been demonstrated that there is a viable approach to take advantage of AutoCAD and AutoLISP to process the solid model representation of the geometry of a workpiece.

The main contribution of this work lies on the CAM features developed based on AutoCAD. A solid 3D model of a workpiece is processed and reconstructed into a 3D-wireframe model by abstracting necessary geometric information. Straight edges, circular edges and intersection edges are then accessed and tool paths are generated for deburring purposes. Also, tool path for plane surfaces and cylindrical surfaces could also be generated by selecting edges that form the surface. In the procedures of tool path generation, the user is allowed to design, determine and modify the tool path.

The simulation environment in virtual reality successfully simulates a kinematic model of the Yamaha Zeta-1 Robot and is used to verify the generated tool path data. It is implemented by taking advantage of the virtual reality toolbox, WorldToolKit Release 6, which is one of the leading software packages in the field of virtual reality programming.

An inverse kinematic model of the robot has been derived. The model is incorporated into the virtual robot model and enables the virtual robot to accept the robot command data and simulate the machining. The running of the simulation demonstrate the correctness of the derived inverse kinematics.

The experiment with the Yamaha Zeta-1 Robot has verified the generated tool path. It demonstrated that the AutoCAD database provides a precise description of the workpiece and the database is usable for CAM purposes. The results have been compared to the one yielded by another approach that generates tool path for the same edges by establishing geometrical equations and calculating all the geometrical properties for the edges [17]. Data points selected from the two approaches used for comparison show exact conformity. However, the advantages of the approach by this work lies on the facts that 1) a complete description of the geometric database is presented by AutoCAD and ACIS; 2) flexibility is provided to switch the workpieces or the geometric entities from one to the other; 3) user interactivity presents a pleasant working platform for the users.

6.2 Recommendations for Future Work

The work demonstrated the use of AutoCAD database in Tool Path Generation. Though AutoCAD provides a good interactive working environment, it is not a CAM software that is used to facilitate the manufacturing processes. By using AutoLISP, this research work starts a CAM toolbox for the purpose of tool path generation. More and more CAM tools could then be designed, created and added into the toolbox other than the existing ones. Also, other CAD databases could also be investigated and taken advantage of.

When dealing with spline curves. the error estimating process for the tool path generation stipulated in this thesis becomes tedious if high precision is required. The work only provides a technical solution for the error estimation and it requires the user be skillful in AutoCAD techniques. However, the error could be fully solved mathematically by programming the spline equation into the tool path generation process. It will consume a lot of computing time because a reconstruction of a 4^{th} order spline is needed.

Although the simulation and animation are based on the kinematics of the robot, dynamic properties of the robot could be added and simulated in the virtual reality environment. If the simulation is based on a dynamic model of the Yamaha Zeta-1 Robot, the forces, torque and other robot arm reactions could then be simulated.

Collision detection is not such a significant feature in the simulation part because the workpieces that have been dealt with are all "regular" geometries. However, this collision detection feature provided by WTK will be more critical when machining "irregular" surfaces or workpieces with more complicated geometry, say, free form surfaces or sculpture surfaces. In such cases, machine tool collision with the workpiece is usually expected in the trial run.

The data produced by this interface is formatted specifically for the Yamaha Zeta-1 Robot. Since the tool path is also saved in absolute coordinates in the 3D space, the tool path could also be used for NC machines by applying NC kinematics.

The highlight of this thesis is making use of AutoCAD as an interactive interface between the engineer and mechanical drawings for manufacturing. This approach and its potential have been demonstrated and the interface has been successfully established. However, this CAM package is not so professional in the sense of computer science. To

make it more functional and even a commercial product, effort and future work should be focused on the understanding of AutoCAD graphics, AutoCAD database, AutoLISP and DCL programming to improve the user interface. These commercial CAD tools could facilitate the user to design and generate machining tool path for complex surfaces and geometry.

