To the Graduate Council:

I am submitting herewith a thesis written by Xiaoquan Fu entitled "Interactive Task Planning for Telerobotics". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

William R. Hamel, Major Professor

We have read this thesis and
recommend its acceptance:

Mongi A. Abidi
(Electrical Engineering)

Reid L. Kress
(Mechanical Engineering)

Accepted for the Council:

Associate Vice Chancellor and
Dean of The Graduate School

INTERACTIVE TASK PLANNING FOR TELEROBOTICS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Xiaoquan Fu
December, 1999

Acknowledgments

Many thanks are necessary because I have been so fortunate to meet so many nice and cordial people during my two year graduate study in the Robotics lab.

First of all, I wish to thank my advisor, Dr. William Hamel, for his particular patience, kindness and advice to make it possible for me to finish this research and thesis. I would also like to thank the members of my graduate committee, Dr. Mongi Abidi and Dr. Reid Kress for their support during the course of this thesis work.

I would like to thank my colleagues in the RTSA team, including Dr. Steven Everett, Surya Singh, Scott Hale, Ge Zhang, Mohammad Khalid, Veronika Gospodareva, Matt Smith, Samuel Richardson, Yasunobu Isoda, Sewoong Kim, Dr. Philip Smith, and Susan Yoder. I would like to recognize Dr. Chris Widner, James Franks and David Lennon for their help during the past years. Most importantly, I would especially like to thank Dr. Steven Everett and Stacey Mitchell, who gave me invaluable help through not only their technical knowledge but also their precious friendship. This thesis would never have been finished so early without them.

My greatest thanks go to my mother and my dead father. My knowledgeable, kind-hearted and honest father is the person who affected me the greatest in the world. With his nonstop perseverance, he made me understand that one should keep his enterprising spirit for his entire life. From my firm and indomitable mother I learned how to face and overcome all obstacles with courage and persistence. Every bit of my success belongs to them.

Lastly, I need to recognize that this work was performed under contract DE-AR26-97FT34314 for the Department of Energy, Federal Energy Technology Center, Morgantown, West Virginia.

Abstract

Radiation and other hazards in nuclear environmental restoration and waste management work necessitate the use of remote operations to protect human workers. In order to increase the productivity and reduce the workload of operators, the execution of telerobot tasks needs to be automated as much as possible. This thesis develops a framework to allow the operator to interactively generate a sequence of computerized action commands that will be automatically accomplished by a telerobot. Microsoft® Visual C++® is used as the main development software to build the system including the design of graphical user interface.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The research and application of robots have been advanced greatly during the past years since the occurrence of the first robot. Their application have been extended from simple manufacturing to various areas like medication, transportation, space exploration, entertainment, and so on. Phillip John McKerrow gave a very brief and precise definition for Robot [8]:

A Robot is a machine which can be programmed to do a variety of tasks, in the same way that a computer is an electronic circuit which can be programmed to do a variety of tasks.

Radiation hazards in the nuclear industry led to the development of teleoperators for remote handling of radioactive material. This thesis describes the development of a methodology to generate the telerobotic system commands in a dismantlement task execution. The second chapter develops background information concerning the RTSA project and the previous work. The system configuration, including the major hardware and software, are described in chapter 3. The fourth chapter explains the principle of task decomposition and the algorithm developed for action generation. The graphical user interface structure used for the entire task planning process is discussed in the fifth chapter. Chapter 6 presents conclusions and suggestions for future work are given in Chapter 7.

Chapter 2

Background

2.1 Overview

Many environmental restoration and waste management (ER&WM) challenges involve radiation or other hazards which will necessitate the use of remote operations to protect human workers from dangerous exposures. However, the remote operation are inherently far less productive than direct human operation since task operations are slow and tedious. As a result, the remote work systems are far more costly than direct human workers. Therefore, improving the productivity of combined human operator/remote equipment systems becomes a main issue. One promising avenue is to supplement conventional remote work systems with robotic planning and control techniques borrowed from manufacturing and other domains [3]. A more efficient telerobotic work system will be yielded through this combination.

That task is actually very difficult because of some formidable technical challenges. Almost always the geometry of the task environment is highly unstructured and uncertain. In many cases, the precision and accuracy of the geometric knowledge are not adequate. All of these factors add difficulties to the mission. Due to the limitations of current technology, the full automation of ER&WM cannot be realized for the time being. However, there are still certain subtasks in which automatic planning and execution can be utilized. The

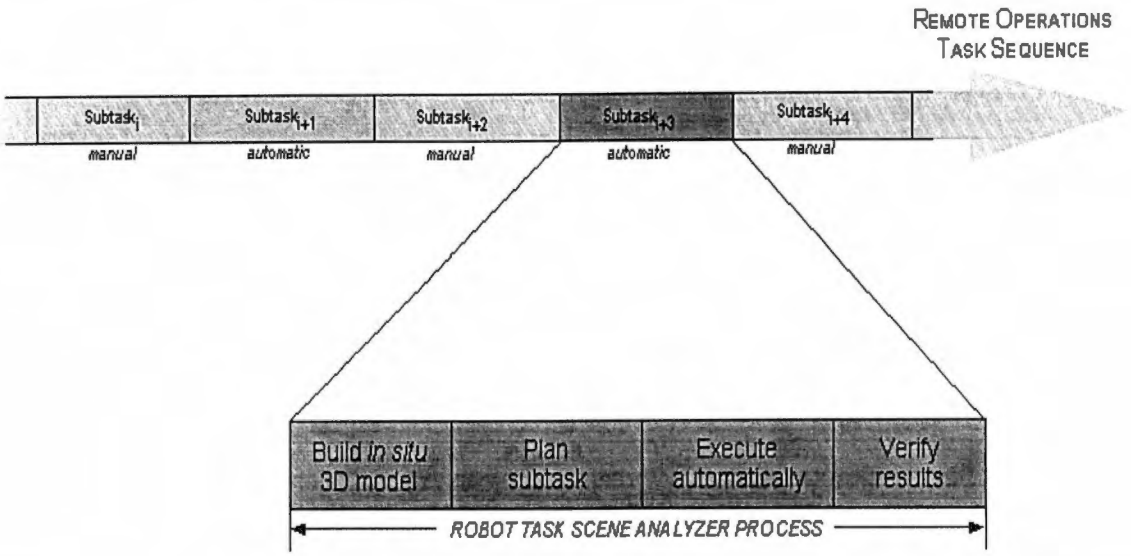

Figure 2.1: Telerobotics Operations Cycle

overall mission faced today is how to maximize automation to minimize the workload of the human operators.

In order to fulfill that task, lots of quantitative data have to be acquired. Most of all, the geometrical information of the working scenario is necessary. Some other data, like the characteristics of the robot, the types of the objects in the working environment and the parameters of the tools are also required to analyze and plan the automated task execution. Complete information is the best guarantee for success of automation.

Figure 2.1 depicts the overall process of the telerobotic operation. The "Build In situ Model" phase is when the geometrical information of the task environment is obtained. Based on those data and other required information, the task is planned and executed with the involvement of teleoperation thereafter.

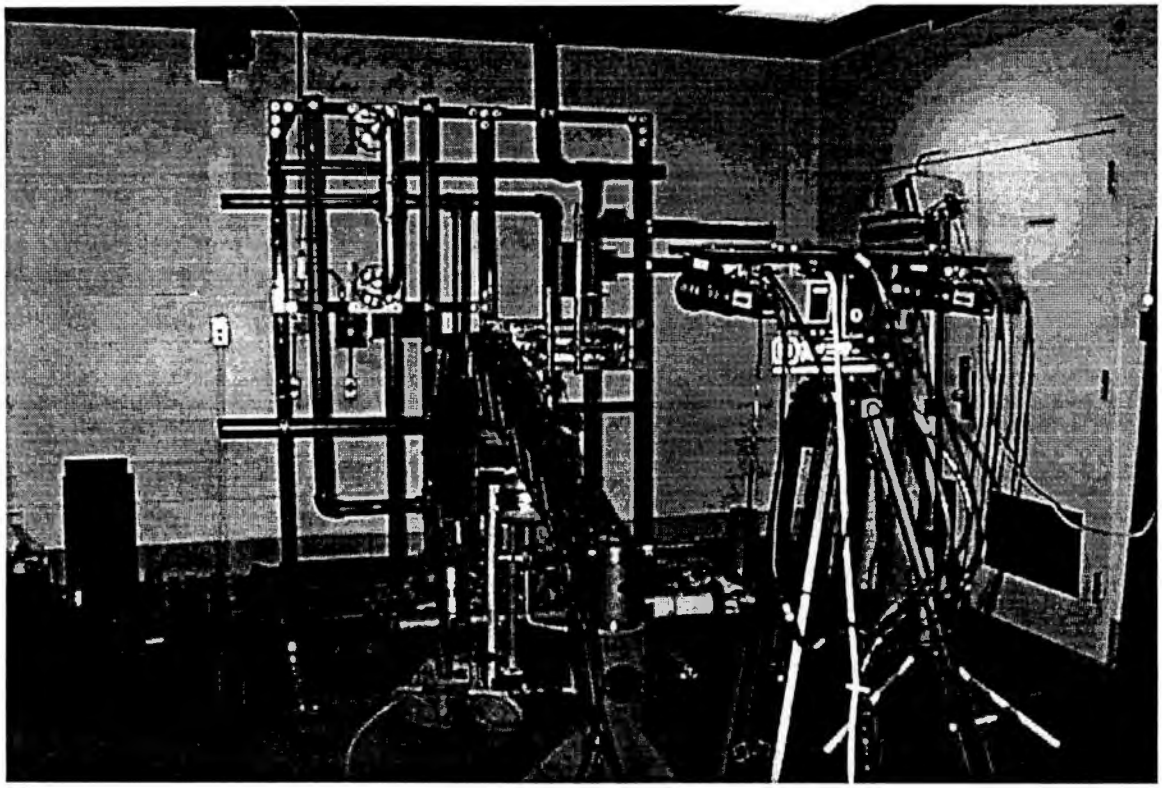

Figure 2.2: Laboratory Test Setup Arrangement

2.2 Previous work

For the "Build In situ Model" phase, the Robot Task Space Analyzer (RTSA) is a human interactive system which allows a remote operator to direct the construction of 3D geometrical descriptions of task objects (eg., pipes, tees, elbows, etc.). In other words, it is in essence a 3D model builder of the near-field of view of the mobile work system [2]. It uses stereo remote vision and laser range finders to gather physical data about objects. Figure 2.2 is the physical arrangement of RTSA laboratory testing system.

To imitate a realistic task space scene, the laboratory task mock up (shown in Figure 2.2) incorporates a large number of standard process piping and components as listed in Table 2.1. Using this type of prior information will help both the model building and the subsequent task planning and execution.

Table 2.1: Modeled Pipe Fittings

Component Type	Schedule	Size (Nominal)	Connection
400lb Flange	N/A	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	Bolted
600lb Flange	N/A	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	Bolted
900lb Flange	N/A	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	Bolted
1500lb Flange	N/A	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	Bolted
45° Elbow	10–160	2 1/2, 3, 4, 5, 6, 8 inch	Welded
90° Elbow	10–160	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	Welded
90° Elbow	10–160	2 and 3 inch	Screwed
Tee	10–160	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	Welded
Tee	10–160	2 inch	Screwed
Straight Pipe	10–160	3/4, 1 1/4, 1 1/2, 2, 2 1/2, 3, 4, 5, 6, 8 inch	N/A

RTSA uses three algorithms to construct the 3D geometrical model of the task scene. (see Figure 2.3) Among them the manual mode is the most robust and reliable [9]. With the stereo camera images, the operator's experience can be employed to specify the type and the size of objects. The location and the orientation information can be obtained from the laser range sensor and the computer algorithm. Then the appropriate models (from pre-constructed model library) can be placed at the right place in 3D space referenced to the mobile robot. The two auto scan algorithms, stereo auto scan and range image auto scan, can be run in the background to search for those indicated objects of interest.

2.3 Subsequent Task

Once the geometrical information of the task scene have been obtained, the plan for the motion of the robot becomes the key to the accomplishment of the automation of the task. As will be discussed later, the robot task will be decomposed into actions and then those actions will be implemented in the programming language.

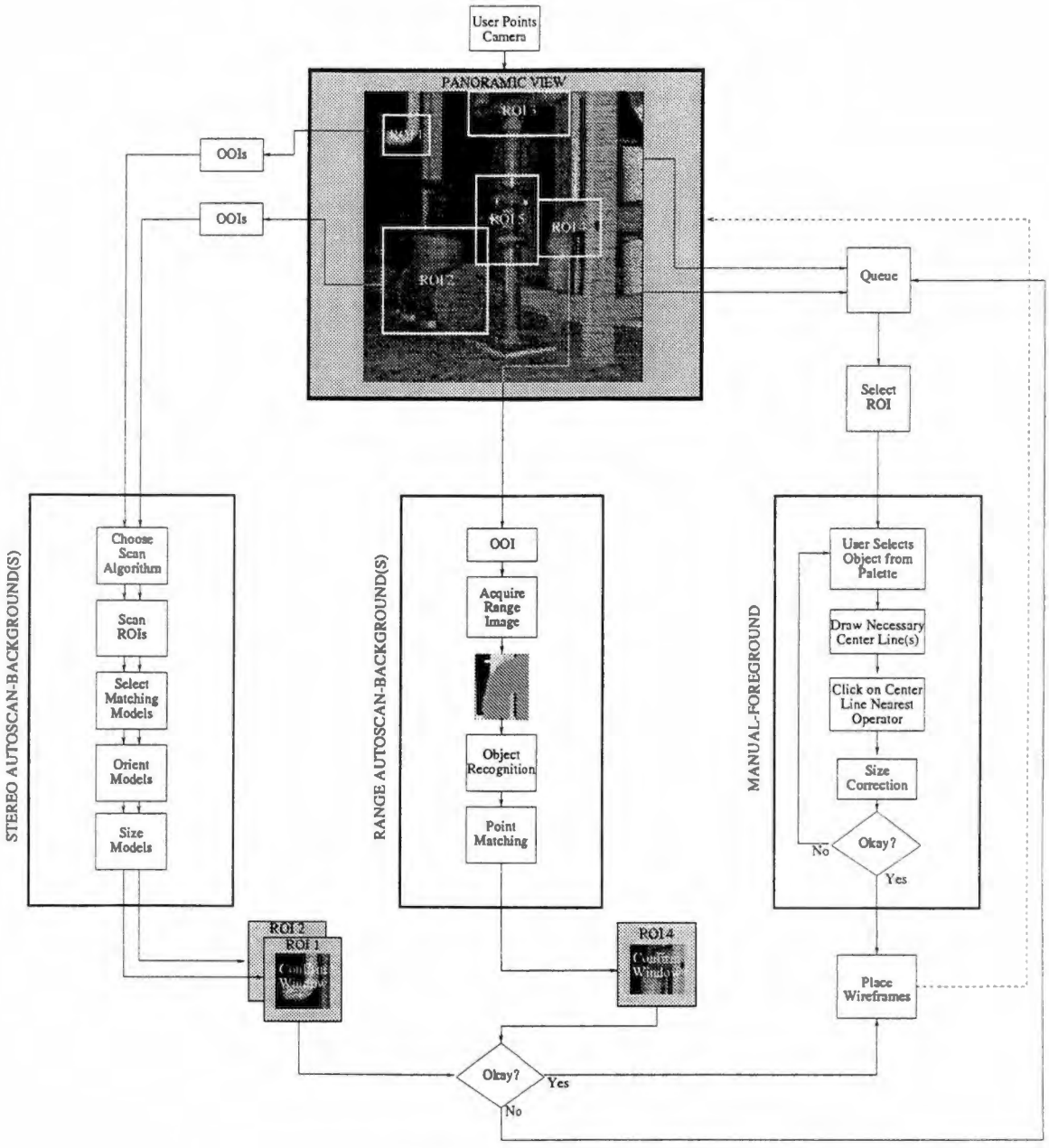

Figure 2.3: RTSA Functional Architecture

Chapter 3

System Configuration

3.1 Hardware

3.1.1 Schilling Manipulator

The Schilling Titan II shown in Figure 3.1, is a six-degree-of-freedom hydraulic manipulator weighing 225 pounds, with a reach of approximately 76 inches and a payload at full extension of 240 pounds [1]. It incorporates a two-finger gripper with a maximum opening of 5 inches. It is securely mounted on the floor of the Robotics and Electromechanical Systems Laboratory (Room M7 in the Dougherty Engineering Building) and is used to test RTSA and the telerobotic system in typical remote operations task.

3.1.2 Stereo Sensor Head

As illustrated in Figure 3.2, the stereo sensor head consists of two Panasonic® CCD cameras, a SICK laser range pointer, and a drive which can rotate about pan and tilt axes. The two digital cameras can produce black and white images with 640 $\times$ 480 resolution. The laser range finder is capable of measuring distances up to 16 meters at a resolution of 1 mm at a rate of about 30 Hz. A tripod is also used to mount the sensor head.

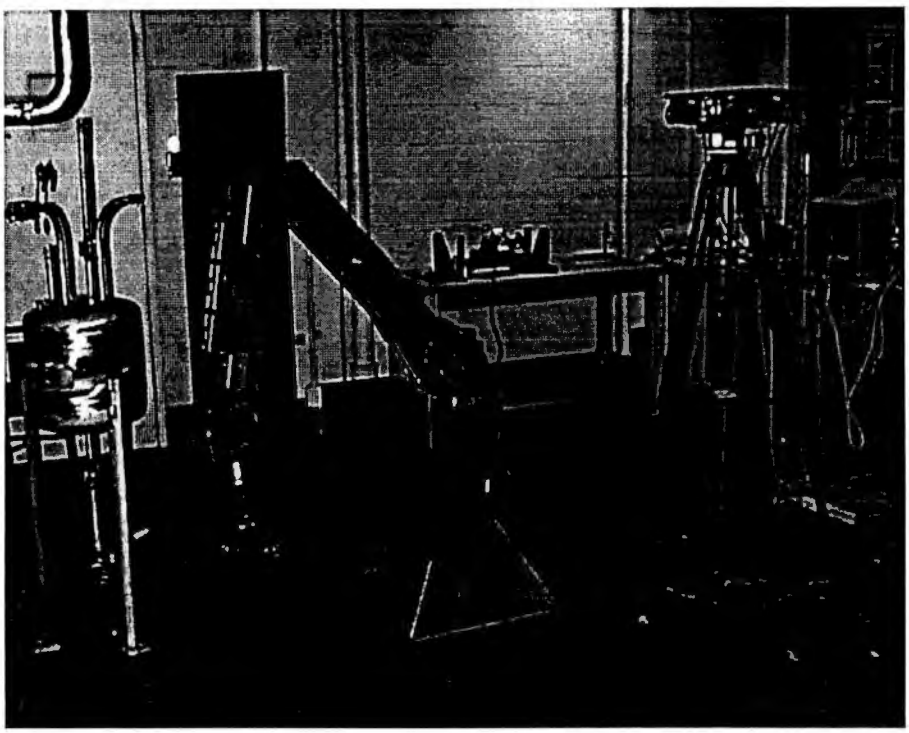

Figure 3.1: The Schilling Manipulator

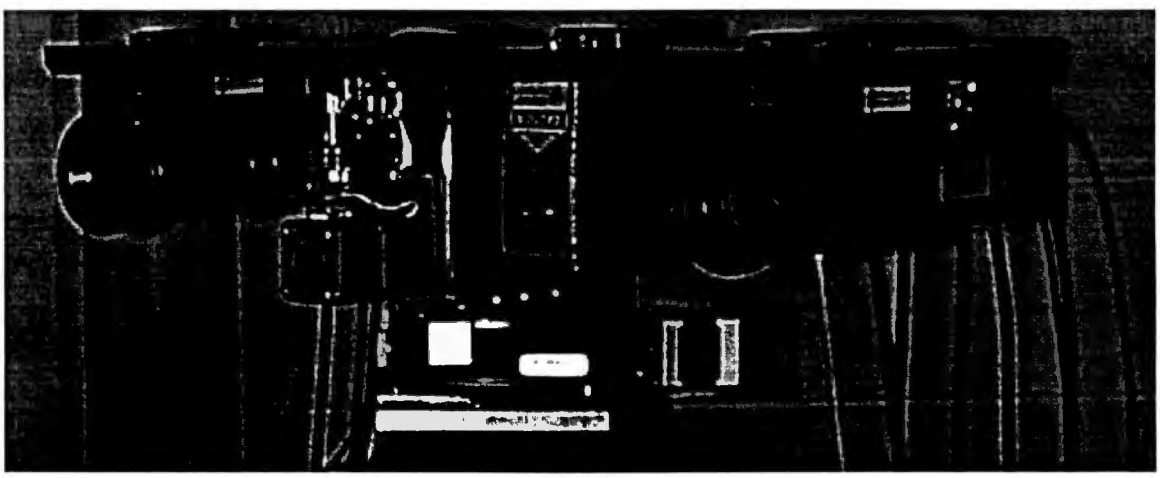

Figure 3.2: Stereo Sensor Head

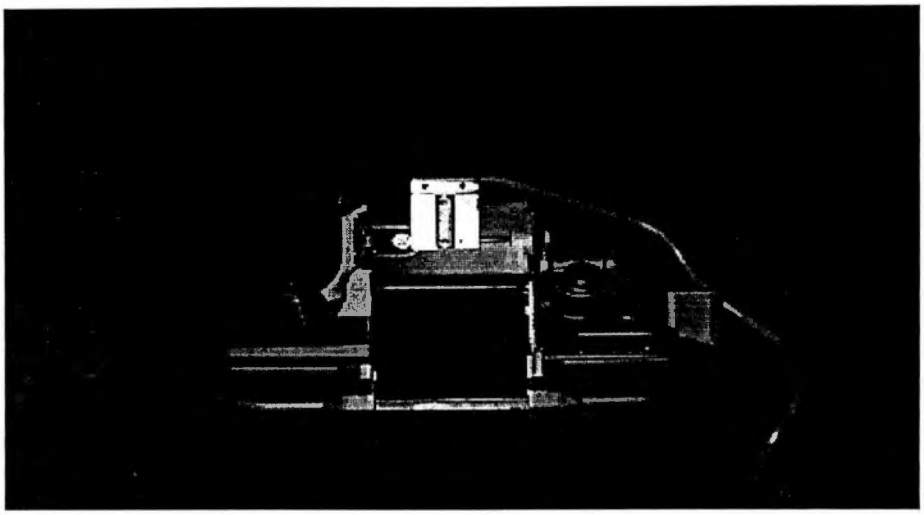

Figure 3.3: Milwaukee Band Saw

3.1.3 Band Saw

Figure 3.3 shows the Milwaukee® Band Saw that is being used as a pipe cutting tool for dismantlement demonstrations. It has a 0.5 inch wide, 10 tooth per inch blade which can be operated at variable speeds from 0-350 SFPM [6]. The saw has a maximum capacity of 4 3/4" x 4 3/4" for square stock and 4 3/4" diameter round stock.

