Automatic robot path generation for manufacturing on sculptured surfaces.

Wenwei. Gong
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/1254
INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

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

UMI

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0800
Automatic Robot Path Generation
for Manufacturing on Sculptured Surfaces

By
Wenwei Gong

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through Industrial and Manufacturing System Engineering
In Partial Fulfillment of the Requirements for
The Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
1998

© Wenwei Gong, 1998
APPROVED BY:

Dr. R. Kent (External Examiner)
Department of Computer Science

Dr. W.H. ElMaraghy (Department Reader)
Industrial & Manufacturing Systems Engineering

Dr. R. Du (Supervisor/Chair)
Industrial & Manufacturing System Engineering

DATE: July 9, 1998
ABSTRACT

Robot path planning is an important task in modern manufacturing. Currently, many commercial systems are capable of automatically generating robot paths with collision detection and correction. However, few can deal with automatic robot path generation on free-form surfaces, which has many applications such as tool and die polishing, tool and die welding, and automotive body panel spray painting.

This thesis presents a new prototype system for automatic robot path generation for manufacturing on free-form surfaces. The proposed system consists of three modules: surface data construction, automatic robot path generation, and robot path simulation.

In the surface data construction module, the part surface is represented as an object composed of a large number of data points and each data point on the surface is described by its coordinate. In practice, the surface data can be obtained from CAD systems or CMM measurement.

The automatic robot path generation module is the key of the research. It was written in Visual C++ and runs as an MS Windows 95 application. Based on the surface data, the robot path on the offset surface is first generated. The points on the robot path are defined by their coordinates and orientations, which are defined by the robot frame orientation angles (W, P, R). Next the robot path is checked against collision. The collision avoidance of tool orientation is determined on the basis of the tool shape representative 3D-rectangle and points of the surfaces, including the original surface and the boundary surface. If a potential collision is found, the robot tool orientation will be modified to avoid the collision. Finally, a teachpoint file, a track file, and a Geometry Point (GP) path file are generated automatically.

The robot path simulation is carried out using a commercial software system called Workspace®. Using the teachpoint file, track file and GP path file generated from the robot path generation module, Workspace® simulation enables users to visualize the robot motion to ensure the generated robot path is appropriate.

To validate the new system, several simulation examples are given using a 6 degrees-of-freedom ABB robot (model IRB2000). The simulation results show the new system is effective.
DEDICATION

To my parents
ACKNOWLEDGMENTS

I wish to express my sincere and deep appreciation for my supervisor, Dr. R. Du, who has been giving me encouragement and support from the very beginning of my study in this department and throughout this thesis.

I am also cordially indebted to Mr. A. Calabia, who helped me with the Matlab programming. Much appreciation is also due to Mr. Y. Yuan, software developer from Robot Simulation Ltd., who helped me to solve some technology issues of the robot simulation.

I particularly want to thank Dr. R. Kent for not only serving on my committee, but also giving many advices and helping me in many ways.

Thanks are also due to Dr. W. ElMaraghy, my committee member, for supporting my work and giving previous advice on preparing this thesis.

Thanks are also due to Ms. Jacque Mummery, Mr. Dave McKenzie, R. Barakat, staffs of Industrial & Manufacturing Systems Engineering, for their support and friendly assistance during my study in this department.

The encouragement and suggestions from my group members are never unforgettable. Finally, I convey my deep gratitude to all others not named here who helped me in various ways.
Table of Contents

ABSTRACT ........................................................................................................... iii
DEDICATION ......................................................................................................... iv
ACKNOWLEDGEMENT ......................................................................................... v
LIST OF FIGURES .............................................................................................. ix
LIST OF TABLES ................................................................................................. xi

CHAPTER

I INTRODUCTION .............................................................................................. 1
  1.1 Background and overview of industrial robots ........................................ 1
  1.2 Literature survey ...................................................................................... 2
  1.3 Motivation and objective of the research .............................................. 3
  1.4 Organization of the thesis ....................................................................... 5

II A BRIEF DESCRIPTION OF INDUSTRIAL ROBOT ....................................... 6
  2.1 Degree of freedom (DOF) ...................................................................... 6
  2.2 Types of robot applications .................................................................. 7
  2.3 End effectors .......................................................................................... 8
  2.4 Robot programming ............................................................................... 10
    2.4.1 Programming of robots .................................................................... 10
    2.4.2 On-line programming ...................................................................... 11
    2.4.2 Off-line programming ...................................................................... 11
    2.4.3 Programming languages of industrial robots .................................. 12
  2.5 Robot simulation ................................................................................... 14
    2.5.1 Graphical simulation ....................................................................... 14

vi
2.5.2 A brief description of Workspace ..................................15
2.5.3 Frames of reference ..................................................15
2.5.4 Using teach points (TPs) ..............................................16
2.5.5 Using geometry points (GPs) ........................................17

III THE PROPOSED METHODS AND RELATED TECHNOLOGIES ..........19

3.1 Object oriented programming .........................................19
   3.1.1 Introduction .......................................................19
   3.1.2 Main objects in the proposed system .........................20
   3.1.3 Some related Visual C++ technologies .......................24

3.2 Surface representation .................................................28
   3.2.1 Surface representation ...........................................28
   3.2.2 Robot tool representation .......................................32

3.3 Collision avoidance algorithm .......................................32
   3.3.1 Principle ..........................................................33
   3.3.2 Related mathematics algorithm ..................................34
   3.3.3 Collision avoidance calculating ................................36

IV IMPLEMENTATION OF THE PROPOSED SYSTEM ......................39

4.1 Overview of the System .................................................39
4.2 Required Data for programming .......................................39
4.3 Programming implementation .........................................40
   4.3.1 Overview ..........................................................40
   4.3.2 Main menu ........................................................42
   4.3.3 Project ............................................................43
LIST OF FIGURES

Figure 2.1: Illustration of an ABB IRB2000 robot ........................................7
Figure 2.2 Welding tool ..............................................................................9
Figure 2.3 Frames of reference ..............................................................16
Figure 3.1 A free-form surface ..............................................................29
Figure 3.2 Original/boundary and offset surface ....................................31
Figure 3.3 Rectangular representation of robot end-effector ..................32
Figure 3.4 Point inside the robot tool triangular .....................................33
Figure 3.5 Robot tool collision avoidance ................................................34
Figure 3.6 Roll, pitch and yaw .................................................................35
Figure 3.7 Top view of the welding process ..........................................38
Figure 3.8 Illustration of the collision correction calculation procedure ....38
Figure 4.1 System models .....................................................................39
Figure 4.2 Required data .....................................................................39
Figure 4.3 Automatic robot path generation .........................................40
Figure 4.4 Diagram of the programming system ....................................41
Figure 4.5 Main menu ............................................................................42
Figure 4.6 Main menu interface in Visual C++ .......................................42
Figure 4.7 Initial project ........................................................................43
Figure 4.8 parameters setup .................................................................45
Figure 4.9 Diagram of the path generation .............................................46
Figure 4.10 Original and offset surface ..................................................47
Figure 4.11 Welding on a convex surface ........................................48

Figure 4.12 Welding on a convex hull with boundary surface .................49

Figure 4.13 Welding on a convex hull ...........................................49
LIST OF TABLES

Table 1.1: A list of major robot manufacturers and their programming languages ......2
Table 3.1 Project initializing object .................................................................20
Table 3.2 Surface points object ........................................................................21
Table 3.3 Parameters setup object .................................................................22
Table 3.4 Parameters setup ............................................................................23
Chapter 1 Introduction

1.1 Background and overview of industrial robots

Since the first robots were introduced to the manufacturing industry in 1961, there has been a rapid growth in the number of robots used in a wide range of applications. Today robots play an essential role in industrial and manufacturing engineering.

In general, robots may substitute for human operators in the following situations [32]:

- Hazardous work environment for human operators;
- Repetitive and heavy work cycles;
- Parts difficult to handle by human operators;
- Infrequent product changeovers;
- Part positioning and orientating;

Enormous research and development activities have been carried out. In developed countries, many large-scale national and international research and development programs have been launched and are still going on. For example, ESPRIT, BRITE, and EUREAKA are large European research programs that focus on robotics. NASA and the European Space Agency also conduct robotics research for tele-operations in outer space. The governments in United States, Canada, and Japan are currently funding large research projects to develop the so-called fourth-generation robots. In the private sector, hundreds of companies are manufacturing and servicing robots of various types, as well as supporting robotic systems and peripherals [40].
Many commercially available industrial robots are widely used in manufacturing and assembly tasks, such as material handling, spot/arc welding, parts assembly, paint spraying, loading and unloading numerically controlled machines. Some robots are also used in space and undersea exploration, and prosthetic arm research. According to a search on the Internet, there are over 100 robot manufacturers around the world. Table 1.1 is a list of some selected robot manufacturer and their programming languages.

Table 1.1: A list of major robot manufacturers and their programming languages

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Programming languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABB</td>
<td>ARLA, RAPID</td>
</tr>
<tr>
<td>Adept</td>
<td>V+</td>
</tr>
<tr>
<td>Comau</td>
<td>PDL2</td>
</tr>
<tr>
<td>Eshed</td>
<td>ACL</td>
</tr>
<tr>
<td>Fanuc</td>
<td>RG2, Karel 2, Karel 3, TP</td>
</tr>
<tr>
<td>IBM</td>
<td>AML/2</td>
</tr>
<tr>
<td>Kawasaki</td>
<td>AS</td>
</tr>
<tr>
<td>Motoman</td>
<td>Inform 1, Inform 2</td>
</tr>
<tr>
<td>Nachi</td>
<td>SLIM</td>
</tr>
<tr>
<td>Panasonic</td>
<td>Parl-1</td>
</tr>
<tr>
<td>PSI</td>
<td>PSI</td>
</tr>
<tr>
<td>RTX</td>
<td>FRTX</td>
</tr>
<tr>
<td>Samsung</td>
<td>FARAL-II</td>
</tr>
<tr>
<td>Seiko</td>
<td>DARL 4</td>
</tr>
<tr>
<td>Toyoda</td>
<td>V+</td>
</tr>
<tr>
<td>TQ</td>
<td>TQ</td>
</tr>
<tr>
<td>Unimation</td>
<td>VAL I/VAL II</td>
</tr>
</tbody>
</table>

A detailed technical literature review is presented as follows.
1.2 Literature survey

Hundreds of publications can be found in the literature. Many of them deal with the control of industrial robots, which improves the robot performance and their application reliability in manufacturing processes [40, 41, 42, 12, 25]. The other category is simulation. Recently, processing capability of personal computers and software environments, especially in the field of graphical user interfaces, has led to development of off-line programming and 3D simulation robot systems for industrial robots [6, 7, 8, 9, 10, 11, 17].

Regarding robot programming, the current robot can deal with relatively simple geometry, such as workpiece with regular geometry, or simple manufacturing processes, such as loading / unloading and spot welding of reinforcing bars and rails [2, 13, 16, 27, 28]. For example, Moriyasu, Hiramoto, and Ohmine developed a multi-pass welding program for arc welding robots [2], which deals with simple welding planes (flat, vertical and horizontal) and groove shape of joints (fillet, single bevel, V, J joint). Kukareko and et al. presented several algorithms and a prototype software system for simulation of arc and spot welding robotic cells [13]. It is noted that these systems can deal with only seams or dots.

Robot off-line programming and simulation systems have been used as an aid to generate the robot paths. However, such systems cannot generate the robot path automatically. For example, Naoki Asakawa and his colleague [14] developed a robot path generation system for spray painting on complicated surfaces. This system can generate a spraying path based on the information of the part. Its first version only dealt with box-shaped workpiece with uniform paint thickness. Then a new version was
developed to deal with sculptured surfaces [18]. This system can automatically generate a spraying-painting path based on CAD data of the workpiece. However, it did not consider the collision problem.

Ge, Takeuchi and Asakawa studied the generation of collision-free robot paths for a polishing robot operating on complicated curved surfaces in 1993 [15]. To obtain the collision-free polishing path, the potential for local collision was checked, and the collision avoidance tool axis vector was determined based on a solid model of the workpiece and the tool shape representative points. This system was further enhanced with more functions, such as tool selection [19, 20, 21, 22].

According to the literature survey, there are no systems that are able to provide a fully automatic robot path generation on free-form surfaces with the consideration of offset surface and boundary surfaces.

