

DESIGN AND IMPLEMENTATION OF ROBOTIC CONTROL FOR INDUSTRIAL APPLICATIONS

By

Desmond Jeffery Will

A thesis submitted in compliance with the full requirements for the

**MAGISTER TECHNOLOGIAE:
ENGINEERING : ELECTRICAL**

In the

Faculty of Engineering

Port Elizabeth Technikon

January 2004

Promoters

Prof Theo van Niekerk

Mr Frank Adlam

Table of Contents

| | |
|-----------------------|------|
| List of Figures | vii |
| Abbreviations | xiii |
| Glossary | xv |

CHAPTER 1 INTRODUCTION

| | |
|---------------------------------------|---|
| 1.1 Objectives..... | 2 |
| 1.2 Hypothesis..... | 4 |
| 1.3 Delimitations of research..... | 5 |
| 1.4 Assumption..... | 5 |
| 1.5 Significance of the study..... | 6 |
| 1.6 Organization of Thesis..... | 6 |

CHAPTER 2 INDUSTRIAL ROBOT CONTROL

| | |
|---|----|
| 2.1 Background to Robotics..... | 8 |
| 2.1.1. Robot Structure..... | 14 |
| 2.1.2. Robot Arm Kinematics and Dynamics..... | 15 |
| 2.1.2.1. Direct and Inverse Kinematics..... | 18 |
| 2.1.2.2. Links, Joints and their parameters..... | 19 |

| | |
|--|----|
| 2.1.3. Robot Motion..... | 20 |
| 2.2 Manipulator Trajectory Control..... | 22 |
| 2.2.1 Robot Programming Language..... | 23 |
| 2.2.2 Characteristics of Robot-level languages..... | 24 |
| 2.2.3 Characteristics of Task-level languages..... | 26 |
| 2.3 Communication Controls..... | 31 |
| 2.3.1 Real-Time..... | 31 |
| 2.3.2 Local Area Networks (LAN's)..... | 32 |
| 2.3.2.1 Communication Models..... | 34 |
| 2.3.2.2 Real-Time Issues of TCP/IP..... | 40 |
| 2.4 Robot Sensing..... | 41 |
| 2.4.1 Machine Vision..... | 42 |
| 2.4.1.1 Background..... | 43 |
| 2.4.1.2 Vision Preprocessing..... | 48 |
| 2.4.1.3 Industrial Machine Vision..... | 54 |
| 2.4.1.4 Fundamentals – The formation of a Digital Image..... | 54 |
| 2.4.1.5 Vision Hardware Components..... | 54 |

| | | |
|---------|------------------------------------|----|
| 2.4.1.6 | Image Acquisition..... | 55 |
| 2.4.1.7 | Application of Vision include..... | 56 |
| 2.5 | Conclusion..... | 57 |

CHAPTER 3 SYSTEM SETUP : HARDWARE AND SOFTWARE ARCHITECTURE

| | | |
|---------|---|----|
| 3.1 | Introduction..... | 58 |
| 3.2 | Implementation Aspects for the Communications between hardware sub-systems..... | 62 |
| 3.3 | Industrial Robot..... | 64 |
| 3.3.1 | Bridge PC..... | 67 |
| 3.3.2 | Robot Vision System..... | 68 |
| 3.3.2.1 | CCD Camera Control..... | 68 |
| 3.3.2.2 | Flash Point 3D Frame Grabber Control..... | 69 |
| 3.4 | Software Architecture..... | 69 |
| 3.4.1 | Vision Recognition Classes and Image API's..... | 71 |
| 3.4.2 | Robot Trajectory Control..... | 74 |
| 3.4.3 | RobComm ActiveX Components and DDE Engine..... | 75 |
| 3.4.3.1 | RobComm ActiveX Components..... | 75 |

| | |
|---|----|
| 3.4.3.2 S4 DDE Server..... | 78 |
| 3.4.4 S4 Robot RAPID Program Structure..... | 80 |
| 3.5 Conclusion..... | 82 |

CHAPTER 4 PC-BASED ROBOT TRAJECTORY PATH CONTROL SYSTEM

| | |
|--|-----|
| 4.1 Robot RAPID Motion Control..... | 87 |
| 4.2 RAPID Program Data Types..... | 91 |
| 4.3 ABB Robot RAPID Program and Motion Control | 92 |
| 4.4 PC-Based Automated Robot Control using Kinematics..... | 101 |
| 4.4.1 Direct Kinematics Solution..... | 102 |
| 4.4.2 Direct Kinematic Computation for an IRB 1400 Robot..... | 103 |
| 4.4.3 Analytical computation of the inverse kinematic model..... | 109 |
| 4.4.4 Software Implementation of Robot Kinematics..... | 112 |
| 4.5 Software Modules for robot motion control..... | 116 |
| 4.6 OFF_LINE Development Environment to Robots Motion..... | 122 |
| 4.7 Communication Control..... | 125 |
| 4.8 Software modules to initiate Motion Control..... | 129 |

| | | |
|-------|---|-----|
| 4.9 | Remote RAPID DDE Robot Programming Environment..... | 145 |
| 4.9.1 | General DDE item syntax..... | 146 |
| 4.9.2 | Access method..... | 147 |
| 4.9.3 | Functional Group..... | 147 |
| 4.9.4 | Variable type..... | 147 |
| 4.9.5 | Variable name..... | 148 |
| 4.9.6 | Digital I/O variables..... | 148 |
| 4.9.7 | Digital I/O name example..... | 149 |
| 4.9.8 | Rapid program variables..... | 149 |
| 4.10 | Conclusion | 150 |

