National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre reference

Our file Notre reference

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

CNC Tool Path Generation for Finish Milling Covex Pockets

A thesis submitted to
the School of Graduate Studies and Research
in partial fulfillment of the requirements for the
degree of Master of Applied Science in
Mechanical Engineering

By
Deepak Manuel

Ottawa-Carleton Institute for
Mechanical and Aeronautical Engineering
University of Ottawa
Ottawa, Ontario, Canada

ISBN 0-612-07853-1

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ABSTRACT

Production efficiency has increased considerably due to the use of computer-numerical controlled (CNC) machines. To machine a component on a CNC machine, part programs which define the cutting tool path are needed. This tool path is usually planned from computer-aided design (CAD) drawings, and converted to a computer-aided manufacturing (CAM) machine input format. However, several production bottlenecks are encountered in this transformation process, leading to escalated production costs and time. This problem can be attributed to two major causes. The first issue is the mode by which the CAD data is converted into CAM data and the second one is the efficiency or optimality of the physical process itself in producing an error-free machined surface.

Considering these drawbacks, an automatic tool path generating system to machine $2^1/_2$ dimensional arbitrarily shaped convex pockets was developed, in which tool path planning is done without the aid of CAD modellers or data exchange formats. This proposed methodology generates a spiral-like tool path by dynamically computing optimal offsets of the pocket boundary contour based on the type and size of the milling cutter, the geometry of the pocket contour and surface finish tolerance requirements.

This part programming system is PC based and simultaneously generates a G-code file, a verification file for simulating the generated tool path on a CAD package and a result file for research purposes.

ACKNOWLEDGEMENTS

I would like to express my genuine appreciation and indebtedness to my supervisor Dr.M.Liang for his valuable guidance and supervision of my graduate studies.

My sincere acknowledgements also go to the members of my defence committee, Dr.B.S.Dhillon and Dr.G.Kardos, for sparing their valuable time and effort to examine my thesis.

I would also like to honestly acknowledge all the help extended to me by my colleagues and friends at the University of Ottawa, Canada.

Most of all, I am eternally grateful to my parents Dr.P.R.Manuel and Dr.May Manuel for their encouragement and support during my entire graduate studies, especially during a very strenuous phase in my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Terminology

Pocket : Refers to a planar area, in which the top and bottom boundary surfaces are enclosed by straight lines and/or circular arc segments.

$2^1/_2$ D : Type of path control in which 2 axes are continuous-path controlled and the third axis is point to point controlled.

Boundary segments : Refers to individual straight lines and circular arcs that make up the contour.

Offset segments : Refers to individual straight lines and circular arcs at an offset distance from the original segment.

Vertex point : Intersection point of the boundary segments or offset segments forming the corners of the contour or offset loops.

Root Loop : The innermost offset loop at the optimal offset distance which has all the geometric orientations of the original contour.

Shadow Loop : A valid offset loop produced by offsetting the root loop with the tool radius.

Nomenclature

x	x coordinate
y	y coordinate
θ	vertex angle
α	angle from x axis
h	x coordinate of circle centre
k	y coordinate of circle centre
a	radius of circle
m	slope of segment
d	offset distance
l	Euclidean distance
i	loop index number
j	segment index number
e	cusp height
R	major radius of milling cutter
r	minor radius of generalized milling cutter
T_c	tool wear compensation factor

Dedicated to

My Lord and Saviour for his countless blessings

CHAPTER 1

INTRODUCTION

1.1 Foreword on tool path planning

Manufacturing process generally consists of two stages. First is the design or conceptual stage, and second is the production or realization stage. With the advent of computers into manufacturing, these stages evolved into what is presently known as computer-aided design (CAD) and computer-aided manufacture (CAM). Computer-aided process planning (CAPP) is a discipline existing between these two stages, and is responsible for determining the machining processes and sequences, to transform the raw material into a finished component. Computer-numerical control (CNC) part programming is the procedure of planning and documenting the sequence of processing steps to be performed on a CNC machine, and tool path planning is the process of determining the path through which the cutting traverses the workpiece.

There are two main types of CNC part programming, they are manual part programming and computer-assisted part programming. The manual method of part programming requires a great deal of human skill, imaginative ability and a sound

technical background. It is also very time consuming, tedious, and subject to various human errors. In the fast changing world that we are in today, it is important that we make use of computers to run CNC machines up to their utmost limits without overly depending on system based, format based or package based part programming method. Since its inception, the major goal in NC programming research has been to develop a method which will simplify part programming, and therefore, the methodology by which the control data is manipulated plays a very crucial role in tool path planning.

Typically, in today's automated manufacturing environment, a large amount of time is spent for transforming design models into product models. Most often, the planning time exceeds the actual execution time. As path planning accounts for a sizeable portion of the planning time, it is highly desirable to have simple and efficient CNC part programming systems which will eventually lead to reduced production costs and production time. Traditionally, path planning methods undergo various processing stages to convert the format of the surface geometry data into an input format for the machine tool. Because of this, the data processing cycle time is increased manyfold resulting in increased operational economics. Also, in conventional tool path planning methods it has been found that offsetting distances are mostly approximated. This is due to inadequate capabilities in CAD programs to determine angles in contours containing circular arc segments. This induces machining errors, tolerance violations, and increased machining time.

To alleviate these problems the industrial, commercial, and the academic community are in general working together towards system independent and format independent CNC milling tool path planning methods, which apart from automatically generating the milling tool paths, will be able to handle all general types of features without restrictions on the types of surfaces, and would be ideally done very efficiently and optimally under specified tolerances.

1.2 Proposed study

In view of the above drawbacks in generating tool paths, particularly for finish machining, the following work was proposed

1. Develop a new tool path generating procedure which consists of
    a) Determining and selecting angles subtended by pocket contour segments
    b) Adapting to various cutting tool topologies
    c) Computing optimal offset distances dynamically to avoid machining errors and maintain tolerances
    d) Optimizing the use of machine controller commands
    e) Generating the output in multiple formats for machining, verification and research
2. Implement the proposed tool path generating procedure on a PC
3. Demonstrate the application of the proposed tool path generating procedure with illustrative examples

1.3 Organization of Thesis

A general introduction to CNC tool path planning and a brief summary of the proposed research work are provided in this chapter. Chapter 2 gives a brief insight into CNC machining and part programming concepts. The literature review for this research work is presented in Chapter 3 and focusses on part programming, tool path planning methods and the state of the art today. Chapter 4 contains the detailed description of the proposed path planning method and a brief description of the system implementation. Chapter 5 demonstrates the application of the system using two example problems and analyzes the proposed path planning methodology. Chapter 6 concludes this thesis by presenting the limitations and advantages of the proposed method and scope of future work. Appendices have also been provided to give an insight into a few properties on analytic geometry, and listings of the source code.

CHAPTER 2

CONCEPTS Of CNC MACHINING

This chapter has been presented to give the reader a better understanding of the concepts of CNC machining and part programming. It starts by giving a brief description of the types of CNC machines and their classification. CNC controllers, machine codes and part programming are also discussed in relevance to the proposed work. An inside view of traditional CNC machining methodologies and their limitations along with possible alleviations to these problems using the proposed method are presented herein.

2.1 CNC machines

Types of CNC machines

Numerically controlled machine tools or NC machines form the backbone in modern automated manufacturing environments like computer integrated manufacturing systems (CIM) and flexible manufacturing systems (FMS). DNC (direct numerical control) machining is the process of controlling multiple machining centres simultaneously using a central computer, and CNC (computer numerical control) machining is the process of

using a local computer to control the motion of a cutting tool in a machining centre. Here on the term NC would synonymously refer to CNC machining in this work.

The most common types of NC machines are NC lathes and NC milling machines. However, other machines like NC drilling machines and NC punching machines etc do exist. In reality NC milling machines far outnumber all other NC machines. This is due to the fact that the majority of manufactured mechanical components can be machined using a milling machine.

Classification of NC milling machines

In an NC milling machine the relative motion between the cutting tool and the workpiece is defined by distances which correspond to the X Y Z axis of a Cartesian coordinate system. NC milling machines are therefore classified by the number of simultaneously or independently controllable axis in them. Commercial NC milling machines are available with multiple controllable axis. However, the most common types are the 2 axis or 2D (two dimensional) and the 3 axis or 3D (three dimensional) continuous-path controlled machines. In a 2 axis machine, linear and circular movement interpolations can be done on two axes only. In a 3 axis machine, linear movement interpolation can be done using all three axes and circular movement interpolation can be done on two axes only. However, some NC machines may also have helical and spherical interpolation support. In between 2D and 3D there exists a hybrid called $2\ ^1/_2$ D in which 2 axes are continuous-path controlled and the third axis is point to point controlled.

NC machine controllers

The relative movement between cutting tool and workpiece is achieved by linear motion of the machine table. The machine table has leadscrews driven by servo motors which in turn are controlled by an NC controller. Typical NC controller units consist of a computer complete with hardware and software to convert part programs into machine tool movements with the aid of transducers. Part programs are essentially blocks of instructions for the controllers to translate into machine tool movements and also set other auxiliary machine control functions. NC controllers are capable of entering, storing, retrieving, editing, and executing part programs. These NC machine controllers can be operated in a manual mode, an interactive mode, or an automatic mode.

NC machine codes

Part programs consist of lines of information written in a coded fashion. These coded lines are interpreted by the NC machine controller which then translates them into machine tool movements and also sets the auxiliary machine control functions. The most important basic codes are the G-codes which describe all tool movements, interpolation modes, operation unit modes, compensations, limits etc. Other part programming codes are F-codes for feed rates, S-codes for spindle speed rates, T-codes for cutting tool selections, and M-codes for auxiliary functions. Putting together all these codes in a planned sequence to carry out a specific machining task is known as G-code part programming.

2.2 NC machining

Part programming and path planning

NC part programming is the procedure by which the sequence of processing steps to be performed on an NC machine is planned and documented. Tool path planning is the specification of the cutting tool motion relative to the workpiece. The simplest way of part programming is done manually using G-codes. Part programs can also be produced by conversational graphic programs in which the programmer defines the workpiece and tool path interactively to produce the G-codes.

Most computer assisted part programming methods generally consist of two stages called pre-processing and post-processing. In the pre-processing stage, geometric data is input and machining surfaces are defined sequentially in a CAD program. This data is then exported as a format file in available formats like Initial Graphics Exchange Specifications (IGES), STandard for Exchange of Product data (STEP), etc. Tool path planning is then carried out with this data and output as an APT file. Then, process parameters are added to this APT file and converted to produce a cutter location or CL file. CAM and integrated CAD/CAM programs use the part definition from CAD to directly output an APT file or a CL (cutter location) file in the pre-processing stage. In the final or post-processing stage, auxiliary operating parameters such as cutting speeds, feed rates etc are then added to the CL file using another program to produce a G-code file.

Limitations of traditional NC machining methods

Between part programming and actual machining there usually is a verifying stage where a part program is checked for errors. The main types of part programming errors are geometric or primary errors and auxiliary or secondary errors. Geometric errors are basically tool path planning errors like wrong input of data in CAD, improper tolerance, improper cutting tool selection, etc. Auxiliary errors are basically setup errors like unsuitable machining parameters (e.g., speed and feed), improper positioning, wrong modes of operation, etc. Although some verifying methods include part program editing with options to test cut or dry run the program, they are limited only to auxiliary errors. Primary errors have to be changed at the source and the whole procedure of pre-processing and post-processing have to be undertaken again.

In practice part programming time may far exceed actual machining time often leading to increased manufacturing time and production costs. Most of this is attributed to the intermediate processing stages in path planning where the raw data is transformed to a machine acceptable form. The academic and the industrial community are endeavouring to alleviate this bottleneck between CAD and CAM.

As can be seen from Figure 2.1, because the proposed path planning method does not make use of any solid modellers and is independent of any data formats, it will skip the pre-processing and post-processing stages of part programming thus narrowing the gap between CAD and CAM.

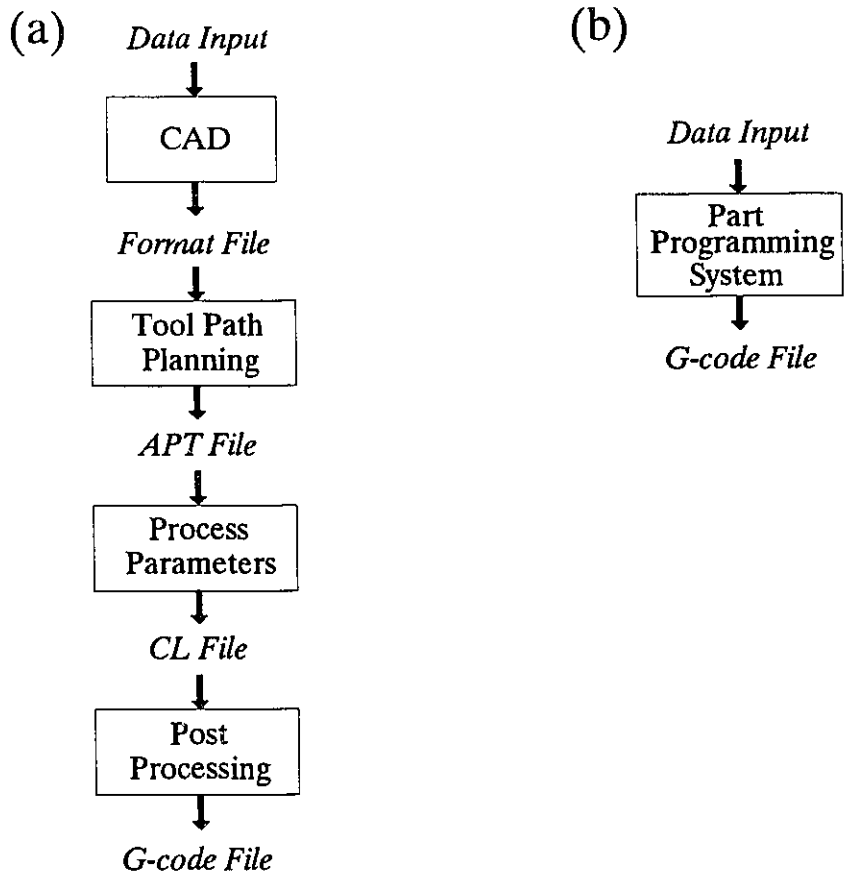


Figure 2.1 Part programming procedure (a) Conventional system (b) Proposed system

CHAPTER 3

LITERATURE SURVEY

In the present day, CAD and CAM have become an integral part of the manufacturing industry. Although they exist as separate disciplines they are interdependent on each other. According to GREENWOOD (1989), tool path generation is one of the most important functions in the integration of CAD/CAM and is vital for the survival of the manufacturing industry. Since production costs and time are directly influenced by this important link, it has been the endeavour of process planners to come out with tool path planning methods which are simple, fast, reliable, flexible, cost effective and obey high level technological criteria.

As the major effort of this thesis is directed towards automatic generation of NC milling tool paths for finish milling of arbitrary convex pockets, it was necessary to have a comprehensive literature survey of the global picture of part programming and the state of the art of tool path planning itself. The topics reviewed in this chapter are tool path planning techniques, strategies for NC milling, NC pocket milling, NC finish machining, and tool path optimization.

3.1 Tool path planning techniques

According to YAU and MENQ (1991), in automatic tool path generation, the main objective is to obtain efficient cutter location data within allowable machining error. Although various classifications of tool path planning techniques exist, KIM (1993) classifies them into three main types as APT (Automatically Programmed Tools) based, Parametric and Offset parametric plane based, and Cartesian plane based techniques

3.1.1 APT based techniques

As mentioned by SOMASUNDARAM and RAMAN (1992), MIT started the research with APT in 1952, and was soon followed by others like KRAL (1986) with various derivatives such as EXAPT (Extended Subset of APT). A typical APT part program consists of geometry statements, motion statements, postprocessor statements, and auxiliary statements.

GROOVER and ZIMMERS (1984) define this by stating that in order to generate the cutter path, the part programmer has to specify the required tool motion with respect to part geometry based on the concept of using a part surface, a drive surface, and a check surface. The idea is to move the tool with its side tangential to the drive surface and its bottom on the part surface until it reaches the check surface. Specific tasks such as pocketing can be carried out using separate macros in the programme. However, it should be pointed out that these high-level macros have very little flexibility and cannot be modified to incorporate high level tool path optimization criteria.

SATYANARAYANA *et al* (1990) developed a programming system similar to APT to generate CNC blocks. Although the authors introduced a few more types of point, line and circle definitions and the system could efficiently handle contours and pockets, it was extremely interactive and this could increase the path planning time.

3.1.2 Parametric and offset parametric based techniques

In this method tool path planning is done on the parametric space. BOBROW (1985) explains this concept as moving the tool between pre-determined equally spaced points on the U and V lines (parallel and tangential to the surface). Cutter location or CL points (x,y,z) are then evaluated from this U-V domain and used for tool path generation.

Another method called offset parametric planning is done by using CL data which is obtained by the calculation of the surface normal unit vector of each cutter contact or CC point and then offsetting it in the direction of the normal vector by tool radius. These CL points on the U-V domain are then used to generate the tool path. However, as pointed out by CHEN *et al* (1993), very time consuming filler and filter operations have to be performed on the parametric data sets to insert insufficient points or remove excess points before the tool path is generated .

3.1.3 Cartesian plane based techniques

This is the simplest and most direct tool path planning method. Tool paths are planned on the x-y axis of a Cartesian coordinate system. As explained by CHOI (1988), the

cutter path is the intersection of the part surface and a vertical plane perpendicular to the x-y plane. CL points are directly extracted from such intersection loci and used for tool path generation.

As mentioned by LAI and WANG (1994), the Cartesian method is also suitable for compound surface machining, as all cutting paths planned on the x-y plane of the Cartesian coordinates are parallel to each other. This is unlike isoparametric machining where each surface has its own variables making it difficult to generate a unified cutter path for compound surfaces. However, it should be noted that detecting and eliminating gouging points is not a trivial task in the Cartesian method of tool path planning.

3.2 Strategies for NC milling

NC milling of surfaces is planned using one of the two major strategies which are direction parallel milling, and contour parallel milling.

3.2.1 Direction parallel milling

Direction parallel milling (also known as zigzag milling or staircase milling) is carried out by moving the cutting tool along a series of offsets parallel to the longest edge of the boundary. At the intersection of the parallel offsets with the boundary contour, the cutter follows the periphery of the boundary contour until it can continue a parallel offset pattern again in an alternate direction.

This strategy works well for face milling simply connected flat surfaces. The study made by HEGRON (1988) shows that there is no significant difference in using other strategies. Further, when applied to boundaries with complex shapes and islands it could involve undesirable tool retractions. As defined by BRUCKNER (1982), tool paths are generated by repeatedly computing and ordering the intersections of a reference line with the boundary along with the tool overhang. The only technological problem inherently connected with this strategy is the determination of the optimal inclination of the reference line.

WANG *et al* (1987) reported that the cutting orientation in staircase milling produces a large impact on the length of cut. They found the average variations for most milling operations to be in the order of 5% to 10% and stated that the selection of the start point significantly affects the tool path.

This strategy is widely applied due to the fact that only one single-offset generation is required. However, from a technological point of view, it is definitively disadvantageous since the tool cuts alternatively with and against the spindle direction.

3.2.2 Contour parallel milling

Contour parallel milling (also known as window milling or spiral milling) is carried out by moving the cutting tool along offset curves equidistant to the boundary contour, and then stepping for the next pass in a concentric fashion. These concentric passes of the

original boundary contour can also be combined to make the cutter follow a spiral-like or window-frame pattern.

PERSSON (1978) is among the first who developed a pioneering method for machining arbitrarily shaped boundary contours consisting of straight lines and circular arcs. It consisted of determining areas belonging to the boundary of the pocket. These areas are formed by the bisectors to the boundary contours which meet at each point on the boundary. After the bisector skeleton is produced, the tool path is then generated from the innermost point of the skeleton. The tool path is composed of curves which are parallel to the corresponding side of the boundary pocket. Unfortunately details are not available about the actual work itself.

SUH and LEE (1990) then developed a procedure to finish cut a boundary profile with a convex or concave free surface bounded by lines, circular arcs and free curves. The cutter location data are then computed directly with better computational efficiency than normal without using an iterative method. However the whole work is assumed to have been modelled on a B-spline based modeller developed by the authors in their laboratory.

Along these lines, HELD *et al* (1994) recently have carried out more general work on this strategy based on proximity maps. Using the concept of computational geometry, i.e. Voronoi diagrams and monotonic pouches, offsets for tool path planning are

generated based on certain technological requirements. It is claimed that the computations are accomplished in $O(n)$ order, provided the Voronoi diagram is available. However, generating the Voronoi diagram is a very complicated task and thus might be a bottleneck for more practical applications of this technique.

3.3 NC Pocket milling

A very large percentage of mechanical parts like dies have boundary contours which are either parallel or normal to the x-y plane and by convention are classified as $2^1/_2$ dimensional or pockets. According to HARENBROOK (1980), more than 80% of all mechanical parts to be machined can be cut using $2^1/_2$D path control. As far as literature is concerned, there seem to be very few serious investigations into the topic of pocket machining, which could be due to the subtle and complicated algorithmic problems.

PREISS (1990) states that pocketing includes many complex situations. It is for this reason very few automatic pocketing computations succeed to solve all cases of pockets. This point is further emphasized by WOODWARK (1986) while discussing automatic offsetting, which is just a part of the automatic pocketing problem, by mentioning that "Detecting and rectifying all these problems automatically are very difficult to implement reliably". These are similar to the computer graphics problems discussed by BLINN (1988) who writes about them as "no picnic". The whole situation is summed up by PREISS (1990) who says that automatic pocketing is a dramatic example of murphy's law, "What can go wrong, will go wrong".

Some innovative studies in the area of pocket machining include the work by PREISS and KAPLANSKY (1985), KIM and JEONG (1995), and TSAI *et al* (1991). PREISS and KAPLANSKY (1985) used artificial intelligence (AI) techniques to generate optimum tool paths. The objective function in their model is based on material that must be removed by the tool at each point, and also on technological data associated with the material. It should however be noted that the tool path is said to be optimum by the authors because the non-cutting motion or air-cutting is minimized.

KIM and JEONG (1995) proposed a procedure for generating tool paths for machining free-form pockets with multiple islands and boundaries defined by closed cubic B-spline curves. It should be mentioned that the stepover or offset distance used is an approximation and not the optimum value which could lead to machining errors. Similarly, TSAI *et al* (1991) developed a physical model of the cutting process, and generated tool paths from Voronoi diagrams which resulted in maximum material removal rate. However the generation of Voronoi diagrams is itself a non trivial task.

3.4 NC Finish machining

Although the topic of rough machining has been addressed extensively in the literature, there seems to be a lack of insight into the problem of finish machining. The major concerns in finish machining tool path planning are to avoid overcutting and undercutting, gouging, and adhering to specified tolerance levels.

Some noteworthy work in the area of finish machining has been done by CUI (1993) who discusses a method to calculate manufacturing errors and apply the results to the selection of a proper tool path to obtain a good surface finish. However, as only zigzag and terraced machining errors are considered, this work could be a good starting point for further research.

Further, LAI and WANG (1994) developed a prototype system for compound surface modelling, in which cutter location or CL points for finish machining are determined directly from the offset surfaces to alleviate the gouging problem. However, the undercutting problem was not considered in their method.

3.5 Tool path optimization

The research into tool path optimization was started by WANG *et al* (1987), and followed by PRABHU *et al* (1990) and LAKKARAJU *et al* (1992), whose work mainly focussed on the optimization of the tool path length for convex polygons with straight-line edges..

VICKERS and BRADLEY (1992) came up with some application oriented research to reduce machining time by maximizing the number of circularly interpolated moves for surface generation. YEUNG and WALTON (1992) developed a heuristic to approximate curves with arc splines thus achieving data point reduction to alleviate the problem of data overflow caused by excess storage in NC controllers.

3.6 Observations

The following observations were made from this literature review. Regarding path planning techniques, According to CHEN *et al* (1993), APT has some inherent disadvantages with reference to computational time. Since it is based on the Newton's iteration technique there are difficulties in locating tool tip positions which do not always converge to a solution. Parametric planning seems to be ideal for sculptured surfaces and a lot of work has been done in this area. However, there is much more scope for research into problems like gouging and spacing of the UV lines. Cartesian coordinate planning is ideal since it is simple and the fact that current NC machine controllers are based on the Cartesian system.

As for tool path strategies, the zigzag approach is suitable for simple surfaces and rough machining as it requires minimum calculations and path planning skills. As analyzed by WANG *et al* (1987), the window strategy although requiring more computations is quite efficient in terms of computational time. It is also highly flexible to optimization criteria like tolerancing and produces reasonably consistent results.

Although some work has been done on data exchange formats like IGES and STEP, these data exchange processes are undesirable as they are generally very time consuming, and errors could creep in during the various transfer processes. In modern shop floor, it is preferred to have a format free tool path generating system.