3.2 Software Tools

3.2.1 Envision

Deneb's Envision® [7] is used as the 3-D model building software for the RTSA project. It provides a physics-based, 3D environment specifically for designing, verifying and prototyping of concept designs involving structures, mechanical systems, and humans. It is opened in Command Line Interface (CLI) mode and is thus commanded through a socket using special text commands.

3.2.2 Microsoft Visual C++

Microsoft Visual C++® 6.0 is chosen as the development software environment. This version includes the Microsoft Foundation Classes and Templates, which is composed of the Microsoft Foundation Class Library (MFC) and the Active Template Library (ATL) [4]. Most classes in the RTSA code are inherited from MFC, a set of C++ classes that together form the skeleton of a 32-bit application for Windows NT.

3.2.3 ControlShell

ControlShell® is the development tool created by Real-Time Innovations (RTI) [10] used to implement the telerobot controller. It allows users to create continuous flow diagrams graphically and link them with finite state machines to dictate the behavior of the system. It is installed on the near real time control computer to receive the commands from the task planner and parse those commands for execution.

Chapter 4

Task Planning

4.1 Planning and control configuration

The execution tasks of a robotic system involve multiple sequences of actions, which are connected or dependent upon each other logically and temporally [12]. The basic structure of the task planning and control system [11] is organized in layer hierarchy as shown in Figure 4.1.

4.2 Task Decomposition

Robots are designed to perform tasks. We use them instead of other machines because they are flexible. Robot task planning is in essence the planning of robot motion sequences to execute tasks. As mentioned in last section, the whole planning structure is a sort of layer hierarchy. Our objective is to tackle the problem of task planning using the tool of hierarchical decomposition. The decomposition hierarchy is:

- Task level
- Action level
- Robot level

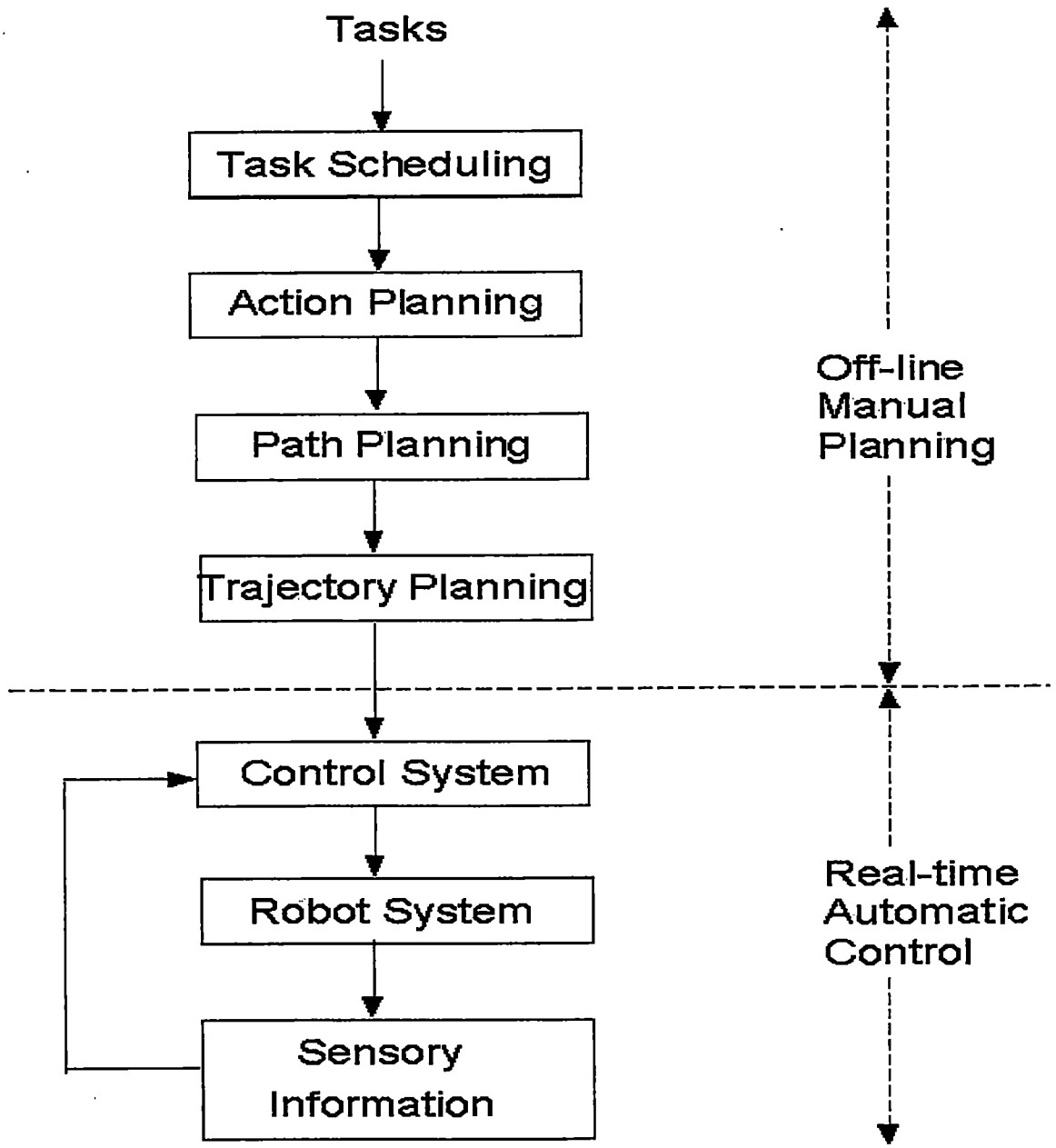

Figure 4.1: Planning and Control of Robotic Systems

- Joint level

Each of these hierarchy levels will be described in detail.

4.2.1 Task level

The user has to specify, in detail, what he wants the robot to do in this level. In remote dismantlement (the focus of RTSA), the principal task of it is to remove objects in the work scene. Examples of objects include pipes and pipe fittings like tees, elbows and flanges etc. Some custom objects, for example, vessels and tanks, also often exist in the work area.

4.2.2 Action level

Tasks can be decomposed into a sequence of primitive actions. A robot performs its task by executing those actions in the proper sequence. Action-level planning requires a model of the world in the domain of the task, which is the main research issue of the first phase of RTSA. Based on the model information and the features of tools and the robot, the operator uses his experience and expertise to identify the key points in the model, to specify tools and to provide cut angle and via point information. All these data are used to determine the sequence of primitive actions. Current actions considered in the work are:

- Move
- Pick tool
- Put back tool
- Change tool
- Cut

Among them, change tool is actually the combination of putting back the held tool and picking the expected tool.

4.2.3 Robot level

A robot can execute a set of low-level primitive operations to accomplish actions. Each action must be mapped into a sequence of these operations. Thus, the robot-level planner deals with the motion of the end effector. From the perspective of the programmer, these operational primitives define the robot in the same way as machine code defines a computer. For this reason, the set of such operations is sometimes referred to as robot machine code. Lots of information, like the location and characteristics of tools, size of the model objects, are needed to decompose the primitive actions into primitive operations. Decomposition of the primitive actions is shown below:

- Move

  (not decomposed)

- Pick tool

  (1) Move to the above point of the tool

  (2) Open gripper

  (3) Move down to the tool

  (4) Close gripper to grasp the tool

  (5) Move up back to the above point

- Put back tool

  (1) Move to the above point of the tool

  (2) Move down to the tool rack

  (3) open gripper to release the tool

  (4) Move up to the above point

- Change tool

Combination of pick tool and put back tool

- Cut

  (1) Turn on cut tool

  (2) Move along the cutting track to cut

  (3) Turn off cut tool

  (4) Back out along the original track to the start point

4.2.4 Joint level

A robot can be thought of as a set of links connected by joints. At this level, position, velocity, and force control systems regulate the actuators to control the robot dynamics. A near real-time computer in RTSA is responsible for receiving the robot machine code and converting those code into joint parameters versus time.

4.3 The flowchart of action-generating code

The code to generate the robot action is the core of the entire task planning code. The flowchart of action-generating part in Figure 4.2 illustrates how the finally planned action is created.

First, three linked lists — the end point list, the action list and the action description list are created. There are two kinds of end points — cut point and via point. Via point is a point that the end effector goes through in order to avoid obstacles. The end point list contains the position and orientation information of all the end points in a task. Those data will determine the position and orientation of the start point and final point of the end effector in each movement operation. The tool used at each end point is also specified in this list. Both the action list and the action description list

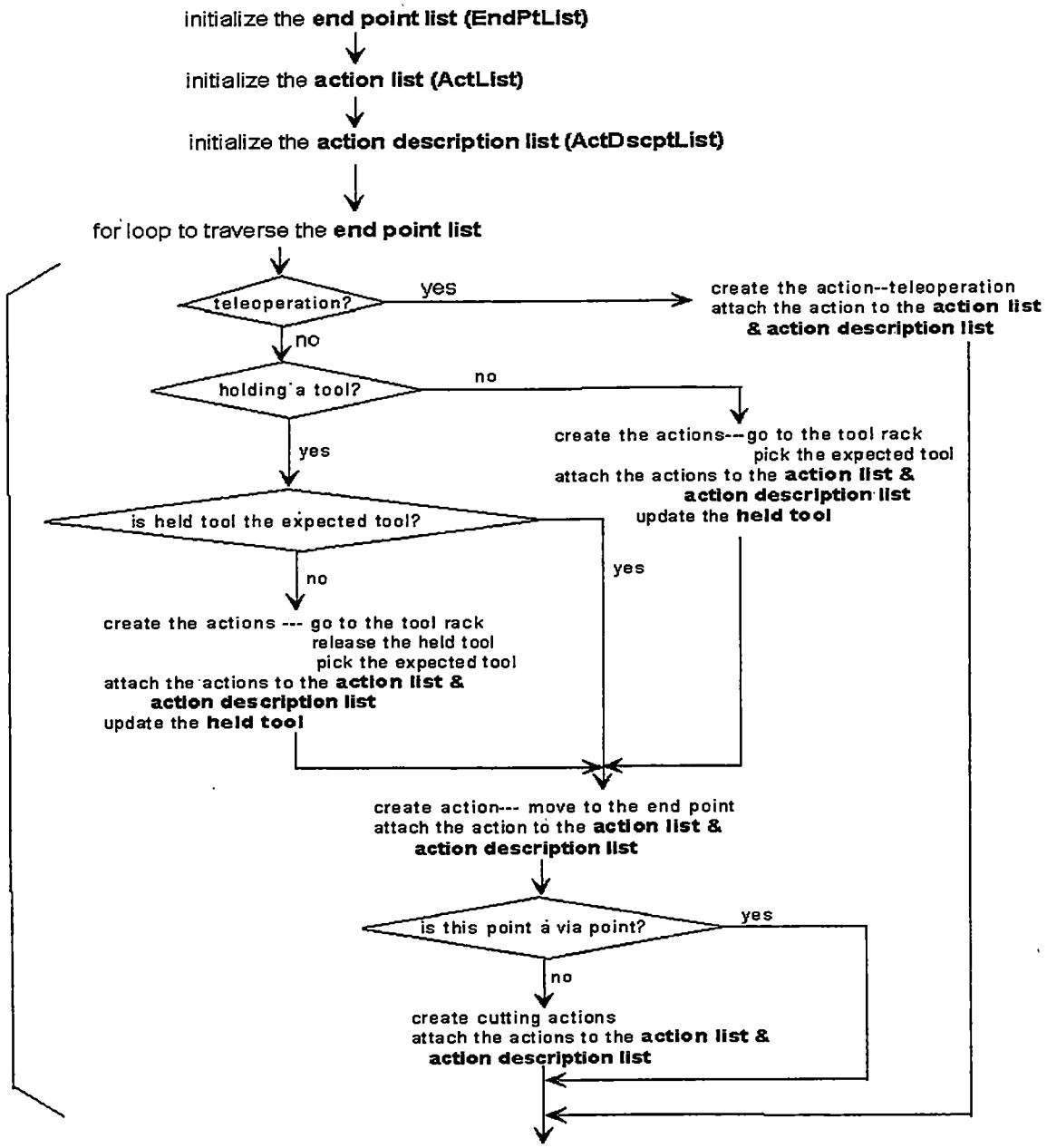

Figure 4.2: Flowchart of action generation

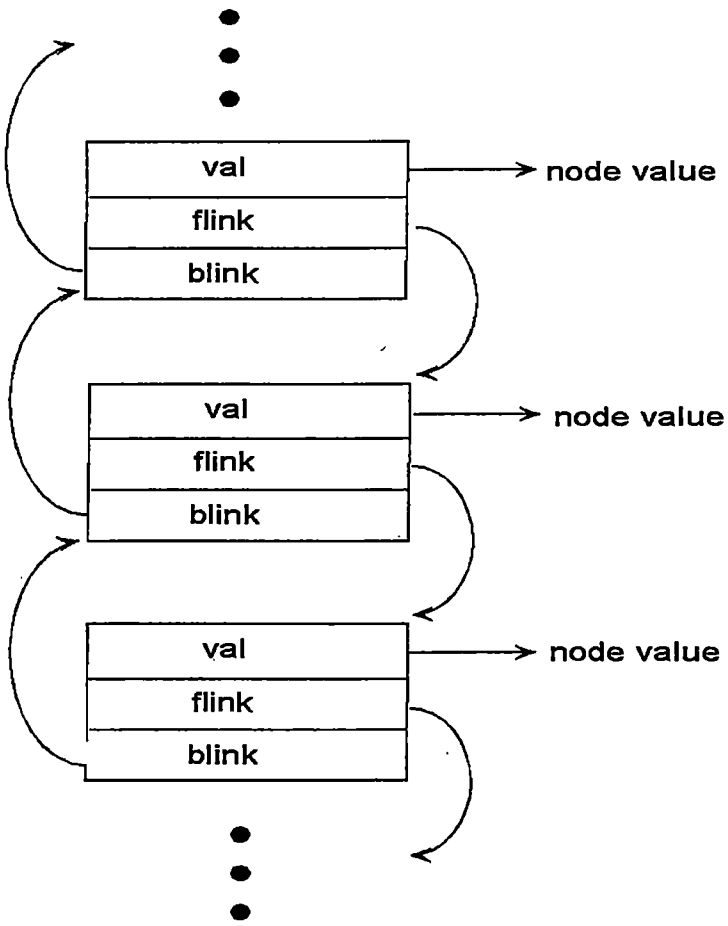

Figure 4.3: The Structure of Doubly Linked List

are derived from the end point list. From the hierarchical point of view, the action list is a level below the action description list since the action list actually encompasses the needed data for each primitive operation whereas the information involved in the action description list is primitive action data. However, they are the same thing in essence. The main purpose of the action description is to give the operator an oversight view of the plan without getting into too much detail.

All the lists mentioned above are doubly linked lists [5] (see Figure 4.3) in which each node has two pointers — one to the next node, and one to the previous node. This is important because one can not only traverse lists forward and backward but

also insert and delete nodes anywhere in the list easily. This convenience allows the operator to insert and delete end points and actions directly.

The program uses a for loop to traverse the end point list. If the robot is not holding a tool, then it must obtain a tool from the tool rack to do the cutting task. Two actions — "go to the tool rack" and "pick the expected tool" are generated and appended to the action list and action description list. But if the robot is holding a cut tool at the moment, the task planner checks to determine if the tool is the expected tool. If the answer is yes, the task can proceed. If the currently held tool is not the required tool, the robot has to go to the tool rack to change cut tool. Similarly, "go to the tool rack", "release the held tool" and "pick the expected tool" are created at this point. After all the work associated with the tool are completed, the robot moves to the end point which can be either via point or cut point. If the end point is a cut point, the robot turns on the tool and moves forward to cut. After cutting, it turns off the tool and returns along the original track. All these operations are included in the "cut actions". If the end point is a via point, the robot does not do any operation except for proceeding to the via point to avoid obstacles. The end point list is traversed until all the end points have been visited and then a complete action list is made to cover all the necessary steps to accomplish this segment of mission.

Chapter 5

The Graphical User Interface Design

The GUI (graphical user interface) design is one of the most important aspects of the planner. The GUI uses windows and menus with common graphic symbols instead of typed system commands to provide an intuitive environment. It is an interactive tool which not only provides a means for the user to input data but also displays some updated data. During the planning procedure, the operator has to understand and evaluate the display, formulate goals and provide command inputs through the mouse or joystick. The task planning code behind the GUI is like a black box, or processor, which reads in commands and data from the operator, manages the data, creates the proper plan based on the RTSA model.

5.1 The Overview of GUI

The GUI of task planning is structured like a tree, which has a root and a couple of branches. The root window contains six options which can invoke a branch window individually. As shown in Figure 5.1, five of these six options are in parallel. From the selection order point of view, each branch window is independent. That is to say, the operator can select any of them in any order without confusing the regular execution of the program. These

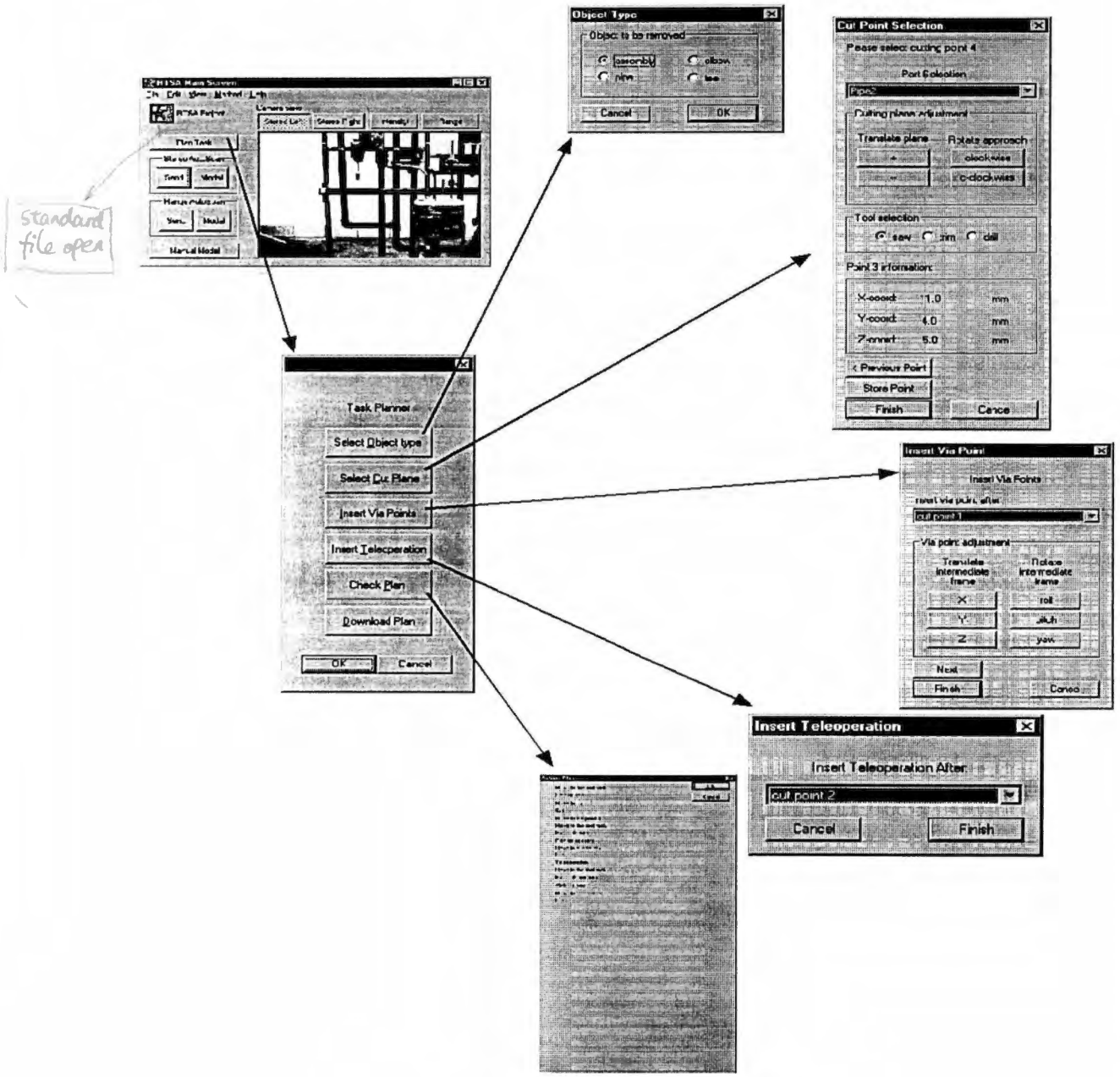

Figure 5.1: Overview of GUI Flow Chart

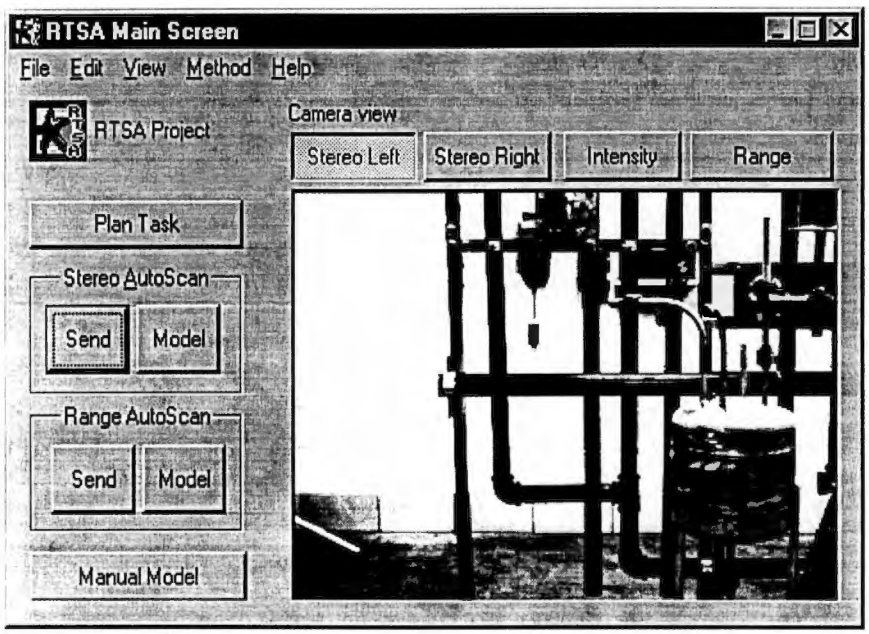

Figure 5.2: The RTSA Main Window

windows are internally associated with each other. If some information is missing because the operator forgot to input data correctly, the system will pop a window to remind and warn the operator about the incomplete data.