There are many CAD/CAM systems, such as CADKEY, Pro-Engineering, Cartia and etc. They are powerful but expensive. However, learning to use these software packages could also be a great help to improve the interface of this AutoCAD-WTK CAM package.

References

1. Gillespie L.K., King Robert E., "Robotic Deburring Handbook", SME 1987

2. YAMAHA Zeta-1 Deburring Robot: User 's Manual, Version 1.2, YAMAHA Corporation, Hamamatsu, January 1990

3. Kral Irvin H., "Numerical Control Programming in APT", Prentice-Hall 1986.

4. Chang, C.H., and Melkanoff, M.A., "NC Machine Programming and Software Design", Prentice-Hall, New Jersey, 1989

5. Loney, Gregory C. and Ozsoy, Tulga M., "NC machining of free from surfaces", Computer-Aided Design, Vol. 19, No. 2, pp. 85-90, 1987

6. Selleck, C.B., Loucks, C.S., "A system for Automated Edge Finishing", Proc. 1992 IEEE international Conference on Systems Engineering, Pittsburgh, pp. 423-429, 1992

7. Ayyadevara, V.R., "Development of an Automated Robotic Deburring Workcell", Master's thesis, Department of Mechanical Engineering, Concordia University, 1996

8. Cheng, R.M.H., Rajagopalan, R., Temple-Raston, M., "The Differential Geometric Modeling of Compressor Blades", Proc. American Control Conference, Baltimore, Maryland, pp. 1913-1917, June 1994

9. Kruth, Jean-Pierre, Klewais, Paul, "Optimization and Dynamic Adaptation of the Cutter Inclination during Five-Axis Milling of Sculptured Surfaces", Annals of the CIRP Vol, 43/1, pp 443-448, 1993

10. Murphy, K.N., Norcross, R.J., Proctor, F.M., "CAD Directed Robotic Probing", Send International Symp. On Robotics and Manufacturing Research, Education, And Application, Albuquerque, November 1988.

11. Kramer, B.M., Bausch, J.J., Gott, R.L., Dombrowski, D.M., "Robotic Deburring", Robotics and Computer-Integrated Manufacturing, Vol. 1, No. ¾, pp. 365-374, 1984

12. Kramer, B.M., Shim, S.S., "Development of a system for Robotic Deburring", Robotics and Computer-Integrated Manufacturing, Vol. 7, No. ¾, pp. 291-295, 1990

13. Stauffer, R.N., "Sensor Simplifies Changeover in Deburring Operation", Robotics Today, Vol. 9, No. 4, August 1987

14. Decamp, W.H., "Breaking the Edge", Manufacturing engineering, July 1989

15. Shoham, M., Srivatsan, R., "Automation of Surface Finishing Processes", Robotics and Computer-Integrated Manufacturing, Vol. 9, No. 3, pp. 219-226, 1992

16. Jones, Frederic H. and Martin, Lloyd, "The AutoCAD Database Book, Accessing and Managing CAD Drawing Information", Third Edition, Ventana Press, 1989

17. Su, Y., "Development of a User Interface for Processing Geometric Data from Off-the-shelf CAD Packages and Motion Planning for Yamaha Zeta-1 Deburring Robot", Master's thesis, Department of Mechanical Engineering, Concordia University, 1999

18. Koren, Yoram, "Computer Control of Manufacturing Systems", McGraw-Hill, Inc. 1983

19. R.M.H. Cheng, J. Li, R. Rajagopalan and H. Hong, "Development of The Concordia SeMAST Program – A secondary Machining Automation Software Tool", Conference on Applications of Automation Science and Technology, 4-26 Hong Kong, November 1998