3.7 Motivation

NC tool path planning has come a long way since its conception. With the advent of powerful computers with highly developed hardware and software configurations, shop floor practice has been leaning towards tool path planning methods which are more system dependant or package dependant. This could be an unhealthy trend since each system or software package has its own advantages and drawbacks and some systems or software packages may be soon replaced by their more competitive rivals. In view of this, this thesis will be directed towards the development of a tool path planning system which is independent of solid modellers and formats. In particular, the study will focus on finish milling of $2^1/_2$ D pockets due to the inadequacy of the effort on this topic, and the fact that pocket milling is a very common application in NC milling. The contour milling strategy was chosen as it produces very consistent tool paths with low computational times, and also because practical factors like orientation of the work piece or the shape of the object do not have any impact on the tool path.

CHAPTER 4

PROPOSED METHODOLOGY FOR TOOL PATH GENERATION

This chapter presents the methodology and algorithms for the automatic generation of NC part programs with optimised tool paths for finish milling to meet the objectives discussed in section 3.7 of Chapter 3. It also contains the relevant flow chart, definitions, terminology, notations, assumptions, and algorithms, used in the method.

4.1 Methodology

The methodology consists of generating a spiral-like tool path by dynamically computing optimal offsets of the machining surface boundary contour based on the type and size of the milling cutter, the geometry of the pocket contour and surface finish tolerance requirements.

The procedure consists of defining the contour elements of the pocket with equations, and classifying them as positive and negative elements depending on the

offsetting direction. The optimum offsetting distance that avoids undercutting at the vertices is then determined. The calculation of this offset is based on the size and geometry of the milling cutter, the angles between the boundary elements of the pocket contour, and the surface finish tolerance requirements specified in terms of the maximum allowable cusp error. Equations of the contour elements at this optimal offset distance are then formed and solved to get the intersection points in the offset loop. This loop offsetting procedure is then repeated till a terminal condition is reached. The surface is then checked for any unmachined area. The first segment or leading edge of the innermost loop is then determined and the centre or start point of the tool path is then calculated and located on this edge. New intersection points are then calculated by solving the equations of the last segment or trailing edge of a loop with the leading edge of the previous loop. The corresponding old intersection points are then replaced with the new intersection points to eliminate tool retractions and maximize on the circular interpolation command and form the spiral-like tool path. This whole tool path generation procedure as shown in Figure 4.1 is explained in detail as follows.

Geometric restrictions on machining surface

- The boundary contour consists of straight line and circular arc segments as most NC milling machines support only linear and circular interpolation commands
- The internal angles between adjacent boundary contour segments $\leq 180^{\circ}$ i.e. the machining area is a convex pocket
- All coordinates of the pocket contour lie in the first quadrant

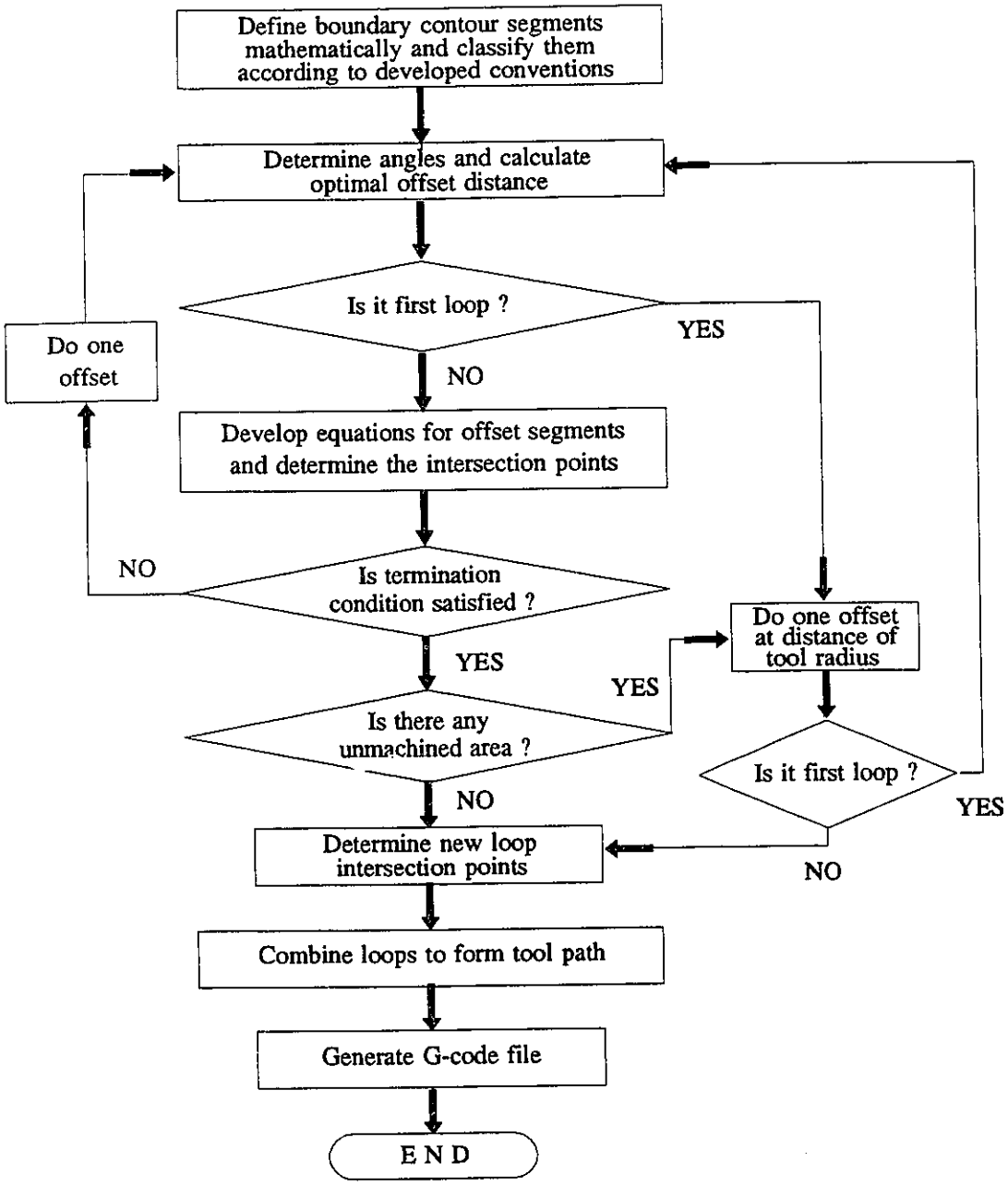


Figure 4.1 **Structure of the proposed tool path generation method**

4.2 Mathematical definition of the machining surface

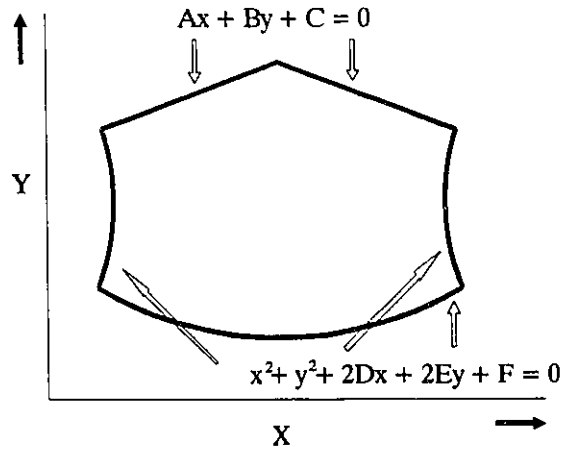


Figure 4.2 Mathematical representation of contour

The first step in tool path planning is to define the machining surface with some form of mathematical representation. Since the proposed method is based on analytic geometry, the pocket boundary contour is defined as a closed loop comprising of lines and circular arc segments which are represented by equations in the standard form. As shown in Figure 4.2, straight line segments are defined by equations in the standard form as:

$$Ax + By + C = 0$$

and circular arc segments are defined by equations in the standard form as:

$$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

4.3 Classification of boundary contour segments

Theoretically, all straight lines and circular arcs can be offset in two directions. However, only one direction is to be selected. To avoid the ambiguity, it becomes

necessary to clearly specify the intended offsetting direction for each segment relative to its position in the contour. It is for this reason that in this part programming method all contour segments are first classified and indexed as positive or negative segments depending on their offsetting direction. These classifications are defined by conventions which were developed based on analytic geometry and are discussed in detail below.

4.3.1 Classification procedure for straight line segments

Straight line segments in the pocket contour are classified as positive and negative segments based on the intended direction of offset. This initializing property will hold true for any particular line segment over its entire offsetting surface. Referring to Figure 4.3.1, the rules for classification are given below.

1) Given two end points (x_1, y_1), (x_2, y_2) of a straight line, the equation in a two point form is obtained by

$$(y - y_1) = \frac{(y_2 - y_1)}{(x_2 - x_1)} (x - x_1)$$

This line is represented in the standard form as:

$$Ax + By + C = 0$$

2) Obtain the equations of the lines which are parallel to the given line and at a perpendicular distance of d (the offset distance)

$$\textit{Offset distance} \quad \pm d = \frac{Ax + By + C}{\pm \sqrt{A^2 + B^2}}$$

(The sign of the denominator is chosen opposite to that of C, and $d=R$ for the initial loop)

3) From the given contour determine if the required offset line and the origin are lying on opposite sides of the given line or on the same side of the given line.

4) If the required offset line and the origin are on the same side of the given line then a negative d is chosen and the given line is classified as a negative line segment and vice-versa.

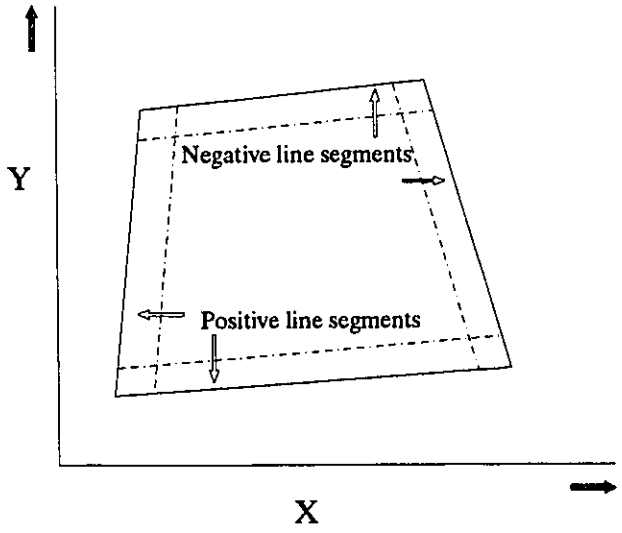


Figure 4.3.1 Classification of line segments

Special Cases

This method can also handle lines with the following special conditions

$x = A$	Lines parallel to Y axis	(slope $=\infty$)
$y = B$	Lines parallel to X axis	(slope = 0)

$Ax + By = 0$ Lines passing through the origin

(Refer *Algorithm for slopes* on page 41)

4.3.2 Classification procedure for circular arc segments

Offsetting a circular arc segment implies that the centre is fixed and only the radius is changed depending on the intended direction of offset. The classification is carried out in three steps: Determine the two possible centre points, select the intended centre point, and classify the arc segment. Details are explained below.

Determining the centre points

For any two given two end points (x_1, y_1), (x_2, y_2) and radius a of a circular arc, there exist two possible centres (h_1, k_1) and (h_2, k_2). They are determined as follows. First the general form equations at the two points are constructed.

$$(x_1 - h)^2 + (y_1 - k)^2 = a^2 \qquad \text{---- (1)}$$

$$(x_2 - h)^2 + (y_2 - k)^2 = a^2 \qquad \text{---- (2)}$$

Equating (1) and (2) and solving the resulting quadratic equation for h and k gives two roots or centre points (h_1, k_1), (h_2, k_2) (Figure 4.3.2)

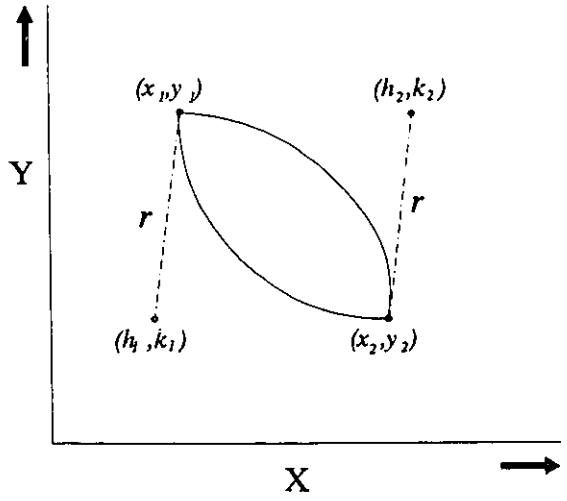


Fig 4.3.2 Possible centre points

Choosing the correct centre

As seen above, when a circular arc is represented by its two end points and radius, the resulting quadratic equation will give two possible centre points. As only one of them represents the intended centre of the given arc segment, the following procedure is followed to obtain the correct centre.

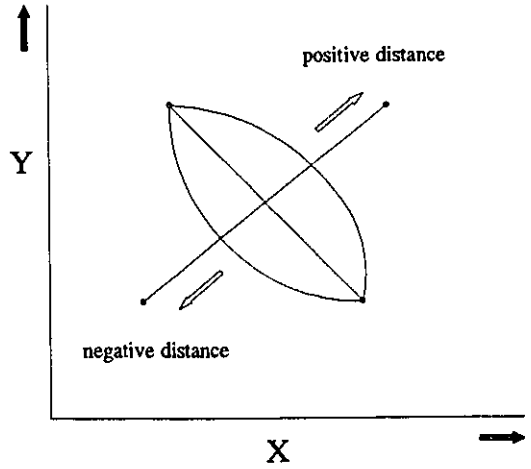


Figure 4.3.3 Choosing the correct centre

1) From the given two points and radius for the arc obtain the two possible centre points (h_1, k_1), (h_2, k_2).

2) Construct the equation of the chord and calculate the perpendicular distances between the chord and the two centres (The perpendicular distances will have the same magnitude but opposite signs).

perpendicular distance from $(h_1, k_1) = \dfrac{A h_1 + B k_1 + C}{\pm \sqrt{A^2 + B^2}}$

perpendicular distance from $(h_2, k_2) = \dfrac{A h_2 + B k_2 + C}{\pm \sqrt{A^2 + B^2}}$

3) From the given contour determine whether the required centre and the origin are lying on the same side or opposite sides of the chord.

4) Referring to Figure 4.3.3, if the required centre and the origin are lying on opposite sides of the chord then the centre which gives a positive distance is chosen as the correct centre and vice-versa.

Classifying arc segment

An arc segment can be classified by comparing its radius with that of the arc from which the offset starts. If the required offset segment needs a smaller radius as compared with the original radius, then the segment is classified as a negative arc segment, otherwise

the segment is classified as a positive arc segment. (Figure 4.3.4).

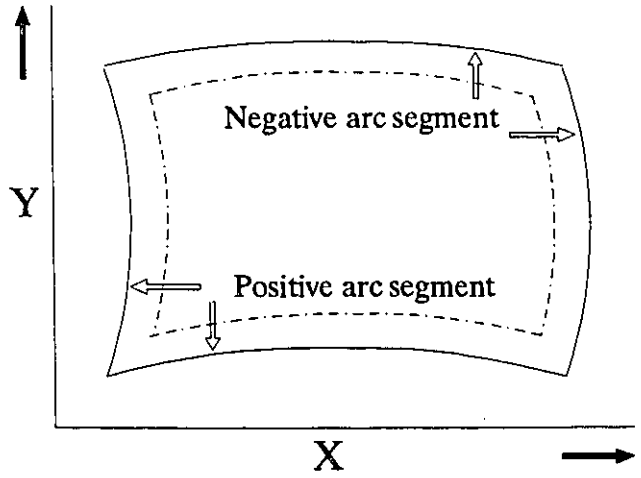


Figure 4.3.4 Classification of arc segments

The circular arc segment is then represented in the standard form as:

$$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

where $D = -h$, $E = -k$, and $F = h^2 + k^2 - r^2$

4.4 Determination of vertex angles

To optimize tool paths, an important task is the dynamic calculation of included angles at vertices, which constitute the contour, irrespective of the type of segments and their combinations. This also incorporates the identification of all possible positive and negative included angles, and selection of the appropriate included angle. Surprisingly, as pointed out by CHOU and YANG (1991), even the most advanced CAD/CAM packages, such as CATIA (1985), SmartCAM (1988), and Unigraphics (1988), do not have features to handle this problem.

The importance of the dynamic determination of included angles lies in their impact on the critical offset, which is defined as the maximum cross-feed length that can be achieved without violating specified tolerances. The offset distance is a function of the angle formed between adjacent segments on the boundary contour. As the maximum error occurs at the vertex subtending the minimum angle, all vertex angles of the contour are determined and the minimum angle is chosen for the calculation of the critical offset.

Angles are calculated using the general slope formula

$$\tan \theta = \frac{\tan \alpha_1 - \tan \alpha_2}{1 + \tan \alpha_1 \tan \alpha_2}$$

where

α_1 = angle of the first adjacent segment with respect to X axis

α_2 = angle of the first adjacent segment with respect to X axis

The angle at a line-line vertex can be found by direct substitution of the above formula. The angles for line-arc, arc-line, and arc-arc vertices are determined as follows. Using analytic geometry, two slopes for a curve at its start point and end point can be determined. By applying this method, the slope of the tangents to the curve at these points are then obtained. These are then substituted in the general slope formula to get the included angle (Figure 4.4).

However, the following special conditions may occur.

1) Lines or tangents parallel to the X axis. (slope = ∞)

i.e., when $(x_2 - x_1) = 0$

Then the value of $\alpha = 90$

2) Lines or tangents parallel to the Y axis. (slope = 0)

i.e., when $(y_2 - y_1) = 0$

Then the value of $\alpha = 0$

3) When the lines or tangents are perpendicular to each other

i.e., when $(\tan \alpha_1 * \tan \alpha_2) = -1$

Then the value of $\theta = 90$

Also, when lines and tangents intersect each other, a major angle and a minor angle are formed, which may include negative angles. As only one of these angles correspond to the contour vertex, it becomes imminent to select the correct angle. All these special conditions have been incorporated in this path planning method and are detailed in *Algorithm for slopes* and *Algorithm for angles.*

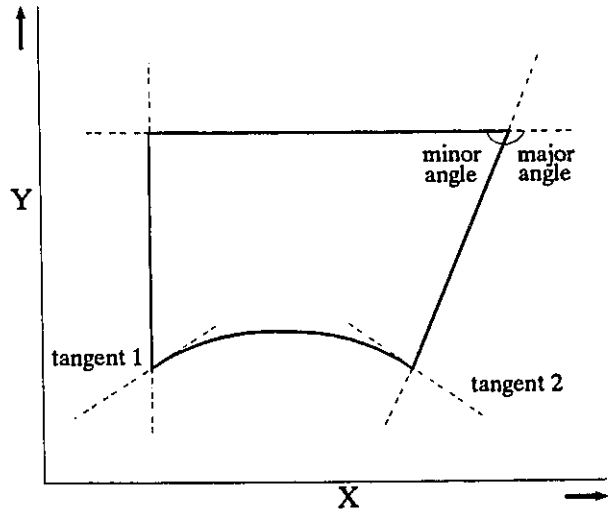


Figure 4.4 Determination of vertex angles

Calculation of vertex angles for possible combinations of boundary segments like Line-Line, Line-Arc, Arc-Line, and Arc-Arc are detailed below

Line - Line combination

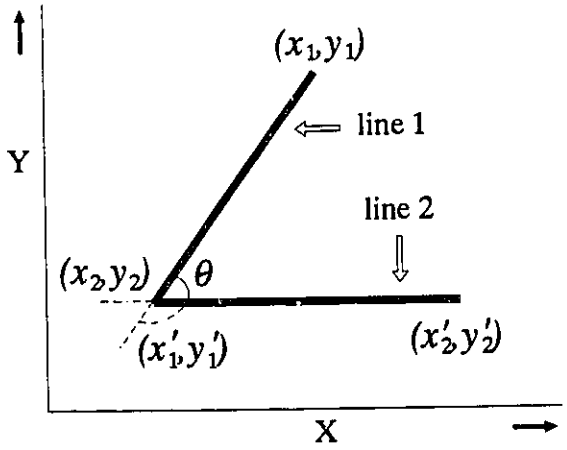


Figure 4.4.1 Line-Line vertex angle

The vertex angle for a line-line combination is determined using the formula

$$\theta = \tan^{-1}\left(\frac{\frac{(y_2 - y_1)}{(x_2 - x_1)} - \frac{(y_2' - y_1')}{(x_2' - x_1')}}{1 + \frac{(y_2 - y_1)}{(x_2 - x_1)} \frac{(y_2' - y_1')}{(x_2' - x_1')}} \right)$$

where

$\frac{(y_2 - y_1)}{(y_2 - y_1)}$ is the slope of line 1

$\frac{(y_2' - y_1')}{(x_2' - x_1')}$ is the slope of line 2

Line - Arc combination

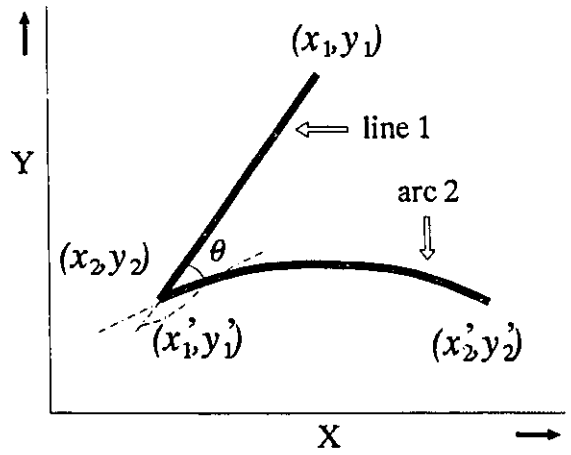


Figure 4.4.2 Line-Arc vertex angle

The vertex angle for a line-arc combination is determined using the formula

$$\theta = \tan^{-1}\left(\frac{\dfrac{(y_2 - y_1)}{(x_2 - x_1)} - \left(- \dfrac{x'_1 + h'}{y'_1 + k'} \right)}{1 + \dfrac{(y_2 - y_1)}{(x_2 - x_1)} \left(- \dfrac{x'_1 + h'}{y'_1 + k'} \right)} \right)$$

where

$\dfrac{(y_2 - y_1)}{(x_2 - x_1)}$ is the slope of line 1

$\left(- \dfrac{x'_1 + h'}{y'_1 + k'} \right)$ is the slope of tangent at the start point of arc 2

Arc - Line combination

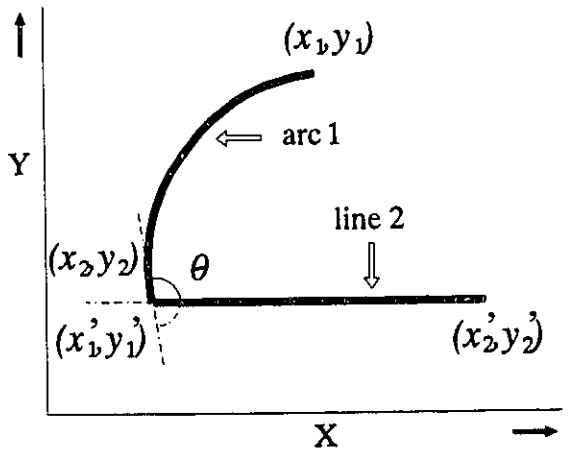


Figure 4.4.3 Arc-Line vertex angle

The vertex angle for an arc-line combination is determined using the formula

$$\theta = \tan^{-1}\left(\frac{\left(-\frac{x_2 + h}{y_2 + k} \right) - \frac{(y'_2 - y'_1)}{(x'_2 - x'_1)}}{1 + \left(-\frac{x_2 + h}{y_2 + k} \right) \frac{(y'_2 - y'_1)}{(x'_2 - x'_1)}} \right)$$

where

$\left(-\frac{x_2 + h}{y_2 + k} \right)$ is the slope of tangent at the end point of arc 1

$\frac{(y'_2 - y'_1)}{(x'_2 - x'_1)}$ is the slope of line 2

Arc - Arc combination

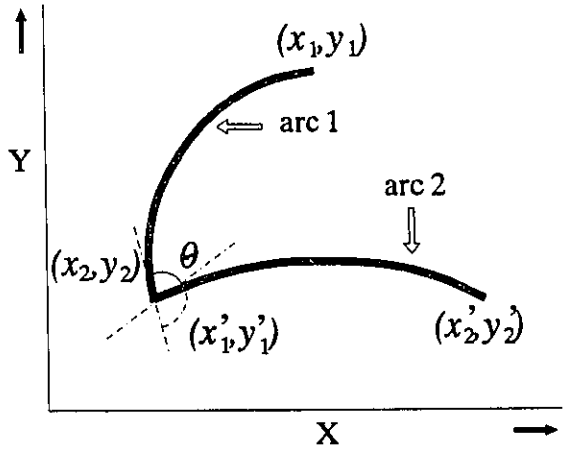


Figure 4.4.4 Arc-Arc vertex angle

The vertex angle for an arc-arc combination is determined using the formula

$$\theta = \tan^{-1}\left(\frac{\left(-\frac{x_2 + h}{y_2 + k} \right) - \left(-\frac{x_1' + h'}{y_1' + k'} \right)}{1 + \left(-\frac{x_2 + h}{y_2 + k} \right)\left(-\frac{x_1' + h'}{y_1' + k'} \right)} \right)$$

where

$\left(-\frac{x_2 + h}{y_2 + k} \right)$ is the slope of tangent at the end point of arc 1

$\left(-\frac{x_1' + h'}{y_1' + k'} \right)$ is the slope of tangent at the start point of arc 2

Algorithm for slopes

IF segment = line

IF $x_1 - x_2 = 0$

slope = 90

ELSE slope = $\frac{(y_2 - y_1)}{(x_2 - x_1)}$

ENDIF

IF segment = arc

IF $y_1 - k = 0$

first slope = 90

ELSE first slope = $-\frac{(h - x_1)}{(k - y_1)}$

IF $y_2 - k = 0$

second slope = 90

ELSE second slope = $-\frac{(h - x_2)}{(k - y_2)}$

ENDIF

Algorithm for angles

IF Vertex = Line-Line

m_1 = slope of line 1 and m_2 = slope of line 2

IF Vertex = Line-Arc

m_1 = slope of line 1 and m_2 = first slope of arc 2

IF Vertex = Arc-Line

m_1 = second slope of arc 1 and m_2 = slope of line 2

IF Vertex = Arc-Arc

m_1 = second slope of arc 1 and m_2 = first slope of arc 2

IF $m_1 = 90$ and $m_2 \neq 90$

Angle = $\left| \tan^{-1} m_2 - 90 \right|$

IF $m_1 \neq 90$ and $m_2 = 90$

Angle = $\tan^{-1} m_2 + 90$

IF $m_1 \neq 90$ and $m_2 \neq 90$

IF $m_1 * m_2 = -1$

Angle = 90

ELSE Angle = $\tan^{-1} \frac{m_1 - m_2}{1 + m_1 m_2}$

IF Angle < 0

Angle = Angle + 180

ENDIF

ENDIF

ENDIF

END

4.5 Machining errors

There are three main types of machining errors that usually occur in NC milling. They are *gouging*, *under/over cutting*, and *cusp or scallop errors*. Since the problem of gouging does not occur in planar milling of simply connected pockets, only the latter two types of errors are addressed in this tool path planning method

4.5.1 Undercutting and overcutting errors

Technically the maximum offset that could be achieved between two parallel passes in a tool path would be equal to the tool diameter. When the tool diameter is used as the offset distance it leads to the problem of undercutting in which small triangular shaped areas at the vertices remain unmachined (Figure 4.5.1). To ensure complete milling of a planar surface in a single pass without leaving unmachined areas between the vertices, the offset distance would have to be less than the tool diameter. The problem of

undercutting may be overcome by offsetting at a distance equal to the tool radius or by machining across the vertices in separate moves. However, this causes overcutting and often leads to poor surface finish. Based on the relationship between the geometry of the contour and the cutting tool, this proposed path planning method calculates the maximum or critical offset distance that can be achieved between parallel passes without leaving any area unmachined.