5.2 The RTSA Main Window

The main window shown in Figure 5.2 is the parent of all other windows in RTSA. It provides the entries to build a model using either manual, stereo autoscan, or range autoscan function. If a task model is available, the "Plan Task" button is pressed to invoke the Task Planner window.

5.3 The Task Planner Window

Figure 5.3 shows the task planner window which provides the operator with a series of options to create a plan to execute autonomous tool retrieval, robot motion, and cutting

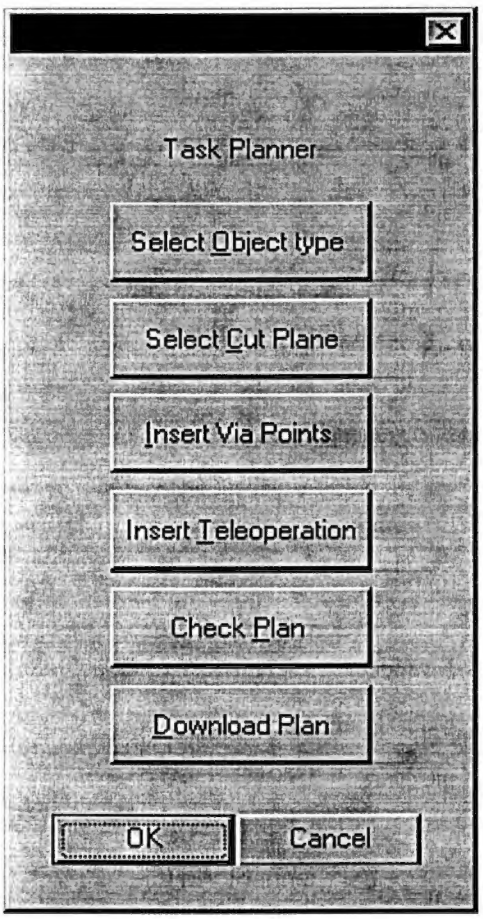

Figure 5.3: The Task Planner Window

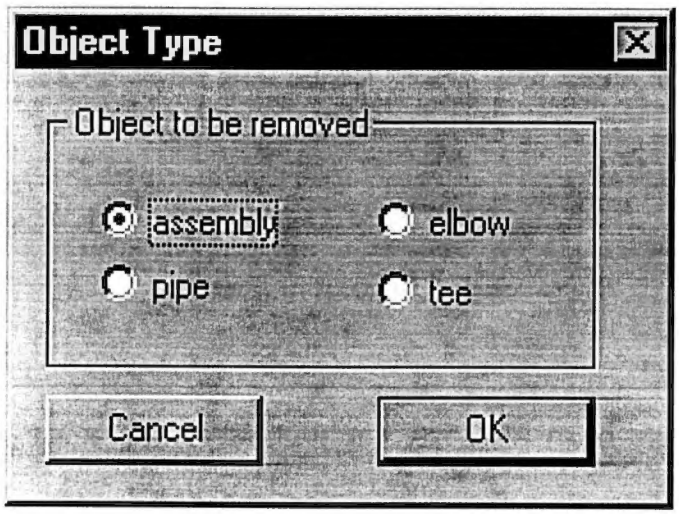

Figure 5.4: The Object Type Window

operations. Windows associated with each button allow the operator to define various aspects of the plan, including choice of type of objects to be removed from the scene, points on the items to be cut in the scene, insertion of via points between cut points, insertion of periods of teleoperation between autonomous operation, and to inspect the resulting plan. Each of these buttons has associated windows as described below except the download plan button which saves the low level commands assembled by the planner to a text file used by the control computer.

5.4 Select The Object Type

The object type window shown in Figure 5.4 will pop up when the "Select Object Type" button in the task planner window is pressed. In this window, the operator specifies the particular type of the object to be cut in the work scene for the next series of robot operations. The default option is "assembly", which means that an interconnected connection of pipes and pipe fittings are to be treated as a single object. This will be a commonly used option because of the flexibility it gives the operator to define the size and shape of an item.

Also assembly can increase the productivity in that the robot can remove larger quantities of equipment with fewer operations.

When assembly is the object type, the operator has to open the other windows from the task planner window to provide cut points, cut tools and insertion of via points and teleoperation because those are necessary information. To streamline the GUI, this window also contains process component options like pipe, elbow and tee. The operator does not need to supply other cutting information since the computer can automatically calculate those parameters based on the stored data for the pipe or pipe fitting models. After considering the specific conditions of the scene and the tradeoff between efficiency and burden on teleoperation, the operator has the freedom to go either way he thinks the best from his experience.

5.5 Select Cut Point

Because of the complexity of typical work scenarios, the selection of cut point locations and cutting angles may be limited in some extent. The cut point selection window shown in Figure 5.5 provides the operator the ability to define these parameters. First an item of the model built previously is chosen from the "Part Selection" drop down list. The selected part is also highlighted in the Envision window and a red cutting plane which defines the cutting position and direction will also appear and be attached to the base frame of the model of the selected part (See Figure 5.6).

In the cut point selection window, four translate and rotate buttons can be used to move and rotate the cutting plane along the center axis of the part until a satisfactory cutting location and direction is determined. For each cut point, the cut tool to be used at that point can also be specified through the "Tool Selection" radio button. The "Store Point" button is pressed to save all the data associated with this cut and x, y, z coordinates of the approach point (the point the robot begins to turn on tool and goes forward to cut)

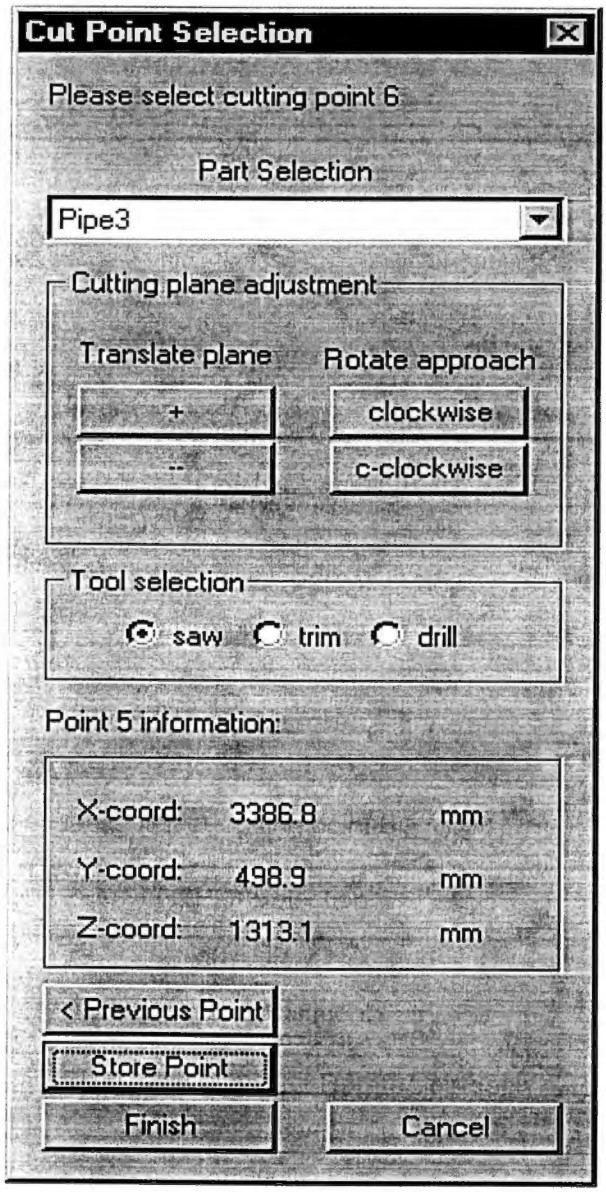

Figure 5.5: The Cut Point Selection Window

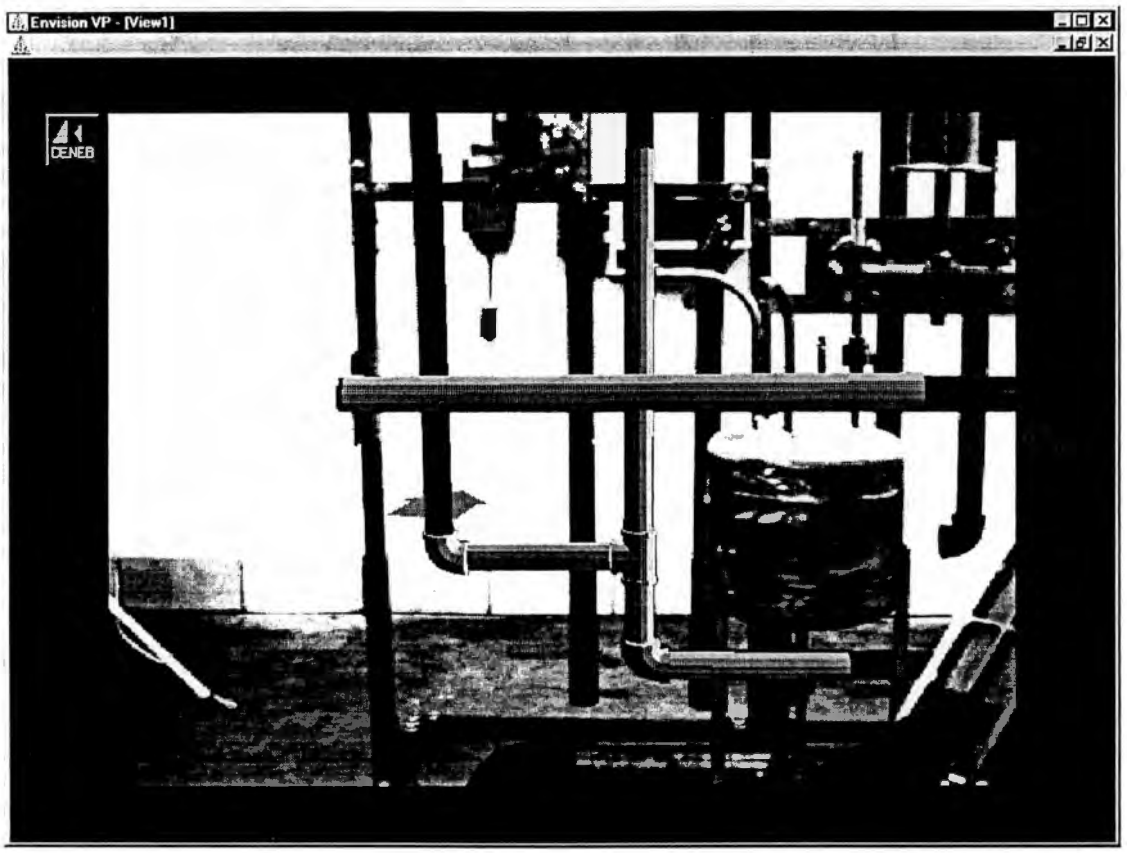

Figure 5.6: Envision Window With Cutting Plane Selected

will be displayed by edit box to provide a reference. If the operator is not content with the selection, he can go back to those points by clicking "Previous Point" button and modify the cutting plane. Even in the worst case — all the former selections are not acceptable, the "Cancel" button can be used to overturn previous decisions. Or the user can press "Finish" button to complete the cut point selection task.

5.6 Insert Via Point

Sometimes the robot cannot move from a cut point to the next cut point, or between the cut point and the tool rack directly, as there are obstacles in the way. With path obstacles, extra points must be inserted in the robot's trajectory. These points are called via points that the end effector of the robot will pass through to avoid obstacles.

After the selection of the cut points, the drop down list in the insert via point window in Figure 5.7 will contain a list of the cut points. Then the operator can select the proper point for insertion of a via point. Currently this window is missing the functionality to specify the position of via points. A promising way to solve this problem is to model a tiny object or specify a point on an object as marks. Then these marks can be moved to the proper position in which the via point should be.

5.7 Insert Teleoperation

The insert teleoperation window shown in Figure 5.8 is similar to the insert via point window shown in Figure 5.7. It also has a drop down list that contains previous operations. The operator uses this window to specify the operation after which the teleoperation will be inserted.

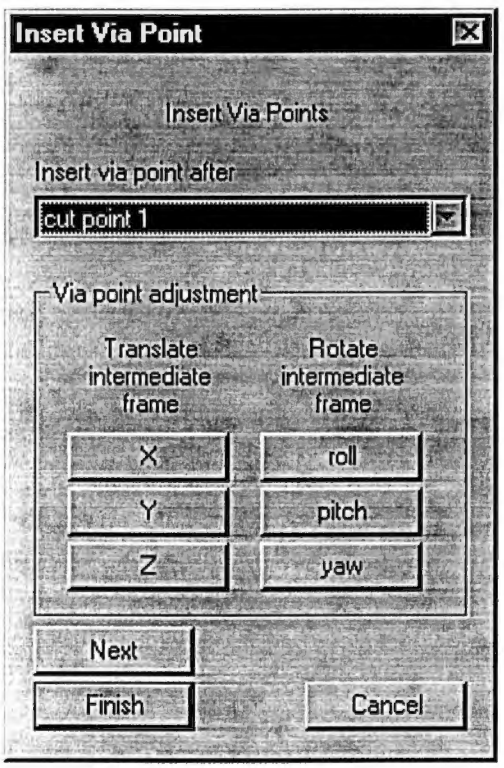

Figure 5.7: The Insert Via Point Window

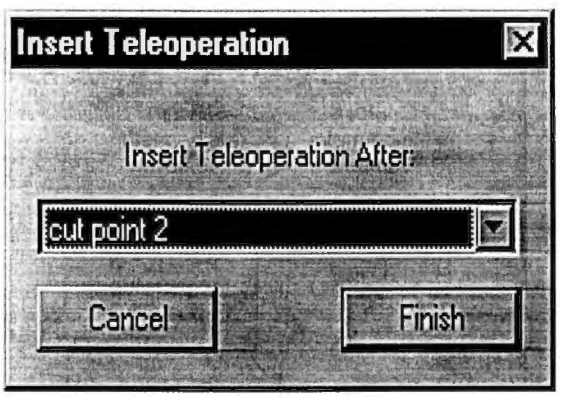

Figure 5.8: The Insert Teleoperation Window

5.8 Inspect Plan

After the task plan has been completed, the operator needs a way to easily inspect it. To inspect the plan, the "Check Plan" button in the task planner window is clicked to display the summary action plan window shown in Figure 5.9. The task plan elements are displayed in that window. The task plan elements are written in a condensed style so that the operator can efficiently check what the robot will do for the next sequence of operations. If the task plan is acceptable, the "Download Plan" button in the task planner window is pressed to save all the robot commands to a command file which will be discussed below.

5.9 Command File

An example of a command file produced by the task planner is shown in Figure 5.10. It is a text document containing the detailed operations that are to be performed by the telerobotic system. Each primary operation is a standard C structure composed of ten elements which are information regarding the opening and closing of the gripper, toggling of tool power, desired final position and orientation of the gripper for the movement operation, and a flag to identify teleoperation. This is a format that can be easily interpreted by the control computer. This command file will be downloaded by file transfer protocol(FTP) over the Ethernet to the telerobot control computer.

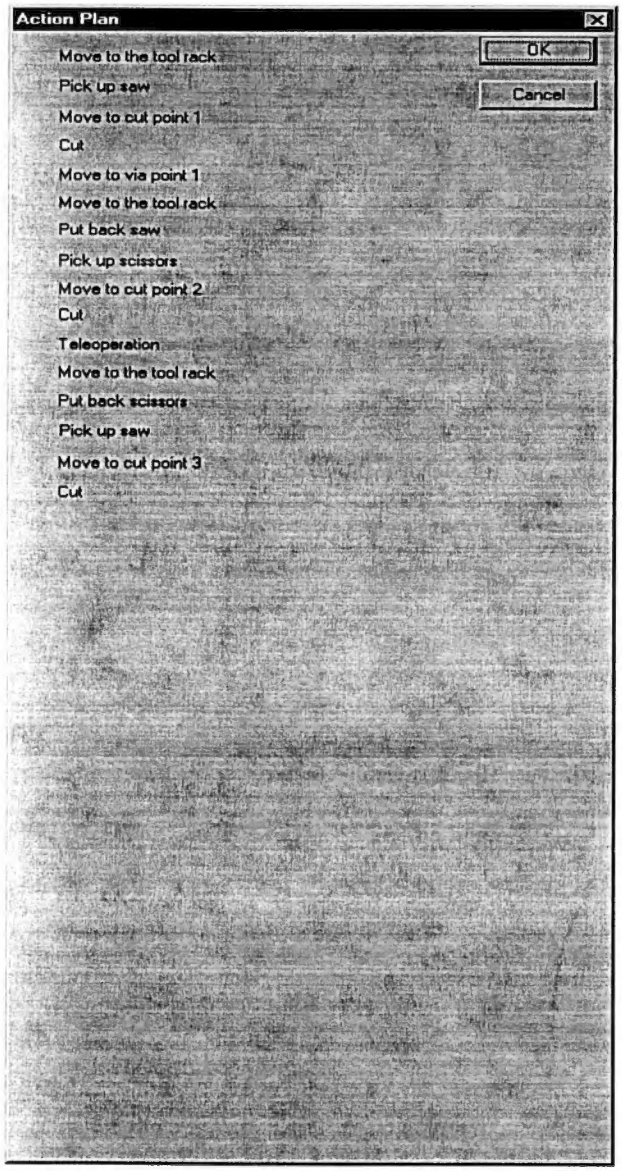

Figure 5.9: The High Level Plan

```
Operation 1:
  teleop    =        0 # not teleoperation
  move      =        1  # a movement operation
  gripAct   =        0  # the gripper remains the original status
  toolAct   =        0  # the tool remains the original status
  fPtX      =   10.000000  # the x coordinate of the final point
  fPtY      =   20.000000  # the y coordinate of the final point
  fPtZ      =   50.000000  # the z coordinate of the final point
  fPtPitch  =    0.000000  # the pitch angle of the final point
  fPtYaw    =    0.000000  # the yaw angle of the final point
  fPtRoll   =    0.000000  # the roll angle of the final point
Operation 2:
  teleop    =        0  # not teleoperation
  move      =        0  # not a movement operation
  gripAct   =        1  # open gripper
  toolAct   =        0  # the tool remains the original status
  fPtX      =    0.000000  # the x coordinate of the final point
  fPtY      =    0.000000  # the y coordinate of the final point
  fPtZ      =    0.000000  # the z coordinate of the final point
  fPtPitch  =    0.000000  # the pitch angle of the final point
  fPtYaw    =    0.000000  # the yaw angle of the final point
  fPtRoll   =    0.000000  # the roll angle of the final point
Operation 3:
  teleop    =        0  # not teleoperation
  move      =        1  # a movement operation
  gripAct   =        0  # the gripper remains the original status
  toolAct   =        0  # the tool remains the original status
  fPtX      =   10.000000  # the x coordinate of the final point
  fPtY      =   20.000000  # the y coordinate of the final point
  fPtZ      =   30.000000  # the z coordinate of the final point
  fPtPitch  =    0.000000  # the pitch angle of the final point
  fPtYaw    =    0.000000  # the yaw angle of the final point
  fPtRoll   =    0.000000  # the roll angle of the final point
Operation 4:
  teleop    =        0  # not teleoperation
  move      =        0  # not a movement operation
  gripAct   =        2  # close gripper
  toolAct   =        0  # the tool remains the original status
  fPtX      =    0.000000  # the x coordinate of the final point
  fPtY      =    0.000000  # the y coordinate of the final point
  fPtZ      =    0.000000  # the z coordinate of the final point
  fPtPitch  =    0.000000  # the pitch angle of the final point
  fPtYaw    =    0.000000  # the yaw angle of the final point
  fPtRoll   =    0.000000  # the roll angle of the final point
Operation 5:
  teleop    =        0  # not teleoperation
  move      =        1  # a movement operation
  gripAct   =        0  # the gripper remains the original status
  toolAct   =        0  # the tool remains the original status
  fPtX      =   10.000000  # the x coordinate of the final point
  fPtY      =   20.000000  # the y coordinate of the final point
  fPtZ      =   50.000000  # the z coordinate of the final point
  fPtPitch  =    0.000000  # the pitch angle of the final point
  fPtYaw    =    0.000000  # the yaw angle of the final point
  fPtRoll   =    0.000000  # the roll angle of the final point
Operation 6:
  teleop    =        0  # not teleoperation
  move      =        1  # a movement operation
  gripAct   =        0  # the gripper remains the original status
  toolAct   =        0  # the tool remains the original status
  fPtX      = 2967.258878  # the x coordinate of the final point
  fPtY      =  706.430949  # the y coordinate of the final point
```

Figure 5.10: An Example of Executable Task File

Chapter 6

Conclusions

Planning and scheduling technology offers considerable promise in automating telerobot operations. Planning and scheduling telerobot operations involves generating a sequence of low-level robot commands from a set of high-level goals. A task planning framework for robot operations utilizing extensive operator interaction was constructed in this thesis. The commands are in a simple format that can be easily parsed and processed by the real time control system of the robot. The working efficiency is increased since the human worker can command the robot to execute the planned actions and work on the other subtasks himself at the same time. The enhancement of productivity will result in greater economical benefit.

