

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2005

A unified robotic kinematic simulation interface.

Zhongqing Ding
University of Windsor

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

Recommended Citation

Ding, Zhongqing, "A unified robotic kinematic simulation interface." (2005). *Electronic Theses and Dissertations*. 857.

<https://scholar.uwindsor.ca/etd/857>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

A UNIFIED ROBOTIC KINEMATIC SIMULATION INTERFACE

by

Zhongqing Ding

A Thesis

Submitted to the Faculty of Graduate Studies and Research
through Industrial and Manufacturing Systems Engineering
in Partial Fulfillment of the Requirements for
The Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2005

© 2005 Zhongqing Ding



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-494-09760-4

Our file Notre référence

ISBN: 0-494-09760-4

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

ABSTRACT

Robotic controller and application programming have evolved along with the application of computer technologies. A PC-based, open architecture controller, off-line programming and simulation system integrated in "one-box" solution presents the latest advancement in robotics.

Open architecture controllers have been proven essential for all aspects of reconfiguration in future manufacturing systems. A Unified Reconfigurable Open Control Architecture (UROCA) research project is under way within the Intelligent Manufacturing Systems (IMS) Centre at the University of Windsor. Applications are for industrial robotic, CNC, and automotive control systems. The UROCA proposed architecture is a reconfigurable system that takes the advantages of different control structure types, thereby integrating them in a way to enhance the controller architecture design.

In order to implement the reconfigurable control strategies and ease application programming in UROCA, a Graphical User Interface (GUI) that provides 3D robotic kinematic modeling and simulation is necessary. The graphical robotic simulation platform is PC-based, and both application and hardware independent. The Object-oriented paradigm introduces a novel software architecture that has the following features: ease of use, extensibility, portability, and reusability.

This research develops a graphical robotic simulation platform by creating an optimized object-oriented design. This design approach implements all components in the Visual C++ programming language and freely distributed graphical library OpenGL, utilizing a single PC running the Windows operating system. The main component of the system includes a GUI, which can generate a kinematic model of most industrial robots by defining all the joint coordinate systems with their orientations and positions. Robot arm lengths and offsets are modified according to their exact values. Thereby, the Denavit-Hartenberg (D-H) parameters describing the arm geometry for direct and inverse kinematic problem solving are automatically generated and saved. The GUI also has general functions of commercial robotic simulation packages such as file saving and opening, various views, simple 3D geometric modeling, and simulation.

The case studies demonstrate that this graphical user interface is very effective in creating the kinematic models, and in verifying the direct and inverse kinematics solutions visually. This software platform was also created for geometric modeling and simulation of the whole work cell.

ACKNOWLEDGEMENTS

I would like to express my sincerest and deepest appreciation to my supervisor Professor Waguih ElMaraghy for giving me the opportunity, his guidance, and for helping me throughout the course of my M.A.Sc program. I would also like to extend my thanks to my Supervisory Committee, Dr. Guoqing Zhang and Dr. Bruce Minaker for their comments and time in reviewing my thesis.

I would like to acknowledge the great help and insight that I received from the IMS Centre directors, Professor Hoda ElMaraghy and Professor Waguih ElMaraghy, for giving me the chance to discuss the research topics with other members through the regular meetings they arranged for us. I learned a lot from these meetings. I would like to extend my thanks to all other members in IMS Centre for their suggestions and encouragement.

Within our IMS Centre, I would acknowledge Ms. Ana M. Djuric for her assistance with the Unified Kinematic Modeler and Solver. I also appreciated the discussion with Dr. ElSayed M. ElBeheiry about the UROCA project in relation to my thesis.

I would like to thank the Industrial and Manufacturing Systems Engineering Department staff: Ms. Jacquie Mummery, Mr. Ram Barakat, Ms. Zaina Batal, and Ms. Monique Gagnon for their support and kind assistance during my study.

Finally, I wish to express my great gratitude to my closest friends Ying Liu and Mike Cogan for their love, understanding, and encouragement, including suggestions on programming issues.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
CHAPTER 1 INTRODUCTION	1
1.1 Introduction to Robotics	1
1.1.1 Introduction to Industrial Robots	2
1.1.2 Kinematics	5
1.1.3 Dynamics and Control	12
1.1.4 Application Programming.....	13
1.2 Introduction to Unified Reconfigurable Open Control Architecture	14
1.3 Motivation and Objectives	16
1.4 Thesis Overview	18
CHAPTER 2	
OVERVIEW OF GRAPHICAL ROBOTIC SIMULATION SYSTEMS	20
2.1 The Main Functionality	20
2.2 Literature Review	22
2.2.1 The Simulation Systems Separated from the Control Systems	22
2.2.2 The Simulation Systems Integrated with the Control Systems	31
2.3 The Theories and Technologies	40
2.3.1 Introduction to Computer Graphics	40
2.3.2 3D Object Model.....	42

CHAPTER 3 DESIGN REQUIREMENTS.....	46
3.1 Generic Puma-Fanuc Model	46
3.2 A Unified Geometric-Based Solution.....	49
3.2.1 Solution for the First Three Joints	52
3.2.2 Solution for the Last Three Joints	55
3.3 Design Requirements	57
CHAPTER 4 DESIGN AND IMPLEMENTATION.....	58
4.1 System Structure	58
4.2 Menu Structure and Main Functions	60
4.2.1 The Graphical User Interface	61
4.2.2 File Functions.....	63
4.2.3 View Functions	63
4.2.4 Geometry Model	63
4.2.5 Kinematic Model	64
4.2.6 Simulation	65
4.3 Design Methodologies	66
4.3.1. Objected-oriented Design Approach.....	66
4.3.2. Library Modules and Dynamic-Link Libraries Design.....	71
4.3.3. Scene Graphs	74
4.3.4. Visual C++implementation.....	79
CHAPTER 5 SIMULATION EXAMPLES AND RESULTS.....	82
5.1 Case Study one for PUMA-like Robots.....	82
5.1.1 Problem Description	82

5.1.2 Inverse Kinematics Solutions	84
5.1.3 Verification	87
5.2 Case Study two for Fanuc-like Robots	90
5.2.1 Problem Description	90
5.2.2 Inverse Kinematics Solutions	92
5.2.3 Verification	94
5.3 Operations of the Interface	96
5.3.1 Kinematic Model	96
5.3.2 Geometric Model	98
5.3.3 Definition of Target Points	98
5.3.4 Simulation	99
CHAPTER 6 CONCLUSIONS AND FUTURE WORKS	100
6.1 Conclusions.....	100
6.2 Contributions	102
6.3 Future Work	102
REFERENCES.....	105
APPENDIX 1: SAMPLE C++ PROGRAMS	110
APPENDIX 2: THE FILE DATA FOR CASE STUDY ONE.....	113
VITA AUCTORIS	115

LIST OF FIGURES

Figure 1.1: Cincinnati Milacron T Robot arm [Fu et al, 1987].....	2
Figure 1.2: PUMA 560 Joint Limits [Fu et al, 1987].....	3
Figure 1.3: Work Envelope.....	4
Figure 1.4: The Direct and Inverse Kinematic Problems.....	6
Figure 1.5: A PUMA Robot Arm Illustrating Joints and Links [Fu et al, 1987].....	7
Figure 1.6: Link Coordinate Frame and its Parameters.....	8
Figure 1.7: PUMA Link Coordinate Frame [Fu et al, 1987].....	9
Figure 1.8: Determination of Link Frames from D-H Parameters.....	10
Figure 2.1: CODE Architecture Modified from [Cimetrix Inc., 2002].....	32
Figure 2.2: Hardware Configuration of the PC-ORC [Hong et al, 2001].....	33
Figure 2.3: Overall Structure of the PC_ORC Modified from [Hong et al, 2001].....	34
Figure 2.4: Class Hierarchy of the Robot Platform [Loffler et al, 2001].....	37
Figure 2.5: Run-time Architecture of The Robotic Platform [Loffler et al, 2001].....	38
Figure 2.6: Application Programmer's Model of Graphical System.....	41
Figure 2.7: Geometric Primitive Types [Angel, 2003].....	44
Figure 3.1: Generic Puma kinematic Structure [Djuric et al, 2004].....	47
Figure 3.2: Generic Fanuc kinematic Structure [Djuric et al, 2004].....	47
Figure 3.3: Generic Puma and Fanuc Kinematic Structure [Djuric et al, 2004].....	48
Figure 3.4: Definition of Various Arm Configurations [Djuric et al, 2004].....	50
Figure 3.5: Position Vector \mathbf{p} [Djuric et al, 2004].....	52
Figure 3.6: LA Projection of \mathbf{p} onto x_0y_0 Plane [Djuric et al, 2004].....	52
Figure 3.7: RA Projection of \mathbf{p} onto x_0y_0 Plane [Djuric et al, 2004].....	52

Figure 3.8: Projection of Vector p onto x_1y_1 Plane [Djuric et al, 2004].....	53
Figure 3.9: Four Combination for Joint 2 Solution [Djuric et al, 2004].....	54
Figure 4.1: Overall System Structure.....	58
Figure 4.2: The Menu Structure of the GUI.....	61
Figure 4.3: The Default Window of the GUI.....	62
Figure 4.4: The World Coordinate System.....	62
Figure 4.5: An Example of Work Cell Objects.....	67
Figure 4.6: Class Hierarchy of the System.....	70
Figure 4.7: Tree Structure for a robot Arm.....	75
Figure 4.8: The Scene Graph of Kinematic Modeling for PUMA Robot.....	77
Figure 4.9: The Corresponding OpenGL Pseudocode.....	78
Figure 5.1: Kinematic Model for ABB IRB 6400.....	83
Figure 5.2: Simulation Path of Case Study One.....	84
Figure 5.3: Visualization for the point 1, solution 8 of Case Study One.....	90
Figure 5.4: Kinematics Model for ARCMate120iL.....	91
Figure 5.5: Simulation Path for Case Study two.....	92
Figure 5.6: Selection of Joint Coordinate System.....	96
Figure 5.7: Modification of the Robot Arm Lengths and Offsets.....	97
Figure 5.8: Translation and Rotation of the Robot.....	97
Figure 5.9: The Edit Dialog of Geometric Modeling for the Cube.....	98
Figure 5.10: The Pendent View and Inverse Kinematic Solution Dialogs of the robot...99	99
Figure 6.1: Robot Arm Control System.....	104
Figure 6.2: The overall architecture of the system.....	104

LIST OF TABLES

Table 1.1: PUMA D-H Parameters.....	10
Table 2.1: Geometric Primitive Types.....	43
Table 3.1: D-H parameters for Puma Kinematic Structure [Djuric et al, 2004].....	48
Table 3.2: D-H parameters for Fanuc Kinematic Structure [Djuric et al, 2004].....	48
Table 3.3: D-H parameters for the GPF Model [Djuric et al, 2004].....	49
Table 3.4: Various Orientations for WRIST [Djuric et al, 2004].....	55
Table 4.1: Common and Specific Functionality for 3dObject Class.....	71
Table 4.2: The Main Functions of the Math Library.....	72
Table 5.1: D-H Parameters for ABB IRB1400.....	83
Table 5.2: The Definition of Target Points of Case Study One.....	84
Table 5.3: The Joint Solutions for Point 1 of Case Study One.....	85
Table 5.4: The Joint Solutions for Point 2 of Case Study One.....	85
Table 5.5: The Joint Solutions for Point 3 of Case Study One.....	85
Table 5.6: The Joint Solutions for Point 4 of Case Study One.....	86
Table 5.7: The Joint Solutions for Point 5 of Case Study One.....	86
Table 5.8: The Joint Solutions for Point 6 of Case Study One.....	86
Table 5.9: The Joint Solutions for Home Position of Case Study One.....	87
Table 5.10: D-H Parameters for the Point 1, Solution 8.....	88
Table 5.11: D-H Parameters for ARCMate120iL.....	91
Table 5.12: The Definition of Target Points for Case Study Two.....	92
Table 5.13: The Joint Solutions for Home Position of Case Study Two.....	93

Table 5.14: The Joint Solutions for point 1 of Case Study Two.....	93
Table 5.15: The Joint Solutions for point 2 of Case Study Two.....	93
Table 5.16: The Joint Solutions for point 3 of Case Study Two.....	94
Table 5.17: D-H Parameters for the Point 2, Solution 7.....	95

CHAPTER 1

INTRODUCTION

This chapter presents an introduction to the principal concepts and technologies involved throughout this research. The first section introduces robotics and especially the Denavit-Hartenberg (D-H) matrix algebra approach for representing robot arm kinematics in detail. In the second section, the Unified Reconfigurable Open Control Architecture (UROCA) project and its latest progress is also briefly introduced. The third section overviews existing graphical robotic simulation systems, with emphasis on their characteristics. Through this, the motivation for developing a unified robotic kinematic simulation interface is explained. The objectives of this research are also described. Finally, the fourth section presents an overview of this thesis.

1.1 Introduction to Robotics

The Robot Institute of America defined that “a robot is a reprogrammable multifunctional manipulator designed to move materials, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks”.

Robotics is concerned with the study of those machines that can replace human beings in the execution of a task with regards to both physical activity and decision making. Robotics is truly a multidisciplinary field that includes mechanical and electronic engineering, computer science, and mathematics.

1.1.1 Introduction to Industrial Robots

An industrial robot consists of:

- 1) A manipulator
- 2) Actuators
- 3) Sensors
- 4) A control system.

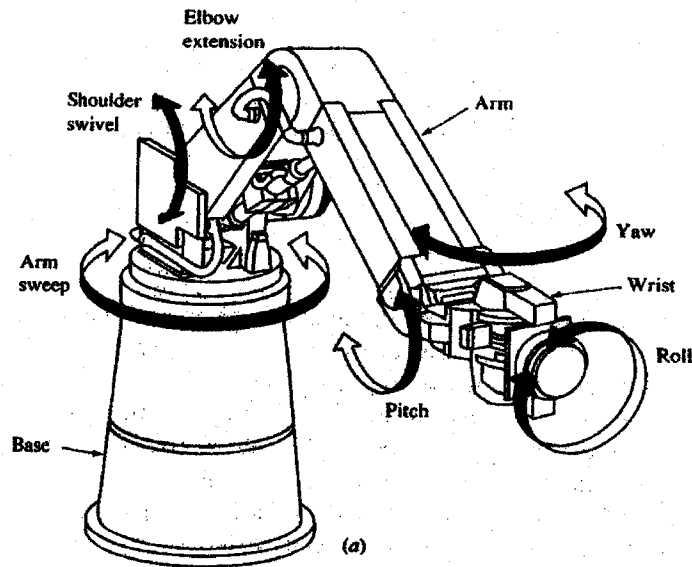


Figure 1.1: Cincinnati Milacron T Robot Arm [Fu et al, 1987]

A manipulator or mechanical structure consists of a sequence of rigid links connected by revolute or prismatic joints. Figure 1.1 illustrates an industrial robot manipulator. A manipulator has a supporting base, an arm that ensures mobility, a wrist that confers dexterity, and an end-effector that performs the desired task. The motion of the joints results in the relative motion of links.

Mechanically, a robot is composed of an arm and a wrist subassembly plus a tool. The arm generally can move with three degrees of freedom. The combination of the arm

movements positions the wrist at the workplace. The wrist consists of three rotary motions called pitch, yaw, and roll. The combination of the wrist motions orients the tools according to the configuration of objects for ease of pickup.

Each joint has its joint limit. For example, the joint limits of the PUMA 560 series robot arm are shown in Figure 1.2. The manipulator structure as well as their joint limits determine the work envelope that represents the portion of the environment which the manipulator's end-effector can access. A work envelope of ABC IRB1400 is produced in Figure 1.3 by Workspace software.

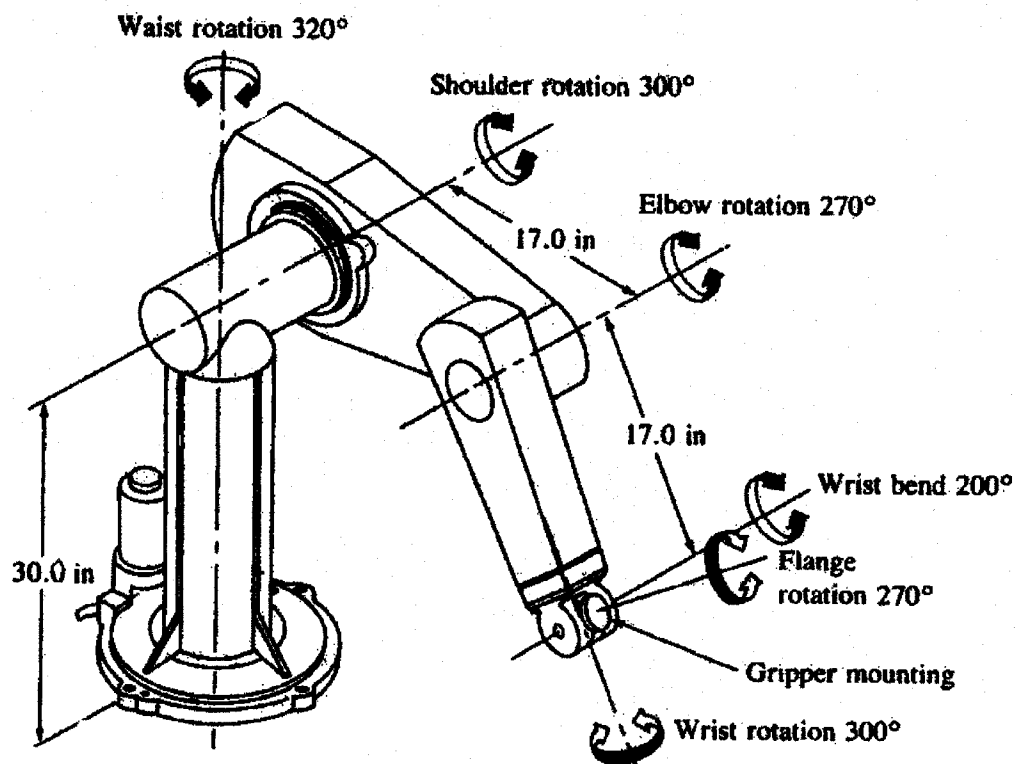


Figure 1.2: PUMA 560 Joint Limits [Fu et al, 1987]

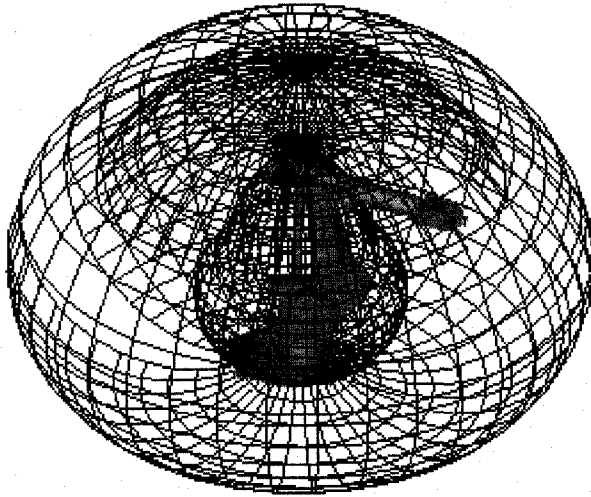


Figure 1.3: Work Envelope

Actuators actuate the joints to set the manipulator in motion; the motors employed are typically electric and hydraulic, and occasionally pneumatic. Sensors measure the status of the manipulator (internal sensors) and the status of the environment (external sensors). A control system is a special kind of computer, which enables control and supervision of manipulator motion.

The main difference between a robot and a numerically controlled machine tool is its versatility. The manipulator's end-effector can have different types of tool as well as the large workspace.

The main applications of industrial robots in a manufacturing process are material handling, arc or spot welding, and painting for vehicle bodies, machining, electronic assembly, water jet, laser and plasma cutting, measurement, and so on.

1.1.2 Kinematics

Robot arm kinematics deals with the analytical study of the geometry of motion of robot arm with respect to the fixed reference coordinate system as a function of time without regard to the forces/moments that cause the motion. Thus, it deals with the analytical description of the spatial displacement of the robot as a function of time, in particular the relations between the joint variable space and the position and orientation of the end-effector of a robot arm.

The direct kinematics problem is defined by the following: Given the joint angle variables $\theta_1, \theta_2, \dots, \theta_n$ and the geometric link parameters, what is the position and orientation of the end-effector of the manipulator with respect to a reference coordinate system?

The inverse kinematics problem is defined by the following: Given a desired position and orientation of the manipulator with respect to a reference coordinate system and geometric link parameters, can the manipulator reach the desired position and orientation? If so, how many different manipulator configurations will satisfy the same condition? What are the resulting joint angle variables?

A simple block diagram indicating the relationship between these two problems is shown in Figure 1.4

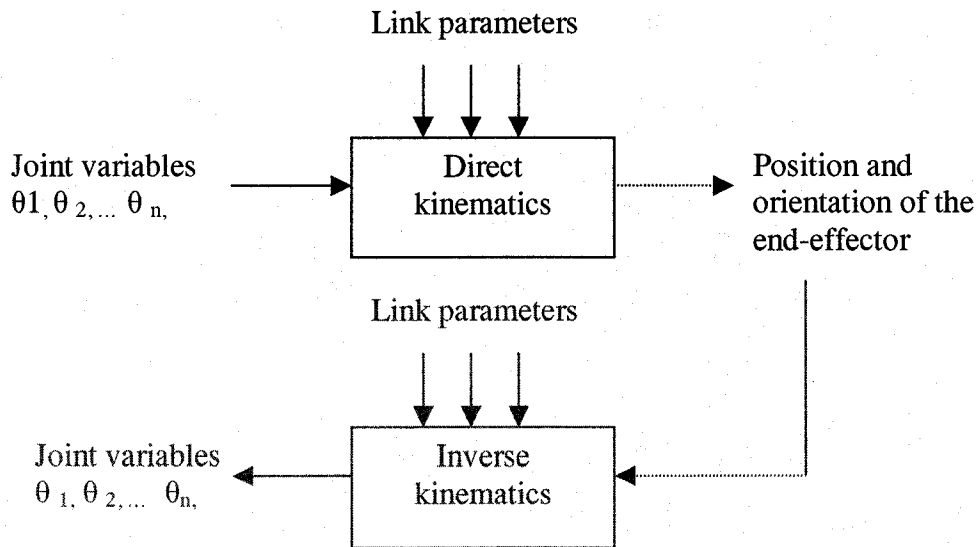


Figure 1.4: The Direct and Inverse Kinematic Problems

A mechanical manipulator consists of a sequence of rigid bodies, called links, connected by either revolute or prismatic joints. Each joint-link pair constitutes one degree of freedom as long as there are no closed loops. Hence for an N degrees of freedom manipulator, there are N joint-link pairs with Link 0 attached to a supporting base, and the last link is attached to a tool. The joints and links are numbered outwardly from the base. An example is illustrated in Figure 1.5.

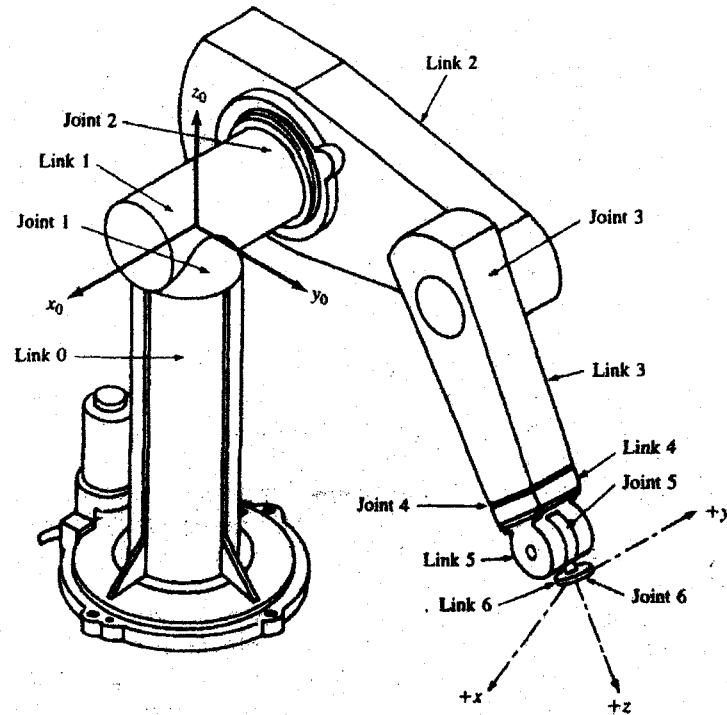


Figure 1.5: A PUMA Robot Arm Illustrating Joints and Links [Fu et al, 1987]

Since the links of a robot arm rotate or translate with respect to a reference coordinate frame, the total spatial displacement of the end-effector is due to the angular rotation and linear translations of the links. In 1955 Denavit and Hartenberg proposed a systematic and generalized approach of utilizing matrix algebra to describe and represent the spatial geometry of links of a robot arm with respect to a reference coordinate frame [Denavit and Hartenberg, 1955]. This method uses a 4x4 homogeneous transformation matrix to describe the spatial relationship between two adjacent rigid mechanical links and reduces the direct kinematic problem to finding an equivalent 4x4 homogeneous transformation matrix that relates the spatial displacement of the end-effector coordinate frame to the reference coordinate frame. The advantage of using the D-H representation is its algorithmic universality in deriving the kinematic equations of a robot arm.