**CHAPTER 5 VISION SENSORY SYSTEM FOR PROFILE
RECOGNITION**

| | | |
|-------|-------------------------------------|-----|
| 5.1 | Software Components for Vision..... | 152 |
| 5.2 | Image Acquisition..... | 153 |
| 5.3 | Image Preprocessing..... | 156 |
| 5.3.1 | Image Filtering..... | 157 |
| 5.3.2 | Noise Cleaning..... | 159 |

| | | |
|-------|--------------------------------------|------|
| 5.3.3 | Averaging..... | 160. |
| 5.3.4 | Image Thresholding..... | 160 |
| 5.4 | Boundary Detection..... | 162 |
| 5.4.1 | Edge Detection..... | 162 |
| 5.4.2 | Edge Linking..... | 162 |
| 5.4.3 | Edge Following and Thinning..... | 164 |
| 5.5 | Extraction of the Image Profile..... | 165 |
| 5.6 | Conclusion..... | 167 |

CHAPTER 6 CONCLUSION

| | | |
|-----|---|------------|
| 6.1 | Project Results..... | 169 |
| 6.2 | Accomplishments and Contributions of final results..... | 171 |
| 6.3 | Problems encountered..... | 172 |
| 6.4 | Possible extensions and conclusions..... | 173 |
| | References | 175 |

List of Figures

CHAPTER 1 INTRODUCTION

Figure 1.1: System architecture for an industrial application.....3

CHAPTER 2 INDUSTRIAL ROBOT CONTROL

Figure 2.1: Illustration of a Cincinnati Milacron T3 robot arm.....10

Figure 2.2: Illustration of various robot arm categories.....11

Figure 2.3: Relationship of fixed automation, programmable automation, and flexible automation as a function of production volume and product variety.....13

Figure 2.4: ABB 1400 Robot Manipulator, 6 Axis [6 DOF].....14

Figure 2.5: ABB 1400 Robot Controller.....15

Figure 2.6: Reference and body-attached co-ordinate system.....16

Figure 2.7: Illustration of an OUVW rotating co-ordinate system.....17

Figure 2.8: Illustration of a Rotation 3x3 Matrix for the 3 axes [x,y,z]17

Figure 2.9: A simple diagram indicating the relationship between direct and inverse robot kinematics.....18

Figure 2.10: A PUMA robot arm illustrating joints and links.....19

Figure 2.11: Link co-ordinate system and its parameters.....20

Figure 2.12: Co-ordinate frame chain.....21

Figure 2.13: Task planner.....27

| | |
|--|----|
| <i>Figure 2.14: Industrial LAN Network Architecture</i> | 34 |
| <i>Figure 2.15: DNA by DEC and Internet model</i> | 35 |
| <i>Figure 2.16: OSI seven-layer model</i> | 37 |
| <i>Figure 2.17: Data flow</i> | 37 |
| <i>Figure 2.18: IP Frame</i> | 39 |
| <i>Figure 2.19: TCP Frame</i> | 39 |
| <i>Figure 2.20: UDP Frame</i> | 40 |
| <i>Figure 2.21: Client-server approach using TCP/IP</i> | 41 |
| <i>Figure 2.22: Robot Vision basic structure</i> | 44 |
| <i>Figure 2.23: This diagram simplifies the relationship between the three functions of machine vision</i> | 45 |
| <i>Figure 2.24: Typical two peak intensity histogram</i> | 52 |
| <i>Figure 2.25: Thresholding to gray-level Image</i> | 52 |
| <i>Figure 2.26: The components of a robot vision system</i> | 55 |

CHAPTER 3 SYSTEM SETUP : HARDWARE AND SOFTWARE ARCHITECTURE

| | |
|--|----|
| <i>Figure 3.1: Communication Interfaces and Software development components for hardware sub-systems</i> | 58 |
| <i>Figure 3.2a: 1400 ABB Robot, Vision Feedback Camera & Bridge PC – system setup</i> .. | 63 |

| | |
|--|----|
| Figure 3.2b: 1400 ABB Robot Controller & Bridge PC – system setup..... | 63 |
| Figure 3.2c: 1400 ABB Robot Controller & Bridge PC Hardware – system setup..... | 64 |
| Figure 3.3: Robot Controller Ethernet Communication Configuration..... | 67 |
| Figure 3.4: Experimental Ethernet TCP/IP Configuration Robot & LAN Connection.. | 68 |
| Figure 3.5: Software architecture – Integrated Robot Vision Control system..... | 70 |
| Figure 3.6: illustrates frame grabber configuration sequence..... | 71 |
| Figure 3.7: Vision Profile Extraction Architecture..... | 73 |
| Figure 3.8 Robot trajectory generation engine..... | 74 |
| Figure 3.9: FactoryWare Configuration..... | 76 |
| Figure 3.10: S4 Robot Controller communication protocols..... | 78 |
| Figure 3.11: DDE addressing structure..... | 79 |
| Figure 3.12: DDE Server Engine..... | 80 |
| Figure 3.13: S4 ABB Controller RAPID program structure..... | 81 |
| Figure 3.14: ABB Robot RAPID trajectory path structure..... | 82 |