20. Head, George O., "AutoLISP In Plain English, Fifth Edition", Ventana Press, Inc. 1995

21. Walsh, D., Kinght, R. L., Valaski, W. R., "AutoCAD 13 Secrets", IDG Books Worldwide, Inc., 1996

22. Auto Reference Manual, Autodesk, Inc. Publication 100752-01, August 6, 1992

23. Mechanical Desktop Student Guide, Autodesk Inc., 1996

24. ACIS Geometric Modeler Format Manual, Version 2.1, 9/96, Spatial Technology Inc., Copyright © 1996

25. Soen, F., Pitzer D., Fulmer, H. M., Boyce, J., McWhirter, K., Peterson, M. T., Gesner, B. R., Beck, J., Coleman, K., Morris, A., Boersma, T., Fitzgerald, J., "Inside AutoCAD Realease 13 for Windos and Windows NTTM", New Riders Publishing, 1995

26. AutoCAD Customization Guide, Autodesk Inc., 1994

27. Craig, John J., "Introduction To Robotics: Mechanics and Control", Second Edition, 1989

28. Lee, Yuan-shin and Chang, Tien-Chien, "2-Phase approach to global tool interference avoidance in 5-axis machining", Computer-Aided Design, Vol.27, No. 10, pp. 715-729, 1995

29. Beier, K. P., "Virtual Reality: A Short Introduction", Web page at *www.vrl.engin.umich.edu/intro.html*, Virtual Reality Laboratory, University of Michigan, June 5, 1999

30. WORLD TOOLKIT Reference Manual & Hardware Guide (Release 6), Sense 8 Corporation, 1996

31. Operation Manual for IDC Series 543 Digimatic Indicator, Manual No. 3041, Mitutoyo Corporation, Tokoy, Japan

Appendix 1

User Interface Based on AutoCAD Platform

The following texts are inserted into the files of AutoCAD.

ACAD.MNS (in AutoCAD) or MCAD.MNS (in Mechanical Desktop)

```
***POP15
ID_Toolpath     [T&oolPath]
                [&CIC Features]^Z(load "cic.lsp")
                [--]
                [->  Edging Features]
ID_Wireframe    [Construct &Wireframe ]^C^C_cic_explode
ID_Rewireframe [Reconstruct W&ireframe ]^C^C_rewire
ID_cicut        [C&utting Entity]^C^C_cic_cut
ID_GToolpath    [->Toolpath]
                [&Line]^C^C_cic_line
                [&Arc ]^C^C_cic_arc
                [&Circle]^C^C_cic_circle
                [<-<-&Intersection]^C^C_cic_intersec
                [--]
                [->  Polishing Features]
ID_Surface      [Construct &Surface ]^C^C_am2sf;_face
ID_GToolpath    [->Toolpath]
                [&Plane surface]^C^C_cic_planesurf
                [<-<-&Cylindrical surface]^C^C_cic_cylsurf
                [--]
ID_TAbout       [  &About...]^C^C_tabout
                [--]
                [  &Experiment]^C^C_experiment


**CIC_Edging
ID_cictb        [_Toolbar("Edging", _Floating, _Hide, 10, 38, 1)]
ID_cicload      [_Button("Load My Lisp", cpyskt16.bmp,
cpyskt32.bmp)]^Z(load "cic.lsp")
ID_conswire     [_Button("Construct Wireframe", wrval16.bmp,
wrval32.bmp)]^C^C_cic_explode
ID_reconswire   [_Button("Rewire", cicwf16.bmp, cicwf32.bmp)]^C^C_rewire
ID_cicut        [_Button("Cutting", cicwf16.bmp,
cicwf32.bmp)]^C^C_cic_cut
ID_TBtlpth      [_Flyout("Toolpath Generation", cictp16.bmp,
cictp32.bmp, _OtherIcon, ACAD.EToolPath)]

**CIC_Polishing
ID_cictb        [_Toolbar("Polishing", _Floating, _Hide, 10, 38, 1)]
ID_consurf      [_Button("Construct Surface", survfa16.bmp,
survfa32.bmp)]^C^C_am2sf;_face
ID_consurAll    [_Button("Convert All", surval16.bmp,
surval32.bmp)]^C^C_am2sf;_objects;_all;
ID_consurAll    [_Button("My surfacing", cicsf16.bmp,
cicsf32.bmp)]^C^C_surface;
ID_TBtlpth      [_Flyout("Toolpath Generation", cictp16.bmp,
cictp32.bmp, _OtherIcon, ACAD.PToolPath)]
```

```
**EToolPath
**EDGING
ID_TbEdging     [_Toolbar("Edging Tool", _Floating, _Hide, 10, 280, 1)]
ID_tpLine       [_Button("Select a Line", cictp16.bmp,
cictp32.bmp)]^C^C_cic_line
ID_tpArc        [_Button("Select an Arc", cicsf16.bmp,
cicsf32.bmp)]^C^C_cic_arc
ID_tpCir        [_Button("Select a Circle", cictp16.bmp,
cictp32.bmp)]^C^C_cic_circle
ID_tpint        [_Button("Select an Intersection", cictp16.bmp,
cictp32.bmp)]^C^C_cic_intersec

**PToolPath
**POLISHING
ID_TbPolishing  [_Toolbar("Polishing Tool", _Floating, _Hide, 10, 280,
1)]
ID_planesurf    [_Button("Polish Plane Surface", cictp16.bmp,
cictp32.bmp)]^C^C_cic_planesurf
ID_cylsurf      [_Button("Polish Cylindrical Surface", cictp16.bmp,
cictp32.bmp)]^C^C_cic_cylsurf


ID_Toolpath     [Reconstruct 3D model and find toolpath for Yamaha]
ID_Wireframe    [Construct wireframe]
ID_Rewireframe  [Reconstruct wireframe]
ID_cicut        [Cutting Entity]
ID_Surface      [Reconstruct surfaces]
ID_GToolpath    [Generate toolpath]
ID_TAbout       [About the toolpath generation]
ID_cictb        [CIC Toolbar]
ID_conswire     [Explode command. Please Explode the solid twice.]
ID_reconswire   [Reconstructing wire]
ID_consurf      [Select solid objects to convert into surfaces]
ID_consurfALL   [Convert all into surfaces]
ID_TBtlpth      [Generate toolpath]
ID_TbEdging     [Find the tool path for Edging]
ID_tpLine       [Generate the toolpath for a straight line]
ID_tpArc        [Generate the toolpath for an arc or ellipse arc]
ID_tpCir        [Generate the toolpath for a circle or an ellipse]
ID_tpint        [Generate the toolpath for an intersection curve]

ID_cicload      [Load cic Lisp file]
ID_planesurf    [Polishing plane surface]
ID_cylsurf      [Polishing cylindrical surface]
ID_TbPolishing  [Find the tool path for Polishing]
```