To describe the translational and rotational relationships between adjacent links, Denavit and Hartenberg proposed a matrix method of systematically establishing a body-attached coordinate frame to each link of robot arm chain. A joint coordinate frame is established at the connection of two links and attached to the second link. The D-H parameters consist of a set of 4 numbers d , θ , a , and α , that describe the position and orientation of a link frame with respect to the preceding link frame along the chain. The D-H parameters definitions are given and used in this thesis as well as the visualization software.

Given two consecutive link frames on a robot manipulator shown in Figure 1.6, frames F_{i-1} and F_i , Frame F_i will be uniquely determined from frame F_{i-1} by use of the parameters d_i , θ_i , a_i , α_i .

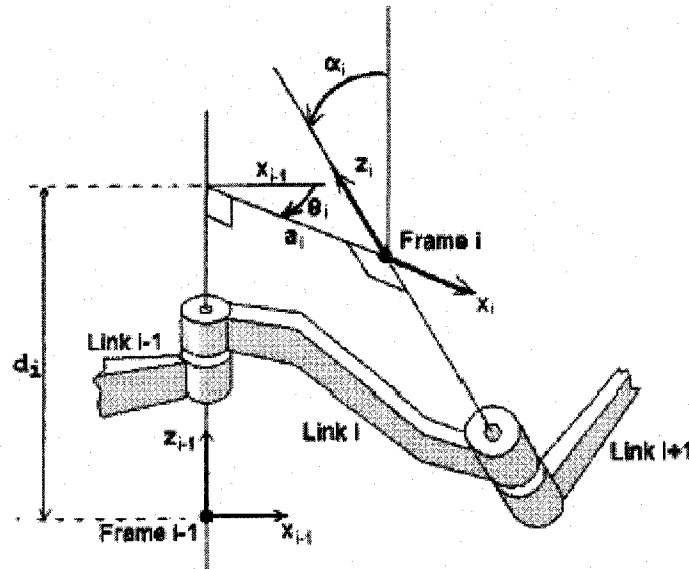


Figure 1.6: Link Coordinate Frame and its Parameters

The parameters are explained here.

- The z vector of any link frame is always on a joint axis.
- d_i is the distance along z_{i-1} axis to the point where the common perpendicular to z_i axis is located. d_i is constant if joint i is revolute and variable when joint i is translational.
- a_i is the offset length of the common perpendicular.
- θ_i is the joint angle from x_{i-1} axis to x_i axis about z_{i-1} axis. θ is variable when joint i is revolute, and constant when joint is translational.
- α_i is the offset angle from z_{i-1} axis to z_i axis about x_i axis.

The link parameters α_i , a_i determine the structure of the link and joint parameters d_i , θ_i determine the relative position of neighboring links. Figure 1.7 shows the link coordinate frame for a PUMA robot. Table 1.1 shows its D-H parameters for home position.

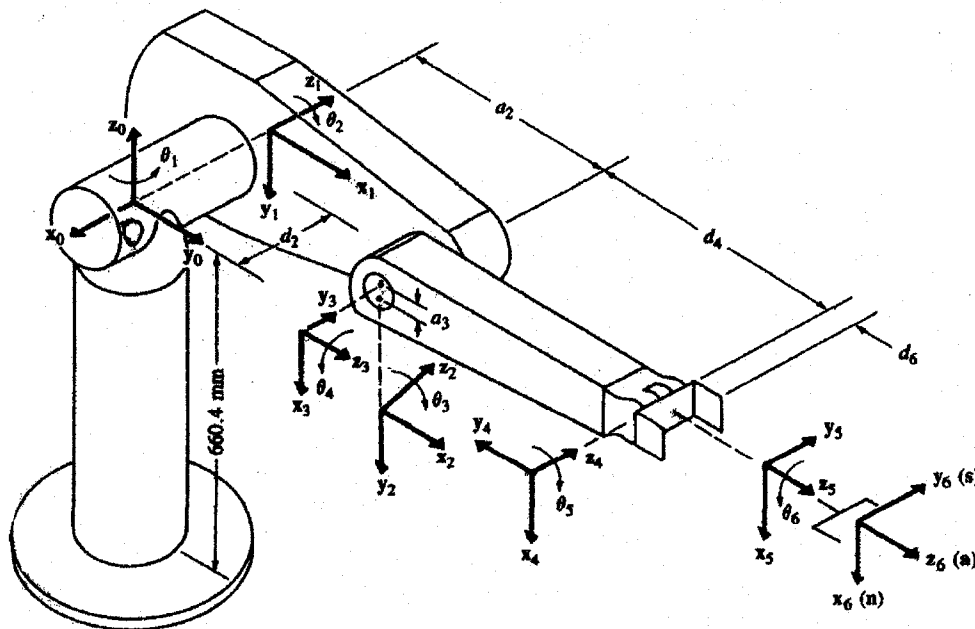


Figure 1.7: PUMA Link Coordinate Frame [Fu et al, 1987]

Joint i	θ_i	α_i	a_i	d_i
1	90°	-90°	0	0
2	0	0	431.8mm	149.09mm
3	90°	90°	-20.32mm	0
4	0	-90°	0	433.07mm
5	0	90°	0	0
6	0	0		56.25

Table 1.1: PUMA D-H Parameters

The mathematical description of robot manipulators is a table of D-H parameters. The table contains one row of four parameters for each link frame. The D-H parameters allow one reference frame to be located exactly with respect to the preceding link frame.

It is assumed that a link coordinate frame B is determined from a preceding coordinate Frame A by the four D-H parameters d , θ , a , and α . As shown in Figure 1.8, Frame B can be located by the process outlined here:

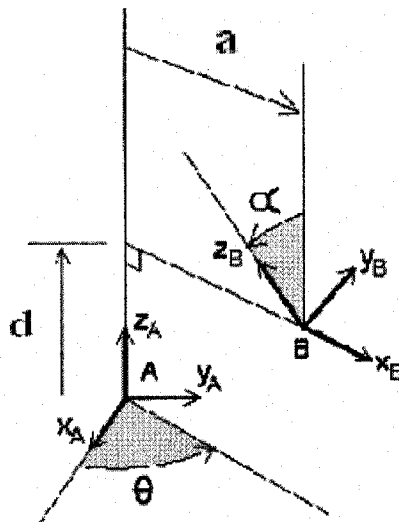


Figure 1.8: Determination of Link Frames from D-H Parameters

- 1) Find the direction of x_B axis by rotating x_A axis by an angle θ about z_A axis;
- 2) Move a distance d on z_A axis;
- 3) Move a distance a in the direction of x_B axis. The position reached is the origin of Frame B. At this point x_B axis is determined as well;
- 4) Rotate z_A axis about x_B axis by an angle α to determine z_B axis.

The y_B axis completes the right-handed coordinate system as required.

Each of these four operations can be expressed by a basic homogeneous rotation-translation matrix. Therefore, ${}^{i-1}A_i$, known as the D-H transformation matrix for adjacent coordinate frames i and $i-1$, is generated by the product of these four basic homogeneous transformation matrices.

$${}^{i-1}A_i = T_{z, d_i} T_{z, \theta_i} T_{x, a_i} T_{x, \alpha_i}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_i & -\sin \alpha_i & 0 \\ 0 & \sin \alpha_i & \cos \alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.1)$$

$$= \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The homogeneous transformation matrix 0T_i , which specifies the position and orientation of the i th coordinate frame with respect to the base coordinate frame, is the chain product of successive coordinate transformation matrices of matrices ${}^{i-1}A_i$

$${}^0T_i = {}^0A_1 A_2 \dots A_{i-1} A_i = \prod_{j=1}^i A_j, \quad \text{for } i = 1, 2, \dots, n \quad (1.2)$$

We define the tool coordinate frame as $[\mathbf{n}, \mathbf{s}, \mathbf{a}]$ and the position vector \mathbf{p} . For $i = 6$, the direct kinematic equation, that is, the position and orientation of the end-effector of manipulator with respect to the base coordinate system, is given by:

$${}^0T_6 = {}^0A_1 A_2 A_3 A_4 A_5 A_6 = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.3)$$

In general, the inverse kinematic problem can be solved by several techniques. The most commonly used methods are matrix algebraic, iterative, or geometric approaches. In Chapter 3, a geometric approach will be presented in detail.

1.1.3 Dynamics and Control

The dynamic model of a manipulator provides a description of the relationship between the joint actuator torques and the motion of the structure. Derivation of the dynamic model of a manipulator plays an important role for simulation of motion, analysis of manipulator structure, and design of control algorithms. Simulating manipulator motion allows testing control strategies and motion planning techniques without the need to use a physically available system. The analysis of a dynamic model can be useful for the mechanical design of prototype arms. Computation of forces and torques is required for designing joints, drives and actuators.

The problem of controlling a manipulator is to determine the time history of the generalized forces (or torques) to be developed by the joint actuators so as to guarantee execution of the commanded task while satisfying given transient and steady-state requirements. There are two types of robot control:

- 1) Joint space control: inverse kinematics transform the operational space to joint space, then control the joint variables.
- 2) Operational space control: directly control the operational space variables, inverse kinematics is embedded into the feedback control loop.

1.1.4 Application Programming

Robot application programming for industrial robots usually includes on-line and off-line programming [Biggs, 2003]. On-line programming, also known as teach and playback, uses either the walk-through or the lead-through method. The walk-through method involves teaching the robot by leading it through the motions the user wishes the robot to perform, while the information about position, velocity and other related variables is recorded by robot's control system. This recorded motion can be played back whenever required. For the lead-through method, the robot is moved to desired positions by actuating its drive mechanism. The position information is recorded by the teach pendant. The advantages of on-line programming are that it requires only a relatively small memory space and is simple to learn and suitable for simple tasks. The main disadvantage is that it is difficult to integrate sensory feedback information into the control system. In addition, if the task is changed, the whole manufacturing system is interrupted during robot programming [Fu et al, 1987].

Off-line programming can be prepared on a computer and downloaded to a robot controller without interrupting the production. Now usually the off-line programming systems offer graphical simulation platforms, in which the robots and other equipment as well as parts in the work cell can be modeled and animated. The main benefits of off-line programming include reducing downtime, better understanding the process through simulation, and decreasing the risk of damage to expensive equipment and injury to the operator. The off-line approach can deal with programs for complicated tasks, while it requires operator expertise and experience in dealing with the robot language and simulation programming environments.

1.2 Introduction to Unified Reconfigurable Open Control Architecture

A Unified Reconfigurable Open Control Architecture, UROCA, is under development by the research team at University of Windsor at IMS Centre for application to industrial robotic, CNC, and automobile systems [ElBeheiry and ElMaraghy, 2004]. The UROCA proposed architecture is a reconfigurable system that takes the advantages of different control structure types, thereby integrating them in a way to enhance the controller architecture design. UROCA reconfigurability deals with both control and software reconfigurations as well as searching for every possible approach for unifying both types of reconfigurations.

In order to unify the reconfigurable control process, similar steps are followed in UROCA approaches:

- 1) Commonalities among robotic, CNC, and automobile systems are exploited. This will avoid reinventing solutions to problems that have been tackled before.
- 2) It employs software modules that are highly reusable within groups or families of similar patterns.
- 3) Moreover, UROCA will favor the design of highly modular modules which may be combined with each other to produce new system, probably in environments quite different from the one for which they were originally developed.

UROCA is intended for use with different industrial machines like robotic, CNC, and automobile systems. The ultimate goal is to have a machine-independent, application-independent architecture. The transition from the technology of open controllers to the technology of universal controllers is emphasized.

As UROCA is intended for controlling a wide variety of industrial machines, it has the feature of easy reconfiguration from one machine to another as well as from one application to another with the lowest amount of change. The Unified Kinematic Modeler and Solver (UKMS) [Djuric et al, 2004] represents one of the most important modules of UROCA. They invented a Generic Puma and Fanuc (GPF) kinematic model for most industrial robots. The GPF model can be reconfigured from one kinematic structure to another by using configuration parameters. A Unified Geometric-Based Solution was presented to solve the inverse kinematics problem based on the GPF model.

1.3 Motivation and Objectives

Simulation and off-line programming of industrial robots are relatively mature technologies. Robotic simulation software plays an important role in robotics research in many areas such as robot design, forward and inverse kinematics analysis, dynamics, control, path planning, etc. for both commercial and educational purposes. There are several types and strategies of graphical robotic simulation software available in commercially available systems.

According to the relationship between control and simulation systems, the robotic simulation packages can be divided into two categories. One is the simulation systems separated from the control systems. The other is the simulation systems integrated with the control systems. For the first type, usually the control systems are vendor-oriented, closed structures which provide common programming languages such as Pascal, C, BASIC, etc with robot control libraries or a robotic command set for application programming.

Development of robotic simulation has been driven by speeding program generation; improving program accuracy; verifying collision-free path; predicting cycle time; and improving robot efficient applications. However, industrial robot control systems and robot simulation systems are often developed by different vendors and are programmed with different, incompatible programming languages [Bernhardt, 1995]. In this approach many algorithms that solve common tasks for simulation and robot control are programmed twice and independently. They appear in both systems. Therefore the

simulation still does not reflect a desired reality. The ideal simulation is that programs developed off-line can be loaded onto and run by real robot controllers without having to correct them in the shop floor.

Along with the advancement of computer technology and manufacturing systems, open structure PC-based control systems came about. The robotic control platforms can achieve a “one-box” solution, which means that the platform is integrated on a single PC, using a single operating system and a single programming language to implement the servo control, 3D simulation, and robotic application development [Loffler, 2001, 2002]. A strong correspondence between the robot simulation and the actual robot controller can be achieved by sharing the same algorithms.

Both types of robot simulation systems have a common characteristic. Different robot systems have different kinematic structures for simulation. Some simulation systems can just be used for a few robot types. For example, RoboCell is designed for SCORBOT-ER 4u and SCORBOT-ER-2u [Intelitek Inc., 2003].

Therefore, in order to implement the reconfigurable control strategies and ease application programming, the commercial robotic simulation packages cannot satisfy the specific requirements of the UROCA project. Thus, the development of a Graphical User Interface (GUI), that provides 3D robotic kinematic modeling and simulation is necessary.

The main objective of the proposed research is not only to provide a GUI software package to implement the reconfigurable kinematic modeling and simulation for most industrial robots, but also to create a software platform that has the features of ease of use, extensibility, portability, and reusability.

Several objectives have been defined for this research. They are:

- 1) Create a software platform for 3D kinematic robotic modeling and simulation.
- 2) Design a GUI for visualization of robot kinematics configuration based on Generic Puma-Fanuc model.
- 3) Automatically generate the D-H parameters from the kinematic modeling.
- 4) Build a library module to implement vector, matrix, and kinematics calculation.
- 5) Design a Dynamic-Link Library (DLL) to unify the geometric modeling.
- 6) Create the robot simulation.
- 7) Verify the results.

1.4 Thesis Overview

This thesis is organized as follows:

Chapter 1 provides an introduction to robotics as well as the main technologies involved. The UROCA project is also introduced. Then the motivation and objectives of this research are presented.

Chapter 2 discusses the main functionality usually implemented in graphical robotic simulation software packages, and presents a literature review of many packages,

classified according to their control systems. The theories and technologies for implementing a graphical robotic simulation are briefly explained as well.

In Chapter 3, the details of the Generic Puma-Faunc (GPF) kinematic model for most industrial robots and a generic solution module using a geometric approach for solving the inverse kinematic problem based on the GPF model are reviewed. Also, the design requirements of the Graphical User Interface (GUI) for implementing the UKMS are described.

Chapter 4 illustrates the hardware and software structure of the GUI, its main functions, and an explanation of software implementation based on the theories and requirements described in Chapter 2 and Chapter 3 respectively.

Simulation examples and evaluation results are shown in Chapter 5. The conclusion and future research are given in Chapter 6.

CHAPTER 2

OVERVIEW OF GRAPHICAL ROBOTIC SIMULATION SYSTEMS

Graphical robotic simulation and off-line programming of industrial robots are today relatively mature technologies. Robotic simulation software plays an important role in robotics research in many areas such as robot design, forward and inverse kinematics analysis, dynamics, control, path planning, etc. for both commercial and educational purposes. There are many graphical robotic simulation software packages available in the market. Within this chapter, first, the common functionality implemented in the graphical robotic simulation software packages is described. Second, a literature review of these packages classified according to their control systems is presented. Also, characteristic analyses are provided for each category to summarize the discussed literature. Finally, the theories and technologies for developing a graphical robotic simulation system are explained.

2.1 The Main Functionality

Generally speaking, simulation is a process of modeling an existing or hypothetical system to examine its properties and behavior. Simulation systems are commonly implemented as software.

Simulation of industrial robots has become an important means for increasing efficiency of robot applications. High precision simulation decreases development costs and increases reliability of safety-critical industrial automation systems.

Robotic simulation software packages are generally 3D graphics-based interactive tools for designing, programming and optimizing the robotics applications through simulation and analysis [Orady et al, 1997]. These packages consist of a robot builder and motion simulator of robot joints through graphic representation. Other components that can be modeled are machine tools, coordinated measuring machines, conveyors, grippers, parts, etc. and the motion of their movable elements can also be simulated. Robots or devices can be constructed using a graphics package as a set of elements of graphic solids that can be assembled in one overall component. Kinematic relationships between the links are then established to form a kinetically defined device. A set of robots and devices can then be used to construct a work cell. Robots, devices and parts are assembled in a certain work cell layout. After constructing the work cell, locations are created to construct motion paths, and to write control logic for the operation of the cell, and then an off-line program (OLP) for the robots can be developed while the robot path motion is simulated. The OLP could be translated to a teaching pendant file, or a robot specific programming language such as "Karel" for FANUC robots [Orady et al, 1997]. These programs can be directly downloaded to the real controller to run the robots and devices.

The robotic simulation systems differ significantly from traditional CAD tools in that they allow the study of geometries, kinematics, dynamics and motion planning.

When a production process is complex, these simulation application tools are beneficial and necessary as in the case of spot welding, arc welding, and painting, assembly, machine tool path verification, inspection and human factors.

The main functions of graphical robotic simulation packages are summarized:

- 1) 3D modeling of robots, machine tools, coordinated measuring machines, conveyors, grippers in the work cell.
- 2) Assembling the robot or device elements in one component.
- 3) Creating forward and inverse kinematics for robots.
- 4) Creating mechanisms of other devices.
- 5) Automatically generating geometric paths.
- 6) Checking the robotic work envelope.
- 7) Collision detection.
- 8) Robot calibration.
- 9) Off-line programming.
- 10) Simulating and optimizing the work cell.

2.2 Literature Review

2.2.1 The Simulation Systems Separated from the Control Systems

Grasp2000

Grasp2000 [BYG Systems Ltd, 2002], by BYG systems Ltd, is a true 3D simulation tool, based on accurate 3D geometry, process parameters and a library of industrial

robots. Grasp2000 enables the creation of accurate 3D models, and real-time interactive simulations for cell layout design, planning, optimisation, and cycle time calculation. As a tool for off-line programming, the instructions can be automatically translated into the required native robot language.

Grasp2000 can generate specific application menus for arc welding, palletising and spraying. The software will find applications in PC-based cell layout and design, analysis, offline programming and process planning throughout the full range of Toshiba SCARA robot applications.

An important factor in off-line programming is the presence of inherent inaccuracies in most robots. Grasp2000 uses in-depth mathematical calculations to calibrate both the robot and 3D model to match the real world. It only requires the demonstration of a number of robot poses, which are then read into Grasp2000 and analyzed by the calibration software without external measuring equipment.

An optional module for discrete event simulation extends Grasp2000's application areas to factory simulation, warehousing, logistics and materials handling.

CimStation Robotics (CSR)

CSR [Applied Computing & Engineering Ltd, 2005], is powerful 3D simulation software that enables manufacturing engineers to quickly simulate and evaluate automation concepts to determine the cost, feasibility and performance of a proposed

robotic system. Using existing in-house CAD data and AC&E's library of commercial robots and accessories to create a detailed simulation of the proposed manufacturing system, CSR accurately simulates interactions between work cell components to optimize equipment selection, fine-tune equipment positioning, and maximize production throughput.

The system is the most comprehensive and easy-to-use robotic simulation tool available and works completely off-line, eliminating the risk of damage to equipment and freeing robots for round-the-clock production. CSR can be purchased in a modular fashion.

Specialized application solutions tailored to the requirements of a particular robotic task provide advanced functionality and ease of use for painting, spot welding, arc welding, polishing, assembly and press operations.

IGRIP

Interactive Graphics Robot Instruction Program (IGRIP) by DELMIA [Cheng, 2000], is an interactive, 3D graphic simulation tool for designing, evaluating, and off-line programming of robotic work cells. Actual robotic/device geometry, motion attributes, kinematics, dynamics, and I/O logic are incorporated to produce extremely accurate simulations. IGRIP optimizes critical factors such as robot motion planning, cycle time prediction, collision detection, calibration, and multiple I/O communication.

The several specific task software modules consist of UltraArc, UltraSpot, UltraPaint, and UltraFinishing, which are designed specially for arc welding, spot welding, painting, and surface finishing work cell applications respectively. Other applications include research and development, articulated design, flexible manufacturing system simulation, nuclear/hazardous duty automation, and general-purpose simulation.

Work cell components can be created in the integral CAD package or imported from other CAD packages via IGES, DXF, and direct translators. A built-in surface modeling package provides modification and/or optimization of imported surface data.

EASY_ROB

EASY-ROB [Anton et al, 2001], 3D Robot Simulation Tool was written in Visual C++ under the Windows operating system. In order to create high quality and high speed rendered images, the graphical capabilities of OpenGL are used. EASY-ROB is a complex and comprehensive modeling and simulation tool. It is especially designed to fulfill requirements for several industrial robotic applications as well as for educational purposes.

The EASY-ROB Basic Model allows the planning and designing of robotic work cell layouts consisting of a robot, tool and environment. A simple 3D CAD system is provided to create basic geometric parameterized primitives like cubes, cones, cylinders, pyramids, etc. In addition, a CAD interface is available to import other 3D formats such as STL. Created and imported geometries are assigned to the robot group to active or

passive joints, to the tool group or to the environment group. Using a 3 button mouse, each geometry can be translated and rotated about its axis, or the operator can enter absolute or relative Cartesian values to set the Cartesian location. A modification of the view point (pan, tilt, zoom in and zoom out) in full shaded mode allows various world views.

The robot motion can be programmed using EASY-ROB standard program commands. A special Teach Window supports the user in writing robot motion programs. The built-in motion planner is implemented for the motion types: Point to point (PTP), Linear (LIN) and Circular (CIRC). The orientation interpolation for the LIN and CIRC motion type is realized for variable, fixed, tangential and quaternion modes. Several on-line output windows allow the operator to monitor robot joint values, Cartesian TCP location, as well as simulation states such as cycle time, step size, override, etc. All data is saved into documented ASCII text files.

Workspace

Workspace, described in [Owens, 1994], [Flow software technologies, 2002] has been developed by a team led by John Owens as the world's first industrial robot simulation software package. Commercially released in 1989, it has been continuously updated over the last decade.

In addition to a library over 140 industrial robot 3D models available to the user, the 3D CAD modeler can create 3D solid objects using Constructive Solid Geometry and

surface objects such as Bspline, Parametric, and Bezier surfaces. The 3D objects also can be imported from other CAD system via SXF or IGES file formats.

The movement of any mechanism may be modeled using a kinematics modeler. The mechanism may have any number of joints in any serial or tree-structure combination. Conveyors, automatic vehicle, and other independently moving objects may also be modeled. Positions and paths for the robot tool to target may be defined in several ways, such as by use of the teach pendant, by clicking the mouse on the screen, or by using geometry points.

Workspace can be used to create and simulate robots in the native language of the robot. For example, users of Fanuc robots may write robot programs in Karel, ABB robot users may write programs in ARLA, or Visual Basic can be used just for simulation. Therefore, there is no need for translating the simulation language to the robot language. It is also possible to transfer existing robot programs from the robot control to Workspace for optimization.

In addition, the simulation can be replayed in real time. Calibration and dynamics modules are also available.

PIN

A graphical robot simulation and off-line programming system, called PIN [Dai and Kampker, 1999], is a PC-based robot simulation and off-line programming system that

runs under the Windows operating system and applies OpenGL for rendering 3D graphics. It has been designed for the welding industry to meet the demands of small and medium sized enterprises allowing the user to completely generate robot programs interactively. One of the key programming functions includes macro programming techniques that are based on the idea of combining often repeated actions such as sequences of torch motion, Tool Centre Point measurement, sensor-based calibration and search operation, etc. An icon-oriented macro editor is another powerful tool. After successful simulation, the robot controller code can be downloaded to the robot controller via an RS232 standard serial interface on a PC.