CHAPTER 4 PC-BASED ROBOT TRAJECTORY PATH CONTROL SYSTEM

| | |
|--|-----|
| <i>Figure 4.1: PC-Based Robot Control Architecture.</i> | 85 |
| <i>Figure 4.2: MOVE command architecture.</i> | 88 |
| <i>Figure 4.3: Motion path type commands.</i> | 89 |
| <i>Figure 4.4: Positioning the robot.</i> | 90 |
| <i>Figure 4.5: Robtarget variable declaration.</i> | 91 |
| <i>Figure 4.6: Quarternions algorithms.</i> | 92 |
| <i>Figure 4.7: RAPID Robot program architecture.</i> | 93 |
| <i>Figure 4.8: Home sub-routine function.</i> | 96 |
| <i>Figure 4.9: Robot camera view sub-routine.</i> | 97 |
| <i>Figure 4.10: CCD camera pixel ration calibration.</i> | 98 |
| <i>Figure 4.11a: Mimic profile object sub-routine.</i> | 99 |
| <i>Figure 4.11b: The direct and inverse kinematics problems.</i> | 100 |
| <i>Figure 4.12: The direct and inverse kinematics problems.</i> | 102 |
| <i>Figure 4.13: Link parameters for ABB IRB 1400 Industrial Robot.</i> | 103 |
| <i>Figure 4.14: ABB Industrial Robot link co-ordinate transformation matrices.</i> | 106 |

| | |
|---|-----|
| <i>Figure 4.15: Planar 3-R Manipulator with the three reference joint angles.</i> | 109 |
| <i>Figure 4.16: Robot Path Engine environment.</i> | 114 |
| <i>Figure 4.17: Program Modules and architecture for motion control.</i> | 117 |
| <i>Figure 4.18: Remote RAPID programming environment.</i> | 122 |
| <i>Figure 4.19: RobComm Ethernet communication setup “ IP:100.100.100.101”.</i> | 127 |
| <i>Figure 4.20: RobComm Active Server establishing communication with ABB Robot Controller.</i> | 128 |
| <i>Figure 4.21: ABB Robot trajectory motion control.</i> | 129 |
| <i>Figure 4.22: software environment for manual robot control commands.</i> | 130 |
| <i>Figure 4.23: RAPID Sub-Routine function commands.</i> | 132 |
| <i>Figure 4.24: software environment for manual robot control commands.</i> | 135 |
| <i>Figure 4.25: software - Robot control tool bar.</i> | 137 |
| <i>Figure 4.26: Co-ordinate Access Database Structure.</i> | 142 |
| <i>Figure 4.27: Microsoft Excel DDE data simulation with DDE RobComm Server.</i> | 146 |

CHAPTER 5 VISION SENSORY SYSTEM FOR PROFILE RECOGNITION

| | |
|--|-----|
| <i>Figure 5.1: Software components to implement a vision sensory system.</i> | 153 |
| <i>Figure 5.2: Image capture via frame grabber.</i> | 154 |

| | |
|---|-----|
| <i>Figure 5.3: Memory allocation for the 3D Flash Point Frame Grabber.....</i> | 155 |
| <i>Figure 5.4: Mechanism utilized to transfer the image into the allocated memory.....</i> | 155 |
| <i>Figure 5.5: 3x3 matrix high-pass convolution filter.....</i> | 158 |
| <i>Figure 5.6: Convolution filter mechanism.....</i> | 158 |
| <i>Figure 5.7: 3x3 matrix low-pass convolution filter.....</i> | 159 |
| <i>Figure 5.8: 3x3 matrix convolution filter.....</i> | 159 |
| <i>Figure 5.9: 3x3 matrix convolution noise smoothing filter.....</i> | 159 |
| <i>Figure 5.10: 3x3 matrix convolution filter.....</i> | 159 |
| <i>Figure 5.11: 3x3 matrix averaging convolution filter.....</i> | 160 |
| <i>Figure 5.12: 3x3 matrix convolution filter implemented as a software call function....</i> | 160 |
| <i>Figure 5.13: Image after threshold mechanism has been applied.....</i> | 161 |
| <i>Figure 5.14: (a) Discontinuity from A to B (b) Principle of linear interpolation.....</i> | 163 |
| <i>Figure 5.15: Block diagram of interpolation principle.....</i> | 164 |
| <i>Figure 5.16: Profile of the object.....</i> | 165 |
| <i>Figure 5.17: Object Profile</i> | 166 |
| <i>Figure 5.18: Object Profile Image Co-ordinate MAP.....</i> | 167 |