Chapter 7

Future Work

There are still some aspects of the task planner discussed here that need to be improved in the future. First, the layout of the user interface could be more graphical and streamlined. For example, it would be more intuitive if the position and order of cut points could be displayed in the Envision window when the operator selects them. This will enable him to have in mind an approximate picture of the operation trajectory that the manipulator will go through and benefit subsequent cut point selection.

Second, the work to provide the functionality to cut a particular pipe fitting automatically has to be completed. This is viable since the parameters of those pipe fittings are already known in the database. The cutting position and orientation can be calculated from those parameters. The monotonous cut point selection work can be reduced in this way.

Another remaining problem is the determination of via points. Via points are not easy to specify because they exist in "free" space. A useful approach is to attach an auxiliary point or object in "free" space and then move it to the proper position to represent the via point.

Lastly, there are currently too many windows in the RTSA user interface. This is against the rule of GUI design and should be avoided. The number of windows needs to be reduced by combining some small windows and optimize the structure of the GUI in the future.

Bibliography

Bibliography

[1] S. Everett, W. Hamel, P. Smith, X. Fu, S. Singh, G. Zhang, M. Khalid, V. Gospodareva, M. Smith, and S. Richardson. *Robot Task Scene Analyzer Technical Overview*, Oct. 1999.

[2] W. Hamel and J. Osborn. Human-interacitive and multi-spectral robot task space modeling. *American Nuclear Society 8th International Topical Meeting on Robotics And Remote Systems, Pittsburgh, Pennsylvania, USA*, May 1999.

[3] W. Hamel, P. Smith, C. Widner, S. Hale, S. Mitchell, X. Fu, S. Singh, and A. Ononye. *Robot Task Space Analyzer Phase One Report*, Sept. 1998.

[4] D. A. Holzgang and L. Napper. *Teach Yourself Visual C++ 5.0.* MIS Press, first edition, 1997.

[5] S. Holzner. *C++ Programming.* Brady Publishing, first edition, 1991.

[6] http://www.bayverte.com/Milwaukee/HTML/6230.html.

[7] http://www.deneb.com.

[8] P. J. McKerrow. *Introduction To Robotics.* Addison-Wesley, first edition, 1991.

[9] S. Mitchell. A graphical user interface for human-interactive, in-situ 3d task space scene modeling. Master of science in mechanical engineering, University of Tennessee, Sept. 1998.

[10] Real-Time Innovations, Inc, 155A Moffett Park Drive, Suite 111, Sunnyvale, CA 94089. *ControlShell User's Manual, Version 6.0*, Jan. 1999.

[11] M. Song. *Interaction of Task Scheduling, Sensing, Planning, and Control In A Robotic Manufacturing Work-Cell.* Doctor of philosophy in system science and mathematics, Washington University, Aug. 1997.

[12] N. Xi and T.-J. Tarn. Modeling and control of multiple segment robotic operations in a perceptive reference frame. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Waseda, Japan*, June 1997.

Appendices

Appendix A

Selected Software Listings

A.1 dllist_struct.h

```
typedef struct dllist {
  struct dllist *flink;
  struct dllist *blink;
  void * val;
} *Dllist;
```

A.2 MyList.h

```
// MyList.h: interface for the MyList class.
//
//////////////////////////////////////////////////////////////////////

#include "dllist_struct.h"

#if !defined(AFX_MYLIST_H__F700E411_3856_11D3_872E_00C04FA35F00__INCLUDED_)
#define AFX_MYLIST_H__F700E411_3856_11D3_872E_00C04FA35F00__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class MyList
{
public:
MyList();
virtual ~MyList();

Dllist List;                // the pointer to the head of the linked list
Dllist tempList;            // to store the pointer to the head of the linked list temporarily

Dllist new_dllist();
    void dll_insert_b(Dllist node, void * v);
void dll_insert_a(Dllist n, void * val);
void dll_append(Dllist l, void * val);
void dll_prepend(Dllist l, void * val);
void dll_delete_node(Dllist node);
int  dll_empty(Dllist l);
void free_dllist(Dllist l);
Dllist dll_first(Dllist d);
Dllist dll_last(Dllist d);
Dllist dll_nil(Dllist l);
Dllist dll_next(Dllist d);
Dllist dll_prev(Dllist l);
void * dll_val(Dllist l);
};

#endif // !defined(AFX_MYLIST_H__F700E411_3856_11D3_872E_00C04FA35F00__INCLUDED_)
```

A.3 MyList.cpp

```
// MyList.cpp: implementation of the MyList class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Rtsa2.h"
#include "MyList.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

MyList::MyList()
{

}

MyList::~MyList()
{

}

// the member functions

Dllist MyList::new_dllist()
{
  Dllist d;

  d = (Dllist) new dllist;
  d->flink = d;
  d->blink = d;
  return d;
}

void MyList::dll_insert_b(Dllist node, void * v)       /* Inserts before a given node */
{
  Dllist newList;

  newList = (Dllist) new dllist;
  newList->val = v;

  newList->flink = node;
  newList->blink = node->blink;
  newList->flink->blink = newList;
  newList->blink->flink = newList;
}

void MyList::dll_insert_a(Dllist n, void * val)        /* Inserts after a given node */
{
  dll_insert_b(n->flink, val);
}

void MyList::dll_append(Dllist l, void * val)     /* Inserts at the end of the list */
{
  dll_insert_b(l, val);
}

void MyList::dll_prepend(Dllist l, void * val)    /* Inserts at the beginning of the list */
{
  dll_insert_b(l->flink, val);
}

void MyList::dll_delete_node(Dllist node) /* Deletes an arbitrary item */
{
  node->flink->blink = node->blink;
  node->blink->flink = node->flink;
  delete (node);
}

int MyList::dll_empty(Dllist l)
{
  return (l->flink == l);
}
```

```
void MyList::free_dllist(Dllist l)
{
  while (!dll_empty(l)) {
    dll_delete_node(dll_first(l));
  }
  delete(l);
}

Dllist MyList::dll_first(Dllist d)
{
  return d->flink;
}

Dllist MyList::dll_last(Dllist d)
{
  return d->blink;
}

Dllist MyList::dll_nil(Dllist l)
{
  return l;
}

Dllist MyList::dll_next(Dllist d)
{
  return d->flink;
}

Dllist MyList::dll_prev(Dllist l)
{
  return (Dllist) (l->blink);
}

void * MyList::dll_val(Dllist l)
{
  return l->val;
}
```

A.4 EndPt_Action_Struct.h

```
struct End_point {
bool telop;    //  1: teleoperation

int is_via;    // flag;   0:this is a cut point       1: this is a via point

int hold_tool;      // 0: empty   1: saw     2: scissors     3: drill

double coord_x;
double coord_y;
double coord_z;

double roll;
double pitch;
double yaw;   };


struct action {
bool teleop;        // flag to judge if the action is teleoperation   1: this action is teleoperation
bool move;          // flag    1: this action is just a movement
int gripAct;        // 0: remain     1: open gripper     2: close gripper
int toolAct;         // 0: remain     1: turn on tool      2: turn off tool
int holdTool;

double fPtX;        // the x coordinate of the final point
double fPtY;        // the y coordinate of the final point
double fPtZ;        // the z coordinate of the final point

double fPtPitch;    // the pitch angle of the final point
double fPtYaw;      // the yaw angle of the final point
double fPtRoll;     // the roll angle of the final point

// double maxSpeed;    // the maximun speed of the movement
};
```

A.5 Rtsa2.h

```
// Rtsa2.h : main header file for the RTSA2 application
```

```
//

#if !defined(AFX_RTSA2_H__9BDE196F_F4D7_11D1_A767_006008179102__INCLUDED_)
#define AFX_RTSA2_H__9BDE196F_F4D7_11D1_A767_006008179102__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 <afxtempl.h>
#include "resource.h"          // main symbols
#include "C:\mil\include\mil.h"      // Matrox Image Library headers
#include "C:\mil\include\mwinmil.h"  //
#include "C:\mil\include\milsetup.h" //
#include "C:\mil\include\milerr.h"   //
#include "pan_tilt.h"                // Pan tilt header
#include "servo_lens.h"              // Servo lens header
#include "laser_range.h"             // Laser range finder header
#include "StereoDlg.h"               // Stereo dialog header
#include "object_model.h"
#include "PopUpDlg.h"
#include "ObjectDlg.h"
#include "CutSelect.h"
#include "PartApprovalDlg.h"
#include "MyList.h"

#define CCD_Offset_X 250 // x-offset of CCD from sensor head base frame in sens head coors (mm)
#define CCD_Offset_Y -5 // y-offset of CCD from sensor head base frame in sens head coors (mm)
#define CCD_Offset_Z -30 // z-offset of CCD from sensor head base frame in sens head coors (mm) -12

#define Lrf_Offset_X 100 // x-offset of LRF from sensor head base frame in sens head coors (mm)
#define Lrf_Offset_Y -12.15 // y-offset of LRF from sensor head base frame in sens head coors (mm)
#define Lrf_Offset_Z 65 // z-offset of LRF from sensor head base frame in sens head coors (mm) -12

#define LrfPanOff 0.019 // Pan misalignment of LRF from camera gaze vector (previously -0.0056)
#define LrfTiltOff -0.01 // Tilt misalignment of LRF from camera gaze vector (previously -0.0244)

#define Lrf_Offset 0 // Laser range finder roll-over offset (mm)
// Desired amount depends on object of interest location
// one rollover = 2047 mm for old LRF

#define Radius_1 16.7005             // Radius of 1'' pipe (mm) (actually 1.315" outer diameter)
#define Radius_2 30.1625  // (actually )
#define Radius_25 36.5125  // (actually )
#define Radius_3 44.450  // (actually )
#define Radius_4 57.150  // (actually 4.5" outer diameter)

//#define Z_World 1495.5
#define Z_World 1485//1487 // Height of sensor head base frame above world coor sys in Envision
#define X_Camera  300    //x position of camera frame in the Pan Tilt Frame
#define Y_Camera  10     //y position of camera frame in the Pan Tilt Frame
#define Z_Camera  -100   //z position of camera frame in the Pan Tilt Frame

//Cameras should be set with tilt of -3.3 degrees as measured by physical scale to be level with the floor

/////////////////////////////////////
// Declare Globals
//
void OnAutoApproved();

/* Stereo autoscan client */
extern unsigned __stdcall StereoASClient(void *);

/* Range autoscan client */
extern unsigned __stdcall RangeASClient(void *);

/////////////////////////////////////
// Set ENVISION mode
//#define NOENVISION
#define ENVISION

/////////////////////////////////////////////////////////////////////////////
// CRtsa2App:
// See Rtsa2.cpp for the implementation of this class
//

class CRtsa2App : public CWinApp
```

```
{
public:
CString FileToOpenAtStartup;
CString UndoFileName;
BOOL FileToOpenFlag;
void ASClient();
CRtsa2App();
long NumberOfDigitizer; // Number of digitizers available on the system
long SizeX; // Buffer Size X
long SizeY; // Buffer Size Y
long Band; // Number of input color bands of the digitizer
long BufferAttributes; // Buffer attributes that will be allocated
long int **nPT_Stat; // Pan--Tilt status array
int *nLSL_Stat; // LeftServoLens status array
int *nRSL_Stat; // RightServoLens status array
long int    nLRF_Stat;          // Laser Range Finder status value
pan_tilt *nPT_Control; // Pan--TIlt class pointer
servo_lens *nLSL_Control; // LeftServoLens class pointer
servo_lens *nRSL_Control; // RightServoLens class pointer
laser_range *nLRF_Control;      // Laser Range Finder class pointer
float nPanAngle; // Pan angle
float nTiltAngle; // Tilt angle
float nZoomNum; // Zoom number
float nFocusNum; // Focus distance
float nApertureNum; // Aperture number
int nScheduleNum; // Enum value of _Schedule_ from Combo
int nSizeNum; // Enum value of _Size_ from Combo
int nTypeNum; // Enum value of _Type_ from Combo
int nAutoScanNum; // Enum value of _Type_ from Combo
int nMethod; // Manual=0, AutoScan Stereo=1, Range=2, Approve=3
int nUserView; // Left stereo=0, Right=1, Range=2, Intensity=3
CStereoDlg* pStereoMode;
int nSocketDesc; // This is the socket descriptor for CLI commands
char** cCliInit;
int nRoiNumber;
CPoint      CalPoint[2];
float       CalZoom[2];
long int    CalPointRange;
float       CalPanAngle;
float       CalTiltAngle;
float       **CameraCal;
CObjectDlg* pManualModeless;
CCutSelect* pCutSelectModeless;
CPartApprovalDlg* pPartAprModeless;
int seq; // Used to keep track of the envision part name
int AprMethod; // 0 is manual, 1 is AutoScan
float PVPanAngle; // Pan angle when the PV image is captured
float PVTiltAngle; // Tilt angle when the PV image is captured
int         holdTool;           // label to show what tool the robot is holding now: used to generate the actions
                                    // 0: holding nothing   1: holding a saw  2: holding a scissors  3: holding a drill
int         RealholdTool;       // the real status label

      bool   firstcheck;                // flag to indicate if this is the first time to check the plan

// int         haveList;           // flag to show if a EndPtList has been created

MyList      EndPtList;          // the linked list of the end point information
MyList      ActList;            // the linked list of the (struct) action information
MyList      ActDscptList;       // the linked list of the strings to describe the action


struct StereoSendStruct Sender;
struct StereoSendStruct *pSender;

//struct objectInfo ManualPart;
struct objectInfo* pManualPart;
//struct objectInfo* pObjListEl;

struct objectInfo* pSASPart;
struct objectInfo SASPart;

struct objectInfo* pRASPart;
struct objectInfo RASPart;

int PipeCnt, ElbCnt, TeeCnt, CustCnt;

int LeftImgCoords[4], RightImgCoords[4]; // Pixel coordinates of image fragments chosen for stereo
// autoscan


CList< int, int> ManualSendList;
```

```
CList< objectInfo, objectInfo> ManualPartList;
CList< objectInfo, objectInfo> AutoscanPartList;
CList< objectInfo, objectInfo> *pAutoscanPartList;

CList< objectInfo, objectInfo> StereoAutoscanPartList;
CList< objectInfo, objectInfo> RangeAutoscanPartList;

CList< objectInfo, objectInfo> MasterPartList;

int AutoWaitFlag;
CPopUpDlg* pPopUpMode;

MIL_ID EventId;
MIL_ID MilApplication; // Display identifier.
MIL_ID  MilSystem; // System identifier.
MIL_ID MilDisplay; // Display identifier.
MIL_ID MilDisplay1; // Display identifier.
MIL_ID MilImage;        // Image buffer identifier(color).
MIL_ID MilImage1;        // Image buffer identifier(monochrome).
MIL_ID MilImage2;        // Image buffer identifier(monochrome).
MIL_ID MilImageLeft; // Small left image identifier
MIL_ID MilImageRight; // Small right image identifier
MIL_ID  MilDigitizer; // Digitizer identifier.
MIL_ID BuffID; // Image buffer ID(color).
MIL_ID BuffID1; // Image buffer ID(monochrome).
MIL_ID BuffID2; // Image buffer ID(monochrome).
MIL_ID BuffIDLeft; // Image buffer ID(monochrome).
MIL_ID BuffIDRight; // Image buffer ID(monochrome).
MIL_ID MainBuffLeft; // Image buffer ID(monochrome).
MIL_ID MainBuffRight; // Image buffer ID(monochrome).
MIL_ID BuffIDLeftSmall; // Image buffer ID(monochrome).
MIL_ID BuffIDRightSmall; // Image buffer ID(monochrome).
MIL_ID MainBuffLeftSmall; // Image buffer ID(monochrome).
MIL_ID MainBuffRightSmall; // Image buffer ID(monochrome).
MIL_ID LeftSASImgFrag; // Image fragment from left stereo image
MIL_ID RightSASImgFrag; // Image fragment from right stereo image

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CRtsa2App)
public:
virtual BOOL InitInstance();
//}}AFX_VIRTUAL

// Implementation

//{{AFX_MSG(CRtsa2App)
// 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()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_RTSA2_H__9BDE196F_F4D7_11D1_A767_006008179102__INCLUDED_)
```

