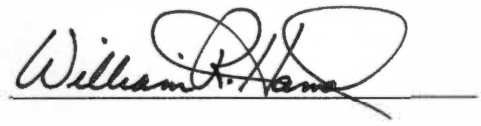
To the Graduate Council:

I am submitting herewith a thesis written by Shuo Yang entitled "Telerobotic Task Planner with Application to D&D Pipe Cutting". I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillments for the degree of Master of Science, with a major in Mechanical Engineering.



William R. Hamel, Major Professor

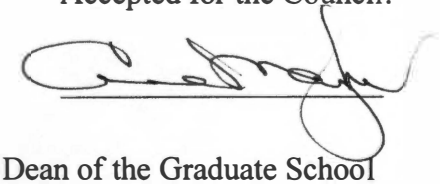
We have read this thesis

and recommend its acceptance:

Reid L. Kress

G. V. Smith

Accepted for the Council:



Vice Provost and Dean of the Graduate School

# Telerobotic Task Planner with Application to D&D Pipe Cutting

A Thesis
Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Shuo Yang

May 2002

# Acknowledgements

Many thanks to those who gave me kindly help…

I would like to thank my advisor, Dr. Bill Hamel who is the major advisor of this thesis, for his support and guidance. By participating in his research, I had the opportunity to choose this interesting topic as my thesis. I wish to express my deep appreciation to his patient direction and boundless energy, even during the holidays. I would like to thank Dr. Reid L. Kress, who gave me much help during my master study. His classes gave me the overall understanding of robotic system. I would like to thank Dr. G. V. Smith who served on my thesis committee and gave me valuable suggests. I would also like to thank Ge Zhang and Pamela Murray for their cooperation during the project.

At last but not least, I would like to thank my parents who have supported me many years.

# Abstract

Task planning is a key component necessary to implement telerobotic systems in remote operations for the deactivation and decommissioning (D&D) of defunct nuclear facilities.

In this thesis, a task planning software module for a telerobotic D&D system is presented. This software helps the operator to visually create a plan for a dismantlement task so that the control component can then interpret the task plan file and execute the plan. The graphical user interface allows the operator to plan a set of tasks efficiently. This software also calculates the manipulator's end-effector orientation and position at approach points, cut points and final points through appropriate kinematics transformations and saves the data to a task plan file.

With this software, the D&D telerobotic system works as an integrated system with real-time multi-mode operation. It is believed that this task planning concept will be applicable to other application domains.

# Table of Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| API | Application programming interface |
| D&D | Decontamination and dismantlement |
| ER&WM | Environment restoration and waste management |
| FSM | Finite state machine |
| GUI | Graphical user interface |
| HMCT | Human-machine cooperative telerobotics |
| MFC | Microsoft foundation classes |
| ORNL | Oak Ridge National Laboratory |
| REMSL | Robotics and Electromechanical System Laboratory |
| RTSA | Robot task space analyzer |
| TP | Task planner |
| TSSA | Task space scene analysis |

# Chapter 1

# Introduction

## 1.1 Background: Telerobotic systems in environmental restoration and waste management

Remote operations for the deactivation and decommissioning (D&D) of defunct nuclear facilities and associated Environmental restoration and waste management (ER&WM) is one of the most challenging applications of teleoperation in the United States. ER&WM challenges in the United States, and around the world, involve radiation and other hazards that are not suitable to human workers' direct contact and operation. It therefore necessitates the use of remote operations to protect human workers from dangerous exposures. The tasks of D&D systems include mainly removing radioactive contaminants from equipment and taking it apart for recycle or disposal. Teleoperated systems are used to perform remote maintenance functions in hazardous environments usually with the fundamental objective of reducing or eliminating human worker exposure to dangers. The challenge results from the unstructured, uncertain and radioactive environment that makes the work inefficient, expensive and slow. Teleoperation not only requires ten to one hundred times the

execution time, but also can result in reduced work quality outputs compared to that of direct contact work by humans. These factors highlight the importance of telerobotics which holds real promise for improving work quality and efficiency by combining teleoperation and robotic automation through computer vision, modeling, control, planning and trajectory generation.

A telerobotic system has been developed at the Robotics and Electromechanical Systems Laboratory at the University of Tennessee in support of the D&D robotic work at the Oak Ridge National Laboratory and the National Energy Technology Laboratory. This system, implementing the remote operation with the combinations of teleoperation and robotic control telerobotic work systems, outperforms traditionally available remote equipment [1]. The remote operation is divided into several subtasks in different operating modes. This scheme is demonstrated in figure 1.1.

## 1.2 Robots Classifications in Terms of Operating Mode

Autonomous robots function with no human modification of their activities after their programming is complete, until maintenance or re-programming is necessary. "Teleoperators are robotic system that synergistically combine human and machine" [10]. While teleoperators are not robots they are robotic systems. They are a kind of robotic device that can remotely control a material handling machine. Because teleoperators take advantage of human creativity and intelligence, they are able to

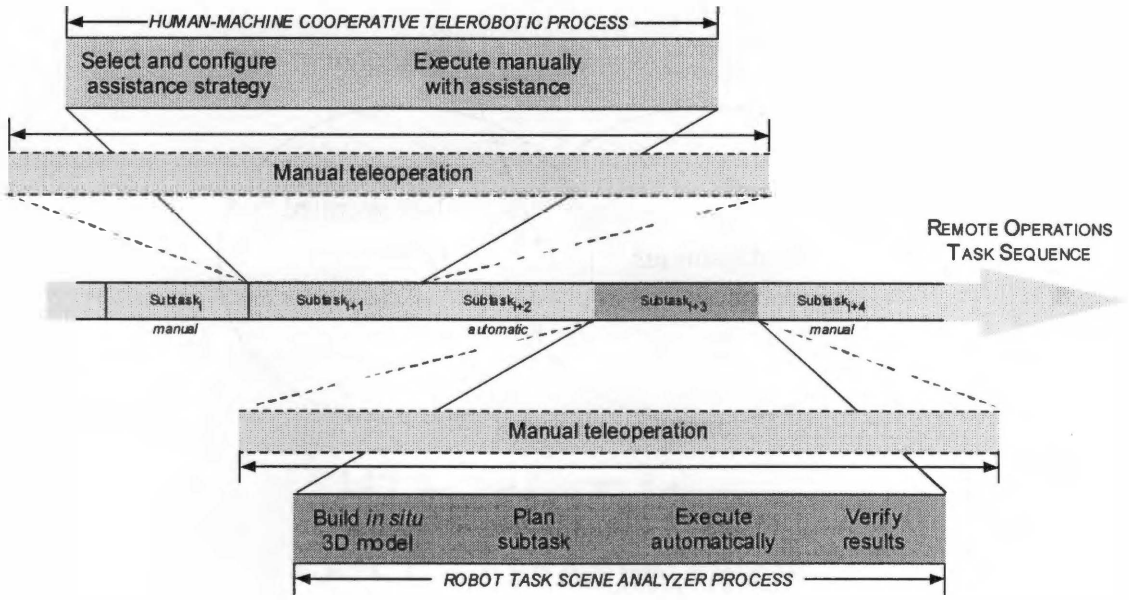



Figure 1.1 D&D Telerobotics system operations cycle [1]

respond to an environment more flexibly and develop new behaviors as required. Teleoperated robots incorporate human inputs as required by the level of control designed into the teleoperated robot. The key difference is that for autonomous robots, human-robot interaction is not routinely required, while for teleoperated robots human-robot interaction is routinely required. Because teleoperators take advantage of human cognitive and perceptual abilities, they can perform more efficiently in unfamiliar and dynamic environments. A typical robot is usually a programmable, automated material handling machine. Teleoperators are a human-machine system, which executes a programmer's stored commands and acts on inputs from a human on-line in real time.

Based on the above concepts, the relationship between autonomous robots and teleoperators is show in figure 1.2

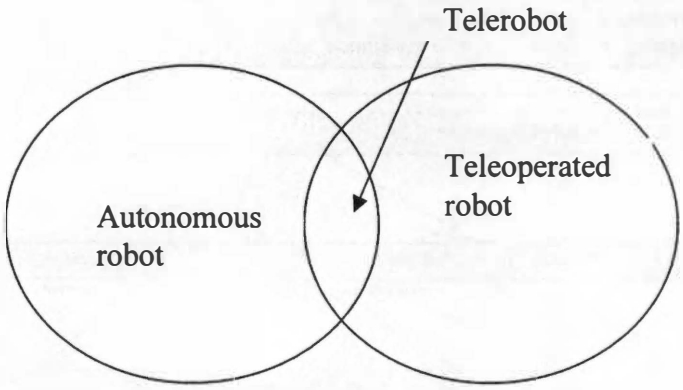



Figure 1.2 Classifications of robotics

## 1.3 Overview of Robot Task Scene Analyzer (RTSA), Task Planer (TP) and Real-time Controller

From the operation cycle shown in figure 1.1, each operation consists of three basis cycles: (1) building 3-D model of work space, (2) planning task and (3) executing planned task.

The first cycle is called task space scene analysis (TSSA). It refers to the process by which the remote work system gathers geometrical and other types of information that are necessary to characterize and analyze the automated task execution. To automate or provide computer assistance to a task or subtask with a telerobotic system, the operator needs to obtain the necessary quantitative 3-D geometrical data about the task space. For example, in a dismantlement scenario the task may be to remove a segment of process piping using remote manipulators and

cutting tools. If such a task is to be automated, it is necessary to describe the location and orientation of each piping element with respect to the remote work system. The robot task scene analyzer (RTSA) is a subsystem that implements TSSA functionality in the D&D task telerobotic system. The RTSA system acquires the geometric information such as position and orientation of a part in the task space and builds a 3-D model of the task location [1].

The second cycle is called task planning. It refers to the process by which the software generates all the necessary information for the real-time control subsystem during the operation from model information obtained in TSSA subsystem. It provides all the necessary information of an overall task. For example, the desired task may include two subtasks, cutting a specific pipe with a saw in autonomous mode and manipulating the pipe in assisted teleoperation mode. The task planning software needs to compute the position and orientation of the manipulator during the autonomous subtask, and set the tool parameters for the autonomous subtask; then specify the operating mode for the teleoperation mode and compute the geometrical constraint useful to the assist function. The task planner (TP) is such a software module that implements task planning functionality in the D&D task telerobotic system.

The third and last cycle is called real-time executing. It refers to the process by which the real-time control subsystem executes the task following the planned task. The real-time control can be categorized into autonomous control and teleoperation.

Execution in autonomous mode requires no operator's on-line commands, and the control module solely completes the execution with the information in the planned task. In most cases, the remote task cannot be performed completely autonomously due to the complicated nature of the task and uncertainty in the environment model which requires that the human operator always maintain direct control. But maneuvering a manipulator is difficult in teleoperation tasks in which contact must be made with the environment by a tool. This difficulty tends to make the human operator experience significant fatigue during long work periods. Fatigue then decreases the efficiency and performance of the human operator and task execution time consequently increases. There is therefore a need to improve the system maneuverability and efficiency in teleoperation by exploiting the autonomous ability of robot. Therefore, a semi-autonomous controller, or human machine cooperative controller, can be used to enhance instead of supersede the human command thereby increasing operator efficiency. The HMCT software module in a D&D telerobot improves the operator efficiency in teleoperation modes.

### 1.4 Task Planning Research Review

Task planning is initially an important research area in manufacturing or other fields where robotic automation has significant applications. With the boom of artificial intelligent research, a lot of work has been done in developing planner for assembly planning [2, 3, and 4].

As the application of telerobotics in D&D systems is so important, lots of research has been carried out all over the United State and the world [5, 6, 7, and 8].

It has been widely accepted that task planning is a key component in the telerobotic system as D. Verna presented the scenario of the teleoperation and virtual reality that "in Virtual Execution, consider the case of path or task planning, in semi-autonomous robotics." [9]

The prototype of our system is mentioned in the technical document in the Automation and Intelligent Systems Focus Group at Lawrence Livermore National Lab, "Telerobotics expertise includes shared control with seamless transfer between teleoperation and autonomous robot control which provides for manual control of one time operations and programmable control for repetitive operations." This system include "task planning, user interface, information base management..." [10].

Typically the subsystems of task planning defines a Euclidean frame for the task description, as Dr. Martin Jagersand in the Computer Science Department at Rochester University presented in his dissertation "task specification and planning are done in a global Euclidean world coordinate frame, and both cameras and robots are calibrated in this frame, ...open loop manipulations described in any of the three frames, namely visual, (local) world or joint, are stored in the *canned motor actions* module. These are typically used to bring about some immediate goal, such as insertion, screwing in, dropping off etc" [11]. Based on this idea, our task planning results are expressed in a fixed Cartesian frame (Robot frame).

The researches in planning based purely on mechanical and geometrical information provide the computation foundation of our task planning. Matthew T. Mason [12] presented a brief overview of this kind of manipulation planning, in which he discussed the planning approach for different mechanical parts.

There is some research on the task planning in the application of teleoperated system. For example, in the research "blind grasping" by D.M. Lane and M. Pickett in 1998 [13], "Geometric and semantic models of the world are used as triggers into pre-stored lists of action sequences". The task planning software in this thesis uses geometrical information from models.

When designing the task planing software in a D&D remote operation system, human-machine interface issues must be fully and in detail considered. The friendly human-machine interface provides more reliable remote operation, which is very important in the D&D applications. For example, in the design of the Man Machine Interface system (MMI) in Consiglio Nazionale delle Ricerche [14], the aim was "build an efficient, general purpose and reliable architecture… one of the most important MMI tasks is to understand a human request, to correctly interpret it and translate it into a series of commands which can be implemented by the controlled machine." This issue is also the first concern in our design of the TP.

## 1.5 Thesis Goal: Task Planner —— a bridge between RTSA and Real-Time Controller

In this thesis, task planning software, called the task planner (TP), for a telerobotic (D&D) system is presented. The TP creates the task description file that gives all necessary information to control real-time execution subsystem through a series of calculation with geometric information from RTSA subsystem so that the high level controller in the real-time execution subsystem then interprets the task plan file and activates the proper assist functions during the teleoperation. Figure 1.2 describes the basic data processing and transmission relationship between the individual modules. With the TP, the D&D telerobotic system works as an integrated system with real-time multi-mode operation.

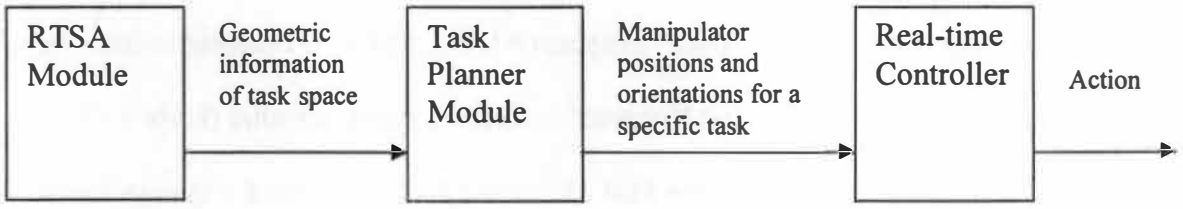



Figure 1.2 Information flow in D&D system

# Chapter 2

# System Configuration and Previous Work

## 2.1 System Configuration in D&D Telerobotic System

### 2.1.1 Modules in D&D System

A telerobotic system as discussed in this thesis consists of the three main modules: Robot Task Space Analyzer (RTSA), real-time execution controller which includes Human Machine Cooperative Telerobotics Controller (HMCT) for teleoperation modes and the Task Planner (TP). The RTSA is a human interactive system that uses sensors, image processing, and a multitude of software modules to obtain necessary geometric information. Real-time controller (including HMCT) is functionally a set of control mechanisms. In the autonomous operation mode, the real-time controller interprets the task plan file directly, transforming the commands, which are listed in the task plan file, from the Cartesian space to joint space and sends it to the low-level controller. In assisted teleoperation, HMCT takes control of the

real-time execution module. The assist functions modify the commands to limit the action of the end-effector according to different assistant modes, which are linear assist, planar assist and force assist, and thereby satisfy the predefined constraints on the end-effector. The HMCT consists of a finite state machine (FSM), interpreter, and assist functions. The FSM manages the operating mode, and the interpreter activates the functional blocks that correspond to the operating mode.