Abbreviations

| | |
|------|---|
| ABB | Asea Brown Boverly |
| ADC | Analog-to-Digital Converter |
| API | Application Interface |
| ATM | Automatic Teller Machine |
| CAD | Computer Aided Design |
| CCD | Charge-Coupled Device |
| CIM | Communication Integration Manufacturing |
| CMYK | cyan, magenta, yellow, and black |
| DAC | Digital Analog Controller |
| DAQ | Data Acquisition Card |
| DCS | Distribution Control System |
| DDE | Dynamics Data Exchange |
| DEC | Digital Equipment Company |
| DNA | Digital Network Architecture |
| DLL | Dynamic Link Library |
| DOF | Degrees of Freedom |
| FP3D | Flash Point Three Dimensional |
| FMS | Flexible Manufacturing System |
| FTAM | File, Transfer, Access, and Management |
| GUI | Graphics User Interface |
| HSB | hue, saturation, and brightness |
| IMV | Industrial Machine Vision |
| IP | Internet Protocol |
| I/O | Inputs/Outputs |
| IKPM | Inverse Kinematics position model |
| LAN | Local Area Network |
| LAT | Local Area Transport |
| MFC | Microsoft Foundation Class |
| MMI | Man Machine Interface |
| OLP | Off-Line Programming |

| | |
|--------|---|
| PC | Personnel Computer |
| RAP | Rapid Application Protocol |
| RGB | Red, Green, and Blue |
| S/N | Signal/Noise |
| TCP | Tool Centre Point |
| TCP/IP | Transmission Control Protocol / Internet Protocol |
| UDP | Universal Data Protocol |
| VGA | Video Graphics Accelerator |
| VT | Virtual Terminal |
| WAN | Wide Area Network |
| 3D | Three Dimensional |
| VPR | Vision Pixel Ratio |

Glossary

A

- Actuator:** A motor or transducer that converts electrical, hydraulic, or pneumatic energy into power for motion or action.
- Anthropomorphic Robot:** Also known as a jointed-arm robot. A robot with all rotary joints and motions similar to a person's arm.
- Application (Computer):** A program which is designed to facilitate the user to perform prescribed tasks.
- Articulated Robot:** A robot arm which contains at least two consecutive revolute joints acting around parallel axes resembling human arm motion. The work envelop is formed by partial cylinders or spheres.
- ActiveX:** An ActiveX control is an extension to Visual Basic Toolbox. When adding an ActiveX component, it becomes a part of the development and run-time environment and provides new functionality for the application.
- Algorithm:** Normally used as a basis for writing a computer program. This is a set of rules with a finite number of steps for solving a problem.

Application Layer:

The highest layer of the 7-Layer OSI model structure, containing all user or application programs.

B

Binary Image:

A digitized image in which the brightness of the pixel can have only two different values, such as white and black.

Binarization:

A process which converts a grayscale image into binary image.

C

Cell:

A manufacturing unit consisting of two or more work stations or machines, and the material transport mechanisms and storage buffers that interconnect them.

Chain Codes:

A set of straight line segments of specified length and direction which are used to represent a boundary. Typically, this representation is established on a rectangular grid using 4- or 8-connectivity.

Classification:

A process of grouping objects together into classes (subpopulations) according to their perceived likenesses or similarities.

Closed-Loop Control:

The use of a feedback loop to measure and compare actual system performance with desired performance. This allows the robot control to make any necessary adjustment.

Computer Vision:

Also known as machine vision. The use of computers or other electronic hardware to acquire, interpret, and process visual information. It involves the use of visual sensors to create an electronic or numerical analog of a visual scene, and computer processing to extract intelligence from this representation.

Configuration:

The description and specification of mechanism, including the kinematics and/or structural features, the number of degree of freedom, the joint travel range, and the type of drive for the robot.

Control system:

A system in which a series of measured values are used to make a decision on manipulating various parameters in the system to achieve a desired value of the original measured value.

Convolution:

An image enhancement technique in which each pixel is subjected to a mathematical operation that groups it with its nearest neighbours and calculates its value accordingly.

Coordinate Transformation:

In robotics, a 4×4 matrix used to describe the positions and orientations of coordinate frames in space. It is a suitable data structure for the description of the relative position and orientation between objects. Matrix multiplication of the transformations establishes the overall relationship between objects.

D**Degree of Freedom:**

The number of independent ways the end effectors can move. It is defined by the number of rotational or translational axes through which motion can be obtained. Every variable representing a degree of freedom must be specified if the physical state of the manipulator is to be completely defined.

E**Edge Detection:**

An image analysis technique in which information about a scene is obtained without acquiring an entire image. Locations of transition from black to white and white to black are recorded, stored, and connected through a process called connectivity to separate objects in the image into blobs. The blobs can then be analyzed and recognized for their respective features.

End Effector:

Also known as end-of-arm tooling or, more simply, hand. The subsystem of an industrial robot system that links the mechanical portion of the robot (manipulator) to the part being handled or worked on, and gives the robot the ability to pick up and transfer parts and/or handle a multitude of differing tools to perform work on parts.

End-of-Arm Tooling:

A device, commonly made up of four distinct elements, which provide for (1) attachment of the hand or tool to the robot tool mounting plate, (2) power for actuation of tooling motions, (3) mechanical linkages, and (4) sensors integrated into the tooling.