CROBOTS

CROBOTS [O'Leary, 1998] is a CAD based robot simulation tool written in the AutoLisp programming language. The software can be used in the design, application and programming of educational and industrial robots. CROBOTS runs inside AutoCAD R14. Therefore, the impressive graphics capability of AutoCAD and the custom developed tools for revolute robot model creation, robot operation cycle planning and robot controller simulation are combined together. This program can perform 3D solid geometric modeling, defining a desired robot trajectory, and 3D simulation of the torque and dynamic response of a robot displayed in 3D graphics output.

ROBOSIM

ROBOSIM [Koseeyaporn, 2003], implemented cooperatively between Vanderbilt University and NASA's Marshall Space Flight Cent, is a robot simulation package for

educational purposes. ROBSIM runs under the Windows operating system. The graphical model utilizes OpenGL for rendering the simulated objects. The LISP programming language based on COMMON LISP is employed to program modeling and simulation of robots and objects by either typing in the interactive command line interface or loading into the simulator via a text-based LISP file. The useful educational simulation tool covers many areas such as path planning programming, robot modeling, forward and inverse kinematics, collision detection and avoidance, and 3D transformations.

RoboCell

RoboCell [Intelitek Inc., 2003] provides a full range of industrial functions in an intuitive interface designed for training environments. RoboCell lets students create, program, simulate and control the entire operation of robotic work cells and flexible manufacturing systems.

This software package integrates four components:

- 1) SCORBASE, a full-featured robotics control software package, which provides a user-friendly tool for robot programming and operation.
- 2) A Graphic Display module that provides 3D simulation of the robot and other devices in a virtual work cell.
- 3) CellSetup, which allows a user to create a new virtual robotic work cell, or modify an existing work cell.
- 4) 3D Simulation Software Demo to demonstrate RoboCell's capabilities.

RoboCell integrates SCORBASE robotic control software with interactive 3D solid modeling simulation software. RoboCell's virtual robots and devices accurately replicate the actual dimensions and functions of the equipment in the robotic work cell. Students can teach positions, write programs and debug robotic applications offline before executing them in an actual work cell. RoboCell allows students to experiment with a variety of simulated work cells, even if the actual work cells do not exist in the lab. Advanced students can even design 3D objects and import them into RoboCell for use in virtual work cells. But the SCORBASE control software is designed just for SCORBOT-ER 4u, and SCORBOT-ER-2u.

The main disadvantages of this type of simulation systems are:

- 1) Industrial robot control systems and robot simulation systems are often developed by different vendors and are programmed with different, incompatible programming languages. In this approach many algorithms which solve common tasks for simulation and robot control are programmed twice and independently. They appear in both systems. Experience has shown that this results in large deviations of simulated and actual behavior and inaccurate cycle times [Bernhardt et al, 1994], [Anton et al, 2001]
- 2) Most simulation systems need a translator from simulation programming to robot control programming. Even though there is no need for translating the simulation language to the robot language in Workspace, different native robot languages are used for different robots.

2.2.2 The Simulation Systems Integrated with the Control Systems

The Cimetrix Open Development Environment (CODE)

CODE [Cimetrix Inc., 2002], developed by Cimetrix, inc., is a family of open architecture machine modeling and motion control software products designed to control the most challenging multi-axis machine control applications. CODE contains both a powerful, easy-to-use offline simulation development environment (CIMulation) and a robust, real-time motion and I/O control system (CIMControl). Since the same application runs with CIMulation and CIMControl, software applications written and tested using CIMulation are guaranteed to work with CIMControl. CODE architecture is illustrated in Figure 2.1.

CODE has been successfully deployed on a wide range of demanding applications in various industries from surface mount (SMT), semiconductor, and electronic assembly to multi-axis robots, packaging and machine tools. Applications can be developed using computer languages such as C++, Visual Basic, Delphi, or any IEC 1131 PLC languages such as ladder logic or flow charts. Here two examples using CODE to develop control and application are introduced.

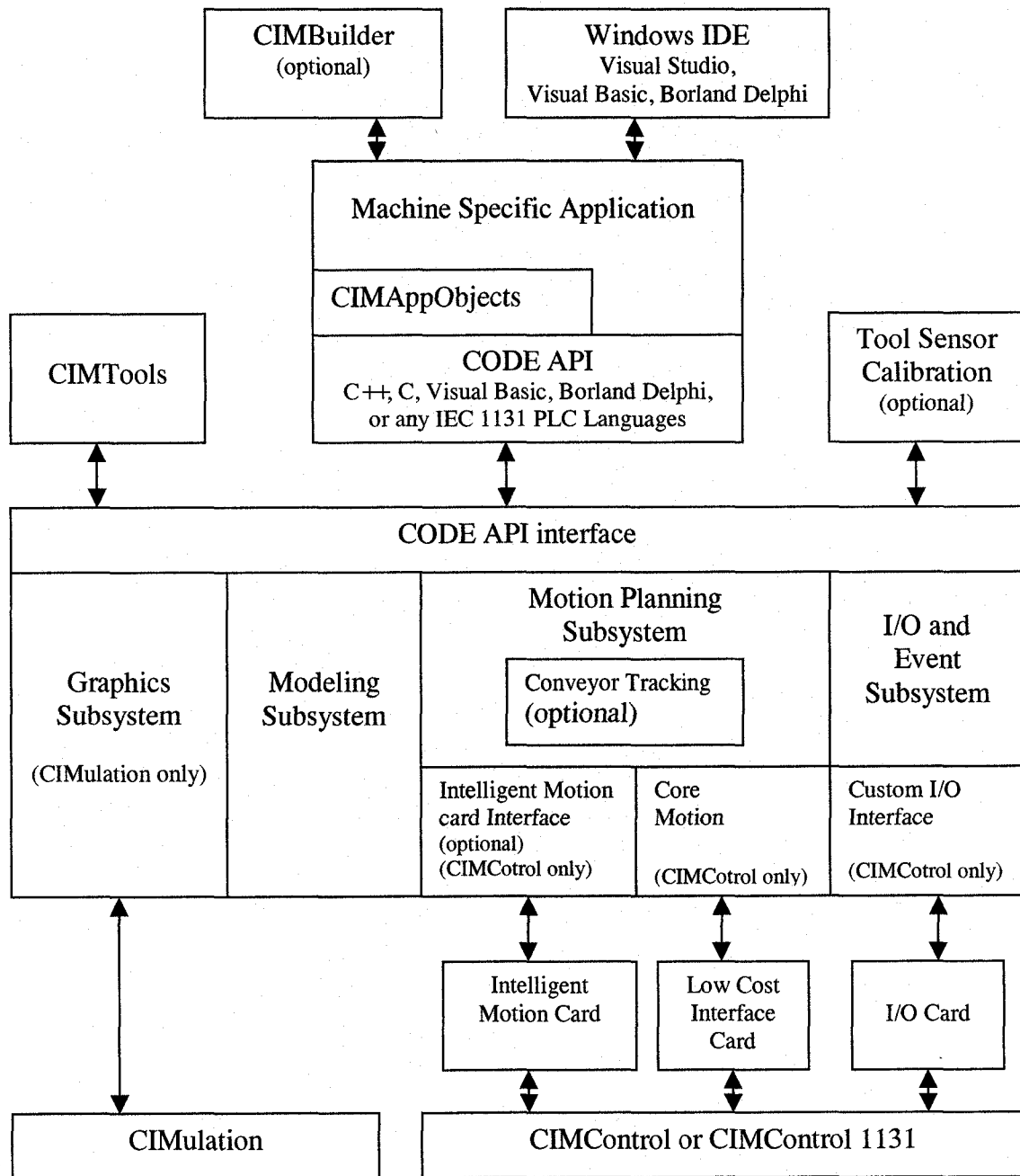


Figure 2.1: CODE Architecture Modified from [Cimetrix Inc., 2002]

A **PC- Based Open Robot Control (PC-ORC)** [Hong et al, 2001], constructed based on a modular and objected-oriented approach, can reconfigure the control system in various production environments. The PC-ORC system was based on the OSACA (Open

System Architecture for Controls within Automation systems, proposed by European countries) reference model and incorporated the commercial simulation environments CODE into its architecture for easier programming and verification.

Figure 2.2 depicts the hardware structure of the PC-ORC application to a SCARA robot. It includes a hardware platform PC, a motion controller PMAC, and a vision board.

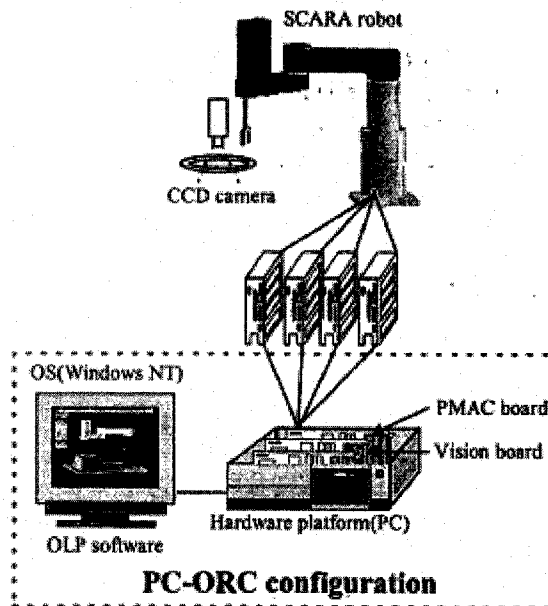


Figure 2.2: Hardware Configuration of the PC-ORC [Hong et al, 2001]

The application software modules are integrated with application objects, the CODE system, and hardware and operating system module. Also the TCP/IP protocol socket provided in C++ is used to read and write the data among objects of the application module. The overall structure of the PC-ORC is shown in Figure 2.3

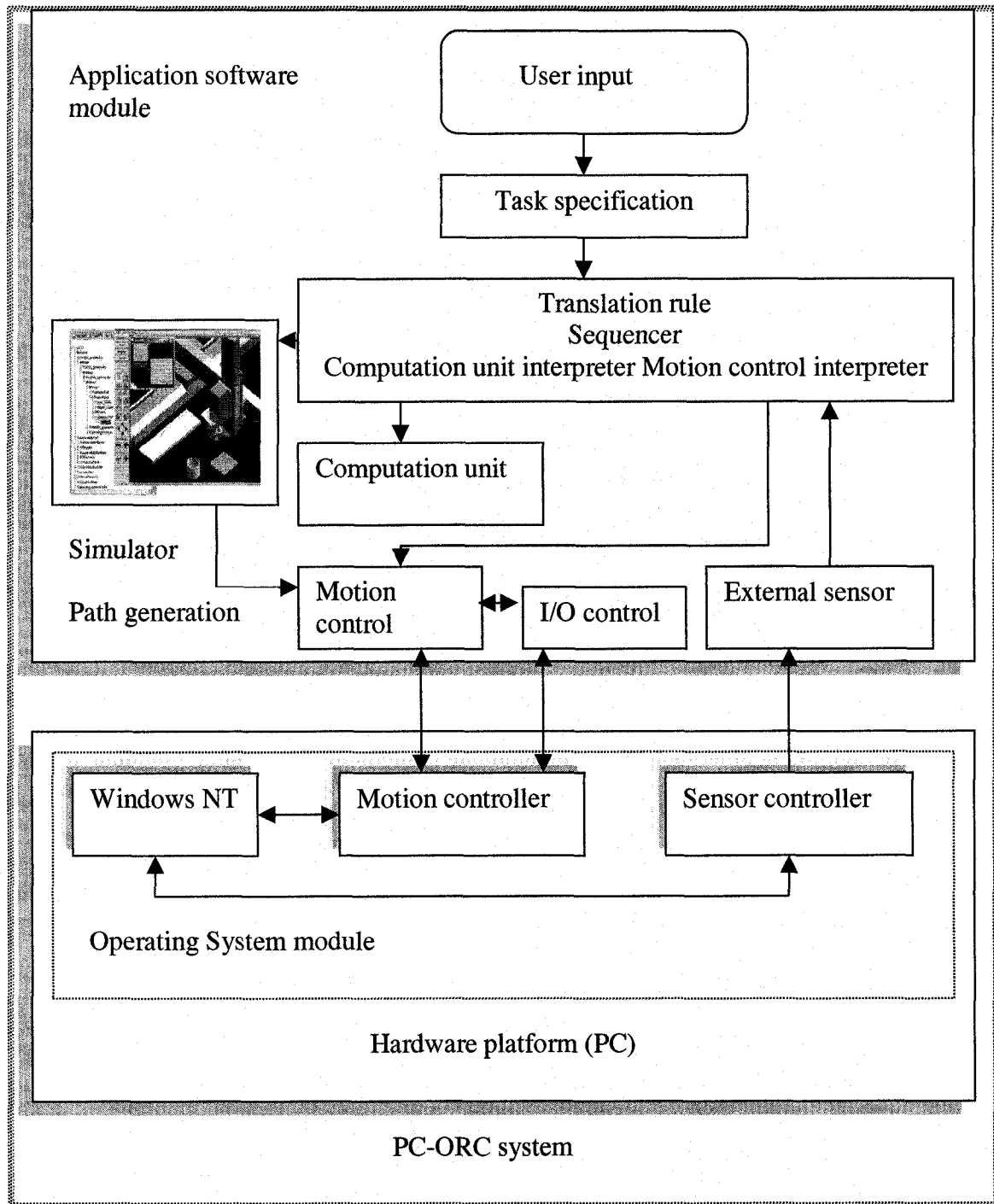


Figure 2.3: Overall Structure of the PC-ORC Modified from [Hong et al, 2001]

A virtual SCARA robot and its operation environment are modeled and simulated according to the set of robot motions specified by the operator using the CODE system. During the simulation, the robot path is updated and consequently sent to the PMAC for

motion control. It is found that virtual simulation can lead to enhanced efficiency of robot task planning and adaptation to the new manufacturing environment.

[Stringham, 1999] explores the use of the CODE simulation package throughout the life cycle of an assembly cell, including project definition, development and maintenance, and discusses the benefits that simulation can provide. Cimatrix Inc. provides a very unique simulation solution.

CODE provides a symmetric client/server architecture that runs on Windows NT. The client is the assembly application software. Two versions of the server are available, a simulation server (CIMulation) and a control server (CIMControl). Using standard programming languages such as C/C++ or Visual Basic, the client assembly application sends motion and I/O commands to a server. If the server is a simulation server, then the commands cause the graphical display to show the results. If the server is a control server, then the machine performs the specified commands. The application can be written such that at run time it can connect to either a simulation server for demonstration or testing, or it can connect to a control server to drive the actual assembly cell. The simulation and control servers are said to be symmetric, because they both have the same interface and behave in the same way; a single client process can talk to either. There is no need for the assembly application to be translated into the language of the controller before being able to run online.

In addition, since the application is written in standard languages and runs on Windows NT, it can utilize industry standard software like database and statistical process control packages available from a wide selection of vendors, and also implement the HMI as a user friendly GUI. The application is also free to take advantage of a wide range of third party hardware cards, including numerous vision cards, which are supported on Windows NT. Because the same application is used for both simulation and control, many more aspects of the software can be tested off-line than in traditional simulation environments. Correctness of the HMI, database queries and updates and other interfaces can all be tested and verified off-line.

Robotic Platform of “one-box” Solution

[Loffler et al, 2001, 2002] described the design and implementation of the Robotic Platform, which achieved a “one-box” solution by creating a very slim and optimized object-oriented design. The robotic platform implements all components such as servo control, trajectory generation, 3D simulation, a graphical user interface on a single standard PC, with a single programming language C++, and on a single operating system QNX Real-Time . This design leads to an open architecture that is less complex, easier to use, and easier to extend.

The class hierarchy diagram is shown in Figure 2.4. The class categories have:

The Core Classes: The classes `RoboticObject`, `FunctionalObject`, and `PhysicalObject` build the basis of all robotic objects. The classes `RoboticPlatform` and `ObjectManager` contain functionality for overall management of robot control programs.

Generic Robotic Classes which are derived from the core classes and cannot be instantiated. Instead, these classes serve as base classes that implement common functionality while also presenting a generic interface to the programmer.

Specific Robotic Classes: Derived from the generic robotic classes, these classes can implement a specific hardware or a specific functional component.

The ControlProgram Class: This class is part of QMotor system. All classes that require a real-time control loop are derived from the ControlProgram class.

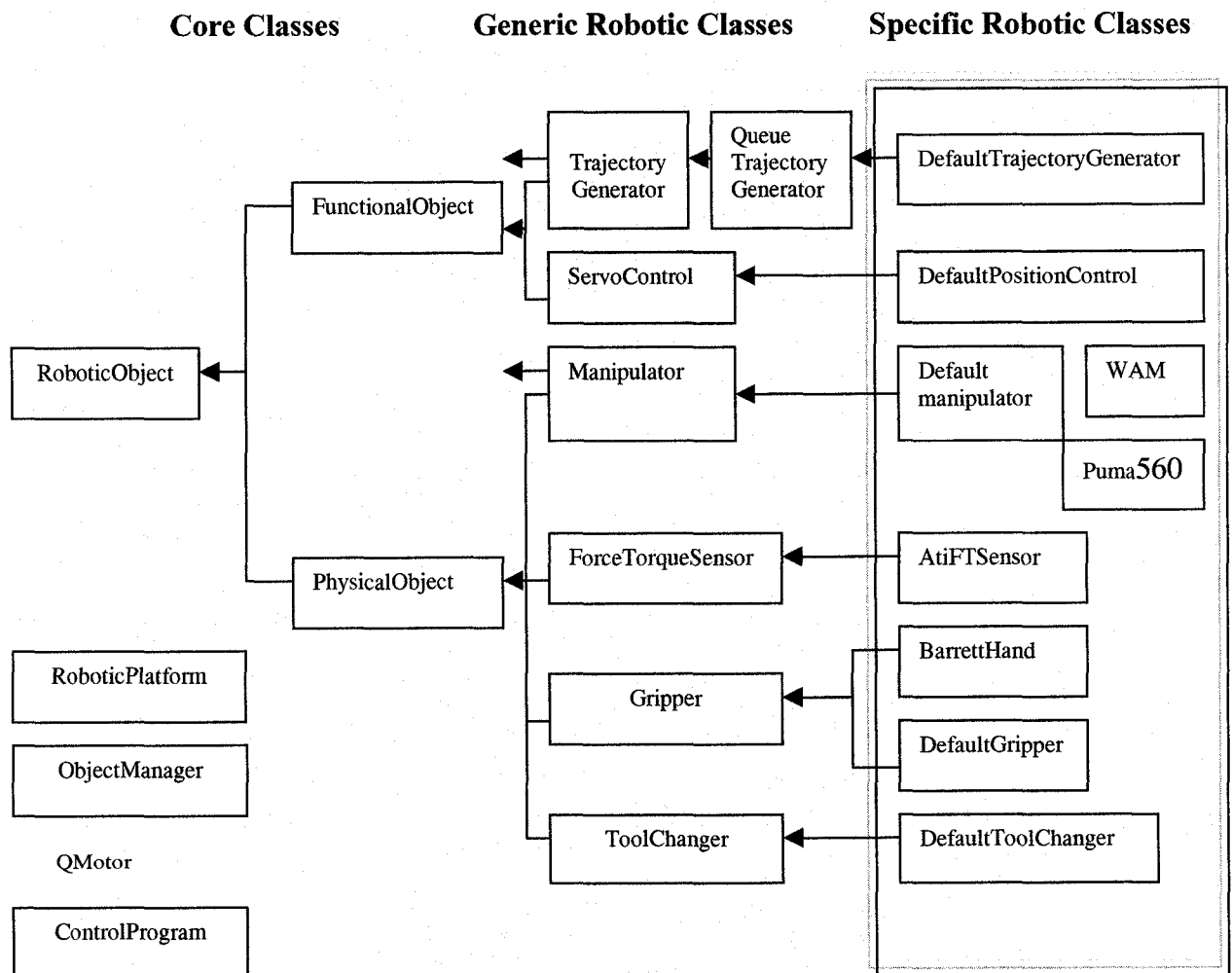


Figure 2.4: Class Hierarchy of the Robotic Platform [Loffler et al, 2001]

In a robot control program, the user instantiates objects from classes. As soon as objects are created, the user can employ their functions. The object manager maintains a list of all currently existing objects. The Scene View is the default GUI of the Robotic Platform. It contains windows to view the 3D scene of the robotic work cell and a list of all objects. The overall runtime architecture is illustrated in Figure 2.5.

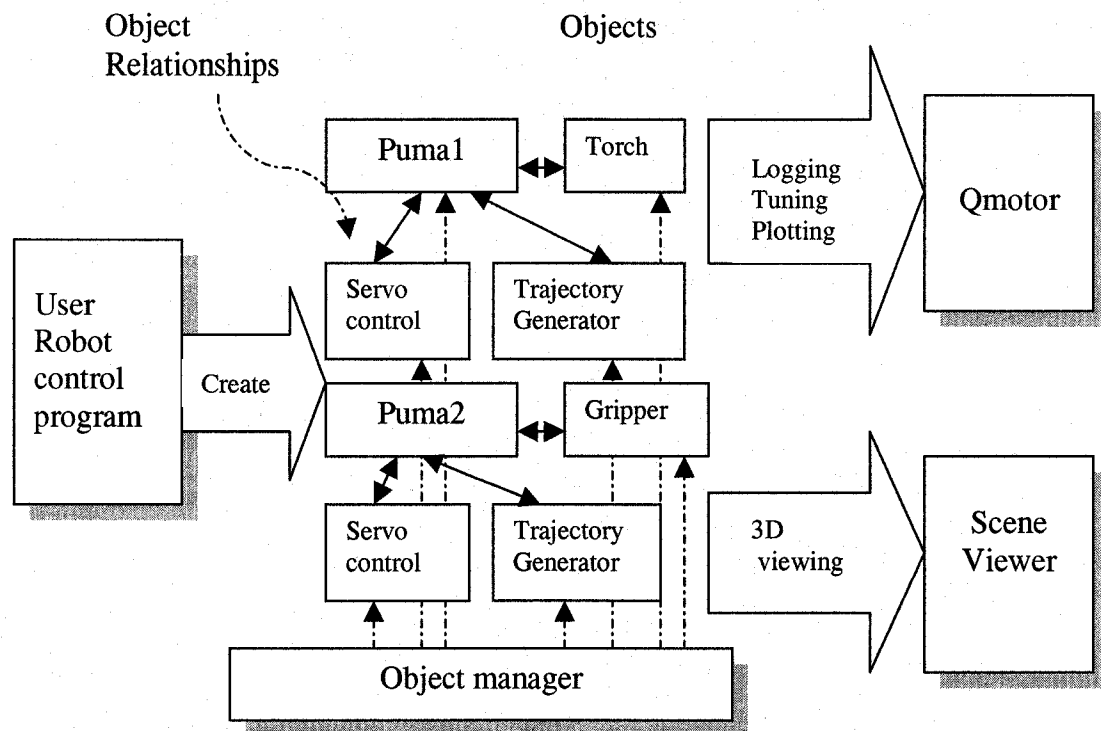


Figure 2.5: Run-time Architecture of the Robotic Platform [Loffler et al, 2001]

From the simulation systems integrated in the control systems reviewed in this section, it is obvious that this type of graphic robotic simulation system has some significant advantages compared to the first type.

- 1) Because they are integrated on a single PC, using a single operating system and a single programming language to implement the servo control, 3D simulation, and

robotic application development, a strong correspondence of robot simulation and actual robot control can be achieved by sharing the same algorithms. Therefore, simulation can get the exact cycle time and eliminate the path deviation between the simulation control and the real robot control.

- 2) The translation between the simulation system and the robot control system is not necessary.
- 3) A homogeneous non-distributed architecture is much smaller and simpler than a distributed inhomogeneous architecture. It is easier to configure, easier to understand, and easier to extend. Simplicity is critical with regard to reuse of the platform for different applications.
- 4) All components of the system are open for extensions and modifications. Flexibility is achieved through all levels.
- 5) A high integration achieved on the single platform allows for simpler and more efficient cooperation between components. Communication between the components has little overhead and is often implemented by just a function call.

2.3 The Theories and Technologies

2.3.1 Introduction to Computer Graphics

Computer graphics is concerned with all aspects of producing pictures or images using a computer. The field began more than 40 years ago with the display of a few lines on a Cathode-ray tube (CRT). Now images that are nearly indistinguishable from photographs can be generated.

The combination of computers, networks, and the complex human visual system, through computer graphics, has led to new ways of displaying information, seeing virtual worlds, and communicating with both other people and machines. The applications of graphics include: display of information, visualization, CAD design, simulation and animation, and user interfaces.

The interface between an application program and a graphic system can be specified through a set of functions that resides in a graphics library. These specifications are called the application programmer's interface (API). The graphics library is a black box whose properties are described by only its inputs and outputs. Nothing needs to be known about its internal working. We can take the simplified view of inputs as function calls from a user program and outputs as primitives displayed on our CRT screen. The application programmer's model of the graphics system is shown in Figure 2.6.