1.3 Motivation and Objective of the research

Robot path generation is an important task in modern manufacturing. Currently, many commercial systems are capable of automatically generating robot paths with collision detection and correction. However, they are limited in dealing with sculpture surfaces. With the rapid progress of personal computers (not only the hardware but also the software), it is possible to develop an automatic robot path generation system for manufacturing on sculptured surfaces.

As an example, in the tool and die industry, owing to the nature of the applications, tooling wears out and needs to be repaired periodically. In addition, in the course of making tools and dies, it is not uncommon to encounter errors due to
miscalculation as well as inaccurate setup and cutting. These errors are typically localized errors and the most effective means of correction is welding. Currently, there are limited research and development activities in this area because the welding zone has a complicated structure and is scattered on sculptured surfaces. Welding on tools and dies has been mainly conducted manually up to this point. It has a number of constraints, such as requiring highly skilled laborers, poor working environment and difficulty in controlling the quality.

This is the motive for the development of an automatic robot path generation system. Such a system will not only fill the technology vacancy in the tool and die industry but also can be used in many other manufacturing systems.

1.4 Organization of the thesis

This thesis is organized as follows. Chapter 2 provides a brief discussion of the industrial robot, its programming and simulation. Chapter 3 contains a description of the proposed methods and related technologies, including object-oriented programming, surface representation, and collision-detection and correction algorithms. Chapter 4 presents a description of the implementation of the proposed system, including an overview of the system, required data for programming and programming implementation. Three examples are presented to verify the implementation of the system. Chapter 5 contains conclusions, and recommendations for future research are presented.
Chapter 2 A Brief Description of Industrial Robot

An industrial robot is a general-purpose, programmable machine designed with certain anthropomorphic characteristics. According to the Robotics Industries Association (RIA), the industrial robot is defined as follows: [B9]

An industrial robot is a re-programmable, multifunctional manipulator designed to move materials, parts, tools, or special devices through variable programmed motions for the performance of a variety of tasks.

In this chapter, a brief description of an industrial robot is given.

2.1 Degree of Freedom (DOF)

Every mechanical point on a robot, except in the gripper or tool, at which some form of drive induces motion in a robot part is called a degree of freedom or axis. In the example of our simulation, an IRB2000 model with 6 degrees of freedom is used. Six degrees of freedom are, in general, sufficient for a robot to reach any position and orientation within its reachable domain. Figure 2.1 is the illustration of this model with six links as follows:

- A robot object (base) - for the "waist" axis
- Link1 - for the "shoulder" axis
- Link2 - for the "elbow" axis
- Link3 - for the "roll" axis
- Link4 - for the "pitch" axis
- Link5 - for the "yaw" axis
• Link6 - for the end effector frame

Figure 2.1: Illustration of an ABB IRB2000 robot

2.2 Types of Robot Applications

Most of the current applications of industrial robots are in manufacturing. The applications can usually be classified into the following three categories: [32]

• Material handling
• Processing operations
• Assembly and inspection

Here we focus on processing operations. Processing applications are those in which the robot performs a processing operation on parts. A distinguishing characteristic
of this category is that the robot is equipped with some type of tool as its end effector. To perform the processes, the robot must manipulate the tool relative to the part during the work cycle. Examples of industrial robot application in the processing category include spot welding, continuous arc welding, spray painting, various machining and other rotating spindle process, and others.

2.3 End effectors

Industrial robots usually do not come equipped with grippers or other tools. The wide variety of tools that can be attached to the end of a robot arm is one of the keys to the robot's great versatility.

Tools and grippers are the two general categories of end effectors, the special devices that attach to the manipulator's wrist to enable the robot to accomplish a specific task. Tools are used in applications where the robot must perform some processing operation on the workpiece. The robot, therefore, manipulates the tool relative to a stationary or slowly moving object. Examples of the tools used as end effectors by robot to perform processing applications include:

- Spot welding gun
- Arc welding tool
- Spray painting gun
- Rotating spindle for drilling, routing, grinding, etc.
- Assembly tool (e.g., automatic screwdriver)
- Heating torch
- Water-jet cutting tool
In each case the robot must not only control the relative position of the tool with respect to the work as a function of time, it must also control the operation of the tool. For this purpose, the robot must be able to transmit control signals to the tool for starting, stopping, and otherwise regulating its actions.

In later simulation examples, a welding tool which was designed by Robot Simulation Ltd. is attached to the IRB2000 model (see figure 2.2).

Figure 2.2 Welding tool
2.4 Robot Programming

2.4.1 Programming of robots

It is the programmability of robots that gives them their great versatility and stimulates the creativity of automation engineers to use them in industrial applications. To do useful work, a robot must be programmed to perform its motion cycle. A robot program can be defined as instructions for the manipulator to follow a path in, combined with peripheral actions that support the work cycle. Examples of the peripheral actions include opening and closing the gripper, performing logical decision making, and communicating with other pieces of equipment in the robot cell. Just as in numerical control, there are several different ways to program a robot.

The most powerful and versatile of programming methods use computer-like robot programming languages. The use of computer-type programming languages became an appropriate programming method as digital computers took over the control function in robotics. Their use has been stimulated by the increasing complexity of tasks that robots are being called on to perform, with the need to embed logical decisions into the robot work cycle.

Whatever the job is, industrial robots need to be programmed. There are basically two methods for programming industrial robots; namely, on-line programming and off-line programming.
2.4.2 On-line Programming

On-line programming can be done by using either the walk-through or the lead-through method. In the walk through method, the programmer moves the robot manually, while information on position, velocity and other related variables are recorded by the robot's control system. This recorded motion can be played back whenever required. In the lead-through method, the robot is moved to a number of desired positions by actuating its drive mechanism, and these positions are recorded by using the teach pendant. The last recorded position is compared with the previous position and the appropriate move command is automatically generated by the control system of the robot.

In on-line programming, the human programmer may be exposed to an unpleasant atmosphere and the motions taught are, at best, at the programmer's skill level. This method seems to be suitable for pick-and-place or materials handling type of tasks, but it becomes a time-consuming process for operations like arc welding or spray painting, in which the entire path must be taught.

2.4.2 Off-line Programming

In off-line programming, new robot programs can be prepared on a computer and downloaded to a robot without interrupting production. The major benefits of off-line programming include reduced downtime, giving greater capability to robots, better understanding of process through simulation, and reduced risk of damage to expensive equipment or of injury to operator. Off-line programming can be done either by using textual languages or graphics-based simulation and programming systems.
With increased use of industrial robots in a variety of production applications, great attention has been devoted to investigation and development of off-line programming of industrial robots. Advanced integrated off-line programming systems include a CAD modeler, and, many systems contain the components for a geometric modeler and graphic animation system, an off-line programming language and simulator, and an interface to the target robot system. In general, off-line programming languages should have the following capabilities:

- User-definable tasks and subroutines
- User-definable end-effectors and robot arms
- Complex data structures and predefined state variables
- Coordinate transformation between frames
- Runtime definition of variables
- High level instructions for tactile sensors and vision
- Decision-making capabilities allowing the robot to recover from unexpected events
- Use of CAD data

2.4.3 Programming languages of industrial robots

Presently, there are hundreds of commercially available robot languages. Most of them are based on such classical programming languages as Pascal, C, Modula-2, BASIC, and Assembler. The robot programming languages can be classified according to the robot reference model, the type of control structure used for data, the type of motion
specification, the sensors, the interfaces to external machines, and the peripherals used.

The following types of robot programming languages are available:

- Point-to-point motion languages
- Basic motion languages at the Assembler level
- Non-structured high level programming languages
- Structured high level programming languages
- NC-type languages
- Task-oriented languages

The Karel language, developed by GM Fanuc Robotics Corporation for use on GM Fanuc robots, was used in later examples to perform the operations of the robot. It has been adapted for Workspace by extending the language to include features designed for simulation in addition to its robot control function. It allows for the creation of sophisticated structured programs with the following features. [48]

- Simple and structured data types
- Arithmetic, relational and Boolean operators
- Control structures for loops and selections
- Condition handlers
- Procedure and function routines
- Motion control statements
- I/O operations
- User interface functions
2.5 Robot simulation

In advanced manufacturing, it is desirable to simulate the manufacturing systems and processes before installing physical equipment. The simulation enables us to rectify errors in design and manufacturing processes before the manufacturing system is set on the floor. The animation and simulation are packaged multimedia production with graphical/textual information.

2.5.1 Graphical simulation

A graphical simulation system for the validation and specification of the robot program is an integral part of an advanced programming system. It must provide a library of emulated robots, transport devices, and end-effectors to build up a cell model quickly. There must be available modeling software for the robot's environment and of the robot itself. The graphic representation enables the operator to check the programmed operation. Accordingly, the systems contain specific tools for:

- Robot modeling
- World modeling
- Description of motion
- Collision detection
- Control code generation

There are a number of graphical simulation systems available in the market, such as MC Autoplace, ROBCAD, CATIA, ADAMS, and Workspace. Workspace 4 was used here to do the simulation.
2.5.2 A brief description of Workspace

Workspace is a sophisticated simulation system designed specifically for the robotics industry. This software package creates and simulates robot programs in the native language of the robot. It can:

- quickly model workcell layouts and evaluate their performance;
- communicate the design concept using state-of-the-art 3D graphics;
- test reach and detect collisions;
- generate robot programs off-line using a mouse driven menu system;
- optimize programming using high-level Workspace functions;
- download programs to a robot controller without the need for post-processing.

In the simulation examples, IRB2000 model and its effector, a welding tool, are created in Workspace. Some basic concepts about Workspace are described later in this section.

2.5.3 Frames of reference

Most of the movement commands in Workspace are dependent on using positions defined relative to co-ordinate frames. Each frame uses the right-hand Cartesian co-ordinate system and rotation about axes is defined using the right-hand rule. There are four frames of references (see Figure 2.3):

- World
- Robot world
- User frame
- User tool
Figure 2.3 Frames of reference

By default, $\text{SUFRAME}$ (position of user frame relative to robot world) and
$\text{SUFOOL}$ (position of the end of tool relative to robot world) are set to equal $\text{SNILP}$, a
position and orientation of $(0,0,0,0,0,0)$.

2.5.4 Using teachpoints (TPs)

Teachpoints are 3D stored positions and orientations. TPs are stored in a
teachpoint file and are linked to a track file when simulating. They are independent of
objects, are language dependent, and can be downloaded to a robot controller. When
using Karel 2 (the default simulation language supplied with Workspace) the format of a
teachpoint is as follows:

$$\text{TPName} = \text{POS}(x, y, z, a, b, c, 'configuration')$$
This is the format written into the teachpoint file. POS is a function which returns a POSITION type variable. The parameters x, y and z are 3D Cartesian values of the teachpoint location. The parameters a, b and c are the three angles of W, P and R which define the rotation around OX, OY and OZ axes and are also called yaw, pitch and roll. As for the configuration string, most robots can achieve a given position and orientation in several different ways, for example with the elbow pointing up or down, or with the shoulder on the left or right of the base. The inverse kinematics algorithm distinguishes between these different solutions by using configuration indicators. So the configuration string in the parameters is used for this purpose. As the robot is moving, Workspace tries to choose a solution to the inverse kinematics algorithm that will maintain the same configuration indicators as the robot's initial position unless a new configuration is specified in the target position variable.

2.5.5 Using geometry points (GPs)

Geometry points are similar to teachpoints in that they are 3D positions used to define movements of a robot. They can be defined using data from the objects within the workcell. However, they differ from TPs in a number of ways:

- GPs are stored within your robot model.
- GPs are attached to objects.
- GPs must be converted to TPs before being downloaded to a robot controller.

The robot controller has no knowledge of objects within its workcell and relies entirely on teachpoint and movement commands to perform its tasks. GPs provide features which
can make it much simpler to program robots on the simulation than using a teach pendant in the real world.

A very powerful feature of Workspace is to use geometry points to define 3D paths for robots. GPs can be stored in GP path file with the following formats:

<GP name>
Pos=<X>, <Y>, <Z>, <W>, <P>, <R>
<motion type>, Term=<termination type>, Vel=<velocity>, Acc=<acceleration>,
Wait=<time delay>, Rout=<subroutine call>, <signal information>, <tool action>
Chapter 3 The Proposed Methods and related technologies

The implementation of the proposed system relies on several developed technologies, such as object-oriented programming (OOP) and template-based classes of Visual C++. The related technologies in Visual C++ and the proposed methods, such as surface representation and collision detection, are discussed here.