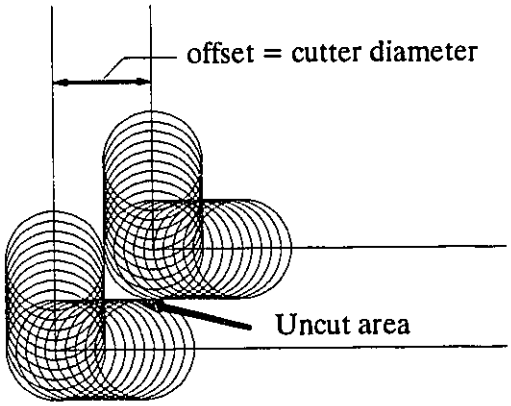


Figure 4.5.1 Undercutting errors

4.5.2 Cusp or scallop errors

Cusp errors are formed when ball-end milling cutters or generalized end milling cutters are used to machine surfaces. They can also be formed when end milling cutters are used at an inclination to the machining surface. These cusp errors occur between adjacent parallel passes in a tool path. Though these cusps remain constant along the parallel length of the pass, they vary at the vertices. From the geometric relationship between the contour and the cutter it can be seen that the maximum error occurs at the vertex subtending the minimum angle. The proposed path planning method uses this

maximum error value in the calculation of the critical offset so as to place a very tight upper bound on the specified tolerance.

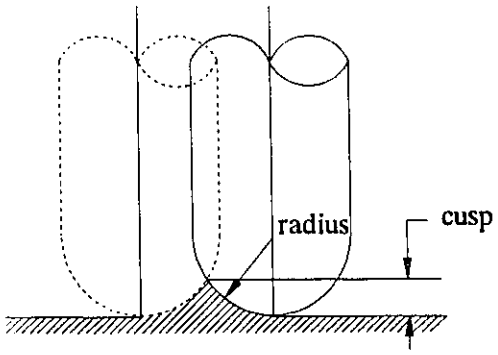


Figure 4.5.2 Cusp or Scallop errors

4.6 Calculating the optimal stepover distance

Since finish machining was the focus of this proposed tool path planning method, the critical offset, also referred to as the optimal stepover, was dynamically calculated as a function of the minimum vertex angle, milling cutter radius, and cusp height.

The most common types of milling cutters used in practice are flat end milling cutters, ball end milling cutters, and generalized end milling cutters. Ball end cutters are mostly used for machining curved surfaces, while generalized end cutters and flat end cutters are used for machining planar surfaces. However, all of them are interchangeable for different applications. Calculations of the critical offset for these milling cutters are detailed below.

Critical offset for end milling cutter

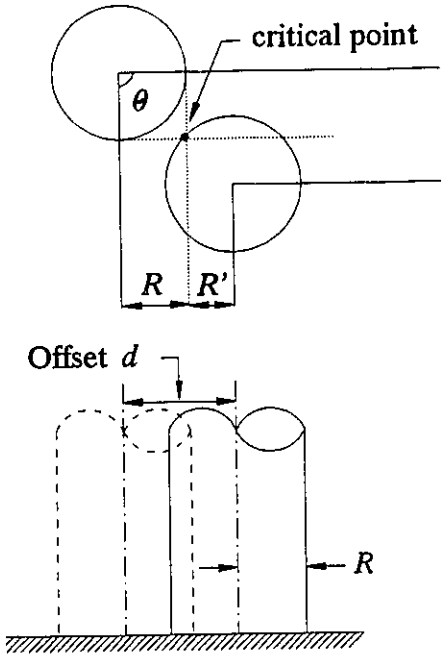


Figure 4.6.1 Flat end milling cutter

The tool geometry and vertex machining errors for a flat end milling cutter is shown in Figure 4.6.1. The critical offset is derived as follows:

$$R' = R \sin \frac{\theta}{2}$$

$$d = R \left(1 + \sin \frac{\theta}{2}\right)$$

where

R --- radius of milling cutter

θ --- vertex angle

Critical offset for ball end milling cutter

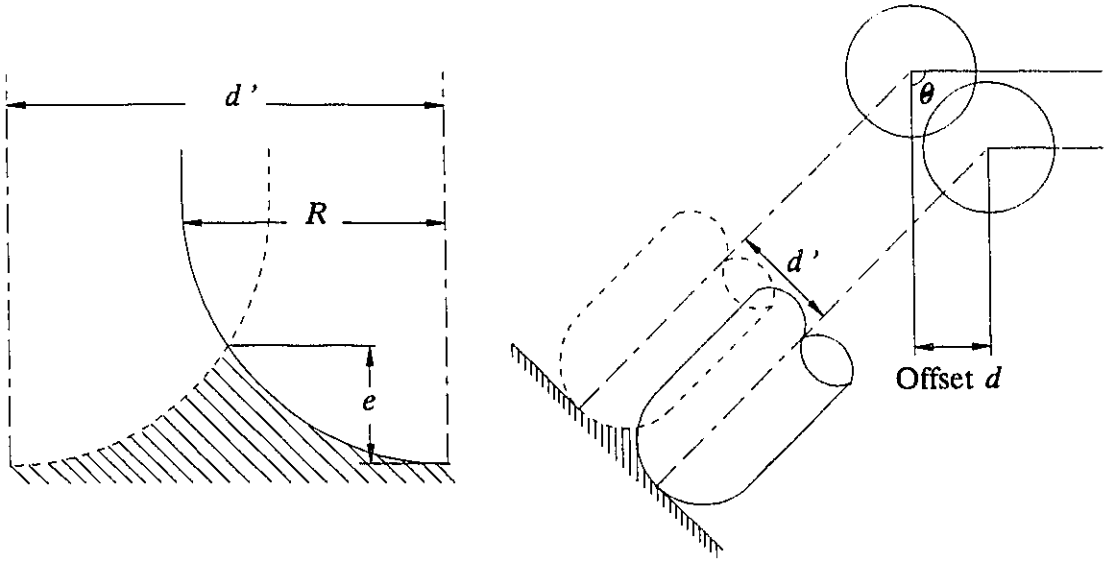


Figure 4.6.2 Ball end milling cutter

The tool geometry and vertex machining errors for a ball end milling cutter is shown in Figure 4.6.2. The critical offset is calculated as follows:

$$d' = 2\sqrt{e(2R-e)}$$

$$d = \left(2\sqrt{e(2R-e)}\right)\sin\frac{\theta}{2}$$

where

R --- radius of milling cutter

θ --- vertex angle

e --- cusp height

Critical offset for generalized end milling cutter

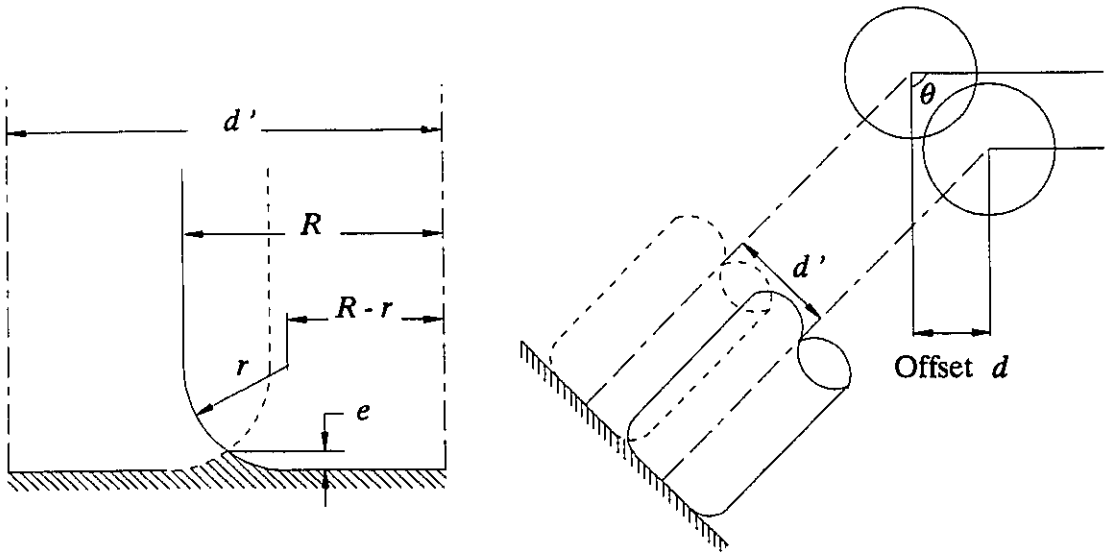


Fig 4.6.3 Generalized end milling cutter

The tool geometry and vertex machining errors for a Generalised end milling cutter is shown in Figure 4.6.3. The critical offset is given by

$$d' = 2\left(R - r + \sqrt{e(2r - e)}\right)$$

$$d = 2\left(R - r + \sqrt{e(2r - e)}\right)\sin\frac{\theta}{2}$$

where

R --- major radius of milling cutter

r --- minor radius of milling cutter

θ --- vertex angle

e --- cusp height

4.7 Tool size compensation factor

Typically all milling cutters are considered to be cylindrical, and their sizes are specified by the diameter. However, in reality these sizes may vary slightly from the specified values depending on the condition of the cutter and its application. Milling cutters used for machining hardened material and with new inserts are usually oversized, while regrinded cutters tend to be undersized which can cause overcutting or undercutting. This size deviation can be expressed as a percentage of the specified cutter diameter and be optionally compensated in the calculation of the critical offset.

$$\text{maximum offset} = T_c\, d$$

4.8 Developing equations for offset segments

Developing offset segment equations is the next step in this tool path generating procedure. To avoid gouging at the boundary walls, the first or initial offset loop is formed at a distance of r the tool radius. Other offset loops are formed at a distance of d the critical offset distance.

Equations of offset segments for forming the offset loops are given below.

Positive line $$\frac{Ax + By + C}{\pm \sqrt{A^2 + B^2}} = + d$$

where the sign of the denominator is chosen opposite to that of C

Positive arc $$(x - h)^2 + (y - k)^2 = (a + d)^2$$

Negative line $$\frac{Ax + By + C}{\pm\sqrt{A^2 + B^2}} = -d$$

where the sign of the denominator is chosen opposite to that of C

Negative arc $$(x - h)^2 + (y - k)^2 = (a - d)^2$$

Equations of offset segments for forming the initial and shadow loops are given below.

Positive line $$\frac{Ax + By + C}{\pm\sqrt{A^2 + B^2}} = +r$$

where the sign of the denominator is chosen opposite to that of C

Positive arc $$(x - h)^2 + (y - k)^2 = (a + r)^2$$

Negative line $$\frac{Ax + By + C}{\pm\sqrt{A^2 + B^2}} = -r$$

where the sign of the denominator is chosen opposite to that of C

Negative arc $$(x - h)^2 + (y - k)^2 = (a - r)^2$$

4.9 Determining the vertex points of offset loops

From the developed offset segments equations the vertex points of the offset contour are then calculated. These segment intersection points are determined by solving the respective pairs of equations in a cyclic anti-clockwise order. The possible combinations of contour offset segment intersections and the algorithms for determining the intersection points are given below.

Line - Line intersection

The offset segment equations for line 1 and line 2 from Figure 4.9.1 are

$$A_1 x + B_1 y + C_1 = 0$$

$$A_2 x + B_2 y + C_2 = 0$$

Solving the above equations simultaneously gives one intersection point (x , y)

Line - Arc and Arc - Line intersection

The offset segment equations for line 1 and arc 2 from Figure 4.9.2 are

$$Ax + By + C = 0$$

$$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

The offset segment equations for arc 1 and line 2 from Figure 4.9.3 are

$$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

$$Ax + By + C = 0$$

Solving the equations simultaneously gives two intersection points (x_1, y_1) and (x_2, y_2)

Arc - Arc intersection

The offset segment equations for arc 1 and arc 2 from Figure 4.9.4 are

$$x^2 + y^2 + 2D_1 x + 2E_1 y + F_1 = 0$$

$$x^2 + y^2 + 2D_2 x + 2E_2 y + F_2 = 0$$

Solving the equations simultaneously gives two intersection points (x_1, y_1) and (x_2, y_2)

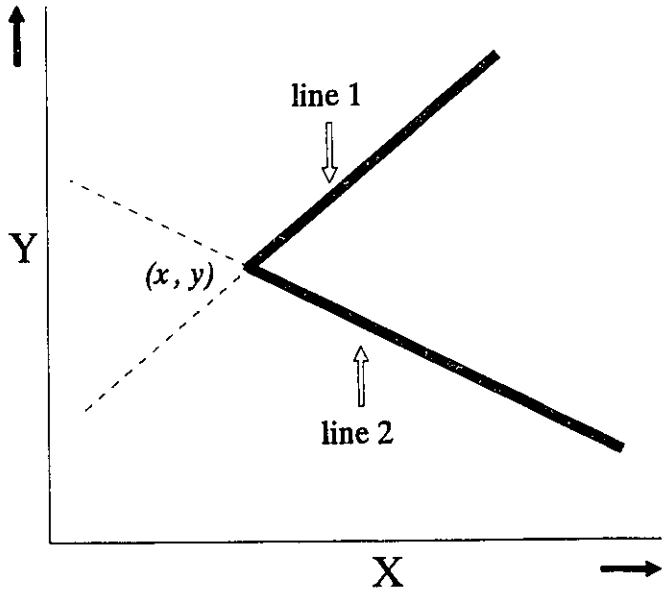


Figure 4.9.1 Line-Line intersection

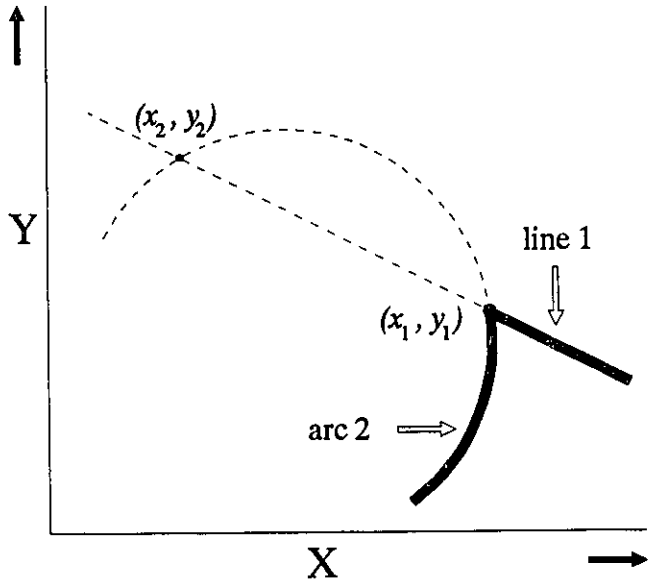


Figure 4.9.2 Line-Arc intersection

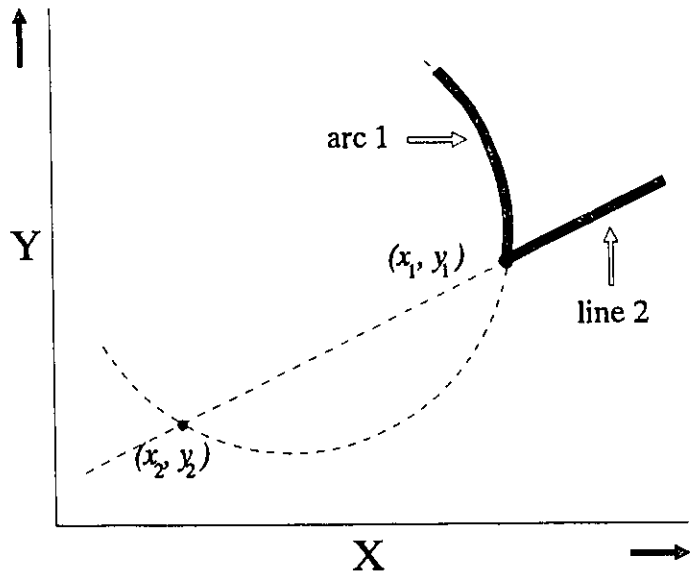


Figure 4.9.3 Arc-Line intersection

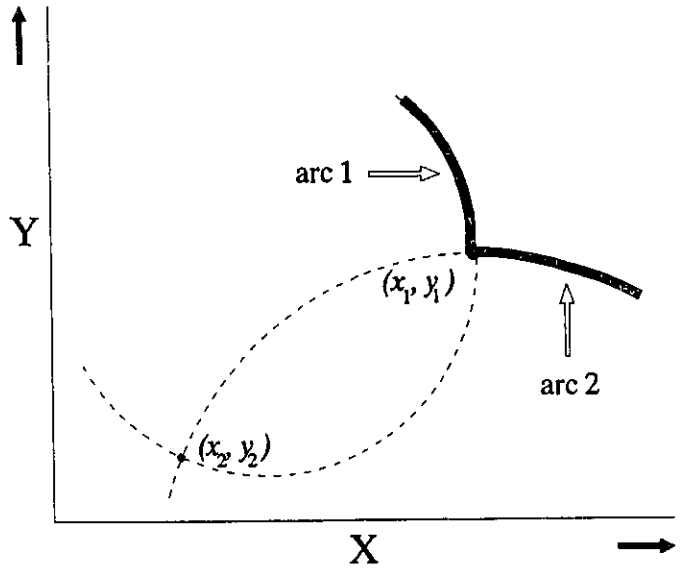


Figure 4.9.4 Arc-Arc intersection

4.10 Determining the correct intersection point

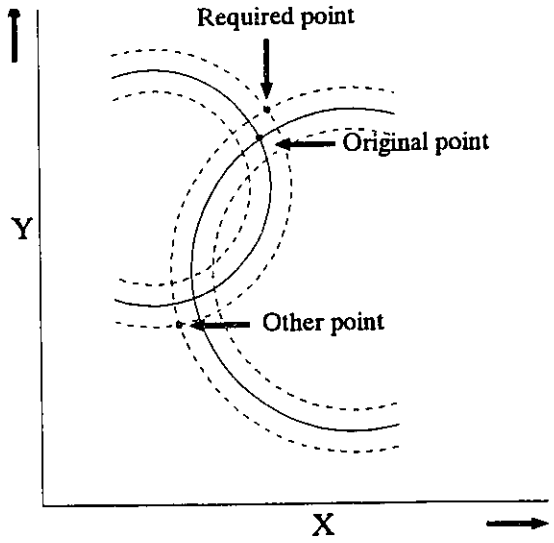


Figure 4.10 Correct intersection point

Offset equations of the second order are solved to get the vertex point of a line-arc, arc-line or an arc-arc combination. Since the solution to a quadratic gives two roots, it means that the given pair of segments intersect at two points, of which only one point is the required vertex point. These two points are equidistant from the original vertex only if the segments are tangential to each other i.e., they intersect at only one point. In all other cases the required point is always nearer to the original vertex point and lies on the bisector of the original angle. This property is used to determine the correct intersection point by calculating the distance from these two points to the original vertex point with the formula for distance given below, and then choosing the point which gives the minimum distance.

$$l = \sqrt{(x - x_1)^2 + (y - y_1)^2}$$

4.11 Dynamic Offsetting

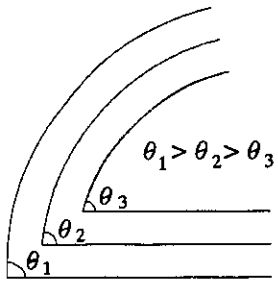


Figure 4.11.1 Arc-Line offset

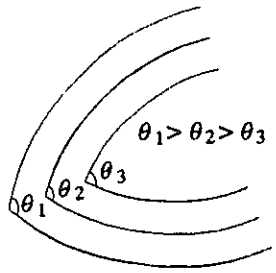


Figure 4.11.2 Arc-Arc offset

Dynamic offsetting of the contour boundary is the main feature of this tool path planning method. It can be seen that at the vertices containing arc segments the subtended angle varies with every offset (Figures 4.11.1 & 4.11.2). Unlike other methods where fixed offset distances are used leading to undercutting errors, this method computes the critical offset distance at every boundary offset thus ensuring that the problem of undercutting is totally eliminated and specified tolerances are maintained.

4.12 Termination criterion for offsetting

The dynamic offsetting procedure described earlier is repeated to get concentric offsets of the original boundary contour. Segments with a length less than *2r-d* are omitted from the calculations. This offsetting process is carried on till a termination condition is reached i.e., an invalid loop is formed. This termination state is identified by the following procedure. Every contour vertex is indexed and initialized with inequality conditions. i.e., the x and y coordinates of the starting point of a segment are indexed

as greater than, equal to, or less than the x and y coordinates of the ending points of that segment. After every offset loop is calculated, the new inequality conditions are checked for orientation. When the orientation does not correspond to the previous loop, it implies that an invalid loop is formed and offsetting is terminated. This criterion is illustrated in Figures 4.12.1 to 4.12.4. From Figure 4.12.3 it can be seen that the y coordinate values of point 3' and 4' are greater than 2' and 1' respectively. After offsetting through a distance of d, it can be seen from Figure 4.12.4 that the corresponding points 3" and 4" are less than 2" and 1" respectively.