F**Feature Extractor:**

A program used in image analysis to compute the values of attributes (features) considered by the user to be possibly useful in distinguishing between different shapes of interest.

Feedback:

The signal or data sent to the control system from a controlled machine or process to denote its response to the command signal.

Frame: A full video image comprising of two fields. A PAL frame has a total 625 lines (an NTSC frame has 525 lines).

Frame Grabber: An image processing peripheral that samples, digitizes and stores a television camera frame in computer memory.

G

Grayscale Image: A digitized image in which the brightness of the pixels can have more than two values which are typically 128 or 256. A grayscale image requires more storage space and more sophisticated image processing than a binary image.

H

Homogeneous Transform: A 4×4 matrix which represents the rotation and translation of vectors in the joint coordinate systems. It is used to compute the position and orientation of any coordinate system with respect to any other coordinate system.

Handshaking: Exchange of predefined signals between two devices establishing a connection.

I

Image Analysis: The interpretation of data received from an imaging device.

Imaging: The analysis of an image to derive the identity, position, orientation, or condition of objects in the scene. Dimensional measurements may also be performed.

Intelligent Robot: A robot that can be programmed to execute performance choices contingent on sensory inputs.

Interface: A shared boundary which might be a mechanical or electrical connection between two devices; it might be a portion of computer storage accessed by two or more programs; or it might be a device for communication with a human operator.

J

Joint: A rotary or linear articulation or axis of rotational or translational (sliding) motion in a manipulator system.

K

Kinematics (Robot):

The study of the mapping of joint coordinates to link coordinates in motion, and inverse mapping of link coordinates to joint coordinates in motion.

L

Linear Interpolation:

A computer function automatically performed in the control that defines the continuum of points in a straight line based on only two taught coordinate positions. All calculated points are automatically inserted between the taught coordinate positions upon playback.

M

Manipulator:

A mechanism, usually consisting of a series of segments, or links, jointed or sliding relative to one another, for grasping and moving objects, usually in several degrees of freedom. A manipulator refers mainly to the mechanical aspect of a robot.

Mathematical Modeling:

Using mathematics, computers and engineering to describe, simulate, analyse and improve processes and systems.

Modular Programming:

A software design methodology which requires components to be developed in isolation so as to facilitate the integration of different modules.

N

Network:

An interconnected group of nodes or stations.

Network architecture:

A set of design principles, including the organization of functions and the description of data formats and procedures, used as the basis for the design and implementation of a network (ISO).

O

Orientation:

Also known as positioning. The consistent movement or manipulation of an object into a controlled position and attitude in space.

P

Path:

A series of positions in space that a robot manipulator or grasped object moves through.

Pixel:

Also known as photo-element or photosite. This is a digital picture or sensor element. Pixel is short for picture-cell.

Peer-to-Peer:

A connection between only two items of equipment.

Protocols: A format set of conventions governing the formatting and relative timing of messages exchange between two communicating systems.

R

Real-time: A system is capable of operating in real-time when it is fast enough to react to the real-world events.

Recognition: A labeling process, that is, is the function of recognition algorithms is to in a scene and to assign a label to that object.

Robot: A robot is a reprogrammable, multifunctional manipulator designed to move material, parts tools, or specialized devices through variable programmed motions for the performance of a variety of tasks.

Robot Calibration (for vision): The act of determining the relative orientation of the camera coordinate system with respect to the robot coordinate system.

Robotics: The science of designing, building, and applying robots.

S

Sensing: The feedback from the environment of the robot which enables the robot to react to its environment. Sensory inputs may come from a variety of sensor types

including proximity switches, force sensors, tactile sensors, and machine vision systems.

Sensor:

A device such as a transducer that detects a physical phenomenon and relays information to a control device.

T

Teach Pendant:

Also known as teach box. A portable, hand-held programming device connected to the robot controller containing a number of buttons, switches, or programming keys used to direct the controller in positioning the robot and interfacing with auxiliary equipment. It is used for teach pendant programming.

Thresholding:

A procedure of binarization of an image by segmenting it to black and white regions (represented by ones and zeroes). The gray level of each pixel is compared to a threshold value and then set to 0 or 1 so that binary image analysis can then be performed.

Tool Centre Point (TCP):

A tool-related reference point that lies along the last wrist axis at a user-specified distance from the wrist.

Trajectory:

A sub-element of a cycle that defines lesser but integral elements of the cycle. A trajectory is made up of points at which the robot performs or passes through an operation, depending on the programming.

Translation: A movement such that all axes remain parallel to what they were (i.e. without rotation).

V

Vision, 2D: The processing of 2D images by a computer vision system to derive the identity, position, orientation, or condition of objects in the scene.

Vision System: A system interfaced with a robot which locates a part, identifies it, directs the gripper to a suitable grasping position, picks up the part, and brings the part to the work area. A coordinate transformation between the camera and the robot must be carried out to enable proper operation of the system.

VGA: Video Graphics Array. This standard utilizes analog signals only offering a resolution of 640x480 pixels, a palette of 256 colours out of 256000 colours and the ability to display 16 colours at the same time.