The TP is a computation module that generates control commands necessary for autonomous execution to the real-time control module. The task plan file describes the execution of task including manipulator and tooling motions.

The TP is an interactive graphical user interface (GUI) system, and the operator is only required to define the end-effector and tooling positions in the 3-D model from RTSA. Detailed data, like the required position and orientation of the end-effector or the manipulating procedure, are automatically generated by the TP. As we can see in figure 2.1, TP plays the key role to link the information between RTSA and real-time controller.

### 2.2.2 Hardware and Software Schematics

For better understanding of the D&D system, some hardware schematics are presented. The present hardware resides in the Robotics and Electromechanical System Laboratory (REMSL) at the University of Tennessee. The current hardware and interconnections are shown in figure 2.2. The set of hardware in use may be

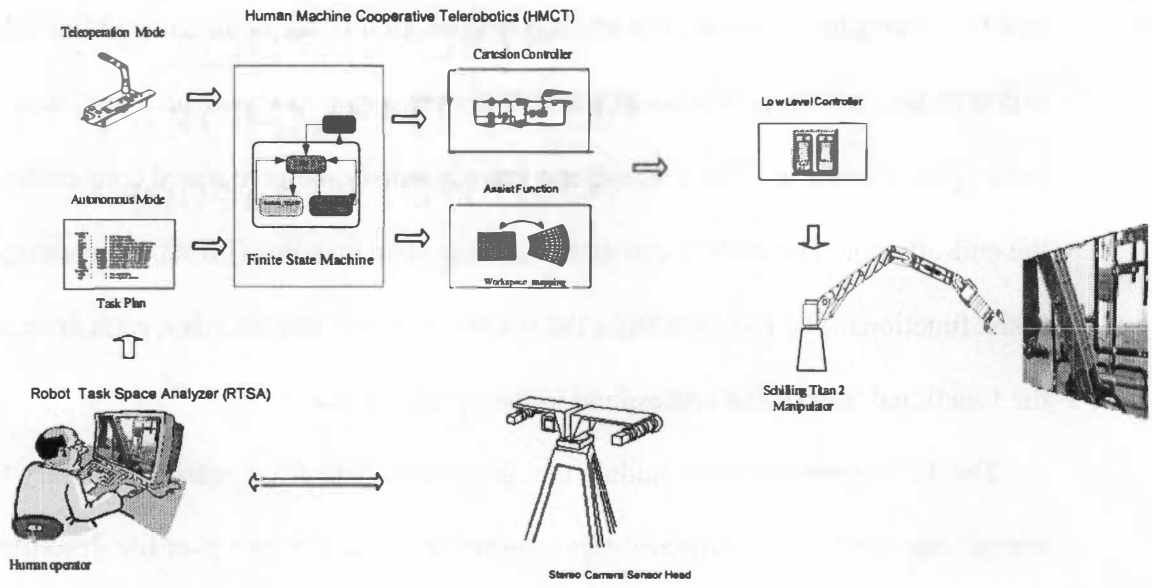



Figure 2.1 Module Configuration of the telerobotic system for D&D [15]

decomposed into a real-time part, including the robot and its peripherals and controller; and non-real-time part, including the RTSA model builder and task planner and sensor head connections [1].

The corresponding software schematic is shown in figure 2.3. It is a detailed software interpretation of figure 2.1, showing the data dependencies and data flow in the D&D telerobotic system in REMSL.

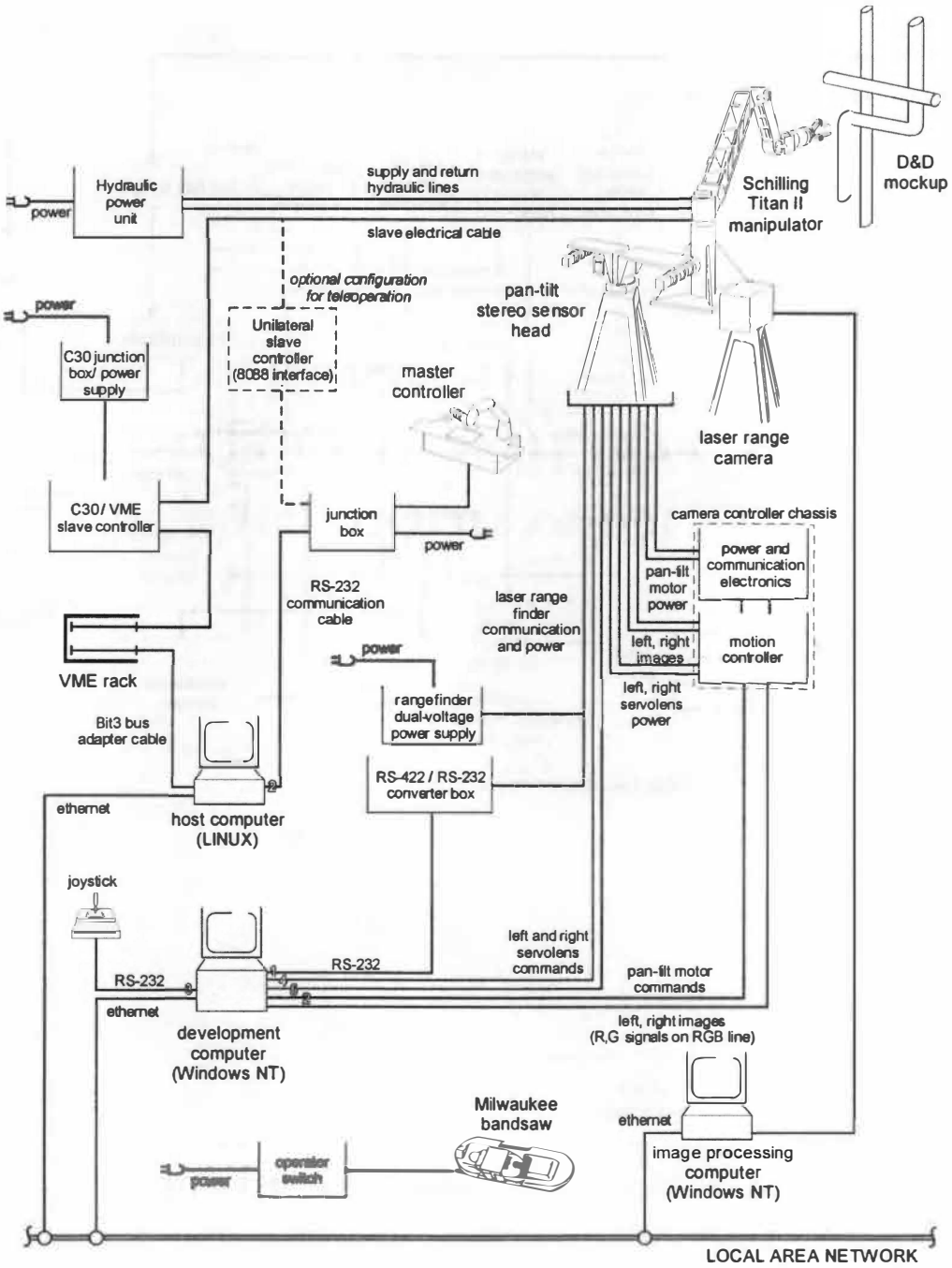



Figure 2.2 System hardware schematic in REMS Lab [1]

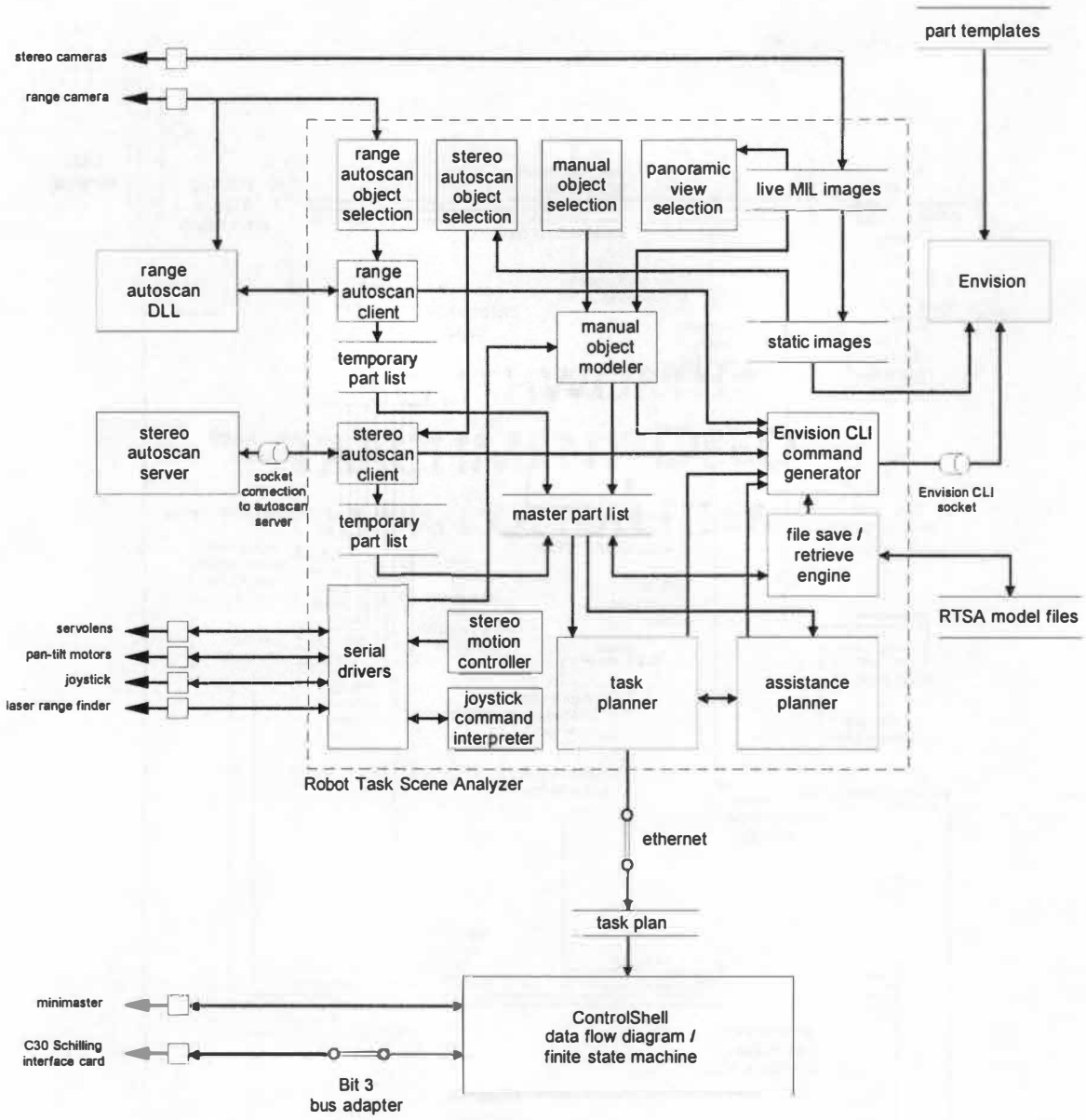



Figure 2.3 System software schematic in REMS Lab [1]

## 2.3 Previous Work and New Features of Work in This Thesis

The three main software modules, RTSA, real-time controller (including HMCT) and TP, in this telerobotic system have been developed in the Robotics and Electromechanical System Lab (REMSL) at the University of Tennessee. During the past three years, the RTSA and real-time controller modules were completed and they have performed satisfactorily. As mentioned in Chapter 1, these two software modules can not work without the information bridge, the task planner.

The original TP module developed by Xiaoquan Fu [16] didn't work effectively. In his work, a double linked list was created with C++ to record the sequence of operation. This data structure is suited to record the operation sequence as a double linked list is easy to implement insertion and deletion. The old task planner divided the operation into atomic tasks. But the key problem of his work was its inability to give correct calculation of the necessary parameters in the autonomous mode. He left the work to be completed by the following programmer. Another problem is that its graphic user interface took little consideration of human-machine interface issue. The main interface in his task planner is show in figure 2.4. An inexperienced operator has difficulty to figure out the sequence of processes when he is planning a task.

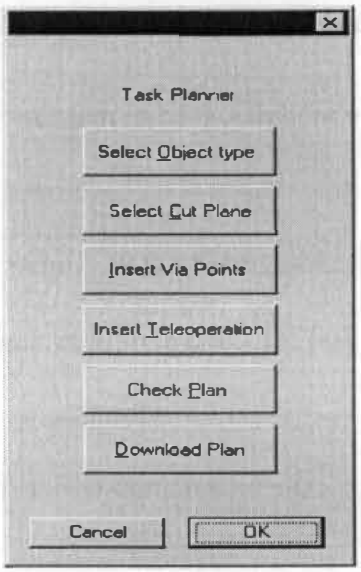



Figure 2.4 Main window in old Task planner [16]

The old task planner could not implement the "download plan" functionality which is very important. As the real-time controller is located on a server and the task planner is located on a client which is connected to the server with Ethernet, the task plan file needs to be downloaded and saved onto the server.

As the HMCT control module was developed after his work, the old task planner had not considered the teleoperation mode. A new task planner with the functionality to output different sets of necessary data according to different control schemes, such as schemes in autonomous mode and linear assist mode, was needed.

The old task planner functionalities and features are listed in table 2.1

Table 2.1 Functionalities and features in old task planner

| **Functionalities and features** | **Comments** |
|---|---|
| Data structure of a double linked list | Valuable work for task planning |
| Atomic task planning | Valuable work for task planning |
| Download functionality | Need to be added in the new task planner |
| Interface feature | Need to revise to an interface that requires less operating experience |

With the software developed in this thesis, the TP is able to plan both autonomous and assisted teleoperation modes. It requires the operator to input his desire step by step, and even a new operator can then make a task plan by just following the sequence of the pop up windows. At last, it realizes the download functionality through Ethernet. A table of comparision between the functionalities of new task planner developed in this thesis and the old task planner is shown in table 2.2

Table 2.2 Functionalities and features in the new task planner

| **Functionalities and features** | **Comments** |
|---|---|
| Data structure to record atomic task | Same as the old one |
| Data computation in autonomous mode | New |
| Interface of the planning process | New, requires operator little familiarity to the system |
| Ability to plan teleoperation along with autonomous operation | New |
| Data computation in teleoperation mode | New |
| Tool parameter setting | New |

# Chapter 3

## Task Planner Concept and Software Structure

### 3.1 Task Planner for a D&D Telerobotic System

According to the concept presented in figure 1.1, an autonomous subtask needs the task planner to provide control information. In the assisted teleoperation modes, the task planner provides some necessary geometrical information to assist the operator. For example, considering the task of using a telerobotic system to accomplish simple cutting, the human operator has the goal of picking up a saw with the manipulator, moving the manipulator from the rack where it picks up the saw to the location where it begins to get ready for cutting and then cutting the part with a desired feedrate in autonomous mode. This is a typical autonomous subtask sequence to be addressed by the task planner in the system.