3.1 Object oriented programming

3.1.1 Introduction

The Object-Oriented Programming (OOP) technology is now widely used because of its features listed below: [29]

- Encapsulation
- Information/Implementation hiding
- State retention
- Object identity
- Messages
- Classes
- Inheritance
- Polymorphism
- Genericity

Using the OOP technology, we have:

- High expressiveness of the modules in the system, by which the complex engineering objects can be realized using simple language
• Good relationships between modules
• Neutrality, or independence

In the proposed system, Visual C++ is used to design the system of automatic robot path generation.

3.1.2 Main objects in the proposed system

In the proposed system, several objects are used including the project initializing object, surface point object, parameter setup object, and etc.

1. Project initializing object

This object is used to construct and modify the initial information and save it in a file (see table 3.1).

<table>
<thead>
<tr>
<th>CProjectInit : public CDialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>OnInitsave(); OnInitDialog()</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

As shown in Table 3.1, the object contains several variables: m_Name, m_Date, m_Operator, m_Customer and m_Note store the information of the project; m_IsNew, m_IsModify and m_IsModifyOK are the signs used by the “New” and “Modify” menu item separately; m_ProjectFile is used to store the project name.

The methods, OnInitsave() and OnInitDialog(), are the override functions of the object of Cdialog.
2. Surface points object

Here is the 3D-point object designed for all surfaces.

Table 3.2 Surface points object

<table>
<thead>
<tr>
<th>CW3DPoint: public CObject</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>FormatPOS()</td>
</tr>
<tr>
<td>FormatW3DPoint()</td>
</tr>
<tr>
<td>Serialize()</td>
</tr>
</tbody>
</table>

As shown in Table 3.2, this object contains a number of variables: m_indexI and m_indexJ are the indices of the surface point; (m_x, m_y, m_z) is the location of the data; (m_a, m_b, m_c) is orientation of the robot tool (W, P, R angle).

The method FormatW3DPoint() provides the text format of this object for display or write to text file.

```cpp
void CW3DPoint::FormatW3DPoint(CString & str)
{
    str.Format(_T("(%i, %i, %.2f, %.2f, %.2f, %.2f, %.2f, %.2f)")), m_indexI,
               m_indexJ,
               m_x, m_y, m_z, m_a, m_b, m_c);
}
```

The method FormatPOS() is designed according to the WorkSpace's teach-point file format. So the system can call this method to write a teach-point file easily.
void CW3DPoint::FormatPOS(CString & str)
{
    str.Format(_T("POS(%.2f, %.2f, %.2f, (%.2f, %.2f, %.2f)"),
               m_x, m_y, m_z, m_a, m_b, m_c);
}

The method Serialize() uses the “Serialization” technology provided by MFC
(Microsoft Foundation Class) library to write and read object to and from binary files (see
3.1.3).

3. Parameters setup object

This object uses CPropertySheet as an interface to setup parameters. A property
sheet is a special kind of dialog box that is generally used to modify the attributes of
some external object, such as the current selection in a view. The property sheet here,
CSetupPSheet, consists of a CPropertySheet object and three CPropertyPage objects,
CSetupRLocation, CSetupWelding and CSetupRobot (see table 3.3). The property sheet is
displayed by the framework as a window with a set of tab indices, with which the user
selects the current page, and an area for the currently selected page.

Table 3.3 Parameters setup object

<table>
<thead>
<tr>
<th>CSetupPSheet : public CPropertySheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSetupRobot : public CPropertyPage</td>
</tr>
<tr>
<td>CSetupWelding : public CPropertyPage</td>
</tr>
<tr>
<td>CSetupRLocation : public CPropertyPage</td>
</tr>
<tr>
<td>Methods: OnInitDialog(), OnOK()</td>
</tr>
<tr>
<td>Variables: See table 3.4</td>
</tr>
</tbody>
</table>

22
The variables store all the parameters of robot, geometry of end-effector and manufacturing, and, robot initial location. All the needed parameters are listed in Table 3.4.

Table 3.4 Parameters setup

<table>
<thead>
<tr>
<th>Robot</th>
<th>Geometry</th>
<th>Robot initial location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of end-effector</td>
<td>End-effector Geometry: Long</td>
<td>X</td>
</tr>
<tr>
<td>Suspend time</td>
<td>Wide</td>
<td>Y</td>
</tr>
<tr>
<td>Speed</td>
<td>High</td>
<td>Z</td>
</tr>
<tr>
<td>UFRAME</td>
<td>Manufacturing thickness</td>
<td>A</td>
</tr>
<tr>
<td>UTOOL</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>USEMAXCCEL</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configuration</td>
</tr>
</tbody>
</table>

The methods OnInitDialog() and OnOK() are the override functions of CpropertyPage. OnInitDialog() is used to initialize the interface of the current page, which includes reading the parameters from the initialization file and writing them to the interface. OnOK() saves the input or updated parameters in the initialization file when the object is closed.
3.1.3 Some related Visual C++ technologies

1. Using initialization file

The initialization file is used to store all the setup parameters in the proposed system for its convenience to be operated on. Here is the structure of the initialization file that stores the setup parameters:

```
[Project]
Name=try4

[Robot Setup]
Acceleration=
Time Delay=
Speed=
Motion Type=
Tool Action=
Termination Type=
Robot Name=

[Robot Frame]

[Geometry]
```

"Project" is the section name, "Name" is the key.

In the program, two functions are used to operate on the initialization file. The "WritePrivateProfileString" function copies a string into the specified section of the initialization file. The "GetPrivateProfileString" function retrieves a string from the specified section in an initialization file. Following is an example of how to use these two functions.
CString savestr;

savestr.Format("%.2f", m_initial_x);

WritePrivateProfileString ("Robot Initial Coordinate", "initial_x",
            savestr, initialfile);

And

CHAR inBuf[80];

GetPrivateProfileString ("Robot Initial Coordinate", "initial_x",
            "0.00", inBuf, 80, initialfile);

2. Serialization

Serialization mechanism is provided in the Microsoft Foundation Class Library (MFC) to allow objects to persist between runs of your program. It is the process of writing or reading an object to or from a persistent storage medium, such as a disk file. MFC supplies built-in support for serialization in the class CObject.

Serialization handles all the details of object pointers and circular references to objects that are used when you serialize an object. MFC uses an object of the CArchive class as an intermediary between the object to be serialized and the storage medium. This object is always associated with a CFile object, from which it obtains the necessary information for serialization, including the file name and whether the requested operation is a read or write. The object that performs a serialization operation can use the CArchive object without regard to the nature of the storage medium.

The following class declaration fragment shows the new member variables and the declaration for the overridden Serialize member function:

25
class CW3DPoint : public CObject
{
    :
    :
    public:
    virtual void Serialize(CArchive& ar);
    :
}

The following example shows how to implement Serialize for the CW3DPoint class declared above:

void CW3DPoint::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << m_index; // add storing code here
    }
    else
    {
        ar >> m_index; // add loading code here
    }
}

Whenever reading or writing objects into a binary file, the procedure is to create a CArchive object, and attach it to an object of class CFile that represents an open file. This can be explained as below:
CFile file;

file.Open("filename", CFile::modeCreate|CFile::modeWrite, &e));

CArchive ar(&file, CArchive::store);
ar<<w3dPoint; // an object of CW3DPoint

3. Template-Based Classes

Using Template-Based Classes to create type-safe collections is more convenient and provides better type safety than using the collection classes not based on templates.

MFC predefines two categories of template-based collections:

- Simple array, list, and map classes: CArray, CList, CMap
- Arrays, lists, and maps of typed pointers: CTypedPtrArray, CTypedPtrList, CTypedPtrMap

The simple collection classes are all derived from class CObject, so they inherit the serialization, dynamic creation and other properties of CObject. The new collection class inherits from the specified base class, and the new class's member functions use encapsulated calls to the base class members to enforce type safety.

In the proposed project, CTypedPtrArray was used to store the large amount of surface data. The typed-pointer array, CTypedPtrArray, takes two parameters: BASE_CLASS and TYPE. These classes can store any data type, which you specify in the TYPE parameter. They were derived from one of the non-template collection classes that store pointers; this base class is specified in BASE_CLASS. For arrays, one can use either CObArray or CPtrArray. Here is an example:
typedef CTypedPtrArray<CObArray,CW3DPoint*> CW3DPointArray;

Benefits from the use of CTypedPtrArray are:

- The CTypedPtrArray class provides a type-safe "wrapper" for objects of class CObArray. When using CTypedPtrArray rather than CPtrArray or CObArray, the C++ type-checking facility helps eliminate errors caused by mismatched pointer types.

- In addition, the CTypedPtrArray wrapper performs much of the casting that would be required if CObArray is used.

- Because all CTypedPtrArray functions are inline, use of this template does not significantly affect the size or speed of program code.

3.2 Surface and robot tool representation

In the proposed system, the surfaces are represented by a large number of data points. The robot tool was represented by a rectangular prism. What follows are the detailed descriptions.

3.2.1 Surface representation

In practice, the data that represent the workpiece can be obtained from various sources such as the design using CAD/CAM systems, or data map from CMM machines or laser scanning. In the proposed system, it is assumed that the data is represented in the form of surface mesh. In other words, a surface model can be built from an object model
composed of a large number of data points and each data point on the surface is described by its coordinate. As an example, Figure 3.1 shows the data points on a free-form surface. (The unit in the figure is mm.) Consequently, the system can deal with sculptured surfaces no matter how complicated they are.

![Figure 3.1 A free-form surface](image)

Data points used to represent sculptured surfaces may be approximated using classical curve/surface representation techniques such as polynomial Bezier, Spline, B-spline, and non-uniform rational B-spline (NURBS) [3, 4, 5]. In the proposed system, a Bezier surface model is used to generate the original or boundary surfaces.

In principle, the robot path generation is the same as the cutter path generation in a finishing machining. According to the literature, there are a number of methods for tool path generation for finish machining [44, 45, 46, 47]. In the presented research, a simple offset method is adopted. Assuming that the data map of a workpiece surface consists of (n+1) x (m+1) data points, then the surface can be approximated by a Bezier surface as defined below: [30]
\[ P(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u) B_{j,m}(v) P_{i,j} \]  
(3.1)

Where, \( u, v \in [0, 1] \) are parameters, \( P_{ij} \)s are the control data points, and

\[ B_{i,n}(x) = \frac{n!}{i!(n-i)!} x^i (1-x)^{n-i} \]  
(3.2)

is the Bernstein polynomials. Equation (3.1) can be written in a matrix form:

\[ P(u, v) = [U][M][G][M]^T [V]^T \]  
(3.3)

Where, \([G]\) is the matrix of the control data points, \([M]\) is a constant matrix, \([U]\) and \([V]\) are parameter vectors. Furthermore, suppose the offset distance is \( d \), then the points on the offset surface can be determined as follows:

\[ P_{\text{offset}} = P_{\text{original}} + d \cdot \vec{N} \]  
(3.4)

Where, \( \vec{N} \) is the normal vector of the original surface at the point, and is defined by

\[ \vec{N} = \frac{P_u \times P_v}{|P_u \times P_v|} \]  
(3.5)

Here, \( P_u \) and \( P_v \) are the partial derivatives:

\[ P_u = \frac{\partial P(u, v)}{\partial u} = \left[ \frac{\partial U}{\partial u} \right] [M][G][M]^T [V]^T \]  
(3.6)

\[ P_v = \frac{\partial P(u, v)}{\partial v} = [U][M][G][M]^T \left[ \frac{\partial V}{\partial v} \right]^T \]  
(3.7)

As an example, Figure 3.2 shows a sculptured surface (original/boundary) and the offset surface. (The unit in the figure is mm.)
It should be pointed out that the above method is applicable only to relatively simple smooth sculptured surfaces. For complicated sculptured surfaces, various issues must be considered such as loops and singular points.

The offset surface is very important for the robot path generation on which the tool position and orientation are determined. For robot welding, the welding tool must be on the offset surface, though its orientation may be varied. For robot grinding, the offset surface is the original surface (i.e., $d = 0$). The position and the orientation of the grinding tool must be aligned to the offset surface.
3.2.2 Robot tool representation

In order to facilitate the calculation of robot path, a robot tool is represented by several key points. For example, as shown in Figure 3.3, a welding tool can be represented by a rectangular prism with 8 corner points. The rectangle completely encloses the tool. Hence, if the rectangle is not gouged into the workpiece, there will be no collisions between the robot tool and the workpiece.