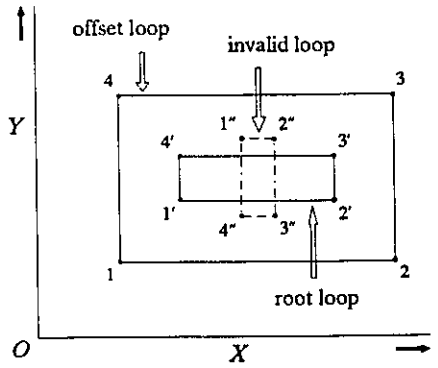


Figure 4.12.1 Termination condition

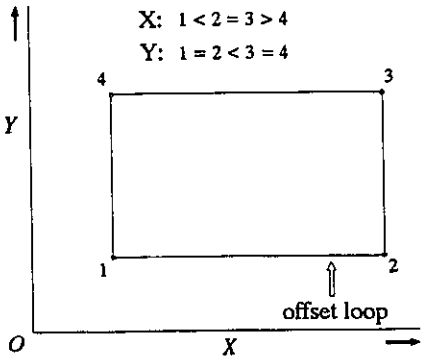


Figure 4.12.2 Offset loop

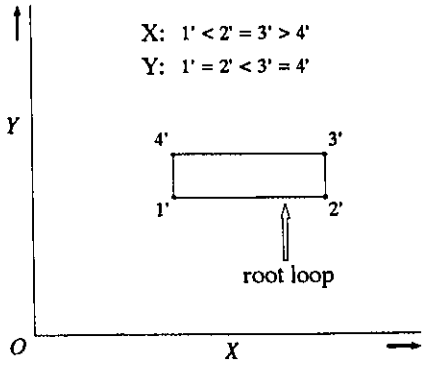


Figure 4.12.3 Root loop

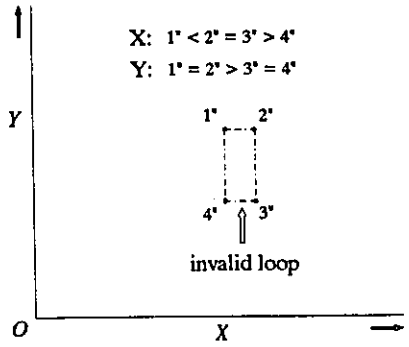


Figure 4.12.4 Shadow loop

Algorithm for Termination criterion

FOR i = 1

IF $x_1 > x_2$ { sequence x = 0 }
IF $x_1 < x_2$ { sequence x = 1 }
IF $x_1 = x_2$ { sequence x = 2 }
IF $y_1 > y_2$ { sequence x = 0 }
IF $y_1 < y_2$ { sequence x = 1 }
IF $y_1 = y_2$ { sequence x = 2 }
ENDIF

FOR i > 1

IF sequence x = 0
    IF $x_1 > x_2$ { } ELSE { offset = END }
IF sequence x = 1
    IF $x_1 < x_2$ { } ELSE { offset = END }
IF sequence x = 2
    IF $x_1 = x_2$ { } ELSE { offset = END }
IF sequence y = 0
    IF $y_1 > y_2$ { } ELSE { offset = END }
IF sequence y = 1
    IF $y_1 < y_2$ { } ELSE { offset = END }
IF sequence y = 2
    IF $y_1 = y_2$ { } ELSE { offset = END }
ENDIF

4.13 Checking for unmachined area

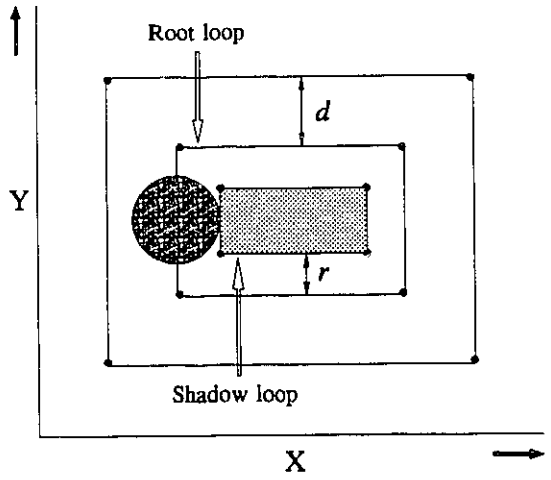


Figure 4.13 Checking for unmachined area

When the termination criterion is satisfied, i.e., when further offsetting at distance d produces an invalid loop, the following step is carried out to check for unmachined areas. As shown in Figure 4.13, the final or root loop is offset by the tool radius r and its orientation is checked with the initialized inequality conditions. If the sequence orientation is satisfied it implies that there is an unmachined area at the centre of the contour. The offset loop enclosing the unmachined area is called a shadow loop, and is then considered as the innermost loop. If the sequence orientation is not satisfied, the root loop is then taken to be the innermost loop of the contour offsets.

4.14 Location of tool path start point

The spiral-like shape tool path which is generated by this method has a centre point which will be used as the start point of the tool path (Figure 4.14). The presence of a

shadow loop implies that the cutter path which follows the root loop will not machine the area of the pocket represented by the shadow loop. In this case the centre is located at the starting point of the first segment of the shadow loop.

The absence of a shadow loop implies that the cutter path which follows the root loop will completely machine all areas of the pocket. In this case the centre of the tool path is located at the starting point of the first segment on the root loop.

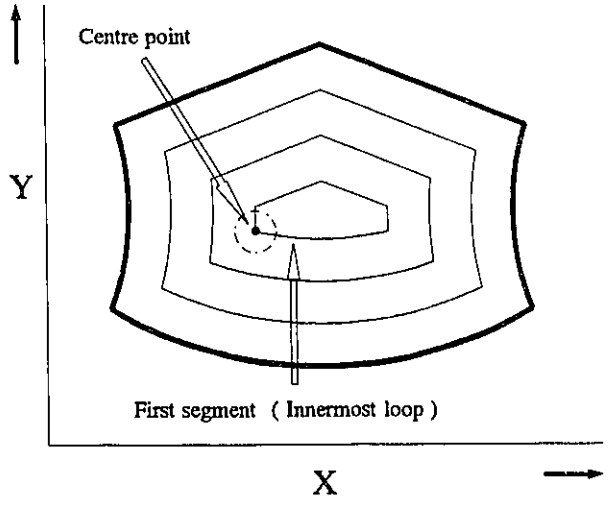


Figure 4.14 Start point of tool path

4.15 Formation of the spiral-like tool path

In planar machining it is necessary to have short linear tool path segments between corresponding vertices to remove uncut material or to move between adjacent loops. This would induce technical problems like increased machining time, reduced tool life, undesired tool retractions, inflated NC program size, and decreased quality of the

finished surface etc. The proposed path planning method overcomes these problems and also maximizes the use of the linear and circular interpolation commands by combining the individual contour loops to form a single spiral-like shaped tool path. This procedure is effected through calculation of new intersection points which are obtained by solving the equations of the trailing segments of offset loops with the leading segments of the previous offset loop (Figure 4.15). The old intersection points are then replaced with the corresponding new intersection points to form the spiral-like shaped final tool path.

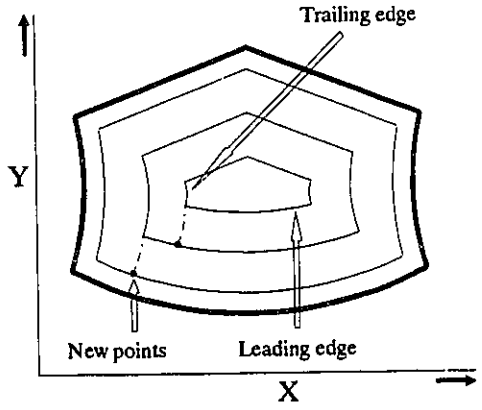


Figure 4.15 Formation of spiral-like tool path

4.16 Generating the output

This tool path planning procedure outputs a verification file, a standard G-code file, and a result file simultaneously. The verification file is to physically view the spiral-like tool path in a CAD package. The purpose is to reveal possible errors before actual cutting. The G-code file contains information about machining and auxiliary parameters. Once

the tool path has been verified, it can be run on an NC machine directly. The result file gives details about the generated tool path such as the number of offset loops, computational time for tool path generation, critical offset angles, etc. These data can be used for further analysis of the performance of the tool path.

4.17 System implementation

The tool path generation system based on the above components has been implemented and tested on a 486 PC following the structure shown in FIgure 4.1. The system is coded in C and the source code is attached in Appendix 2. The functional symbols of the system are also explained in Appendix 2. The screen prompt message is in italic font and the user input is in bold font. The application of the system will be illustrated in Chapter 5 using two examples

CHAPTER 5

APPLICATION AND DISCUSSIONS

This chapter consists of two sections. The first section illustrates the implementation of the tool path generation method developed in Chapter 4. Each sub-section of Chapter 4 is illustrated step by step simultaneously with an example. Although several hundred examples were generated with this method, an example of a symmetric pocket contour, and an example of an asymmetric pocket contour, consisting of all possible combinations of straight line segments and circular arc segments were selected as illustrations. The illustration section of the part programming method can be broadly classified into three main sub-headings as data input, segment intersections and dynamic offsetting, and offset termination and data output. The discussions section details the major advantages of the proposed part programming system.

The system was implemented, timed and analyzed on an IBM compatible 486 processor based PC. The screen message is depicted in italics and the user input is depicted by bold lettering.

5.1 Illustrative examples

Functional elements of the programming system

x[i][j] The first element denotes the loop number and the second element denotes the x coordinate of the Cartesian point

y[i][j] The first element denotes the loop number and the second element denotes the y coordinate of the Cartesian point

line[i][j][k] The first element denotes the loop number, the second element denotes the segment number, and the third element stores the coefficients of x and y, and the constant in that order

circle[i][j][k] The first element denotes the loop number, the second element denotes the segment number, and the third element stores the coefficients of x and y, the constant, the segment radius, x and y coordinates of the centre in that order

angle[j] Denotes the minimum of the vertex angles of every loop

line_slope[j] Denotes the slope of the line

arc_slope_1[j] Denotes the slope of an arc at its start point

arc_slope_2[j] Denotes the slope of an arc at its end point

index[j] Denotes the segment as a line or an arc

segment_combination[j] Denotes the type of combination of two segments at a vertex

sequence_x[j] Denotes sequence of the x coordinates as detailed in section 4.12

sequence_y[j] Denotes sequence of the y coordinates as detailed in section 4.12

offset_index[j] Denotes the direction of offset for the segments

vertex_points Denotes the number of vertices in the boundary contour

tool_index Denotes the type of milling cutter used

tool_radius Denotes the major radius of the milling cutter

minor_radius Denotes the minor radius of the milling cutter

offset Denotes the optimum offset for the contour loop

Example 1

(Symmetric shaped part, flat end milling cutter)

The illustrative pocket contour shown in Figure 5.1 consists of Line-Line, Line-Arc, Arc-Line, and Arc-Arc segment combinations. Line segments are denoted by their starting points and ending points, and circular arc segments are denoted by their starting points, ending points and radius. A flat end milling cutter of one inch was chosen for this example. Since the flat end milling cutter is used, the cusp problem does not exist and the major concern is to maximize material removal rate subject to the undercutting-free restriction. The step by step path generation procedure using the proposed system is described below.

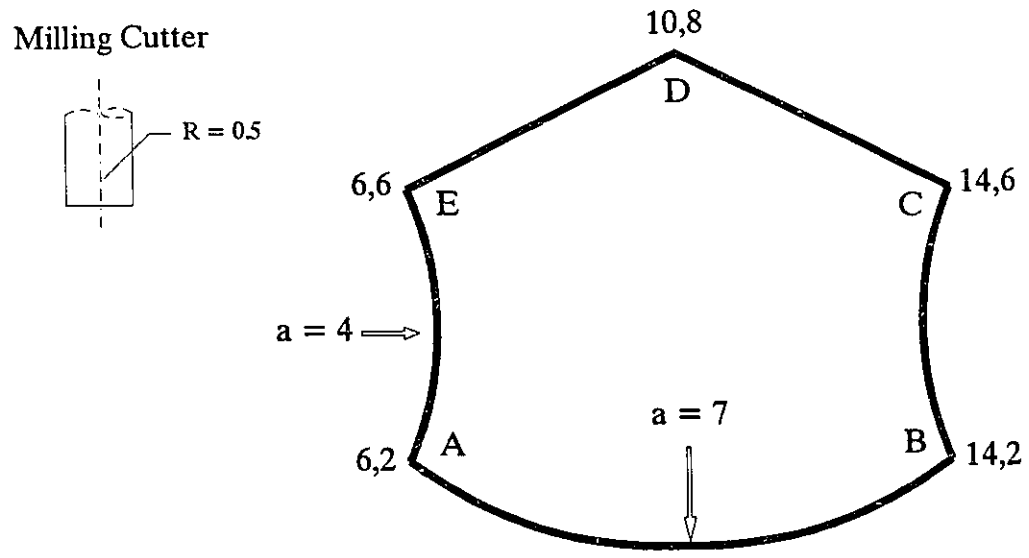


Figure 5.1 Geometric input data available for pocket example 1

Data input

In the following, the italic text represents screen message.

Input the total number of vertex points

5

Note that this data is stored as a constant called vertex_points. Following this, the computer prompts:

Input x & y values, and type & segment combination for each segment

for segment type: 0 means line and 1 means circle

for segment combination: 1-line-line, 2-line-arc, 3-arc-line and 4-arc-arc.

Here the segment type indicates the type of current boundary segment, either line or circle; segment combination means the combination of the current boundary segment and the immediate next boundary segment.

If the start vertex is (6,2), then the following data are typed:

6 2 1 4

14 2 1 3

14 6 0 1

10 8 0 2

6 6 1 4

Notice that the data are input in counter-clockwise sequence. The four numbers in the first line represent the x and y coordinates of the start vertex, segment type (circle in this case) and segment combination (arc-arc, i.e., arc AB and arc BC in this case). The data associated with the other four vertices are similarly defined and input. Then, the following prompt appears:

Input the value of the radius

7 *The centres are (10.00, 7.74) and (10.00. -3.744)*

4 *The centres are (17.46, 4.00) and (10.53. 4.00)*

4 *The centres are (9.46, 4.00) and (2.53. 4.00)*

Notice that the two circle centres for each input radius are immediately calculated and appear beside the user input radius. Next the user is requested to provide the sign for each segment. The sign is then used by the system to determine the correct arc centre. The message is:

Give the sign of the perpendicular distance to choose the correct centre of arc segment. Type 1 for positive distance and 0 for negative distance. This input has to be given by looking at the basic figure

1 *The centre is (10.00, 7.74)*

1 *The centre is (17.46, 4.00)*

0 *The centre is (2.53. 4.00)*

As shown above, the correct arc centre is determined and shown beside the sign type input by the user.

Corresponding to section 4.3.1 and 4.3.2, the system demands the user to classify the line or arc type. The prompts are shown below:

Give the sign of the distance for straight line segment #

Type 0 for negative and 1 for positive

Give the sign of the distance for circular arc segment #

Type 0 for negative and 1 for positive

Since the radius of the arc offset from arc AB will be smaller, the sign is 0. The radius of the arc offset from arc BC is increased as compared to that of arc BC, the sign is 1. For line segment CD, the sign is 0 since both the offset line and origin are on the same side of CD. Similarly the signs for segments DE and EA can be determined. Accordingly the input data are as follows:

0

1

0

0

1

This information is stored in an array called offset_index[ ]. With these data, the angles between segments are calculated as discussed in section 4.4 and detailed in *Algorithm for Angles*. The minimum angle is then determined and stored in Angle[ ]

Please specify the type of tool and its radius

1 - Flat end milling cutter,

2 - Ball end milling cutter. Specify tolerance

3 - Generalised end milling cutter. Specify tolerance and minor radius

1

0.5

Input operational unit mode. Type 0 for metric or 1 for imperial units

1

Specify spindle speed (RPM)

1500

Specify feed rate (m/min)

0.5

The input values are stored in tool_type, tool_radius, tolerance, and minor_radius. These values are then used to compute the optimal stepover distance as detailed in section 4.6 and written into offset. Sequence_x[ ] and sequence_y[ ] are also determined and initialized using the method shown in 4.12. The spindle speed and the feed rate inputs are user defined and are used in the generation of the G-code file.

Segment intersections and dynamic offsetting

This part of the program is the heart of the path planning method. It first constructs the equations of the offset segments as described in section 4.8. This is done using the values from Offset_index[ ],offset, line[ ][ ][ ] and circle[ ][ ][ ]. The resulting values are then updated into their respective arrays. From this updated data, the intersections of the offset segments are determined as explained in the *Algorithms for Intersection* from section 4.9. The correct intersection points or the vertices of the offset loop are then determined using the procedure shown in section 4.10. The dynamic offsetting procedure detailed in section 4.11 is then carried out to generate multiple concentric loops of the pocket boundary contour as shown in Figure 5.2 and tabulated in Table 5.1

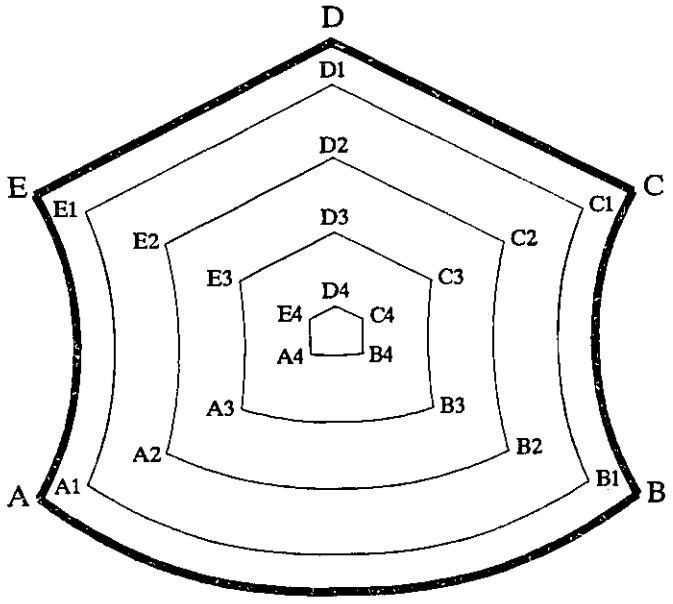


Figure 5.2 Contour offset loops with intersection points

Contour Loop	Minimum Vertex Angle in Degrees	Optimum Offset Distance	Vertex Point	Cartesian Coordinates
Boundary	86.56	0.5	A	6.00, 2.00
			B	14.00, 2.00
			C	14.00, 6.00
			D	10.00, 8.00
			E	6.00, 6.00
1	93.31	0.8636	A1	6.64, 2.17
			B1	13.35, 2.17
			C1	13.32, 5.77
			D1	10.00, 7.44
			E1	6.67, 5.77
2	98.74	0.8794	A2	7.71, 2.59
			B2	12.28, 2.59
			C2	12.27, 5.34
			D2	9.99, 6.47
			E2	7.72, 5.34
3	97.86	0.8769	A3	8.72, 3.16
			B3	11.27, 3.16
			C3	11.27, 4.85
			D3	9.99, 5.49
			E3	8.72, 4.85
4	94.13	0.8660	A4	9.65, 3.88
			B4	10.34, 3.88
			C4	10.35, 4.33
			D4	9.99, 4.51
			E4	9.64, 4.33

Table 5.1 Details of contour offset loops

Offset termination and data output

Dynamic offsetting is terminated when the termination condition discussed in section 4.12 is satisfied. The check for unmachined areas as illustrated in section 4.13 is then carried out using sequence_x[ ] and sequence_y[ ]. The centre of the tool path is then located depending on the conditions mentioned in section 4.14. From the data of line[ ][ ][ ] and circle[ ][ ][ ], and by using conditions from section 4.8 to 4.10, the concentric offset loops are combined to form the spiral-like tool path following the procedure given in section 4.15. The result is shown in Figure 5.3 and tabulated in Table 5.2.

The result of the part programming system is then output as a G-code file, an analysis file and a verification file for simulating the generated tool path on a CAD package (for this example AutoCAD R-12 is used).

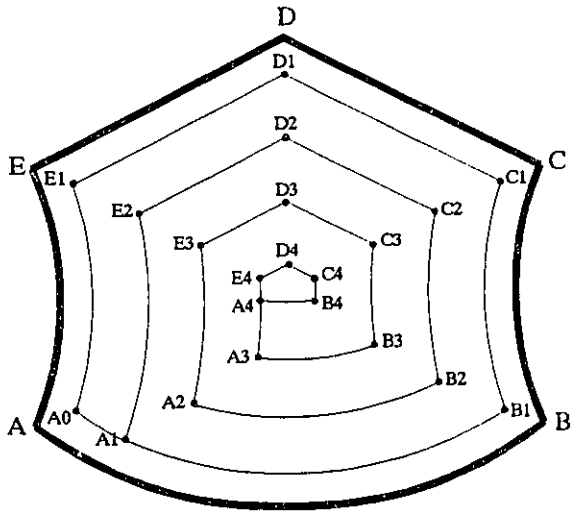


Figure 5.3 Final spiral-like tool path pattern

Final Tool path sequence: A4 - B4 - C4 - D4 - E4 - A3 - B3 - C3 - D3 - E3 - A2 -B2 - C2 - D2 - E2 - A1 - B1 - C1 - D1 - E1 - A0 - A1

Contour Loop	Vertex points	Original Coordinates	Replaced Coordinates	Final Coordinates
Innermost	A4	9.65, 3.88		9.65, 3.88
	B4	10.34, 3.88		10.34, 3.88
	C4	10.35, 4.33		10.35, 4.33
	D4	10.00, 4.51		10.00, 4.51
	E4	9.64, 4.33		9.64, 4.33
3	A3	8.72, 3.16	9.51, 3.03	9.51, 3.03
	B3	11.27, 3.16		11.27, 3.16
	C3	11.27, 4.85		11.27, 4.85
	D3	10.00, 5.49		10.00, 5.49
	E3	8.72, 4.85		8.72, 4.85
2	A2	7.71, 2.59	8.48, 2.32	8.48, 2.32
	B2	12.28, 2.59		12.28, 2.59
	C2	12.27, 5.34		12.27, 5.34
	D2	10.00, 6.47		10.00, 6.47
	E2	7.72, 5.34		7.72, 5.34
Initial	A1	6.65, 2.17	7.39, 1.78	7.39, 1.78
	B1	13.35, 2.17		13.35, 2.17
	C1	13.32, 5.77		13.32, 5.77
	D1	10.00, 7.44		10.00, 7.44
	E1	6.67, 5.77		6.67, 5.77
	A0	6.64, 2.17		6.64, 2.17
	A1	6.65, 2.17		7.39, 1.78

Table 5.2 Details of final tool path

Part program (G-code) output for Example 1

N0 G71

N10 G94 F 0.5

N20 G96 S 1500

N30 G00 X9.654985 Y3.880030

N40 G03 X10.345016 Y3.880031 R3.879902

N50 G02 X10.351921 Y4.335663 R7.120098

N60 G01 X10.000000 Y4.511624

N70 G01 X9.648080 Y4.335665

N80 G02 X9.586231 Y3.005717 R7.120098

N90 G03 X11.277418 Y3.162416 R4.756875

N100 G02 X11.279441 Y4.852389 R6.243125

N110 G01 X10.000000 Y5.492110

N120 G01 X8.720561 Y4.852390

N130 G02 X8.543043 Y2.299759 R6.243125

N140 G03 X12.288153 Y2.593546 R5.636365

N150 G02 X12.270582 Y5.340118 R5.363635

N160 G01 X10.000000 Y6.475410

N170 G01 X7.729419 Y5.340120

N180 G02 X7.418219 Y1.779296 R5.363635

N190 G03 X13.350854 Y2.174843 R6.500000

N200 G02 X13.329493 Y5.776237 R4.500000

N210 G01 X10.000000 Y7.440983

N220 G01 X6.670507 Y5.776237

N230 G02 X6.649146 Y2.174843 R4.500000

N240 G03 X7.418219 Y1.779296 R6.500000

N250 M00

Example 2

(Asymmetric shaped part, ball end milling cutter)

The asymmetric pocket contour example shown in Figure 5.4 consists of arbitrary Line-Line, Line-Arc, Arc-Line, and Arc-Arc segment combinations. Line segments are denoted by their starting points and ending points, and circular arc segments are denoted by their starting points, ending points and radius. A ball end milling cutter of radius 0.4, with a tolerance of 0.2 for the cusp error was chosen for this example.