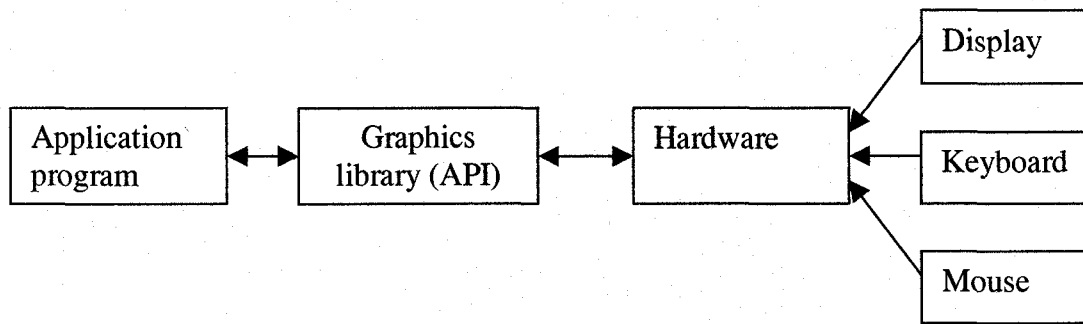


Figure2.6: Application Programmer's Model of Graphics System

The standard 3D graphics libraries perform lighting, shading, texture mapping, hidden-surface removal and animation on a Windows platform. Their main features include texture mapping, z-buffering, double buffering, lighting effects, smooth shading, material properties and transformation matrices. The video board built-in graphics library routines can speed up a program twenty to fifty times.

The synthetic-camera model is the basis for a number of popular APIs, including OpenGL, PHIGS, Direct3D, VRML, and JAVA-3D. We need functions in the API to specify objects, viewer, light sources, and material properties. A good API may contain hundreds of functions, which can be divided into seven groups by their functionality

- 1) Primitive functions: define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, line segments, polygons, pixels, text, and various types of curves and surfaces.
- 2) Attribute functions: govern the way that a primitive appears on the display such as color for a line, the pattern for a polygon, a type for the titles on a graph
- 3) Viewing functions: allow us to specify various views by describing the position and orientation of a synthetic camera and so on.

- 4) Transformation functions: carry out transformations of objects such as rotation, translation, and scaling.
- 5) Input functions: interact with keyboards, mice, and data tablets.
- 6) Control functions: communicate with the window system, initialize our programs, deal with any errors.
- 7) Inquiry functions: take care of differences between devices.

2.3.2 3D Object Model

Under a low cost PC platform, 3D objects can be easily modeled by using popular freely distributed 3D graphics libraries such as OpenGL or Direct3D. These 3D graphics libraries allow users to implement their own application that is capable of displaying 3D scenes with high visual quality without too much effort, and simultaneously reducing implementation time and cost.

OpenGL, developed by SGI (Silicon Graphics, Inc.), is a platform and hardware independent graphics library designed to be easily portable yet rapidly executable. It can be operated on PCs, workstations, and supercomputers under X Windows, OS/2, Microsoft Windows 9X and NT. OpenGL can be used from different computer languages such as C, C++, Fortran, Ada, or Java for scientific visualization research and commercial simulation packages.

In contrast, Direct3D from Microsoft running only on Windows operating systems, has more capability to cooperatively work with 3D graphics hardware accelerators. Therefore

it increases rendering speed and is widely used in Windows game programming with the 3D graphics hardware accelerator. On the other hand, Direct3D is built on the Component Object Model (COM) architecture and so, it is more complicated to use than OpenGL.

In general, 3D objects based on both graphics libraries are usually constructed from primitive objects. Thus, 3D objects can be represented by either wire-frame or surface models. The primitive types that are supported by both 3D graphics libraries can be categorized in Table 2.1.

Primitive Types	OpenGL	Direct3D
Point	X	X
Lines	X	X
Line Strip	X	X
Line Loop	X	-
Triangle	X	X
Triangle Strip	X	X
Triangle Fan	X	X
Quads	X	-
Quads Strip	X	-
Polygon	X	-

Table 2.1: Geometric Primitive Types

Most of the primitive types in OpenGL are also available in Direct3D. Although the quad and polygon primitive types are not provided in Direct3D, they can be substituted by triangles, triangles strip, or triangle fan. This is intentionally done in order to optimize the speed of rendering on graphics hardware. To model each primitive object, it is

required to provide a collection of vertices to a graphics library where each vertex is used to specify the coordinate of a point in 2D or 3D space. Figure 2.7 describe the primitive objects from vertices.

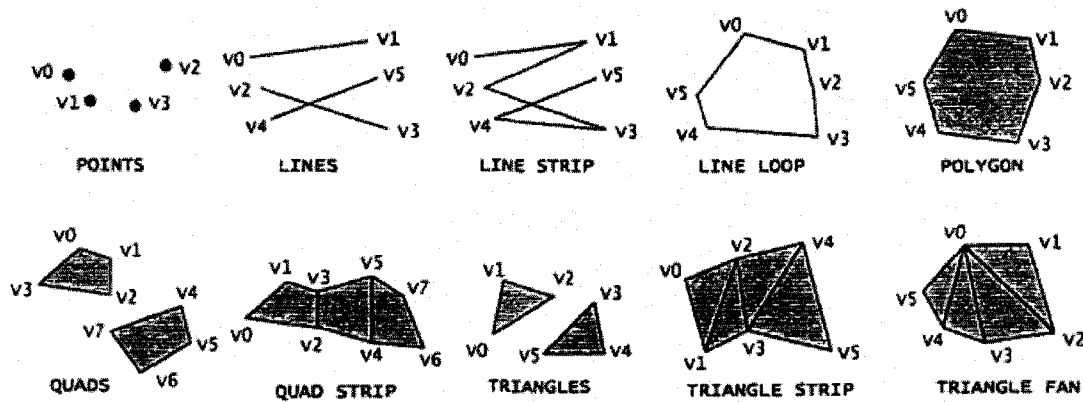


Figure 2.7: Geometric Primitive Types [Angel, 2003]

Therefore, based on the graphics library, solid-like 3D objects in simulation space are usually enclosed surface models composed of collections of polygonal surfaces. But the 3D models are complex. It is very difficult to build a complicated model just using primitives in a graphics library. Usually, we build 3D models in powerful CAD packages, then import them to the OpenGL or Direct3D application development environment for simulation or animation. For example, a 3D object model such as the *.SAT file building in CATIA are interpreted by the ACIS toolkit. OpenGL can create 3D geometry from the interpreted data.

To display these objects on the monitor screen, it is necessary to choose and set the projection matrix, which is used to map the 3D coordinate system into 2D coordinates on the screen. For both Direct3D and OpenGL, there are two projection methods available: perspective and orthographic projection. Usually, the perspective transformation is

preferred for a realistic point of view; the farther object is smaller than the closer object. On the other hand, orthographic projection is normally employed for CAD modeling which focuses on specifying the dimension of 3D objects rather than realistic appearance.

CHAPTER 3

DESIGN REQUIREMENTS

The Unified Reconfigurable Open Control Architecture (UROCA) is intended for controlling a wide variety of industrial machines. Therefore, as a first step it needs to investigate the commonality in different industrial machines. Considering the large amount of similarities that exist among the industrial 6R robotic systems, [Djuric et al, 2004] classified them into two main types (Puma-type and Fanuc-type) and created a unified kinematic structure called Generic Puma-Fanuc (GPF) model for them. A generic solution provides the geometric approach for solving the inverse kinematics problem based on the GPF model. The Unified Kinematic Modeler and Solver (UKMS) is the technological foundation of the unified robotic kinematic simulation interface design. This chapter reviews the UKMS in detail.

3.1 Generic Puma-Fanuc Model

The kinematic structures of 197 different industrial robots from 11 different manufacturers have been classified into a Puma-type group, a Fanuc-type group, and the other group. According to this classification 174 robots have 6 rotational joints and either

Puma or Fanuc kinematic structures depicted in Figure 3.1 and Figure 3.2 respectively.

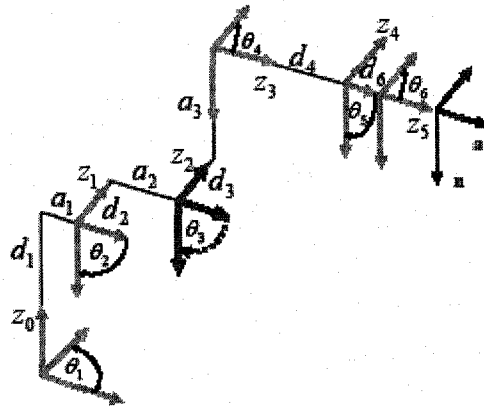


Figure 3.1: Generic Puma Kinematic Structure [Djuric et al, 2004]

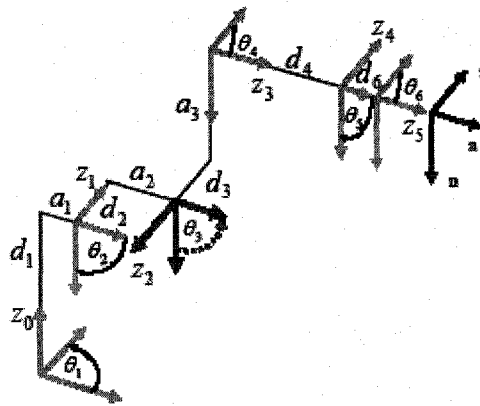


Figure 3.2: Generic Fanuc Kinematics Structure [Djuric et al, 2004]

The only different between the two kinematic structures is the direction of joint 3 which reflects the twist angle α_2 . $\alpha_2 = 0^\circ$ is for the Puma group and 180° for the Fanuc group as indicated in Table 3.1 and Table 3.2

Joint	θ_i	d_i	a_i	α_i
1	θ_1	d_1	a_1	-90°
2	θ_2	d_2	a_2	0
3	θ_3	d_3	a_3	90°
4	θ_4	d_4	0	-90°
5	θ_5	0	0	90°
6	θ_6	d_6	a_6	$\pm 180^\circ; 0$

Table 3.1: D-H parameters for Puma Kinematic Structure [Djuric et al, 2004]

Joint	θ_i	d_i	a_i	α_i
1	θ_1	d_1	a_1	-90°
2	θ_2	d_2	a_2	180°
3	θ_3	d_3	a_3	90°
4	θ_4	d_4	0	-90°
5	θ_5	0	0	90°
6	θ_6	d_6	a_6	$\pm 180^\circ; 0$

Table 3.2: D-H parameters for Fanuc Kinematic Structure [Djuric et al, 2004]

Hence, the two kinematic models can be merged together into a unified, reconfigurable kinematic model called a GPF model as illustrated in Figure 3.3 and Table 3.3.

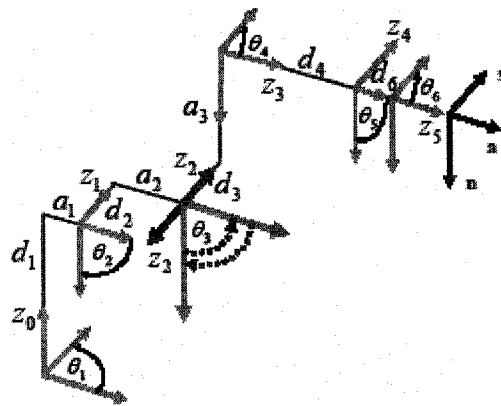


Figure 3.3: Generic Puma and Fanuc Kinematic Structure [Djuric et al, 2004]

Joint	θ_i	d_i	a_i	α_i
1	θ_1	d_1	a_1	-90°
2	θ_2	d_2	a_2	$\pm 180^\circ; 0$
3	θ_3	d_3	a_3	90°
4	θ_4	d_4	0	-90°
5	θ_5	0	0	90°
6	θ_6	d_6	a_6	$\pm 180^\circ; 0$

Table 3.3: D-H parameters for the GPF Model [Djuric et al, 2004]

The GPF model can be reconfigured from one kinematic structure to another by using configuration parameters $K_1, K_2, K_3, K_4, K_5, K_6$ defined in equation (3.1).

$$\begin{aligned} K_1 &= \sin \alpha_1, K_2 = \cos \alpha_2, K_3 = \sin \alpha_3 \\ K_4 &= \sin \alpha_4, K_5 = \sin \alpha_5, K_6 = \cos \alpha_6 \end{aligned} \quad (3.1)$$

3.2 A Unified Geometric-Based Solution

The inverse kinematics problem stated as follows. Given geometric link parameters and a desired position and orientation of the end-effector with respect to a reference coordinate system, how does one determine the joint variables to satisfy the same condition? From the position \mathbf{p} of end-effector, we calculate the values for the first three joint variables. The last three joint variables are calculated from the orientation vector $[\mathbf{n}, \mathbf{s}, \mathbf{a}]$ and joint values of the first three joints.

A geometric approach to solving the inverse kinematics problem was developed by [Djuric et al, 2004]. Below excerpts the main concepts and equations.

According to the various joint coordinate frames and human arm geometry, the different arm configurations of the Puma type and Fanuc type robots can be described with three configuration indicators: Arm (A), Elbow (E), Wrist (W), and extra Flip (F) as shown in Figure 3.4. The first two configuration indicators determine one solution from the possible four solutions for the first three joints. The third indicator doubles the number of possible solutions.

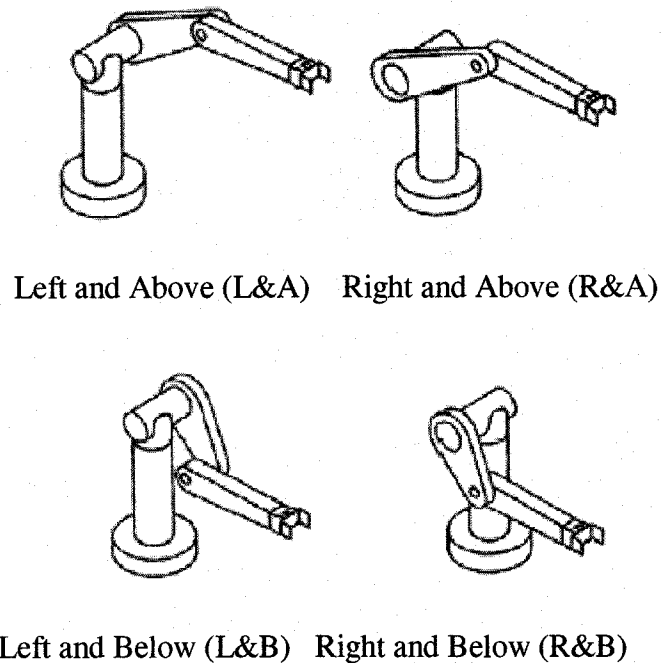


Figure 3.4: Definition of Various Arm Configurations [Djuric et al, 2004]

[Lee and Ziegler, 1984] introduced the definition of various configurations according to the human arm geometry:

- 1) Right Arm (RA): when positive θ_2 moves the wrist in a positive direction z_0 , when θ_3 is not active.
- 2) Left Arm (LA): when positive θ_2 moves the wrist in a negative direction z_0 , when θ_3 is not active.

- 3) Above Arm (AA): when the position of the wrist of the Right/Left Arm with respect to the shoulder coordinate system has a negative/positive coordinate value along the y_2 axis
- 4) Below Arm (BA): when the position of the wrist of the Right/Left Arm with respect to the shoulder coordinate system has a positive /negative coordinate value along the y_2 axis.
- 5) Wrist Down (WD): when the s vector of the hand coordinate system and the y_5 vector of the coordinate system have a positive dot product, $s \cdot y_5 > 0$.
- 6) Wrist Up (WU): when the s vector of the hand coordinate system and the y_5 vector of the coordinate system have a negative dot product, $s \cdot y_5 < 0$.
- 7) The forth indicator is introduced as a Flip (F) or Dot not Flip (DF) indicator.

According to the previous definitions of the robot configurations we have the following basic definitions:

$$\begin{aligned} A &= \begin{Bmatrix} +1, RA \\ -1, LA \end{Bmatrix}, \quad E = \begin{Bmatrix} +1, AA \\ -1, BA \end{Bmatrix} \\ W &= \begin{Bmatrix} +1, WD \\ -1, WU \end{Bmatrix}, \quad F = \begin{Bmatrix} +1, F \\ -1, DF \end{Bmatrix} \end{aligned} \quad (3.2)$$

The equations for calculating these indicators are:

$$\begin{aligned} A &= \text{sign}[-d_4 K_3 \sin(K_2 \theta_2 + \theta_3) - l_3 \cos(K_2 \theta_3 + \theta_2) - l_1 - l_2 \cos \theta_2] \\ E &= A \text{sign}\{l_4 \sin \theta_3 \cos \theta_4 - K_3 d_4 \cos \theta_3 + l_3 \sin \theta_3\} \\ W &= \begin{cases} \text{sign}(s \cdot z_4) & \text{if } s \cdot z_4 \neq 0 \\ \text{sign}(n \cdot z_4) & \text{if } s \cdot z_4 = 0 \end{cases} \end{aligned} \quad (3.3)$$

3.2.1 Solution for the First Three Joints

For the calculation of the first three joints, we need to define a vector projection, which points from the origin of the shoulder coordinate system to the point where the last three joints intersect. From Figure 3.5, we calculate the position vector. $\mathbf{p} = \mathbf{p}_6 - \mathbf{d}_6 \cdot \mathbf{a} = (p_x, p_y, p_z)^T$.

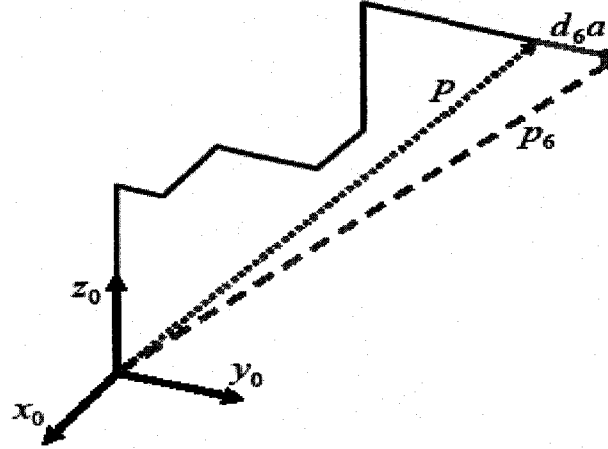


Figure 3.5: Position Vector P [Djuric et al, 2004]

Solution for join1:

To calculate θ_1 , we need to project vector \mathbf{p} onto the x_0y_0 plane as depicted in Figure 3.6 and 3.7

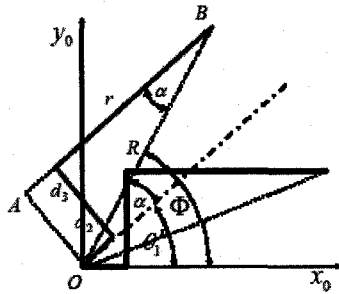


Figure 3.6: LA Projection of \mathbf{p} onto x_0y_0 Plane [Djuric et al, 2004]

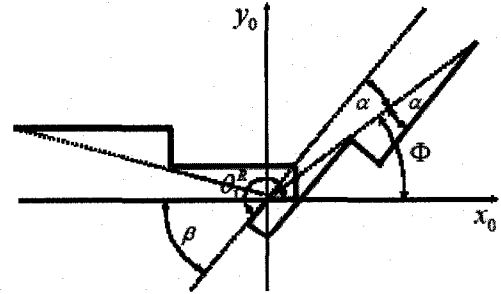


Figure 3.7: RA Projection of \mathbf{p} onto x_0y_0 Plane [Djuric et al, 2004]

$$r = \sqrt{p_x^2 + p_y^2 - (d_2 + K_2 d_3)^2}$$

$$\theta_1 = \tan^{-1} \left(\frac{-A.p_y.r - p_x(d_2 + K_2 d_3)}{-A.p_x.r - p_y(d_2 + K_2 d_3)} \right) \quad (3.4)$$

Solution for joint2:

The projections of the position vector \mathbf{p} onto the plane x_1, y_1 are shown in Figures 3.8. The diagram for the two combinations LA and RA and the two combinations EA and EB are shown in Figure 3.9.

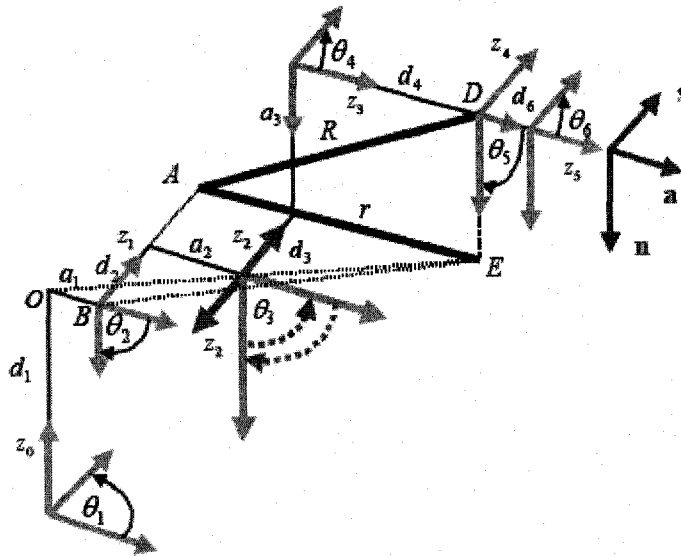


Figure 3.8: Projection of Vector \mathbf{p} onto x_1, y_1 Plane [Djuric et al, 2004]

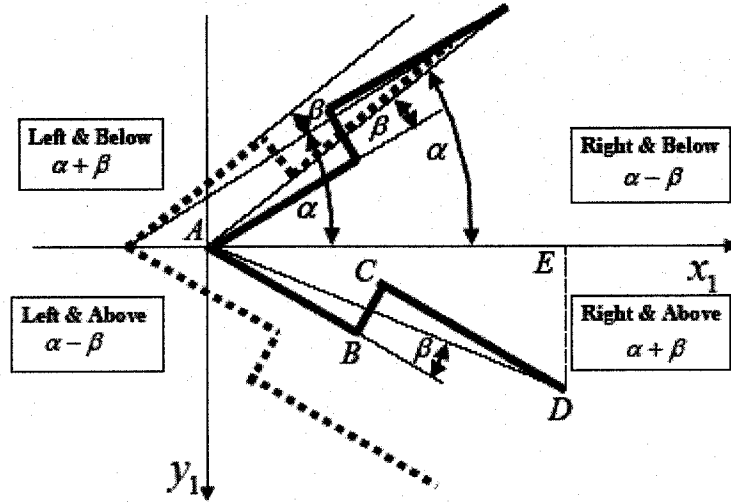


Figure 3.9: Four Combinations for Joint 2 Solutions [Djuric et al, 2004]

$$r = \sqrt{(p_x - l_1 \cos \theta_1)^2 + (p_y - l_1 \sin \theta_1)^2 - (d_2 + K_2 d_3)^2}$$

$$R = \sqrt{r^2 + (p_z - |d_1|)^2}$$

$$\sin = \frac{(|d_1| - p_z)}{R}, \cos = -\frac{r}{R} \cdot A$$

$$\cos \beta = \frac{R^2 + l_2^2 - l_3^2 - d_4^2}{2Rl_2}, \sin \beta = \sqrt{1 - \cos^2 \beta}$$

$$\sin \theta_2 = \sin \varphi \cdot \cos \beta + A \cdot E \cos \varphi \sin \beta$$

$$\cos \theta_2 = \cos \varphi \cos \beta - A \cdot E \sin \varphi \sin \beta$$

$$\theta_2 = \tan^{-1} \left(\frac{\sin \theta_2}{\cos \theta_2} \right) \quad (3.5)$$

Solution for joint3:

The calculation of the angle θ_3 requires projecting \mathbf{p} onto the $x_2 y_2$ plane.

$$\sin \beta = \frac{d_4}{\sqrt{l_3^2 + d_4^2}}, \cos \beta = \frac{|l_3|}{\sqrt{l_3^2 + d_4^2}}$$

$$\cos \phi = \frac{l_2^2 + l_3^2 + d_4^2 - R^2}{2l_2 \sqrt{l_3^2 + d_4^2}}$$

$$\sin \phi = K_2 \cdot A.E \sqrt{1 - \cos^2 \phi}$$

$$\sin \theta_3 = \sin(\phi - \beta) = \sin \phi \cdot \cos \beta - \cos \phi \cdot \sin \beta$$

$$\cos \theta_3 = \cos(\phi - \beta) = \cos \phi \cdot \cos \beta + \sin \phi \cdot \sin \beta$$

$$\theta_3 = \tan^{-1} \left(\frac{\sin \theta_3}{\cos \theta_3} \right) \quad (3.6)$$

3.2.2 Solution for the Last Three Joints

Solution for joint4:

Wrist Orientation	$\Omega = \mathbf{s} \cdot \mathbf{y}_5$ or $\mathbf{n} \cdot \mathbf{y}_5$	Wrist	$M = \text{Wrist} \cdot \text{sign}(\Omega)$
Down	\geq	+1	+1
Down	$<$	+1	-1
Up	\geq	-1	-1
UP	$<$	-1	+1

Table 3.4: Various Orientations for the WRIST [Djuric et al, 2004]

$$\theta_4 = \arctg \left(\frac{M \cdot K_2 [(\cos(\theta_1 + K_2 \cdot \theta_3))a_x + \sin \theta_1 \cos(\theta_2 + K_2 \cdot \theta_3)a_y + K_1 \sin(\theta_2 + K_2 \cdot \theta_3)a_z]}{M K_1 K_2 K_3 (a_x \sin \theta_1 - a_y \cos \theta_1)} \right) \quad (3.7)$$

Solution for joint5:

$$\begin{aligned}
\sin \theta_5 &= K_6 \{ [\cos \theta_1 \cos(\theta_2 + K_2 \theta_3) \cos \theta_4 + K_2 K_3 \sin \theta_1 \sin \theta_4] a_x \\
&+ [\sin \theta_1 \cos(\theta_2 + K_2 \theta_3) \cos \theta_4 - K_2 K_3 \sin \theta_1 \sin \theta_4] a_y + K_1 (\sin(\theta_2 + K_2 \theta_3) \cos \theta_4) a_z \} \\
\cos \theta_5 &= -K_6 \{ [K_3 K_4 \cos \theta_1 \sin(\theta_3 + K_2 \theta_2)] a_x + K_3 K_4 [\sin \theta_1 \sin(\theta_3 + K_2 \theta_2)] a_y \\
&- K_2 K_3 K_4 \cos(\theta_3 + K_2 \theta_2) a_z \}
\end{aligned}$$

$$\theta_5 = \tan^{-1} \left(\frac{\sin \theta_5}{\cos \theta_5} \right) \quad (3.8)$$

Solution for joint6:

$$\begin{aligned}
\sin \theta_6 &= [K_4 K_5 \{ \cos \theta_1 \cos(\theta_2 + K_2 \theta_3) \sin \theta_4 - K_2 K_3 K_4 K_5 \sin \theta_1 \sin \theta_4 \}] n_x \\
&+ [K_4 K_5 (\sin \theta_1 \cos(\theta_2 + K_2 \theta_3) \sin \theta_4 + K_2 K_3 K_4 K_5 \cos \theta_1 \cos \theta_4)] n_y \\
&+ [K_1 K_4 K_5 (\sin(\theta_2 + K_2 \theta_3) \sin \theta_4)] n_z \\
\cos \theta_6 &= -K_6 \{ [K_4 K_5 \cos \theta_1 \cos(\theta_2 + K_2 \theta_3) \sin \theta_4 - K_2 K_3 K_4 K_5 \sin \theta_1 \cos \theta_4] s_x \\
&+ [K_4 K_5 [\sin \theta_1 \cos(\theta_2 + K_2 \theta_3) \sin \theta_4 + K_2 K_3 K_4 K_5 \cos \theta_1 \cos \theta_4] s_y \\
&+ K_1 K_4 K_5 \sin(\theta_2 + K_2 \theta_3) \sin \theta_4] s_z \}
\end{aligned}$$

$$\theta_6 = \tan^{-1} \left(\frac{\sin \theta_6}{\cos \theta_6} \right) \quad (3.9)$$

3.3 Design Requirements

From the Generic Puma and Fanuc (GPF) model and the Unified Kinematic Modeler and Solver (UKMS), the solution for the joint angle variables contain proposed reconfiguration parameters $K_1, K_2, K_3, K_4, K_5, K_6$ and all the non-zero D-H parameters. This provides us with a systematic approach for constructing the GPF model and developing the UKMS by using a hybrid graphical and computational means.