![Figure 3.3 Rectangular prism representation of robot tool](image)

3.3 Collision avoidance algorithm

In general, two types of collisions may occur: local collision and global collision. The former is defined as the collision between the tool and the workpiece (including its original surface and the boundary surface) and the latter is defined as the collision between the robot arm and the workpiece. In this context, only the former is considered and referred to as collision.

Potential collision between the robot tool and the workspace must be detected and corrected in order to carry out the manufacturing process automatically. To obtain the collision-free path, the collision is subjected to checking and the tool orientation for
collision avoidance is determined on the basis of the tool shape representative surfaces and workpiece surfaces, including the original and boundary surface. In this section, the principle and algorithm for calculating collision-free motion paths will be described.

3.3.1 Principle

The collision problem is detected between the robot tool and the workspace surface (not only the original surface, but also the boundary, if exists). In principle, if any points on these surfaces fall inside the robot tool rectangle, a collision happens. So to detect a collision is to detect whether any points of the workpiece surfaces are in the tool representative rectangular prism. For every point, there are three situations, the point is inside (see Figure 3.4), on and outside the robot tool rectangular prism. If all the points fall outside the robot tool rectangular prism, no collision exists. For the other two situations, there is a predicted collision between the robot tool and the surfaces. In order to avoid the collision problem, the tool orientation will be adjusted by changing W, P and R angles in the robot frame.

![a. 3D view](image1)

![b. 2D view](image2)

*Figure 3.4 Point inside the robot tool triangular*
Figure 3.5 illustrates this method. If a collision happens (Figure 3.5(b)), the robot tool will change its orientation (Figure 3.5(c)) until no collision exists.

![Image of robot tool collision avoidance]

a. No collision  

b. Collision happens  

c. Orientation adjusting

Figure 3.5 Robot tool collision avoidance

Below some related mathematical algorithms are described in detail.

3.3.2 Related mathematical algorithm

In this section the related mathematical algorithms are discussed.

1. Formation of a plane though 3 points

With the three points $Q_1$, $Q_2$ and $Q_3$, 

\[
Q_1 = \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} \quad Q_2 = \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} \quad Q_3 = \begin{bmatrix} X_3 \\ Y_3 \\ Z_3 \end{bmatrix}
\]

A surface, $\Delta Q$, can be found by the following equation:

\[
A_qX + B_qY + C_qZ + D_q = 0
\]  

(3.8)

Where the coefficients $A_q$, $B_q$, $C_q$ and $D_q$ can be found based on the determinant:
\[
\begin{vmatrix}
X & Y & Z & 1 \\
X_1 & Y_1 & Z_1 & 1 \\
X_2 & Y_2 & Z_2 & 1 \\
X_3 & Y_3 & Z_3 & 1 \\
\end{vmatrix} = 0
\]  
\[(3.9)\]

Hence:
\[
A_q = (Y_2Z_3 - Y_3Z_2) + (Y_3Z_1 - Y_1Z_3) + (Y_1Z_2 - Y_2Z_1)
\]  
\[(3.10)\]
\[
B_q = (X_3Z_2 - X_2Z_3) + (X_1Z_3 - X_3Z_1) + (X_2Z_1 - X_1Z_2)
\]  
\[(3.11)\]
\[
C_q = (X_3Y_2 - X_2Y_3) + (X_1Y_3 - X_3Y_1) + (X_2Y_1 - X_1Y_2)
\]  
\[(3.12)\]
\[
D_q = X_1(Y_2Z_3 - Y_3Z_2) + Y_1(X_3Z_2 - X_2Z_3) + Z_1(X_2Y_3 - X_3Y_2)
\]  
\[(3.13)\]

2. Rotation matrix with rotating angles representation

The robot tool can be rotated around 3 axes, OX, OY and OZ. Let the two frames before and after rotation be \(R_0\) and \(R_1\). \(W\), \(P\) and \(R\) are three rotation angles around axis of OX, OY and OZ (see Figure 3.6).

![Roll, pitch and yaw](image)

Then the transformation matrix from \(R_0\) to \(R_1\) is:
\[ M(R_0 \rightarrow R_1) = \begin{bmatrix} \cos P \cos R & \cos R \sin P \sin W - \sin P \cos W & \cos R \sin P \cos W - \sin P \sin W \\ \sin R \cos P & \sin R \sin P \sin W + \cos P \cos W & \sin R \sin P \cos W - \cos P \sin W \\ -\sin P & \cos P \sin W & \cos P \cos W \end{bmatrix} \]  

(3.14)

Let us choose a point P fixed in \( R_0 \) coordinate system to be \((A_i, B_i, C_i)\) and in \( R_1 \) to be \((A'_i, B'_i, C'_i)\), the transformation can be represented as follows:

\[
\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = M(R_0 \rightarrow R_1) \begin{bmatrix} A \\ B \\ C \end{bmatrix} \]

(3.15)

3.3.3 Collision avoidance calculating

In order to avoid the collision problem between the robot tool and the surfaces, two steps are taken. The first step is to check whether there are any surface points falling inside the robot tool rectangular prism. Then the orientation of the robot tool will be adjusted by increasing/decreasing the \( W, P \) and \( R \) angle, until there is no collision.

1. Collision detecting

In this step, the points on the surface will be checked to see whether there are any points on the surface in the robot tool rectangular prism. This can be done by calculating the distances of the points to the six planes of the robot tool rectangular prism. Set the length, wide and height of the rectangular prism as \( L, W \) and \( H \). Let the 6 planes of the tool rectangular prism be:

\[ \Pi_i : A_i X + B_i Y + C_i Z + D_i = 0 \quad (i = L, R, F, E, T, B) \]

(3.16)
Where, \( i = L \) (left), \( R \) (right), \( F \) (front), \( E \) (end), \( T \) (top) and \( B \) (bottom). Then, the

distance of a point, \( P = [X_p, Y_p, Z_p]^T \), to the 6 planes can be calculated using the following

equation:

\[
\text{Dist}(P, \Pi_i) = \frac{|A_i X_p + B_i Y_p + C_i Z_p + D_i|}{\sqrt{A^2 + B^2 + C^2}}, \quad i = L, R, F, E, T, B
\]  

(3.17)

It can be shown that \( P \) falls inside the tool rectangular prism under any of the following

conditions:

\[
D(P, \Pi_L) < L \quad \text{and} \quad D(P, \Pi_R) < L \quad (a)
\]

\[
D(P, \Pi_F) < W \quad \text{and} \quad D(P, \Pi_E) < W \quad (b)
\]

\[
D(P, \Pi_T) < H \quad \text{and} \quad D(P, \Pi_B) < H \quad (c)
\]

Consequently, there will be a collision between the robot tool and the surfaces. Next the

avoidance of this problem will be discussed.

2. Robot tool orientation adjusting

When a collision has been detected, the tool orientation is then adjusted to correct

the collision. Equation 3.15 will be used to calculating the new coordinate values of the

points after adjusting the orientation. See Figure 3.7 (top view of figure 3.5), considering

the practical situation in our design, the tool will go along the direction of \( X \)-axis. So

there are two rotations around \( OY \) and \( OZ \), namely \( P \) and \( R \), these angles need to be

adjusted to avoid the collision.
In general, there are many possible ways to change P and R angles. A simple search method is shown in Figure 3.8. First, the pitch angle P is increased by $\Delta P$ ($\Delta P = 5^\circ$ is used in the study). If the collision is not corrected, then P is increased by $\Delta P$ again and again. If P is increased four times and the collision is still not corrected, then P is set back by $\Delta P$ and the roll angle R is changed by $\Delta R$ ($\Delta R = \pm 5^\circ$). This search scheme continues until the collision is corrected. It is noted that this algorithm is not optimal but will provide a feasible solution for most applications. In most applications, adjusting P angle is enough to correct the potential collisions. At present, in our examples, only P angle is used to solve the collision problem.
Chapter 4 Implementation of the Proposed System

4.1 Overview of the System

The proposed automatic robot path generation for manufacturing on sculptured surfaces is an integrated system consisting of three models, surface data construction, automatic robot path generation and robot path simulation, as shown in figure 4.1.

![Figure 4.1 System models](image)

4.2 Required Data for programming

As for an automatic path generation system, all the needed data (see figure 4.2) for the programming should be given first.

![Figure 4.2 Required data](image)

Before any other operation, the system should read the initial information into the memory. Then the system reads workspace data, including original, boundary and offset...
surface, from other systems. For other data, the system uses the setup parameter object for the user to input all the needed data.

4.3 Programming implementation

In the proposed system, the first module was implemented using Matlab. The second module was developed by Visual C++. The third module was implemented using Workspace®. The second module will be discussed here in more detail.

4.3.1 Overview

The second module of the whole system was developed with Visual C++. Here is the structure of this subsystem.

![Diagram of Automatic robot path generation]

Figure 4.3 Automatic robot path generation

Here, the subsystem reads all the needed surface data from other systems. With these surface data and the setup parameters, such as robot, end-effector geometry and tool operations, this subsystem will check the potential collision, and if any potential collision exists, the tool orientation will be adjusted. Then the robot path, stored in the teach point
file and the track file, will be generated automatically. In addition, the generated path can then be displayed in a list box with all the points the robot will pass. Below is the diagram of the programming system.

![Diagram of the programming system]

Figure 4.4 Diagram of the programming system
The implementation of this subsystem in Visual C++ will be discussed in the later sections.

4.3.2 Main menu

The system is realized using the menu below.

<table>
<thead>
<tr>
<th>Project</th>
<th>Surface</th>
<th>Setup</th>
<th>Path Generation</th>
<th>View</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Display</td>
<td>Parameters</td>
<td>Generation</td>
<td>Tool Bar</td>
<td>About Welding</td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td>Display</td>
<td></td>
<td>Status Bar</td>
<td></td>
</tr>
<tr>
<td>Modify</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.5 Main menu

Below is the Interface of this main menu in Visual C++.

Figure 4.6 Main menu interface presented in Visual C++

The main menu contains 6 items, which will be discussed in more detail below.
4.3.3 Project

In this sub-menu, all the needed information of the project and its management are included. It offers “New”, “Load”, ”Modify”, ”Close”, and ”Exit” menu items.

“New” initializes the project’s information, such as name (the project ID), date, the customer and operator information, and note. For every operation of this system, one and only one project can be opened in the memory. Figure 4.7 shows the interface of this object.

![Initial Project](image)

Figure 4.7 Initial project

If the user clicks “SAVE”, and, if “NAME” and “DATE” are not null, all the information will be saved in a project initialization file along with the needed information for the project, such as subdirectory will be created automatically.
“Load” Loads an existing project into memory, for the user to continue with other operations.

The “Modify” menu item, whose interface is the same as the “New”, is used to modify the information of the current project except the name and date.

“Close” closes the currently opened project. When the current project is closed, no other operations other than the “New” and “Load” can be activated.

4.3.4 Surface

This pull-down menu item only includes a "Display" item. It reads all the needed surface data, including original, boundary and offset surfaces, from other systems and displays these surfaces.

4.3.5 Setup

This pull-down menu item also includes only one item, "Parameters". It sets up all the needed parameters, including the parameters of robot, geometry of end-effector and manufacturing process, and, robot $FRAME and $UTOOL data. Figure 4.8 is its user interface. The interface has three pages shown one at a time. The user clicks the tab at the top to select the page.
4.3.6 Path generation

This pull-down menu includes items of "Generation" and "Display".

"Generation" generates the robot path, the teach-point file and track file automatically.

Figure 4.9 illustrates the robot path generation procedure. In addition, the generated path can be viewed in the list box with text format with the "Display" item.

4.3.7 View and help

"View" provides the convenience for user to display or not to display a tool bar or a status bar in the main menu interface. "Help" gives some information of the system.
Figure 4.9 Diagram of the path generation
4.4 Examples

In order to validate the automatic robot path generation system, a large number of examples, which dealt with robot welding, were tested. Here, three examples are presented using a six degree-of-freedom ABB Robot (model IRB2000).

4.4.1 Example 1

In the previous of this Chapter, we have presented the implementation of the automatic robot path generation. In this section, one example, which is for robot welding on a convex freeform surface without boundary, will be given to illustrate the effectiveness of the system. Figure 4.10 is the original and offset surface (Unit in this figure is mm).