A.6 Rtsa2.cpp

```
// Rtsa2.cpp : Defines the class behaviors for the application.
//

#include "StdAfx.h"
#include "Rtsa2.h"
#include "Rtsa2Dlg.h"
#include "StereoDlg.h"
#include "shlibdefs.h"
#include "SendCliCommand.h"

/***STEREO HEAD INCLUDE FILES***/
#include<process.h>
#include"rtsa.h"
/*********************************/
```

```
// WE NEED TO PUT THIS SOMEWHERE
extern unsigned __stdcall ApprovalCheck(void *params);

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CRtsa2App

BEGIN_MESSAGE_MAP(CRtsa2App, CWinApp)
//{{AFX_MSG_MAP(CRtsa2App)
// NOTE - the ClassWizard will add and remove mapping macros here.
//    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRtsa2App construction

CRtsa2App::CRtsa2App()
{
// TODO: add construction code here,
// Place all significant initialization in InitInstance
nUserView = 0;
nMethod = 0;
pStereoMode = NULL;
// pPopUpMode = NULL;
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CRtsa2App object

CRtsa2App theApp;

/////////////////////////////////////////////////////////////////////////////
// CRtsa2App initialization

BOOL CRtsa2App::InitInstance()
{
if (!AfxSocketInit())
{
AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
return FALSE;
}

// 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
Enable3dControls(); // Call this when using MFC in a shared DLL
#else
Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

// Parse the command line to see if launched as OLE server
if (RunEmbedded() || RunAutomated())
{
// Register all OLE server (factories) as running.  This enables the
//  OLE libraries to create objects from other applications.
COleTemplateServer::RegisterAll();
}
else
{
// When a server application is launched stand-alone, it is a good idea
//  to update the system registry in case it has been damaged.
COleObjectFactory::UpdateRegistryAll();
```

```
}

// Allocate an application
MappAlloc(M_DEFAULT,&MilApplication);
// Allocate a system
  MsysAlloc(M_SYSTEM_SETUP,M_DEF_SYSTEM_NUM,M_COMPLETE,&MilSystem);

// Allocate a display
    MdispAlloc(MilSystem, M_DEFAULT, M_DISPLAY_SETUP, M_DEFAULT, &MilDisplay);
// Allocate a second display
    MdispAlloc(MilSystem, M_DEFAULT, M_DISPLAY_SETUP, M_DEFAULT, &MilDisplay1);

// Inquire number of digitizers available on the system
MsysInquire(MilSystem,M_SYS_DIGITIZER_NUM,&NumberOfDigitizer);
// Digitizer is available
if (NumberOfDigitizer)
{
// Allocate a digitizer
MdigAlloc(MilSystem,M_DEFAULT,M_CAMERA_SETUP,M_DEFAULT,&MilDigitizer);
// Stop live grab when window is disable
      MsysControl(MilSystem,M_STOP_LIVE_GRAB_WHEN_DISABLED,M_ENABLE);
// Inquire digitizer informations
   SizeX = MdigInquire(MilDigitizer,M_SIZE_X,M_NULL);
SizeY = MdigInquire(MilDigitizer,M_SIZE_Y,M_NULL);
Band = MdigInquire(MilDigitizer,M_SIZE_BAND,M_NULL);
}

// This is where to change the input channel that is synchronized since the
// master sync may not come from the frame grabber.
MdigChannel(MilDigitizer, M_CH1);


BufferAttributes=M_IMAGE+M_DISP+M_GRAB;
// Allocate a buffer
   BuffID = MbufAllocColor(MilSystem, Band, 640L, 480L, 8+M_UNSIGNED, BufferAttributes, &MilImage);

BuffID1 = MbufAlloc2d(MilSystem, 640L, 480L, 8+M_UNSIGNED, BufferAttributes, &MilImage1);
BuffID2 = MbufAlloc2d(MilSystem, 640L, 480L, 8+M_UNSIGNED, BufferAttributes, &MilImage2);

BuffIDLeftSmall = MbufAlloc2d(MilSystem, 320L, 240L, 8+M_UNSIGNED, BufferAttributes, &MilImageLeft);
BuffIDRightSmall = MbufAlloc2d(MilSystem, 320L, 240L, 8+M_UNSIGNED, BufferAttributes, &MilImageRight);

MainBuffLeft = MbufAlloc2d(MilSystem, 640L, 480L, 8+M_UNSIGNED, BufferAttributes, &MilImage1);
MainBuffRight = MbufAlloc2d(MilSystem, 640L, 480L, 8+M_UNSIGNED, BufferAttributes, &MilImage2);

MbufAlloc2d(MilSystem, 320L, 240L, 8+M_UNSIGNED, BufferAttributes, &MainBuffLeftSmall);
MbufAlloc2d(MilSystem, 320L, 240L, 8+M_UNSIGNED, BufferAttributes, &MainBuffRightSmall);


// Allocate a temporary memory space for image fragments to be sent to the
// stereo autoscan server

MbufAlloc2d(MilSystem, 160L, 160L, 8+M_UNSIGNED, BufferAttributes, &LeftSASImgFrag);
MbufAlloc2d(MilSystem, 160L, 160L, 8+M_UNSIGNED, BufferAttributes, &RightSASImgFrag);

/***STEREO HEAD LOCAL VARIABLES***/
struct pt_stat_struct *pt_struct;
struct sl_stat_struct *lsl_struct,*rsl_struct;
struct lrf_stat_struct *lrf_struct;
unsigned int pt_id,lsl_id,rsl_id,lrf_id;
int pt_flag,lsl_flag,rsl_flag,lrf_flag;
/************************************/

/***ENVISION LOCAL VARIABLES***/
CString CliCmd;
CString CliRsp;
/*******************************/

/***STEREO HEAD--MEMORY ALLOCATION***/
/* Allocate memory for status array */
nPT_Stat=(long int **)calloc(3,sizeof(long int *));
nPT_Stat[0]=nPT_Stat[1]=(long int *)calloc(13,sizeof(long int));
nPT_Stat[2]=nPT_Stat[1]+6;

/* Allocate memory for left lens status array */
nLSL_Stat=(int *)calloc(6,sizeof(int));

/* Allocate memory for right lens status array */
nRSL_Stat=(int *)calloc(6,sizeof(int));

/* Allocate memory for pan/tilt status structure */
```

```
pt_struct=(pt_stat_struct *)calloc(1,sizeof(pt_stat_struct));

/* Allocate memory for LSL status structure */
lsl_struct=(sl_stat_struct *)calloc(1,sizeof(sl_stat_struct));

/* Allocate memory for RSL status structure */
rsl_struct=(sl_stat_struct *)calloc(1,sizeof(sl_stat_struct));

/* Allocate memory for LRF status structure */
lrf_struct=(lrf_stat_struct *)calloc(1,sizeof(lrf_stat_struct));

/***STEREO HEAD--SERIAL PORT ALLOCATION AND INITIALIZATION***/
/* Open/Initialize the pan/tilt unit */
pan_tilt pt_control(2,3,nPT_Stat);
nPT_Control=&pt_control;

/* Open/Initialize the left servo-lens */
servo_lens lsl_control(4,nLSL_Stat);
nLSL_Control=&lsl_control;

/* Open/Initialize the right servo-lens */
servo_lens rsl_control(5,nRSL_Stat);
nRSL_Control=&rsl_control;

/* Open/Initialize the LRF */
laser_range lrf_control(1);
nLRF_Control=&lrf_control;

/* Clear status thread termination flags */
pt_flag=lsl_flag=rsl_flag=lrf_flag=1;

/***STEREO HEAD--ASSIGN STATUS STRUCTURE MEMBERS ***/
/* Assign status structure class pointers */
pt_struct->pt_stat_class=nPT_Control;
lsl_struct->sl_stat_class=nLSL_Control;
rsl_struct->sl_stat_class=nRSL_Control;
lrf_struct->lrf_stat_class=nLRF_Control;

/* Assign status structure status matrix pointers */
pt_struct->pt_stat_array=nPT_Stat;
lsl_struct->sl_stat_array=nLSL_Stat;
rsl_struct->sl_stat_array=nRSL_Stat;
lrf_struct->lrf_stat_val=&nLRF_Stat;

/* Assign status structure thread termination flags */
pt_struct->pt_term_flag=&pt_flag;
lsl_struct->sl_term_flag=&lsl_flag;
rsl_struct->sl_term_flag=&rsl_flag;
lrf_struct->lrf_term_flag=&lrf_flag;

// Must be before status monitor threads
// Initiallize the socket for sending CLI commands to Envision
nSocketDesc = InitCliSocket();

/***STEREO HEAD--THREAD OFF STATUS MONITOR***/
/* Pan/Tilt status thread */
_beginthreadex(NULL,0,get_pt_status,(void *)pt_struct,0,&pt_id);
/* Left ServoLens status thread */
_beginthreadex(NULL,0,get_sl_status,(void *)lsl_struct,0,&lsl_id);
/* Right ServoLens status thread */
_beginthreadex(NULL,0,get_sl_status,(void *)rsl_struct,0,&rsl_id);
/* Laser Range Finder status thread */
_beginthreadex(NULL,0,get_lrf_status,(void *)lrf_struct,0,&lrf_id);


// Initialize cameral calibration memory
CameraCal=(float **)calloc(2,sizeof(float *));

// Set the envision name counter to zero

//(Someone changed this to a 2, I changed it back (?)) SEE 5-21-99
seq = 0;

//  Pipe, elbow, and tee running counts, used for username

PipeCnt = 0;
TeeCnt = 0;
ElbCnt = 0;
CustCnt = 0;

RealholdTool = 0;    // at the very beginning, the robot is holding nothing
```

```
firstcheck = 1;  // will be the first time to check the plan


/// ************************
/// ** File Open Engine  **
/// ************************

// Popup Filename

// Parse command line for standard shell commands, DDE, file open
      CCommandLineInfo cmdInfo;
      ParseCommandLine(cmdInfo);

   CString CommandFeature = cmdInfo.m_strFileName;

   if(CommandFeature != "")
   {
    FileToOpenFlag = TRUE;
FileToOpenAtStartup = CommandFeature;
}
   else
   {
   }

/// ******** END File Open Engine

// Point the AutoScan Parts list pointer
pAutoscanPartList=&AutoscanPartList;
pSender = &Sender;

// **STEREO HEAD--START CONTROL DIALOG**
pStereoMode = new CStereoDlg;                    // modaless dialog box -- stereo head control window
pStereoMode->Create(IDD_STEREO_DIALOG,NULL);

// **Main Program Windows (With Menu)**
CRtsa2Dlg dlg;
m_pMainWnd = &dlg;
int nResponse = dlg.DoModal();
if (nResponse == IDOK || nResponse==IDCANCEL)
{
// ******** Code here to handles when the dialog is dismissed with OK *********

/***STEREO HEAD--RETURN TO HOME***/
nPT_Control->pan_vel_scale(1);
nPT_Control->tilt_vel_scale(1);
nPT_Control->pan_abs(0,nPT_Stat);
nPT_Control->tilt_abs(0,nPT_Stat);
nLSL_Control->zoom(Def_Zoom,nLSL_Stat);
nLSL_Control->focus(Def_Focus,nLSL_Stat);
nLSL_Control->iris(Def_Iris,nLSL_Stat);
nRSL_Control->zoom(Def_Zoom,nRSL_Stat);
nRSL_Control->focus(Def_Focus,nRSL_Stat);
nRSL_Control->iris(Def_Iris,nRSL_Stat);

// Destroy Stereo Control Window
pStereoMode->DestroyWindow();

/***STEREO HEAD--FREE MEMORY***/

free(nPT_Stat[0]);
free(nPT_Stat);
free(nLSL_Stat);
free(nRSL_Stat);
free(pt_struct);
free(lsl_struct);
free(rsl_struct);
free(lrf_struct);
free(CameraCal[0]);
free(CameraCal[1]);
free(CameraCal);

net_close_socket( nSocketDesc );
}
else
{
AfxMessageBox("I should have Quit by Now", MB_OK, NULL);
}

// Since the dialog has been closed, return FALSE so that we exit the
//  application, rather than start the application's message pump.
```

```
return FALSE;
}

void CRtsa2App::ASClient()
{

}
```

A.7 CutSelect.h

```
#if !defined(AFX_CUTSELECT_H__129F4F8F_2D7D_11D3_8718_00C04FA35F00__INCLUDED_)
#define AFX_CUTSELECT_H__129F4F8F_2D7D_11D3_8718_00C04FA35F00__INCLUDED_

#include "EndPt_Action_Struct.h"
//#include "dllist_struct.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CutSelect.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CCutSelect dialog

class CCutSelect : public CDialog
{
// Construction
public:
struct End_point * ptr_EndPt;          // pointer to the newly created end point
CString promptMsgStr;
int selectedCutPntNo ;          // to judge how many cutting points have been selected
char MsgStr[256];               // to store the prompt information
char MsgStr1[40];
char MsgStr2[30];
char MsgStr3[30];
char MsgStr4[30];
CCutSelect(CWnd* pParent = NULL);   // standard constructor

// the position and orientation of the base frame of the model
double baseX;
double baseY;
double baseZ;
double baseR;
double baseP;
double baseYaw;

double r11;
double r12;
double r13;
double r14;
double r21;
double r22;
double r23;
double r24;
double r31;
double r32;
double r33;
double r34;
double a11;
double a12;
double a13;
double a14s;        // start point
double a14f;        // final point
double a21;
double a22;
double a23;
double a24s;
double a24f;
double a31;
double a32;
double a33;
double a34s;
double a34f;


CString ENVpartname;
      CString devName;
```

```
float transPosSel; // Translation along x-direction of cutting plane frame
// from original pipe base frame
float rollPosSel; // Roll around x-direction of cutting plane frame
// from original pipe base frame
float pipeLen; // Selected pipe length

void *  ptr;          // used to traverse the linked list
int    firstPnt;      // a label to judge if the selected point is the first cutting point

char buffer[20];       // used for test
int  flag;          // used for test

// Dialog Data
//{{AFX_DATA(CCutSelect)
enum { IDD = IDD_CUTPNTSELECT };
int m_PartSelection;
CString m_CutPtNo;
int m_ToolRadio;
CString m_PointInfo;
CString m_xCoord;
CString m_yCoord;
CString m_zCoord;
//}}AFX_DATA


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

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CCutSelect)
afx_msg void OnCloseupPartselect();
afx_msg void OnDropdownPartselect();
virtual void OnCancel();
virtual void OnOK();
virtual BOOL OnInitDialog();
afx_msg void OnTranspos();
afx_msg void OnTransneg();
afx_msg void OnRotcw();
afx_msg void OnRotccw();
afx_msg void OnNextptButton();
afx_msg void OnPrevpntButton();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CUTSELECT_H__129F4F8F_2D7D_11D3_8718_00C04FA35F00__INCLUDED_)
```

A.8 CutSelect.cpp

```
// CutSelect.cpp : implementation file

#include "stdafx.h"
#include "Rtsa2.h"
#include "CutSelect.h"
#include "SendCliCommand.h"
#include "math.h"
#include "offset.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCutSelect dialog
```

```
CCutSelect::CCutSelect(CWnd* pParent /*=NULL*/)
: CDialog(CCutSelect::IDD, pParent)
{
//{{AFX_DATA_INIT(CCutSelect)
m_PartSelection = -1;
m_CutPtNo = _T("Please select cutting point 1");
m_ToolRadio = -1;
m_PointInfo = _T("");
m_xCoord = _T("");
m_yCoord = _T("");
m_zCoord = _T("");
//}}AFX_DATA_INIT
}


void CCutSelect::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CCutSelect)
DDX_CBIndex(pDX, IDC_CUTPART_SELECT, m_PartSelection);
DDX_Text(pDX, IDC_CUTPTNO_EDIT, m_CutPtNo);
DDX_Radio(pDX, IDC_SAW_RADIO, m_ToolRadio);
DDX_Text(pDX, IDC_POINTINFO_EDIT, m_PointInfo);
DDX_Text(pDX, IDC_XCOORD_EDIT, m_xCoord);
DDX_Text(pDX, IDC_YCOORD_EDIT, m_yCoord);
DDX_Text(pDX, IDC_ZCOORD_EDIT, m_zCoord);
//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCutSelect, CDialog)
//{{AFX_MSG_MAP(CCutSelect)
ON_CBN_CLOSEUP(IDC_CUTPART_SELECT, OnCloseupPartselect)
ON_CBN_DROPDOWN(IDC_CUTPART_SELECT, OnDropdownPartselect)
ON_BN_CLICKED(IDC_TRANSPOS, OnTranspos)
ON_BN_CLICKED(IDC_TRANSNEG, OnTransneg)
ON_BN_CLICKED(IDC_ROTCW, OnRotcw)
ON_BN_CLICKED(IDC_ROTCCW, OnRotccw)
ON_BN_CLICKED(IDC_NEXTPT_BUTTON, OnNextptButton)
ON_BN_CLICKED(IDC_PREVPNT_BUTTON, OnPrevpntButton)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCutSelect message handlers

BOOL CCutSelect::Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect,
                                        CWnd* pParentWnd, UINT nID, CCreateContext* pContext)
{
return CDialog::Create(IDD, pParentWnd);
}

BOOL CCutSelect::OnInitDialog()
{
CDialog::OnInitDialog();

firstPnt = 1;          // will select the first point

m_ToolRadio = 0;       // Set the default cutting mode as saw
UpdateData(FALSE);

transPosSel = 0;
rollPosSel = 0;

flag = 0;

// Fill listing in combo box of all parts in MasterPartList database

      POSITION SearchPos;
struct objectInfo ObjListEl;
struct objectInfo *pObjListEl;

SearchPos = ((CRtsa2App*)AfxGetApp())->MasterPartList.GetHeadPosition();

while(SearchPos!=NULL)
{
ObjListEl = ((CRtsa2App*)AfxGetApp())->MasterPartList.GetNext(SearchPos);

pObjListEl = &ObjListEl;
```

```
CComboBox* pPartBox = (CComboBox*) GetDlgItem(IDC_CUTPART_SELECT);
pPartBox->AddString(pObjListEl->Username);
}

// XFU adds on 07/06/99:   to exhibit the prompting words

selectedCutPntNo = 1;            // the next selected cutting point will be point (1+1)=2
      UpdateData(FALSE);

return TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}


void CCutSelect::OnCloseupPartselect()
{

int i;
char cliMsg[50], coordStr[50], seqNo[10];
struct objectInfo ObjListEl;
struct objectInfo* pObjListEl;
POSITION SearchPos;
CString cliCmd, resp;

// Get user selected device from combo box (unless no parts are shown in list)

UpdateData(TRUE);

if(m_PartSelection != -1)
{

// Count up to the "m_PartSelection" element in the master part list to get selected part

SearchPos = ((CRtsa2App*)AfxGetApp())->MasterPartList.GetHeadPosition();

for (i=0; i<m_PartSelection; i++)
{
((CRtsa2App*)AfxGetApp())->MasterPartList.GetNext(SearchPos);
}

ObjListEl = ((CRtsa2App*)AfxGetApp())->MasterPartList.GetNext(SearchPos);
pObjListEl = &ObjListEl;

ENVpartname = pObjListEl->ENVpart;

           pipeLen = pObjListEl->len;

// Indicate to user which part was selected (by rendering red)

sprintf(cliMsg,"SET '" + ENVpartname + "' COLOR TO $RED");
SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

           // Determine what the name will be when the cutting plane is inserted into Envision
// (Equivalent to first unused "Dummy#")

((CRtsa2App*)AfxGetApp())->seq = 0;

do
{
(((CRtsa2App*)AfxGetApp())->seq)++;
sprintf(seqNo,"%d",(((CRtsa2App*)AfxGetApp())->seq));

if (((CRtsa2App*)AfxGetApp())->seq != 1)
devName = "Dummy#"+(CString)seqNo;
else
devName = "Dummy";

sprintf(cliMsg,"INQUIRE DEVICE '"+devName+"' RENDER");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

// while test for device existence is true, continue in loop
// (existence tested by inquiring render property, assumes part default rendering as '$flat')

} while(!resp.Compare("0: $flat;"));


// Retrieve cutting plane, position it coincident to the pipe frame, and make it red, transparent

sprintf(cliMsg,"RETRIEVE PART 'templates/cutPlane'");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);
```

```
cliCmd = "SET PART '" + devName + ":cutPlane' RENDER TO $TRANSPARENT";
resp = SendCliCommand( cliCmd,((CRtsa2App*)AfxGetApp())->nSocketDesc);

cliCmd = "SET PART '" + devName + ":cutPlane' COLOR TO $RED";
resp = SendCliCommand( cliCmd,((CRtsa2App*)AfxGetApp())->nSocketDesc);

sprintf(coordStr,"%.2f, %.2f, %.2f, %.2f, %.2f, %.2f",
pObjListEl->pos.x, pObjListEl->pos.y, pObjListEl->pos.z,0,0,0);
cliCmd = "PLACE PART '" + devName + ":cutPlane' AT COORD " + CString(coordStr);
resp = SendCliCommand( cliCmd,((CRtsa2App*)AfxGetApp())->nSocketDesc);

sprintf(coordStr,"%.2f, %.2f, %.2f",pObjListEl->ornt.y, pObjListEl->ornt.p, pObjListEl->ornt.r);
cliCmd = "ROTATE PART '" + devName + ":cutPlane' TO " + CString(coordStr) + " IN 1";
resp = SendCliCommand( cliCmd,((CRtsa2App*)AfxGetApp())->nSocketDesc);

cliCmd = "UPDATE GRAPHICS";
resp = SendCliCommand( cliCmd,((CRtsa2App*)AfxGetApp())->nSocketDesc);

baseR = pObjListEl->ornt.r;
baseP = pObjListEl->ornt.p;
baseYaw = pObjListEl->ornt.y;
baseX = pObjListEl->pos.x;
baseY = pObjListEl->pos.y;
baseZ = pObjListEl->pos.z;

}

UpdateData(FALSE);
}

void CCutSelect::OnDropdownPartselect()
{

int i;
      POSITION SearchPos;
char cliMsg[50];
struct objectInfo ObjListEl;
struct objectInfo* pObjListEl;
CString resp;

UpdateData(TRUE);

transPosSel = 0;

if(m_PartSelection != -1)
{
SearchPos = ((CRtsa2App*)AfxGetApp())->MasterPartList.GetHeadPosition();

for (i=0; i<m_PartSelection; i++)
{
((CRtsa2App*)AfxGetApp())->MasterPartList.GetNext(SearchPos);
}

ObjListEl = ((CRtsa2App*)AfxGetApp())->MasterPartList.GetNext(SearchPos);
pObjListEl = &ObjListEl;

ENVpartname = pObjListEl->ENVpart;

// Turn off selection indicator (return to yellow rendering)

sprintf(cliMsg,"SET '" + ENVpartname + "' COLOR TO $YELLOW");
SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

// Delete cutting plane object from Envision model

sprintf(cliMsg,"DELETE PART '" + devName + ":cutPlane'");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

}
}

void CCutSelect::OnCancel()
{
char cliMsg[50];
CString resp;

// Delete cutting plane object from Envision model

sprintf(cliMsg,"DELETE PART '" + devName + ":cutPlane'");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);
```

```
//AfxMessageBox(resp);

// Turn off selection indicator (return to yellow rendering)

sprintf(cliMsg,"SET '" + ENVpartname + "' COLOR TO $YELLOW");
SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

CDialog::OnCancel();
}

void CCutSelect::OnOK()              // if the "Finish" buttom is pressed (Be careful: not the "OK" button),  XFU
{
char cliMsg[50];
CString resp;

if (!firstPnt)             // if there are some end points in the list
{
    ( ((CRtsa2App*)AfxGetApp())->EndPtList ).List = ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList;
                        // the operator hits the "Finish" button--
                  // that is to say: the operator is satisfied with the selected points, then the temporary
                        // list is used as the formal list

    ((CRtsa2App*)AfxGetApp())->firstcheck = 1;    // will be the first time to check the new plan
}

// Save cutting plane location information...

// Delete cutting plane object from Envision model

sprintf(cliMsg,"DELETE PART '" + devName + ":cutPlane'");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

// Turn off selection indicator (return to yellow rendering)

sprintf(cliMsg,"SET '" + ENVpartname + "' COLOR TO $YELLOW");
SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

CDialog::OnOK();
}


void CCutSelect::OnTranspos()
{
char cliMsg[50];
CString resp;

// Send command to translate cutting plane in Envision (uses a 10mm translation)
// Limit motion so cutting plane remains on pipe

if (transPosSel<=pipeLen)
{
sprintf(cliMsg,"TRANSLATE PART '" + devName + ":cutPlane' BY 0,0,10 IN 1");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

sprintf(cliMsg,"UPDATE GRAPHICS");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

transPosSel+=10;
}
}

void CCutSelect::OnTransneg()
{
char cliMsg[50];
CString resp;

// Send command to translate cutting plane in Envision (uses a 10mm translation)
// Limit motion so cutting plane remains on pipe

if (transPosSel>=0)
{
sprintf(cliMsg,"TRANSLATE PART '" + devName + ":cutPlane' BY 0,0,-10 IN 1");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

sprintf(cliMsg,"UPDATE GRAPHICS");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

transPosSel-=10;
}
}
```

```
void CCutSelect::OnRotcw()
{
char cliMsg[50];
CString resp;

// Send command to translate cutting plane in Envision (uses a 10mm translation)

sprintf(cliMsg,"ROTATE PART '" + devName + ":cutPlane' ABOUT Z_AXIS BY 5 IN 1");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

sprintf(cliMsg,"UPDATE GRAPHICS");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

rollPosSel+=5;
}

void CCutSelect::OnRotccw()
{
char cliMsg[50];
CString resp;

// Send command to translate cutting plane in Envision (uses a 10mm translation)

sprintf(cliMsg,"ROTATE PART '" + devName + ":cutPlane' ABOUT Z_AXIS BY -5 IN 1");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);
//AfxMessageBox(resp);

sprintf(cliMsg,"UPDATE GRAPHICS");
resp = SendCliCommand(cliMsg,((CRtsa2App*)AfxGetApp())->nSocketDesc);

rollPosSel-=5;
}

// XFU added on 07/06/99

void CCutSelect::OnNextptButton()
{
// save the selected cutting point position and orientation information to the linked list

UpdateData(TRUE);

if (firstPnt)
( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList =
                        ( ((CRtsa2App*)AfxGetApp())->EndPtList ).new_dllist();
                                        // create a new linked list: End Point List

firstPnt = 0;                           // the subsequent points are not the first point any more


// Enable the " < Previous point " button
GetDlgItem( IDC_PREVPNT_BUTTON )->EnableWindow( TRUE );


r11 = cos(baseR)*cos(baseP);
r12 = cos(baseR)*sin(baseP)*sin(baseYaw) - sin(baseR)*cos(baseYaw);
r13 = cos(baseR)*sin(baseP)*cos(baseYaw) + sin(baseR)*sin(baseYaw);
r14 = baseX;
r21 = sin(baseR)*cos(baseP);
r22 = sin(baseR)*sin(baseP)*sin(baseYaw) + cos(baseR)*cos(baseYaw);
r23 = sin(baseR)*sin(baseP)*cos(baseYaw) - cos(baseR)*sin(baseYaw);
r24 = baseY;
r31 = -sin(baseP);
r32 = cos(baseP)*sin(baseYaw);
r33 = cos(baseP)*cos(baseYaw);
r34 = baseZ;

a11 = r11*cos(rollPosSel);
a12 = -r11*sin(rollPosSel) + r12*cos(rollPosSel);
a13 = r13;
a14s = r11*(cos(rollPosSel)*TOOLEE_X - sin(rollPosSel)*TOOLEE_Y + CENTERTOTOOL*cos(rollPosSel))
     + r12*(sin(rollPosSel)*TOOLEE_X + cos(rollPosSel)*TOOLEE_Y + CENTERTOTOOL*sin(rollPosSel))
 + r13*(TOOLEE_Z+transPosSel) + r14;
a14f = r11*(cos(rollPosSel)*TOOLEE_X - sin(rollPosSel)*(TOOLEE_Y-CENTERTOTOOL))
     + r12*(sin(rollPosSel)*TOOLEE_X + cos(rollPosSel)*(TOOLEE_Y-CENTERTOTOOL))
 + r13*(TOOLEE_Z+transPosSel) + r14;
a21 = r21*cos(rollPosSel) + r22*sin(rollPosSel);
a22 = -r21*sin(rollPosSel) + r22*cos(rollPosSel);
a23 = r23;
a24s = r21*(cos(rollPosSel)*TOOLEE_X - sin(rollPosSel)*TOOLEE_Y + CENTERTOTOOL*cos(rollPosSel))
     + r22*(sin(rollPosSel)*TOOLEE_X + cos(rollPosSel)*TOOLEE_Y + CENTERTOTOOL*sin(rollPosSel))
 + r23*(TOOLEE_Z + transPosSel) + r24;
```

```
a24f = r21*(cos(rollPosSel)*TOOLEE_X - sin(rollPosSel)*(TOOLEE_Y-CENTERTOTOOL))
     + r22*(sin(rollPosSel)*TOOLEE_X + cos(rollPosSel)*(TOOLEE_Y-CENTERTOTOOL))
 + r23*(TOOLEE_Z + transPosSel) + r24;
a31 = r31*cos(rollPosSel) + r32*sin(rollPosSel);
a32 = -r31*sin(rollPosSel) + r32*cos(rollPosSel);
a33 = r33;
a34s = r31*(cos(rollPosSel)*TOOLEE_X - sin(rollPosSel)*TOOLEE_Y + CENTERTOTOOL*cos(rollPosSel))
     + r32*(sin(rollPosSel)*TOOLEE_X + cos(rollPosSel)*TOOLEE_Y + CENTERTOTOOL*sin(rollPosSel))
 + r33*(TOOLEE_Z + transPosSel) + r34;
a34f = r31*(cos(rollPosSel)*TOOLEE_X - sin(rollPosSel)*(TOOLEE_Y-CENTERTOTOOL))
     + r32*(sin(rollPosSel)*TOOLEE_X + cos(rollPosSel)*(TOOLEE_Y-CENTERTOTOOL))
 + r33*(TOOLEE_Z + transPosSel) + r34;


ptr_EndPt = new End_point;          // create a new end point

ptr_EndPt->telop = 0;               // not a teleoperation
     ptr_EndPt->is_via = 0;              // this is a cut point, not a via point

ptr_EndPt->hold_tool = m_ToolRadio + 1;  // the appropriate tool for this cut point

ptr_EndPt->coord_x = a14s;
ptr_EndPt->coord_y = a24s;
ptr_EndPt->coord_z = a34s;

ptr_EndPt->roll = atan2(a21, a11);
ptr_EndPt->pitch = atan2(-a31, sqrt(a32*a32 + a33*a33));
ptr_EndPt->yaw = atan2(a32, a33);


     ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList,
                    (void *) ptr_EndPt);         // append this end point to the End Point List


// update the text (prompting words)

selectedCutPntNo++;        // update the cutting point number
sprintf(MsgStr, "Please select cutting point %i", selectedCutPntNo);
m_CutPtNo = (CString)MsgStr;
     UpdateData(FALSE);

// update the point coordinate information
sprintf(MsgStr1, "Point %i information:", selectedCutPntNo-1);
m_PointInfo = (CString)MsgStr1;
sprintf(MsgStr2, "%8.1lf", ptr_EndPt->coord_x);
m_xCoord = (CString)MsgStr2;
sprintf(MsgStr3, "%8.1lf", ptr_EndPt->coord_y);
m_yCoord = (CString)MsgStr3;
sprintf(MsgStr4, "%8.1lf", ptr_EndPt->coord_z);
m_zCoord = (CString)MsgStr4;
     UpdateData(FALSE);
}

void CCutSelect::OnPrevpntButton()        // the "< Previous Point" button is hitted
{
    // first destroy the last selected End_point

    delete ( (struct End_point *) ((( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_last
                                        ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList ))->val) );
                                      // destroy the End_point structure of the last node

( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_delete_node( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_last
                ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList ) );      // destroy the last node itself


// update the text (prompting words)

selectedCutPntNo--;        // update the cutting point number: select the deleted point again
sprintf(MsgStr, "Please select cutting point %i", selectedCutPntNo);
m_CutPtNo = (CString)MsgStr;
     UpdateData(FALSE);

// update the information of the last selected point

     if ( !( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_empty
               ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList ) )
                         // if the list is not empty--there is at least one cut point in the list
{
sprintf(MsgStr1, "Point %i information:", selectedCutPntNo-1);
m_PointInfo = (CString)MsgStr1;
sprintf(MsgStr2, "%8.1lf", ((struct End_point *)(( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_last
```

```
                                        ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList ) )->val))->coord_x );
m_xCoord = (CString)MsgStr2;
sprintf(MsgStr3, "%8.1lf", ((struct End_point *)(( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_last
                                        ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList ) )->val))->coord_y);
m_yCoord = (CString)MsgStr3;
sprintf(MsgStr4, "%8.1lf", ((struct End_point *)(( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_last
                                        ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).tempList ) )->val))->coord_z);
m_zCoord = (CString)MsgStr4;
     UpdateData(FALSE);
}
else                                // there is no cut point left -- so delete all the information exhibition
{
sprintf(MsgStr1, "");
sprintf(MsgStr2, "");
sprintf(MsgStr3, "");
sprintf(MsgStr4, "");
          m_PointInfo = (CString)MsgStr1;
          m_xCoord = (CString)MsgStr2;
          m_yCoord = (CString)MsgStr3;
m_zCoord = (CString)MsgStr4;
UpdateData(FALSE);

// disable the " < Previous point " button
GetDlgItem( IDC_PREVPNT_BUTTON )->EnableWindow( FALSE );
}

}
```