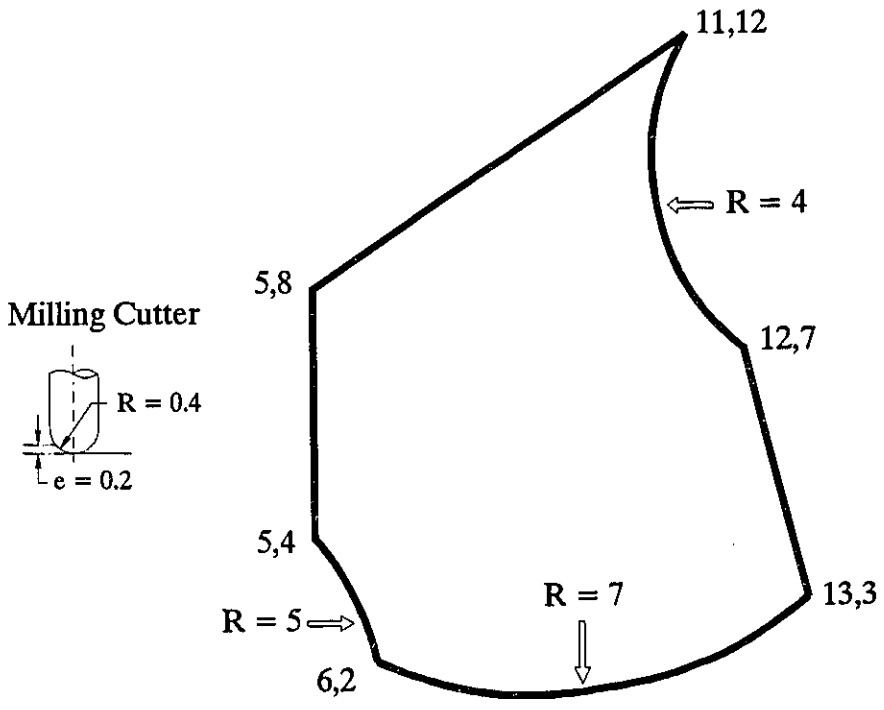


Figure 5.4 Geometric input data available for asymmetric pocket example

Contour Loop Number	Minimum Vertex Angle in Degrees	Optimum Offset Distance
Initial	28.02	0.4
2	45.85	0.2699
3	53.93	0.3141
5	61.43	0.3539
6	68.34	0.3891
7	74.64	0.4200
8	80.39	0.4471
Root	85.61	0.4707

Table 5.3 Details of dynamic offsetting for example 2

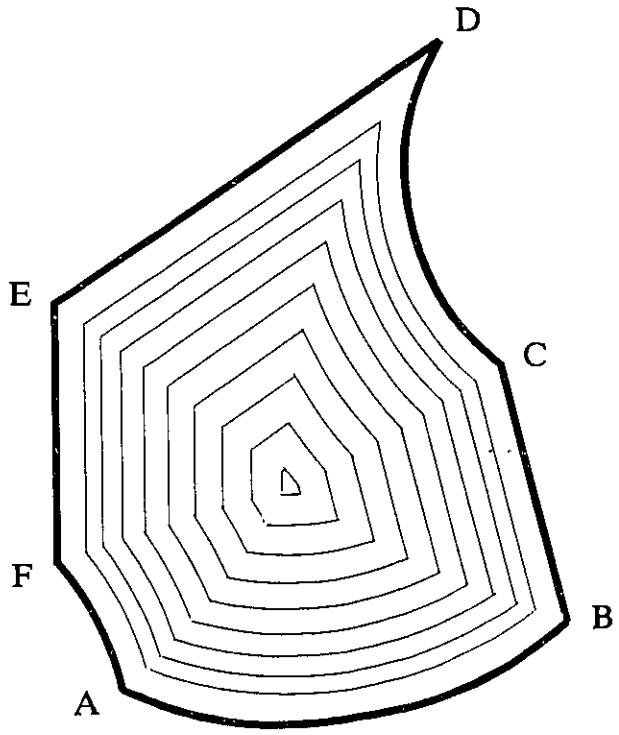


Figure 5.5 Contour offset loops for asymmetric example

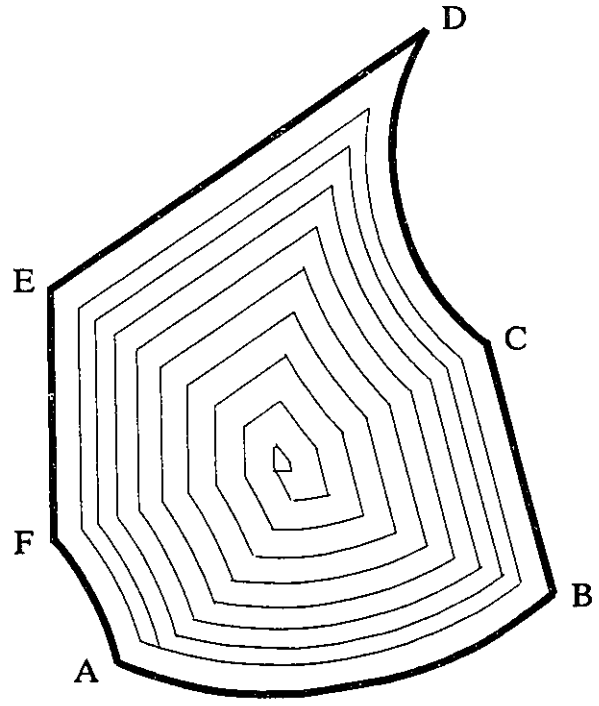


Figure 5.6 Final spiral-like tool path pattern for example 2

Part program (G-code) output for Example 2

N0 G70

N10 G94 F 0.500000

N20 G96 S 1500.000000

N30 G00 X8.578949 Y5.025767

N40 G03 X8.838963 Y5.030538 R3.455679

N50 G01 X8.799424 Y5.188692

N60 G02 X8.600296 Y5.430495 R7.544321

N70 G01 X8.544321 Y5.393179

N80 G01 X8.544321 Y5.086434

N90 G02 X8.828446 Y4.550181 R8.544321

N100 G03 X9.434103 Y4.625743 R3.934873

N110 G01 X9.236333 Y5.416824

N120 G02 X8.668868 Y6.148116 R7.065127

N130 G01 X8.065127 Y5.745623

N140 G01 X8.065127 Y4.956265

N150 G02 X8.519595 Y4.076943 R8.065127

N160 G03 X10.003418 Y4.289599 R4.405663

N170 G01 X9.664581 Y5.644948

N180 G02 X8.771866 Y6.876904 R6.594337

N190 G01 X7.594337 Y6.091885

N200 G01 X7.594337 Y4.824262

N210 G02 X8.188853 Y3.649534 R7.594337

N220 G03 X10.534096 Y4.010528 R4.852811

N230 G01 X10.070304 Y5.865695

N240 G02 X8.910216 Y7.596111 R6.147189

N250 G01 X7.147189 Y6.420759

N260 G01 X7.147189 Y4.694724

N270 G02 X7.856435 Y3.267303 R7.147189

N280 G03 X11.025809 Y3.775703 R5.272889

N290 G01 X10.450452 Y6.077133

N300 G02 X9.086140 Y8.302409 R5.727111

N310 G01 X6.727111 Y6.729723

N320 G01 X6.727111 Y4.568952

N330 G02 X7.528924 Y2.929991 R6.727111

N340 G03 X11.476518 Y3.577296 R5.662020

N350 G01 X10.801611 Y6.276923

N360 G02 X9.301645 Y8.991703 R5.337980

N370 G01 X6.337980 Y7.015926

N380 G01 X6.337980 Y4.448558

N390 G02 X7.212923 Y2.637957 R6.337980

N400 G03 X11.883073 Y3.410283 R6.015929

N410 G01 X11.120069 Y6.462298

N420 G02 X9.558228 Y9.658996 R4.984071

N430 G01 X5.984071 Y7.276224

N440 G01 X5.984071 Y4.335482

N450 G02 X6.915201 Y2.391817 R5.984071

N460 G03 X12.241637 Y3.271337 R6.330089

N470 G01 X11.401941 Y6.630117

N480 G02 X9.856466 Y10.298322 R4.669911

N490 G01 X5.669911 Y7.507285

N500 G01 X5.669911 Y4.231969

N510 G02 X6.642630 Y2.192074 R5.669911

N520 G03 X12.548141 Y3.158193 R6.600000

N530 G01 X11.643430 Y6.777039

N540 G02 X10.195364 Y10.902713 R4.400000

N550 G01 X5.400000 Y7.705803

N560 G01 X5.400000 Y4.140457

N570 G02 X6.334908 Y2.298513 R5.400000

N580 G03 X6.642630 Y2.192074 R6.600000

N590 M00

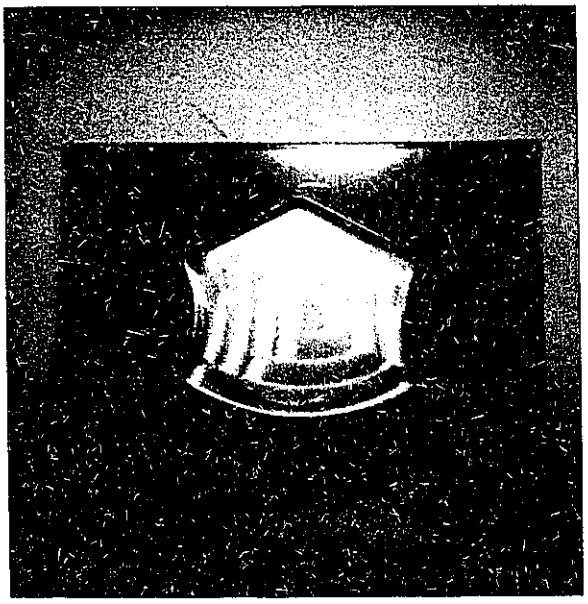


Figure 5.7 Photograph of machined example 1

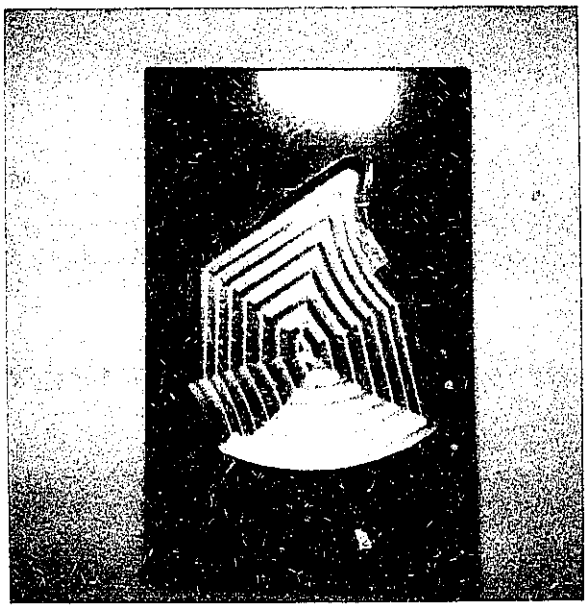


Figure 5.8 Photograph of machined example 2

5.2 Discussions

The part programming system illustrated above is analyzed in this section with emphasis on part programming time, machining time, tool retractions, tolerance and surface finish, tool life, and NC program size.

Part programming time

Part programming time, whether off-line or on-line plays a very crucial role in the economics of production as mentioned in Chapters 2 and 3. In the shop floor, part programming and in particular tool path planning poses a major bottleneck in the linkage of CAD and CAM. This is mostly due to the various data exchange formats existing today. By simple reasoning, it can be seen that the fastest method to generate a part program would be to directly convert geometric data into machining data. Since the proposed method does not convert the Cartesian inputs into any other data format, but rather, directly plans the tool path with this data and outputs the G-codes, part programming time is significantly reduced.

Computational time for examples 1 and 2 are respectively 0.05 seconds 0.10 seconds. This represents time taken from the instant geometric data was input to the time the G-code file, analysis file and a verification file were output. Typically, it can take several hours using some of the traditional methods to plan, verify and generate part programs even for very simple surfaces.

Machining time

It is expected that machining time can be greatly reduced by eliminating very short tool path segments. When short tool path segments exist, the maximum feed rate can be achieved only on a small portion of the segment. Further, VICKERS and BRADLEY (1992) reported that a reduction in machining time of up to 5:1 could be achieved by eliminating short tool path segments, and maximizing the use of circular interpolation controller commands.

Unlike other methods which generate spiral-like tool paths by moving from loop to loop with added short segments, the proposed method combines individual offset loops by simply extending the existing contour offset segments. Further, this action reduces CL points, and thus the stop-start motions.

Tool retractions and plunges

Air cutting and tool retractions signify inefficiency in path planning and should be avoided as much as possible. Pure zig-zag and window milling strategies involve tool retractions and plunges which would ultimately increase the machining time. Technically speaking, additional computations to determine the various exit and entrance angles have to be performed. However, in practice this step is generally ignored. The proposed method totally eliminates tool retractions and tool plunges by machining the entire pocket in a single continuous move by following a spiral-like shaped pattern. Therefore, many problems present in pure zig-zag and window milling are avoided.

NC program size

Short segments mean additional statements in the program which inflate the program size. According to YEUNG and WALTON (1992) these large programs could exceed the storage capacity of the CNC controller, which in turn could cause buffer underflow and data starvation. The proposed path planning methodology minimizes the number of data points for any given surface by eliminating redundant points in the tool path. This naturally results in smaller G-code files.

Tolerance and surface finish

In most tool path patterns, short tool movements are necessary to move from loop to loop or move along the vertices to remove uncut material. These short tool movements across the surface, along with changes in cutting direction cause dwell marks and also induce oscillations along the tool path. Both these factors contribute to significant reduction in surface finish. Further, intense oscillations could damage the machine tool as well.

With the proposed tool path method the tool moves from loop to loop in a continuous motion without leaving any uncut material, thus eliminating the above problems. Further, it can be seen from Figure 5.4 that the problem of undercutting has been avoided at the optimum offset distance for example 1. Also, using the values from Table 5.3, to estimate the tolerances, it was gathered that the tolerances were maintained for example 2. Thus, the proposed tool path method maintains the specified tolerance over the entire machining surface without a single violation.

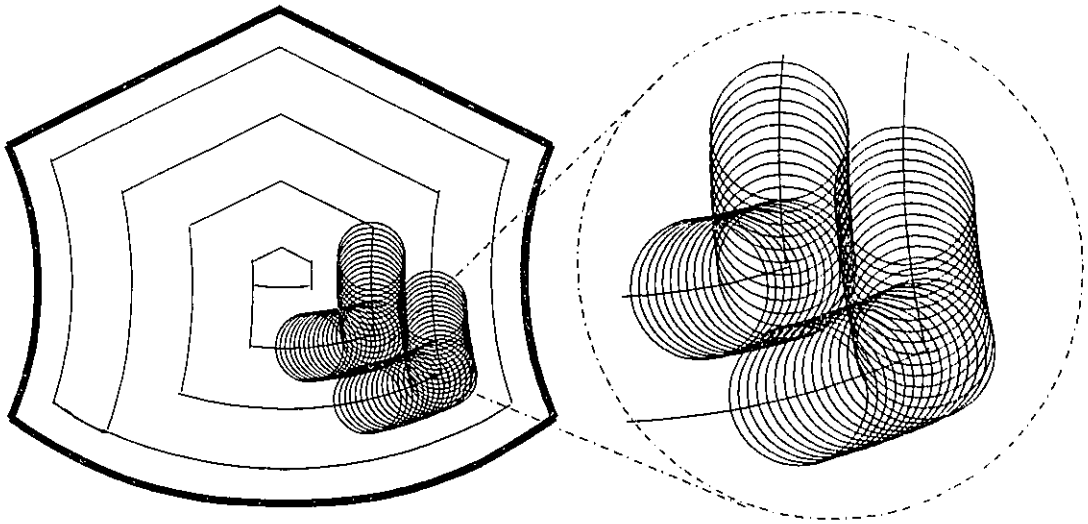


Figure 5.9 Critical elimination of undercutting at vertices

Tool life

As analyzed by RAMAN and LAKKARAJU (1993), and discussed by KALPAKJIAN (1992), tool life is affected by

(a) alternation of up and down milling,

(b) up milling (if the workpiece has hard scale), and

(c) number of tool exits and entrances.

These are in turn dependant on tool path. With the proposed tool path generation method, unidirectional milling can be achieved in most cases and thus up and down milling alternations are avoided. As a result, pure down milling can be achieved.

Further, as mentioned earlier, the entire process can mostly be performed with a single continuous tool motion. The number of tool exits and entrances have been reduced to the minimum: single exit and single entrance. Longer tool life can therefore be expected.

CHAPTER 6

CONCLUSIONS

This chapter consists of two parts which conclude this dissertation. The first part gives a summary of the proposed part programming system, and the second part contains recommendations for future work.

6.1 Summary

An automatic part program generating system to machine $2^1/_2$ D arbitrarily shaped convex pockets has been developed. Tool path planning was done without the aid of solid modellers (CAD packages) and their related data exchange formats. This methodology generates a spiral-like tool path by dynamically computing optimal offsets of the pocket boundary contour. As a result, the undercutting is completely avoided and tolerance requirements are strictly adhered to. The part programming system has been coded in C and implemented in a PC environment. Given Cartesian form geometric input data, the system simultaneously generates a G-code file, a verification file for simulating the tool path on a CAD package and an analysis file for research purposes.

The advantages are summarized as follows:

Geometric modelling

In the presented system, tool path planning was carried out without the aid of a CAD modeller to directly transform geometric data into machine input data without going through intermediate data exchange formats. The presented path planning methodology can handle arbitrarily shaped convex pockets consisting of straight lines and circular arcs.

Tool path optimization

This system automatically generates part programs with optimized tool paths (in respect to the number of tool movements to completely machine a surface at a specified tolerance) by incorporating a methodology to determine the angle bounded by circular arcs, which is a major limitation in CAD. Using this methodology, the system totally eliminates undercutting by computing critical offsets, and also maintains the specified tolerances and avoids gouging over the entire machining surface. This path planning methodology reduces the number of cutter location (CL) points and the number of stop-start motions, thus generating less number of G-codes for any given surface and smoother operation of the machine tool. This methodology incorporates a 100% down milling mode which leads to longer tool life.

System performance

Computational time is reasonably fast as the part programming system eliminates pre-

processing and post-processing stages found in conventional part programming approaches and can simultaneously generate output files in multiple formats. System is PC based and can be easily run on any AT upward system, and can thus easily and inexpensively replace more sophisticated and expensive tool path planning systems.

6.2 Recommendations for future work

The following are a few topics where the proposed system can be extrapolated for future work:

Handling of non-convex pockets

The proposed methodology is based on generating tool paths for convex pockets where the internal angle between adjacent contour segments is less than 180°. Suitable modifications to the segment classification, angles, and offset termination algorithms are to be made to extend the capability of this methodology to handle non-convex pockets with internal angles exceeding 180°. The main obstacle is in the automatic determination of self intersecting segments. If this is achieved it would indeed be a valuable contribution in tool path planning, as up to now there is no method available which avoids the intersection of every pair of contour offset segments.

Pockets containing islands

The framework for extending the proposed methodology to handle an arbitrarily shaped pocket containing an arbitrarily shaped island has been proposed. However, finer details

have to be worked on to incorporate it into the present system. A similar approach could further extend the model to handle pockets containing multiple arbitrary islands.

3 D machining

Without any modifications the present methodology can be directly applied for rough machining $2^1/_2$ D arbitrarily shaped objects. Further, using a contour mapping approach and by chaining the different Z levels with suitable depth of cuts, the method could be used for roughing and finishing a 3D object thus forming a complete part programming system.

CAD interfacing

Although the proposed tool path generating presented here forms a self-contained system, it cannot be ignored that most of the machine drawings in the shop floor are in the form of CAD drawings. Hence, it may be logical to interface this path planning system with CAD packages. The system can be made to read Cartesian geometric input data directly from any standard data exchange format file, then carry out the dynamic tool path generation procedure and generate the output files automatically.

REFERENCES

ALTING, L., and ZHANG, H., 1989, CAPP: state of the art survey. *International Journal of Production Research*, 27, 553-585.

ARMSTRONG, G.T., 1982, A study of automatic generation of non-invasive NC machine paths from geometric models, PhD thesis, University of Leeds.

BEDWORTH, D.D., HENDERSON, M.R., and WOLFE, P.M., 1991, *Computer-Integrated Design and Manufacturing*. Industrial Engineering and Management Science (New York: Mc Graw-Hill).

BEGG, V., 1984, *Developing Expert CAD Systems* (London: Kogan Page).

BISSCHOP, J.J., STEGEMAN, H., and STRIEKWOLD, M.E.A., 1988, The sequencing of point operations on a CNC-machining centre using a microcomputer. *International Journal of Production Research,* 26(8), 1375-1383.

BLINN, J.F., 1988, Fractional Invisibility. *IEEE Computer Graphics and Applications*, 8(6), 77-84.

BOBROW, J.E., 1985, NC machine tool path generation from CSG part representations. *Computer-Aided Design.* 17(2), 69-75.

BRINK, R.W., 1935, *Analytic Geometry*. D.Appleton-Century, London, UK.

BROOMHEAD, P., and EDKINS, M., 1986. Generating NC data at the machine tool for the manufacture of free-form surfaces. *International Journal of Production Research.*

24(1), 1-14.

BRUCKNER, L.K., 1982, Geometric Algorithms for $2^1/_2$ D Roughing Process of Sculptured Surfaces. In *Proc. Joint Anglo-Hingarian Seminar on Computer-Aided Geometric Design*, Budapest, Hungary.

CATIA Users Manual, 1985, Version 2, Release 1, Dassault Systems.

CHASEN. H., 1978, *Geometric Pinciples and Proedures for Computer Graphics Applications*, (NJ: Prentice-Hall).

CHEN, N.R., 1990. Feature-based numerical control program generation from computer-aided design models, PhD thesis, University of Southern California, Los Angeles.

CHEN, Y.D., NI, J., and WU, S.M., 1993, Real-time tool path generation for machining IGES surfaces. *Transactions of the ASME*, 115, 480-486.

CHOI, B.K., LEE, C.S., HWANG, J.S., and JUN, C.S., 1988, Compound surface modelling and machining, *Computer-Aided Design*, 20(3), 127-136.

CHOU, J.J., 1989, Numerical control milling machine tool path generation for regions bounded by free-form curves and surfaces, PhD thesis, University of Utah.

CHOU, J.J., and YANG, D.C.H., 1991, Command generation for three axis CNC machining, *Journal for Engineering and Industry*, 113, 305-310.

CUI, Z., 1993, How to obtain a good surface finish in NC machining of free-formed surfaces. *Computers in Industry*, 23, 227-233.

ESHEL, G., 1986, Automatic generation of process outlines of forming and machining process. PhD thesis, Purdue University.

FAUX, I.D., and PRATT, M.J., 1981, *Computational Geometry for Design and Manufacture*. Mathematics and its Applications (Chichester: Ellis Horwood).

FERSTENBERG, R.A., 1988, Geometric considerations for automatic generation of optimised cutter paths for 3-axis numerically controlled milling, PhD thesis, Cornell University.

GREENWOOD, F., 1989, *Introduction to Computer-Integrated Manufacturing* (FL: Harcourt Brace Jovanovich).

GRIFFITHS, J.G., 1994, A tool path based on Hilbert's curve, *Computer-Aided Design.*

GROOVER, M.P., 1987, *Automation,Production Systems, and Computer-Integrated Manufacturing*. (NJ: Prentice-Hall).

GROOVER, M.P., and ZIMMERS, E,W., 1984, *CAD/CAM: Computer-Aided Design and Manufacturting*. (NJ: Prentice-Hall).

GUYDER, M., 1990, Automating the optimization of 2.5 axis milling, *Computers Industry*, 15, 163-168.

HARENBROOK, D., 1980, *The connection of CAD and CAM by means of the program package PRORENI/NC*. Technical report, Ruhr-University, Bochum, FRG.

HANSEN, A.B., 1989, Tool positioning and path generation for computer-aided manufacturing. PhD thesis, University of Southern California, Los Angeles.

HEGRON, G., 1988, *Image-Synthesis.* MIT Press, Cambridge, MA, USA.

HELD, M., LUKACS, G., and ANDOR, L., 1994, Pocket machining based on contour-parallel tool paths generated by means of proximity maps. *Computer-Aided Design*, 26 189-203.

IWATA, K., OBA, F., and SUGIMURA, N., 1983, Analysis of shape generation process of machine tools for designing machine cells. *Manufacturing Systems*, 12(2), 137-148.

JOSEPH, A.T., 1989, Knowledge acquisition and implementation in an expert process planning system, PhD thesis, University of Manchester, (UMIST).

KALPAKJIAN, S., 1992, *Manufacturing Engineering and Technology,* 2nd edition. (Addision Wesley, New York).

KIM, K., and JEONG, J., 1995, Tool path generation for machining free-form pockets with islands. *Computers in Industrial Engineering*, 28(2), 399-407.

KIM, K.I., and KIM, K., 1993, Parametric tool path planning based on new variable step strategy for sculptued surfaces. *Transactionsof the North American Manufacturing Research Institution of SME*, 95-102.

KRAL, I.H., 1986, *Numerical Control Programming in APT.* (NJ: Prentice-Hall).

LAI, J-Y., and WANG, D-J., 1994, A Strategy for finish cutting path generation of compound surfaces. *Computers in Industry*, 25, 189-209.

LAKKARAJU, R.K., RAMAN, S., and IRANI, S.A., 1992, An analytic model for optimization of NC tool cutting path. *International Journal of Production Research*, 30,

109-127.