![Figure 4.10 Original and offset surface](image)

Figure 4.11 is the welding process in this example.
4.4.2 Example 2

The robot path of this second example is shown in Figure 4.12. In this example, the workpiece surface is a convex hull with a boundary surface. As the robot moves on, a collision will occur. By applying the collision detection and correction algorithms described in the previous section, the robot path is modified and the collision is corrected. In the figure, the crosses represent the points on the surfaces and the arrow lines represent the robot paths.
4.4.1 Example 3

The third example is shown in Figure 4.13. Although it is similar to the previous example, the surface is more complicated. However, it is also enough to change P angle to correct the potential collision.
Chapter 5 Conclusion and future work

5.1 Conclusion

In this thesis, a PC-based interactive software system of automatic robot path
generation for manufacturing free-form surfaces was developed. This system consists of
three modules: surface data construction, automatic robot path generation, and robot path
simulation.

In this system, the paths of a robot with 6 degrees of freedom are generated
automatically. The robot paths are stored in track file, teach-point file and GP path file
for the robot simulation. The possibility of potential collisions is checked and collision-
free tool orientations are determined based on the surface data and the tool shape
representative surfaces. In addition, the system has the following features:

- This paper presents a prototype automatic robot path generation system for
  manufacturing on sculptured surfaces in a job shop environment. It has many
  potential applications such as painting of automobile body panels and welding of
  molds and dies.
- To detect the collision between the workpiece surfaces and the robot tool, the
  tool is approximated by a rectangular prism.
- To correct the collision, a simple search algorithm is developed. It changes the
  pitch angle repeatedly until there is no collision between the robot tool and the
  workpiece. From many examples, this method can handle most applications.
- The system is written using Visual C++ and Matlab languages. It runs as a
  Windows 95 application.
• The simulation results indicate that the new system generates collision-free robot paths with high accuracy and reliability.

• The collision between the robot tool and the surfaces is detected not only between the robot tool and the original surface, but also with the boundary surface (if exists).

• With the simulation examples, the system is found effective and easy to use for off-line programming of automatic robot path generation for manufacturing free-form surfaces.

5.2 Future research

An automatic robot path generation system for manufacturing free-form surface was developed. However future work are required here to make this system more flexible and powerful.

• An algorithm for computing the offset surface of complicated sculptured surfaces will be developed and implemented.

• An algorithm on how to change the R angle will be developed with respect to the relationship between the robot tool rectangular prism and the normal vector of the surface.

• More complicated surface examples will be given to improve the potential collision correcting algorithm by changing P and R angles.

• OLE technologies will be used to integrate the simulation animation into the Visual C++ program.
References


Microcomputers in Civil Engineering, 12, pp.43-56, 1997.

1997.


1985.


[40] *Robot Technology and Applications*, Ulrich Rembold, editor. Marcel Dekker Inc.,
New York, 1990


Appendix A: Some important classes in Visual C++ program

1. Brief introduction of some main class

1.1 CCnApp

This is the application class. It includes other project specific headers (including Resource.h). This class was created by AppWizard. Below are its member functions and variables. (See cn.h and cn.cpp)

```cpp
CCnApp
  - CCnApp()
  - InitInstance()
  - OnAppAbout()
  - m_server
```

1.2 CmainFrame

This class is derived from CFrameWnd and controls all SDI frame features in the main frame window. Below are its member functions and variables. (See mainfrm.h and mainfrm.cpp)

```cpp
CMainFrame
  - AssertValid()
  - CMainFrame()
  - ~CMainFrame()
  - Dump(CDumpContext & dc)
  - OnCreate(LPCREATESTRUCT lpCreateStruct)
  - OnPathDisplay()
  - OnPathGeneration()
  - OnProjectClose()
  - OnProjectDelete()
  - OnProjectLoad()
  - OnProjectModify()
  - OnProjectNew()
  - OnSParameters()
  - OnSurfaceDisplay()
  - PreCreateWindow(CREATESTRUCT & cs)
  - m_CanBeRun
  - m_ProjectFile
  - m_wndStatusBar
  - m_wndToolBar
```
In this class, the member functions of OnProject(), OnProject(), OnProject(), OnProject() and OnProject() are called by the menu items in the "Project" popup menu. OnSurfaceDisplay(), OnParameters(), OnPathDisplay() and OnPathGeneration() are called by other menu items in the main menu. The member variable m_ProjectFile is used to store the current project name, m_CanBeRun is a boolean variable to save the state of whether any project is loaded, m_wndStatusBar and m_wndToolBar are control bar embedded members.

1.3 ProjectInit

This is the project initializing class and is used to initialize or modify the information of the project. The menu items "New" and "Modify" of the "Project" popup menu use the object of this class. Here are its member functions and variables. (See projectinit.h and projectinit.cpp)

```cpp
CProjectInit
-- CProjectInit(CWnd * pParent = NULL)
-- Create(LPCTSTR lpszClassName, LPCTSTR lpw
-- DoDataExchange(CDataExchange * pDX)
-- DoModal()
-- OnDestroy()
-- OnInitDialog()
-- OnInitSaved()
-- Serialize(CArchive & archive)
-- m_Customer
-- m_Date
-- m_IsModify
-- m_IsNew
-- m_Name
-- m_Note
-- m_Operator
-- m_ProjectFile
```

The member variables, m_Customer, m_Date, m_Name, m_note and m_Operator are used to store the information of the project. The user inputs all these information in
the dialog interface. Since the menu items of "New" and "Modify" in the "Project" popup menu use the object created by the same class, m_IsNew and m_IsModify are used here for this purpose. The variable m_ProjectFile is used to store the file name that saves all these information. The member functions DoModal(), OnCancel(), OnInitDialog() and OnInitsave are called by the dialog box, Serialize() is used to read/store data from/to file.

1.4 SetupPSheet

This class is derived from CPropertySheet. Objects of class CPropertySheet represent property sheets. The property sheet here consists of a CPropertySheet object and three CPropertyPage objects, m_setuplocation, m_setupwelding and m_setuprobot. Here is the list of the member functions and variables. (See setupsheets.h and setsheet.cpp)

```cpp
CSetupPSheet
    - CSetupPSheet(LPCTSTR pCaption, CWnd * pParentWnd = NULL, UINT iSelectPage = 0)
    - CSetupPSheet(UINT nIDCaption, CWnd * pParentWnd = NULL, UINT iSelectPage = 0)
    - "CSetupPSheet()
    - m_ProjectFile
    - m_setuplocation
    - m_setuprobot
    - m_setupwelding
```

The member variable m_ProjectFile refer to the project name. Other member variables, m_setuplocation, m_setupwelding and m_setuprobot, refer to the three CPropertyPage objects. These objects are used to setup the parameters of the robot, robot frame and geometry. Below are more details about these three objects.

a. CSetupRLocation

The object of this class is used to set up the robot frame parameters. Here are the member variables and functions.
The member variables here begin with "m_T_" and "m_U_" refer to the coordinate date of the $UTOOL and $UFRAME.

b. CSetupRobot

The object of this class is used to set up the robot parameters. Here are the member variables and functions. (See setuprobot.h and setuprobot.cpp)
The variable m_name refer to the name of the robot, m_speed is the speed of the robot tool moving. Other variables are the GP path parameters.

c. CSetupWelding

The object of this class is used to set up the geometry parameters. Here are the member variables and functions.

```cpp
CSetupRobot
  CSetupRobot()
  ~CSetupRobot()
  DoDataExchange(CDataExchange * pDX)
  OnInitDialog()
  OnOK()
  m_acceleration
  m_mtype
  m_ProjectFile
  m_name
  m_speed
  m_taction
  m_tdelay
  m_ttype
```

The variables m_high, m_long and m_wide are the height, length and width of the robot tool. The variable m_thickness refers to the manufacturing thickness.

2. Main programming of Visual C++

1. cn.h and cn.cpp

   a. cn.h

   // cn.h : main header file for the CN application
   #if defined(AFX_CN_H_78E95A09_54F3_11D1_982A_444553540000_INCLUDED)
   #define AFX_CN_H_78E95A09_54F3_11D1_982A_444553540000_INCLUDED
   #if _MSC_VER >= 1000
   #pragma once
   #endif // _MSC_VER >= 1000
   #ifndef __AFXWIN_H__
   #error include 'stdafx.h' before including this file for PCH
   #endif

   #include "resource.h" // main symbols
   // CCnApp:
// See cn.cpp for the implementation of this class

class CCnApp : public CWinApp
{
    public:

    CCnApp();

    // Overrides
    // ClassWizard generated virtual function overrides
    /// {AFX_VIRTUAL(CCnApp)
    public:
    virtual BOOL InitInstance();
    ///}AFX_VIRTUAL
    // Implementation
    COleTemplateServer m_server;
    // Server object for document creation
    /// {AFX_MSG(CCnApp)
   afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code!
    ///}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

Microsoft Developer Studio will insert additional declarations immediately before the previous line.

b.cn.cpp

// cn.cpp : Defines the class behaviors for the application.
#include "stdafx.h"
#include "cn.h"
#include "MainFrm.h"
#include "cnDoc.h"
#include "cnView.h"
#ifdef __DEBUG
#define new DEBUG_NEW
#endif

static char THIS_FILE[] = __FILE__;
#endif

BEGIN_MESSAGE_MAP(CCnApp, CWinApp)
    /// {AFX_MSG_MAP(CCnApp)
    ON_COMMAND(IDC_APP_ABOUT, OnAppAbout)
    ///}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(IDC_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(IDC_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

CCnApp::CCnApp()
{


```cpp
// The one and only CCnApp object
CCnApp theApp;
// This identifier was generated to be statistically unique for your app.
// You may change it if you prefer to choose a specific identifier.
// (78E95A04-54F3-11D1-952A-444553540000)
static const CLSID clsid =
{ 0x78e95a04, 0x54f3, 0x11d1, { 0x98, 0x2a, 0x44, 0x45, 0x53, 0x54, 0x0, 0x0 } };

// CCnApp initialization
BOOL CCnApp::InitInstance()
{
    // Initialize OLE libraries
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.
    #ifdef _AFXDLL
        Enable3Controls(); // Call this when using MFC in a shared DLL
    #else
        Enable3ControlsStatic(); // Call this when linking to MFC statically
    #endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(5); // Load standard INI file options (including MRU)
    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CCnDoc),
        RUNTIME_CLASS(CMainFrame), // main SDI frame window
        RUNTIME_CLASS(CCnView));
    AddDocTemplate(pDocTemplate);
    // Connect the COleTemplateServer to the document template.
    // The COleTemplateServer creates new documents on behalf
    // of requesting OLE containers by using information
    // specified in the document template.
    m_server.ConnectTemplate(clsid, pDocTemplate, TRUE);
    // Note: SDI applications register server objects only if /Embedding
    // or /Automation is present on the command line.
    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Check to see if launched as OLE server
```
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{
    // Register all OLE server (factories) as running. This enables the
    // OLE libraries to create objects from other applications.
    COleTemplateServer::RegisterAll();

    // Application was run with /Embedding or /Automation. Don't show the
    // main window in this case.
    return TRUE;
}

// When a server application is launched stand-alone, it is a good idea
// to update the system registry in case it has been damaged.
if (server.VerifyRegistry())
     m_server.UpdateRegistry(OAT_DISPATCH_OBJECT);
COleObjectFactory::UpdateRegistryAll();

// Dispatch commands specified on the command line
if (cProcessShellCommand(cmdInfo))
     return FALSE;

// The one and only window has been initialized, so show and update it.
if (!m_pMainWnd->ShowWindow(SW_SHOW))
    return TRUE;

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

    // Dialog Data
    IActionResult(IDC_ABOUTBOX);

    // ClassWizard generated virtual function overrides
    DECLARE_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    DECLARE_VIRTUAL
    // Implementation
    protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    // No message handlers
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    pDX->BEGIN_MESSAGE_MAP(CAboutDlg)
    pDX->END_MESSAGE_MAP()
2. projectinit.h and projectinit.cpp
   a. projectinit.h

   ifndef _MSC_VER >= 1000
   #pragma once
   ifndef _MSC_VER >= 1000
   // ProjectInit.h : header file
   //////////////////////////////////////////////////////////////////////
   // CProjectInit dialog
   //extern CString myexternal_ProjectName;//status
   class CProjectInit : public CDialog
   {
   // Construction
   public:
      CProjectInit(CWnd* pParent = NULL); // standard constructor