The main objective of the proposed research is not only to provide a GUI software package to implement the functionality described in section 3.1 and section 3.2, but also to create a software platform, which has the features of easy of use, extensibility, portability, and reusability.

CHAPTER 4

DESIGN AND IMPLEMENTATION

This chapter presents a Graphical User Interface (GUI) for implementing the Unified Kinematic Modeler and Solver (UKMS), described in Chapter 3. The hardware and software structure of the GUI and their main functions are discussed. The design methodologies involved in the software platform are explained as well.

4.1 System Structure

The 3D GUI is being developed on a PC platform and runs in the Windows environment, which makes it easy to port this system to PC-based robot control systems. The overall system structure is illustrated in Figure 4.1.

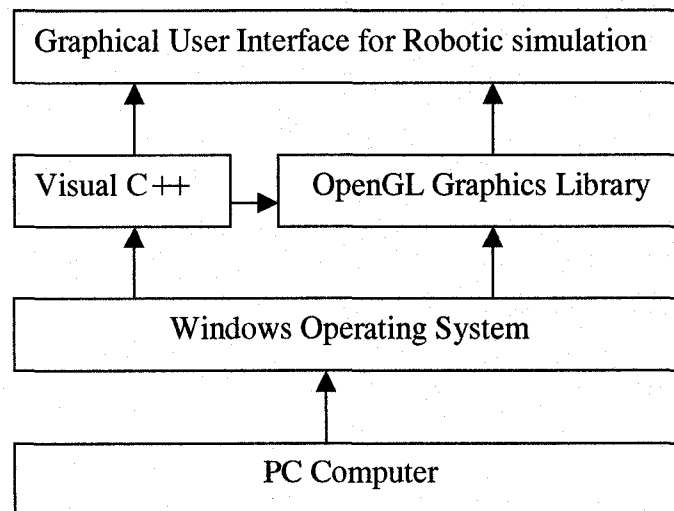


Figure 4.1: Overall System Structure

To reduce development effort and complexity, the robotic simulation platform is based on general purpose tools and technologies.

PC Technology. While in the past graphic workstations provided the powerful processing required for 3D simulation, the PC has caught up or even exceeded the performance of workstations. Compared to the UNIX workstation, a PC based system allows for a greater variety of hardware and software components. Additionally, these components and the PC itself are usually less expensive than their UNIX counterparts.

Window Operating System. A graphical user interface is one of the main successes of the Windows operating system. It is based on a user-friendly concept that reduces the requirement of computer skills for operation.

Object-oriented Programming in Visual C++. Visual C++ employs the Microsoft Foundation Class (MFC) to generate the graphical user interface such as windows, menus, and dialog boxes under the Windows system and to automatically create an application shell through the object-oriented programming paradigm. One advantage of Visual C++ is to simplify the graphical user interface design. The other advantage is object-oriented programming. Object-oriented programming is widely accepted and effectively used because it makes programs well-structured, modular, and reusable. With regard to developing a complex computer application, object-oriented programming has several benefits over procedural programming. First, it provides a much easier programming interface. For example, it is very easy and convenient to add new classes

and modify existing classes by the use of the class wizard. Second, object-oriented programming allows for a system architecture that is very flexible, but yet simple. That is, the components (classes) of the system can have built-in default functions and default settings. The programmer can use these default functions to reduce the code size or override it for a specific application. Finally, object-oriented programming supports generic programming which facilitates the development of components that are independent from a specific implementation. All of the above benefits are based on the general concepts of object-oriented programming: 1). Abstraction, 2) Encapsulation, 3). Polymorphism, and 4). Inheritance.

OpenGL Graphics Library. With the release of Windows NT 3.5, OpenGL became a part of the Windows operating system. Now with support for OpenGL in Windows and low priced graphics accelerators becoming readily available even on PC machines, OpenGL is becoming more attractive every day. OpenGL allows you to create high quality 3D images without dealing with the heavy math usually associated with computer graphics.

4.2 Menu Structure and Main Functions

One objective of this research is to develop an interactive menu for the construction of kinematic modeling and simulation for most industrial robots. The software allows the user to create kinematic models of robots by using just two steps with a joint coordinate system and arm lengths and offsets.

The default supervising GUI, which is opened automatically at the startup of the system, is a Multiple Document Interface (MDI) application. The menu structure of GUI is shown in Figure 4.2.

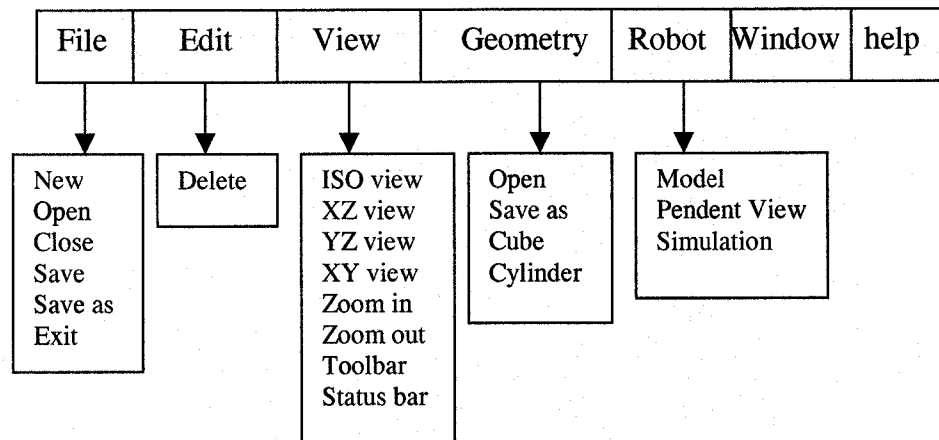


Figure 4.2: The Menu Structure of the GUI.

4.2.1 The Graphical User Interface

The default window is shown in Figure 4.3. The window comprises:

- 1) A Menu Bar that contains all command menus and options.
- 2) A Toolbar that contains icons for the most commonly used options.
- 3) A working area which is a 3D scene viewer showing images of the work cell objects.
- 4) A Status Bar that includes five columns for displaying the status message, the selected object name, the origin coordinate X, Y, Z values of the selected object respectively.

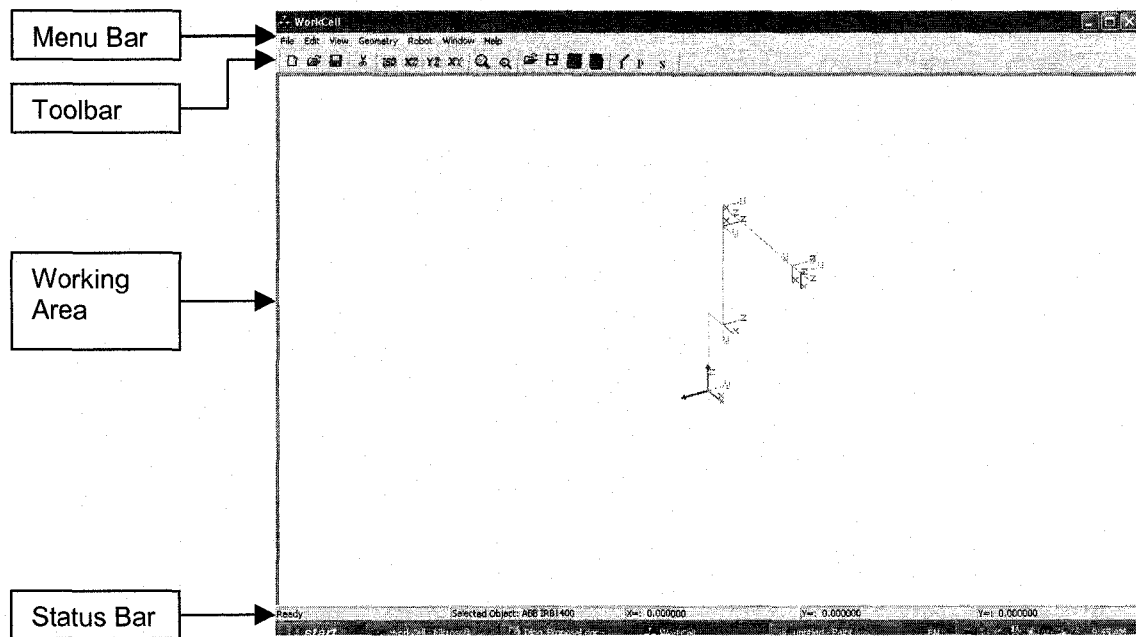


Figure 4.3: The Default Window of GUI

For the scene viewer, the world coordinate system is known as the global coordinate system and adopts a right-hand rectangular Cartesian coordinate system where the x, y, and z axes are in the same relative orientation as forefinger, second finger, and thumb respectively of a right hand shown in Figure 4.4.

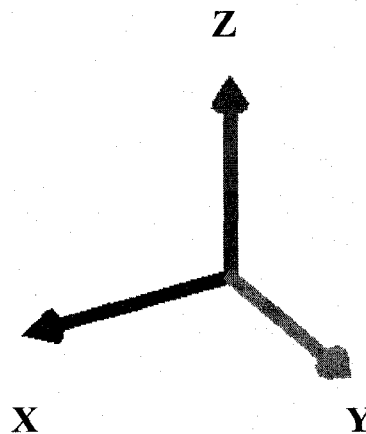


Figure 4.4: The World Coordinate System

4.2.2 File Functions

The work cell data which includes the robot kinematic model and the geometric data related to other objects are stored in a *.wld file. The File menu contains the usual Windows functions that allow you to open, close, and save files, and to exit the software. In addition, you can open the most-recently opened work cell files from this menu.

4.2.3 View Functions

The view position and direction affect how the model appears to the user when it is displayed. This system supports various standard views such as ISO view, XZ(front) view, YZ(side) view, XY(top) view, and Zoom in, Zoom out. Through this menu you can toggle the display of the Toolbar and Status Bar.

4.2.4 Geometry Model

An important task is the planning and designing of robotic work cell layouts which consist of robots, tools and environment. A simple 3D CAD system is provided to create basic geometric parameterized primitives like cubes, cones, cylinders. Using a 3 button mouse, we can modify the dimensions and color of the selected object and move the object in the scene. The selected object can be translated about the world coordinate system, and rotated about its own coordinate axis. All objects such as robots, box, cylinders, target points can be selected and saved as a *.obj file. Any other applications can load these files for use. A *.obj file format is same as the *.wld file.

4.2.5 Kinematic Model

Through the Robot- >Model menu, the kinematic structures of robots can be modeled inside the program. The robot data can be saved for future use by the Geometry- >Save as menu like other objects. To represent a robot's kinematic model in this software, we need to define all the joint coordinate frames with their positions and orientations. The procedures that create a kinematic model are:

- 1) Select the joint coordinate frame with its orientations with arbitrary link length and offset.
- 2) Modify the robot arm lengths and offsets with exact values.

The robot is like the other objects. It can be translated about the world coordinate system, and rotated about its own coordinate axis.

After the kinematic model has been created, the direct kinematics can be visualized by the pendent view. The pendent view allows the user to manipulate a robot by changes the joint variable values. Meanwhile the robot moves to the new location. The pendent view is an accurate way for the user to create a Target Point (TP).

The pendent view is divided into three sections. The top section is labeled "Joint values" and show the numbers of the six joint variables θ_i . The second section is labeled "End-effector" and displays the absolute position and orientation of the tool frame of the robot with regard to the base coordinate frame. The third section has no label. A Home button moves the robot to the home position. After the position and orientation of the end-effector have been input, the Inverse button calculates and displays the eight

solutions of six joint variables θ_i . If any one is selected, the robot will move to that location. The Learn TP button saves the current data as a Target Point (TP).

If the user changes the six joint variables θ_i in the first section, the robot will immediately move according to the θ_i values. Meanwhile, the direct kinematic problem will be calculated, and the absolute position and orientation of the tool frame of the robot will show in second section. If the user inputs the values in second section and presses the inverse button, the inverse kinematic problem will be solved. If one of the eight solutions is selected, the values of the six joint variables will be shown in the top section.

4.2.6 Simulation

The required robot path is given by a set of points. Those points are target locations of the robot's end-effector and are called Target Points (TPs). Each point is defined with its position and orientation. We need to calculate robot joint values for each point depending on the position and orientation of the point.

A path is a list of all the TPs that a robot follows during a sequence of motions. We can generate a TP using the Learn TP button on the pendent view. After the user has created the path, the simulation function can be used to simulate the activity of the robot following the path.

4.3 Design Methodologies

4.3.1. Objected-oriented Design Approach

In the classic approach to programming, a programmer designs the data structure followed by functions and procedures to process the data according to the data structure. This approach is called procedural programming because it starts with the procedures.

However, the world is object-oriented. To build a machine, first we think about the component objects and their purposes and behaviors. Then we select tools and procedures.

The object relationship can be classified into a group and attachment relationship. Objects may be grouped together to form a new object. This method is useful in building a complex object from many other objects. A group of objects can be treated as a single object in which all components can move together. Attachment is useful when a single object needs to be associated with other objects. An attachment is a one-way link in which one object (the child) is attached to another object (the parent). If a command is applied to the parent object, then the command is also applied to all the children. If a command is applied to a child, then it is not applied to the parent. For example, the robot arm is built by attaching the next link with the previous link. If Link 1 moves, then all the other links move in conjunction, maintaining their relative position. However, if link 2 moves, then Link 1 will not move.

A robotic work cell is a production cell which consists of a set of computer controlled manufacturing machinery, such as robots, numerical control (NC) machines, presses, conveyors, automated guided vehicles (AGV), etc. An automated factory can have one or more work cells. The components in a work cell are called objects. An example of work cell objects is shown in Figure 4.5.

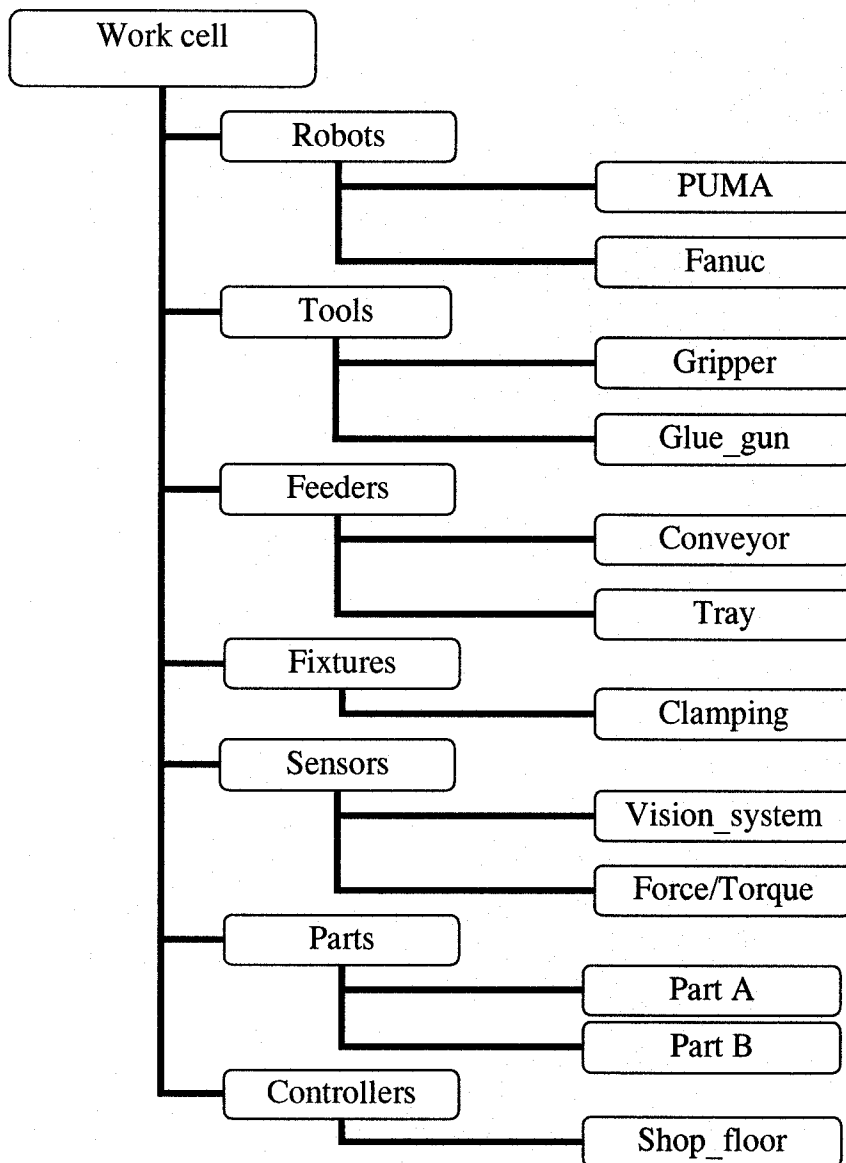


Figure 4.5: An Example of Work Cell Objects

The Object-Oriented Programming paradigm expresses computer programs in a similar way that people perceive the world.

The general concepts of object-oriented programming include Abstraction, Encapsulation, Inheritance, and Polymorphism. All Objects and object relationships can be modeled in C++ classes. A class is an abstract data type. Abstraction is the definition of an abstract data type. Its definition combines the data and the functions related to that object. Hence, the design of a robotic simulation platform results from grouping data and functions in a number of classes in meaningful way.

Encapsulation describes the design of a class in an object oriented language such as C++. Class definition gives its implementation and interface. The implementation, which is consists of data members and private member functions, is hidden from the user program. The interface, which consists of public member functions, is visible to the user. The object-oriented software design resembles the way that everything else in the world is designed. Anything can be seen as a collection of objects. Each object has a function that it performs. Each object knows how to perform its own functions with little knowledge of how the other objects perform their function. But it knows how to interact with the other objects.

Object-oriented programming uses the properties of class design to model the relationship between objects of different classes. There are fundamental interclass relationships: specialization, composition, and collaboration. The specialization

relationship is also called the IS-A relationship. The derived class is a specialized version of a base class using inheritance. For example, an engineer is a kind of employee, which is a kind of person. The composition relationship, also called the HAS-A relationship, represents the relationship that describes the other class objects that compose an object of a class. The HAS-A relationship can be detailed into a group and attachment relationship. For example, a table has four legs and a cover. A six DOF robot has six links. Collaboration is also called the USE-A relationship. When objects of one class use services of objects of other classes, collaboration exists.

A class can use part of the functionality and the data of another class, called the base class, by being derived from the base class. This process is called Inheritance. It relies heavily on code reuse and eliminates redundancy in the system. To extend the system, the user creates new classes. Usually new classes will be derived from one of already existing classes to minimize coding effort.

In Polymorphism, a derived class customizes the behavior of functions derived from the base class to meet the behavioural requirements of the derived class.

All objects can be described by their shape, position and orientation, object relationships in 3D geometric modeling. The shape includes geometry and material and color. If we think about what goes into describing a scene, in addition to our graphical objects and attributes, we have other objects, such as the lights and cameras which are

concepts in a 3D design. The class hierarchy of 3D geometric modeling is illustrated in Figure 4.6

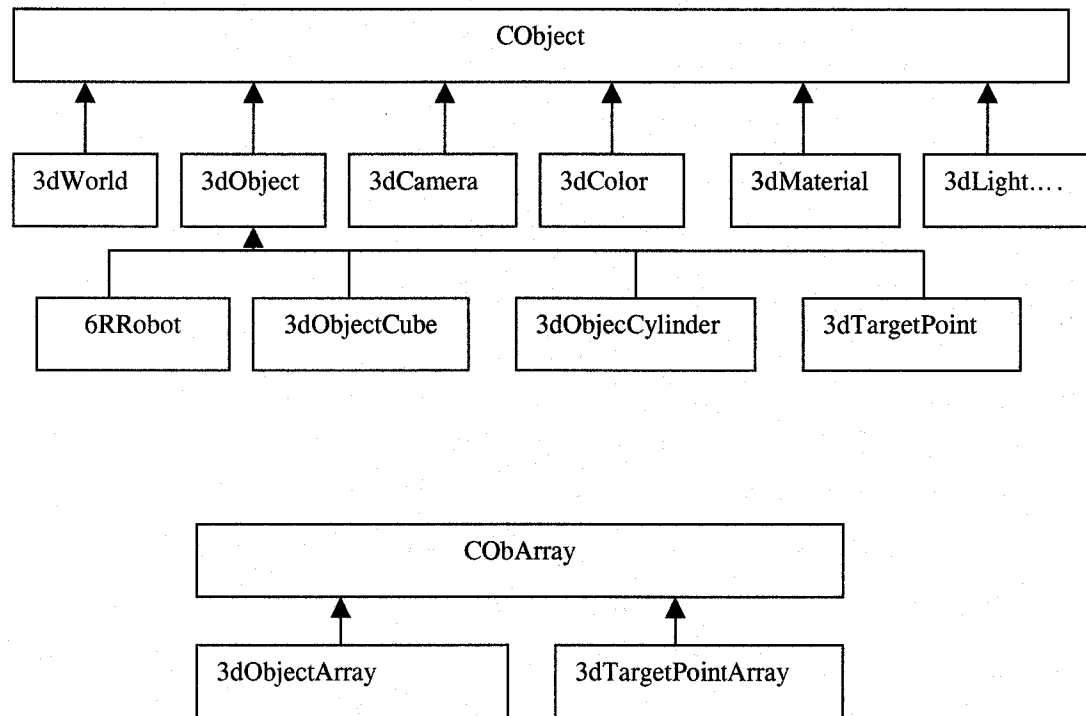


Figure 4.6: Class Hierarchy of the System

As mentioned earlier, the object-oriented design is to distinguish between common functionality/data and specific functionality/data. This concept is demonstrated in the design of the 3dObject class in Table 4.1.