TP is a software module responsible for creating a task plan from RTSA model information. It provides the operator with a set of windows to select the operating mode, set tool parameters, select the object to be cut, save the task plan file onto the

real-time controller server, etc. It calculates the manipulator position and orientation for a desired autonomous subtask; it gives the geometrical constraints for assisted teleoperation.

TP of this D&D telerobotic remote work system needs to provide the operator with a very flexible working environment. In this concern, the TP module must provide operators a very friendly graphical user interface environment.

### 3.2 Task Planner Implementation

All functionalities, such as computation of position and orientation, mentioned above are implemented in the task planner software presented in this thesis. The TP module is created to provide assistance to the operator in formulating a plan for dismantlement of a selected subtask operation. The task planner allows the operator to choose parts that are to be removed, tools in use, methods of assistance, and subtask automation. With the task planner, the operator is able to plan the operation in advance and to check the plan before execution.

The task planner software gives the operator a friendly graphic user interface environment so that the desired input is obtained by completing a sequence of straightforward selections. It also allows the operator to make reasonable changes during the task planning.

As the basic functionality of the task planner is to compute and then provide the real-time controller necessary information for subtask execution, the most important requirement of the task planner is to compute the necessary coordination and

orientation parameters with respect to different operating modes so that the real-time controller can work well under the corresponding mode and save the task plan information in a format interpretable to the real-time control subsystem. Thus the real-time control subsystem can complete the desired tasks of the operator according to the task plan file. The following section presents different information needed by the real-time controller in the various operating modes.

## 3.3 Task Planning with Telerobot Operating Modes

There are three categories of operating modes in the D&D telerobot: autonomous, teleoperation with assist function and pure teleoperation.

As a task may include a sequence of operations in different modes, the task planner must be able to implement the computation of different parameters in the corresponding mode. The task planner provides a task plan file which is read by the real-time control subsystem, and this task plan file need to set the flags to the FSM in HMCT module so as to specify the corresponding operating mode in an individual operation. The required manipulator geometrical information in different modes is listed in table 3.1.

All operating modes require a state variable to trigger the FSM in the real-time controller.

### 3.3.1 Autonomous mode

In the autonomous mode, the operator does not intercept any information from the system and does not input any command during the operation. In the applications that

Table 3.1 Required geometrical information in different mode

| **Operating Mode** | **Required manipulator geometrical information** |
|---|---|
| Autonomous | 3-D position and orientation of cut point, approach point and final point |
| Teleoperation with Linear Assist Function | 3-D position and orientation at two different point in the constrain line |
| Teleoperation with Planar Assist Function | 3-D coordinates and orientations at any three points not aligned in the same line within the constraint plane. |
| Teleoperation with Velocity Assist Function | 3-D coordinates and orientations at the cutting point |
| Pure Teleoperation | No geometrical information is needed |

the workspace is highly constructed, this operating mode gives better performance, such as precision and quality of workmanship, than man-driven operation, and also gives higher operation efficiency.

In the autonomous mode, the manipulator obtains the approach point where it get ready to cut, and then move to the cut point where it touches the part, and then feeds at a set feedrate to a final point where the manipulator retreats.

The real-time controller module needs the following data to implement an operation under the autonomous mode.

1. A mode trigger to specify the autonomous operating mode.

2. 3-D coordinates and orientations of the manipulator at cutting point described with respect to the robot frame
3. 3-D coordinates and orientations of the manipulator at approach point described with respect to the robot frame
4. 3-D coordinates and orientations of the manipulator at final point described with respect to the robot frame

### 3.3.2 Assisted teleoperation modes

The other operating modes are associated with teleoperation. In many applications, the unstructured nature of workspace and limitations of the current sensor and computer decision-making technologies prohibit the use of completely autonomous modes. In these systems operators provide an input (control signal) through a manual controller which is an integral part of the system control loop. Teleoperation in the HMCT D&D system is classified into two large categories: teleoperation with assist function and pure teleoperation.

The remote work efficiency will be greatly improved by enhancing the nature of teleoperation by providing a computer-assist control function, which allows the operator to complete the teleoperation task faster and more efficiently. For example, it may be desirable to remove a set of bolts with teleoperation, but provide assistance in maintaining alignment with the bolts during approach. This assistance is then defined when the task plan is formulated and downloaded with the plan file.

The assist functions interact with the task planner and allow motion constraints to be implemented during teleoperation. These functions are useful during tooling tasks such as maintaining alignment with bolts during removal. They are defined in the task plan and implemented in HMCT module.

Teleoperation with assist function can be further categorized into teleoperation with position assist function and teleoperation with velocity assist function. These concepts and strategies proposed by Everett [17] are based on the philosophy that system parameters rather than direct commands from the operator should be altered on-line.

#### 3.3.2.1 Teleoperation with Linear Assist Function

In the Linear Assist Function mode, the motion of the manipulator is constrained along a given line. In this mode, the HMCT module ensures linear movement of the manipulator along a specified line. One significant application of this mode is in operations such as drilling holes at an arbitrary orientation. The Linear Assist Function needs the following data:

1. a mode trigger to specify the Teleoperation with Linear Assist mode
2. 3-D coordinates and orientations of the manipulator at any point in the constraint line with respect to the robot frame
3. 3-D coordinates and orientations of the manipulator at another point in the constraint line with respect to the robot frame

#### 3.3.2.2 Teleoperation with Planar Assist Function

In the Planar Assist Function, the motion of the manipulator is constrained within a plane, in which the tooling operation is desired. In this mode, the HMCT module ensures that the manipulator's velocity perpendicular to the tooling direction is zero. This function ensures that the track of the manipulator is constrained within the constraint plane. This result is obtained by defining the velocity mapping as a matrix which scales the master's velocity and the velocity which is commanded to the slave. In the appropriate frame of reference, the element in the desired direction is multiplied by one and all the rest are cancelled by multiplying those elements by zero [17]. The plane in which cutting is desired is used as an input. This assistance function ensures that the tool is constrained within the desired plane by multiplying velocity row which are not in the desired direction by zero. [18]

By simple geometry, any three non-co-linear points uniquely define a plane. The Planar Assist Function needs the following data:

1. a mode trigger to specify the Teleoperation with Planar Assist mode
2. 3-D coordinates and orientations of the manipulator at any three points not aligned in the same line within the constraint plane

#### 3.3.2.3 Teleoperation with Velocity Assist Function

In the Velocity Assist Function mode, there is a mapping between the master and slave velocities. It is useful in the approach and avoidance of objects in the workspace [18].

Velocity Assist Function needs the following data:

1. a mode trigger to specify the Teleoperation with Velocity Assist mode.
2. 3-D coordinates and orientations of the manipulator at the cutting point with respect to the robot frame
3. a desired velocity of tooling

### 3.3.3 Pure Teleoperation

Pure teleoperation is routinely used in remote operations. The task planner is set up to send a flag to inform the HMCT module to revert a particular subtask to complete manual control through pure teleoperation. In this mode, the operator issues the control the manipulator directly. Pure Teleoperation mode requires:

1. set a trigger to HMCT controller so as to transfer the complete control to the operator through the manual controller.

# Chapter 4

# Transformations and Computations

## 4.1 Coordinate Systems

A manipulator can be schematically represented from a mechanical viewpoint as a kinematic chain of rigid bodies connected by means of joints. One end of the chain is constrained to a base, while the end-effector is mounted to the other end. The resulting motion of the structure is obtained by composite motion of each link with respect to the previous one. Operational space is a space that "position of the end-effector can be given by a minimal number of coordinates with regard to the geometry of the structure, and orientation can be specified in terms of minimal representation (Euler angles) describing the rotation of the end-effector with respect to the bas frame" [19]. Therefore, in order to manipulate an object in operational space, it is necessary to describe the end-effector position and orientation.

### 4.1.1 Space Description of a Rigid Body—Position and Orientation

A rigid body is completely described in space by its position and orientation with respect to a reference coordinate frame. The position of a rigid body in space is expressed in terms of the position of a suitable point on the body with respect to a reference frame (translation), while its orientation is expressed in terms of the components of the unit vectors of a frame attached to the body—with origin in the above point—with respect to the same reference frame (rotation). All the frames in this thesis are orthogonal frames. As shown in figure 4.1, the position of a point O_ on the rigid body with respect to the coordinate frame O-xyz, in which x, y, z are the unit vectors of the frame axes, is expected by the relation

$$o' = o'_x\, x + o'_y\, y + o'_z\, z$$

where $o'_x, o'_y, o'_z$ denote the components of the vector $o'$ along the frame axes; the position of $o'$ can be compactly written as (3x1) vector

$$o' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix}$$

In order to describe the rigid body orientation, it is convenient to consider an orthogonal frame attached to the body and express its unit vectors with respect to the reference frame. Let $O'\text{-}x'y'z'$ be such a frame with origin in $O'$ and $x', y', z'$ be the unit vectors of the frame axes. These vectors are expressed with respect to the reference frame O-xyz by Eq (4.1):

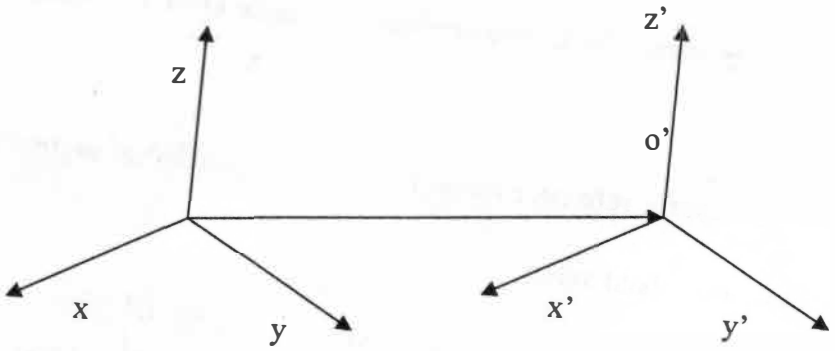



Figure 4.1 Position and orientation of a rigid body

$$\begin{cases} x' = x'_x\, x + x'_y\, y + x'_z\, z \\ y' = y'_x\, x + y'_y\, y + z'_z\, z \\ z' = z'_x\, x + z'_y\, y + z'_z\, z \end{cases} \quad \text{---- (4.1)}$$

### 4.1.2 Rotation and Roll-Pitch-Yaw Angles

With the above relationships, the components of each unit vector are the direction cosines of the axes of frame $O'$–$x'\,y'\,z'$ with respect to the reference frame O-xyz.

From above equation, we get a matrix R, which is

$$R = \begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} \text{--- (4.2)}$$

here R is called the rotation matrix. It's an orthogonal matrix in that $R^T R = I$, so $R^T = R^{-1}$.

Actually, a new frame can be obtained via a sequence of elementary rotations from the reference frame. The rotation matrices of frame $O'-x'y'z'$ with respect to frame O-xyz are

(1) suppose that the reference frame $O-xyz$ is rotated by an angle counter-clockwise $\alpha$ about axis z,

then the rotation matrix is $R_z(\alpha)=\begin{bmatrix}\cos\alpha & -\sin\alpha & 0\\ \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix}$ --- (4.3)

(2) suppose that the reference frame $O-xyz$ is rotated by an counter-clockwise angle $\beta$ about axis y,

then the rotation matrix is $R_y(\beta)=\begin{bmatrix}\cos\beta & 0 & \sin\beta\\ 0 & 1 & 0\\ -\sin\beta & 0 & \cos\beta\end{bmatrix}$ ---- (4.4)

(3) suppose that the reference frame $O-xyz$ is rotated by an counter-clockwise angle $\gamma$ about axis x,

then the rotation matrix is $R_x(\gamma)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\gamma & -\sin\gamma\\ 0 & \sin\gamma & \cos\gamma\end{bmatrix}$ --- (4.5)

If the new frame rotates successively about axes of current frames, i.e., each rotation is defined with respect to the preceding one, composition of successive rotations is then obtained by post-multiplication of the rotation matrices following the order of rotations. If a new frame rotates successively about the axes of the initial frame, i.e., each rotation is defined with respect to the fixed frame, composition of

successive rotations is then obtained by pre-multiplication of the rotation matrices following the order of rotations.

Roll-pitch-yaw-angles are very useful representations of orientation in the robotics and nautical fields. It decomposes any rotation of a new frame from a reference frame into three elementary rotations as follows:

(1) Rotate the reference frame by the angle $\psi$ about axis x (yaw); this rotation is described by the matrix $R_x(\psi)$ which is formally defined in (4.5)

(2) Rotate the reference frame by the angle $\vartheta$ about axis y (pitch); this rotation is described by the matrix $R_y(\vartheta)$ which is formally defined in (4.4)

(3) Rotate the reference frame by the angle $\varphi$ about axis z (roll); this rotation is described by the matrix $R_z(\varphi)$ which is formally defined in (4.3)

The resulting frame is obtained by composition of rotations with respect to the fixed frame, so the overall rotation matrix is pre-multiplication of the matrices of the elementary rotation. According to the rule given above, the resulting rotation matrix is

$$R(\phi) = R_z(\varphi)R_y(\vartheta)R_x(\psi)$$
$$= \begin{bmatrix} c\varphi * c\vartheta & c\varphi * s\vartheta * s\psi - s\varphi * c\psi & c\varphi * s\vartheta * c\psi + s\varphi * s\psi \\ s\varphi * c\vartheta & s\varphi * s\vartheta * s\psi - c\varphi * c\psi & s\varphi * s\vartheta * c\psi - c\varphi * s\psi \\ -s\vartheta & c\varphi * s\psi & c\varphi * c\psi \end{bmatrix} \text{---- (4.6)}$$

In Eq. (4.6), $c\varphi$ =cos( $\varphi$ ) and $s\varphi$ =sin( $\varphi$ ); the other notations have similar meaning.

The inverse solution to a given rotation matrix is $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ in the roll-pitch-yaw representation.

The solution for $\varphi$ in the range $(-\pi/2, \pi/2)$ is

$$\begin{cases} \varphi = A\tan 2(r_{21}, r_{11}) \\ \vartheta = A\tan 2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \psi = A\tan 2(r_{32}, r_{33}) \end{cases} \text{--- (4.7)}$$

There is another solution for $\varphi$ in the range $(\pi/2, 3\pi/2)$ [19]. In this thesis, we take the solution of (4.7).

### 4.1.3 Homogeneous Transformation

Homogeneous coordinates were initially introduced by Forest in 1969 into computer graphics to overcome a number of problems involved in matrix calculations. Homogeneous coordinates can be viewed as the addition of an extra coordinate to each vector. In robotics, we are normally only interested in rotation, and translation transformations, as we are normally dealing with solid physical objects. Therefore extended from R3 space, homogeneous transformation deals with 4 dimensional transformations.

In robotics, we are normally interested in the expression of objects in three-dimensional space. Therefore the general, 4x4, three-dimensional transformation matrix has the following form [20]:

$$T = \begin{bmatrix} \begin{matrix} rotation \\ part \\ R \end{matrix} & \vdots & \begin{matrix} traslation \\ part \end{matrix} \\ \cdots\cdots & + & \cdots\cdots \\ 0 & \vdots & 1 \end{bmatrix} = \begin{bmatrix} 3\times 3 & \vdots & 3\times 1 \\ \cdots\cdots & + & \cdots\cdots \\ 1\times 3 & \vdots & 1\times 1 \end{bmatrix} \quad \text{--- (4.8)}$$

In this homogenous coordinate system, it is necessary to make the meaning of notations clear for further discussion:

${}^{R}T_{N}$ is the transformation matrix between a new frame and original reference frame. Under this notation,

$${}^{R}p = {}^{R}T_{N}\,{}^{N}p \quad \text{---- (4.9)}$$

Analogous to the rule of matrices, a sequence of coordinate transformations with successive rotation about axes of current frames is composed by the post-multiplication.

$${}^{n}p = {}^{1}A_{0}\,{}^{2}A_{1} \cdots {}^{n}A_{n-1}\,{}^{0}p \quad \text{--- (4.10)}$$

## 4.2 Coordinate Frames Definition Used in D&D Telerobotic System

Figure 4.2 gives the scene of the task space in our D&D system application.