      CString m_ProjectFile;
      boolean m_IsModify;
      boolean m_IsNew;
   // Dialog Data
   //{{AFX_DATA(CProjectInit)
   enum { IDD = IDD_PROJECTINIT };
   CString m_Name;
   COleDateTime m_Date;
   CString m_Operator;
   CString m_Customer;
   CString m_Note;
   //}}AFX_DATA

   void DoDataExchange(pDX)
   { CDialog::DoDataExchange(pDX);
// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CProjectInit)
public:
    virtual BOOL Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName,
DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID,
CCreateContext* pContext = NULL);
    virtual int DoModal();
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CProjectInit)
    virtual void OnCancel();
   afx_msg void OnInitSave();
    virtual BOOL OnInitDialog();
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
    virtual void Serialize(CArchive& archive);

protected:
    DECLARE_SERIAL(CProjectInit);
};
//}}AFX_INSERT_LOCATION}
#endif //defined(AFX_PROJECTINIT_H__255DEF2B_53C9_11D1_982A_444553540000__INCLUDED_)

b. projectinit.cpp
// Projectinit.cpp : implementation file

#include "stdafx.h"
#include "cn.h"
#include "ProjectInit.h"
#include <direct.h> // for _mkdir()
#include <errno.h> // for error no: EACCES

#if defined(_DEBUG)
#define new DEBUG_NEW
#define THIS_FILE __FILE__
#endif

// CProjectInit dialog
CProjectInit::CProjectInit(CWnd* pParent /*=NULL*/)
    : CDialog(CProjectInit::IDD, pParent)
{
    m_isNew=FALSE;
    m_isModify=FALSE;
   //}}AFX_DATA_INIT(CProjectInit)
m_Name = _T("");
m_Date = CDate::GetDate();
T_Name = _T("");
m_Customer = _T("");
m_Note = _T("");
//}}AFX_DATA_INIT

}}AFX_DATA_INIT

void CProjectInit::DoDataExchange(CDataExchange* pDX)
{
   CDIalog::DoDataExchange(pDX);
   ///////////////////////////////////////////////////////////////////////////
   //AFX_DATA_MAP(CProjectInit)
   DDX_Text(pDX, IDC_ENAME, m_Name);
   DDV_MaxChars(pDX, m_Name, 20);
   DDX_Text(pDX, IDC_EDATE, m_Date);
   DDX_Text(pDX, IDC_EOPERATOR, m_Operator);
   DDV_MaxChars(pDX, m_Operator, 20);
   DDX_Text(pDX, IDC_ECUSTOMER, m_Customer);
   DDV_MaxChars(pDX, m_Customer, 40);
   DDX_Text(pDX, IDC_ENOTE, m_Note);
   DDV_MaxChars(pDX, m_Note, 200);
   ///////////////////////////////////////////////////////////////////////////
   }AFX_DATA_MAP

BEGIN_MESSAGE_MAP(CProjectInit, CDIalog)
   ///////////////////////////////////////////////////////////////////////////
   //AFX_MSG_MAP(CProjectInit)
   ON_BN_CLICKED(IDC_INITSAVE, OnInitsave)
   ///////////////////////////////////////////////////////////////////////////
   }AFX_MSG_MAP

END_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////////////

//Serialze
void CProjectInit::Serialize(CArchive& archive)
{
   CProjectInit::Serialize(archive);
   if (archive.IsStoring())
      archive << m_Name << m_Date << m_Operator << m_Customer << m_Note;
   else
      archive >> m_Name >> m_Date >> m_Operator >> m_Customer >> m_Note;
}

IMPLEMENT_SERIAL(CProjectInit, CDIalog, 1)

///////////////////////////////////////////////////////////////////////////

BOOL CProjectInit::OnInitDialog()
{
   CDIalog::OnInitDialog();
   //Called by "New" Menu Item
   if (m_IsNew)
   {
      //Create the new project
      GetDlgItem(IDC_ENAME)->EnableWindow(TRUE);
      GetDlgItem(IDC_EDATE)->EnableWindow(TRUE);
   }
   //Called by "Modify" Menu Item
   if (m_IsModify)
   {
      //Modify the existed project
      GetDlgItem(IDC_ENAME)->EnableWindow(FALSE);
      GetDlgItem(IDC_EDATE)->EnableWindow(FALSE);
      //load the existed message and display them on the dialog

67
CString filename=m_ProjectFile;
if (filename.isEmpty())
{
    MessageBox(_T("The Name of the project does not existed !\n\n"),
               _T("Modifying Failure"), MB_ICONSTOP);
    EndDialog(-1);
    return -1;
}
else
{
    CFile modifyfile;
    CFileException e;
    if (!modifyfile.open(filename, CFile::modeRead, &e))
    {
        MessageBox(_T("File: "+m_Name+ " cannot be opened!!\nPlease use another name or load it from the disk!"),
                                   _T("Open project file Failure"), MB_ICONSTOP | MB_OK);
        EndDialog(-1);
        return -1;
    }
    else
    {
        CArchive ar(&modifyfile, CArchive::load);
        ar>>m_Name>>m_Date>>m_Operator>>m_Customer>>m_Note;
        ar.Close();
        modifyfile.Close();
        UpdateData(FALSE);
    }
}
return TRUE;

/////////////////////////////////////////////////////////////////////////////
// CProjectInit message handlers
void CProjectInit::OnCancel()
{
    CDialog::OnCancel();
}
void CProjectInit::OnInitSave()
{
    CFile file;
    CFileException e;
    UpdateData(TRUE);
    //Check whether the name is empty
    if (m_Name.isEmpty())
    {
        MessageBox(_T("The Name of the project should not be empty !"+m_Name),
                                   _T("Saving Failure"), MB_ICONSTOP );
        return;
    }
    //check whether the file is existed.
    CFileFind finder;
    CString pFile=":\projects\"+m_Name+".prj";
if (lm_IsModify)
{
    BOOL iffInd=finder.FindFile(pFile);
    if (iffInd!=0)
    {
        MessageBox(_T("File: "+m_Name+ " is already existed!!\nPlease use another name or load it from the disk!");
        _T("Saving Failure"), MB_ICONSTOP);
        finder.Close();
        return;
    }
}

//Create the new project file or recreate the modified file

if (!file.Open(pFile,CFile::modeCreate|CFile::modeWrite,&e))
{
    MessageBox("File could not be Modified\n\n");
    return;
}
else
{
    //Write information to the project file *.prj
    CArchive ar(&file, CArchive::store);
    ar<<m_Name<<m_Date<<m_Operator<<m_Customer<<m_Note;
    ar.Close();
}
file.Close();

//Create the project sub-directory _mkdir if NEW!
if (lm_IsModify)
{
    CString pPath=_T(\"\.\projects\\"+m_Name);
    if (_mkdir(pPath)!=0)
    {
        if (_mkdir(pPath)==EACCESS)
        {
            MessageBox(_T("The project path: "+pPath+ " already have been created!!\nPlease check it in the Explore\n"),
            _T("Path Creating Failure"), MB_ICONSTOP);
            return;
        }
        if (_mkdir(pPath)==-1)
        {
            MessageBox(_T("The project path: "+pPath+ " can not be created!!\nPlease check it with University of Windsor!\n"),
            _T("Project Directory Creating Failure"),
            MB_ICONSTOP);
            return;
        }
    }
    else
    {
        //Copy the standard *.ini file to the subdirectory
        CString IniOrgFile=_T(\"\.\projects\project.ini\");
    }
}

69
BOOL iffind = finder.FindFile(IniOrgFile);
if (iffind != 0)
{
    CString IniDesFile = "\projects\" + m_Name + "\" + m_Name + ".ini";
    BOOL copyOK = CopyFile(IniOrgFile, IniDesFile, FALSE); // Overwrite the existed file
    if (copyOK != 0)
    {
        // Write information to the Initial file
        WritePrivateProfileString("Project", "Name",
        m_Name, IniOrgFile);
    }
    else
    {
        MessageBox(_T("The project initial file: \n\n" + IniOrgFile + " \n\n" " can not be copied!!\n\n" "Please check it with University of Windsor!"),
        _T("Project Initial File copying Failure"),
        MB_ICONSTOP);
        return;
    }
} else
{
    MessageBox(_T("File: " + IniOrgFile + " is not existed!!")
        "\n\nPlease Contact with University of Windsor!")
    _T("Copy Failure"), MB_ICONSTOP);
    finder.Close();
    return;
}

m_ProjectFile = pFile;
CDialog::OnOK();

}
### Define

```c++
#define AFX_SETUPPSHEET_H__B6E9FF45_59BB_11D1_982A_444553540000__INCLUDED
#include "SetupRobot.h" // Added by ClassView
#include "SetupWelding.h" // Added by ClassView
#include "SetupRLocation.h" // Added by ClassView
#include "SetupGLocation.h" // Added by ClassView
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// SetupPSheet : header file

// CSetupPSheet
class CSetupPSheet : public CPropertySheet
{
    DECLARE_DYNAMIC(CSetupPSheet)
    // Construction
    public:
        CSetupPSheet(UINT nIDCaption, CWnd* pParentWnd = NULL, UINT nSelectPage = 0);
        CSetupPSheet(LPCTSTR pszCaption, CWnd* pParentWnd = NULL, UINT nSelectPage = 0);
    // Attributes
    public:
        // Operations
    public:
        // Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSetupPSheet)
        //}}AFX_VIRTUAL
    // Implementation
    public:
        CString m_ProjectFile;
        CSetupRLocation m_setuprlocation;
        CSetupWelding m_setupwelding;
        CSetupRobot m_setuprobot;
    virtual ~CSetupPSheet();
    // Generated message map functions
    protected:
        //{{AFX_MSG(CSetupPSheet)
        // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

---

### Include

```c++
// SetupPSheet : header file

// CSetupPSheet
class CSetupPSheet : public CPropertySheet
{
    DECLARE_DYNAMIC(CSetupPSheet)
    // Construction
    public:
        CSetupPSheet(UINT nIDCaption, CWnd* pParentWnd = NULL, UINT nSelectPage = 0);
        CSetupPSheet(LPCTSTR pszCaption, CWnd* pParentWnd = NULL, UINT nSelectPage = 0);
    // Attributes
    public:
        // Operations
    public:
        // Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSetupPSheet)
        //}}AFX_VIRTUAL
    // Implementation
    public:
        CString m_ProjectFile;
        CSetupRLocation m_setuprlocation;
        CSetupWelding m_setupwelding;
        CSetupRobot m_setuprobot;
    virtual ~CSetupPSheet();
    // Generated message map functions
    protected:
        //{{AFX_MSG(CSetupPSheet)
        // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```
b. setsheet.cpp
   // SetupPSheet.cpp : implementation file
   #include "stdafx.h"
   #include "cn.h"
   #include "SetupPSheet.h"
   #ifdef _DEBUG
   #define new DEBUG_NEW
   #undef THIS_FILE
   static char THIS_FILE[] = __FILE__;
   #endif
   ///////////////////////////////////////////////////////////////////////////
   // CSetupPSheet
   IMPLEMENT_DYNAMIC(CSetupPSheet, CPropertySheet)
   CSetupPSheet::CSetupPSheet(UINT nIDCaption, CWnd* pParentWnd, UINT iSelectPage)
   :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
   {
   }
   CSetupPSheet::CSetupPSheet(LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelectPage)
   :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
   {
     AddPage(&m_setuprobot);
     AddPage(&m_setupweiding);
     AddPage(&m_setuplocation);
   }
   CSetupPSheet::~CSetupPSheet()
   {
   }
   BEGIN_MESSAGE_MAP(CSetupPSheet, CPropertySheet)
   END_MESSAGE_MAP()

4. setuprobot.h and setuprobot.cpp
a. setuprobot.h
   #if
   !defined(AFX_SETPROBOT_H__B6E9FF43_59BB_11D1_982A_444553540000__INCLUDED_)
   #define AFX_SETPROBOT_H__B6E9FF43_59BB_11D1_982A_444553540000__INCLUDED_ D
   #if _MSC_VER >= 1000
   #pragma once
   #endif // _MSC_VER >= 1000
   // SetupRobot.h : header file
   ///////////////////////////////////////////////////////////////////////////
   // CSetupRobot dialog
class CSetupRobot : public CPropertyPage
{
    DECLARE_DYNCREATE(CSetupRobot)

    // Construction
    public:
    CString m_ProjectFile;
    CSetupRobot();
    ~CSetupRobot();