CHAPTER 1 INTRODUCTION

“One machine can do the work of a hundred ordinary men, but no machine can do the work of one extraordinary man “ [1]

- Elbert Hubbard

Background to Industrial Robot Automation

With the pressing need for increased productivity and delivery of end products of uniform quality, industry is turning more and more to computer-based automation.

At the present time, most of industrial automated manufacturing is carried out by special-purpose machines, designed to perform specific functions in a manufacturing process.

The inflexibility and generally high cost of these machines often referred to as hard automation systems, have led to a broad-based interest in the use of robots capable of performing a variety of manufacturing functions in a more flexible working environment and at lower production costs.

A robot is a reprogrammable general-purpose manipulator with external sensors that can perform various assembly tasks. A robot may possess intelligence, which is normally due to computer algorithms associated with its controls and sensing systems. Industrial robots are general-purpose, computer-controlled manipulators consisting of several rigid links connected in series by revolute or prismatic joints.

Most of today's industrial robots, though controlled by mini and microcomputers are basically simple positional machines. They execute a given task by playing back a

prerecorded or preprogrammed sequence of motion that has been previously guided or taught by the hand-held control teach box. Moreover, these robots are equipped with little or no external sensors for obtaining the information vital to its working environment.

As a result robots are used mainly for relatively simple, repetitive tasks. More research effort has been directed in sensory feedback systems, which has resulted in improving the overall performance of the manipulator system.

An example of a sensory feedback system would be: *a vision Charge-Coupled Device (CCD) system*. This can be utilized to manipulate the robot position dependant on the surrounding robot environment (various object profile sizes). This vision system can only be used within the robot movement envelope.

1.1 Objectives

Due to the rapid changes in the manufacturing environment, there has become a growing need for integrated vision based systems and automated remote robot trajectory motion control. Figure 1.1 illustrates the architecture proposed to fulfill the overall objective for an industrial application.

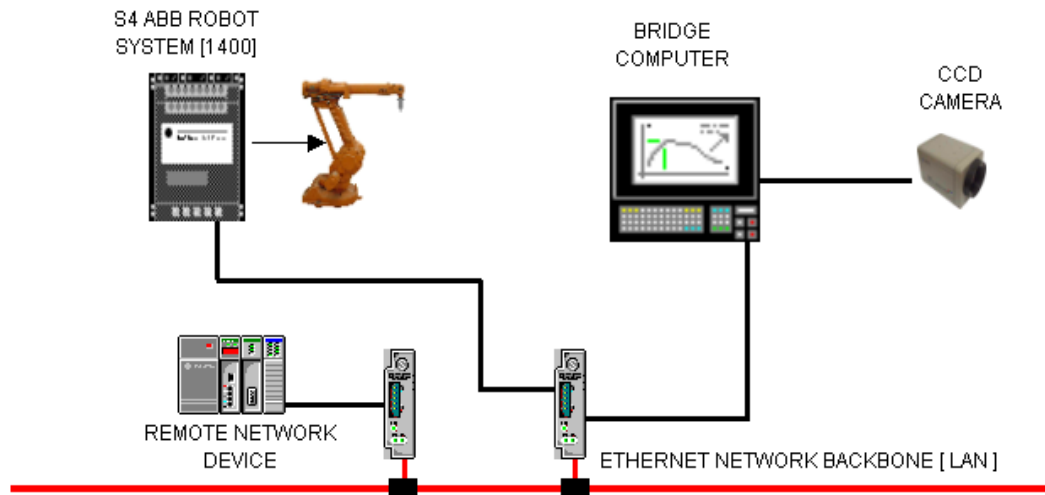


Figure 1.1: System architecture for an industrial application

The system overview comprises of the following base components, illustrated in Figure 1.1:

- Industrial Asea Brown Boverly (ABB) robot Manipulator and Controller
- Standards AMD Bridge Personal Computer (PC)
- CCD Vision Camera
- Ethernet Network system

In order to achieve the objective of a vision-based automatic robotic trajectory motion control, the following sub-problems have been identified:

- Study the fundamentals of an ABB robot (ABB 1400 Series), its control mechanism and principles in order to develop a user interface for industrial robotic communication.

- Study the RAPID program language of the industrial robot in order to realize basic operation of movement and execution of simple tasks.
- Investigate the remote communication for industrial specialized machines via standard Ethernet communication.
- Build an interface for robot communication based on Ethernet hardware for local and remote access to industrial robots. The remote access will be based on TCP/IP protocols.
- Study the kinematics of the industrial robot and its possible operations (movement) for industrial applications such as, material handling, sealing, etc.
- Develop a software interface to manipulate industrial robot path motion according to operational tasks. Interpolation and dynamic control may be taken into account in the path control. The control mechanism may be achieved via an industrial SCADA (MMI).
- Develop algorithms to extract the object profile from a dynamic image, which can be utilized to provide the industrial robot with position feedback. This creates an automated closed loop system.

1.2 Hypothesis

A vision-based CCD sensory system can be integrated with an industrial robot to sense an object's profile and manipulate the robot's position online, according to the object profile, resulting in increased robot flexibility and system performance.