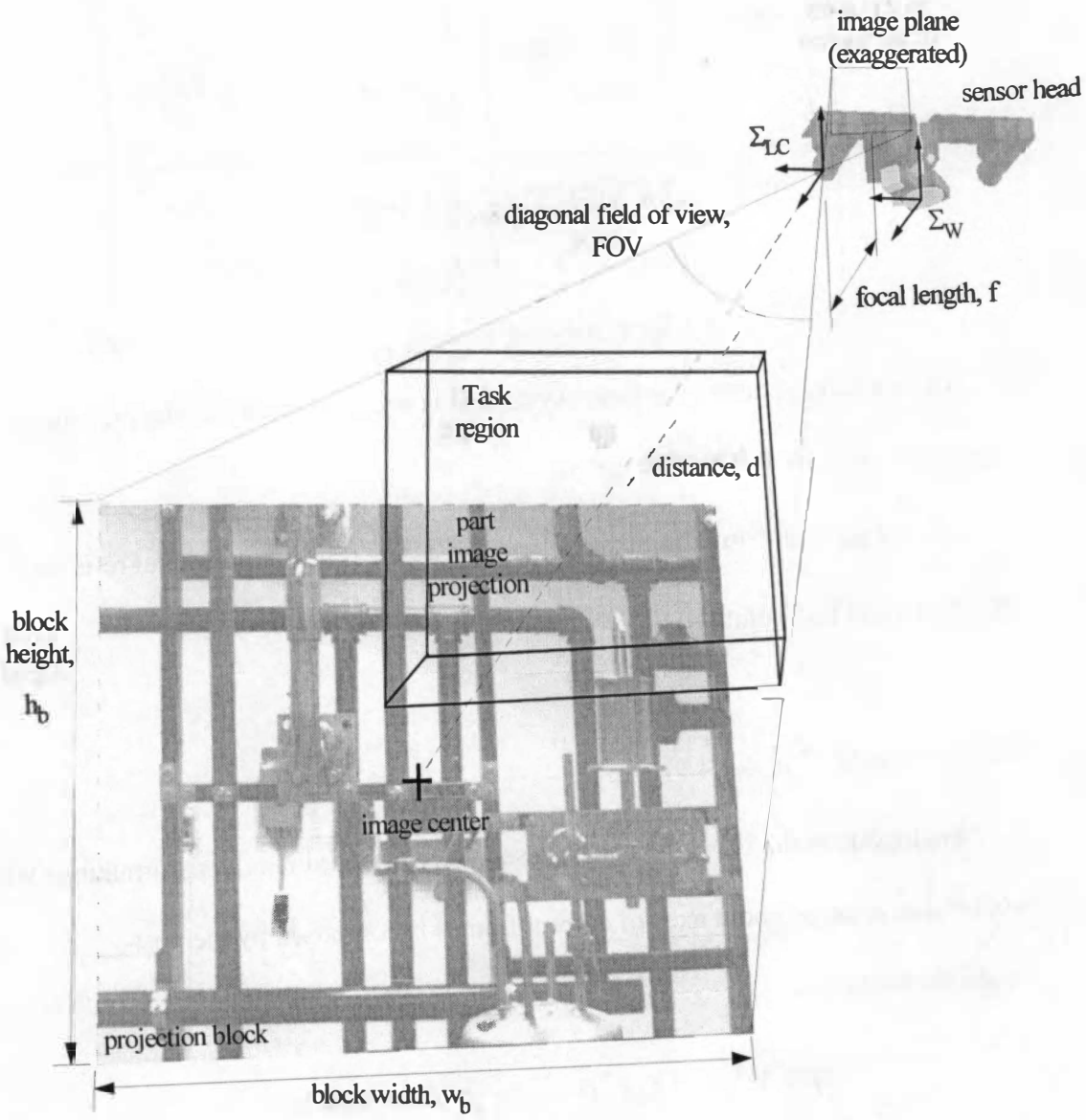



Figure 4.2 Layout of task space [1]

### 4.2.1 Sensor Head Coordinate Frames

This section is mainly from reference [1]. As it is important to understand the overall frame transformation and data flow, the computation and definition in RTSA is presented briefly here.

Figure 4.3 shows the coordinate frames fixed on the pan-tilt sensor head. In the present configuration shown in figure 4.3, the pan and tilt angles both read zero. The following sections will describe the transformation between these frames.

#### 4.2.1.1 World frame ($\Sigma_W$)

The world frame is to set the general reference of the system. All the transformations and computations are related to this frame ultimately. This frame is fixed on the underside of the pan motor assembly with its z-axis along the pan axis.

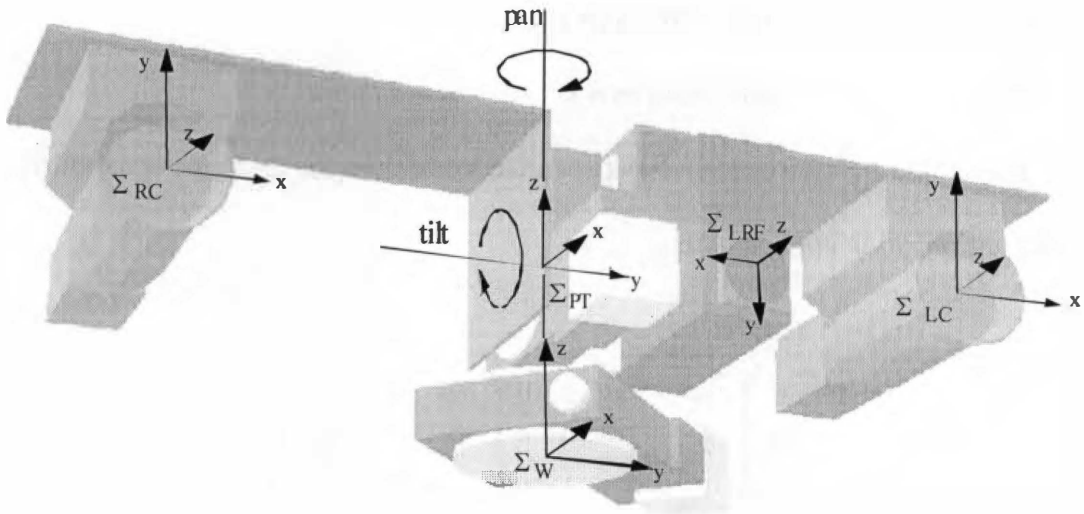



Figure 4.3 Sensor head coordinate frames [1]

The x-axis is attached such that it points in the same direction as the cameras when the pan motor assembly is mounted on the tripod, at $-135^{o}$ from the encoder, and the pan angle dial reads $0^{o}$.

#### 4.2.1.2 Pan-tilt frame ($\Sigma_{PT}$)

This frame's origin is placed at the intersection of the pan and tilt axes, and it is oriented identically to the world frame when the pan and tilt angles are at their zero positions. It is fixed with respect to the cameras, and thus its position is determined by the offset from the world frame and the pan and tilt angles. We can imagine that the Pan-tilt frame is obtained from the world frame by three steps sequentially with respect to the current frame:

(1) translate by an offset Zw along z-axis

(2) rotate by a pan angle about axis z

(3) rotate by a tilt angle about axis y

Therefore, the transformation between the world and pan-tilt frame is as follows, postmultiplication applied here:

$$
{}^{W}T_{PT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & Z_W \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\varphi & -s\varphi & 0 & 0 \\ s\varphi & c\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\vartheta & 0 & s\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ -s\vartheta & 0 & c\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$= \begin{bmatrix} c\varphi\, c\vartheta & -s\varphi & c\varphi\, s\vartheta & 0 \\ s\varphi\, c\vartheta & c\varphi & s\varphi\, s\vartheta & 0 \\ -s\vartheta & 0 & c\vartheta & z_w \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{--- (4.11)}$$

where

$z_W$ = offset

$\varphi$ = pan angle

$\vartheta$ = tilt angle

For our specific system setup in REMSL in this thesis, the offset is 108.3mm

#### 4.2.1.3 Laser range finder frame ($\Sigma_{LRF}$)

The laser range finder is a sensor to measures the distance from a laser spot to its surface. All coordinate information is expressed initially in this frame. As the laser range finder rotates about pan axis and tilt axis, $\Sigma_{LRF}$ is a kinetic frame. So the geometric information with respect to $\Sigma_{LRF}$ must be expressed in a fixed frame before further processing. This transformation will be discussed in the next section. Here we first discuss the transformation between frame LRF and frame PT.

The LRF is fixed relative to the PT frame on the surface of the laser range finder where the distance reads zero. Although they should be the same orientation, small errors in the mounting of the laser range finder on the sensor head require there to be a fixed transformation between the two. If there is a pan and tilt angle to get from the

PT frame to the LRF frame, we get a similar transformation as that from the W to PT frame, along with the transformation for the rearrangement of axis directions. The pan and tilt angle offset values for the mounting of the laser range finder are calibrated experimentally by adjusting the numbers until parts are being placed against the background screen correctly.

$$
{}^{PT}T_{LRF} = \begin{bmatrix} c\varphi_{LRF}\, c\vartheta_{LRF} & -s\varphi_{LRF} & c\varphi_{LRF}\, s\vartheta_{LRF} & {}^{PT}x_{LRF} \\ s\varphi_{LRF}\, c\vartheta_{LRF} & c\varphi_{LRF} & s\varphi_{LRF}\, s\vartheta_{LRF} & {}^{PT}y_{LRF} \\ -s\vartheta_{LRF} & 0 & c\vartheta_{LRF} & {}^{PT}z_{LRF} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} s\varphi_{LRF} & -c\varphi_{LRF}\, s\vartheta_{LRF} & c\varphi_{LRF}\, c\vartheta_{LRF} & {}^{PT}x_{LRF} \\ -c\varphi_{LRF} & -s\varphi_{LRF}\, s\vartheta_{LRF} & s\varphi_{LRF}\, c\vartheta_{LRF} & {}^{PT}y_{LRF} \\ 0 & -c\vartheta_{LRF} & -s\vartheta_{LRF} & {}^{PT}z_{LRF} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{--- (4.12)}
$$

where for our specific system setup in REMSL,

${}^{PT}x_{LRF}$ = 75 mm

${}^{PT}y_{LRF}$ = 92 mm

${}^{PT}z_{LRF}$ = -11 mm

$\varphi_{LRF}$ = 0.019 deg

$\vartheta_{LRF}$ = -0.01 deg

$\Sigma_{LC}$ and $\Sigma_{RC}$ and two frames defined in RTSA, which are not useful to the computation in this thesis[1].

#### 4.2.1.4 Calculations in Sensor Head Frames

As mentioned above, all original geometric information is with respect to the Laser range finder frame, which is not a fixed frame. Information needs to be expressed with respect to a fixed frame –world frame. The transformation matrix from the world frame to the laser range finder frame may be calculated by multiplying the transformations from world to pan-tilt and from pan-tilt to laser frames. The world frame coordinates of the laser spot may then be determined by transforming the spot coordinate in the laser frame with this result. This procedure is executed as follows:

$$^{w}p = {}^{W}T_{PT}\,{}^{PT}T_{LRF}\,{}^{LRF}p \qquad \text{--- (4.13)}$$

where $^{w}p$ and $^{LRF}p$ are the coordinates of the laser spot with respect to individual frame.

The coordinates from the laser range finder may be expressed in the laser range finder frame as

$$ {}^{LRF}p = \begin{bmatrix} 0 \\ 0 \\ d_{LRF} \\ 1 \end{bmatrix} $$

where $d_{LRF}$ is the distance read from the laser range finder. Thus,

$$ {}^{w}p = \begin{bmatrix} c\varphi\, c\vartheta & -s\varphi & c\varphi\, s\vartheta & 0 \\ s\varphi\, c\vartheta & c\varphi & s\varphi\, s\vartheta & 0 \\ -s\vartheta & 0 & c\vartheta & z_w \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ * \begin{bmatrix} s\varphi_{LRF} & -c\varphi_{LRF}\, s\vartheta_{LRF} & c\varphi_{LRF}\, c\vartheta_{LRF} & {}^{PT}x_{LRF} \\ -c\varphi_{LRF} & -s\varphi_{LRF}\, s\vartheta_{LRF} & s\varphi_{LRF}\, c\vartheta_{LRF} & {}^{PT}y_{LRF} \\ 0 & -c\vartheta_{LRF} & -s\vartheta_{LRF} & {}^{PT}z_{LRF} \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ d_{LRF} \\ 1 \end{bmatrix} = $$

$$ \begin{bmatrix} c\varphi\, c\vartheta\left(d_{LRF}\, c\varphi_{LRF}\, c\vartheta_{LRF} + {}^{PT}x_{LRF}\right) - s\varphi\left(d_{LRF}\, s\varphi_{LRF}\, c\vartheta_{LRF} + {}^{PT}y_{LRF}\right) + c\varphi\, s\vartheta\left(-d_{LRF}\, s\vartheta_{LRF} + {}^{PT}z_{LRF}\right) \\ s\varphi\, c\vartheta\left(d_{LRF}\, c\varphi_{LRF}\, c\vartheta_{LRF} + {}^{PT}x_{LRF}\right) + c\varphi\left(d_{LRF}\, s\varphi_{LRF}\, c\vartheta_{LRF} + {}^{PT}y_{LRF}\right) + s\varphi\, s\vartheta\left(-d_{LRF}\, s\vartheta_{LRF} + {}^{PT}z_{LRF}\right) \\ -s\vartheta\left(d_{LRF}\, c\varphi_{LRF}\, c\vartheta_{LRF} + {}^{PT}x_{LRF}\right) + c\vartheta\left(-d_{LRF}\, s\vartheta_{LRF} + {}^{PT}z_{LRF}\right) \end{bmatrix} $$

--- (4.14)

### 4.2.2 Part frames [1]

Based on the preceding computation, the RTSA module gets a vector describing the origin of the part frame with respect to the world frame, which constrains the

position of original of the part frame. It also gets the roll-pitch-yaw angles of the part frame with respect to the world frame, which constrains the orientation of the part frame. With this information, a part frame (in figure 4.4) is uniquely set.

To select a pipe in the work space, the operator needs to choose two points along this pipe. Then the part frame is defined in the following way:

(1) The original of the part frame is virtually set at the centerline of the first chosen point from the viewpoint of the laser range finder.