Common Functions/data—3dObject	Specific Functions/data--6RRobot
<ol style="list-style-type: none"> 1. Color 2. Material 3. Light 4. Origin 5. Transformation 6. Rotation 7. (virtual) RenderObject() 	<ol style="list-style-type: none"> 1. Joint selections, joint lengths and offsets 2. RenderObject() 3. Serialize()
	Specific Functions/data—3dObjectCube
	<ol style="list-style-type: none"> 1. Height, Depth, Width 2. RenderObject() 3. Serialize()
	Specific Functions/data—3dObjectCylinder
	<ol style="list-style-type: none"> 1. Radius, Height 2. RenderObject() 3. Serialize()
	Specific Functions/data—3dTargetPoint
	<ol style="list-style-type: none"> 1. RenderObject() 2. Serialize()

Table 4.1: Common and Specific Functionality/Data for 3dObject Class

4.3.2. Library Modules and Dynamic-Link Libraries Design

Often, a family of applications will have some functionality in common. Sometimes a set of application functionality that will be used in an application is needed to be shared with another programmer. Perhaps the functionality will be used in a number of applications. Or some functionality is needed to be separated from the rest of the application for organization purposes. Therefore, a set of functionality can be placed into a library module (LIB), a self-contained compiled file. LIB is one of the first means available to share a compiled code with other programmers. The math class will be packaged into a library file for multiple applications.

The Visual C++ math library just has some simple functions. Therefore the math library for vector, matrix algebra for solving the direct and inverse kinematics problems is needed to be developed. The main functions of the math library are shown in Table 4.2.

Function Type	Function Name	Description
Math	<ol style="list-style-type: none"> 1. Radiansf() 2. Degreesf() 3. Cosf() 4. Sinf() 5. Sqrf() 6. Sqrtf() 7. Atan2f() 8. Fabs() 	<ol style="list-style-type: none"> 1. Chang degree to radian 2. Change radian to degree 3. Calculate cosine 4. Calculate sine 5. x^2 6. $\sqrt{}$ 7. Calculate arctangent, 2 parameters 8. x (All data type: float)
Vector	<ol style="list-style-type: none"> 1. VecClearf() 2. Vec3f() 3. VecCopy3f() 4. VecSubf() 5. VecAddf() 6. VecScalef() 7. VecDotf() 8. VecCrossf 9. VecLenf() 	<ol style="list-style-type: none"> 1. Zero a vector 2. Create a vector 3. Copy two vectors 4. Subtract two vectors 5. Add two vectors 6. Multiply a vector with scale 7. Dot product 8. Cross product 9. Calculate the length
Matrix	<ol style="list-style-type: none"> 1. ZeroMatrix() 2. IdentityMatrix() 3. Translate3D 4. Scale3D 5. Rotate3D 6. MultiplyMatrices 7. MatrixCopy 8. TransposeMatrix 	<ol style="list-style-type: none"> 1. Zero a matrix 2. Identify a matrix 3. Translate a matrix 4. Multiply a scale to a matrix 5. Rotate a matrix 6. Multiply two matrices 7. Copy two matrices 8. Transpose a matrix
Robot Related	<ol style="list-style-type: none"> 1. AdjaTransMatrix() 2. DirectKinematics() 3. InverseKinematics() 	<ol style="list-style-type: none"> 1. Generate adjacent transformation matrix 2. Calculate the direct Kinematics 3. Calculate the inverse kinematics

Table 4.2: The main Functions of the Math Library

Dynamic-Link Libraries (DLLs) [Chapman, 1998] are similar to library modules. The shared functionality can be placed into DLLs instead of library modules. The difference is in the way that the applications link to the library. With a library module, the application is linked to functionality in the library during the compiling and building process. With a DLL, the application links to the functionality in the library file when the application is run. The library file remains a separate file that is referenced and called by each application.

There are several reasons for creating DLLs instead of library module files. First, the size of the application executable files can be reduced because all the applications can share a single copy of the functionality distributed in one DLL, instead of duplicating the same functionality in each application. This method saves disk space on any systems where the applications are installed. Second, if the exported interface for the DLL does not change, the functionality in the DLLs can be updated and modified without having to update the application executable. Finally, DLLs can be used in any other Windows programming language such as C, Matlab, and so on. The shared functionality is available to a wider number of programmers, not just Visual C++ programmers.

The classes of 3D geometric modeling are packaged in to one DLL file called glOOP.dll. The glOOP is a MFC extension DLL, which is the easiest to code and create because they can be treated just like any other collection of classes. For any class that is exported from the DLL, the only thing is to add is the `AFX_EXT_CLASS` macro in the class declaration. This macro exports the class, making it accessible to Visual C++

applications. The one drawback to creating a MFC extension DLL is that they can be used with only other C++ compilers which support MFC such as Borlands's and Symantec's C++ compilers.

4.3.3. Scene Graphs

3D programming is associated with a scene graph. It is the totality of the objects that describe a scene, and there may be hierarchical relationships among these objects. We can represent the relationships among parts of the models both abstractly and visually with graphs. Mathematically, a graph consists of a set of nodes and a set of edges. Edges connect pairs of nodes and have a direction associated with them. Directed graphs have their edges leaving one node and entering another.

A tree is the most important of the types of graph. A tree is a directed graph without closed loops. Except for the root node, each node in a tree has a parent node and one or more child nodes. A node without children is called a terminal node. A tree is a hierarchical method of expressing the relationships in the physical model.

Robotics provides many opportunities for developing hierarchical models. For the six degrees of freedom PUMA robot shown in Figure 1.5, a tree structure in Figure 4.7 shows the relationships among the parts of the robot arm.

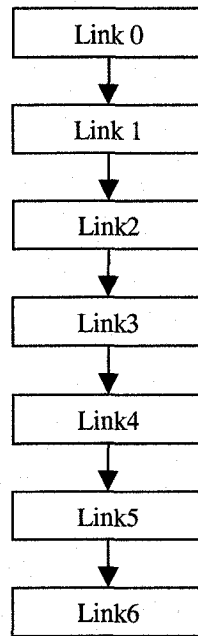


Figure 4.7: Tree Structure for a Robot Arm

After these hierarchies and information about objects have been expressed in a tree, a traversal algorithm has to be used to draw the objects. The traversal must visit every node. At each node we must position and display the objects. There are two tree-traversal algorithms: a depth-first or breadth-first search. In this paper a depth-first search is exploited. The trees shall be traversed left to right, depth first. That is, The traversal starts with the left branch, follow it to the left as deep as it can go, then go back up to the first right branch, and proceed recursively. This order of traversal is called a pre-order traversal.

There are two ways to write a tree-traversal function. One approach is that the traversal is done explicitly in the code. This approach relies on the application programmer to push and pop the required matrices and attributes. The code was hard-wired for one particular

example and thus would be difficult to extend or use dynamically. The second approach is that the traversal is done recursively. The code is simpler as the storage of matrices and attributes is done implicitly. One of the nice aspects of this traversal method is that it is completely independent of the particular tree. The other good aspect is that it is flexible. We can add or remove dynamic nodes rather than static nodes.

Scene Graphs are a convenient way to represent complex objects that have to obey certain constraints. The nodes of a scene graph contain pieces of an object to be drawn. The piece can possibly be empty (no node). Each edge has a transformation associated with it. There is one special node called the root, which will be the one the drawing is started. To draw a scene graph, the traversal is started from the root node. First, the piece stored at the root will be drawn. Then, for each edge out of the root node, the following steps will be done:

- 1) Save the current modelview matrix by pushing it onto the matrix stack.
- 2) Multiply the modelview matrix on the right by the transformation associated with the edge.
- 3) Call the drawing procedure recursively, pretending that the endpoint of the edge is the root.
- 4) Restore the original modelview matrix, by popping it from the matrix stack.

For example, the scene graph of kinematics modeling for the PUMA robot in Figure 1.5 is shown in Figure 4.8. The corresponding OpenGL pseudocode is illustrated in Figure 4.9. The code of the robot arm is traversed explicitly.

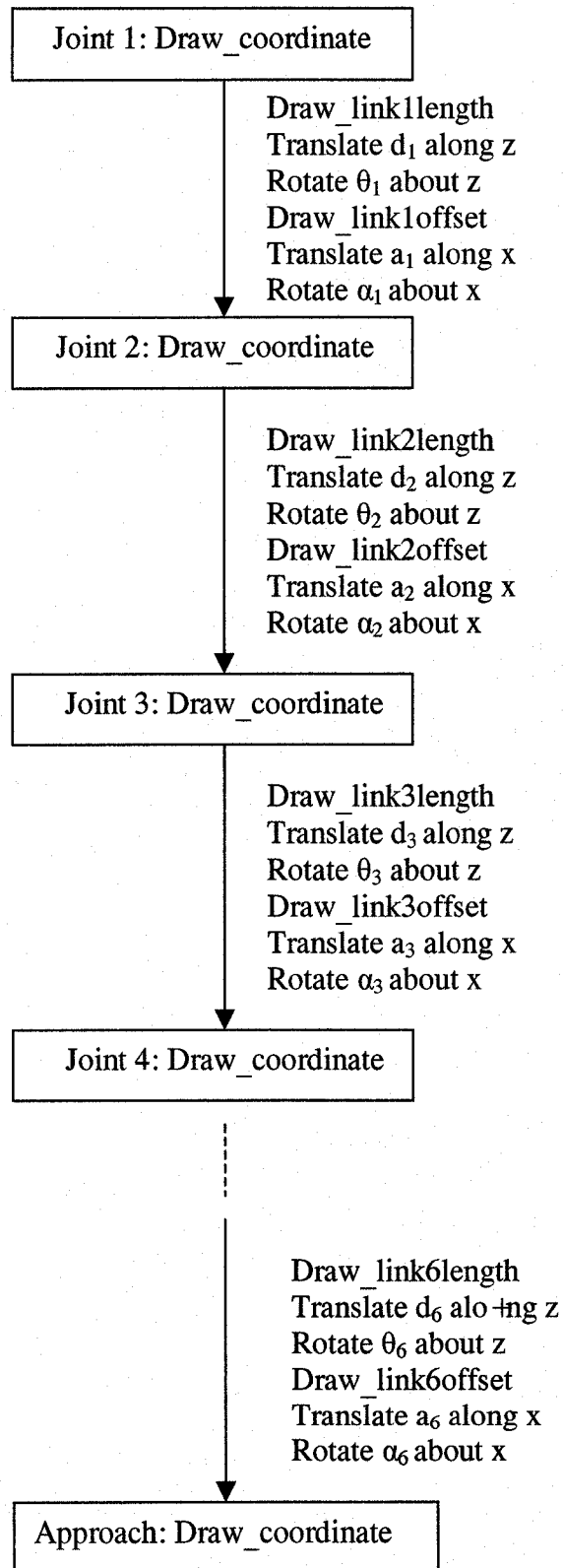


Figure 4.8: The Scene Graph of Kinematic Modeling for PUMA Robot

```

glPushMatrix();           // Start joint1
Draw_coordinate;
glPushMatrix();           // Start joint2
Draw_link1length
glTranslated(0.0,0.0,d1);
glRotated(Theta1,0.0,0.0,1.0);
Draw_link1offset
glTranslated(a1,0.0,0.0);
glRotated(Alpha1,1.0,0.0,0.0);
Draw_coordinate;
glPushMatrix();           // Start joint3
Draw_link2length;
glTranslated(0.0,0.0,d2);
glRotated(Theta2,0.0,0.0,1.0);
Draw_link2offset;
glTranslated(a2,0.0,0.0);
glRotated(Alpha2,1.0,0.0,0.0);
Draw_coordinate;
glPushMatrix();           // Start joint4
Draw_link3length;
glTranslated(0.0,0.0,d3);
glRotated(Theta3,0.0,0.0,1.0);
Draw_link3offset;
glTranslated(a3,0.0,0.0);
glRotated(Alpha3,1.0,0.0,0.0);
Draw_coordinate;
glPushMatrix();           // Start joint5
:
glPushMatrix();           // Start joint6
:
glPushMatrix();           // Start Approach
:
glPopMatrix();             //End Approach
glPopMatrix();             //End Joint6
glPopMatrix();             //End Joint5
glPopMatrix();             //End Joint4
glPopMatrix();             // End joint3
glPopMatrix();             // End joint2
glPopMatrix();             // End joint1

```

Figure 4.9: The Corresponding OpenGL Pseudocode

In a more general setting, the dynamic approach is needed to create structures that change interactively. For example, we can use this form to write an application that will let us edit figures, add and remove parts as desired. This is implemented by using dynamic object array concept in Visual C++.

4.3.4. Visual C++ implementation

The software for implementing the 3D GUI includes three Visual C++ projects: MathLib, glOOP, WorkCell. The MathLib is a static library to implement the math functionality. The glOOP is a Dynamic-Link Library for designing the classes of 3D geometric modeling. As mentioned before, the default supervising GUI implemented in the WorkCell project is a Multiple Document Interface (MDI) application. An MDI application is a document-centric application that allows users to work on not only multiple documents at one time, but also multiple types of documents.

Visual C++ allows for object-oriented GUI programming, and automatically creates the application shell. When an MDI application is created, five main classes are automatically created. They are:

- 1) CWorkCellApp—The CWinApp derived class
- 2) CMainFrame—The CMDIFrameWnd derived class
- 3) CChildFrame—The CMDIChildWnd derived class
- 4) CWorkCellDoc—The CDocument derived class
- 5) CWorkCellView—The CView derived class

The main classes in WorkCell project are described in Table 4.3

No.	Class Name	Class description
1	CWorkCellApp	Pass messages to the frame window and view objects.
2	CMainFrame	Hold the menu, toolbar, scrollbars, and any other visible objects attached to the frame
3	CChildFrame	Hold the CView class, pass messages and events to the view class.
4	CWorkCellDoc	House the document, , pass and receive information with CView
5	CWorkCellView	Display the document, pass and receive information with CDocument
6	C3dObjectPropSheet	Implement object property sheet
7	C3dPageCube	Implement cube definition property page
8	C3dPageCylinder	Implement cylinder definition property page
9	C3dPage6RRobotJ	Implement the robot joint selection property page
10	C3dPage6RRobotL	Implement the robot joint length and offset modification property page
11	C3dPageColor	Implement the color modification property page
12	C3dCoordinatePage	Implement the object origin coordinate modification property page
13	C6RPendentDlg	Implement the pendent view dialog
14	C6RInverseSolution	Implement the inverse kinematics solution dialog

Table 4.3: the Main Classes in WorkCell Project

In Figure 4.6, all object classes defined in this system are derived from the CObject class. The CObject is the root base class for most of the Microsoft Foundation Class Library (MFC). The CObject class contains many useful features that can be incorporated into your own program objects, including serialization support, run-time class information, and object diagnostic output. A class derived from CObject can exploit these CObject features.

In any object-oriented program, objects must be grouped and stored into collections of different types and sizes. The MFC provides several easy-to-use classes and templates to help with this common requirement. The CObArray is one of those classes. The CObArray class holds any objects that derived from CObject class. It is limited in size only by the amount of memory in the system. These object arrays are similar to C arrays, but they can dynamically shrink and grow as necessary. Each object in an array has a zero-based position which is used to located and reference the object. Therefore, this feature of CObArray is used to interactively insert, edit, and delete an object in the 3D geometric modeling.

CHAPTER 5

SIMULATION EXAMPLES AND RESULTS

In this chapter two case studies are given by using the GUI to create kinematic model and simulation. There are two ways to verify the inverse kinematics solutions. One is to use the solutions to calculate the direct kinematics and compare with the input. The other is to use the solutions to redraw the robot model. The results can be visualized. People can see if the end-effector will reach the target points. The target points are drawn according to the input data. The operation of the GUI for case study one is also included.

5.1 Case Study one for PUMA-like Robots

5.1.1 Problem Description

The first case study considers the PUMA-like robot, Fanuc ABB IRB6400. Its kinematics model and D-H parameters are shown in Figure 5.1 and in Table 5.1 respectively.

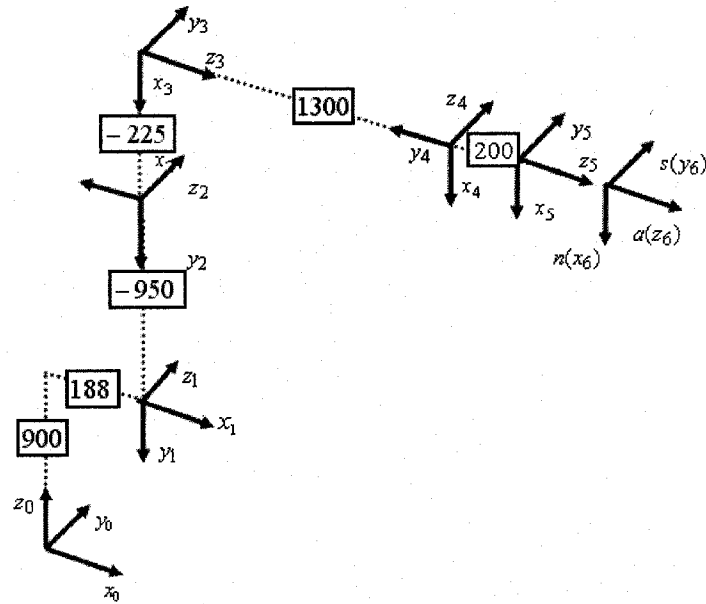


Figure 5.1: Kinematic Model for ABB IRB6400

Joint i	θ_i	α_i	a_i	d_i
1	0	-90°	188mm	900mm
2	-90°	0	-950mm	0
3	0	90°	-225mm	0
4	0	-90°	0	1,300mm
5	0	90°	0	0
6	0	0	0	200mm

Table 5.1: D-H Parameters for ABB IRB6400

The given path illustrated in Figure 5.2 is on the edge of a cube and includes six points and home position. The target points shown in Table 5.2 are defined with their positions $p = (p_x, p_y, p_z)^T$ and the rotated angles: yaw, pitch, roll about the principal axes of the robot's base coordinate frame.

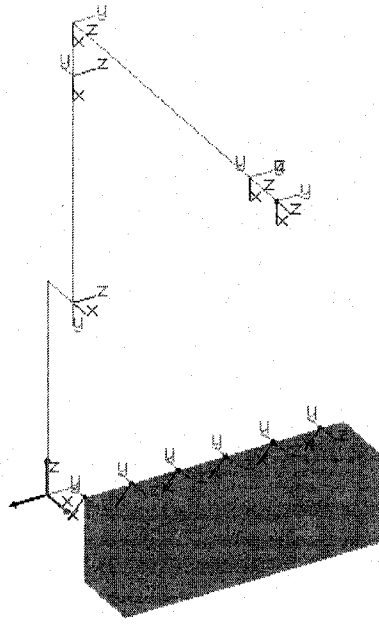


Figure 5.2: Simulation Path of Case Study One

Points	P_x	P_y	P_z	Yaw	Pitch	Roll
1	1,399.5	-650.25	875.5	-135	0	90
2	1,399.5	-450.25	875.5	-135	0	90
3	1,399.5	-250.25	875.5	-135	0	90
4	1,399.5	-50.25	875.5	-135	0	90
5	1,399.5	150.25	875.5	-135	0	90
6	1,399.5	350.25	875.5	-135	0	90
7	1,688	0	2,075	0	90	0

Table 5.2: Definition of Target Points of Case Study One

5.1.2 Inverse Kinematics Solutions

The eight inverse kinematics solutions for each point are calculated. The eight solutions are shown in Table 5.3-5.9.

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	150.504	150.504	-29.496	-29.496	150.504	150.504	-29.496	-29.496
Joint2	-41.679	49.129	-120.826	111.411	-41.679	49.129	-120.826	111.411
Joint3	-3.931	-156.431	175.657	23.982	-3.931	-156.431	175.657	23.982
Joint4	-113.039	38.323	58.477	-140.417	66.961	-141.677	-121.523	39.583
Joint5	138.025	-97.013	133.78	-74.994	-138.025	97.0133	-133.78	74.994
Joint6	-68.431	56.853	-80.217	39.257	111.569	-123.147	99.784	-140.743

Table 5.3: The Joint Solutions for Point 1 of Case Study One

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	157.083	157.083	-22.917	-22.917	157.083	157.083	-22.917	-22.917
Joint2	-46.066	53.901	-116.945	106.907	-46.066	53.901	-116.945	106.907
Joint3	3.271	-163.632	169.834	29.805	3.271	-163.632	169.834	29.805
Joint4	-113.141	40.650	58.586	-136.456	66.859	-139.351	-121.414	43.544
Joint5	134.905	-91.172	130.257	-70.979	-134.905	91.172	-130.257	70.979
Joint6	-62.066	60.131	-74.258	41.914	117.934	119.869	105.742	138.086

Table 5.4: The Joint Solutions for Point 2 of Case Study One

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	164.365	164.365	-15.635	-15.635	164.365	164.365	-15.635	-15.635
Joint2	-49.203	57.353	-114.04	103.481	-49.203	57.353	-114.04	103.481
Joint3	8.349	-168.71	165.595	30.044	8.349	-168.71	165.595	34.044
Joint4	-115.06	43.089	57.410	-132.228	64.940	-136.911	-122.59	47.772
Joint5	131.262	-85.410	126.08	-66.868	-131.262	85.410	-126.08	66.868
Joint6	56.930	64.126	-68.942	45.003	123.07	115.875	111.058	-134.997

Table 5.5: The Joint Solutions for Point 3 of Case Study One

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1,	-1,-1,-1
Joint1	172.201	172.201	-7.799	-7.799	172.201	172.201	-7.799	-7.799
Joint2	-51.087	59.443	-112.243	101.335	-51.087	59.443	-112.243	101.335
Joint3	11.366	-171.727	163.028	36.611	11.366	-171.727	163.028	36.0611
Joint4	-118.355	45.407	55.184	-127.875	61.645	-134.593	-124.816	52.125
Joint5	127.243	-79.668	121.426	-62.564	-127.243	79.668	-121.426	62.564
Joint6	52.687	68.728	-64.1008	48.395	127.313	-111.272	115.899	131.605

Table 5.6: The Joint Solutions for Point 4 of Case Study One

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1,	-1,-1,-1
Joint1	-179.639	-179.639	0.361	0.361	-179.639	-179.639	0.361	0.361
Joint2	-51.684	60.109	-111.665	100.64	-51.684	60.109	-111.665	100.64
Joint3	12.316	-172.678	162.212	37.427	12.316	-172.678	162.212	37.427
Joint4	-122.586	47.354	52.183	-123.568	57.414	-132.646	-127.817	56.433
Joint5	122.945	-74.008	116.48	-58.061	-122.945	74.008	-116.48	58.061
Joint6	-49.098	73.858	-59.613	51.949	130.903	-106.142	120.387	-128.051

Table 5.7: The Joint Solutions for Point 5 of Case Study One

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1,	-1,-1,-1
Joint1	-171.513	-171.513	8.487	8.487	-171.513	-171.513	8.487	8.487
Joint2	-50.976	59.319	-112.35	101.464	-50.976	59.319	-112.35	101.464
Joint3	11.188	-171.549	163.18	36.458	11.188	-171.549	163.18	36.458
Joint4	-127.296	48.669	48.714	-119.545	52.704	-131.331	-131.287	60.455
Joint5	118.462	-68.647	111.452	-53.504	-118.462	68.647	-111.452	53.504
Joint6	-46.051	79.426	-55.473	55.537	133.949	-100.575	124.527	-124.463

Table 5.8: The Joint Solutions for Point 6 of Case Study One

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	180	180	0	0	180	180	0	0
Joint2	4.55028	65.516	-174.217	90	4.550	65.516	-174.217	90
Joint3	-28.2731	-132.088	-160.361	0	-28.273	-132.088	-160.361	0
Joint4	180	180	0	0	0	0	180	0
Joint5	66.277	23.428	64.579	0	-66.277	-23.428	-64.579	0
Joint6	0	0	0	0	180	180	180	0

Table 5.9: The Joint Solutions for Home Position of Case Study One

5.1.3 Verification

There are two ways to verify the inverse kinematics solutions. One is to use the solutions to calculate the direct kinematics and compare with the input. The other is to use the solutions to redraw the robot model. The results can be visualized. People can see if the end-effector will reach the target points. The target points are drawn according to the input data.