A.9 InsertTeleopDLG.h

```
#if !defined(AFX_INSERTTELEOPDLG_H__B64269D8_5116_11D3_8749_00C04FA35F00__INCLUDED_)
#define AFX_INSERTTELEOPDLG_H__B64269D8_5116_11D3_8749_00C04FA35F00__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// InsertTeleopDLG.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CInsertTeleopDLG dialog

class CInsertTeleopDLG : public CDialog
{
// Construction
public:
CInsertTeleopDLG(CWnd* pParent = NULL);   // standard constructor
int ViaPtInd;                             // the index number of the via point
int CutPtInd;                             // the index number of the cut point
void * ptr;                               // the pointer used to traverse the linked list
char * PrePntMessage;
char PtIndex[5];
struct End_point * pEndPt;          // pointer to the newly created end point

// Dialog Data
//{{AFX_DATA(CInsertTeleopDLG)
enum { IDD = IDD_INSERTTELEOP_DIALOG };
int m_preTeleop;
//}}AFX_DATA


// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CInsertTeleopDLG)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CInsertTeleopDLG)
afx_msg void OnCloseupInsertteleopCombo();
virtual BOOL OnInitDialog();
virtual void OnOK();
virtual void OnCancel();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```

```
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_INSERTTELEOPDLG_H__B64269D8_5116_11D3_8749_00C04FA35F00__INCLUDED_)
```

A.10 InsertTeleopDLG.cpp

```
// InsertTeleopDLG.cpp : implementation file
//

#include "stdafx.h"
#include "Rtsa2.h"
#include "InsertTeleopDLG.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CInsertTeleopDLG dialog


CInsertTeleopDLG::CInsertTeleopDLG(CWnd* pParent /*=NULL*/)
: CDialog(CInsertTeleopDLG::IDD, pParent)
{
//{{AFX_DATA_INIT(CInsertTeleopDLG)
m_preTeleop = -1;
//}}AFX_DATA_INIT
}

void CInsertTeleopDLG::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CInsertTeleopDLG)
DDX_CBIndex(pDX, IDC_INSERTTELEOP_COMBO, m_preTeleop);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CInsertTeleopDLG, CDialog)
//{{AFX_MSG_MAP(CInsertTeleopDLG)
ON_CBN_CLOSEUP(IDC_INSERTTELEOP_COMBO, OnCloseupInserrteleopCombo)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CInsertTeleopDLG message handlers

BOOL CInsertTeleopDLG::OnInitDialog()
{
CDialog::OnInitDialog();

ViaPtInd = 0;
CutPtInd = 0;

CComboBox* pPreTeleopBox;

for ( ptr = (void *) ( (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List)->flink);
          ptr != (void *) (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List);
       ptr = (void *) ( ((Dllist)ptr)->flink) )      // for loop to traverse the end point list
{
if ( ((struct End_point *)(((Dllist)ptr)->val))->telop )   // if it is teleoperation here
{
PrePntMessage = strdup("teleoperation");
pPreTeleopBox = (CComboBox*) GetDlgItem(IDC_INSERTTELEOP_COMBO);
pPreTeleopBox->AddString(PrePntMessage);
}
else if ( ((struct End_point *)(((Dllist)ptr)->val))->is_via )   // this is a via point
{
ViaPtInd++;
                PrePntMessage = strdup("via point ");
_itoa(ViaPtInd, PtIndex, 10);
strcat(PrePntMessage, PtIndex);
pPreTeleopBox = (CComboBox*) GetDlgItem(IDC_INSERTTELEOP_COMBO);
pPreTeleopBox->AddString(PrePntMessage);
}
```

```
else                                                    // this is a cut point
{
CutPtInd++;
PrePntMessage = strdup("cut point ");
_itoa(CutPtInd, PtIndex, 10);
strcat(PrePntMessage, PtIndex);
pPreTeleopBox = (CComboBox*) GetDlgItem(IDC_INSERTTELEOP_COMBO);
pPreTeleopBox->AddString(PrePntMessage);
}
}

return TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}

void CInsertTeleopDLG::OnCloseupInsertteleopCombo()
{
UpdateData(TRUE);

if(m_preTeleop != -1)
{
ptr = (void *) ( (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List)->flink);

for (int i=0; i<m_preTeleop; i++)
{
ptr = (void *) ( ((Dllist)ptr)->flink);
}
pEndPt = new End_point;           // create a new end point

           pEndPt->teleop = 1;               // this is a  teleoperation

     pEndPt->hold_tool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool;  // the expected tool keeps unchanged

( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_insert_a( (Dllist)ptr, (void *) pEndPt);
                                    // add this end point right after the specified node in the End Point List

}
}

void CInsertTeleopDLG::OnOK()      // the "Finish" button is clicked
{
((CRtsa2App*)AfxGetApp())->firstcheck = 1;    // will be the first time to check the new plan

CDialog::OnOK();
}

void CInsertTeleopDLG::OnCancel()
{
if(m_preTeleop != -1)
    ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_delete_node( ((Dllist)ptr)->flink );

CDialog::OnCancel();
}
```

A.11 ObjectType.h

```
#if !defined(AFX_OBJECTTYPE_H__D930D371_33D6_11D3_8723_00C04FA35F00__INCLUDED_)
#define AFX_OBJECTTYPE_H__D930D371_33D6_11D3_8723_00C04FA35F00__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ObjectType.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CObjectType dialog

class CObjectType : public CDialog
{
// Construction
public:
CObjectType(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
//{{AFX_DATA(CObjectType)
enum { IDD = IDD_SELECTOBJECTTYPE_DIALOG };
int m_cutObjt;
//}}AFX_DATA
```

```
// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CObjectType)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CObjectType)
virtual BOOL OnInitDialog();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_OBJECTTYPE_H__D930D371_33D6_11D3_8723_00C04FA35F00__INCLUDED_)
```

A.12 ObjectType.cpp

```
// ObjectType.cpp : implementation file
//

#include "stdafx.h"
#include "Rtsa2.h"
#include "ObjectType.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CObjectType dialog


CObjectType::CObjectType(CWnd* pParent /*=NULL*/)
: CDialog(CObjectType::IDD, pParent)
{
//{{AFX_DATA_INIT(CObjectType)
m_cutObjt = -1;
//}}AFX_DATA_INIT
}

void CObjectType::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CObjectType)
DDX_Radio(pDX, IDC_ASSEMB_RADIO, m_cutObjt);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CObjectType, CDialog)
//{{AFX_MSG_MAP(CObjectType)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CObjectType message handlers

BOOL CObjectType::OnInitDialog()
{
CDialog::OnInitDialog();

m_cutObjt = 0;          // set the default cut object as "assembly"
UpdateData(FALSE);

return TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}
```

A.13 ViaPntDlg.h

```
#if !defined(AFX_VIAPNTDLG_H__52E3C589_3AB2_11D3_8730_00C04FA35F00__INCLUDED_)
#define AFX_VIAPNTDLG_H__52E3C589_3AB2_11D3_8730_00C04FA35F00__INCLUDED_

// #include "EndPt_Action_Struct.h"     // added by XFU on 08/13/99

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ViaPntDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CViaPntDlg dialog

class CViaPntDlg : public CDialog
{
// Construction
public:
CViaPntDlg(CWnd* pParent = NULL);   // standard constructor
void * ptr;                         // used to traverse the linked list
char *PrePtMessage;
char PtIndex[5];
struct End_point * pEndPt;           // pointer to the newly created end point
char test[20];

int CutPtIndex;                     // the index of the cut point
int ViaPtIndex;                     // the index of the via point

// Dialog Data
//{{AFX_DATA(CViaPntDlg)
enum { IDD = IDD_VIAPNT_DIALOG };
int m_PreViaPt;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CViaPntDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CViaPntDlg)
virtual BOOL OnInitDialog();
afx_msg void OnCloseupViaptCombo();
virtual void OnOK();
virtual void OnCancel();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_VIAPNTDLG_H__52E3C589_3AB2_11D3_8730_00C04FA35F00__INCLUDED_)
```

A.14 ViaPntDlg.cpp

```
// ViaPntDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Rtsa2.h"
#include "ViaPntDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CViaPntDlg dialog
```

```
CViaPntDlg::CViaPntDlg(CWnd* pParent /*=NULL*/)
: CDialog(CViaPntDlg::IDD, pParent)
{
//{{AFX_DATA_INIT(CViaPntDlg)
m_PreViaPt = -1;
//}}AFX_DATA_INIT
}


void CViaPntDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CViaPntDlg)
DDX_CBIndex(pDX, IDC_VIAPT_COMBO, m_PreViaPt);
//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CViaPntDlg, CDialog)
//{{AFX_MSG_MAP(CViaPntDlg)
ON_CBN_CLOSEUP(IDC_VIAPT_COMBO, OnCloseupViaptCombo)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CViaPntDlg message handlers

BOOL CViaPntDlg::OnInitDialog()
{
CDialog::OnInitDialog();

ViaPtIndex = 0;
CutPtIndex = 0;

CComboBox* pPreViaPtBox;

for ( ptr = (void *) ( (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List)->flink);
         ptr != (void *) (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List);
       ptr = (void *) ( ((Dllist)ptr)->flink) )      // for loop to traverse the end point list
{


if ( ((struct End_point *)(((Dllist)ptr)->val))->telop )   // if it is teleoperation here
{
PrePtMessage = strdup("teleoperation");
pPreViaPtBox = (CComboBox*) GetDlgItem(IDC_VIAPT_COMBO);
pPreViaPtBox->AddString(PrePtMessage);
}
else if ( ((struct End_point *)(((Dllist)ptr)->val))->is_via )   // this is a via point
{
ViaPtIndex++;
          PrePtMessage = strdup("via point ");
_itoa(ViaPtIndex, PtIndex, 10);
strcat(PrePtMessage, PtIndex);
pPreViaPtBox = (CComboBox*) GetDlgItem(IDC_VIAPT_COMBO);
pPreViaPtBox->AddString(PrePtMessage);
}
else                                                       // this is a cut point
{
CutPtIndex++;
PrePtMessage = strdup("cut point ");
_itoa(CutPtIndex, PtIndex, 10);
strcat(PrePtMessage, PtIndex);
pPreViaPtBox = (CComboBox*) GetDlgItem(IDC_VIAPT_COMBO);
pPreViaPtBox->AddString(PrePtMessage);
}
}

return TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}

void CViaPntDlg::OnCloseupViaptCombo()
{
     UpdateData(TRUE);

if(m_PreViaPt != -1)
{

ptr = (void *) ( (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List)->flink);
```

```
for (int i=0; i<m_PreViaPt; i++)
{
ptr = (void *) ( ((Dllist)ptr)->flink);
}

pEndPt = new End_point;          // create a new end point

          pEndPt->telop = 0;              // not a teleoperation
          pEndPt->is_via = 1;             // this is a via point

     pEndPt->hold_tool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool; // the expected tool keeps unchanged

// dummy parameters, need to change in the future
    pEndPt->coord_x = 2;
          pEndPt->coord_y = 3;
          pEndPt->coord_z = 4;

          pEndPt->roll = 5;
    pEndPt->pitch = 6;
    pEndPt->yaw = 7;
//

    ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_insert_a( (Dllist)ptr, (void *) pEndPt);
                              // add this end point right after the specified node in the End Point List

}
}

void CViaPntDlg::OnOK()
{
((CRtsa2App*)AfxGetApp())->firstcheck = 1;    // will be the first time to check the new plan

CDialog::OnOK();
}

void CViaPntDlg::OnCancel()
{
if(m_PreViaPt != -1)
    ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_delete_node( ((Dllist)ptr)->flink );

CDialog::OnCancel();
}
```

A.15 ActDscDLG.h

```
#if !defined(AFX_ACTDSCDLG_H__EEF96489_5569_11D3_874D_00C04FA35F00__INCLUDED_)
#define AFX_ACTDSCDLG_H__EEF96489_5569_11D3_874D_00C04FA35F00__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ActDscDLG.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CActDscDLG dialog

class CActDscDLG : public CDialog
{
// Construction
public:
CActDscDLG(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
//{{AFX_DATA(CActDscDLG)
enum { IDD = IDD_ACTDSC_DIALOG };
CString m_act1;
CString m_act10;
CString m_act11;
CString m_act12;
CString m_act13;
CString m_act14;
CString m_act15;
CString m_act16;
CString m_act17;
CString m_act18;
CString m_act19;
```

```
CString m_act2;
CString m_act20;
CString m_act21;
CString m_act22;
CString m_act23;
CString m_act24;
CString m_act25;
CString m_act26;
CString m_act27;
CString m_act28;
CString m_act29;
CString m_act3;
CString m_act30;
CString m_act31;
CString m_act32;
CString m_act33;
CString m_act34;
CString m_act35;
CString m_act36;
CString m_act37;
CString m_act38;
CString m_act4;
CString m_act5;
CString m_act6;
CString m_act7;
CString m_act8;
CString m_act9;
//}}AFX_DATA


// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CActDscDLG)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CActDscDLG)
// NOTE: the ClassWizard will add member functions here
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_ACTDSCDLG_H__EEF96489_5569_11D3_874D_00C04FA35F00__INCLUDED_)
```

A.16 ActDscDLG.cpp

```
// ActDscDLG.cpp : implementation file
//

#include "stdafx.h"
#include "Rtsa2.h"
#include "ActDscDLG.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CActDscDLG dialog


CActDscDLG::CActDscDLG(CWnd* pParent /*=NULL*/)
: CDialog(CActDscDLG::IDD, pParent)
{
//{{AFX_DATA_INIT(CActDscDLG)
m_act1 = _T("");
m_act10 = _T("");
m_act11 = _T("");
```

```
m_act12 = _T("");
m_act13 = _T("");
m_act14 = _T("");
m_act15 = _T("");
m_act16 = _T("");
m_act17 = _T("");
m_act18 = _T("");
m_act19 = _T("");
m_act2 = _T("");
m_act20 = _T("");
m_act21 = _T("");
m_act22 = _T("");
m_act23 = _T("");
m_act24 = _T("");
m_act25 = _T("");
m_act26 = _T("");
m_act27 = _T("");
m_act28 = _T("");
m_act29 = _T("");
m_act3 = _T("");
m_act30 = _T("");
m_act31 = _T("");
m_act32 = _T("");
m_act33 = _T("");
m_act34 = _T("");
m_act35 = _T("");
m_act36 = _T("");
m_act37 = _T("");
m_act38 = _T("");
m_act4 = _T("");
m_act5 = _T("");
m_act6 = _T("");
m_act7 = _T("");
m_act8 = _T("");
m_act9 = _T("");
//}}AFX_DATA_INIT
}

void CActDscDLG::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CActDscDLG)
DDX_Text(pDX, IDC_EDIT1, m_act1);
DDX_Text(pDX, IDC_EDIT10, m_act10);
DDX_Text(pDX, IDC_EDIT11, m_act11);
DDX_Text(pDX, IDC_EDIT12, m_act12);
DDX_Text(pDX, IDC_EDIT13, m_act13);
DDX_Text(pDX, IDC_EDIT14, m_act14);
DDX_Text(pDX, IDC_EDIT15, m_act15);
DDX_Text(pDX, IDC_EDIT16, m_act16);
DDX_Text(pDX, IDC_EDIT17, m_act17);
DDX_Text(pDX, IDC_EDIT18, m_act18);
DDX_Text(pDX, IDC_EDIT19, m_act19);
DDX_Text(pDX, IDC_EDIT2, m_act2);
DDX_Text(pDX, IDC_EDIT20, m_act20);
DDX_Text(pDX, IDC_EDIT21, m_act21);
DDX_Text(pDX, IDC_EDIT22, m_act22);
DDX_Text(pDX, IDC_EDIT23, m_act23);
DDX_Text(pDX, IDC_EDIT24, m_act24);
DDX_Text(pDX, IDC_EDIT25, m_act25);
DDX_Text(pDX, IDC_EDIT26, m_act26);
DDX_Text(pDX, IDC_EDIT27, m_act27);
DDX_Text(pDX, IDC_EDIT28, m_act28);
DDX_Text(pDX, IDC_EDIT29, m_act29);
DDX_Text(pDX, IDC_EDIT3, m_act3);
DDX_Text(pDX, IDC_EDIT30, m_act30);
DDX_Text(pDX, IDC_EDIT31, m_act31);
DDX_Text(pDX, IDC_EDIT32, m_act32);
DDX_Text(pDX, IDC_EDIT33, m_act33);
DDX_Text(pDX, IDC_EDIT34, m_act34);
DDX_Text(pDX, IDC_EDIT35, m_act35);
DDX_Text(pDX, IDC_EDIT36, m_act36);
DDX_Text(pDX, IDC_EDIT37, m_act37);
DDX_Text(pDX, IDC_EDIT38, m_act38);
DDX_Text(pDX, IDC_EDIT4, m_act4);
DDX_Text(pDX, IDC_EDIT5, m_act5);
DDX_Text(pDX, IDC_EDIT6, m_act6);
DDX_Text(pDX, IDC_EDIT7, m_act7);
DDX_Text(pDX, IDC_EDIT8, m_act8);
DDX_Text(pDX, IDC_EDIT9, m_act9);
//}}AFX_DATA_MAP
```

```
}


BEGIN_MESSAGE_MAP(CActDscDLG, CDialog)
//{{AFX_MSG_MAP(CActDscDLG)
// NOTE: the ClassWizard will add message map macros here
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CActDscDLG message handlers
```

A.17 TaskPlan.h

```
#if !defined(AFX_TASKPLAN_H__D930D372_33D6_11D3_8723_00C04FA35F00__INCLUDED_)
#define AFX_TASKPLAN_H__D930D372_33D6_11D3_8723_00C04FA35F00__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// TaskPlan.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CTaskPlan dialog

class CTaskPlan : public CDialog
{
// Construction
public:
CTaskPlan(CWnd* pParent = NULL);   // standard constructor

struct action * pAct;              // pointer to an action structure
void * ptr;                        // used to traverse the linked list
Dllist Dptr;                       // used to traverse the action description list
Dllist ptAct;                       // pointer to traverse the action list
char * descPtr;                    // the pointer to the string which is used to describe the action
char *actPlan[38];

int    CutPtNo;                    // index to indicate the order(or sequence) number of the cut point
int    ViaPtNo;                    // index to indicate the order(or sequence) number of the via point
int    optIdx;                     // index of the primitive operation

char Buffer[20];                   // used for debug
char test[40];
char EndPtIndex[5];


// Dialog Data
//{{AFX_DATA(CTaskPlan)
enum { IDD = IDD_TASKPLAN_DIALOG };
// NOTE: the ClassWizard will add data members here
//}}AFX_DATA


// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CTaskPlan)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CTaskPlan)
afx_msg void OnObjectTypeBUTTON();
afx_msg void OnCutPointBUTTON();
afx_msg void OnCheckPlanBUTTON();
afx_msg void OnViaPointBUTTON();
afx_msg void OnInsertteleopButton();
afx_msg void OnDownloadTrajectoryBUTTON();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_TASKPLAN_H__D930D372_33D6_11D3_8723_00C04FA35F00__INCLUDED_)
```

A.18 TaskPlan.cpp

```
// TaskPlan.cpp : implementation file
//

#include "stdafx.h"
#include "Rtsa2.h"
#include "TaskPlan.h"
#include "ObjectType.h"
#include "ViaPntDlg.h"
#include "InsertTeleopDLG.h"
#include "toolRackPos.h"
#include "ActDscDLG.h"
#include "offset.h"
#include "math.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTaskPlan dialog


CTaskPlan::CTaskPlan(CWnd* pParent /*=NULL*/)
: CDialog(CTaskPlan::IDD, pParent)
{
//{{AFX_DATA_INIT(CTaskPlan)
// NOTE: the ClassWizard will add member initialization here
//}}AFX_DATA_INIT
}


void CTaskPlan::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CTaskPlan)
// NOTE: the ClassWizard will add DDX and DDV calls here
//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CTaskPlan, CDialog)
//{{AFX_MSG_MAP(CTaskPlan)
ON_BN_CLICKED(IDC_ObjectType_BUTTON, OnObjectTypeBUTTON)
ON_BN_CLICKED(IDC_CutPoint_BUTTON, OnCutPointBUTTON)
ON_BN_CLICKED(IDC_CheckPlan_BUTTON, OnCheckPlanBUTTON)
ON_BN_CLICKED(IDC_ViaPoint_BUTTON, OnViaPointBUTTON)
ON_BN_CLICKED(IDC_INSERTTELEOP_BUTTON, OnInsertteleopButton)
ON_BN_CLICKED(IDC_DownloadTrajectory_BUTTON, OnDownloadTrajectoryBUTTON)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTaskPlan message handlers

// XFU added on 07/05/99

void CTaskPlan::OnObjectTypeBUTTON()
{
CObjectType ObjtType;
int nObjtTypeResponse = ObjtType.DoModal();
if (nObjtTypeResponse == IDOK)
{

}
if (nObjtTypeResponse == IDCANCEL)
{

}
}

void CTaskPlan::OnCutPointBUTTON()
```

```
{
CCutSelect CPnt;
int nCutPntResponse = CPnt.DoModal();
if (nCutPntResponse == IDOK)
{}
}

void CTaskPlan::OnCheckPlanBUTTON()
{
((CRtsa2App*)AfxGetApp())->holdTool = ((CRtsa2App*)AfxGetApp())->RealholdTool;

      if (((CRtsa2App*)AfxGetApp())->firstcheck)
            // if this is the first time to check the plan, traverse the end point list to generate the action plan
      {
            ( ((CRtsa2App*)AfxGetApp())->ActList ).List = ( ((CRtsa2App*)AfxGetApp())->ActList ).new_dllist();
                        // create a new linked list: Action List
            ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List =
                              ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).new_dllist();
                                    // create a new linked list: Action Description List
   }

// first traverse the End Point List to create the Action List and the Action Description List

if ( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_empty( ( ((CRtsa2App*)AfxGetApp())->EndPtList ).List ) )
                                                      // if the end point list is empty
{
MessageBox("No EndPtList Right Now!!!");
}
else                                          // there IS an EndPtList
{
            if (((CRtsa2App*)AfxGetApp())->firstcheck)
             // if this is the first time to check the plan, traverse the end point list to generate the action plan
       {
         CutPtNo = 0;                                 // the index number of the cut point
         ViaPtNo = 0;                                 // the index number of the via point

         for ( ptr = (void *) ( (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List)->flink);
              ptr != (void *) (( ((CRtsa2App*)AfxGetApp())->EndPtList ).List);
       ptr = (void *) ( ((Dllist)ptr)->flink) )      // for loop to traverse the end point list
        {

 if ( ((struct End_point *)(((Dllist)ptr)->val))->telop )   // if it should be teleoperation here
 {
 // generate the action description:   teleoperation

 descPtr = strdup("Teleoperation");

 // append this action description to the action description list
 ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

 // generate the primitive operation: teleoperation

                  pAct = new action;

 pAct->teleop = 1;                              // this action is teleoperation
 pAct->holdTool = ((struct action *)(( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
               ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

 // append this action to the Action List
 ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

 }

 else        // it is not teleoperation
 {
   CutPtNo++;

   if ( ((CRtsa2App*)AfxGetApp())->holdTool == 0 )     // the robot has no tool in hand
   {
   // generate the action description:   move to the tool rack

   descPtr = strdup("Move to the tool rack");

   // append this action description to the action description list
   ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append
                                       ( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

   // generate the action description:    pick up the expected tool

   descPtr = strdup("Pick up ");
```

```
if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
    strcat(descPtr, "saw");
else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )// the expected tool is scissors
                strcat(descPtr, "scissors");
else                                                    // the expected tool is drill
strcat(descPtr, "drill");

// append this action description to the action description list
( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

// generate the primitive operation: move to the above point of the tool

                pAct = new action;

pAct->teleop = 0;                               // this action is not teleoperation
pAct->move = 1;                                 // this action is a movement action
pAct->gripAct = 0;                              // the gripper remains the original status
pAct->toolAct = 0;                              // the tool remains the original status

pAct->holdTool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool;

                if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
{
    pAct->fPtX = SAWABOVE_X;
    pAct->fPtY = SAWABOVE_Y;
pAct->fPtZ = SAWABOVE_Z;
    pAct->fPtPitch = SAWABOVE_PITCH;
    pAct->fPtYaw = SAWABOVE_YAW;
    pAct->fPtRoll = SAWABOVE_ROLL;
}
else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )// the expected tool is scissors
{
    pAct->fPtX = SCISSORSABOVE_X;
    pAct->fPtY = SCISSORSABOVE_Y;
pAct->fPtZ = SCISSORSABOVE_Z;
    pAct->fPtPitch = SCISSORSABOVE_PITCH;
    pAct->fPtYaw = SCISSORSABOVE_YAW;
    pAct->fPtRoll = SCISSORSABOVE_ROLL;
}
else     // the expected tool is drill
{
    pAct->fPtX = DRILLABOVE_X;
    pAct->fPtY = DRILLABOVE_Y;
pAct->fPtZ = DRILLABOVE_Z;
    pAct->fPtPitch = DRILLABOVE_PITCH;
    pAct->fPtYaw = DRILLABOVE_YAW;
    pAct->fPtRoll = DRILLABOVE_ROLL;
}


// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


// generate the primitive operation: open gripper

pAct = new action;

pAct->teleop = 0;                               // this action is not teleoperation
pAct->move = 0;                                 // this action is not a movement action
pAct->gripAct = 1;                              // open gripper
pAct->toolAct = 0;                              // the tool remains the original status

pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


// generate the primitive operation: move down to reach the tool
pAct = new action;

pAct->teleop = 0;                               // this action is not teleoperation
pAct->move = 1;                                 // this action is a movement action
pAct->gripAct = 0;                              // the gripper remains the original status
pAct->toolAct = 0;                              // the tool remains the original status

pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged
```

```
if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
{
    pAct->fPtX = SAW_X;
    pAct->fPtY = SAW_Y;
pAct->fPtZ = SAW_Z;
    pAct->fPtPitch = SAW_PITCH;
    pAct->fPtYaw = SAW_YAW;
    pAct->fPtRoll = SAW_ROLL;
}
else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )   // the expected tool is scissors
{
    pAct->fPtX = SCISSORS_X;
    pAct->fPtY = SCISSORS_Y;
pAct->fPtZ = SCISSORS_Z;
    pAct->fPtPitch = SCISSORS_PITCH;
    pAct->fPtYaw = SCISSORS_YAW;
    pAct->fPtRoll = SCISSORS_ROLL;
}
else     // the expected tool is drill
{
    pAct->fPtX = DRILL_X;
    pAct->fPtY = DRILL_Y;
pAct->fPtZ = DRILL_Z;
    pAct->fPtPitch = DRILL_PITCH;
    pAct->fPtYaw = DRILL_YAW;
    pAct->fPtRoll = DRILL_ROLL;
}


// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

// generate the primitive operation: close gripper

pAct = new action;

pAct->teleop = 0;                           // this action is not teleoperation
pAct->move = 0;                             // this action is not a movement action
pAct->gripAct = 2;                          // close gripper
pAct->toolAct = 0;                          // the tool remains the original status

pAct->holdTool = ((struct action *)(( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

// generate the primitive operation: move up back to the above point

pAct = new action;

pAct->teleop = 0;                           // this action is not teleoperation
pAct->move = 1;                             // this action is  a movement action
pAct->gripAct = 0;                          // the gripper remains the original status
pAct->toolAct = 0;                          // the tool remains the original status

pAct->holdTool = ((struct action *)(( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
{
    pAct->fPtX = SAWABOVE_X;
    pAct->fPtY = SAWABOVE_Y;
pAct->fPtZ = SAWABOVE_Z;
    pAct->fPtPitch = SAWABOVE_PITCH;
    pAct->fPtYaw = SAWABOVE_YAW;
    pAct->fPtRoll = SAWABOVE_ROLL;
}
else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )   // the expected tool is scissors
{
    pAct->fPtX = SCISSORSABOVE_X;
    pAct->fPtY = SCISSORSABOVE_Y;
pAct->fPtZ = SCISSORSABOVE_Z;
    pAct->fPtPitch = SCISSORSABOVE_PITCH;
    pAct->fPtYaw = SCISSORSABOVE_YAW;
    pAct->fPtRoll = SCISSORSABOVE_ROLL;
}
else     // the expected tool is drill
{
    pAct->fPtX = DRILLABOVE_X;
    pAct->fPtY = DRILLABOVE_Y;
```

```
pAct->fPtZ = DRILLABOVE_Z;
      pAct->fPtPitch = DRILLABOVE_PITCH;
      pAct->fPtYaw = DRILLABOVE_YAW;
      pAct->fPtRoll = DRILLABOVE_ROLL;
  }


  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

  // update the currently held tool
  ((CRtsa2App*)AfxGetApp())->holdTool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool;

  }

  else                                              // the robot has a tool in hand
  {
  if ( ((CRtsa2App*)AfxGetApp())->holdTool != ((struct End_point *)(((Dllist)ptr)->val))->hold_tool )
  // if the holded tool is not the expected tool
  {
      // generate the action description:   move to the tool rack

      descPtr = strdup("Move to the tool rack");

      // append this action description to the action description list
      ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

// generate the action description:    put back the currently holded tool

      descPtr = strdup("Put back ");

      if ( ((CRtsa2App*)AfxGetApp())->holdTool == 1 )         // the currently holded tool is saw
          strcat(descPtr, "saw");
      else if ( ((CRtsa2App*)AfxGetApp())->holdTool == 2 )    // the currently holded tool is scissors
                  strcat(descPtr, "scissors");
      else                                                 // the currently holded tool is drill
      strcat(descPtr, "drill");

      // append this action description to the action description list
      ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);


      // generate the action description:    pick up the expected tool

      descPtr = strdup("Pick up ");

      if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )         // the expected tool is saw
          strcat(descPtr, "saw");
      else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )    // the expected tool is scissors
                  strcat(descPtr, "scissors");
      else                                                  // the expected tool is drill
      strcat(descPtr, "drill");

      // append this action description to the action description list
      ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

  // generate the primitive operation: move to the above point of the currently held tool

                  pAct = new action;

  pAct->teleop = 0;                             // this action is not teleoperation
  pAct->move = 1;                               // this action is a movement action
  pAct->gripAct = 0;                            // the gripper remains the original status
  pAct->toolAct = 0;                            // the tool remains the original status

  pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

  if ( ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool == 1 )   // the currently held tool is saw
  {
      pAct->fPtX = SAWABOVE_X;
      pAct->fPtY = SAWABOVE_Y;
pAct->fPtZ = SAWABOVE_Z;
      pAct->fPtPitch = SAWABOVE_PITCH;
      pAct->fPtYaw = SAWABOVE_YAW;
      pAct->fPtRoll = SAWABOVE_ROLL;
  }
  else if ( ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool == 2 )
                                                        // the currently held tool is scissors
```

```
  {
      pAct->fPtX = SCISSORSABOVE_X;
      pAct->fPtY = SCISSORSABOVE_Y;
pAct->fPtZ = SCISSORSABOVE_Z;
      pAct->fPtPitch = SCISSORSABOVE_PITCH;
      pAct->fPtYaw = SCISSORSABOVE_YAW;
      pAct->fPtRoll = SCISSORSABOVE_ROLL;
  }
  else     // the currently held tool is drill
  {
      pAct->fPtX = DRILLABOVE_X;
      pAct->fPtY = DRILLABOVE_Y;
  pAct->fPtZ = DRILLABOVE_Z;
      pAct->fPtPitch = DRILLABOVE_PITCH;
      pAct->fPtYaw = DRILLABOVE_YAW;
      pAct->fPtRoll = DRILLABOVE_ROLL;
  }

  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


  // generate the primitive operation: move down to the original place of the tool
  pAct = new action;

  pAct->teleop = 0;                              // this action is not teleoperation
  pAct->move = 1;                                // this action is a movement action
  pAct->gripAct = 0;                             // the gripper remains the original status
  pAct->toolAct = 0;                             // the tool remains the original status

  pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                    ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

  if ( ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                          ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool == 1 )
                                                                    // the currently held tool is saw
  {
      pAct->fPtX = SAW_X;
      pAct->fPtY = SAW_Y;
pAct->fPtZ = SAW_Z;
      pAct->fPtPitch = SAW_PITCH;
      pAct->fPtYaw = SAW_YAW;
      pAct->fPtRoll = SAW_ROLL;
  }
  else if ( ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                         ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool == 2 )
// the currently held tool is scissors
  {
      pAct->fPtX = SCISSORS_X;
      pAct->fPtY = SCISSORS_Y;
pAct->fPtZ = SCISSORS_Z;
      pAct->fPtPitch = SCISSORS_PITCH;
      pAct->fPtYaw = SCISSORS_YAW;
      pAct->fPtRoll = SCISSORS_ROLL;
  }
  else     // the currently held tool is drill
  {
      pAct->fPtX = DRILL_X;
      pAct->fPtY = DRILL_Y;
pAct->fPtZ = DRILL_Z;
      pAct->fPtPitch = DRILL_PITCH;
      pAct->fPtYaw = DRILL_YAW;
      pAct->fPtRoll = DRILL_ROLL;
  }


  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


  // generate the primitive operation: open gripper

  pAct = new action;

  pAct->teleop = 0;                              // this action is not teleoperation
  pAct->move = 0;                                // this action is not a movement action
  pAct->gripAct = 1;                             // open gripper
  pAct->toolAct = 0;                             // the tool remains the original status

  pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                    ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged
```