(2) The direction of x y z axes of the part frame is set by: in the first step, the coordinates of each of the chosen points is calculated so that the vector connecting the two points, ${}^{PT}L_1$, may be specified. Ideally, this vector is parallel to the axis of the pipe, and the unit vector in this direction is used as the z-vector in the part frame, $\Sigma_p$. The cross product between this vector

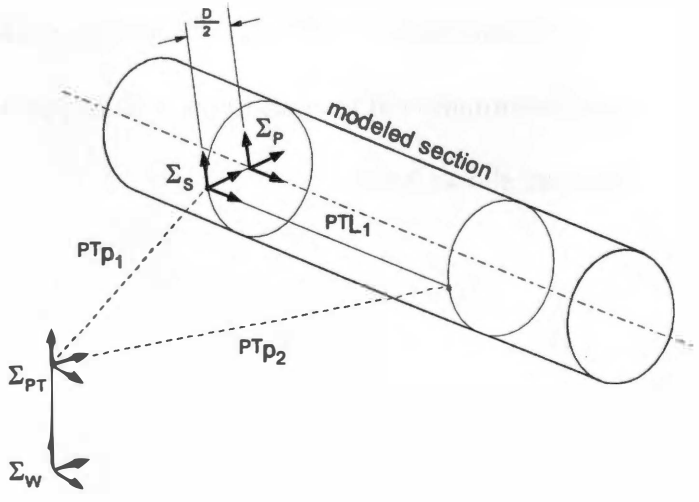



Figure 4.4 Pipe placement coordinate frames [1]

and the direction to the first point chosen, ${}^{PT}p_1$, defines a tangent to the pipe cross-section, which is used to determine the y unit vector of the pipe frame $\Sigma_p$. The final vector necessary to define the base frame is the x unit vector, which is determined by taking a cross product of the z and y vectors and points toward the centerline of the pipe.

From the above definitions, it is easy to understand that the part frame is a frame fixed to the pipe to be cut.

Having defined the part frame, based on the preceding computation, the RTSA module gets a vector with respect to the world frame describing the origin of the part frame. This constrains the position of the origin of the part frame. It also provides the roll-pitch-yaw angles of the part frame with respect to the world frame. This constrains the orientation of the part frame.

RTSA module uses C++ structures to record the roll-pitch-yaw offsets of the origin and other geometric information of the part frame with respect to the world frame. The data structures are shown below:

```
struct Position3D {
      float x;
      float y;
      float z;
      };
struct Orientation {
      float y;                // yaw (rotation about X axis)
      float p;                // pitch (rotation about Y axis)
      float r;                // roll (rotation about Z axis)
      };
```

With information in the above data structures, a part frame is uniquely set, and the transformation between part frame and world frame is

$$
{}^{World}T_{Part} =
\begin{bmatrix}
\cos R * \cos P & \cos R * \sin P * \sin Y - \sin R * \cos Y & \cos R * \sin P * \cos Y + \sin R * \sin Y & off\_x \\
\sin R * \cos P & \sin R * \sin P * \sin Y + \cos R * \cos Y & \sin R * \sin P * \cos Y - \cos R * \sin Y & off\_y \\
-\sin P & \cos P * \sin Y & \cos P * \cos Y & off\_z \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

--- (4.15)

In Eq. (4.15), R is roll angle; P is pitch angle; Y is yaw angle; and off_x, off_y and off_z are the coordinates of the origin of the part frame with respect to the world frame.

### 4.2.3 Cutting Frame

In our computation, we define another frame, the cutting frame, for the convenience of both calculation and clarity. The coordinates of key points in the computation are fairly simple within the cutting frame. For example, the cutting point is just the origin of the cutting frame, i.e. (0,0,0) and the approach point (to be described later) is always on the x-axis of the cutting frame, i.e. $(\Delta,0,0)$.

The origin of the cutting frame is same as the cutting point, which the operator can adjust during the planning.

The cutting frame is obtained by a sequence of translation and rotation in the following way:

(1) The direction of the z axis of the cutting frame is identical to that of the z axis of the part frame. The direction of the *x* and y axis is from the rotation by the angle that the operator sets about the z axis.

(2) The cutting frame is initially determined by translating the part frame along part frame's inverse current x-direction with a distance of the pipe radius so that the new base frame lies on the surface of the pipe. Then the new frame is translated along the current z axis with an amount adjusted by the operator.

The transformation matrix, ${}^{Part}T_{cut}$, between the cut and part frames is shown in figure 4.4 is given by a translation along the pipe (z) axis by an amount delta and rotation along the pipe (z) axis by an angle A. The radius of the pipe is r. Therefore

$$ {}^{Part}T_{cut} = \begin{bmatrix} \cos A & -\sin A & 0 & r\cos A \\ \sin A & \cos A & 0 & r\sin A \\ 0 & 0 & 1 & delta \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{--- (4.16)} $$

### 4.2.4. Robot Frame

The robot frame is the frame with respect to which the controller in the HMCT module interprets control signals and sends commands. The required information for an operation point includes:

(1) Roll-pitch-yaw angles of the manipulator end-effector with respect to the robot frame.

(2) Coordinates of the operation point with respect to the robot frame.

All coordinate and orientation information sent to the real-time controller should be described in the robot frame, a fixed frame in the system. The robot frame is defined on the floor at the base of the robot shaft. The orientation of the robot frame axes is approximately the same as that of the world frame.

Similar to the relation between the PT frame and the LRF frame, the robot frame and the world frame should be the same orientation. Small errors in the mounting of the base of the robot arm require a fixed transformation between the two frames. With system calibration, the roll-pitch-yaw angles are obtained with small number, and the origin of the robot frame is different from the origin of the world frame, so there are translations along x,y and z axes. Anologous to the transformation between PT frame and LRF frame, the transformation between robot frame and world frame is

$^{Robot}T_{world}=$

$$\begin{bmatrix} \cos R*\cos P & \cos R*\sin P*\sin Y-\sin R*\cos Y & \cos R*\sin P*\cos Y+\sin R*\sin Y & off\_x \\ \sin R*\cos P & \sin R*\sin P*\sin Y+\cos R*\cos Y & \sin R*\sin P*\cos Y-\cos R*\sin Y & off\_y \\ -\sin P & \cos P*\sin Y & \cos P*\cos Y & off\_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

--- (4.17)

Number we obtained with system calibration in REMSL are

R = 1.2 degrees

P = -1 degree

Y = -1 degree

Where R, P and Y are the roll, pitch and yaw angles respectively of the world frame with respect to the robot frame, which are small amount but not negligible. The offsets,

off_x = -1165 mm

off_y = -22 mm

off_z = 1331 mm

are those of the world frame with respect to the robot frame.

### 4.3 Task Planner Computation

It is very simple and easy to describe a point in the cutting frame. All the control signals related to geometric information in HMTC module are with respect to robot frame, and the transformation from the cutting frame to the robot frame must be obtained. Actually, all computation in the task planner module follows this transformation path: cutting frame→ part frame→ world frame→ robot frame. The relation between the coordinates with respect to the cutting frame and those with respect to the robot frame is

$$^{robot}p = {}^{robot}T_{cut} * {}^{cut}p$$

Transformation from the cutting frame to the robot frame can be decomposed into a sequence of transformations discussed in the preceding sections: 1) transformation from cutting frame to part frame; 2) transformation from part frame to world frame and 3)transformation from world frame to robot frame, so

$$^{robot}T_{cut} = {}^{robot}T_{world} * {}^{world}T_{part} * {}^{part}T_{cut} \qquad \text{--- (4.18)}$$

On the right side of Eq. 4.18, all transformation matrices are known from the preceding sections. So the transformation from cutting frame to robot frame can be calculated directly. Let the overall transformation matrix be

$$ {}^{Robot}T_{cut} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{--- (4.19)} $$

### 4.3.1 Cut Point

The cut point is the point where the tool begins the cut i.e., it is the point where the tool begins to touch the part. The operator can set the cut point to a desired position at will in the cut selection window (to be described in 5.3) of the task planner by adjusting the direction about the part's axis and position along the part's axis. The angle A in Eq. (4.16) changes when the operator adjusts the direction and the offset delta. Although the cutting frame changes with the adjustment, the coordinate of the cut point with respect to the cutting frame is always (0,0,0). Therefore the coordinate of cut point with respect to robot frame is

$$ {}^{Robot}p = {}^{Robot}T_{cut} * \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} $$

The end-effector holds the cutting tool such as a saw. The HMCT module needs the orientation of the end-effector as input, so the task planner module must calculate

that parameter. The roll-pitch-yaw angles of the end-effector with respect to the robot frame can be calculated from transformation matrix (4.19) as

$$\text{Roll} = \text{atan2}\left(b_{21}, b_{11}\right)$$

$$\text{Pitch} = \text{atan2}\left(-b_{31}, \sqrt{b_{32}^2 + b_{33}^2}\right)$$

$$\text{Yaw} = \text{atan2}\left(b_{32}, b_{33}\right)$$

so the tool as held in the end-effector is parallel to the desired cutting plane.

### 4.3.2 Approach Point

The approach point is the point where the manipulator begins the tooling operation with a feedrate that the operator sets in the controller.

The coordinates of the approach point with respect to the cutting frame is always (d,0,0) because this point lies on x-axis according to the cutting frame definition. So the coordinate of the cut point with respect to the robot frame is

$$^{\text{Robot}}p = {}^{\text{Robot}}T_{\text{cut}} * \begin{bmatrix} d_1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Currently,

We experimentally set $d_1$ = -25 mm for the approach point

At the approach point, the roll, pitch, and yaw angles are the same as the roll, pitch, and yaw angles of the cut point. This is easy to understand as the manipulator keeps the orientation during the cutting status.

### 4.3.3 Final Point

The final point is the point where the manipulator completes the tooling operation and stops going forward at the specified the feed rate and begins to retract.

The coordinate of the cut point with respect to cutting frame is always (d,0,0). So the coordinate of the cut point with respect to the robot frame is

$$^{Robot}p = {}^{Robot}T_{cut} * \begin{bmatrix} d_2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Currently,

$d_2$ = 100 mm for the approach point

And also, at the approach point, the roll, pitch, and yaw angles are the same as the roll, pitch, and yaw angles of the cut point.

### 4.3.4 Constraint Point

The constraint point is a special point needed by the planar assist function mode. As three points not aligned in a line forms a plane, we need a constraint point to set the correct cutting plane. As we know from our definition of the cutting frame, the cutting plane is identical to the x-y plane of the cutting frame. So we pick a point at y axis with respect to the cutting frame as the constraint point.

The coordinate of the constraint point with respect to the cutting frame is always (0, $d_3$,0). So the coordinate of cut point with respect to robot frame is

$$ {}^{Robot}p = {}^{Robot}T_{cut} * \begin{bmatrix} 0 \\ d_3 \\ 0 \\ 1 \end{bmatrix} $$

In our calculation, $d_3$=50 mm.

## 4.4 Task Plan Outputs

Operation in the autonomous mode needs the positions and orientations of the end-effector at the cutting point, approach point and final point.

Operation in the teleoperation with linear assist function mode needs the positions of the end-effector at the approach point and final point. Operation in the teleoperation with planar assist function mode needs the positions of the end-effector at the approach point, final point and constraining point. Operation in the teleoperation with velocity assist function mode needs the positions of the end-effector at the final point, where the controller sends a command to stop. Refer to figure 4.5 for an example of the task plan file. Command line 12 specifies pure teleoperation; command line 13 specifies a via point inserted by the operator; Command line 14 specifies teleoperation with linear assist function.

The task plan file is transmitted over the Ethernet by the task planner to the real-time control computer when the operator feels satisfied with the result. Each action in

the task plan file gives Boolean information about the end-effector state and the task time duration. The format of the file is such that it can be parsed by a state transition component in ControlShell called 'fetchandparse.' A trajectory generator there creates a smooth motion along a path from the manipulator's current position to the next given position. If the plan is already in place and being executed, and it is found that teleoperation is necessary, it may be halted and control transferred to the operator.

```
1 move  x -265.0  y 1400.0  z 960.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 100  TimeDur 15
2 move  x -265.0  y 1580.0  z 895.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 100  TimeDur 15
3 move  x -265.0  y 1572.0  z 895.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 70  TimeDur 15
4 move  x -265.0  y 1572.0  z 1100.0  roll 90.0  pitch 45.0  yaw 0.0  Gripper 70  TimeDur 15
5 move  x 100.0  y 1200.0  z 1300.0  roll 90.0  pitch 45.0  yaw 0.0  Gripper 70  TimeDur 15
6 move  x 1138.4  y 287.7  z 1659.8  roll 8.6  pitch -8.9  yaw 178.6  Gripper 70  TimeDur 15
7 move  x 1162.8  y 291.5  z 1663.6  roll 8.6  pitch -8.9  yaw 178.6  Gripper 70  TimeDur 15
8 move  x 1262.2  y 295.2  z 1667.5  roll 8.6  pitch -8.9  yaw 178.6  Gripper 70  TimeDur 90
9 move  x 1162.8  y 291.5  z 1663.6  roll 8.6  pitch -8.9  yaw 178.6  Gripper 70  TimeDur 90
10 move  x 1138.4  y 287.7  z 1659.8  roll 8.6  pitch -8.9  yaw 178.6  Gripper 70  TimeDur 15
11 move  x 800.0  y 200.0  z 0.0  roll 0.0  pitch 0.0  yaw 0.0  Gripper 70  TimeDur 15
12 teleop_manual
13 move  x 600.0  y 1000.0  z 1400.0  roll 0.0  pitch 0.0  yaw 90.0  Gripper 70  TimeDur 15
14 teleop_assist Flag 'L'  P1  x 861.9  y 73.5  z 1273.6  P2  x 1357.2  y 102.9  z 1393.8
15 move  x -265.0  y 1400.0  z 1100.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 70  TimeDur 15
16 move  x -265.0  y 1572.0  z 895.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 70  TimeDur 15
17 move  x -265.0  y 1572.0  z 895.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 100  TimeDur 15
18 move  x -265.0  y 1400.0  z 1100.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 100  TimeDur 15
19 move  x -265.0  y 1000.0  z 1100.0  roll 90.0  pitch 43.0  yaw 0.0  Gripper 100  TimeDur 15
```

Figure 4.5 Sample of the Task Plan File

# Chapter 5

# Graphical User Interface

## 5.1 Graphical User Interface in Task Planner

As mentioned in Chapter 3, an operator working long periods is subject to fatigue. Therefore the task planner of this telerobotic remote work system needs to provide the operator a very flexible operation environment. In this concern, the task planner module must provide operators a friendly graphic user interface environment.

The graphical user interface of the task planner allows the operator to plan a set of subtasks for the system to execute. The interface should maximize user efficiency, therefore it should minimize the planning time and knowledge needed to accurately run the system. Based on these requirements, a basic task planner design was constructed. It prompts the operator through the required inputs with a minimum number of windows. The output of the task planner is a list of actions which are downloaded over the Ethernet as a text file to the real-time control computer.

The basic structure of the task planner is shown in figure 5.1. As shown, the task planner allows all possible tooling scenarios to be planned from twelve simple

windows. The structure also allows the operator to edit his planning inputs before saving.

### 5.2. Developing GUI with MFC

The RTSA module and task planner module is installed on a machine with Window NT operating system. The graphical user interface program must be compatible with this operating system.

There are two ways to develop the graphical user interface with VC++. The first is to use the API functions defined in Win32. In this approach, the program can directly utilize the API and explicitly handle all of the details associated with a Windows program. The second way to program for Windows uses a special C++ class library, which encapsulates the API functions. By far, the most popular Windows programming class library is Microsoft Foundation Classes (MFC). MFC is powerful tool that offers significant advantages in some situations. MFC masks the complex features of Windows API, which are unnecessary in most situations. MFC also masks the detailed mechanism of message communication, which improves the efficiency of software development. In this thesis, we develop the software in the MFC approach [21].