ACAD.LSP (in AutoCAD) or MCAD.LSP (in Mechanical Desktop)

```
(setvar "cmdecho" 0)
(command "layer" "n" "Tool" "c" "5" "Tool" "")
(command "layer" "n" "Arrow" "c" "1" "Arrow" "")

(load "cic.lsp")
(PRINC)
```

Appendix 2

Modules of the CAM Package

Modules written in AutoLISP:

1. CIC.LSP: The function of this file is to load in all the developed CAM modules in AutoCAD platform. This is file is located in the subdirectory c:\mcad\win. It can not be loaded unless the following AutoLISP code is written in mcad.lsp file, which is load automatically every time when MCAD is launched.

(load "cic.lsp")

Same as in mcad.lsp, the cic.lsp is constructed as:

(load "c:\\filter\\debug\\lisp.lsp")

(load "c:\\mcad\\win\\cic_dialog.lsp")

(load "c:\\autolisp\\mathbox.lsp")

(load "c:\\autolisp\\cic_line.lsp")

(load "c:\\autolisp\\cic_circle.lsp")

The file listed in cic.lsp are those modules which construct the tool path generation part of the CAM package. Each of them defines a specific command for user to call in the MCAD platform, such as cic_line, cic_circle and etc. Once the cic.lsp is loaded in, all those commands could be called to implement their unique tasks.