MARSHALL, S., and GRIFFITHS, J.G., 1994, A survey of cutter path construction techniques for milling machines. *International Journal of Production Research*, 32(12), 2861-2877.

PERSSON, H., 1978, NC machining of arbitrary shaped pockets. *Computer-Aided Design*, 10,(3), 169-174.

PRABHU, P.V., GRAMOPADHYE, A.K., and WANG, H.P., 1990, A general mathematical model for optimizing NC tool paths for face milling of flat convex surfaces. *International Journal of Production Research*, 28(1), 101-130.

PREISS, K., 1990, Automated mill pocketing computations. *Advanced Geometric Modelling for Engineering Applications*, (BV: North-Holland).

PREISS, K., and KAPLANSKY, E., 1985, Automated part programming for CNC milling by artificial intelligence techniques. *Journal of Manufacturing Systems*, 4(1), 51-63.

RAMAN, S., and LAKKARAJU, R., 1993, The effect of tool life and other process variables in NC path planning. *Computers in Industrial Engineering*, 24(2), 315-328.

SOMASUNDARAM, S., and RAMAN, S., 1992, NC path planning for face milling of convex polygonal surfaces: further analysis. *Journal of Design and Manufacturing*, 2, 19-28.

SATYANARAYANA, B., RAO, P.N., TEWARI, N.K., 1990, An interactive programming system for milling contours and pockets. *The International Journal of Advancced Manufacturing Technology*, 5, 188-213.

SmartCAM Programming Manual, 1988, Point Control, Oregan.

SUH, Y.S., and LEE, K., 1990, NC milling tool path generation for arbitrary pockets defined by sculptured surfaces. *Computer-Aided Design*, 22(5) 273-284.

TSAI, M.D., TAKATA, S., INUI, M., KIMURA, F., SATA, T., 1991, Operation planning based on cutting process models. *Annals of the CIRP*, 40(1), 95-98.

Unigraphics II Version Six Users Manual, 1988, Mc Donnell Douglas Manufacturing System.

VAIL, P.S., 1988, *Computer-Integrated Manufacturing*, Technology (MA: PWS-Kent).

VICKERS, G.W., LY, M., and OETTER, R.G., 1990, *Numerically Controlled Machine Tools*, (Ellis Horwood), England.

VICKERS, G.W., and BRADLEY, C., 1992, Curved surface machining through circular arc interpolation. *Computers in Industry*, 19, 329-337.

WANG, H., CHANG, H., WYSK, R.A., and CHANDAWARKAR, A., 1987, On the efficiency of NC tool path planning for face milling operations. *ASME Transactions, Journal of Engineering for Industry*, 109, 370-376.

SMITH, W.K., 1972, *Analytic Geometry*. Collier-Macmillan, London, UK.

WOODWARK, J., 1986, *Computing Shape*. Butterworths, Guildford, UK.

YAU, H.T., and MENQ, C.H., 1991, Concurrent process planning for machining and inspection of sculptured surfaces. *Transactionsof the North American Manufacturing*

Research Institution of SME, 320-326.

YEUNG, M., and WALTON, D.J., 1992, Data point reduction for NC tool path generation on overdetermined data set. *Information Control Problems in Manufacturing Technology*, 117-122.

ZHANG, D., 1986, CSG solid modelling and automatic NC machining of blend surfaces, PhD thesis, University of Bath.

Appendix 1

(Some important concepts in analytic geometry)

Concepts of Analytic Geometry

Equation of a circle in standard form:

$$(x_1 - h)^2 + (y_1 - k)^2 = a^2$$

Equation of a circle in general form:

$$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

Equation of a line in two-point form:

$$(y - y_1) = \frac{(y_2 - y_1)}{(x_2 - x_1)} \quad (x - x_1)$$

Equation of a line in general form:

$$Ax + By + C = 0$$

Equation of a line passing through the origin:

$$Ax + By = 0$$

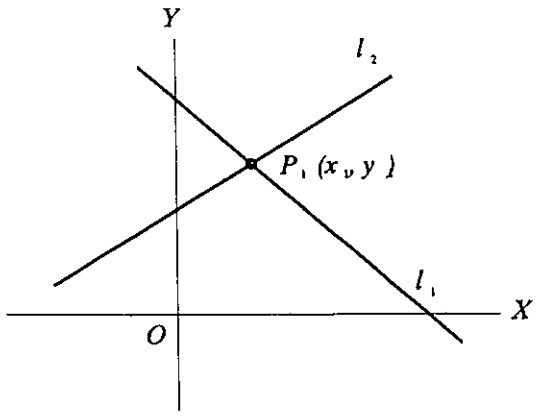
Equation of a line parallel to the X axis:

$$By + C = 0$$

Equation of a line parallel to the Y axis:

$$Ax + C = 0$$

The point of intersection of two straight lines:



The coordinate of the point of intersection of two straight lines may be found by solving the equations of the lines simultaneously for x and y. For P_1, the point of intersection, is common to the two lines, so that its coordinates, unlike the coordinates of any other point, satisfy both equations.

Let the equations of two lines be

(1) $A_1x + B_1y + C_1 = 0$

(2) $A_2x + B_2y + C_2 = 0$

For simplicity assume that $B_1 \neq 0$, $A_2 \neq 0$, $B_2 \neq 0$, $C_2 \neq 0$. The slopes of the lines are $m_1 = -A_1/B_1$ and $m_2 = -A_2/B_2$. The condition for the two lines to be parallel is therefore

(3) $$\frac{A_1}{A_2} = \frac{B_1}{B_2}$$

If this condition is satisfied, the lines, being parallel, in general have no point in common. In algebraic language, equations (1) and (2) are then inconsistent. However

if

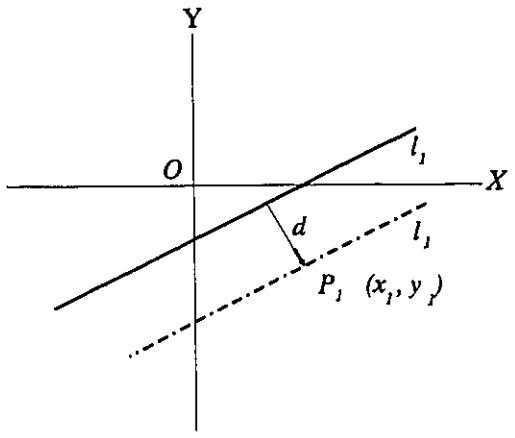
(4) $\frac{A_1}{A_2} = \frac{B_1}{B_2} = \frac{C_1}{C_2}$ *the two equations are equivalent since one equation can be obtained from the other merely by multiplying by a constant, and the lines coincide. They have infinitely many common points. The correspondence can be indicated as follows.*

If $\frac{A_1}{A_2} \neq \frac{B_1}{B_2}$ *the equations have one set of simultaneous solutions; the lines intersect at one point.*

If $\frac{A_1}{A_2} = \frac{B_1}{B_2} \neq \frac{C_1}{C_2}$ *the equations are inconsistent or have no solution; the lines are parallel and have no common point.*

If $\frac{A_1}{A_2} = \frac{B_1}{B_2} = \frac{C_1}{C_2}$ *the equations are equivalent; there are infinitely many sets of solutions; the lines coincide; they have infinitely many common points.*

Distance from a line to a point



Given a straight line, l, whose equation in the standard form is $Ax + By + C = 0$ *and a point* $P_1 : (x_1 , y_1)$. *Let us denote by d the distance from l to* P_1 , *and agree that d shall be counted positive if the direction from the line to the point is the same as the direction from the origin to the line. That is, d is positive if the origin and the point* P_1 *are on opposite sides of line l , and negative if the origin and* P_1 *are on the same side of l.*

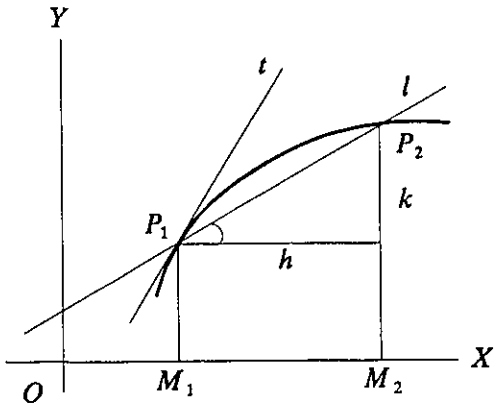
When the equation of the line is given in the form $Ax + By + C = 0$

the formula for distance becomes

$$d = \frac{A x + B y + C}{\pm \sqrt{A^2 + B^2}}$$

where the sign of the denominator is opposite to the sign of C.

Slope of a curve



Let $P_1 : (x_1 , y_1)$ *be a given point of a given curve. The slope of the curve at* P_1 *is defined*

as the slope of the tangent to the curve at P_1 .

Through $P_1 : (x_1 , y_1)$ *draw a line, l. It will meet the curve in a neighbouring point* P_2 . *If we denote by h the difference between the abscissas of* P_2 *and* P_1 , *and by k the difference between their ordinates, the coordinates of* P_2 *will be* (x_1+h , y_1+k). *If we let l rotate about* P_1 , *the coordinates of* (x_1 , y_1) *will remain fixed, while the numbers h and k will vary. And when* P_2 *moves down the curve toward* P_1 , *h and k will tend toward zero.*

Denote by m the variable slope of the line l, and by m_1 *the slope of the curve at* P_1 , *that is, its slope is evidently equal to*

$$m = \frac{(y_1 + k) - y_1}{(x_1 + h) - x_1} = \frac{k}{h}$$

Since t is the limiting position of l, the variable slope m tends toward m_1 , *the slope of t . Our task then, is to find the limit of the fraction k/h when* P_2 *moves along the curve to* P_1 . *In order to do this we must clearly make use of the fact that* P_1 *and* P_2 *lie on the curve. We get two equations by substituting their coordinates* (x_1 , y_1) *and* (x_1+h), (y_1+k) *in the equation of the curve. We solve these two equations for k/h and then take the limit of this fraction as h and k tend toward zero.*

Slope of a tangent to a circle

The equation of a circle in the general form

(1) $$x^2 + y^2 + 2Dx + 2Ey + F = 0$$

When the method to find the slope of the curve defined by (1) at the point (x_1 , y_1) *is applied, we write*

(2) $(x_1 + h)^2 + (y_1 + k)^2 + 2D(x_1 + h) + 2E(y_1 + k) + F = 0$

From this subtract

(3) $x_1^2 + y_1^2 + 2 D x_1 + 2 E y_1 + F = 0$

we get

(4) $h^2 + 2 x_1 h + k^2 + 2 y_1 k + 2 D h + 2 E k = 0$

Therefore

(5) $$\frac{k}{h} = - \frac{(2 x_1 + h + 2 D)}{(2 y_1 + k + 2 E)}$$

Then when h and k tend toward zero, in the limit, as the slope of the curve at (x_1 , y_1), *we have the slope of the tangent to the circle as*

$$m_1 = - \frac{(x_1 + D)}{(y_1 + E)}$$

Appendix 1

(C code for part programming system)