The Microsoft Foundation Class Library (MFC) is an "application framework" for programming in Microsoft Windows. Written in C++, MFC provides much of the code necessary for managing windows, menus, and dialog boxes; performing basic input/output; storing collections of data objects; and so on. All the programmer needs

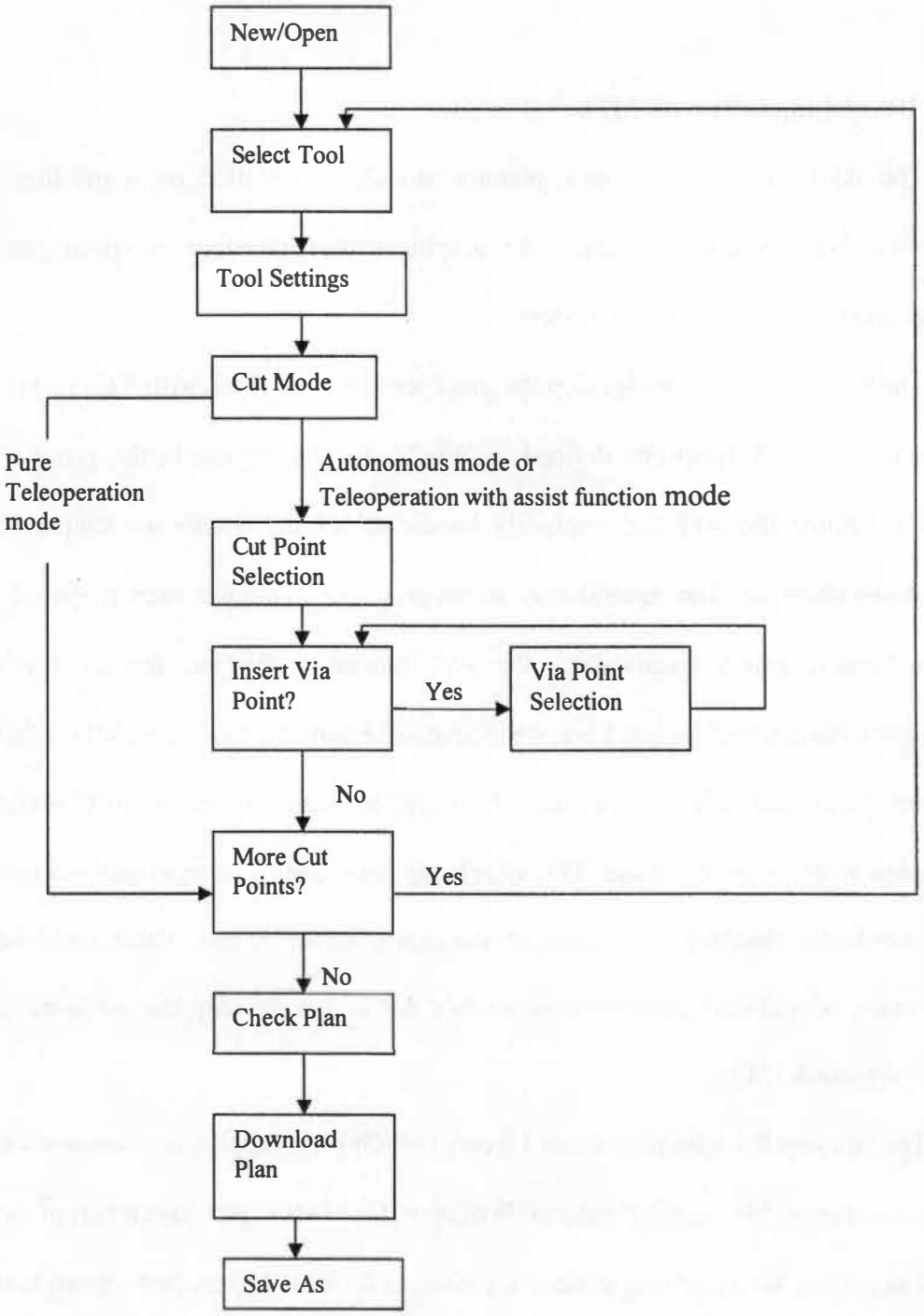



Figure 5.1 Task planner scheme

to

do is add application-specific code into this framework. Given the nature of C++ class programming, it is easy to extend or override the basic functionality MFC supplies. MFC shortens development time; makes code more portable; provides tremendous support without reducing programming freedom and flexibility; and gives easy access to "hard to program" user-interface elements [23].

To develop graphical user interface with MFC, the programmer needs to understand the MFC hierarchy (see appendix 1)

## 5.3 Windows for parameter setting in Task Planner

The task planner is a large set of functions responsible for creating a task plan from operator and model information. It provides the operator with a set of windows to select the object to be cut, tools to use, etc. The final result is a linked list of actions which are downloaded as a text file through the Ethernet to the control computer. The windows are briefly described in this section.

Starting Task Planner

As the TP is the bridge between the RTSA module and the real-time controller module and it utilizes the geometric information from RTSA, the task planner is entered through the Main RTSA window.

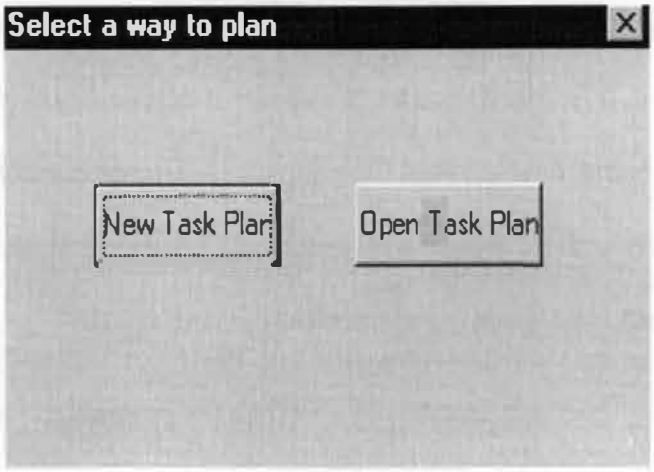



Figure 5.2 New/Open Window

**New/Open window**

The New/Open window (figure 5.2) allows the operator to begin a new task plan file by clicking the button "New Task Plan". On clicking the "New" button, the operator begins a process to plan a task, which may include as many subtasks in figure 1.1 as he desires.

**Select Tool window**

The Select tool window (figure 5.3) allows the tool to be used in the tooling operation to be selected for an individual subtask. The operator needs only to select from a tool list as the tool in the operation, and then click "OK" after done. Though the present system provides only functionality of sawing, saw, drill, grippers and scissors are selectable here as tools demonstration.

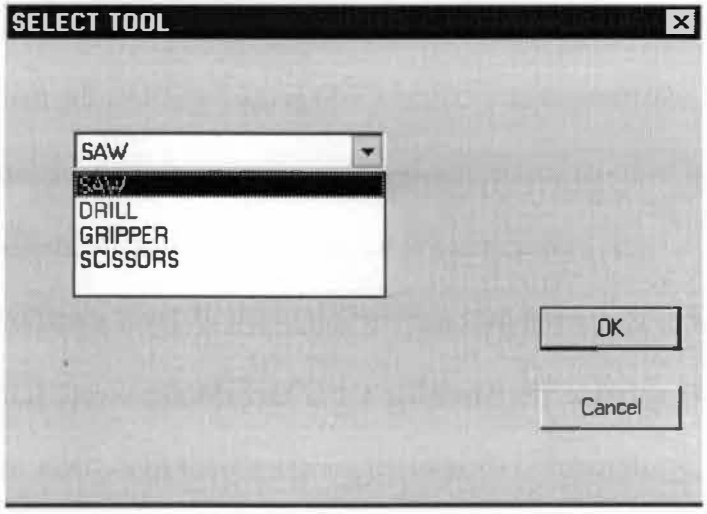



Figure 5.3 Select Tool window

**Tool settings window**

In the Tool settings window (figure 5.4), the tool specific parameters for that operation are selected. For example, if the selected tool in the preceding window is a saw or drill, the operator needs to set the tool feedrate during the operation in this window. A class member variable "m_Feedrate" in the CToolSetting{ }class is used here to record the tool setting information. See appendix 3 for the dialog classes for each dialog interface.

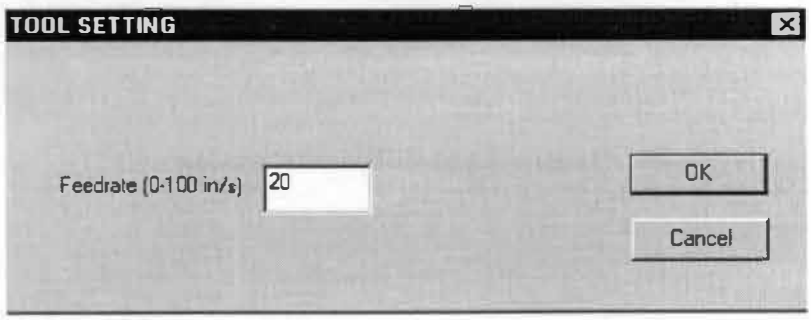



Figure 5.4 Tool Setting window

**Mode selection window**

The Mode selection window (figure 5.5) is used to select the mode of operation. Choices are autonomous mode, teleoperation using an assist function, or pure teleoperation. Linear, planar, and velocity assist functions are available. Forms of assistance may be chosen in the construction of a high level plan for particular tasks. A class member variable "m_Mode" in the CAssistModeSelect{ }class is used here to record the mode information so as to trigger the proper mode later in the real-time controller mode later.

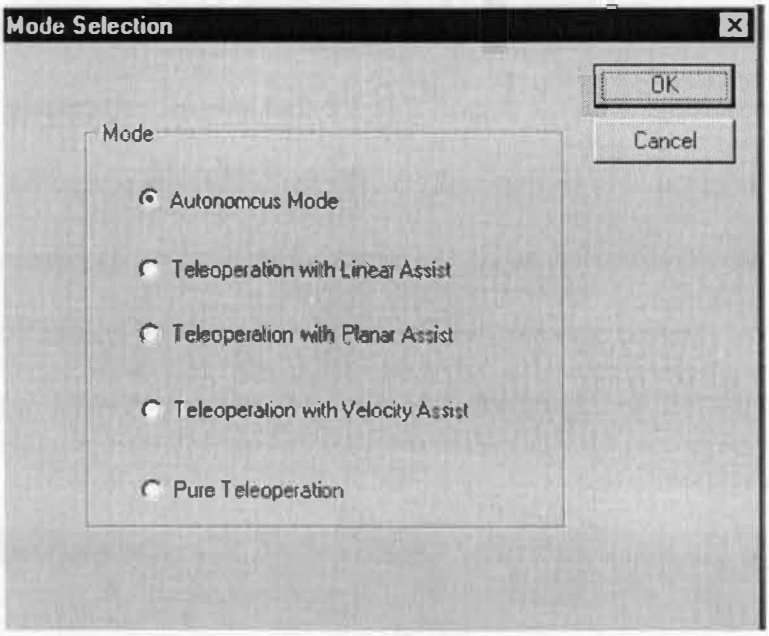



Figure 5.5 Mode Selection window

**Cut point selection window**

Next, in the Cut point selection window (figure 5.6), the cut plane is selected from the model previously created in the RTSA module [1]. At the top of this window is a drop list used to select the part which is to be cut. When a part is selected, it is highlighted in red in the Envision window and a cutting plane in the shape of a flat arrow appears along the axis of the part. Buttons on the window allow the operator to translate the cutting plane along the z-axis of the part and to rotate the direction of the approach around the z-axis. When the selections have been made, the operator stores this point. The computation process is shown in appendix 2. Here one thing needs to be clarified: though the computation is developed in this thesis, the data structure to record the geometrical information is from Fu's work [16].

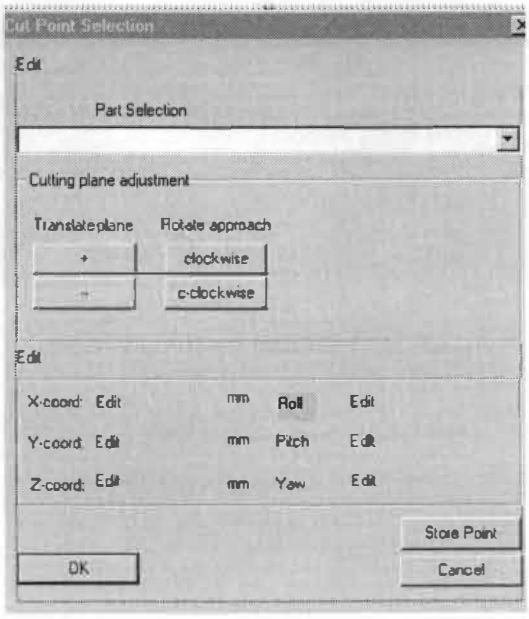



Figure 5.6 Cut Point Selection window

**Via point windows**

Once the cutting point has been selected, any needed via points can be added through the Via point windows (figures. 5.7 and 5.8).

A via point is an extra point added by the operator into the motion trajectory of the manipulator. Via points may be included to ensure that the manipulator moves around expected obstacles between cuts or during tool retrieval and replacement. After cut points have been selected from the previously described window, the drop list in this window will contain a list of those cutting operations. The operator may then choose to insert via points after any of the operations in this list. They will be included in the final motion trajectory created for download to the controller. The via point coordinates are entered manually.

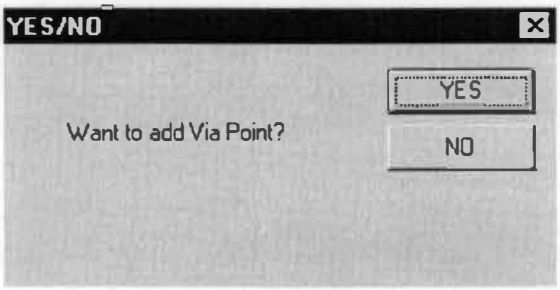



Figure 5.7 Via Point Insertion window

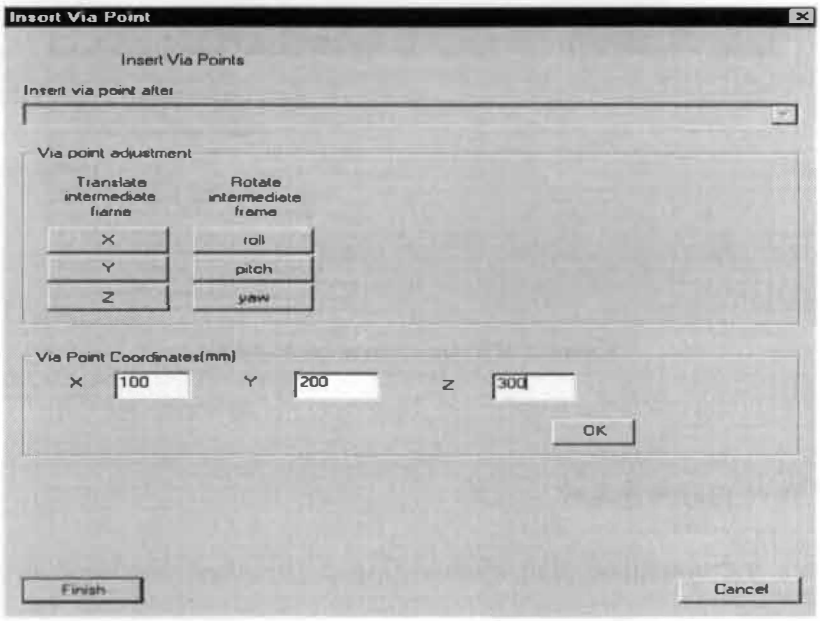



Figure 5.8 Via Point Coordinates Input window

**More cuts window**

So far, a subtask planning is completed. With preceding windows, the operator can specify all the information including operating mode for that subtask. As the overall task of this telerobotic system consist of as many subtask as desired, the task planner needs to ask the operator to decide if he needs an additional subtask, either in the same mode as just planed or in a different mode.