1.3 Delimitations of research

- The remote RAPID program language environment will not include a fully automatic environment. It will focus only on the basic robot movement.
- No Ethernet hardware will be designed and manufactured for both the Bridge PC and Robot controller, neither will it include the TCP/IP protocol software which is used to communicate with the robot via the network software layers. The Ethernet software protocol will be handled via the ProComm software platform, which has been developed by the manufacturer, ABB. The use of this platform reduces software development time.
- RobComm frees one from underlying communication protocols therefore in this study more time can be spent developing the user interface. RobCOMM uses ActiveX controls (Rimbase.ocx), which enables the user to interact between standard software packages such as Visual C and Visual Basic and the industrial robot control system during real time operations.
- The user interface will be developed using only Visual C and Visual Basic. This will focus only on the visual control of the industrial robot, such as system status and axis orientation position.

1.4 Assumption

The necessary hardware and software tools required to do the research will be available, as well as providing full functionality to achieve all of the abovementioned objectives.

1.5 Significance of the study

Currently in industry, industrial robots are beginning to take over repetitive tasks, which were previously performed by humans. Industrial robots significantly improve the quality of the end product. It also results in improved efficiency as high product volumes are produced.

This research will attempt to illustrate the fact that robotic equipment should possess some form of sensory feedback, which would give the robot the ability to automatically manipulate the robot path without the intervention of human interaction.

1.6 Organization of Thesis

Objectives, hypotheses, delimitations, and significance of this research project are introduced in Chapter 1. Chapter 2 analyzes the relevant theories, corresponding components, related technology, and up-to-date development in the robot sensory feedback devices, as well as remote automated robot control in terms of literature survey. Chapter 3 describes the overall system setup, hardware architecture, software components, implementation of subsystems, and integration of individual subsystems to form a platform for profile recognition and integrated robot control via a PC-Based system. Chapter 4 involves the architecture of PC-Based robot trajectory path planning system, with emphasis on remote robot programming environment, robot kinematics, as well as their implementation. Chapter 5 involves the architecture of robot vision recognition system, digital image processing techniques, algorithms of profile extraction, as well as their implementation. Chapter 6 describes the system integration, providing a

detailed insight of all the components required to achieve the overall system objective. Chapter 7 provides the conclusion to the research, introducing possible future extensions and developments to this research platform.

CHAPTER 2

INDUSTRIAL ROBOT CONTROL

This chapter will serve as a background to topics and mathematic fundamentals related to this dissertation. This includes robotic manufacturing systems and robotic interfacing software. New technologies and trends related to these areas will also be discussed. In order to understand the project as a whole and its relevance to manufacturing, industrial applications will also be highlighted.

Background to Robotics

With a pressing need for increased productivity and the delivery of end products of uniform quality, industry is turning more and more towards computer-based automation. Most automated manufacturing tasks, at the present time, are carried out by special-purpose machines designed to perform a predetermined function in a manufacturing process. The inflexibility and generally high cost of these machines, often called *hard automation systems*, have led to a broad-based interest in the use of robots capable of performing a variety of manufacturing functions in a more flexible working environment. This also results in lower production costs.

The word **ROBOT** originated from the Czech word “robota”, meaning – **WORK**. Webster’s dictionary defines a robot as: “ *an automatic device that performs functions ordinarily ascribed to human beings.* “

A definition used by the Robot Institute of America gives a more precise description of an industrial robot: “ a robot is a reprogrammable multi-functional manipulator designed

to move materials, parts, tools or specialized devices, through variable programmed motions for the performance of a variety of tasks.“ **In short, a robot is a reprogrammable general-purpose manipulator with external sensors that can perform assembly tasks.** With this definition, a robot must possess intelligence, which is normally due to computer algorithms associated with its control and sensing systems.

An industrial robot is a general-purpose, computer-controlled manipulator consisting of several rigid links connected in series by revolute or prismatic joints. One end of the chain is attached to a supporting base, while the other end is free and equipped with a tool to manipulate objects or perform assembly tasks. The motion of the joints results in relative motion of the links. Mechanically, a robot is composed of an arm, wrist and tool. The work volume is the sphere of influence of a robot whose arm can deliver the wrist subassembly unit to any point within the sphere. The arm subassembly generally can move within 3 degrees of freedom (3DOF)[15][16].

The wrist subassembly unit usually consists of **three rotary motions.** These motions are defined as -

- pitch,
- yaw; and
- roll.

Hence, for a six-jointed robot the arm subassembly is the positioning mechanism, while the wrist subassembly is the orientation mechanism.