/* Source code for part programming system */

```
# include <float.h>
# include <string.h>
# include <stdio.h>
# include <math.h>
# include <time.h>
float x[50][10],y[50][10],line[50][10][5],cir[50][10][7];
float angle[10],slope[10],c_i_slope[15],c_ii_slope[15];
int index[20],point_max[20],seg_comb_index[15],seqx[15],seqy[15];
int off_eq_index[20];
FILE* write_data;
FILE* gcode;
FILE* cl;
clock_t clock(void);
void main()
{
 clock_t start,end;
 int i,j,k,l,m,n,total_points,chord_index,message,tool_type_index;
 int total_loops,signal,signal_term,control;
 int mode,inc;
 float speed,feed,newx,newy;
 float slope_l,radius,cen_x_i,cen_x_ii,cen_y_i,cen_y_ii,cenx,ceny;
 float tempp,tempq,tempr,temps,tempt;
 float chorda,chordb,chordc,chord_dis_i,chord_dis_ii;
 float min,tool_rad,tolerance,minor_rad,offset;
 float intp,intq,intr,intn,intm,first_x,first_y,second_x,second_y;
 float optdisa,optdisb;
 puts("Give the total number of vertex points (integer)\n");
 scanf("%d",&total_points);
 i=1;
 write_data = fopen("result.out","w");
 gcode = fopen("gcode.out","w");
 cl = fopen("cl.out","w");
 for(j=1 ; j <= total_points ; j++)
  {
   printf("Give x & y values and type & segment comb. for %d point\n",j);
```

```
  puts("0 means line and 1 means circle for type of segment\n");
  puts("1-line-line,2-line-arc,3-arc-line and 4-arc-arc for seg. comb.\n");
  scanf("%f %f %d %d",&x[1][j],&y[1][j],&index[j],&seg_comb_index[j]);
 }
i=1;
for(j=1 ; j <= total_points ; j++)
 {
  if(index[j] == 0)
   {
    if(j == total_points)
     {
   if((x[i][1] - x[i][j]) == 0.0)
    {
     line[i][j][1] = 1.0;
     line[i][j][2] = 0.0;
     line[i][j][3] = -1.0*x[i][j];
    }
   else
    {
     slope_1 = (y[i][1] - y[i][j])/(x[i][1] - x[i][j]);
     line[i][j][1] = -1.0*slope_1;
     line[i][j][2] = 1.0;
     line[i][j][3] = (slope_1*x[i][j]) - y[i][j];
    }
     }
    else
     {
   if((x[i][j+1] - x[i][j]) == 0.0)
    {
     line[i][j][1] = 1.0;
     line[i][j][2] = 0.0;
     line[i][j][3] = -1.0*x[i][j];
    }
   else
    {
     slope_1 = (y[i][j+1] - y[i][j])/(x[i][j+1] - x[i][j]);
     line[i][j][1] = -1.0*slope_1;
```

```
    line[i][j][2] = 1.0;
    line[i][j][3] = (slope_1*x[i][j]) - y[i][j];

   }
    }
  }
 if(index[j] == 1)
  {
   puts("Give the value of the radius\n");
   scanf("%f",&radius);
   if(j == total_points)
    {
  if((x[i][j] - x[i][1]) == 0.0)
   {
    cen_y_i = (y[i][j] + y[i][1])/2.0;
    cen_y_ii = (y[i][j] + y[i][1])/2.0;
    cen_x_i = x[i][j] + sqrt(pow(radius,2) - pow(cen_y_i - y[i][j],2));
    cen_x_ii = x[i][j] - sqrt(pow(radius,2) - pow(cen_y_ii - y[i][j],2));
   }
  else
   {
    tempp = ( pow(x[i][j],2) - pow(x[i][1],2) + pow(y[i][j],2)
        - pow(y[i][1],2) ) / ( 2.0*(x[i][j] - x[i][1]) );
    tempq = (y[i][j] - y[i][1])/(x[i][j] - x[i][1]);
    tempr = pow(tempq,2) + 1.0;
    temps = (2.0*x[i][j]*tempq) - (2.0*y[i][j]) - (2.0*tempp*tempq);
    tempt = pow(x[i][j],2) + pow(y[i][j],2) + pow(tempp,2) -
        (2.0*x[i][j]*tempp) - pow(radius,2);
    cen_x_i = (     (2.0*tempp*tempr) + (tempq*temps) -
          (tempq*(sqrt(pow(temps,2) - (4.0*tempr*tempt))) )  )
            /   (2.0*tempr);
    cen_x_ii = (    (2.0*tempp*tempr) + (tempq*temps) +
          (tempq*(sqrt(pow(temps,2) - (4.0*tempr*tempt)))))
            /    (2.0*tempr);
    cen_y_i = ((-1.0*temps) + sqrt((pow(temps,2) -
(4.0*tempt*tempr))))/(2.0*tempr);
    cen_y_ii = ((-1.0*temps) - sqrt((pow(temps,2) -
```

```
(4.0*tempt*tempr))))/(2.0*tempr);
      }
       }
      else
       {
     if((x[i][j] - x[i][j+1]) == 0.0)
      {
       cen_y_i = (y[i][j] + y[i][j+1])/2.0;
       cen_y_ii = (y[i][j] + y[i][j+1])/2.0;
       cen_x_i = x[i][j] + sqrt(pow(radius,2) - pow(cen_y_i - y[i][j],2));
       cen_x_ii = x[i][j] - sqrt(pow(radius,2) - pow(cen_y_ii - y[i][j],2));
      }
     else
      {
       tempp = (pow(x[i][j],2) - pow(x[i][j+1],2) +
           pow(y[i][j],2) - pow(y[i][j+1],2) ) /(2.0*(x[i][j] - x[i][j+1]));
       tempq = (y[i][j] - y[i][j+1])/(x[i][j] - x[i][j+1]);
       tempr = pow(tempq,2) + 1.0;
       temps = (2.0*x[i][j]*tempq) - (2.0*y[i][j]) - (2.0*tempp*tempq);
       tempt = pow(x[i][j],2) + pow(y[i][j],2) + pow(tempp,2) -
           (2.0*x[i][j]*tempp) - pow(radius,2);
       cen_x_i = ((2.0*tempp*tempr) + (tempq*temps) -
             (tempq*(sqrt(pow(temps,2) - (4.0*tempr*tempt)))))/(2.0*tempr);
       cen_x_ii =((2.0*tempp*tempr) + (tempq*temps) +
             (tempq*(sqrt(pow(temps,2) - (4.0*tempr*tempt)))))/(2.0*tempr);
       cen_y_i = ((-1.0*temps) + sqrt((pow(temps,2) -
(4.0*tempt*tempr))))/(2.0*tempr);
       cen_y_ii = ((-1.0*temps) - sqrt((pow(temps,2) -
(4.0*tempt*tempr))))/(2.0*tempr);
      }
       }
      printf("%f %f %f %f\n",cen_x_i,cen_x_ii,cen_y_i,cen_y_ii);
      if(j == total_points)
       {
     if((x[i][1] - x[i][j]) == 0.0)
      {
       chorda = 1.0;
```

```
 chordb = 0.0;
 chordc = -1.0*x[i][j];
 }
else
 {
 slope_1 = (y[i][1] - y[i][j])/(x[i][1] - x[i][j]);
 chorda = -1.0*slope_1;
 chordb = 1.0;
 chordc = (slope_1*x[i][j]) - y[i][j];
 }
  }
 else
  {
if((x[i][j+1] - x[i][j]) == 0.0)
 {
  chorda = 1.0;
  chordb = 0.0;
  chordc = -1.0*x[i][j];
 }
else
 {
  slope_1 = (y[i][j+1] - y[i][j])/(x[i][j+1] - x[i][j]);
  chorda = -1.0*slope_1;
  chordb = 1.0;
  chordc = (slope_1*x[i][j]) - y[i][j];
 }
  }
 if(chordc == 0.0)
  {
puts("The chord is passing through the origin. So please\n");
puts("give the sign of the perpendicular distance in order\n");
puts("to choose the correct centre - Type 1 for positive distance\n");
puts("and 0 for negative distance. This input has to be given by\n");
puts("looking at the basic figure\n");
scanf("%d",&chord_index);
chord_dis_i = ((chorda*cen_x_i) + (chordb*cen_y_i)) /
          (sqrt(pow(chorda,2) + pow(chordb,2)));
```

```
    chord_dis_ii = ((chorda*cen_x_ii) + (chordb*cen_y_ii)) /
              (sqrt(pow(chorda,2) + pow(chordb,2)));
    if((chord_dis_i <= 0.0) && (chord_index == 0))
     {
      cenx = cen_x_i;
      ceny = cen_y_i;
     }
    if((chord_dis_ii <= 0.0) && (chord_index == 0))
     {
      cenx = cen_x_ii;
      ceny = cen_y_ii;
     }
    if((chord_dis_i >= 0.0) && (chord_index == 1))
     {
      cenx = cen_x_i;
      ceny = cen_y_i;
     }
    if((chord_dis_ii >= 0.0) && (chord_index == 1))
     {
      cenx = cen_x_ii;
      ceny = cen_y_ii;
     }
      }
     else
      {
    puts("The chord is a either a general line with x,y intercepts\n");
    puts("or a line parallel ti x/y axis. So please give the sign of\n");
    puts("the per. distance to choose the correct centre - Type 1 for positive
distance\n");
    puts("and 0 for negative distance. This input has to be given by\n");
    puts("looking at the basic figure\n");
    scanf("%d",&chord_index);
    chord_dis_i = ((chorda*cen_x_i) + (chordb*cen_y_i) + (chordc))/
     (((-1.0*chordc)/(abs(chordc)))*sqrt(pow(chorda,2) + pow(chordb,2)));
    chord_dis_ii = ((chorda*cen_x_ii) + (chordb*cen_y_ii) + (chordc))/
     (((-1.0*chordc)/(abs(chordc)))*sqrt(pow(chorda,2) + pow(chordb,2)));
    if((chord_dis_i <= 0.0) && (chord_index == 0))
```

```
  {
   cenx = cen_x_i;
   ceny = cen_y_i;
  }
 if((chord_dis_ii <= 0.0) && (chord_index == 0))
  {
   cenx = cen_x_ii;
   ceny = cen_y_ii;
  }
 if((chord_dis_i >= 0.0) && (chord_index == 1))
  {
   cenx = cen_x_i;
   ceny = cen_y_i;
  }
 if((chord_dis_ii >= 0.0) && (chord_index == 1))
  {
   cenx = cen_x_ii;
   ceny = cen_y_ii;
  }
   }
  printf("The x and y of centre are %f and %f\n",cenx,ceny);
  cir[i][j][1] = -2.0*cenx;
  cir[i][j][2] = -2.0*ceny;
  cir[i][j][3] = pow(cenx,2)+pow(ceny,2)-pow(radius,2);
  cir[i][j][4] = radius;
  cir[i][j][5] = cenx;
  cir[i][j][6] = ceny;
  }

}
for(j=1 ; j <= total_points ; j++)
 {
  if(j == total_points)
   {
    if(x[i][j] > x[i][1])
     { seqx[j] = 0;}
    if(x[i][j] < x[i][1])
```

```
    { seqx[j] = 1;}
   if(x[i][j] == x[i][1])
    { seqx[j] = 2;}
   if(y[i][j] > y[i][1])
    { seqy[j] = 0;}
   if(y[i][j] < y[i][1])
    { seqy[j] = 1;}
   if(y[i][j] == y[i][1])
    { seqy[j] = 2;}
  }
 else
  {
   if(x[i][j] > x[i][j+1])
    { seqx[j] = 0;}
   if(x[i][j] < x[i][j+1])
    { seqx[j] = 1;}
   if(x[i][j] == x[i][j+1])
    { seqx[j] = 2;}
   if(y[i][j] > y[i][j+1])
    { seqy[j] = 0;}
   if(y[i][j] < y[i][j+1])
    { seqy[j] = 1;}
   if(y[i][j] == y[i][j+1])
    { seqy[j] = 2;}
  }
}
i=0;j=0;k=0;m=0;n=0;signal=0;signal_term=0;
for(i=1 ;   ; i++)
{
    /* -------- The slopes of the segments begin here */
 if(signal == 1)
  { i = i-1 ;  signal_term = 1; }
 for(j=1 ; j <= total_points ; j++)
  {
   if( j == 1 )
    {
     for(k=1 ; k <= total_points ; k++)
```

```
 {
if(index[k] == 0)
 {
  if(k == total_points)
   {
    if( (x[i][1] - x[i][k]) == 0.0 )
     { slope[k] = 90.0; }
    else
     { slope[k] = (y[i][1] - y[i][k]) / (x[i][1] - x[i][k]); }
   }
  else
   {
    if( (x[i][k+1] - x[i][k]) == 0.0 )
     { slope[k] = 90.0; }
    else
     { slope[k] = (y[i][k+1] - y[i][k]) / (x[i][k+1] - x[i][k]); }
   }
 }
if(index[k] == 1)
 {
  if(k == total_points)
   {
    if( (y[i][k] - cir[i][k][6]) == 0.0 )
     { c_i_slope[k] = 90.0; }
    else
     { c_i_slope[k] = (cir[i][k][5] - x[i][k]) / (y[i][k] - cir[i][k][6]); }
    if( (y[i][1] - cir[i][k][6]) == 0.0 )
     { c_ii_slope[k] = 90.0; }
    else
     { c_ii_slope[k] = (cir[i][k][5] - x[i][1]) / (y[i][1] - cir[i][k][6]); }
   }
  else
   {
    if( (y[i][k] - cir[i][k][6]) == 0.0 )
     { c_i_slope[k] = 90.0; }
    else
     { c_i_slope[k] = (cir[i][k][5] - x[i][k]) / (y[i][k] - cir[i][k][6]); }
```

```
            if( (y[i][k+1] - cir[i][k][6]) == 0.0 )
             { c_ii_slope[k] = 90.0; }
            else
             { c_ii_slope[k] = (cir[i][k][5] - x[i][k+1]) / (y[i][k+1] - cir[i][k][6]);
}
           }
          }
           }
         }
        }
       k=0;j=0;
         /* End of finding slopes of segments and beginning of angles of segments
*/
       for(j=1 ; j <= total_points ; j++)
        {
         if(j == total_points)
          {
           if(seg_comb_index[j] == 1)
            {
          if( (((int) slope[j]) == 90) && (((int) slope[1]) != 90) )
           { angle[j] = abs( ((180.0/3.141592654)*atan(slope[1])) - 90.0 ); }
          if( (((int) slope[j]) != 90) && (((int) slope[1]) == 90) )
           { angle[j] = ((180.0/3.141592654)*atan(slope[j])) + 90.0 ; }
          if( (((int) slope[j]) != 90) && (((int) slope[1]) != 90) )
           {
            if((slope[j]*slope[1]) == (-1.0))
             { angle[j] = 90.0; }
            else
             {
              angle[j] = (180.0/3.141592654)*atan((slope[j] - slope[1]) / (1.0 +
(slope[j]*slope[1])));
              if(angle[j] < 0.0)
               { angle[j] = angle[j] + 180.0; }
             }
           }
            }
          if(seg_comb_index[j] == 2)
```

```
        {
      if( (((int) slope[j]) == 90) && (((int) c_i_slope[1]) != 90) )
       { angle[j] = abs( ((180.0/3.141592654)*atan(c_i_slope[1])) - 90.0 ); }
      if( (((int) slope[j]) != 90) && (((int) c_i_slope[1]) == 90) )
       { angle[j] = ((180.0/3.141592654)*atan(slope[j])) + 90.0 ; }
      if( (((int) slope[j]) != 90) && (((int) c_i_slope[1]) != 90) )
       {
        if((slope[j]*c_i_slope[1]) == (-1.0))
         { angle[j] = 90.0; }
        else
         {
          angle[j] = (180.0/3.141592654)*atan((slope[j] - c_i_slope[1]) / (1.0 +
(slope[j]*c_i_slope[1])));
          if(angle[j] < 0.0)
           { angle[j] = angle[j] + 180.0; }
         }
       }
        }
       if(seg_comb_index[j] == 3)
        {
      if( (((int) c_ii_slope[j]) == 90) && (((int) slope[1]) != 90) )
       { angle[j] = abs( ((180.0/3.141592654)*atan(slope[1])) - 90.0 ); }
      if( (((int) c_ii_slope[j]) != 90) && (((int) slope[1]) == 90) )
       { angle[j] = ((180.0/3.141592654)*atan(c_ii_slope[j])) + 90.0 ; }
      if( (((int) c_ii_slope[j]) != 90) && (((int) slope[1]) != 90) )
       {
        if((c_ii_slope[j]*slope[1]) == (-1.0))
         { angle[j] = 90.0; }
        else
         {
          angle[j] = (180.0/3.141592654)*atan((c_ii_slope[j] - slope[1]) / (1.0
+
(c_ii_slope[j]*slope[1])));
          if(angle[j] < 0.0)
           { angle[j] = angle[j] + 180.0; }
         }
       }
```

```
            }
           if(seg_comb_index[j] == 4)
            {
           if( (((int) c_ii_slope[j]) == 90) && (((int) c_i_slope[1]) != 90) )
            { angle[j] = abs( ((180.0/3.141592654)*atan(c_i_slope[1])) - 90.0 ); }
           if( (((int) c_ii_slope[j]) != 90) && (((int) c_i_slope[1]) == 90) )
            { angle[j] = ((180.0/3.141592654)*atan(c_ii_slope[j])) + 90.0 ; }
           if( (((int) c_ii_slope[j]) != 90) && (((int) c_i_slope[1]) != 90) )
            {
             if((c_ii_slope[j]*c_i_slope[1]) == (-1.0))
              { angle[j] = 90.0; }
             else
              {
               angle[j] = (180.0/3.141592654)*atan((c_ii_slope[j] - c_i_slope[1]) /
(1.0
   +
      (c_ii_slope[j]*c_i_slope[1])));
              if(angle[j] < 0.0)
               { angle[j] = angle[j] + 180.0; }
             }
            }
            }
           }
          else
           {
            if(seg_comb_index[j] == 1)
             {
           if( (((int) slope[j]) == 90) && (((int) slope[j+1]) != 90) )
            { angle[j] = abs( ((180.0/3.141592654)*atan(slope[j+1])) - 90.0 ); }
           if( (((int) slope[j]) != 90) && (((int) slope[j+1]) == 90) )
            { angle[j] = ((180.0/3.141592654)*atan(slope[j])) + 90.0 ; }
           if( (((int) slope[j]) != 90) && (((int) slope[j+1]) != 90) )
            {
             if((slope[j]*slope[j+1]) == (-1.0))
              { angle[j] = 90.0; }
             else
              {
```

```
        angle[j] = (180.0/3.141592654)*atan((slope[j] - slope[j+1]) / (1.0 +
(slope[j]*slope[j+1])));
        if(angle[j] < 0.0)
         { angle[j] = angle[j] + 180.0; }
       }
      }
       }
      if(seg_comb_index[j] == 2)
       {
      if( (((int) slope[j]) == 90) && (((int) c_i_slope[j+1]) != 90) )
       { angle[j] = abs( ((180.0/3.141592654)*atan(c_i_slope[j+1])) - 90.0 ); }
      if( (((int) slope[j]) != 90) && (((int) c_i_slope[j+1]) == 90) )
       { angle[j] = ((180.0/3.141592654)*atan(slope[j])) + 90.0 ; }
      if( (((int) slope[j]) != 90) && (((int) c_i_slope[j+1]) != 90) )
       {
        if((slope[j]*c_i_slope[j+1]) == (-1.0))
         { angle[j] = 90.0; }
        else
         {
          angle[j] = (180.0/3.141592654)*atan((slope[j] - c_i_slope[j+1]) / (1.0
+
(slope[j]*c_i_slope[j+1])));
        if(angle[j] < 0.0)
         { angle[j] = angle[j] + 180.0; }
       }
      }
       }
      if(seg_comb_index[j] == 3)
       {
      if( (((int) c_ii_slope[j]) == 90) && (((int) slope[j+1]) != 90) )
       { angle[j] = abs( ((180.0/3.141592654)*atan(slope[j+1])) - 90.0 ); }
      if( (((int) c_ii_slope[j]) != 90) && (((int) slope[j+1]) == 90) )
       { angle[j] = ((180.0/3.141592654)*atan(c_ii_slope[j])) + 90.0 ; }
      if( (((int) c_ii_slope[j]) != 90) && (((int) slope[j+1]) != 90) )
       {
        if((c_ii_slope[j]*slope[j+1]) == (-1.0))
         { angle[j] = 90.0; }
```

```
            else
             {
              angle[j] = (180.0/3.141592654)*atan((c_ii_slope[j] - slope[j+1]) /
(1.0 +
       (c_ii_slope[j]*slope[j+1])));
               if(angle[j] < 0.0)
                { angle[j] = angle[j] + 180.0; }
              }
             }
              }
             if(seg_comb_index[j] == 4)
              {
             if( (((int) c_ii_slope[j]) == 90) && (((int) c_i_slope[j+1]) != 90) )
              { angle[j] = abs( ((180.0/3.141592654)*atan(c_i_slope[j+1])) - 90.0 ); }
             if( (((int) c_ii_slope[j]) != 90) && (((int) c_i_slope[j+1]) == 90) )
              { angle[j] = ((180.0/3.141592654)*atan(c_ii_slope[j])) + 90.0 ; }
             if( (((int) c_ii_slope[j]) != 90) && (((int) c_i_slope[j+1]) != 90) )
              {
               if((c_ii_slope[j]*c_i_slope[j+1]) == (-1.0))
                { angle[j] = 90.0; }
               else
                {
                 angle[j] = (180.0/3.141592654)*atan((c_ii_slope[j] - c_i_slope[j+1]) /
    (1.0
      + (c_ii_slope[j]*c_i_slope[j+1])));
               if(angle[j] < 0.0)
                { angle[j] = angle[j] + 180.0; }
              }
             }
              }
            }
           }
          j=0;
             /* End of finding angles of segments */
          min=angle[1];
          for(j=2 ; j <= total_points; j++)
           {
```

```
  if(angle[j] <= min)
  {
   min = angle[j];
  }
 }
j=0;
printf("The minimum angle is equal to %f\n",min);
/* Begin of offset equations----------*/
if(i == 1)
 {
  puts("Please specify the type of tool and its major radius (float)\n");
  puts("1-Flat mill,2-Ball end milland 3-Generalised mill\n");
  scanf("%d %f",&tool_type_index,&tool_rad);
  if(tool_type_index == 2)
   {
    puts("Specify tolerance\n");
    scanf("%f",&tolerance);
   }
  if(tool_type_index == 3)
   {
    puts("Specify tolerance and minor radius\n");
    scanf("%f %f",&tolerance,&minor_rad);
   }
 }
if(i == 1)
 { offset = tool_rad; }
else
 {
  if(signal == 1)
   { offset = tool_rad; }
  else {
  if(tool_type_index == 1)
   { offset = (tool_rad)*(1.0 + sin(3.14157*min/360.0)); }
  if(tool_type_index == 2)
   {
    offset = 2.0*(sin(3.14157*min/360.0))*
        sqrt(tolerance*(2.0*tool_rad - tolerance));
```

```
   }
  if(tool_type_index == 3)
   {
    offset = 2.0*(sin(3.14157*min/360.0))*
        (tool_rad - minor_rad +
        sqrt(tolerance*(2.0*tool_rad - tolerance)));
   }
  }
 }
printf("Offset dist. is %f\n",offset);
fprintf(write_data,"Offset dist. is %f\n",offset);
for(j=1 ; j <= total_points ; j++)
 {
  if(index[j] == 0)
   {
   if(line[i][j][3] == 0.0)
    {
   printf("We have the %d segment as a line through the origin\n",j);
   puts("Give the sign of the distance- 0 for -ve and 1 for +ve\n");
   scanf("%d",&off_eq_index[j]);
   line[i+1][j][1] = line[1][j][1];
   line[i+1][j][2] = line[1][j][2];
   if(off_eq_index[j] == 0)
    {
     line[i+1][j][3] = offset*sqrt(pow(line[1][j][1],2) +
           pow(line[1][j][2],2)) ;
    }
   if(off_eq_index[j] == 1)
    {
     line[i+1][j][3] = -1.0*offset*sqrt(pow(line[1][j][1],2) +
           pow(line[1][j][2],2)) ;
    }
   if(off_eq_index[j] == 0)
    { off_eq_index[j] = 1; }
   else
    { off_eq_index[j] = 0; }
    }
```

```
else
 {

if(i == 1)
 {
  printf("%d segment is a general line( maybe llel to x/y axis) \n",j);
  puts("Give the sign of the distance- 0 for -ve and 1 for +ve\n");
  scanf("%d",&off_eq_index[j]);
 }
line[i+1][j][1] = line[1][j][1];
line[i+1][j][2] = line[1][j][2];
if(off_eq_index[j] == 0)
 {
  line[i+1][j][3] = line[i][j][3] +
            (offset*(-line[1][j][3]/abs(line[1][j][3]))*
            sqrt(pow(line[1][j][1],2) +
            pow(line[1][j][2],2)) );
 }

 if(off_eq_index[j] == 1)
  {
   line[i+1][j][3] = line[i][j][3] -
            offset*(-1.0*line[1][j][3]/abs(line[1][j][3]))*
            sqrt(pow(line[1][j][1],2) +
            pow(line[1][j][2],2)) ;
  }

  }
}
if(index[j] == 1)
{
 cir[i+1][j][1] = cir[1][j][1];
 cir[i+1][j][2] = cir[1][j][2];
 cir[i+1][j][5] = cir[1][j][5];
 cir[i+1][j][6] = cir[1][j][6];
 if(i == 1)
  {
```

```
            printf("Segment %d is a circle - increment or decrement the radius\n",j);
            puts("0 for increment and 1 for decrement\n");
            scanf("%d",&off_eq_index[j]);
             }
            if(off_eq_index[j] == 0)
             {
            cir[i+1][j][4] = cir[i][j][4] + offset;
            cir[i+1][j][3] = pow(cir[1][j][5],2) + pow(cir[1][j][6],2) -
                     pow((cir[i][j][4] + offset),2);
             }
            if(off_eq_index[j] == 1)
             {
            cir[i+1][j][4] = cir[i][j][4] - offset;
            cir[i+1][j][3] = pow(cir[1][j][5],2) + pow(cir[1][j][6],2) -
                     pow((cir[i][j][4] - offset),2);
             }
           }
         }
        j=0;
         /* offset equations ends here and intersection begins here*/
        if(i==1)
         { start = clock();}
        for(j=1 ; j <= total_points ; j++)
         {
          if(seg_comb_index[j] == 1)
           {
            if(j == total_points)
             {
            if((line[i+1][j][2]*line[i+1][1][1] - line[i+1][j][1]*line[i+1][1][2])  ==
0.0
   )
             { puts("The lines do no intersect - Please abort program\n");
               getchar();getchar();
             }
            else
             {
               x[i+1][1] = (line[i+1][1][2]*line[i+1][j][3] -
```

```
line[i+1][1][3]*line[i+1][j][2]) /
                (line[i+1][j][2]*line[i+1][1][1] - line[i+1][j][1]*line[i+1][1][2])
;
          y[i+1][1] = (line[i+1][j][1]*line[i+1][1][3] -
line[i+1][1][1]*line[i+1][j][3]) /
                (line[i+1][j][2]*line[i+1][1][1] - line[i+1][j][1]*line[i+1][1][2])
;
        }
        }
       else
        {
      if((line[i+1][j][2]*line[i+1][j+1][1] - line[i+1][j][1]*line[i+1][j+1][2])
==
0.0 )
        {puts("The lines do no intersect - Please abort program\n");
         getchar();getchar();
        }
      else
        {
          x[i+1][j+1] = (line[i+1][j+1][2]*line[i+1][j][3] -
line[i+1][j+1][3]*line[i+1][j][2]) /
                (line[i+1][j][2]*line[i+1][j+1][1] -
line[i+1][j][1]*line[i+1][j+1][2]) ;
          y[i+1][j+1] = (line[i+1][j][1]*line[i+1][j+1][3] -
line[i+1][j+1][1]*line[i+1][j][3]) /
                (line[i+1][j][2]*line[i+1][j+1][1] -
line[i+1][j][1]*line[i+1][j+1][2]) ;
        }
        }
      }
     if(  (seg_comb_index[j] == 2) || (seg_comb_index[j] == 3)  )
      {
       if(index[j] == 0)
        {
      if(j == total_points)
       {
        if(line[i+1][j][1] == 0.0)
```

```
                {
                 y[i+1][1] =  -line[i+1][j][3] / line[i+1][j][2] ;
                 intp = cir[i+1][1][1];
                 intq = (pow(line[i+1][j][3],2) / pow(line[i+1][j][2],2)) -
                    ( (cir[i+1][1][2] * line[i+1][j][3]) / line[i+1][j][2] )  +
                    cir[i+1][1][3];
                 if((pow(intp,2) - 4.0*intq)  < 0.0 )
                  { puts("The lines do no intersect - Please abort program\n");
                    getchar();getchar();
                  }
                 else
                  {
                   first_x = ( -intp +  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
                   second_x = ( -intp -  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
                   optdisa = sqrt( pow(first_x - x[i][1],2) + pow(y[i+1][1] - y[i][1],2)
);
                   optdisb = sqrt( pow(second_x - x[i][1],2) + pow(y[i+1][1] -
y[i][1],2)
    );
                   if( optdisa <= optdisb )
                    { x[i+1][1] = first_x; }
                   else
                    { x[i+1][1] = second_x; }
                  }
                 }
                else
                 {
                  intp = pow(line[i+1][j][1],2) + pow(line[i+1][j][2],2) ;
                  intq = (2.0*line[i+1][j][2]*line[i+1][j][3]) -
                     (line[i+1][j][1]*line[i+1][j][2]*cir[i+1][1][1]) +
                     (pow(line[i+1][j][1],2)*cir[i+1][1][2]) ;
                  intr = pow(line[i+1][j][3],2) -
                     (line[i+1][j][1]*line[i+1][j][3]*cir[i+1][1][1]) +
                     (pow(line[i+1][j][1],2)*cir[i+1][1][3]) ;
                  if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
                   { puts("The line and circle do no intersect - Please abort program\n");
                     getchar();getchar();
```

```
                }
               else
                {
                 first_x = ( (-2.0*intp*line[i+1][j][3]) + (line[i+1][j][2]*intq) -
                    (line[i+1][j][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i+1][j][1]*intp) ;
                 second_x = ( (-2.0*intp*line[i+1][j][3]) + (line[i+1][j][2]*intq) +
                    (line[i+1][j][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i+1][j][1]*intp) ;
                 first_y = ( -intq + sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp) ;
                 second_y = ( -intq - sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp)
;
                 optdisa = sqrt( pow(first_x - x[i][1],2) + pow(first_y - y[i][1],2) );
                 optdisb = sqrt( pow(second_x - x[i][1],2) + pow(second_y -
y[i][1],2) );
                 if( optdisa <= optdisb )
                  { x[i+1][1] = first_x ; y[i+1][1] = first_y; }
                 else
                  { x[i+1][1] = second_x ; y[i+1][1] = second_y; }
                }
               }
              }
             else
              {
               if(line[i+1][j][1] == 0.0)
                {
                 y[i+1][j+1] =  -line[i+1][j][3] / line[i+1][j][2] ;
                 intp = cir[i+1][j+1][1];
                 intq = (pow(line[i+1][j][3],2) / pow(line[i+1][j][2],2)) -
                    ( (cir[i+1][j+1][2] * line[i+1][j][3]) / line[i+1][j][2] )  +
                    cir[i+1][j+1][3];
                 if((pow(intp,2) - 4.0*intq)  < 0.0 )
                  { puts("The lines do no intersect - Please abort program\n");
                    getchar();getchar();
                  }
                 else
                  {
```

```
            first_x = ( -intp +  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
            second_x = ( -intp -  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
            optdisa = sqrt( pow(first_x - x[i][j+1],2) + pow(y[i+1][j+1] -
        y[i][j+1],2) );
            optdisb = sqrt( pow(second_x - x[i][j+1],2) + pow(y[i+1][j+1] -
        y[i][j+1],2) );
            if( optdisa <= optdisb )
             { x[i+1][j+1] = first_x; }
            else
             { x[i+1][j+1] = second_x; }
           }
         }
        else
         {
          intp = pow(line[i+1][j][1],2) + pow(line[i+1][j][2],2) ;
          intq = (2.0*line[i+1][j][2]*line[i+1][j][3]) -
              (line[i+1][j][1]*line[i+1][j][2]*cir[i+1][j+1][1]) +
              (pow(line[i+1][j][1],2)*cir[i+1][j+1][2]) ;
          intr = pow(line[i+1][j][3],2) -
              (line[i+1][j][1]*line[i+1][j][3]*cir[i+1][j+1][1]) +
              (pow(line[i+1][j][1],2)*cir[i+1][j+1][3]) ;
          if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
           { puts("The line and circle do no intersect - Please abort program\n");
             getchar();getchar();
           }
          else
           {
            first_x = ( (-2.0*intp*line[i+1][j][3]) + (line[i+1][j][2]*intq) -
                (line[i+1][j][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
      (2.0*line[i+1][j][1]*intp) ;
            second_x = ( (-2.0*intp*line[i+1][j][3]) + (line[i+1][j][2]*intq) +
                (line[i+1][j][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
      (2.0*line[i+1][j][1]*intp) ;
            first_y = ( -intq + sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp) ;
            second_y = ( -intq - sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp)