Therefore, finally, the operator is asked whether more cuts should be added to the plan in the More cuts window (figure 5.9). If more cuts are needed in the task plan file, the operator is prompted to begin the cut plane selection process again with figure 5.8. It begins planning another subtask. If not, the plan is completed by checking "No"and downloading it.

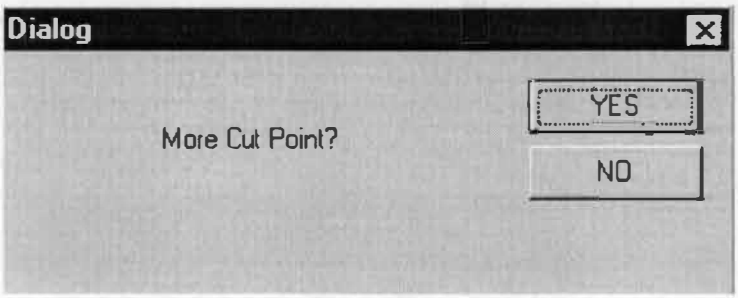



Figure 5.9 More Operation window

**Check and Save plan window**

The Check and download plan window (Fig. 5.10) allows the basic structure of the file to be checked. It also allows the operator to choose to download the task plan file to the location of choice. When opened, the check plan window displays a simple alphanumeric list representing the results of the high-level task planning operations.

**Save as window**

The task plan file can be saved into any location through the Save as window (Fig. 5.11).

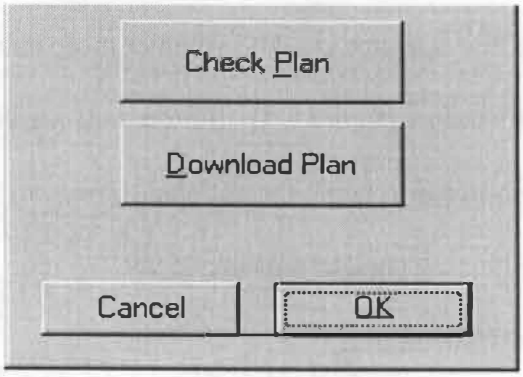



Figure 5.10 Check and Save Plan window


Figure 5.11 Save As window

These windows work together to form the task planner. An example application will be presented in the following chapter.

# Chapter 6

## Experimental Results and Conclusions

This chapter describes the experimental results and gives some conclusions with regard to the requirements presented in Chapter 1. Finally, future work is discussed.

### 6.1 Experimental Results

#### 6.1.1 Performance verification experiment

A task plan file that can be executed through the real-time controller is shown in Fig. 6.1. Commands in the first five lines control the manipulator to pick up a tool. Commands in lines 6-10 control the manipulator's execution of an autonomous cutting subtask. Line 11 commands the manipulator to move to a via point. Line 12 sets the flag to trigger a pure teleoperation in which the operator has total control of the manipulator. The commands in lines 13-18 tell the manipulator to put down the tool and return to the initial manipulator position.

```
1 move x -265.0 y 1400.0 z 960.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 100 TimeDur 15
2 move x -265.0 y 1580.0 z 895.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 100 TimeDur 15
3 move x -265.0 y 1572.0 z 895.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 70 TimeDur 15
4 move x -265.0 y 1572.0 z 1100.0 roll 90.0 pitch 45.0 yaw 0.0 Gripper 70 TimeDur 15
5 move x 100.0 y 1200.0 z 1300.0 roll 90.0 pitch 45.0 yaw 0.0 Gripper 70 TimeDur 15
6 move x 1138.4 y 287.7 z 1659.8 roll 8.6 pitch -8.9 yaw 178.6 Gripper 70 TimeDur 15
7 move x 1162.8 y 291.5 z 1663.6 roll 8.6 pitch -8.9 yaw 178.6 Gripper 70 TimeDur 15
8 move x 1262.2 y 295.2 z 1667.5 roll 8.6 pitch -8.9 yaw 178.6 Gripper 70 TimeDur 90
9 move x 1162.8 y 291.5 z 1663.6 roll 8.6 pitch -8.9 yaw 178.6 Gripper 70 TimeDur 90
10 move x 1138.4 y 287.7 z 1659.8 roll 8.6 pitch -8.9 yaw 178.6 Gripper 70 TimeDur 15
11 move x 800.0 y 200.0 z 0.0 roll 0.0 pitch 0.0 yaw 0.0 Gripper 70 TimeDur 15
12 teleop_manual
13 move x 600.0 y 1000.0 z 1400.0 roll 0.0 pitch 0.0 yaw 90.0 Gripper 70 TimeDur 15
14 move x -265.0 y 1400.0 z 1100.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 70 TimeDur 15
15 move x -265.0 y 1572.0 z 895.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 70 TimeDur 15
16 move x -265.0 y 1572.0 z 895.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 100 TimeDur 15
17 move x -265.0 y 1400.0 z 1100.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 100 TimeDur 15
18 move x -265.0 y 1000.0 z 1100.0 roll 90.0 pitch 43.0 yaw 0.0 Gripper 100 TimeDur 15
```

Figure 6.1 Actual plan file generated by TP and executable with real-time controller

Experiments were done to verify that the task planner worked correctly. The task planner is working correctly if the points that the manipulator moves through are those designated by the operator during the task planning phase. In order to determine whether this condition was being met, a set of experiments was conducted.

Seven experiments were conducted and each was repeated three times. In each of the seven experiments, a task plan was created that would move the manipulator from the tool location to a cut point by way of the approach point. A cut would be made and then the manipulator would return the tool to the tool rack. The seven cut points were set by placing small pieces of white tape on pipes throughout the workspace.

The location of the seven pieces of tape was determined by placing the laser point on each piece of tape and reading the coordinates in the task planner interface. In the experiments, the manipulator was allowed to execute the task plan until the approach point was reached. The manipulator was then stopped at the approach point. The approach point was found in line 6 of figure 6.1 for the example presented. Then, the actual position of the end-effector was measured at that point. The location of the end-effector should agree with the coordinates specified for the approach point in the task plan. The distance between the approach point and its cut point was fixed within the task plan code at 25 mm. Measurements of the end-effector location were made with a tape measure and string. The distances in the Robot Frame between the end-effector location and the cut point in the x, y, and z directions were then measured as $\Delta x$, $\Delta y$, and $\Delta z$ as shown in Table 6.1. $\Delta x$, the difference between the manipulator position and the cut point in the x direction, is about 25mm. $\Delta y$ and $\Delta z$ measured similarly are within 10 mm. The distance between the end-effector location and cut point was also calculated and is given as d in Table 6.1.

The distance, d, was then compared to the correct 25 mm by a mean squared error (MSE) calculation.

$$\text{MSE} = \frac{\sum_{i=1}^{N}(d_i - d_0)^2}{N} \qquad \text{--- (6.1)}$$

Table 6.1 Difference between end-effector location at approach point and cut point.

| Point | Measurement | $\Delta x$ (mm) | $\Delta y$ (mm) | $\Delta z$ (mm) | $d=\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$ (mm) |
|---|---|---|---|---|---|
| #1 | 1 | 22 | 7 | -3 | 23.3 |
| | 2 | 23 | 5 | -2 | 23.6 |
| | 3 | 23 | 5 | -1 | 23.6 |
| #2 | 1 | 25 | 3 | 0 | 25.2 |
| | 2 | 26 | 1 | -1 | 26.0 |
| | 3 | 25 | 3 | -1 | 25.2 |
| #3 | 1 | 21 | 10 | 4 | 23.6 |
| | 2 | 21 | 9 | 3 | 23.0 |
| | 3 | 22 | 8 | 3 | 23.6 |
| #4 | 1 | 24 | 5 | 0 | 24.5 |
| | 2 | 23 | 5 | 0 | 23.6 |
| | 3 | 24 | 4 | 1 | 24.4 |
| #5 | 1 | 25 | 2 | 0 | 25.1 |
| | 2 | 24 | 3 | 1 | 24.2 |
| | 3 | 25 | 2 | -1 | 25.1 |
| #6 | 1 | 27 | 4 | -1 | 27.3 |
| | 2 | 26 | 3 | -3 | 26.3 |
| | 3 | 27 | 3 | -3 | 27.3 |
| #7 | 1 | 22 | 5 | 3 | 22.8 |
| | 2 | 23 | 4 | 2 | 23.4 |
| | 3 | 23 | 4 | 2 | 23.4 |

In Eq. (6.1), $d_0$=25mm, N=21. The result of the MSE is $1.96\,mm^2$. Therefore, the average difference between the desired and calculated distance is 1.4 mm. This result is satisfactory for D&D applications.

For a pipe approximately perpendicular to the floor, the roll-pitch-yaw angles in the task plan file are nearly $0-0-180^0$. The roll-pitch-yaw angles are not exactly $0-0-180^0$ because the selected pipe is not exactly perpendicular to the floor. And for a pipe approximately parallel to the floor, the roll-pitch-yaw angles in the task plan file are nearly $0-0-90^0$. The roll-pitch-yaw angles are not exactly $0-0-90^0$ because the selected pipe is not exactly parallel to the floor. During the experiments, the actual roll-pitch-yaw angles of the manipulator were not measured. However, it cut the pipe perpendicular to the pipe's longitudinal axis as intended, which proved that the manipulator's orientation under the instruction of task plan was correct.

### 6.1.2 Human-machine Interface expriment

An additional experiment was performed to determine whether the interface of the task planner was user friendly. Several students in REMSL, who were not familiar with the new task planner interface, were asked to construct a task plan that included three subtasks in different operating modes. All students were able to create a simple task plan including two autonomous subtasks within 3 minutes. This result suggests that the new task planner can be successfully used by operators that have little prior knowledge of the system.

## 6.2 Conclusion

From the results of the experiment presented in section 6.1.1, it can be concluded that the new task planner correctly computes the geometric information required by ControlShell. Therefore, the most important requirement of the TP for the present system has been satisfied.

The experiments of section 6.1.2 demonstrate that the task planner also provides a user friendly interface to an operator. This will greatly reduce training times for new operators. Reduced training times will result in great financial savings for the overall system operating according to [1]. The task plan file can be saved onto the server through an Ethernet connection. The required functionality for the task planner software has been met as shown in this thesis.

## 6.3 Future Work

The task planner developed in this thesis successfully creates a command file that can be interpreted by the real-time controller. However, it could be improved in the following ways.

The via point insertion step currently requires the operator to input the via point coordinates in a dialog box. This is inconvenient and may at times be impossible in a real and complex work space. The operator probably is not capable of quantatizing the coordinates of a desired a via point to avoid obstacles in the current via point insertion window. Therefore, an interactive graphical interface using manual control

of the manipulator end-effector is needed that will allow the operator to "quantify" the via point the workspace.

Another improvement would be to allow the operator to edit a previously created task plan file. This would allow existing plans to be modified to meet the current needs of the task.

Currently, the task plan file is a text file that can be interpreted by the real-time controller. If the operator could save the work space environment in the task plan file, a graphical preview of the planned operation could be produced.

# References

[1] William R. Hamel, "Robot Task Scene Analyzer Technical Overiew", DOE Final Technical Report, Contract number DE-AR26-97FT34314, 2000

[2] L. Homem de Mello, "Task Sequence Planning for Robotic Assembly", Dissertation for Doctor of Philosophy, Carnegie Mellon University, May 1989

[3] P. Khosla and R. Mattikalli, "Determining the assembly sequence from a 3-D model", Journal of Mechanical Working Technology, Sep. 1989, pp153-162

[4] A. Swaminathan and K. S. Barber, "An experience-based assembly sequence planner for mechanical assemblies", IEEE International Conference on Robotics and Automation, May 1995, Vol.2, pp 1278-1283

[5] Laura Dussinger and D. Scott Williams, "RRCS – Reconfigurable Remote Control System", Proc. on ANS 7th Topical Meeting on Robotics and Remote Systems, 1997, pp. 16-21

[6] Scott A. Couture, "Robust Telerobotics- an Integrated System for Waste Handling, Characterization and Sorting", Proc. on ANS 7th Topical Meeting on Robotics and Remote Systems, 1997, pp. 492-499

[7] John Tourellott, "Interactive Computer-Enhanced Remote Viewing System", Proc. on ANS 7th Topical Meeting on Robotics and Remote Systems, 1997, pp. 74-79

[8] Richard Hooper and Mark W. Noakes, "Kinematic Control Modes for Teleoperation of Redundant Robots" ", Proc. on ANS 7th Topical Meeting on Robotics and Remote Systems, 1997, pp. 38-43

[9] D. Verna, Dans Nagib Callaos, Susana Esquivel et Jamika Burge éditeurs "Virtual Reality and Tele-Operation: a Common Framework", SCI'2001, Fifth World Multi-Conference on Systemics, Cybernetics and Informatics, Sheraton World Resort, Orlando, Florida, USA. volume 3, pp 499- 504

[10] Robotics and Process System Division, Oak Ridge National Lab http://www.ornl.gov/rpsd/humfac/ accessed December 18, 2001

[11] Martin Jagersand, "On-line Estimation of Visual-Motor Models for Robot Control and Visual Simulation", Dissertation for Doctor of Philosophy in Computer Science, University of Rochester, August 1996

[12] Matthew. T. Mason, "The Mechanics of Manipulation", IEEE International Conference on Robotics and Automation, March 1985, pp 544-548

[13] D.M. Lane and M. Pickett, "Task Planning for Dextrous Manipulation Using Blind Grasping Tele-Assistance", International Journal of Systems Science, Volume 29, No 5, 1998, pp 513-527

[14] P. Virgili, R.Bono, G. Veruggio "General Purpose Human Computer Interface for Telerobotics in Hostile Environments", Proc. on ANS 7th Topical Meeting on Robotics and Remote Systems, 1997, pp. 939-946

[15] William R. Hamel, Pamela Murray, Ge Zhang, Sewoong Kim and Shuo Yang, "Human Machine Cooperative Telerobotics Final Report", DOE technical report, Contract Number DE-AR26-97FT34315, 2001

[16] Fu, Xiaoquan, "Interactive task planning for telerobotics", Thesis for Master of Science in Mechanical Engineering, University of Tennessee, May 1999

[17] S. E. Everett "Human-Machine Cooperative Telerobotics Using Uncertain Sensor and Model Data" Dissertation for Doctor of Philosophy in Mechanical Engineering, University of Tennessee, August 1998

[18] Karan A. Manocha, "Development and Implentation of Teleoperation Assist Functions" Thesis for Master of Science in Mechanical Engineering, University of South Florida, December 2000

[19] L. Sciavicco and B. Siciliano "Modeling and Control of Robot Manipulators" (2nd edition), The Springer-Verlag Inc., London, 2000

[20] Phillip John Mckerrow "Introduction to Robotics", Addison-Wesley Publishing Company, 1992

[21] Herbert Schildt, "Windows NT 4 Programming, from the ground up", McGraw-Hill Company, USA, 1997

[22] Richard M. Jones, "Introduction to MFC Programming with Visual C++", Prentice Hall Inc., 1999

[23] "Controlshell User Manual (version 7.0)", Technical Support Group, Real-Time Innovations Inc., 2000

# Appendices

# Appendix 1

## Microsoft Foundation Classes(MFC) Hierarchy

**Microsoft Foundation Class Library Version 6.0**

# Appendix 2

## Source Code of Computation

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

        UpdateData(TRUE);