The first example is to calculate the direct kinematics for home position.

$${}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6.$$

$$= \begin{bmatrix} 1 & 0 & 0 & 188 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 900 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & -950 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -225 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1,300 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 200 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 & 1,688 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 2,075 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots\dots (5.1)$$

The second example of calculating the direct kinematics for point 1 by using solution eight is given. Accordingly, the D-H parameters table is changed to Table 5.10 which is used to calculate the direct kinematics equation (5.2)

Joint i	θ_i	α_i	a_i	d_i
1	-29.496°	-90°	188mm	900mm
2	111.411°	0	-950mm	0
3	23.982°	90°	-225mm	0
4	39.583°	-90°	0	1,300mm
5	74.994°	90°	0	0
6	-140.743°	0	0	200mm

Table 5.10: D-H Parameters for the Point 1, Solution 8

$${}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6.$$

$$= \begin{bmatrix} 0.8704 & 0 & 0.4924 & 163.633 \\ -0.4924 & 0 & 0.8704 & -92.5643 \\ 0 & -1 & 0 & 900 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.3651 & -0.9310 & 0 & 346.8 \\ 0.9310 & -0.3651 & 0 & -884.437 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\dots \begin{bmatrix} 0.9137 & 0 & 0.4065 & -205.576 \\ 0.4065 & 0 & -0.9137 & -91.4513 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.7707 & 0 & -0.6372 & 0 \\ 0.6372 & 0 & 0.7707 & 0 \\ 0 & -1 & 0 & 1,300 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\dots \begin{bmatrix} 0.2589 & 0 & 0.9659 & 0 \\ 0.9659 & 0 & -0.2589 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.7743 & 0.6328 & 0 & 0 \\ -0.6328 & -0.7743 & 0 & 0 \\ 0 & 0 & 1 & 200 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & 0 & 1,399.5 \\ -0.7071 & 0 & 0.7071 & -650.25 \\ -0.7071 & 0 & -0.7071 & 875.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots \dots \dots (5.2)$$

After one of the inverse kinematics solutions is selected, it will be shown in the first section of the pendent view. The robot kinematic structure will be redrawn simultaneously. Figure 5.3 is the screen shot for the point 1, solution 8.

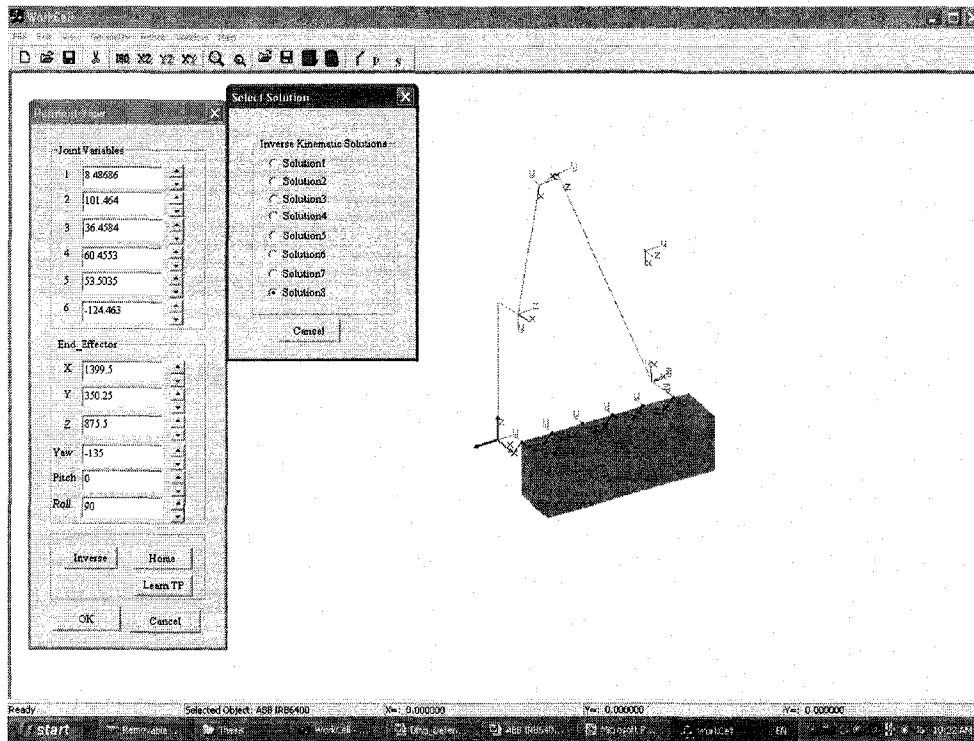


Figure 5.3: Visualization for the point 1, solution 8 of Case Study One

The results of direct and inverse calculation are saved in the file DirectKinematics and InverseKinematics respectively.

5.2 Case Study two for Fanuc-like Robots

5.2.1 Problem Description

The second case considers the Fanuc-like robot, ARCMate120iL. Its kinematics model and D-H parameters are shown in Figure 5.4 and in Table 5.11 respectively.

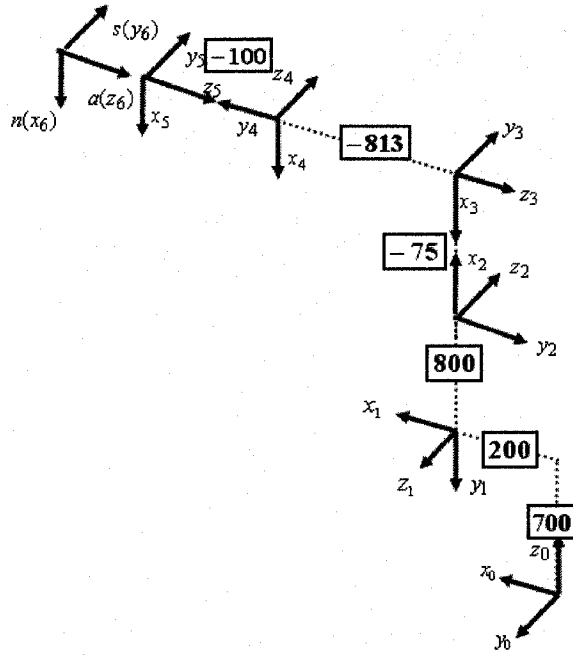


Figure 5.4: Kinematics Model for ARCMate120iL

Joint I	θ_i	α_i	a_i	d_i
1	0	-90°	200mm	700mm
2	-90°	180°	800mm	0
3	180°	90°	-75mm	0
4	0	-90°	0	-813mm
5	0	90°	0	0
6	0	0	0	-100mm

Table 5.11: D-H Parameters for ARCMate120iL

The given path illustrated in Figure 5.5 is four target points in the work cell. The points shown in Table 5.12 are defined with their positions $p = (p_x, p_y, p_z)^T$ and the rotated angles: yaw, pitch, roll about the principal axes of the robot's base coordinate frame.

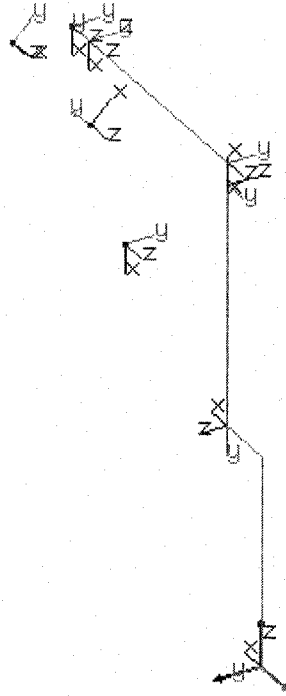


Figure 5.5: The Simulation Path for Case Study two

Points	P_x	P_y	P_z	Yaw	Pitch	Roll
1	1,113	0	1,575	180	-90	0
2	1,113	200	1,575	180	-90	45
3	1,000.5	0	1,300.25	180	-90	0
4	800.25	0	1,000.75	45	-90	0

Table 5.12: The Definition of Target Points for Case Study two

5.2.2 Inverse Kinematics Solutions

. The eight inverse kinematics solutions for each point are calculated. The eight solutions are in Table 5.13-5.16.

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	180	180	0	0	180	180	0	0
Joint2	-121.664	-166.726	-90	-4.207	-121.664	-166.726	-90	-4.207
Joint3	-50.687	-139.854	180	-10.541	-50.687	-139.854	180	-10.541
Joint4	180	180	0	0	0	0	0	180
Joint5	-19.023	-63.128	0	-83.666	19.023	63.128	0	83.666
Joint6	0	0	0	0	180	180	0	180

Table 5.13: The Joint Solutions for Home Position of Case Study Two

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	-168.832	-138.832	11.169	11.169	-168.832	-168.832	11.169	11.169
Joint2	-123.653	-165.605	-88.598	-4.250	-123.653	-165.605	-88.598	-4.250
Joint3	-53.760	-136.781	-178.581	-11.960	-53.760	-136.781	-178.581	-11.960
Joint4	150.132	167.3	-89.914	-11.268	-29.868	12.700	90.086	168.732
Joint5	-12.012	-59.278	11.168	-81.819	12.012	59.278	-11.168	81.819
Joint6	74.769	51.574	134.919	46.624	-105.231	-128.426	-45.081	-133.376

Table 5.14: The Joint Solutions for Point 2 of Case Study Two

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1,+1,+1	+1,-1,+1	-1,+1,+1	-1,-1,+1	+1,+1,-1	+1,-1,-1	-1,+1,-1	-1,-1,-1
Joint1	180	180	0	0	180	180	0	0
Joint2	-111.763	168.983	-96.638	15.452	-11.763	168.983	-96.638	15.452
Joint3	-16.966	-173.575	154.318	15.141	-16.966	-173.575	154.318	15.141
Joint4	0	0	180	180	180	180	0	0
Joint5	-4.797	72.558	-19.044	89.689	4.797	-72.558	19.044	-89.689
Joint6	-45	-45	-45	-45	135	135	135	135

Table 5.15: The Joint Solutions for Point 3 of Case Study Two

Solutions	1	2	3	4	5	6	7	8
A, E, W	+1, +1, +1	+1, -1, +1	-1, +1, +1	-1, -1, +1	+1, +1, -1	+1, -1, -1	-1, +1, -1,	-1, -1, -1
Joint1	180	180	0	0	180	180	0	0
Joint2	-106.676	143.623	-101.361	-106.676	-106.676	143.623	-101.361	39.332
Joint3	12.822	156.636	127.049	12.822	12.822	156.636	127.049	42.410
Joint4	180	180	0	0	0	0	180	180
Joint5	29.499	-76.986	41.590	-93.078	-29.499	76.986	-41.590	93.078
Joint6	0	0	0	0	180	180	180	180

Table 5.16: The Joint Solutions for Point 4 of Case Study Two

5.2.3 Verification

The first example is to calculate the direct kinematics for home position.

$${}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6.$$

$$= \begin{bmatrix} 1 & 0 & 0 & 200 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 700 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & -800 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 75 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\dots \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -813 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -100 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & -1 & 1,113 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1,575 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots \dots \dots (5.3)$$

The second example of calculating the direct kinematics for point 2 by using solution 7 is given. Accordingly, the D-H parameters table is changed to Table 5.17 which is used to calculate the direct kinematics equation (5.4).

Joint I	θ_i	α_i	a_i	d_i
1	11.169°	-90°	200mm	700mm
2	-88.598°	180°	800mm	0
3	-178.581°	90°	-75mm	0
4	90.086°	-90°	0	-813mm
5	-11.168°	90°	0	0
6	-45.081°	0	0	-100mm


Table 5.17: D-H Parameters for the Point 2, Solution 7

$$\begin{aligned}
 {}^0A_6 &= {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6 \\
 &= \begin{bmatrix} 0.9811 & 0 & -0.1937 & 196.212 \\ 0.1937 & 0 & 0.9811 & 38.739 \\ 0 & -1 & 0 & 700 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.0245 & -0.9997 & 0 & 19.577 \\ -0.9997 & -0.0245 & 0 & -799.76 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &\dots \begin{bmatrix} -0.9997 & 0 & -0.0248 & 74.977 \\ -0.0248 & 0 & 0.9997 & 1.8574 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.0015 & 0 & -1 & 0 \\ 1 & 0 & -0.0015 & 0 \\ 0 & -1 & 0 & -813 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &\dots \begin{bmatrix} 0.981 & 0 & -0.194 & 0 \\ -0.194 & 0 & -0.981 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.706 & 0.708 & 0 & 0 \\ -0.708 & 0.706 & 0 & 0 \\ 0 & 0 & 1 & -100 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 0 & 0 & -1 & 1,113 \\ -0.7071 & -0.7071 & 0 & 200 \\ -0.7071 & 0.7071 & 0 & 1,575.06 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots\dots(5.4)
 \end{aligned}$$

5.3 Operations of the Interface

After the file WrokCell.exe is run, the GUI will display on the screen. There are a few steps for simulation of robots by using the GUI. An example how to use this GUI to finish the case study one is given below.

5.3.1 Kinematic Model

The user can create the kinematic model by selecting the Robot→Model in the menu or clicking the  icon in the toolbar. The procedures that create a kinematic model are:

- 1) Select the joint coordinate frame with arbitrary link length and offset as shown in Figure 5.6. The sequence of the selections is 12, 22, 34, 42, 51, 62, 72. That is: for joint 1, select the second type; for joint 2, select the second type; and so on.

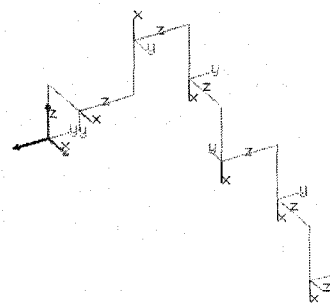
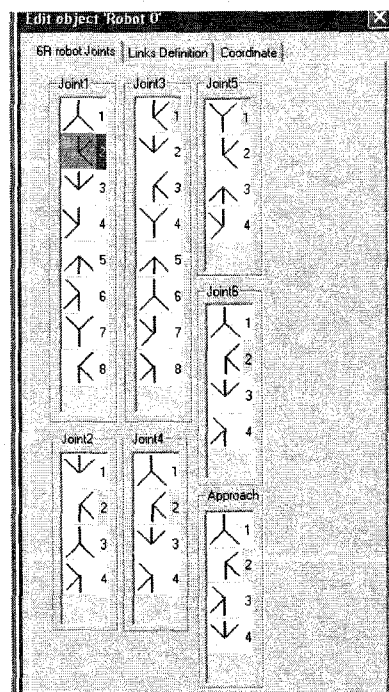


Figure 5.6: Selection of Joint Coordinate System

- 2) Modify the robot arm lengths and offsets with exact values as shown in Figure 5.7.

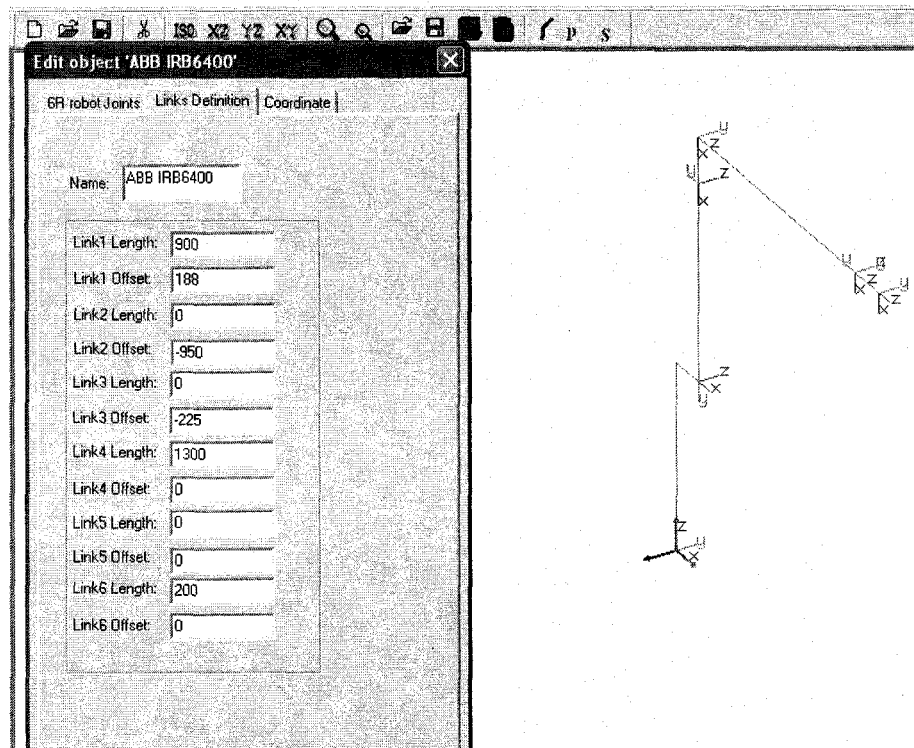


Figure 5.7: Modification of the Robot Arm Lengths and Offsets

3) Translate and rotate the robot in the scene, Shown in Figure 5.8

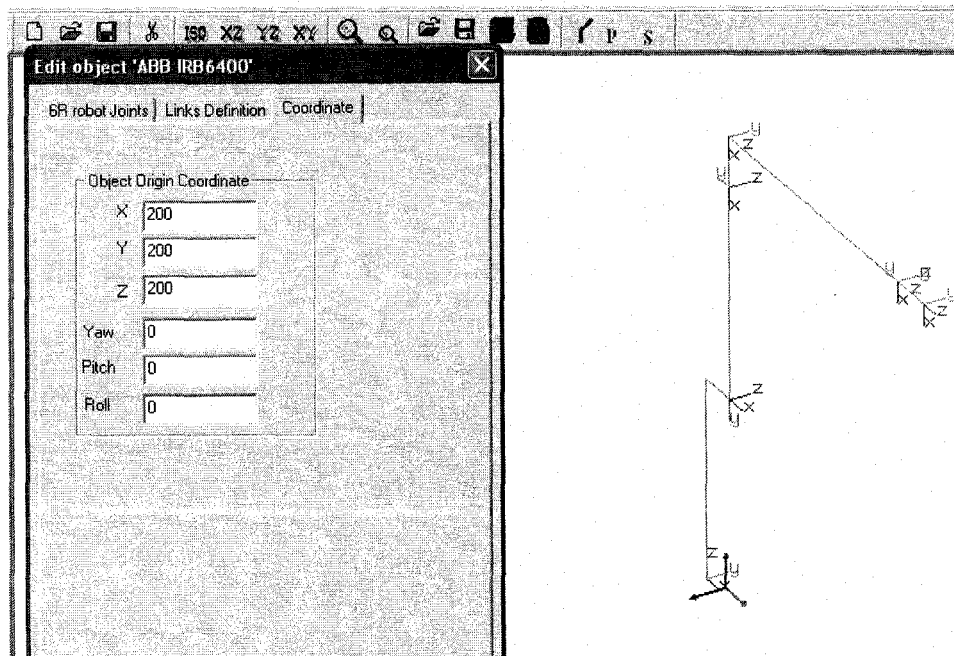



Figure 5.8: Translation and Rotation of the Robot

5.3.2 Geometric Model

By selecting the Geometry→Cube in the menu or clicking the  icon in the toolbar, a default cube will be generated. When left double clicking on the cube, the edit dialog shown in Figure 5.9 will open. The cube dimensions, color, the origin coordinate of the cube can be modified through the edit dialog. The width, depth, height are 1,100.5, 300.5, 350 respectively. The x, y, z, yaw, pitch, roll of cube's origin coordinate are 100, 1,550, 700, 0, 0, 0.

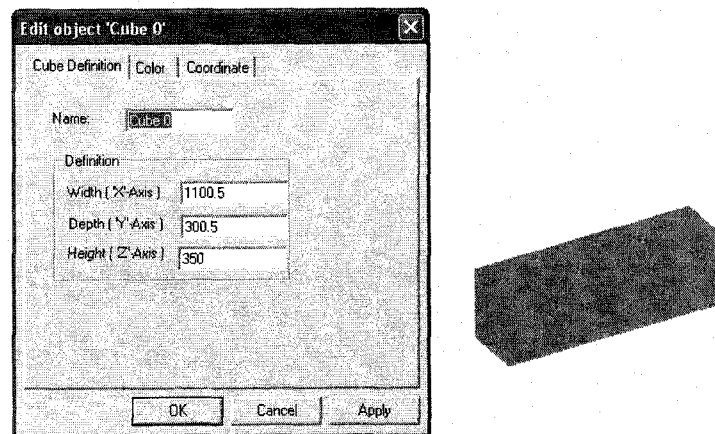



Figure 5.9: The Edit Dialog of Geometric Modeling for the Cube

5.3.3 Definition of Target Points

- 1) Select the robot by left clicking on the robot.
- 2) Select the Robot→Pendent View in the menu or left click the  icon in the toolbar; the pendent view dialog will open.
- 3) Input the x, y, z, yaw, pitch, roll values of the end-effect in Table 5.2. Left clicking the Inverse button to open the inverse kinematics solution dialog . Select

one of the eight solutions to show the joint variables on the first section of the pendent view. The robot will move to the target in the same time. The two dialogs are illustrated in Figure 5.10. The inverse kinematics solution dialog will be closed by Left clicking the Cancel button.

- 4) Left clicking the Learn TP button to generate one target point.
- 5) Repeat the step 3 and step 4 to create the all target points.

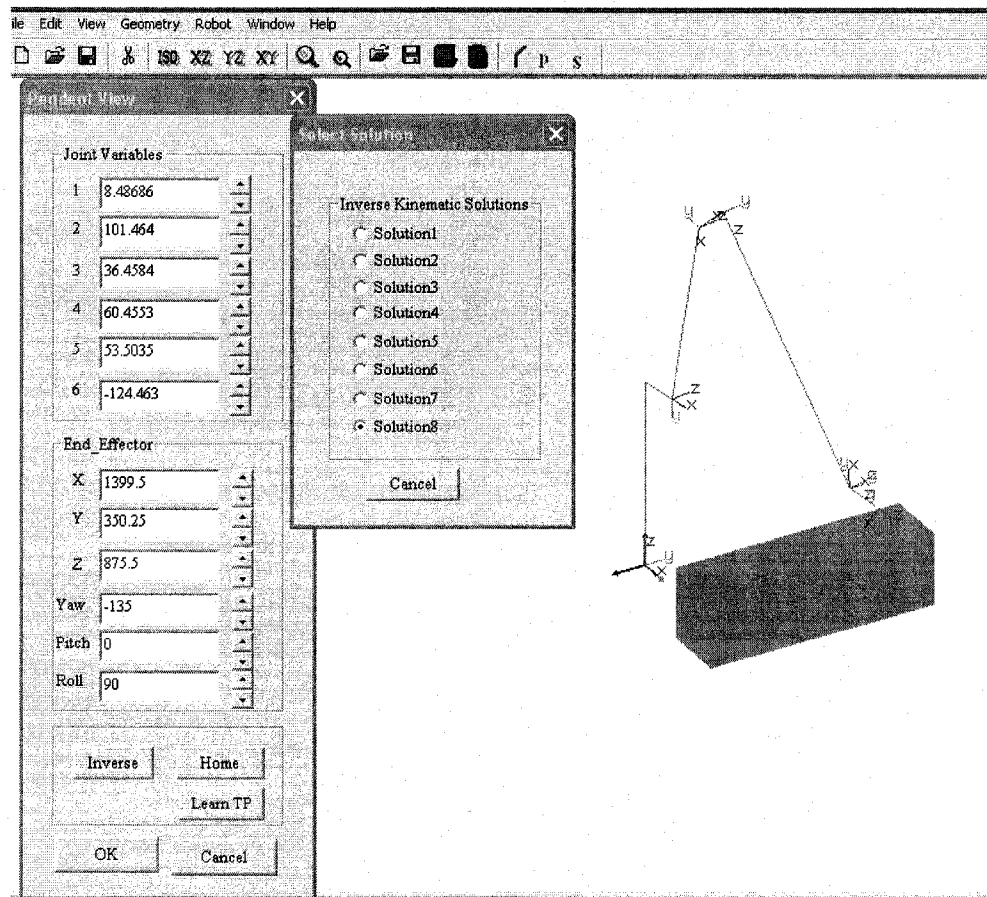



Figure 5.10: The Pendent View and Inverse Kinematics Solution Dialogs of the robot

5.3.4 Simulation

By selecting the Robot→Simulation in the menu or left click the  icon in the toolbar, the simulation will start automatically.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this research, a generalized approach for systematic 3D kinematic modeling and for solving the inverse kinematics problem was presented. A Unified Kinematic Modeler and Solver (UKMS) was developed on a PC-based computer under the Windows operating system using Visual C++. The kinematic structure falling within the scope of our Generic Puma and Fanuc (GPF) model of any 6R industrial robots can be easily generated from the graphical user interface by selecting the specific joint coordinate frame and by inputting arm lengths and offsets. The inverse kinematics problem was solved using a geometric approach, and the direct kinematics problem were also implemented for robot simulation.

The object-oriented approach has been applied to the realization of 3D kinematic modeling and simulation. Object-oriented design represents a relatively new method of software system design that provides a means for the development of robust software systems. It makes software programs well-structured, modular, and reusable. A class definition can embody the functional specification of operational knowledge about each object while hiding the implementation details. Generic programming facilitates the development of components that are independent from a specific implementation. The code can be reused from derived classes. Object-oriented programming allows for a

system architecture that is very flexible, yet simple. How objected-oriented principles can be utilized to extend the system for new objects shown in the geometry class was also demonstrated. The coding effort required for extension is significantly smaller. Additionally, there is no need to modify source code when extensions are needed.