2. LISP.LSP: This is the AutoLISP file that is generated by the filter. It will re-construct the 3D model so that the database of every line, arc, circle, intersection curve and surfaces could be accessed. Similar reconstruction can be done by using AutoCAD

command "explode" and MCAD command "am2sf". However, the resulting database can not be conveniently accessed because of AutoCAD's unique database structure. It may be convenient in 3D modeling but not for our case. Hence, a C program will do the filtering and automatically write a AutoLISP sequence to reconstruct the 3D model. This has been detailed in the previous section.

3. MATHBOX.LSP: It consists of many subroutines for vector and matrix calculation. So far, it has functions as follows:

(rond real_arg): function to round a real argument to the closest integer.

Example:

(rond -5.5) returns -6

(rond -5.499) returns -5

(mk_vect pt1 pt2); function to generate a vector given the end points in cartesian coordinates.

Example: (mk_vec '(2 3 4) '(1 5 7)) return (-1 2 3)

(mk_unit_vect v1): function to generate a unit vector from a given vector.

Example: (mk_unit_vect (-1 2 3) returns (-0.26726 0.53452 0.80178)

(mk_i_line ent_det): function to make the unit vector in the direction of a given line, the input ent_det list should be formatted as "LINE" <st pt> <end pt>, the output is the unit direction vector in WCS.

Example: (mk_i_line ("line" '(2 3 4) '(1 5 7))) returns (-0.26726 0.53452 0.80178)

(shl v1): function to shift a list left by one, place appending the erstwhile first atom as the last atom of the new list.

Example: (shl '(2 3 4)) returns (3 4 2)

(shr v1): function to shift right by one, place with the last atom becoming the first element.

Example: (shr '(2 3 4)) returns (4 2 3)

(x_prod v1 v2): function for vector cross products.

(rotrans pnt Xdir Ydir Zdir Origin): function to do the rotation transformation for a point from some ucs to wcs. Arguments are: ucs coordinates pnt A (x_{ucs}, y_{ucs}, z_{ucs}),, ucs unit vector ***X, Y, Z***, which are (x_X, y_X, z_X), (x_Y, y_Y, z_Y), (x_Z, y_Z, z_Z), and ucs original point ***P*** (x_p, y_p, z_p), returning the wcs coordinates (x_{wcs}, y_{wcs}, z_{wcs}),. In fact, this function is doing a homogeneous transformation as follows,

$$\begin{bmatrix} x_{wcs} \\ y_{wcs} \\ z_{ucs} \end{bmatrix} = \begin{bmatrix} x_X & x_Y & x_Z \\ y_X & y_Y & y_Z \\ z_X & z_Y & z_Z \end{bmatrix} \begin{bmatrix} x_{ucs} \\ y_{ucs} \\ z_{ucs} \end{bmatrix} + \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$$

(invmat v1 v2 v3): function to inverse a 3x3 matrix, return the inversed matrix.

4. CIC_LINE.LSP: In this module, a cic_line command is defined and will be later called in the MCAD platform. This function call enables the user to pick one of the lines in the 3D model that is to be processed and generate tool path based on user's request.

5. CIC_CIRCLE.LSP: This module generates a circular tool path. The defined command is 'cic_circle'. Not only could it do the circles, it also applies to arcs. The structure of this module is quite similar to CIC_LINE.LSP, however, since the circle or arc itself could form a plane and the local z axis is then defined.

6. CIC_INTERSEC.LSP: Tool path generation for an intersection curve is defined here. The command name is "cic_intersec". This program can recognize open spline or close spline itself. When it is dealing with close spline, AutoCAD command is used to break the close spline curve into two open spline curve and then generate the tool path.

7. CIC_PLANE.LSP: This module generates tool path for a plane surface. Two straight edges should be specified such that the plane is then defined within the two straight lines. In the cases of irregular shape of plane where difficulties involved in selecting two straight lines, users can create or build two straight lines themselves by using AutoCAD techniques.

8. CIC_CYL.LSP: This is the module that generates tool path for cylindrical surface. The polishing direction should be parallel to the cylinder axis and along the surface.