        /* cos needs argument in radians, baseP,R,Yaw are in degrees */

        r11 = cos(PI/180*baseR)*cos(PI/180*baseP);
        r12 = cos(PI/180*baseR)*sin(PI/180*baseP)*sin(PI/180*baseYaw) - sin(PI/180*baseR)*cos(PI/180*baseYaw);
        r13 = cos(PI/180*baseR)*sin(PI/180*baseP)*cos(PI/180*baseYaw) + sin(PI/180*baseR)*sin(PI/180*baseYaw);
        r14 = baseX;
        r21 = sin(PI/180*baseR)*cos(PI/180*baseP);
        r22 = sin(PI/180*baseR)*sin(PI/180*baseP)*sin(PI/180*baseYaw) + cos(PI/180*baseR)*cos(PI/180*baseYaw);
        r23 = sin(PI/180*baseR)*sin(PI/180*baseP)*cos(PI/180*baseYaw) - cos(PI/180*baseR)*sin(PI/180*baseYaw);
        r24 = baseY;
        r31 = -sin(PI/180*baseP);
        r32 = cos(PI/180*baseP)*sin(PI/180*baseYaw);
        r33 = cos(PI/180*baseP)*cos(PI/180*baseYaw);
        r34 = baseZ;


//------------------------------------------------------------
        double radius;
        radius=Radius_2; //30.1625 mm
        //To add comments here
        //a11 = r11*cos(rollPosSel); Fu's a11 formula
        a11 = r11*cos(PI/180*rollPosSel)+r12*sin(PI/180*rollPosSel);// revised formula from above line
        a12 = -r11*sin(PI/180*rollPosSel) + r12*cos(PI/180*rollPosSel);
        a13 = r13;
        a14s= radius*cos(PI/180*rollPosSel)*r11+radius*sin(PI/180*rollPosSel)*r12+transPosSel*r13+baseX;
        a24s= radius*cos(PI/180*rollPosSel)*r21+radius*sin(PI/180*rollPosSel)*r22+transPosSel*r23+baseY;
        a34s= radius*cos(PI/180*rollPosSel)*r31+radius*sin(PI/180*rollPosSel)*r32+transPosSel*r33+baseZ;

        a21 = r21*cos(PI/180*rollPosSel) + r22*sin(PI/180*rollPosSel);
        a22 = -r21*sin(PI/180*rollPosSel) + r22*cos(PI/180*rollPosSel);
        a23 = r23;
        a31 = r31*cos(PI/180*rollPosSel) + r32*sin(PI/180*rollPosSel);
        a32 = -r31*sin(PI/180*rollPosSel) + r32*cos(PI/180*rollPosSel);
        a33 = r33;

        //change the coordinates from world frame to Robot frame


        //double b11, b12, b13, b14, b21, b22, b23, b24, b31, b32, b33, b34;

                b11=cos(Row_RW)*cos(Pitch_RW)*a11+a21*(cos(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)-
                sin(Row_RW)*cos(Yaw_RW))+
                a31*(cos(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)+sin(Row_RW)*sin(Yaw_RW));
                b12=cos(Row_RW)*cos(Pitch_RW)*a12+a22*(cos(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)-
                sin(Row_RW)*cos(Yaw_RW))+
                a32*(cos(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)+sin(Row_RW)*sin(Yaw_RW));
                b13=cos(Row_RW)*cos(Pitch_RW)*a13+a23*(cos(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)-
                sin(Row_RW)*cos(Yaw_RW))+
                a33*(cos(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)+sin(Row_RW)*sin(Yaw_RW));
```

```
b21=sin(Row_RW)*cos(Pitch_RW)*a11+a21*(sin(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)+cos(Row_
RW)*cos(Yaw_RW))+
a31*(sin(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)-cos(Row_RW)*sin(Yaw_RW));
b22=sin(Row_RW)*cos(Pitch_RW)*a12+a22*(sin(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)+cos(Row_
RW)*cos(Yaw_RW))+
a32*(sin(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)-cos(Row_RW)*sin(Yaw_RW));
b23=sin(Row_RW)*cos(Pitch_RW)*a13+a23*(sin(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)+cos(Row_
RW)*cos(Yaw_RW))+
a33*(sin(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)-cos(Row_RW)*sin(Yaw_RW));

b31=-sin(Pitch_RW)*a11+a21*cos(Pitch_RW)*sin(Yaw_RW)+a31*cos(Pitch_RW)*cos(Yaw_RW);
b32=-sin(Pitch_RW)*a12+a22*cos(Pitch_RW)*sin(Yaw_RW)+a32*cos(Pitch_RW)*cos(Yaw_RW);
b33=-sin(Pitch_RW)*a13+a23*cos(Pitch_RW)*sin(Yaw_RW)+a33*cos(Pitch_RW)*cos(Yaw_RW);

b14=cos(Row_RW)*cos(Pitch_RW)*a14s+a24s*(cos(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)-
sin(Row_RW)*cos(Yaw_RW))+
a34s*(cos(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)+sin(Row_RW)*sin(Yaw_RW))+OFFSET_X;
b24=sin(Row_RW)*cos(Pitch_RW)*a14s+a24s*(sin(Row_RW)*sin(Pitch_RW)*sin(Yaw_RW)+cos(Ro
w_RW)*cos(Yaw_RW))+
a34s*(sin(Row_RW)*sin(Pitch_RW)*cos(Yaw_RW)-cos(Row_RW)*sin(Yaw_RW))+OFFSET_Y;
b34=-
sin(Pitch_RW)*a14s+a24s*cos(Pitch_RW)*sin(Yaw_RW)+a34s*cos(Pitch_RW)*cos(Yaw_RW)+OFFS
ET_Z;


cut_x=(-2*radius-OFFSET_MANIPULATOR)*b11+b14;
cut_y=(-2*radius-OFFSET_MANIPULATOR)*b21+b24;
cut_z=(-2*radius-OFFSET_MANIPULATOR)*b31+b34;
app_x=(-2*radius-OFFSET_MANIPULATOR-DISTANCE)*b11+b14;
app_y=(-2*radius-OFFSET_MANIPULATOR-DISTANCE)*b21+b24;
app_z=(-2*radius-OFFSET_MANIPULATOR-DISTANCE)*b31+b34;
fin_x=(-2*radius-OFFSET_MANIPULATOR+3*radius+20)*b11+b14;
fin_y=(-2*radius-OFFSET_MANIPULATOR+3*radius+20)*b21+b24;
fin_z=(-2*radius-OFFSET_MANIPULATOR+3*radius+20)*b31+b34;
plane_x=-DISTANCE*b12+b14;
plane_x=-DISTANCE*b22+b24;
plane_x=-DISTANCE*b32+b34;


// update the text (prompting words)


// update the point coordinate information
sprintf(MsgStr1, "Point coordinates:");
m_PointInfo = (CString)MsgStr1;
sprintf(MsgStr2, "%8.1lf", cut_x);
m_xCoord = (CString)MsgStr2;
sprintf(MsgStr3, "%8.1lf", cut_y);
m_yCoord = (CString)MsgStr3;
sprintf(MsgStr4, "%8.1lf", cut_z);
m_zCoord = (CString)MsgStr4;
UpdateData(FALSE);

CComboBox* pOKBox;
pOKBox = (CComboBox*)GetDlgItem(IDOK);

pOKBox->EnableWindow(TRUE);


}
```

```
void CCutSelect::OnStore()
{
	// TODO: Add your control notification handler code here
	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
	((CRtsa2App*)AfxGetApp())->b_Firstpoint=firstPnt;


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

	ptr_EndPt->hold_tool =((CRtsa2App*)AfxGetApp())->RealholdTool;
	ptr_EndPt->mode = ((CRtsa2App*)AfxGetApp())->m_AssistMode;
	   ptr_EndPt->is_via = 0;		// this is a cut point, not a via point
	if(ptr_EndPt->mode!=0)
	{
		ptr_EndPt->telop = 1;  // a teleoperation
	}
	else
	{
		ptr_EndPt->telop = 0; // not a teleoperation
	}

	ptr_EndPt->coord_x = cut_x;
	ptr_EndPt->coord_y = cut_y;
	ptr_EndPt->coord_z = cut_z;

	ptr_EndPt->roll = atan2(b21, b11);
	ptr_EndPt->pitch = atan2(-b31, sqrt(b32*b32 + b33*b33));
	ptr_EndPt->yaw = atan2(b32, b33);


	//SHUO ADD BELOW
	ptr_EndPt->app_coord_x=app_x;
	ptr_EndPt->app_coord_y=app_y;
	ptr_EndPt->app_coord_z=app_z;
	ptr_EndPt->fin_coord_x=fin_x;
	ptr_EndPt->fin_coord_y=fin_y;
	ptr_EndPt->fin_coord_z=fin_z;

	if(ptr_EndPt->mode==2)		// mode==2(Tele/Planar) need a point to define a plane
	{
		ptr_EndPt->plane_coord_x=plane_x;
		ptr_EndPt->plane_coord_y=plane_y;
		ptr_EndPt->plane_coord_z=plane_z;

	}

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


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

}
```

# Appendix 3

## Source code of Interface

```
// Selectway.cpp : implementation file
//

#include "stdafx.h"
#include "rtsa2.h"
#include "Selectway.h"
#include "SelectToolDlg.h"

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

/////////////////////////////////////////////////////////////////////////////
/
// CSelectway dialog

//...

void CSelectway::OnNewTask()
{
	// TODO: Add your control notification handler code here
	((CRtsa2App*)AfxGetApp())->b_ContinueTaskPlan=true;
	((CRtsa2App*)AfxGetApp())->b_Firstpoint=1;
	while(((CRtsa2App*)AfxGetApp())->b_ContinueTaskPlan)
	{
		((CRtsa2App*)AfxGetApp())->firstcheck = 1;
		CSelectToolDlg Selectedtool;
		Selectedtool.m_Tooltype=((CRtsa2App*)AfxGetApp())
		->RealholdTool;
		Selectedtool.DoModal();
	}
	//EndDialog(IDOK);

}
```

```
// ToolSetting.cpp : implementation file

void CToolSetting::OnOK()
{
	// TODO: Add extra validation here
	CAssistModeSelect m_ModeSelect;
	m_ModeSelect.m_Mode=((CRtsa2App*)AfxGetApp())->m_AssistMode;
	EndDialog(IDOK);
	m_ModeSelect.DoModal();


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

	CDialog::OnOK();
}
```

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

#include "stdafx.h"
#include "Rtsa2.h"
#include "ViaPntDlg.h"
#include "ViaPointPrompt.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;
	m_Coor_X = 0.0f;
	m_Coor_Y = 0.0f;
	m_Coor_Z = 0.0f;
	//}}AFX_DATA_INIT
}


void CViaPntDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CViaPntDlg)
	DDX_CBIndex(pDX, IDC_VIAPT_COMBO, m_PreViaPt);
	DDX_Text(pDX, IDC_EDIT_X, m_Coor_X);
	DDX_Text(pDX, IDC_EDIT_Y, m_Coor_Y);
	DDX_Text(pDX, IDC_EDIT_Z, m_Coor_Z);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CViaPntDlg, CDialog)
	//{{AFX_MSG_MAP(CViaPntDlg)
	ON_CBN_CLOSEUP(IDC_VIAPT_COMBO, OnCloseupViaptCombo)
	ON_EN_CHANGE(IDC_EDIT_X, OnChangeEditX)
	ON_EN_CHANGE(IDC_EDIT_Y, OnChangeEditY)
	ON_EN_CHANGE(IDC_EDIT_Z, OnChangeEditZ)
	ON_BN_CLICKED(IDC_BUTTON_OK, OnButtonOk)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

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

void CViaPntDlg::OnCloseupViaptCombo()
```

```
{
        CComboBox* pViaPtBox;
        pViaPtBox = (CComboBox*)GetDlgItem(IDOK);

        pViaPtBox->EnableWindow(TRUE);


        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
                pEndPt->mode = ((CRtsa2App*)AfxGetApp())->m_AssistMode;

// dummy parameters, need to change in the future
                pEndPt->coord_x = m_Coor_X;
                pEndPt->coord_y = m_Coor_Y;
                pEndPt->coord_z = m_Coor_Z;

                pEndPt->roll = 0;
                pEndPt->pitch = 0;
                pEndPt->yaw = 0;
//

                ( ((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
        CViaPointPrompt m_ViaPoint;
        EndDialog(IDOK);
        m_ViaPoint.DoModal();
        CDialog::OnOK();
```

```
}

void CViaPntDlg::OnCancel()
{

       if(m_PreViaPt != -1)
           ( ((CRtsa2App*)AfxGetApp())->EndPtList ).dll_delete_node(
((Dllist)ptr)->flink );

       CViaPointPrompt m_ViaPoint;
       EndDialog(IDOK);
       m_ViaPoint.DoModal();

       CDialog::OnCancel();
}

void CViaPntDlg::OnChangeEditX()
{
       // TODO: If this is a RICHEDIT control, the control will not
       // send this notification unless you override the
       CDialog::OnInitDialog()
       // function and call CRichEditCtrl().SetEventMask()
       // with the ENM_CHANGE flag ORed into the mask.

       // TODO: Add your control notification handler code here
       int Temp;
       Temp=(int)GetDlgItemInt(IDC_EDIT_X);
       m_Coor_X=Temp;


}

void CViaPntDlg::OnChangeEditY()
{
       // TODO: If this is a RICHEDIT control, the control will not
              // send this notification unless you override the
       CDialog::OnInitDialog()
       // function and call CRichEditCtrl().SetEventMask()
       // with the ENM_CHANGE flag ORed into the mask.

       // TODO: Add your control notification handler code here
       int Temp;
       Temp=(int)GetDlgItemInt(IDC_EDIT_Y);
       m_Coor_Y=Temp;


}

void CViaPntDlg::OnChangeEditZ()
{
       // TODO: If this is a RICHEDIT control, the control will not
              // send this notification unless you override the
       CDialog::OnInitDialog()
       // function and call CRichEditCtrl().SetEventMask()
       // with the ENM_CHANGE flag ORed into the mask.

       // TODO: Add your control notification handler code here
       int Temp;
```

```
        Temp=(int)GetDlgItemInt(IDC_EDIT_Z);
        m_Coor_Z=Temp;


}

void CViaPntDlg::OnButtonOk()
{
        // TODO: Add your control notification handler code here

        CComboBox* pViaPtBox;
        pViaPtBox = (CComboBox*)GetDlgItem(IDC_VIAPT_COMBO);

        pViaPtBox->EnableWindow(TRUE);

        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");
                        pViaPtBox = (CComboBox*) GetDlgItem(IDC_VIAPT_COMBO);
                        pViaPtBox->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);
                        pViaPtBox = (CComboBox*) GetDlgItem(IDC_VIAPT_COMBO);
                        pViaPtBox->AddString(PrePtMessage);
                }
                else
                // this is a cut point
                {
                        CutPtIndex++;
                        PrePtMessage = strdup("cut point ");
                        _itoa(CutPtIndex, PtIndex, 10);
                        strcat(PrePtMessage, PtIndex);
                        pViaPtBox = (CComboBox*) GetDlgItem(IDC_VIAPT_COMBO);
                        pViaPtBox->AddString(PrePtMessage);
                }
        }
        return;


}
```

# Vita

Shuo Yang was born in Shenyang, China on December 3, 1973. He entered Shanghai Jiao Tong University in 1992. He received his Bachelor of Science degree in Instrumentation in July, 1996 at Shanghai Jiao Tong University. He enrolled in University of Tennessee to pursue his Master of Science in Mechanical Engineering specialized in the robotics and control.