    // Dialog Data
    //{{AFX_DATA(CSetupRobot)
        enum { IDD = IDD_PAGE_ROBOT };        int m_acceleration;
        CString m_mtype;
        CString m_taction;
        int m_tdelay;
        CString m_ttype;
        CString m_rname;
        int m_speed;
    //}}AFX_DATA

    // Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CSetupRobot)
        public:
        virtual void OnOK();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

    // Implementation
    protected:
        // Generated message map functions
    //{{AFX_MSG(CSetupRobot)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif //

b. setuprobot.cpp
// SetupRobot.cpp : implementation file
#include "stdafx.h"
#include "cn.h"
#include "SetupRobot.h"
#include "PubFunction.h"

73
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

static char THIS_FILE[] = __FILE__;

intelligence

IMPLEMENT_DYNAMIC(CSetupRobot, CPropertyPage)
CSetupRobot::CSetupRobot() : CPropertyPage(CSetupRobot::IDD)
{

    AX_DATA_INIT(CSetupRobot)
      m_acceleration = 0;
      m_rtype = ID("7");
      m_taction = ID("5");
      m_tdelay = 0;
      m_ttype = ID("8");
      m_name = ID("5");
      m_speed = 0;
    AX_DATA_INIT
}
CSetupRobot::~CSetupRobot()
{}

void CSetupRobot::DoDataExchange(CDataExchange* pDX)
{

    CPropertyPage::DoDataExchange(pDX);
    AX_DATA_MAP(CSetupRobot)
        DDX_Text(pDX, IDC_ACCELERATION, m_acceleration);
        DDX_Text(pDX, IDC_MTYPE, m_rtype);
        DDX_Text(pDX, IDC_TACTION, m_taction);
        DDX_Text(pDX, IDC_TDELAY, m_tdelay);
        DDX_Text(pDX, IDC_TTYPE, m_ttype);
        DDX_Text(pDX, IDC_RNAME, m_name);
        DDX_Text(pDX, IDC_SPEED, m_speed);
    AX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CSetupRobot, CPropertyPage)
END_MESSAGE_MAP()

// CSetupRobot message handlers

void CSetupRobot::OnOK()
{

    UpdateData();
    //Modify the project initial file
    CString filepath=m_ProjectFile.GetLength-4;
    CStri message="The initial file of the project:\n"+initialfile+"\n"+"Does not existed!!\n
    BOOL iffind=findFile(initialfile, message);
    if (iffind==0)
    {
        return;
    }

    74
else
{
    CString savestr;
    //save information to the initial file
    savestr.Format("%d", m_acceleration);
    WritePrivateProfileString("Robot Setup", "Acceleration", savestr, initialfile);
    savestr.Format("%s", m_mtype);
    WritePrivateProfileString("Robot Setup", "Motion Type", savestr, initialfile);
    savestr.Format("%s", m_taction);
    WritePrivateProfileString("Robot Setup", "Tool Action", savestr, initialfile);
    savestr.Format("%s", m_type);
    WritePrivateProfileString("Robot Setup", "Termination Type", savestr, initialfile);
    savestr.Format("%s", m_mame);
    WritePrivateProfileString("Robot Setup", "Robot Name", savestr, initialfile);
    savestr.Format("%d", m_tdelay);
    WritePrivateProfileString("Robot Setup", "Time Delay", savestr, initialfile);
    savestr.Format("%d", m_speed);
    WritePrivateProfileString("Robot Setup", "Speed", savestr, initialfile);
}

CPropertyPage::OnOK();
}

BOOL CSetupRobot::OnInitDialog()
{
    CPropertyPage::OnInitDialog();
    // Read initial information from *.ini file
    CString filepath=m_ProjectFile.Left(m_ProjectFile.GetLength()-4);
    CString initialfile=findInitialFile(filepath);
    CString message="The Initial file of the project:"+initialfile+" does not exist!!";
    BOOL iffind=findFile(initialfile,message);
    if (iffind==0)
    {
        EndDialog(-1);
        return FALSE;
    }
    else
    {
        CHAR inBuf[80];
        GetPrivateProfileString("Robot Setup", "Acceleration", "100", inBuf, 80, initialfile);
        sscanf(inBuf, "%d", &m_acceleration);
        GetPrivateProfileString("Robot Setup", "Time Delay", "0", inBuf, 80, initialfile);
        sscanf(inBuf, "%d", &m_tdelay);
        GetPrivateProfileString("Robot Setup", "Speed", "100", inBuf, 80, initialfile);
        sscanf(inBuf, "%d", &m_speed);
        GetPrivateProfileString("Robot Setup", "Motion Type", "Linear", inBuf, 80, initialfile);
        m_mtype=inBuf;
        //MessageBox("m_mtype="+m_mtype);
        GetPrivateProfileString("Robot Setup", "Tool Action", "No Tool Action", inBuf, 80, initialfile);
        //sscanf(inBuf, "%s", &m_taction);
        m_taction=inBuf;
        GetPrivateProfileString("Robot Setup", "Termination Type", "0", inBuf, 80, initialfile);
    }
}

75
// sscanf(inBuf, "%s", &m_type);
    m_type=strdup(inBuf);
GetPrivateProfileString("Robot Setup", "Robot Name", "IRB2000", inBuf, 80, initialfile);

    // sscanf(inBuf, "%s", &m_name);
    m_name=strdup(inBuf);

} UpdateData(FALSE);
// MessageBox("M_Type="+m_type);
return TRUE; // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
Appendix B: Matlab program for the path generation

1. Original and Offset surface calculating program

With the control points, this program is used to calculate the original and offset surface points. Here is the program.

```matlab
% ***** Bezier original and Offset surface calculating *****

% Get control points of the Bezier surface from disk
load gx.dat;  % x values
load gy.dat;  % y values
load gz.dat;  % z values

% Arrays to store values of the surface point
px=zeros(21);
py=zeros(21);
pz=zeros(21);

% M is the n*m constant matrix
M=[-1 3 -3 1;3 -6 3 0;-3 3 0 0;1 0 0 0];

% Surface points calculating
k=0;
for i=0:0.05:1
    U=[i^3 i^2 i 1];
    for j=0:0.05:1
        V=[j^3 j^2 j 1];
        k=k+1;
        px(k)=U*M*gx*M*V;
        py(k)=U*M*gy*M*V;
        pz(k)=U*M.gz*M*V;
    end
end

% surface point plotting
figure(1);
for i=1:1:20
    k3=((i-1)*21+1):i*21;
    plot3(px(k3),py(k3),pz(k3),"*");
    hold on;
xlabel('X');
ylabel('Y');
end
grid on;
```
% Save original px py pz -ascii;
px1=zeros(21);
py1=zeros(21);
pz1=zeros(21);
px1=px;
py1=py;
pz1=pz;
save original px1 py1 pz1;

% Calculating Normal vector of the surface
k=0;
for i=0:0.05:1
    U=[i^3 i^2 i 1];
    UU=[3*i^2 2*i 1 0];
    for j=0:0.05:1
        v=[j^3 j^2 j 1];
        vv=[3*j^2 2*j 1 0];
        dpxu=UU*M*gx*M*v;
        dpxv=UU*M*gx*M*vv;
        dpyu=UU*M*gy*M*v;
        dpyv=UU*M*gy*M*vv;
        dpzu=UU*M*gz*M*v;
        dpzv=UU*M*gz*M*vv;
    k=k+1;
    Nx=dpyu*dpzv-dpzu*dpyv;
    Ny=dpzu*dpvx-dpxu*dpzv;
    Nz=dpvx*dpyv-dpyu*dpvx;
    a=sqrt(Nx*Nx+Ny*Ny+Nz*Nz);
    if a==0
        nx(k)=Nx/a;
        ny(k)=Ny/a;
        nz(k)=Nz/a;
    else
        nx(k)=Nx;
        ny(k)=Ny;
        nz(k)=Nz;
    end
end

% Values of the points on the offset surface
clx=zeros(21);
cly=zeros(21);
clz=zeros(21);

Calculating the offset surface points
k=0;
for u=0:0.05:1
    for v=0:0.05:1
        k=k+1;
        clx(k)=px(k)+nmx(k)*20;
        cly(k)=py(k)+nny(k)*20;
        clz(k)=pz(k)+nnz(k)*20;
        d(k)=sqrt((clx(k)-px(k))^2+(cly(k)-py(k))^2+(clz(k)-pz(k))^2);
    end
end

Save offset values
px2=zeros(21);
py2=zeros(21);
pz2=zeros(21);
px2=clx;
py2=cly;
pz2=clz;
save offset px2 py2 pz2

Original&Offset surface plotting
figure(2);
for i=1:1:21
    k3=((i-1)*21+1):1:i*21;
    hndl=plot3(px1(k3),py1(k3),pz1(k3));
    hold on;
end
for i=1:1:21
    k3=((i-1)*21+1):1:i*21;
    hndl=plot3(px2(k3),py2(k3),pz2(k3));
    set(hndl,'Color','red');
end
grid on;
xlabel('X');
ylabel('Y');
zielab('Z');
title('Original and Offset Surface');
2. Robot path generation program

This program is used to generation of the robot path. It includes collision
detection and correction, Teachpoint file and track file generation, and GP path files
generation. Here is the detail of the program.

```
%******************************************************************************%
%** ALGORITHM FOR COLLISION DETECTION AND CORRECTION **
%******************************************************************************%

%Input file name of the teachpoint and track file
fname = input('Enter Gps&TPs File Name between quotes: ');
gpname='Original.pth'; % GP path file name of the original surface
tpname=strcat(fname,char(35),'.kl'); % Teach point file name
trkname=strcat(fname,'.kl'); % Track file name
quot='"';
quotN="LDN";
of_path='Off_path.pth'; % GP path file name of the offset surface

fprintf(tpname, '-- WORKSPACE teachpoint file
');
%load $uframe & $utool value in workspace
%$uframe value in workspace xframe = 950; yframe = 0; zframe = 1585;
%$utool value in Workspace xtool = 210; ytool = 0; ztool = -198;
load frame;

%******************************************************************************%
%* Tool rectangular definition *
%P(i) points co-ordinates at Home position
Xp0(1) = 10; Xp0(2) = 10; Xp0(3) = 10; Xp0(6) = 10;
Xp0(3) = -340; Xp0(4) = -340; Xp0(7) = -340; Xp0(8) = -340;

Yp0(1) = -35; Yp0(4) = -35; Yp0(5) = -35; Yp0(8) = -35;
Yp0(2) = 35; Yp0(3) = 35; Yp0(6) = 35; Yp0(7) = 35;

Zp0(1) = -15; Zp0(2) = -15; Zp0(3) = -15; Zp0(4) = -15;
Zp0(5) = 100; Zp0(6) = 100; Zp0(7) = 100; Zp0(8) = 100;
%6 "Tool rectangle planes" Definition
plane=[1 2 3
      5 6 7 %Zlength l3
      2 3 7
      4 5 8 %Y length l2
      1 2 6
      3 4 8]; % X length l1

%"Tool rectangle geometry"
```
% load geometry parameters from disk file
% l1=350; % length l2=70; % width l3=135; % height
load geometry;

%******************************************************************************
%* Surfaces Points loading
%* % load original and boundary surface points data
%* % arrays of px1,py1,pz1 store these points values
load original;
sizel=size(px1);
sizel1=size(1);
% number of points on every path
sizel2=size(2);
% number of paths
sizel11=size(1)*size(2);
% number of total points

% load offset surface points data
load offset;
sizel2=size(px2);
sizel11=size(1);
sizel22=size(2);
sizel111=size(1)*size(2);

%*** Arrays to store points on the generated path
px3=zeros(sizel11+2,sizel22);
py3=zeros(sizel11+2,sizel22);
px3=zeros(sizel11+2,sizel22);
Rotate_P=zeros(sizel11+2,sizel22);
Rotate_W=zeros(sizel11+2,sizel22);

% coordinate changing into Workspace
px1=px1+xtool; py1=py1+ytool; pz1=pz1+ztool;
px2=px2+xtool; py2=py2+ytool; pz2=pz2+ztool;