```
  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

  // generate the primitive operation: move up back to the above point

  pAct = new action;

  pAct->teleop = 0;                          // this action is not teleoperation
  pAct->move = 1;                            // this action is  a movement action
  pAct->gripAct = 0;                         // the gripper remains the original status
  pAct->toolAct = 0;                         // the tool remains the original status

  pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                    ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;
                                                                  // the hold tool keeps unchanged


  if ( ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                     ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool == 1 )
 // the currently held tool is saw
  {
      pAct->fPtX = SAWABOVE_X;
      pAct->fPtY = SAWABOVE_Y;
pAct->fPtZ = SAWABOVE_Z;
      pAct->fPtPitch = SAWABOVE_PITCH;
      pAct->fPtYaw = SAWABOVE_YAW;
      pAct->fPtRoll = SAWABOVE_ROLL;
//       pAct->maxSpeed =
  }
  else if ( ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                     ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool == 2 )
              // the currently held tool is scissors
  {
      pAct->fPtX = SCISSORSABOVE_X;
      pAct->fPtY = SCISSORSABOVE_Y;
pAct->fPtZ = SCISSORSABOVE_Z;
      pAct->fPtPitch = SCISSORSABOVE_PITCH;
      pAct->fPtYaw = SCISSORSABOVE_YAW;
      pAct->fPtRoll = SCISSORSABOVE_ROLL;
//       pAct->maxSpeed =
  }
  else     // the currently held tool is drill
  {
      pAct->fPtX = DRILLABOVE_X;
      pAct->fPtY = DRILLABOVE_Y;
pAct->fPtZ = DRILLABOVE_Z;
      pAct->fPtPitch = DRILLABOVE_PITCH;
      pAct->fPtYaw = DRILLABOVE_YAW;
      pAct->fPtRoll = DRILLABOVE_ROLL;
//       pAct->maxSpeed =
  }

    // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


  // generate the primitive operation: move to the above point of the expected tool

                  pAct = new action;

  pAct->teleop = 0;                          // this action is not teleoperation
  pAct->move = 1;                            // this action is a movement action
  pAct->gripAct = 0;                         // the gripper remains the original status
  pAct->toolAct = 0;                         // the tool remains the original status

  pAct->holdTool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool;

  if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
  {
      pAct->fPtX = SAWABOVE_X;
      pAct->fPtY = SAWABOVE_Y;
pAct->fPtZ = SAWABOVE_Z;
      pAct->fPtPitch = SAWABOVE_PITCH;
      pAct->fPtYaw = SAWABOVE_YAW;
      pAct->fPtRoll = SAWABOVE_ROLL;
  }
  else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )   // the expected tool is scissors
  {
      pAct->fPtX = SCISSORSABOVE_X;
      pAct->fPtY = SCISSORSABOVE_Y;
```

```
pAct->fPtZ = SCISSORSABOVE_Z;
      pAct->fPtPitch = SCISSORSABOVE_PITCH;
      pAct->fPtYaw = SCISSORSABOVE_YAW;
      pAct->fPtRoll = SCISSORSABOVE_ROLL;
  }
  else     // the expected tool is drill
  {
      pAct->fPtX = DRILLABOVE_X;
      pAct->fPtY = DRILLABOVE_Y;
pAct->fPtZ = DRILLABOVE_Z;
      pAct->fPtPitch = DRILLABOVE_PITCH;
      pAct->fPtYaw = DRILLABOVE_YAW;
      pAct->fPtRoll = DRILLABOVE_ROLL;
  }


  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


  // generate the primitive operation: move down to reach the tool
  pAct = new action;

  pAct->teleop = 0;                              // this action is not teleoperation
  pAct->move = 1;                                // this action is a movement action
  pAct->gripAct = 0;                             // the gripper remains the original status
  pAct->toolAct = 0;                             // the tool remains the original status

  pAct->holdTool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool;

  if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
  {
      pAct->fPtX = SAW_X;
      pAct->fPtY = SAW_Y;
pAct->fPtZ = SAW_Z;
      pAct->fPtPitch = SAW_PITCH;
      pAct->fPtYaw = SAW_YAW;
      pAct->fPtRoll = SAW_ROLL;
  }
  else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )   // the expected tool is scissors
  {
      pAct->fPtX = SCISSORS_X;
      pAct->fPtY = SCISSORS_Y;
pAct->fPtZ = SCISSORS_Z;
      pAct->fPtPitch = SCISSORS_PITCH;
      pAct->fPtYaw = SCISSORS_YAW;
      pAct->fPtRoll = SCISSORS_ROLL;
  }
  else     // the expected tool is drill
  {
      pAct->fPtX = DRILL_X;
      pAct->fPtY = DRILL_Y;
pAct->fPtZ = DRILL_Z;
      pAct->fPtPitch = DRILL_PITCH;
      pAct->fPtYaw = DRILL_YAW;
      pAct->fPtRoll = DRILL_ROLL;
  }


  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


  // generate the primitive operation: close gripper

  pAct = new action;

  pAct->teleop = 0;                              // this action is not teleoperation
  pAct->move = 0;                                // this action is not a movement action
  pAct->gripAct = 2;                             // close gripper
  pAct->toolAct = 0;                             // the tool remains the original status

  pAct->holdTool = ((struct action *)(( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                    ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;
// the held tool keeps unchanged

  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

  // generate the primitive operation: move up back to the above point
```

```
  pAct = new action;

  pAct->teleop = 0;                              // this action is not teleoperation
  pAct->move = 1;                                // this action is  a movement action
  pAct->gripAct = 0;                             // the gripper remains the original status
  pAct->toolAct = 0;                             // the tool remains the original status

  pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                    ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;
// the held tool keeps unchanged

  if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 1 )   // the expected tool is saw
  {
      pAct->fPtX = SAWABOVE_X;
      pAct->fPtY = SAWABOVE_Y;
  pAct->fPtZ = SAWABOVE_Z;
      pAct->fPtPitch = SAWABOVE_PITCH;
      pAct->fPtYaw = SAWABOVE_YAW;
      pAct->fPtRoll = SAWABOVE_ROLL;
  }
  else if ( ((struct End_point *)(((Dllist)ptr)->val))->hold_tool == 2 )   // the expected tool is scissors
  {
      pAct->fPtX = SCISSORSABOVE_X;
      pAct->fPtY = SCISSORSABOVE_Y;
pAct->fPtZ = SCISSORSABOVE_Z;
      pAct->fPtPitch = SCISSORSABOVE_PITCH;
      pAct->fPtYaw = SCISSORSABOVE_YAW;
      pAct->fPtRoll = SCISSORSABOVE_ROLL;
  }
  else     // the expected tool is drill
  {
      pAct->fPtX = DRILLABOVE_X;
      pAct->fPtY = DRILLABOVE_Y;
pAct->fPtZ = DRILLABOVE_Z;
      pAct->fPtPitch = DRILLABOVE_PITCH;
      pAct->fPtYaw = DRILLABOVE_YAW;
      pAct->fPtRoll = DRILLABOVE_ROLL;
  }

  // append this action to the Action List
  ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

      // update the currently holded tool
      ((CRtsa2App*)AfxGetApp())->holdTool = ((struct End_point *)(((Dllist)ptr)->val))->hold_tool;

  }

  }


        if ( !(((struct End_point *)(((Dllist)ptr)->val))->is_via) )
                          // if this end point is not a via point -- that is to say, this point is a cut point
  {
// generate the action description:   move to the cut point #

      descPtr = strdup("Move to cut point ");
      _itoa( CutPtNo, EndPtIndex, 10);
                strcat(descPtr, EndPtIndex);

      // append this action description to the action description list
      ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

      // generate the primitive operation:  move to the cut point
  pAct = new action;

  pAct->teleop = 0;                              // this action is not teleoperation
  pAct->move = 1;                                // this action is  a movement action
  pAct->gripAct = 0;                             // the gripper remains the original status
  pAct->toolAct = 0;                             // the tool remains the original status

  pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                    ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

  pAct->fPtX = ((struct End_point *)(((Dllist)ptr)->val))->coord_x;
  pAct->fPtY = ((struct End_point *)(((Dllist)ptr)->val))->coord_y;
  pAct->fPtZ = ((struct End_point *)(((Dllist)ptr)->val))->coord_z;
  pAct->fPtPitch = ((struct End_point *)(((Dllist)ptr)->val))->pitch;
  pAct->fPtYaw = ((struct End_point *)(((Dllist)ptr)->val))->yaw;
  pAct->fPtRoll = ((struct End_point *)(((Dllist)ptr)->val))->roll;

  // append this action to the Action List
```

```
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

// generate the action description:  cut

descPtr = strdup("Cut ");

// append this action description to the action description list
     ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);

//  generate the primitive operation:   turn on the tool
pAct = new action;

pAct->teleop = 0;                              // this action is not teleoperation
pAct->move = 0;                                // this action is not a movement action
pAct->gripAct = 0;                             // the gripper remains the original status
pAct->toolAct = 1;                             // turn on the tool

pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

//  generate the primitive operation:   move to cut
pAct = new action;

pAct->teleop = 0;                              // this action is not teleoperation
pAct->move = 1;                                // this action is a movement action
pAct->gripAct = 0;                             // the gripper remains the original status
pAct->toolAct = 0;                             // the tool remains the original status

pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

pAct->fPtPitch = ((struct End_point *)(((Dllist)ptr)->val))->pitch;
pAct->fPtYaw = ((struct End_point *)(((Dllist)ptr)->val))->yaw;
pAct->fPtRoll = ((struct End_point *)(((Dllist)ptr)->val))->roll;
pAct->fPtX = ((struct End_point *)(((Dllist)ptr)->val))->coord_x - 2*CENTERTOTOOL*cos(pAct->fPtRoll)*cos(pAct->fPtPitch);
pAct->fPtY = ((struct End_point *)(((Dllist)ptr)->val))->coord_y - 2*CENTERTOTOOL*sin(pAct->fPtRoll)*cos(pAct->fPtPitch);
pAct->fPtZ = ((struct End_point *)(((Dllist)ptr)->val))->coord_z - 2*CENTERTOTOOL*(-sin(pAct->fPtPitch));

// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

//  generate the primitive operation:   turn off the tool
pAct = new action;

pAct->teleop = 0;                              // this action is not teleoperation
pAct->move = 0;                                // this action is not a movement action
pAct->gripAct = 0;                             // the gripper remains the original status
pAct->toolAct = 2;                             // turn off the tool

pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged


// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);


//  generate the primitive operation:   back off to the start point
pAct = new action;

pAct->teleop = 0;                              // this action is not teleoperation
pAct->move = 1;                                // this action is a movement action
pAct->gripAct = 0;                             // the gripper remains the original status
pAct->toolAct = 0;                             // the tool remains the original status

pAct->holdTool = ((struct action *)((( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                  ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

pAct->fPtX = ((struct End_point *)(((Dllist)ptr)->val))->coord_x;
pAct->fPtY = ((struct End_point *)(((Dllist)ptr)->val))->coord_y;
pAct->fPtZ = ((struct End_point *)(((Dllist)ptr)->val))->coord_z;
pAct->fPtPitch = ((struct End_point *)(((Dllist)ptr)->val))->pitch;
pAct->fPtYaw = ((struct End_point *)(((Dllist)ptr)->val))->yaw;
pAct->fPtRoll = ((struct End_point *)(((Dllist)ptr)->val))->roll;

// append this action to the Action List
( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);
```

```
   } // end of "if ( !(((struct End_point *)((Dllist)ptr)->val))->is_via) )" loop
   else  // this is a via point
   {
   CutPtNo--;    // since this end point is not a cut point
   ViaPtNo++;    // update the via point index number

   // generate the action description:   move to the via point #

        descPtr = strdup("Move to via point ");
       _itoa( ViaPtNo, EndPtIndex, 10);
                  strcat(descPtr, EndPtIndex);

       // append this action description to the action description list
       ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List, (void *) descPtr);


       // generate the primitive operation:  move to the via point
   pAct = new action;

   pAct->teleop = 0;                              // this action is not teleoperation
   pAct->move = 1;                                // this action is  a movement action
   pAct->gripAct = 0;                             // the gripper remains the original status
   pAct->toolAct = 0;                             // the tool remains the original status

   pAct->holdTool = ((struct action *)(( ((CRtsa2App*)AfxGetApp())->ActList ).dll_last
                     ( ( ((CRtsa2App*)AfxGetApp())->ActList ).List ))->val))->holdTool;  // the held tool keeps unchanged

   pAct->fPtX = ((struct End_point *)(((Dllist)ptr)->val))->coord_x;
   pAct->fPtY = ((struct End_point *)(((Dllist)ptr)->val))->coord_y;
   pAct->fPtZ = ((struct End_point *)(((Dllist)ptr)->val))->coord_z;
   pAct->fPtPitch = ((struct End_point *)(((Dllist)ptr)->val))->pitch;
   pAct->fPtYaw = ((struct End_point *)(((Dllist)ptr)->val))->yaw;
   pAct->fPtRoll = ((struct End_point *)(((Dllist)ptr)->val))->roll;

   // append this action to the Action List
   ( ((CRtsa2App*)AfxGetApp())->ActList ).dll_append( ( ((CRtsa2App*)AfxGetApp())->ActList ).List, (void *) pAct);

   }

        } // end of "else"


   }   // end of the for loop to traverse the end points

 }  // end of " if (((CRtsa2App*)AfxGetApp())->firstcheck) "

}

     ((CRtsa2App*)AfxGetApp())->firstcheck = 0;   // will not be the first time to check the plan next time

CActDscDLG ActDscDlg;

int actNum=0;        // to get the number of actions in the action description list
for ( Dptr =  ( (( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List)->flink);
         Dptr !=  (( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List);
         Dptr =  ( ((Dllist)Dptr)->flink) )      // for loop to traverse the action description list
{
actNum++;
}

for (int i=0; i<38; ++i)
{
actPlan[i] = strdup("");
}

int j=0;
for ( Dptr =  ( (( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List)->flink);
         Dptr !=  (( ((CRtsa2App*)AfxGetApp())->ActDscptList ).List);
         Dptr =  ( ((Dllist)Dptr)->flink) )      // for loop to traverse the action description list

{
      actPlan[j] = strdup( (char *)(Dptr->val) );
   ++j;
}

ActDscDlg.m_act1 = (CString)actPlan[0];
ActDscDlg.m_act2 = (CString)actPlan[1];
ActDscDlg.m_act3 = (CString)actPlan[2];
ActDscDlg.m_act4 = (CString)actPlan[3];
ActDscDlg.m_act5 = (CString)actPlan[4];
ActDscDlg.m_act6 = (CString)actPlan[5];
```

```
ActDscDlg.m_act7 = (CString)actPlan[6];
ActDscDlg.m_act8 = (CString)actPlan[7];
ActDscDlg.m_act9 = (CString)actPlan[8];
ActDscDlg.m_act10 = (CString)actPlan[9];
ActDscDlg.m_act11 = (CString)actPlan[10];
ActDscDlg.m_act12 = (CString)actPlan[11];
ActDscDlg.m_act13 = (CString)actPlan[12];
ActDscDlg.m_act14 = (CString)actPlan[13];
ActDscDlg.m_act15 = (CString)actPlan[14];
ActDscDlg.m_act16 = (CString)actPlan[15];
ActDscDlg.m_act17 = (CString)actPlan[16];
ActDscDlg.m_act18 = (CString)actPlan[17];
ActDscDlg.m_act19 = (CString)actPlan[18];
ActDscDlg.m_act20 = (CString)actPlan[19];
ActDscDlg.m_act21 = (CString)actPlan[20];
ActDscDlg.m_act22 = (CString)actPlan[21];
ActDscDlg.m_act23 = (CString)actPlan[22];
ActDscDlg.m_act24 = (CString)actPlan[23];
ActDscDlg.m_act25 = (CString)actPlan[24];
ActDscDlg.m_act26 = (CString)actPlan[25];
ActDscDlg.m_act27 = (CString)actPlan[26];
ActDscDlg.m_act28 = (CString)actPlan[27];
ActDscDlg.m_act29 = (CString)actPlan[28];
ActDscDlg.m_act30 = (CString)actPlan[29];
ActDscDlg.m_act31 = (CString)actPlan[30];
ActDscDlg.m_act32 = (CString)actPlan[31];
ActDscDlg.m_act33 = (CString)actPlan[32];
ActDscDlg.m_act34 = (CString)actPlan[33];
ActDscDlg.m_act35 = (CString)actPlan[34];
ActDscDlg.m_act36 = (CString)actPlan[35];
ActDscDlg.m_act37 = (CString)actPlan[36];
ActDscDlg.m_act38 = (CString)actPlan[37];

int nActDscDlgResponse = ActDscDlg.DoModal();

if ( nActDscDlgResponse == IDOK)
{

}
if ( nActDscDlgResponse == IDCANCEL)
{

}

}

void CTaskPlan::OnViaPointBUTTON()
{
// TODO: Add your control notification handler code here

     CViaPntDlg ViaPntDlg;
int nViaPntDlgResponse = ViaPntDlg.DoModal();
if (nViaPntDlgResponse == IDOK)
{

}
if (nViaPntDlgResponse == IDCANCEL)
{

}
}

void CTaskPlan::OnInserttelecpButton()
{
// TODO: Add your control notification handler code here

CInsertTelecpDLG TelecpDlg;
int nTelecpDlgResponse = TelecpDlg.DoModal();
if (nTelecpDlgResponse == IDOK)
{

}
if (nTelecpDlgResponse == IDCANCEL)
{

}
}

void CTaskPlan::OnDownloadTrajectoryBUTTON()
{
  // TODO: Add your control notification handler code here
```

```
  if ( ((CRtsa2App*)AfxGetApp())->firstcheck )
  {
  MessageBox("Please check the plan first");
  }
  else
  {

((CRtsa2App*)AfxGetApp())->RealholdTool = ((CRtsa2App*)AfxGetApp())->holdTool;   // update the held tool

FILE * PlanFile;
//PlanFile = fopen("d:/stacey/Projects/Rtsa2/stacey/Projects/Rtsa2/planfile", "w");
PlanFile = fopen("x:/TaskPlan/planfile", "w");

optIdx = 0;

for ( ptAct =  ( (( ((CRtsa2App*)AfxGetApp())->ActList ).List)->flink);
          ptAct !=  (( ((CRtsa2App*)AfxGetApp())->ActList ).List);
          ptAct =  ( ((Dllist)ptAct)->flink) )      // for loop to traverse the action list

{
        optIdx++;
  fprintf(PlanFile, "Operation %i:\n", optIdx);

  fprintf(PlanFile, "   teleop     =  ");
  fprintf(PlanFile, "%11d   ", ((struct action *)(ptAct->val))->teleop );
        if ( ((struct action *)(ptAct->val))->teleop == 1 )           // if the action is teleoperation
  fprintf(PlanFile, "# begin teleoperation\n");
  else
  fprintf(PlanFile, "# not teleoperation\n");

  fprintf(PlanFile, "   move       =  ");
  fprintf(PlanFile, "%11d   ", ((struct action *)(ptAct->val))->move );
  if ( ((struct action *)(ptAct->val))->move == 1 )           // if the action is a movement action
  fprintf(PlanFile, "# a movement operation\n");
  else
  fprintf(PlanFile, "# not a movement operation\n");

  fprintf(PlanFile, "   gripAct    =  ");
  fprintf(PlanFile, "%11d   ", ((struct action *)(ptAct->val))->gripAct );
        if ( ((struct action *)(ptAct->val))->gripAct == 0 )
  fprintf(PlanFile, "# the gripper remains the original status\n");
  else if ( ((struct action *)(ptAct->val))->gripAct == 1 )
  fprintf(PlanFile, "# open gripper\n");
  else
  fprintf(PlanFile, "# close gripper\n");

  fprintf(PlanFile, "   toolAct    =  ");
  fprintf(PlanFile, "%11d   ", ((struct action *)(ptAct->val))->toolAct );
        if ( ((struct action *)(ptAct->val))->toolAct == 0 )
  fprintf(PlanFile, "# the tool remains the original status\n");
  else if ( ((struct action *)(ptAct->val))->toolAct == 1 )
  fprintf(PlanFile, "# turn on tool\n");
  else
  fprintf(PlanFile, "# turn off tool\n");

  fprintf(PlanFile, "   fPtX       =  ");
  fprintf(PlanFile, "%11lf   ", ((struct action *)(ptAct->val))->fPtX );
  fprintf(PlanFile, "# the x coordinate of the final point\n" );

  fprintf(PlanFile, "   fPtY       =  ");
  fprintf(PlanFile, "%11lf   ", ((struct action *)(ptAct->val))->fPtY );
  fprintf(PlanFile, "# the y coordinate of the final point\n" );

  fprintf(PlanFile, "   fPtZ       =  ");
  fprintf(PlanFile, "%11lf   ", ((struct action *)(ptAct->val))->fPtZ );
  fprintf(PlanFile, "# the z coordinate of the final point\n" );

  fprintf(PlanFile, "   fPtPitch   =  ");
  fprintf(PlanFile, "%11lf   ", ((struct action *)(ptAct->val))->fPtPitch );
  fprintf(PlanFile, "# the pitch angle of the final point\n" );

  fprintf(PlanFile, "   fPtYaw     =  ");
  fprintf(PlanFile, "%11lf   ", ((struct action *)(ptAct->val))->fPtYaw );
  fprintf(PlanFile, "# the yaw angle of the final point\n" );

  fprintf(PlanFile, "   fPtRoll    =  ");
  fprintf(PlanFile, "%11lf   ", ((struct action *)(ptAct->val))->fPtRoll );
  fprintf(PlanFile, "# the roll angle of the final point\n" );

}
```

```
fclose(PlanFile);
  }
}
```

Vita

Xiaoquan Fu was born on March 9, 1972 in Changsha, Hunan, China. He was the only child of Songru Fu and Jianying Wen. He graduated with a Bachelor's Degree in Electrical Engineering from Hunan University in July, 1993. In August 1997 he came to the United States to begin his study and work in the Robotics and Electromechanical Systems Laboratory in the Mechanical Engineering Department at the University of Tennessee.

The author is continuing his graduate study in a new area in the Department of Electrical and Computer Engineering at the University of Tennessee. He enjoys everything full of competition and challenge.