The library module for the vector, matrix algebra, and for solving the kinematics problem is another way to share a set of functionality with other programmers for multiple applications. They reuse the programming code and save disk space. The Dynamic-Link Library for the implementation of geometric modeling made the software structure more organizable.

Scene Graphs are a convenient way to represent complex objects as well as the hierarchical relationships among these objects in both abstract and visual ways. The CObject and CObArray classes are powerful root base class for most of the Microsoft Foundation Class Library (MFC). The customized classes derived from those classes. Thus, this design makes it very easy to dynamically insert, edit, and delete an object in the 3D geometric modeling.

The user-friendly man machine interface makes a system that can be used easily. Standardized menu styles such as pull-down menus are used. For most menu functions, their related toolbars are designed.

The extensive case studies demonstrated that this graphical user interface is very effective in creating the kinematic model and verifying the direct and inverse kinematics solutions visually. The inverse kinematics solutions can be verified by calculating their direct kinematics equation or redrawing the robot kinematic model according to the solutions.

6.2 Contributions

- 1) A unified kinematic modeler, based on object-oriented design and programming, was developed.
- 2) The direct and inverse kinematics problem for the GPF model was solved and packaged into a library module, which can be used for different programmer in Visual C++ computer language.
- 3) A software platform, which has the features of easy of use, extensibility, portability, and reusability, was created for 3D geometric modeling and simulation.

6.3 Future Work

There are a number of research and development issues related to the 3D graphical user interface that need further investigation.

1. An improvement for the inverse kinematic algorithm should be done by adding the joint limits, unreachable conditions, and the singularity.

2. An algorithm should be developed to optimize the eight inverse kinematics problem solutions. Just one solution of joint variables will be provided to the simulation system.
3. The manufacturing model of industrial robots should be introduced for better visualization and dynamic analysis and control design.
4. The 3D graphical software platform as well as the object-oriented design and scene graph methodologies should be extended to the graphical robotic simulation system which will include the servo control, 3D simulation, and robotic application development to fit the requirements of a reconfigurable control process in UROCA.
5. A joint space control method can be developed. Because the inverse kinematics solutions transform the operational space to joint space. The joint variables became the input of the robot arm control design. One approach to robot arm control system design is to treat each joint of the robot arm as a simple joint servomechanism. That is, single manipulator joint is controlled independently from the others. The example of joint space control is the PUMA 560 series robot arm. A robot-arm control system is shown in Figure 6.1. The objective of this system is to control the angular motion about z axis. A controller can be designed for the servo system. The controller should make the system stable, has a good damping of the oscillations, and good tracking performance.

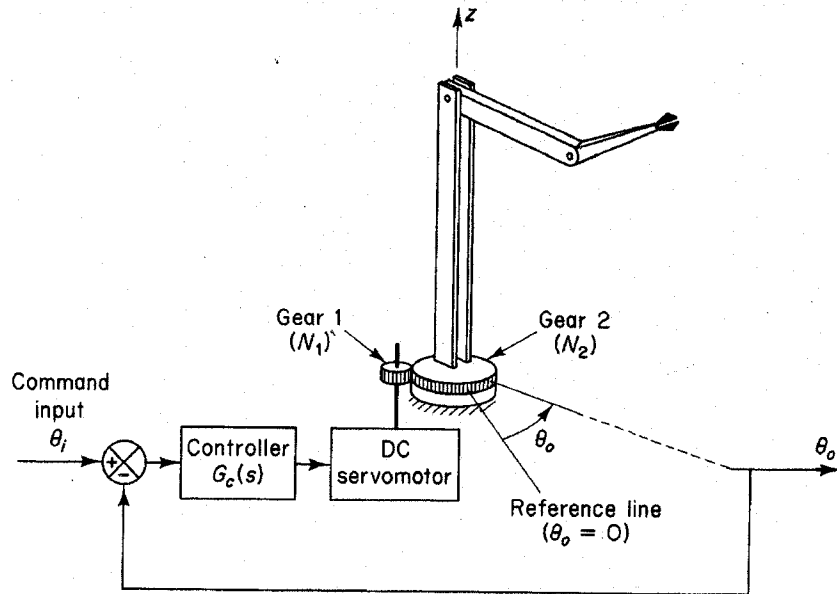


Figure 6.1: Robot Arm Control System

6. RT-LAB Simulation and experiment of the controller design can be conducted.

The RT-LAB can be used to implement the controller design. There are few steps.

- 1) Automatically receive the input θ_i from this GUI.
- 2) Build the dynamic model of the robot arm.
- 3) Run simulation off-line for isolating and debugging various problems.
- 4) Execute the simulation in real-time.
- 5) After the control design is verified by the RT-LAB, the real robot can be connected to the system. The overall architecture of the system is

illustrated in Figure 6.2.

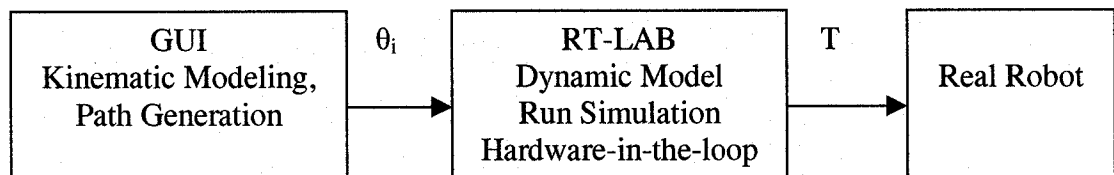


Figure 6.2: The overall architecture of the system

REFERENCES

- 1) Angel, E., 2003, "Interactive Computer Graphics: A Top-Down Approach with OpenGL", 3rd ed, Boston: Addison Wesley, c2003, ISBN: 0-201-77343-0.
- 2) Anton, S., Fries, T., Horsch, T., Schroer, F. W., Willnow, C., Wolf, C., 2001, "A framework for Realistic Robot Simulation and Visualisation", url:[http:// www.easy-rob.com/data/Easy-Rob-Tech-Articles.pdf](http://www.easy-rob.com/data/Easy-Rob-Tech-Articles.pdf).
- 3) Applied Computing & Engineering Ltd, 2005, "AC&E CimStation Robotics", url: <http://www.acel.co.uk/robotics.html>.
- 4) Bernhardt, R., Schreck, G., Willnow, C., 1995, "Realistic robot simulation", Computing & Control Engineering Journal, Vol.6, pp.174 -176.
- 5) Biggs, G., MacDonald, B., 2003, "a survey of robot programming systems", In Proceedings of the Australian Conference on Robotics and Automation, CSIRO, Brisbane, Australia, December 1-3.
- 6) BYG Systems Ltd, 2002, "Grasp2000 User Manual",url:http://www.staffs.ac.uk/personal/engineering_and_technology/sow1/Robotics/grasp/.
- 7) Chapman, D., 1998, "Sams teach yourself Visual C++6 in 21 days", a division of macmillan computer publishing, ISBN: 0-673-31240-9.
- 8) Cheng, F. S., 2000, "A Methodology for Developing Robotic Workcell Simulation Models", Simulation Conference, Proceedings, Winter, Vol. 2. pp. 1265-1271.
- 9) Cimetrix Inc., 2002, "CODE 6TM Machine control software for high-performance applications", url:<http://www.cimetrix.com/pdfs/code6.pdf>

- 10) Dai, W., Kampker, M., 1999, "PIN- A PC-based robot simulation and offline programming system using macro programming techniques", The 25th Annual Conference of the IEEE Industrial Electronics Society, Vol.1, pp.442-446.
- 11) Denavit, J., Hartenberg, R. S., 1955, "A kinematic notion for lower-pair mechanisms based on matrices", J App Mech 77, pp.215-221.
- 12) Djuric, A. M., 1999, "Economical industrial work cell modeling simulation and layout design", Master's Thesis, University of Windsor.
- 13) Djuric, A. M., ElMaraghy, W. H, ElBeheiry, E. M., 2004, "Unified integrated modeling of robotic systems", NRC International Workshop on Advanced Manufacturing, June 2004, London, Canada.
- 14) ElBeheiry, E., ElMaraghy, W., ElMaraghy, H., 2004, "The structured design of a reconfigurable control process", Proceeding of CIRP Design Seminar, Cairo, Egypt.
- 15) Flow software technologies, 2002, "WORKSPACE 5.03 User Manual". url: <http://www.workspace.com>.
- 16) Fu, K. S., Gonzalez, R.C., Lee, C. S. G., 1987, "Robotics: Control, Sensing, Vision, and Intelligence", McGraw-Hill, Inc., ISBN: 0-07-022625-3.
- 17) Gourdeau, R., 1997, "Object-oriented programming for robotic manipulator simulation", Robotics & Automation Magazine, IEEE, Vol. 4, Issue: 3 pp.21 – 29.
- 18) Hong, K. S., Kim, J. G., Huk, C. D., Choi, K. H., Lee, S., 2001, "A PC-based open robot control system: PC-ORC ", Industrial Electronics, 2001. Proceedings. ISIE 2001. IEEE International Symposium on, Vol.3, pp.1901-1906.
- 19) Intelitek Inc., 2003, "RoboCell User Manual". url: [http://great-lakes-training.com/documents/100346-D-RoboCell-usb-v49\(0311\).pdf](http://great-lakes-training.com/documents/100346-D-RoboCell-usb-v49(0311).pdf).

- 20) Kim, G. T., Cha, S. D., Bae, D. H., 1997, "Task.o object modeling approach for robot workcell programming", Computer Software and Applications Conference, 1997. COMPSAC 97. Proceedings, pp.109 – 114.
- 21) Koseeyaporn, P., 2003, "Component-based robotic simulation", Ph.D. Thesis, University of Vanderbilt.
- 22) Lapham, J., 1999, "RobotScript: the introduction of a universal robot programming language", Industrial Robot: An International Journal, Vol.26, pp.17-25.
- 23) Lee, C. S. G., Ziegler, M., 1984, "A geometric approach in solving the inverse kinematics of PUMA robots", IEEE Aero Electronic Sys 20(6), pp.695-706.
- 24) Loffler, M.S., Chitrakaran, V.K., Dawson, D.M, 2001, "Design and implementation of the Robotic Platform", Control Applications, Proceedings of the 2001 IEEE International Conference on, 5-7 Sept. PP.357 – 362.
- 25) Loffler, M.S., Dawson, D.M., Zergeroglu, E., Costescu, N.P., 2001, "Object-oriented techniques in robot manipulator control software development", American Control Conference, 2001. Proceedings of the 2001, Vol.6, pp.4520 – 4525.
- 26) Loffler, M., 2001, "New object-oriented and PC-based approaches to robot control software", Ph.D. Thesis, Clemson University.
- 27) Loffler, M.S., Costescu, N.P., Dawson, D.M., 2002, "QMotor 3.0 and the QMotor robotic toolkit: a PC-based control platform", Control Systems Magazine, IEEE, Vol.22, PP.12 – 26.
- 28) Mak, K.L., Lau, H.Y.K, Wong, S.T.W., 1999, "Object-oriented specification of automated manufacturing systems". Robotics and Computer-Integrated Manufacturing, Elsevier, Vol.15, pp.297-312.

- 29) Miller, D.J., Lennox, R.C., 1991, "An object-oriented environment for robot system architecture", IEEE Control Systems. pp.14-23.
- 30) Meynard, Jean-Paul, 2000, "Control of industrial robots through high-level task programming", Master's Thesis, Linköpings universitet, Sweden.
- 31) Montagnier, P., Steiner, S. J., 2000, "Design of a graphical user interface (GUI) between a PC-based CAD system and a flexible assembly robotic cell", Factory 2000 - The Technology Exploitation Process, Fifth International Conference on Vol.435, pp.162-169.
- 32) Nnaji, B. O., 1993, "Theory of automatic robot assembly and programming". Chapman & Hall. ISBN: 0-412-39310-7.
- 33) O'Leary, J. J., 1998, "CROBOTS: CAD Based Robot Simulation Tool", Master's Thesis, Memorial University of Newfoundland.
- 34) Orady, E. A., Osman, T. A., Bailo, C. P., 1997, "Virtual reality software for robotics and manufacturing cell simulation", Computers & Industrial Engineering, Vol.33, pp.87-90.
- 35) Orady, E.A., Osman, T. A., Bailo, C. P., 1997, "Capability study of robotics and manufacturing cell simulation software", Computers & Industrial Engineering, Vol.33, pp.83-86.
- 36) Owens, J., 1994, "WORKSPACE-a microcomputer-based industrial robot simulator and off-line programming system", Next Steps for Industrial Robotics, IEE Colloquium on , 17 May 1994, pp.1-4.

- 37) Prinz, M., Liu, H. C., Nnaji, O., Lueth, T., 1996, "from CAD-based kinematic modeling to automated robot programming", robotics & computer-integrated manufacturing, Vol.12, No.1, pp.99-109.
- 38) Robinette, M. F., Manseur, R., 2001, "ROBOT-DRAW, an internet-based visualization tool for robotics education", IEEE transactions on education, Vol.44, No.1, pp.29-34.
- 39) Rooks, B. W., 1997, "Off-line programming: a success for the automotive industry", Industrial Robot, Vol.24, No.1, pp.30-34.
- 40) Sciavicco, L., Siciliano, B., 1996, "Modeling and control of robot manipulators", The McGraw-Hill Companies, Inc. ISBN: 0-07-057217-8.
- 41) Stringham, R., 1999, "Simulation tools ease and speed assembly cell development", Assembly Automation, Vol.19, No.2, pp.121-125.
- 42) Wittenberg, G., 1995, "Developments in offline programming: an overview", Industrial Robot, An International Journal, Vol.22, pp.21-23.
- 43) Yasuda, G., 1999, "An object-oriented multitasking control environment for multirobot system programming and execution with 3D graphic simulation", International Journal of production Economics, Vol.60-61, pp.241-250.

APPENDIX 1: SAMPLE C++ PROGRAMS

Below is the implementation of the kinematic model for six rotational joints robot using OpenGL and C++ languages.

```
// Create robot kinematic model

glNewList(listJoint11, GL_COMPILE);
    CCoordinate *pCoordinate11=new CCoordinate(100.0,100.0,100.0,1);
    pCoordinate11->Render();
glEndList ();

glPushMatrix();    // Start joint1
    glRotated(m_iLinkTheta0, 0.0, 0.0, 1.0);
    glRotated(m_iLinkAlpha0, 0.0, 0.0, 0.0);
    glCallList(listJoint11);

glPushMatrix();    // Start joint2
    glColor4f(0.0f,1.0f,0.0f,1.0f);
    glBegin(GL_LINES);
        glVertex3f( 0.0, 0.0, 0.0);
        glVertex3f( 0.0, 0.0, m_iLinkd1);
    glEnd();
    glTranslated(0.0,0.0,m_iLinkd1);
    glRotated(m_iLinkTheta1, 0.0, 0.0, 1.0);
    glColor4f(0.0f,1.0f,0.0f,1.0f);
    glBegin(GL_LINES);
        glVertex3d( 0.0, 0.0, 0.0);
        glVertex3d( m_iLinkl1, 0.0, 0.0);
    glEnd();
    glTranslated(m_iLinkl1, 0.0, 0.0);
    glRotated(m_iLinkAlpha1, 1.0, 0.0, 0.0);
    glCallList(listJoint11);

glPushMatrix();    // Start joint3
    glColor4f(0.0f,1.0f,0.0f,1.0f);
    glBegin(GL_LINES);
        glVertex3f( 0.0, 0.0, 0.0);
        glVertex3f( 0.0, 0.0, m_iLinkd2);
    glEnd();
    glTranslated(0.0,0.0,m_iLinkd2);
    glRotated(m_iLinkTheta2, 0.0, 0.0, 1.0);
    glColor4f(0.0f,1.0f,0.0f,1.0f);
    glBegin(GL_LINES);
        glVertex3d( 0.0, 0.0, 0.0);
```

```

    glVertex3d( m_iLinkl2, 0.0, 0.0);
glEnd();
glTranslated(m_iLinkl2, 0.0, 0.0);
glRotated(m_iLinkAlpha2, 1.0, 0.0, 0.0);
glCallList(listJoint11);

glPushMatrix(); // Start joint4
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glBegin(GL_LINES);
    glVertex3f( 0.0, 0.0, 0.0);
    glVertex3f( 0.0, 0.0, m_iLinkd3);
glEnd();
glTranslated(0.0, 0.0, m_iLinkd3);
glRotated(m_iLinkTheta3, 0.0, 0.0, 1.0);
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glBegin(GL_LINES);
    glVertex3d( 0.0, 0.0, 0.0);
    glVertex3d( m_iLinkl3, 0.0, 0.0);
glEnd();
glTranslated(m_iLinkl3, 0.0, 0.0);
glRotated(m_iLinkAlpha3, 1.0, 0.0, 0.0);
glCallList(listJoint11);

glPushMatrix(); // Start joint5
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glBegin(GL_LINES);
    glVertex3f( 0.0, 0.0, 0.0);
    glVertex3f( 0.0, 0.0, m_iLinkd4);
glEnd();
glTranslated(0.0, 0.0, m_iLinkd4);
glRotated(m_iLinkTheta4, 0.0, 0.0, 1.0);
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glBegin(GL_LINES);
    glVertex3d( 0.0, 0.0, 0.0);
    glVertex3d( m_iLinkl4, 0.0, 0.0);
glEnd();
glTranslated(m_iLinkl4, 0.0, 0.0);
glRotated(m_iLinkAlpha4, 1.0, 0.0, 0.0);
glCallList(listJoint11);

glPushMatrix(); // Start joint6
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glBegin(GL_LINES);
    glVertex3f( 0.0, 0.0, 0.0);
    glVertex3f( 0.0, 0.0, m_iLinkd5);
glEnd();

```

```

glTranslated(0.0,0.0,m_iLinkd5);
glRotated(m_iLinkTheta5, 0.0, 0.0, 1.0);
glColor4f(0.0f,1.0f,0.0f,1.0f);
glBegin(GL_LINES);
    glVertex3d( 0.0, 0.0, 0.0);
    glVertex3d( m_iLinkl5, 0.0,0.0);
glEnd();
glTranslated(m_iLinkl5,0.0,0.0);
glRotated(m_iLinkAlpha5,1.0,0.0,0.0);
glCallList(listJoint11);

glPushMatrix();                // Start Approach
glColor4f(0.0f,1.0f,0.0f,1.0f);
glBegin(GL_LINES);
    glVertex3f( 0.0, 0.0, 0.0);
    glVertex3f( 0.0, 0.0,m_iLinkd6);
glEnd();
glTranslated(0.0,0.0,m_iLinkd6);
glRotated(m_iLinkTheta6, 0.0, 0.0, 1.0);
glColor4f(0.0f,1.0f,0.0f,1.0f);
glBegin(GL_LINES);
    glVertex3d( 0.0, 0.0, 0.0);
    glVertex3d( m_iLinkl6, 0.0,0.0);
glEnd();
glTranslated(m_iLinkl6,0.0,0.0);
glRotated(m_iLinkAlpha6,1.0,0.0,0.0);
glCallList(listJoint11);

    glPopMatrix();            //End Approach
    glPopMatrix();            //End Joint6
    glPopMatrix();            //End Joint5
    glPopMatrix();            //End Joint4
    glPopMatrix();            // End joint3
    glPopMatrix();            // End joint2
    glPopMatrix();            // End joint1

```

APPENDIX 2: THE FILE DATA FOR CASE STUDY ONE

```
// Version: 100
C6RRobot {
  Joint1    <0.000000 -90.000000 900.000000 188.000000 >
  Joint2    <90.000000 0.000000 0.000000 -950.000000 >
  Joint3    <0.000000 90.000000 0.000000 -225.000000 >
  Joint4    <0.000000 -90.000000 1300.000000 0.000000 >
  Joint5    <0.000000 90.000000 0.000000 0.000000 >
  Joint6    <0.000000 0.000000 200.000000 0.000000 >
  Joint0    <90.000000 0.000000 >
  Selection <12 22 34 42 51 62 72 >
  Name      <ABB IRB6400 >
  Color     <0.000000 1.000000 0.000000 1.000000 > // RGBA
  Origin    <0.000000 0.000000 0.000000 0.000000 >
  Rotation  <0.000000 0.000000 0.000000 >
  Translate <0.000000 0.000000 0.000000 >
}
C3dObjectCube {
  Depth    <300.500000 >
  Height   <350.000000 >
  Width    <1100.500000 >
  Name     <Cube 0 >
  Color    <0.000000 1.000000 0.000000 1.000000 > // RGBA
  Origin   <100.000000 1550.000000 700.000000 0.000000 >
  Rotation <0.000000 0.000000 0.000000 >
  Translate <0.000000 0.000000 0.000000 >
}
C3dTargetPoint {
  Thetas   <-29.496044 111.410835 23.982050
            39.582520 74.993484 -140.742645 >
  Target    <1399.500000 -650.250000 875.500000
            -135.000000 0.000000 90.000000 >
  Base     <90.000000 0.000000 >
  Name     <TargetPoint 0 >
  Color    <0.000000 1.000000 0.000000 1.000000 > // RGBA
  Origin   <0.000000 0.000000 0.000000 0.000000 >
  Rotation <0.000000 0.000000 0.000000 >
  Translate <0.000000 0.000000 0.000000 >
}
C3dTargetPoint {
  Thetas   <-22.917351 106.907448 29.804518
            43.543766 70.978897 -138.085571 >
  Target    <1399.500000 -450.250000 875.500000
            -135.000000 0.000000 90.000000 >
  Base     <90.000000 0.000000 >
  Name     <TargetPoint 1 >
  Color    <0.000000 1.000000 0.000000 1.000000 > // RGBA
  Origin   <0.000000 0.000000 0.000000 0.000000 >
  Rotation <0.000000 0.000000 0.000000 >
  Translate <0.000000 0.000000 0.000000 >
}
C3dTargetPoint {
  Thetas   <-15.635084 103.481071 34.043659
            47.771908 66.867920 -134.997482 >
```

```

    Target    <1399.500000 -250.250000 875.500000
              -135.000000 0.000000 90.000000 >
    Base      <90.000000 0.000000 >
    Name      <TargetPoint 2 >
    Color     <0.000000 1.000000 0.000000 1.000000 > // RGBA
    Origin    <0.000000 0.000000 0.000000 0.000000 >
    Rotation  <0.000000 0.000000 0.000000 >
    Translate <0.000000 0.000000 0.000000 >
  }
  C3dTargetPoint {
    Thetas    <-7.798542 101.334740 36.610901
              52.124931 62.563545 -131.604599 >
    Target    <1399.500000 -50.250000 875.500000
              -135.000000 0.000000 90.000000 >
    Base      <90.000000 0.000000 >
    Name      <TargetPoint 3 >
    Color     <0.000000 1.000000 0.000000 1.000000 > // RGBA
    Origin    <0.000000 0.000000 0.000000 0.000000 >
    Rotation  <0.000000 0.000000 0.000000 >
    Translate <0.000000 0.000000 0.000000 >
  }
  C3dTargetPoint {
    Thetas    <0.361441 100.639908 37.427048
              56.432465 58.060997 -128.051163 >
    Target    <1399.500000 150.250000 875.500000
              -135.000000 0.000000 90.000000 >
    Base      <90.000000 0.000000 >
    Name      <TargetPoint 5 >
    Color     <0.000000 1.000000 0.000000 1.000000 > // RGBA
    Origin    <0.000000 0.000000 0.000000 0.000000 >
    Rotation  <0.000000 0.000000 0.000000 >
    Translate <0.000000 0.000000 0.000000 >
  }
  C3dTargetPoint {
    Thetas    <8.486863 101.463837 36.458447
              60.455292 53.503487 -124.463440 >
    Target    <1399.500000 350.250000 875.500000
              -135.000000 0.000000 90.000000 >
    Base      <90.000000 0.000000 >
    Name      <TargetPoint 6 >
    Color     <0.000000 1.000000 0.000000 1.000000 > // RGBA
    Origin    <0.000000 0.000000 0.000000 0.000000 >
    Rotation  <0.000000 0.000000 0.000000 >
    Translate <0.000000 0.000000 0.000000 >
  }
  C3dTargetPoint {
    Thetas    <0.000000 90.000000 0.000000
              0.000000 0.000000 0.000000 >
    Target    <1688.000000 0.000000 2075.000000
              0.000000 90.000000 0.000000 >
    Base      <90.000000 0.000000 >
    Name      <TargetPoint 7 >
    Color     <0.000000 1.000000 0.000000 1.000000 > // RGBA
    Origin    <0.000000 0.000000 0.000000 0.000000 >
    Rotation  <0.000000 0.000000 0.000000 >
    Translate <0.000000 0.000000 0.000000 >
  }

```

VITA AUCTORIS

NAME: Zhongqing Ding

PLACE OF BIRTH: Sichuan, P. R. China

EDUCATION: University of Windsor, Windsor, Ontario
2003-2005 M. A. Sc.
Chongqing University, Chongqing, P. R. China
1988-1991 M. A. Sc.
1984-1988 B. A. Sc.