;
            optdisa = sqrt( pow(first_x - x[i][j+1],2) + pow(first_y -
```

```
y[i][j+1],2) );
                optdisb = sqrt( pow(second_x - x[i][j+1],2) + pow(second_y -
        y[i][j+1],2) );
                if( optdisa <= optdisb )
                 { x[i+1][j+1] = first_x ; y[i+1][j+1] = first_y; }
                else
                 { x[i+1][j+1] = second_x ; y[i+1][j+1] = second_y; }
                }
               }

              }
               }
              else
               {
             if(j == total_points)
              {
               if(line[i+1][1][1] == 0.0)
                {
                 y[i+1][1] =  -line[i+1][1][3] / line[i+1][1][2] ;
                 intp = cir[i+1][j][1];
                 intq = (pow(line[i+1][1][3],2) / pow(line[i+1][1][2],2)) -
                    ( (cir[i+1][j][2] * line[i+1][1][3]) / line[i+1][1][2] )  +
                    cir[i+1][j][3];
                 if((pow(intp,2) - 4.0*intq)  < 0.0 )
                  { puts("The lines do no intersect - Please abort program\n");
                    getchar();getchar();
                  }
                 else
                  {
                   first_x = ( -intp +  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
                   second_x = ( -intp -  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
                   optdisa = sqrt( pow(first_x - x[i][1],2) + pow(y[i+1][1] - y[i][1],2)
);
                   optdisb = sqrt( pow(second_x - x[i][1],2) + pow(y[i+1][1] -
y[i][1],2)
    );
                   if( optdisa <= optdisb )
```

```
                { x[i+1][1] = first_x; }
               else
                { x[i+1][1] = second_x; }
              }
             }
            else
             {
              intp = pow(line[i+1][1][1],2) + pow(line[i+1][1][2],2) ;
              intq = (2.0*line[i+1][1][2]*line[i+1][1][3]) -
                 (line[i+1][1][1]*line[i+1][1][2]*cir[i+1][j][1]) +
                 (pow(line[i+1][1][1],2)*cir[i+1][j][2]) ;
              intr = pow(line[i+1][1][3],2) -
                 (line[i+1][1][1]*line[i+1][1][3]*cir[i+1][j][1]) +
                 (pow(line[i+1][1][1],2)*cir[i+1][j][3]) ;
              if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
               { puts("The line and circle do no intersect - Please abort program\n");
                 getchar();getchar();
               }
              else
               {
                first_x = ( (-2.0*intp*line[i+1][1][3]) + (line[i+1][1][2]*intq) -
                   (line[i+1][1][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
      (2.0*line[i+1][1][1]*intp) ;
                second_x = ( (-2.0*intp*line[i+1][1][3]) + (line[i+1][1][2]*intq) +
                   (line[i+1][1][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
      (2.0*line[i+1][1][1]*intp) ;
                first_y = ( -intq + sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp) ;
                second_y = ( -intq - sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp)
;
                optdisa = sqrt( pow(first_x - x[i][1],2) + pow(first_y - y[i][1],2) );
                optdisb = sqrt( pow(second_x - x[i][1],2) + pow(second_y -
y[i][1],2) );
                if( optdisa <= optdisb )
                 { x[i+1][1] = first_x ; y[i+1][1] = first_y; }
                else
                 { x[i+1][1] = second_x ; y[i+1][1] = second_y; }
               }
```

```
    }
   }
  else
   {
    if(line[i+1][j+1][1] == 0.0)
     {
      y[i+1][j+1] =  -line[i+1][j+1][3] / line[i+1][j+1][2] ;
      intp = cir[i+1][j][1];
      intq = (pow(line[i+1][j+1][3],2) / pow(line[i+1][j+1][2],2)) -
         ( (cir[i+1][j][2] * line[i+1][j+1][3]) / line[i+1][j+1][2] )  +
         cir[i+1][j][3];
      if((pow(intp,2) - 4.0*intq)  < 0.0 )
       { puts("The lines do no intersect - Please abort program\n");
         getchar();getchar();
       }
       else
       {
        first_x = ( -intp +  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
        second_x = ( -intp -  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
        optdisa = sqrt( pow(first_x - x[i][j+1],2) + pow(y[i+1][j+1] -
y[i][j+1],2) );
        optdisb = sqrt( pow(second_x - x[i][j+1],2) + pow(y[i+1][j+1] -
y[i][j+1],2) );
        if( optdisa <= optdisb )
         { x[i+1][j+1] = first_x; }
        else
         { x[i+1][j+1] = second_x; }
       }
     }
    else
     {
      intp = pow(line[i+1][j+1][1],2) + pow(line[i+1][j+1][2],2) ;
      intq = (2.0*line[i+1][j+1][2]*line[i+1][j+1][3]) -
         (line[i+1][j+1][1]*line[i+1][j+1][2]*cir[i+1][j][1]) +
         (pow(line[i+1][j+1][1],2)*cir[i+1][j][2]) ;
      intr = pow(line[i+1][j+1][3],2) -
         (line[i+1][j+1][1]*line[i+1][j+1][3]*cir[i+1][j][1]) +
```

```
                (pow(line[i+1][j+1][1],2)*cir[i+1][j][3]) ;
              if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
               { puts("The line and circle do no intersect - Please abort program\n");
                 getchar();getchar();
               }
              else
               {
                first_x = ( (-2.0*intp*line[i+1][j+1][3]) + (line[i+1][j+1][2]*intq)
-
                    (line[i+1][j+1][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i+1][j+1][1]*intp) ;
                second_x = ( (-2.0*intp*line[i+1][j+1][3]) +
(line[i+1][j+1][2]*intq)
        +
                    (line[i+1][j+1][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i+1][j+1][1]*intp) ;
                first_y = ( -intq + sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp) ;
                second_y = ( -intq - sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp)
;
                optdisa = sqrt( pow(first_x - x[i][j+1],2) + pow(first_y -
y[i][j+1],2) );
                optdisb = sqrt( pow(second_x - x[i][j+1],2) + pow(second_y -
        y[i][j+1],2) );
                if( optdisa <= optdisb )
                 { x[i+1][j+1] = first_x ; y[i+1][j+1] = first_y; }
                else
                 { x[i+1][j+1] = second_x ; y[i+1][j+1] = second_y; }
               }
             }

            }
             }
           }
          if(seg_comb_index[j] == 4)
           {
            if(j == total_points)
             {
```

```
            if( (cir[i+1][j][1] - cir[i+1][1][1]) == 0.0 )
             {
              if( (cir[i+1][j][2] - cir[i+1][1][2]) == 0.0 )
               { puts("Circles are the same - Please abort\n");
                 getchar() ; getchar();
               }
              else
               {
                y[i+1][1] = (cir[i+1][1][3] - cir[i+1][j][3]) / (cir[i+1][j][2] -
        cir[i+1][1][2]) ;
                intq = cir[i+1][j][1];
                intr = pow(y[i+1][1],2) + (cir[i+1][j][2]*y[i+1][1]) + cir[i+1][j][3]
;
                if( (pow(intq,2) - 4.0*intr)  < 0.0 )
                 { puts("The Circles do no intersect - Please abort program\n");
                   getchar() ; getchar();
                 }
                else
                 {
                  first_x = ( -intq +  sqrt(pow(intq,2) - 4.0*intr) ) / 2.0;
                  second_x = ( -intq -  sqrt(pow(intq,2) - 4.0*intr) ) / 2.0;
                  optdisa = sqrt( pow(first_x - x[i][1],2) + pow(y[i+1][1] - y[i][1],2)
);
                  optdisb = sqrt( pow(second_x - x[i][1],2) + pow(y[i+1][1] -
y[i][1],2)
    );
                  if( optdisa <= optdisb )
                   { x[i+1][1] = first_x ; }
                  else
                   { x[i+1][1] = second_x ; }
                 }
               }
              else
               {
                 intm = (cir[i+1][1][3] - cir[i+1][j][3]) /  (cir[i+1][j][1] -
cir[i+1][1][1])
```

```
    ;
                intn = (cir[i+1][1][2] - cir[i+1][j][2]) /  (cir[i+1][j][1] -
cir[i+1][1][1]) ;
                intp = pow(intn,2) + 1.0;
                intq = (2.0*intm*intn) + (cir[i+1][j][1]*intn) + (cir[i+1][j][2]) ;
                intr = pow(intm,2) + (cir[i+1][j][1]*intm) +  cir[i+1][j][3] ;
                if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
                 { puts("The Circles do no intersect - Please abort program\n");
                   getchar() ; getchar();
                 }
                else
                 {
                  first_x = ( (2.0*intm*intp) - (intn*intq) +
                      (intn * sqrt(pow(intq,2) - (4.0*intp*intr))) ) / (2.0*intp);
                  second_x = ( (2.0*intm*intp) - (intn*intq) -
                      (intn * sqrt(pow(intq,2) - (4.0*intp*intr))) ) / (2.0*intp);
                  first_y =  ( -intq +  sqrt(pow(intq,2) - (4.0*intp*intr)) ) /
(2.0*intp);
                  second_y =  ( -intq -  sqrt(pow(intq,2) - (4.0*intp*intr)) ) /
(2.0*intp);
                  optdisa = sqrt( pow(first_x - x[i][1],2) + pow(first_y - y[i][1],2) );
                  optdisb = sqrt( pow(second_x - x[i][1],2) + pow(second_y -
y[i][1],2) );
                  if( optdisa <= optdisb )
                   { x[i+1][1] = first_x ; y[i+1][1] = first_y ; }
                  else
                   { x[i+1][1] = second_x ; y[i+1][1] = second_y ; }
                 }
               }
                }
               else
                {
               if( (cir[i+1][j][1] - cir[i+1][j+1][1]) == 0.0 )
               {
                if( (cir[i+1][j][2] - cir[i+1][j+1][2]) == 0.0 )
                 { puts("Circles are the same - Please abort\n");
                   getchar() ; getchar();
```

```
      }
     else
      {
       y[i+1][j+1] = (cir[i+1][j+1][3] - cir[i+1][j][3]) / (cir[i+1][j][2] -
cir[i+1][j+1][2]) ;
       intq = cir[i+1][j][1];
       intr = pow(y[i+1][j+1],2) + (cir[i+1][j][2]*y[i+1][j+1]) +
cir[i+1][j][3] ;
       if( (pow(intq,2) - 4.0*intr)  < 0.0 )
        { puts("The Circles do no intersect - Please abort program\n");
          getchar() ; getchar();
        }
       else
        {
         first_x = ( -intq +  sqrt(pow(intq,2) - 4.0*intr) ) / 2.0;
         second_x = ( -intq -  sqrt(pow(intq,2) - 4.0*intr) ) / 2.0;
         optdisa = sqrt( pow(first_x - x[i][j+1],2) + pow(y[i+1][j+1] -
y[i][j+1],2) );
         optdisb = sqrt( pow(second_x - x[i][j+1],2) + pow(y[i+1][j+1] -
y[i][j+1],2) );
         if( optdisa <= optdisb )
          { x[i+1][j+1] = first_x ; }
         else
          { x[i+1][j+1] = second_x ; }
        }
      }
     else
      {
        intm = (cir[i+1][j+1][3] - cir[i+1][j][3]) /  (cir[i+1][j][1] -
cir[i+1][j+1][1]) ;
       intn = (cir[i+1][j+1][2] - cir[i+1][j][2]) /  (cir[i+1][j][1] -
cir[i+1][j+1][1]) ;
       intp = pow(intn,2) + 1.0;
       intq = (2.0*intm*intn) + (cir[i+1][j][1]*intn) + (cir[i+1][j][2]) ;
       intr = pow(intm,2) + (cir[i+1][j][1]*intm) +  cir[i+1][j][3] ;
       if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
```

```
                { puts("The Circles do no intersect - Please abort program\n");
                  getchar() ; getchar();
                 }
                else
                 {
                  first_x = ( (2.0*intm*intp) - (intn*intq) +
                      (intn * sqrt(pow(intq,2) - (4.0*intp*intr))) ) / (2.0*intp);
                  second_x = ( (2.0*intm*intp) - (intn*intq) -
                      (intn * sqrt(pow(intq,2) - (4.0*intp*intr))) ) / (2.0*intp);
                  first_y =  ( -intq +  sqrt(pow(intq,2) - (4.0*intp*intr)) ) /
(2.0*intp);
                  second_y =  ( -intq -  sqrt(pow(intq,2) - (4.0*intp*intr)) ) /
(2.0*intp);
                  optdisa = sqrt( pow(first_x - x[i][j+1],2) + pow(first_y -
y[i][j+1],2) );
                  optdisb = sqrt( pow(second_x - x[i][j+1],2) + pow(second_y -
        y[i][j+1],2) );
                  if( optdisa <= optdisb )
                   { x[i+1][j+1] = first_x ; y[i+1][j+1] = first_y ; }
                  else
                   { x[i+1][j+1] = second_x ; y[i+1][j+1] = second_y ; }
                 }
              }
               }
             }
           }
          j=0;
          /*puts("Reached till terminal cond\n");
          getchar();
          getchar();*/
          for(j=1 ; j <= (total_points) ; j++)
           {
            if(j == total_points)
             {
              if(seqx[j] == 0)
            { if(x[i+1][j] > x[i+1][1])
              { } else { control = 1; } }
```

```
 if(seqx[j] == 1)
{ if(x[i+1][j] < x[i+1][1])
  { } else { control = 1; } }
  if(seqx[j] == 2)
{ if(abs(x[i+1][j] - x[i+1][1]) <= 0.001)
  { } else { control = 1; } }
  if(seqy[j] == 0)
{ if(y[i+1][j] > y[i+1][1])
  { } else { control = 1; } }
  if(seqy[j] == 1)
{ if(y[i+1][j] < y[i+1][1])
  { } else { control = 1; } }
  if(seqy[j] == 2)
{ if(abs(y[i+1][j] - y[i+1][1]) <= 0.001)
  { } else { control = 1; } }
 }
else
 {
  if(seqx[j] == 0)
{ if(x[i+1][j] > x[i+1][j+1])
  { } else { control = 1; } }
  if(seqx[j] == 1)
{ if(x[i+1][j] < x[i+1][j+1])
  { } else { control = 1; } }
  if(seqx[j] == 2)
{ if(abs(x[i+1][j] - x[i+1][j+1]) <= 0.001)
  { } else { control = 1; } }
  if(seqy[j] == 0)
{ if(y[i+1][j] > y[i+1][j+1])
  { } else { control = 1; } }
  if(seqy[j] == 1)
{ if(y[i+1][j] < y[i+1][j+1])
  { } else { control = 1; } }
  if(seqy[j] == 2)
{ if(abs(y[i+1][j] - y[i+1][j+1]) <= 0.001)
  { } else { control = 1; } }
```

```
  }
 if( control != 1 )
  {
   /*puts("Satisfied\n");
   getchar();
   getchar();*/
  }
 else
  {
   if(signal_term == 1)
    {
  signal_term = 0;
  signal = 2;
  break;
    }
   signal = 1;
   control = 0;
   break;
  }
 }
j=0;
if((signal == 1) && (signal_term == 1))
 {
  total_loops = i+1;
  printf("Total loops is %d\n",total_loops);
  break;
 }
if((signal == 2) && (signal_term == 0))
 {
  total_loops = i;
  printf("Total loops is %d\n",total_loops);
  break;
 }

}/* i loop ends here*/
i=0;j=0;
end = clock();
```

```
    fprintf(write_data,"Computation time is %f\n",(end-start)/(CLK_TCK));
    for(i=1 ; i <= total_loops ; i++)
     {
      for(j=1; j <= total_points ; j++)
       {
        fprintf(write_data,"%f %f\n",x[i][j],y[i][j]);
       }
      fprintf(write_data,"%s\n"," ");
     }
    i=0;j=0;
    puts("Do you want inch mode or metric mode - 0-inch;1-metric\n");
    scanf("%d",&mode);
    if(mode == 0)
     { fprintf(gcode,"N%d G70\n",0 ); }
    else
     { fprintf(gcode,"N%d G71\n",0 ); }
    puts("Give the speed-float\n");
    scanf("%f",&speed);
    puts("Give the feed rate-float\n");
    scanf("%f",&feed);
    fprintf(gcode,"N10 G94 F%f\n",feed);
    fprintf(gcode,"N20 G96 S%f\n",speed);
    inc = 10;
    for(i=total_loops ; i >= 3 ; i--)
     {
      if(i == total_loops)
       {
        fprintf(gcode,"N%d G00 %f
%f\n",(20+inc),x[total_loops][1],y[total_loops][1]);
        fprintf(cl,"%f %f\n",x[total_loops][1],y[total_loops][1]);
        inc = inc + 10;
       }
      for(j=1 ; j <= (total_points - 1) ; j++)
       {
        if(index[j] == 0)
         { fprintf(gcode,"N%d G01 %f %f\n",(20+inc),x[i][j+1],y[i][j+1]);
        fprintf(cl,"%f %f\n",x[i][j+1],y[i][j+1]);
```

```
    inc = inc + 10;
     }
    else
     {
      if(off_eq_index[j] == 0)
    { fprintf(gcode,"N%d G02 %f %f
%f\n",(20+inc),x[i][j+1],y[i][j+1],cir[i][j][4]);
      fprintf(cl,"%f %f %f\n",x[i][j+1],y[i][j+1],cir[i][j][4]);
      inc = inc + 10;
    }
      else
    { fprintf(gcode,"N%d G03 %f %f
%f\n",(20+inc),x[i][j+1],y[i][j+1],cir[i][j][4]);
      fprintf(cl,"%f %f %f\n",x[i][j+1],y[i][j+1],cir[i][j][4]);
      inc = inc + 10;
    }
     }
   }
  j=0;
  j = total_points;
  if(seg_comb_index[total_points] == 1)
   {
    newx = (line[i-1][1][2]*line[i][j][3] - line[i-1][1][3]*line[i][j][2]) /
       (line[i][j][2]*line[i-1][1][1] - line[i][j][1]*line[i-1][1][2]) ;
    newy = (line[i][j][1]*line[i-1][1][3] - line[i-1][1][1]*line[i][j][3]) /
       (line[i][j][2]*line[i-1][1][1] - line[i][j][1]*line[i-1][1][2]) ;
    fprintf(gcode,"N%d G01 %f %f\n",(20+inc),newx,newy);
    fprintf(cl,"%f %f\n",newx,newy);
    inc = inc + 10;
   }
  if(seg_comb_index[total_points] == 2)
   {
     if(line[i][j][1] == 0.0)
      {
       newy =  -line[i][j][3] / line[i][j][2] ;
       intp = cir[i-1][1][1];
       intq = (pow(line[i][j][3],2) / pow(line[i][j][2],2)) -
```

```
          ( (cir[i-1][1][2] * line[i][j][3]) / line[i][j][2] )  +
          cir[i-1][1][3];
        if((pow(intp,2) - 4.0*intq)  < 0.0 )
         { puts("The line-arc do no intersect - Please abort program\n");
           getchar();getchar();
         }
        else
         {
          first_x = ( -intp +  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
          second_x = ( -intp -  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
          optdisa = sqrt( pow(first_x - x[i][1],2) + pow(newy - y[i][1],2) );
          optdisb = sqrt( pow(second_x - x[i][1],2) + pow(newy - y[i][1],2) );
          if( optdisa <= optdisb )
           { newx = first_x; }
          else
           { newx = second_x; }
         }
       }
      else
       {
        intp = pow(line[i][j][1],2) + pow(line[i][j][2],2) ;
        intq = (2.0*line[i][j][2]*line[i][j][3]) -
            (line[i][j][1]*line[i][j][2]*cir[i-1][1][1]) +
            (pow(line[i][j][1],2)*cir[i-1][1][2]) ;
        intr = pow(line[i][j][3],2) -
            (line[i][j][1]*line[i][j][3]*cir[i-1][1][1]) +
            (pow(line[i][j][1],2)*cir[i-1][1][3]) ;
        if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
         { puts("The line and circle do no intersect - Please abort program\n");
           getchar();getchar();
         }
        else
         {
          first_x = ( (-2.0*intp*line[i][j][3]) + (line[i][j][2]*intq) -
             (line[i][j][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
(2.0*line[i][j][1]*intp) ;
          second_x = ( (-2.0*intp*line[i][j][3]) + (line[i][j][2]*intq) +
```

```
                    (line[i][j][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i][j][1]*intp) ;
                first_y = ( -intq + sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp) ;
                second_y = ( -intq - sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp)
;
                optdisa = sqrt( pow(first_x - x[i][1],2) + pow(first_y - y[i][1],2) );
                optdisb = sqrt( pow(second_x - x[i][1],2) + pow(second_y -
y[i][1],2) );
                if( optdisa <= optdisb )
                 { newx = first_x ; newy = first_y; }
                else
                 { newx = second_x ; newy = second_y; }
               }
              }
              fprintf(gcode,"N%d G01 %f %f\n",(20+inc),newx,newy);
              fprintf(cl,"%f %f\n",newx,newy);
              inc = inc + 10;
            }
            if(seg_comb_index[total_points] == 3)
            {
              if(line[i-1][1][1] == 0.0)
               {
                newy =  -line[i-1][1][3] / line[i-1][1][2] ;
                intp = cir[i][j][1];
                intq = (pow(line[i-1][1][3],2) / pow(line[i-1][1][2],2)) -
                   ( (cir[i][j][2] * line[i-1][1][3]) / line[i-1][1][2] )  +
                   cir[i][j][3];
                if((pow(intp,2) - 4.0*intq)  < 0.0 )
                 { puts("The lines do no intersect - Please abort program\n");
                   getchar();getchar();
                 }
                else
                 {
                  first_x = ( -intp +  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
                  second_x = ( -intp -  sqrt(pow(intp,2) - 4.0*intq) ) / 2.0;
                  optdisa = sqrt( pow(first_x - x[i][1],2) + pow(newy - y[i][1],2) );
                  optdisb = sqrt( pow(second_x - x[i][1],2) + pow(newy - y[i][1],2) );
```

```
                if( optdisa <= optdisb )
                 { newx = first_x; }
                else
                 { newx = second_x; }
                }
               }
              else
               {
               intp = pow(line[i-1][1][1],2) + pow(line[i-1][1][2],2) ;
               intq = (2.0*line[i-1][1][2]*line[i-1][1][3]) -
                  (line[i-1][1][1]*line[i-1][1][2]*cir[i][j][1]) +
                  (pow(line[i-1][1][1],2)*cir[i][j][2]) ;
               intr = pow(line[i-1][1][3],2) -
                  (line[i-1][1][1]*line[i-1][1][3]*cir[i][j][1]) +
                  (pow(line[i-1][1][1],2)*cir[i][j][3]) ;
               if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
                { puts("The line and circle do no intersect - Please abort program\n");
                  getchar();getchar();
                }
               else
                {
                first_x = ( (-2.0*intp*line[i-1][1][3]) + (line[i-1][1][2]*intq) -
                   (line[i-1][1][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i-1][1][1]*intp) ;
                second_x = ( (-2.0*intp*line[i-1][1][3]) + (line[i-1][1][2]*intq) +
                   (line[i-1][1][2] * sqrt(pow(intq,2) - (4.0*intp*intr))) ) /
        (2.0*line[i-1][1][1]*intp) ;
                first_y = ( -intq + sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp) ;
                second_y = ( -intq - sqrt(pow(intq,2) - (4.0*intp*intr)) ) / (2.0*intp)
;
                optdisa = sqrt( pow(first_x - x[i][1],2) + pow(first_y - y[i][1],2) );
                optdisb = sqrt( pow(second_x - x[i][1],2) + pow(second_y -
y[i][1],2) );
                if( optdisa <= optdisb )
                 { newx = first_x ; newy = first_y; }
                else
                 { newx = second_x ; newy = second_y; }
```

```
                }
               }
              if(off_eq_index[total_points] == 0)
               {
                fprintf(gcode,"N%d G02 %f %f
%f\n",(20+inc),newx,newy,cir[i][j][4]);
                fprintf(cl,"%f %f %f\n",newx,newy,cir[i][j][4]);
                inc = inc + 10;
               }
              else
               {
                fprintf(gcode,"N%d G03 %f %f
%f\n",(20+inc),newx,newy,cir[i][j][4]);
                fprintf(cl,"%f %f %f\n",newx,newy,cir[i][j][4]);
                inc = inc + 10;
               }
            }
           if(seg_comb_index[total_points] == 4)
            {
             if( (cir[i][j][1] - cir[i-1][1][1]) == 0.0 )
              {
               if( (cir[i][j][2] - cir[i-1][1][2]) == 0.0 )
                { puts("Circles are the same - Please abort\n");
                  getchar() ; getchar();
                }
               else
                {
                 newy = (cir[i-1][1][3] - cir[i][j][3]) / (cir[i][j][2] - cir[i-1][1][2]) ;
                 intq = cir[i][j][1];
                 intr = pow(newy,2) + (cir[i][j][2]*newy) + cir[i][j][3] ;
                 if( (pow(intq,2) - 4.0*intr)  < 0.0 )
                  { puts("The Circles do no intersect - Please abort program\n");
                    getchar() ; getchar();
                  }
                 else
                  {
                   first_x = ( -intq +  sqrt(pow(intq,2) - 4.0*intr) ) / 2.0;
```

```
                second_x = ( -intq -  sqrt(pow(intq,2) - 4.0*intr) ) / 2.0;
                optdisa = sqrt( pow(first_x - x[i][1],2) + pow(newy - y[i][1],2) );
                optdisb = sqrt( pow(second_x - x[i][1],2) + pow(newy - y[i][1],2) );
                if( optdisa <= optdisb )
                 { newx = first_x ; }
                else
                 { newx = second_x ; }
               }
              }
             }
            else
             {
               intm = (cir[i-1][1][3] - cir[i][j][3]) /  (cir[i][j][1] - cir[i-1][1][1]) ;
               intn = (cir[i-1][1][2] - cir[i][j][2]) /  (cir[i][j][1] - cir[i-1][1][1]) ;
               intp = pow(intn,2) + 1.0;
               intq = (2.0*intm*intn) + (cir[i][j][1]*intn) + (cir[i][j][2]) ;
               intr = pow(intm,2) + (cir[i][j][1]*intm) +  cir[i][j][3] ;
               if( (pow(intq,2) - (4.0*intp*intr))  < 0.0 )
                { puts("The Circles do no intersect - Please abort program\n");
                  getchar() ; getchar();
                }
               else
                {
                 first_x = ( (2.0*intm*intp) - (intn*intq) +
                     (intn * sqrt(pow(intq,2) - (4.0*intp*intr))) ) / (2.0*intp);
                 second_x = ( (2.0*intm*intp) - (intn*intq) -
                     (intn * sqrt(pow(intq,2) - (4.0*intp*intr))) ) / (2.0*intp);
                 first_y =  ( -intq +  sqrt(pow(intq,2) - (4.0*intp*intr)) ) /
(2.0*intp);
                 second_y =  ( -intq -  sqrt(pow(intq,2) - (4.0*intp*intr)) ) /
(2.0*intp);
                 optdisa = sqrt( pow(first_x - x[i][1],2) + pow(first_y - y[i][1],2) );
                 optdisb = sqrt( pow(second_x - x[i][1],2) + pow(second_y -
y[i][1],2) );
                 if( optdisa <= optdisb )
                  { newx = first_x ; newy = first_y ; }
                 else
```

```
                { newx = second_x ; newy = second_y ; }
              }
            }
             if(off_eq_index[total_points] == 0)
              {
               fprintf(gcode,"N%d G02 %f %f
%f\n",(20+inc),newx,newy,cir[i][j][4]);
               fprintf(cl,"%f %f %f\n",newx,newy,cir[i][j][4]);
               inc = inc + 10;
              }
             else
              {
               fprintf(gcode,"N%d G03 %f %f
%f\n",(20+inc),newx,newy,cir[i][j][4]);
               fprintf(cl,"%f %f %f\n",newx,newy,cir[i][j][4]);
               inc = inc + 10;
              }
          }
        }
       i=2;j=0;
       for(j=1 ; j <= (total_points) ; j++)
        { if(j == total_points)
          {
           if(index[j] == 0)
            { fprintf(gcode,"N%d G01 %f %f\n",(20+inc),x[i][1],y[i][1]);
           fprintf(cl,"%f %f\n",x[i][1],y[i][1]);
           inc = inc + 10;
            }
           else
            {
             if(off_eq_index[j] == 0)
           { fprintf(gcode,"N%d G02 %f %f
%f\n",(20+inc),x[i][1],y[i][1],cir[i][j][4]);
             fprintf(cl,"%f %f %f\n",x[i][1],y[i][1],cir[i][j][4]);
             inc = inc + 10;
           }
             else
```

```
            { fprintf(gcode,"N%d G03 %f %f
%f\n",(20+inc),x[i][1],y[i][1],cir[i][j][4]);
              fprintf(cl,"%f %f %f\n",x[i][1],y[i][1],cir[i][j][4]);
              inc = inc + 10;
             }
              }
            }
            else
            {
             if(index[j] == 0)
              { fprintf(gcode,"N%d G01 %f %f\n",(20+inc),x[i][j+1],y[i][j+1]);
             fprintf(cl,"%f %f\n",x[i][j+1],y[i][j+1]);
             inc = inc + 10;
              }
             else
              {
               if(off_eq_index[j] == 0)
             { fprintf(gcode,"N%d G02 %f %f
          %f\n",(20+inc),x[i][j+1],y[i][j+1],cir[i][j][4]);
               fprintf(cl,"%f %f %f\n",x[i][j+1],y[i][j+1],cir[i][j][4]);
               inc = inc + 10;
             }
               else
             { fprintf(gcode,"N%d G03 %f %f
          %f\n",(20+inc),x[i][j+1],y[i][j+1],cir[i][j][4]);
               fprintf(cl,"%f %f %f\n",x[i][j+1],y[i][j+1],cir[i][j][4]);
               inc = inc + 10;
             }
              }
            }
           }
          i=0;j=0;
          fprintf(gcode,"N%d M00\n",(20+inc));

         }
```