% Write original/boundary surface data to GP file
joint='Linear';
for i=1:sizel11
  if i==1 joint='Joint';
  else joint='Linear';
  end
  numstr = num2str(i);
  GP = strcat('IRB2000','GP',numstr);
  fprintf(gpname,
'%5s,POS=%3.2f,%3.2f,%3.2f,GP,px1(i)+xframe,py1(i)+yframe,pz1(i)+zframe);
  fprintf(gpname, ',0,0,0,%5s,Term=100, Vel=100, Acc=100, Wait=0, Rout=.No signal.No
tool action\n',joint);
end
% write offset points to files
of_file='Off_data.pth';
for i=1:size_of
    numstr = num2str(i);
    GP = strcat('IRB2000','GP',numstr);
    fprintf(of_file,
        '%5s,POS=%3.2f,%3.2f,%3.2f,GP,px2(i)+xframe,py2(i)+yframe,pz2(i)+zframe);
    fprintf(of_file, ',0,0,0,%s,Term=100, Vel=100, Acc=100, Wait=0, Rout=, No signal, No
    tool action
    \n''joint);
end

%*****************************************************************************
%*****************************************************************************
%*****************************************************************************
% ALGORITHM For GOUGING DETECTION
%*****************************************************************************
%*****************************************************************************
% Initializing Robot World frame Angles
W=0; P=0; R=0;
Np=8;  % Number of P(i) Points on Tool intersect=0;  % Number of detected intersections

% Beginning of the Algorithm

width_path=py2(size_of1+1)-py2(1);
check_path=fix((l2/(2*width_path)));
PorW='P';

flag='T';
k2=1;
while k2<=size_of2  % path number on the offset surface
    if (P>=180)&&(W>=180)  % no solution
        break;
    else
        P=0;
        w=0;
P_Last=0;
W_Last=0;
    end

k1=1;
while k1<=size_of1  % points number on current path of the offset surface
    cP = cos(-P*(pi/180));
sP = sin(-P*(pi/180));

    for j=1:1:Np  % P points on the tool rectangular: Pi (Xp(i), Yp(i), Zp(i))
        Xp(j) = cP*Xp0(j) - Zp0(j)*sP + px2((k2-1)*size_of1+k1) + xframe;
        Yp(j) = Yp0(j) + py2((k2-1)*size_of1+k1) + yframe;
        Zp(j) = Xp0(j)*sP + Zp0(j)*cP + pz2((k2-1)*size_of1+k1) + zframe +30;
    end

82
Equation calculation of the 6 tool planes
for a=1:1:6

\[
A_p(a) = (Y_p(\text{plane}(a,2)) \times Z_p(\text{plane}(a,3)) - Y_p(\text{plane}(a,3)) \times Z_p(\text{plane}(a,2))) - \\
(Y_p(\text{plane}(a,1)) \times Z_p(\text{plane}(a,3)) - Y_p(\text{plane}(a,3)) \times Z_p(\text{plane}(a,1))) + \\
(Y_p(\text{plane}(a,1)) \times Z_p(\text{plane}(a,2)) - Y_p(\text{plane}(a,2)) \times Z_p(\text{plane}(a,1)))
\]

\[
B_p(a) = -(X_p(\text{plane}(a,2)) \times Z_p(\text{plane}(a,3)) - X_p(\text{plane}(a,3)) \times Z_p(\text{plane}(a,2))) + \\
(X_p(\text{plane}(a,1)) \times Z_p(\text{plane}(a,3)) - X_p(\text{plane}(a,3)) \times Z_p(\text{plane}(a,1))) - \\
(X_p(\text{plane}(a,1)) \times Z_p(\text{plane}(a,2)) - X_p(\text{plane}(a,2)) \times Z_p(\text{plane}(a,1)))
\]

\[
C_p(a) = (Y_p(\text{plane}(a,3)) \times X_p(\text{plane}(a,2)) - Y_p(\text{plane}(a,2)) \times X_p(\text{plane}(a,3))) - \\
(Y_p(\text{plane}(a,3)) \times X_p(\text{plane}(a,1)) - Y_p(\text{plane}(a,1)) \times X_p(\text{plane}(a,3))) + \\
(Y_p(\text{plane}(a,2)) \times X_p(\text{plane}(a,1)) - Y_p(\text{plane}(a,1)) \times X_p(\text{plane}(a,2)))
\]

\[
D_p(a) = -(X_p(\text{plane}(a,1)) \times (Y_p(\text{plane}(a,2)) \times Z_p(\text{plane}(a,3))) - \\
Y_p(\text{plane}(a,3)) \times Z_p(\text{plane}(a,2))) + \\
Y_p(\text{plane}(a,1)) \times (X_p(\text{plane}(a,2)) \times Z_p(\text{plane}(a,3)) - \\
X_p(\text{plane}(a,3)) \times Z_p(\text{plane}(a,2))) - \\
Z_p(\text{plane}(a,1)) \times (X_p(\text{plane}(a,2)) \times Y_p(\text{plane}(a,3)) - \\
X_p(\text{plane}(a,3)) \times Y_p(\text{plane}(a,2)))
\]

end

Beginning of the collision-free detection
for n=((k2-1)\times size_or1+1):(k2\times size_or1)
inter=0;
cond(1)=0; cond(2)=0; cond(3)=0;
for \( \tau = 1:1:6 \)

\[
ume = \text{abs}(A_p(\tau) \times (xfr1(n)+xframe) + B_p(\tau) \times (yfr1(n)+yframe) + \\
C_p(\tau) \times (zfr1(n)+zframe) + D_p(\tau))
\]

\[
div = \sqrt{A_p(\tau)^2 + B_p(\tau)^2 - C_p(\tau)^2}
\]

\[\text{dist}(\tau) = \text{nume}/\text{div}\]
end

\[A=0; B=0;\]
if dist(1)<13
    A=1;
end
if dist(2)<13
    B=1;
end
if \(\text{and}(A, B) = 1\)
    cond(3)=1;
end

\[A=0; B=0;\]
if dist(3)<12
A=1;
end
if dist(4)<12
  B=1;
end
if and(A,B)==1
  cond(2)=1;
end
A=0; B=0;
if dist(5)<11
  A=1;
end
if dist(6)<11
  B=1;
end
if and(A,B)==1
  cond(1)=1;
end
for r=1:1:3
  if cond(r)==1
    inter=inter+1;
  end
end
if inter==3
  if PorW='P'
    if ((P-P_Last)>=20)
      P=P-5;
      P_Last=P;
      W=W+5;
      PorW='W';
    else
      P=P+5;
    end
  else
    if ((W-W_Last)>=20)
      W=W-5;
      W_Last=W;
      P=P+5;
      PorW='P';
    else
      W=W+5;
    end
end
end

break;
end
end

if (P>=180)||(W>=180)
  break;
end

if inte==3
  P_Last=P;
  W_Last=W;
  if rem(k1, size_ofl)==1
    px3((k2-1)*(size_ofl+2)+k1 +0)=px2((k2-1)*size_ofl+k1);
    py3((k2-1)*(size_ofl+2)+k1 +0)=py2((k2-1)*size_ofl+k1);
    pz3((k2-1)*(size_ofl+2)+k1 +0)=pz2((k2-1)*size_ofl+k1)+40;
    Rotate_P((k2-1)*(size_ofl+2)+k1 +0)=0;
    Rotate_W((k2-1)*(size_ofl+2)+k1 +0)=0;
  End

  px3((k2-1)*(size_ofl+2)+k1 +1)=px2((k2-1)*size_ofl+k1);
  py3((k2-1)*(size_ofl+2)+k1 +1)=py2((k2-1)*size_ofl+k1);
  pz3((k2-1)*(size_ofl+2)+k1 +1)=pz2((k2-1)*size_ofl+k1);
  Rotate_P((k2-1)*(size_ofl+2)+k1 +1)=P;
  Rotate_W((k2-1)*(size_ofl+2)+k1 +1)=W;

  if rem(k1, size_ofl)==0
    px3((k2-1)*(size_ofl+2)+k1 +2)=px2((k2-1)*size_ofl+k1);
    py3((k2-1)*(size_ofl+2)+k1 +2)=py2((k2-1)*size_ofl+k1);
    pz3((k2-1)*(size_ofl+2)+k1 +2)=pz2((k2-1)*size_ofl+k1)+40;
    Rotate_P((k2-1)*(size_ofl+2)+k1 +2)=P;
    Rotate_W((k2-1)*(size_ofl+2)+k1 +2)=W;
  end

  k1=k1+1;
  end
end
k2=k2+1;
end

%***********Automatic generation of teachpoint file and GP file***********
if flag=='T' %have solution
  k2=1;
  while k2<=size_of2
    k1=1;
    .
while k1 <= size_of1+2

    num = num2str((k2-1)*(size_of1+2)+k1);
    TP = strcat('TP', num);
    GP = strcat('TRB2000', 'GP', num);

% ***** TP tile
    fprintf(tname, '%5s = POS(%3.2f, %3.2f, TP, px3((k2-1)*(size_of1+2)+k1),
            py3((k2-1)*(size_of1+2)+k1)),
    fprintf(tname, ', %3.2f, %3.2f, %3.2f, %3.2f, %s)n, pz3((k2-1)*(size_of1+2)+k1), W, Rotate_P((k2-1)*(size_of1+2)+k1), R, quot);

% ***** GP path
    fprintf(of_path, '%5s, POS=%3.2f,%3.2f,%3.2f, GP, px3((k2-1)*(size_of1 + 2) +
            k1) + xframe, py3((k2-1)*(size_of1+2)+k1) + yframe, pz3((k2-1)*(size_of1+2)+k1) + zframe);
    fprintf(of_path, ', %3.2f,%3.2f,0', W, Rotate_P((k2-1)*(size_of1+2)+k1));
    fprintf(of_path, ', %s, Term=100, Vel=100, Acc=100, Wait=0, Rout=, No signal,
            No tool action)n, joint);

    k1 = k1 + 1;
    end
    k2 = k2 + 1;
    end

% ******** Automatic generation of track file of the robot path********
    fprintf(trkname, 'PROGRAM %5sn', fname);
    fprintf(trkname, '---! LANGUAGE KAREL 2\n');
    fprintf(trkname, '---! MEMORY 36000\n');
    fprintf(trkname, '---! ROBOT IRB2000\n');
    fprintf(trkname, '---! TEACHPOINT DECLARATIONS\n');
    fprintf(trkname, '\n');

    fprintf(trkname, 'VAR\n');
    i = 1;
    while i <= (size_of1+2)*size_of2
        num = num2str(i);
        TP = strcat('TP', num);
        fprintf(trkname, ' %5s: POSITION\n', TP);
        i = i + 1;
    end
    fprintf(trkname, '\n');
    fprintf(trkname, '\n');

    fprintf(trkname, 'BEGIN\n');
    fprintf(trkname, ' SUBFRAME=POS(950, 0, 1585, 0, -0, 0, %s)n, quot);
    fprintf(trkname, ' SUTOOL=POS(210, 0, -198, -0, -0, -0, %s)n, quot);
```c
fprintf(trkname,
    $USEMAXACCEL=TRUE
); fprintf(trkname,
    %\%INCLUDE %5s\n', fname);
fprintf(trkname, \n);
i=1;
while i<=size_of2
    num = num2str((i-1)*(size_of1+2)+1);
    TP=strcat('TP', num);
    fprintf(trkname,
        MOVE TO %5s\n', TP);
    num = num2str((i-1)*(size_of1+2)+2);
    TP=strcat('TP', num);
    fprintf(trkname,
        MOVE TO %5s\n', TP);
    fprintf(trkname,
        --! ARCWELDON 200, 80\n);
    fprintf(trkname,
        $SPEED=250\n');

j=3;
while j<=size_of1+1
    num = num2str((i-1)*(size_of1+2)+j);
    TP=strcat('TP', num);
    fprintf(trkname,
        MOVE TO %5s\n', TP);
    j=j+1;
end
fprintf(trkname,
    --! ARCWELDOFF\n);
    num = num2str((i)*(size_of1+2));
    TP=strcat('TP', num);
    fprintf(trkname,
        MOVE TO %5s\n', TP);
    fprintf(trkname, \n);
    fprintf(trkname,
        $SPEED=400\n');

i=i+1;
end

fprintf(trkname, \n);
fprintf(trkname, \nMOVE TO SUTOOL\n);
fprintf(trkname, 'END %5s', fname);
end
```
VITA AUCTORIS

NAME: Wenwei Gong

PLACE OF BIRTH: Jiangyin, Jiangsu, China

EDUCATION:
- University of Windsor, Windsor, Ontario
  1997 - 1998 M.S.Sc
- SouthEast University, Nanjing, P.R.China
  M.A.Sc. 1993
- HarBin Institute of Technology, HarBin, P.R.China
  B.E. 1990