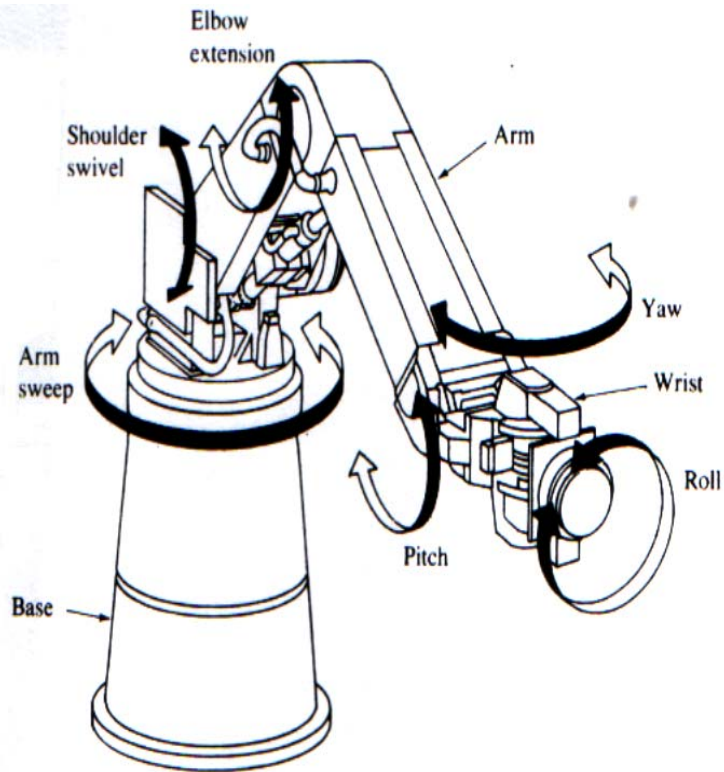


Figure 2.1: Illustration of a Cincinnati Milacron T3 robot arm [1]

Many commercially available industrial robots are widely used in manufacturing and assembly tasks, such as material handling, spot / arc welding, parts assembly, spray painting, loading and unloading numerically controlled machines.

Robots are defined into four basic motion defining categories, illustrated in Figure 2.2.

- a. Cartesian Co-ordinates
- b. Cylindrical Co-ordinates
- c. Spherical Co-ordinates
- d. Revolute or Articulate Co-ordinates

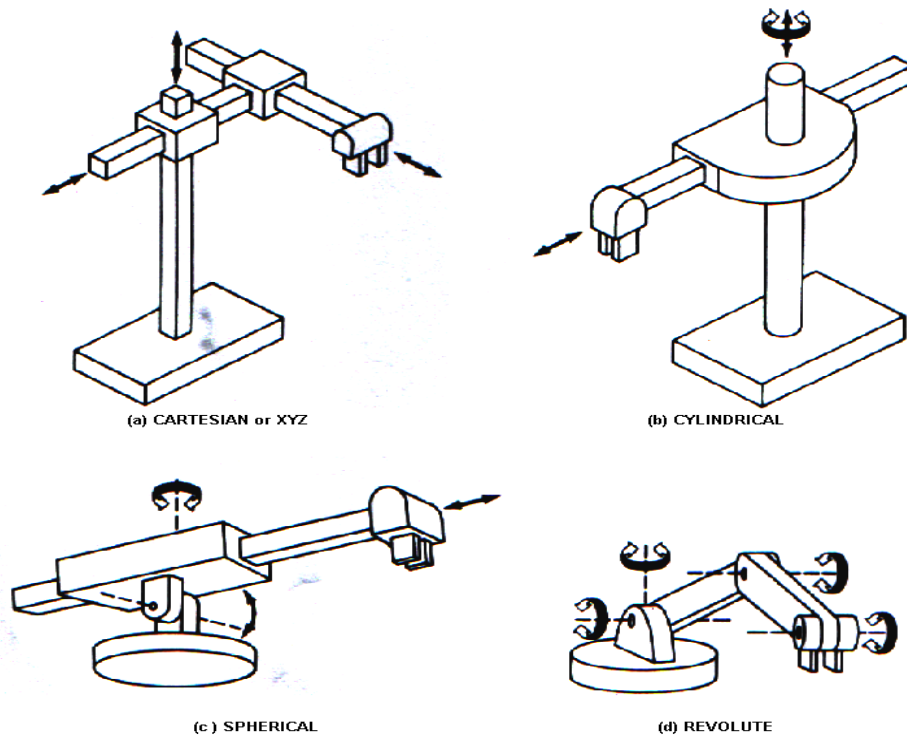


Figure 2.2: Illustration of various robot arm categories [1]

Most of today's industrial robots are controlled by mini- and micro-computers and are basically simple positional machines. They execute a given task by playing back prerecorded or preprogrammed sequences of motion that have been previously guided or taught by the user with a hand-held control-teach pendant. Moreover, these robots are equipped with little or no external sensors for obtaining the information vital to its working environment. As a result, robots are used mainly for relatively simple, repetitive tasks. More research effort is being directed towards improving the overall performance of the manipulator system. *Automation and Robotics* are two closely related technologies.

AUTOMATION is defined as: *“a technology that is concerned with the use of mechanical, electronic, and computer-based systems in the operation and control of production.”*

Examples include - transfer lines, mechanized assembly machines, feedback control systems, numerically controlled machine tools and robots. Accordingly, robotics is a form of industrial automation.

There are three broad classes of industrial automation:

- (i) **Fixed Automation** – is used when the volume of production is very high and it is therefore appropriate to design specialized equipment to process the product. An example of this would be in the automotive industry, where highly integrated transfer lines consisting of several dozen workstations are used to perform machining operations on engine and transmission components.
- (ii) **Programmable Automation** – is used when the volume of production is relatively low and there are a variety of products to be made. In this case the production equipment is designed to be adaptable to variations in product configuration. This adaptability feature is accomplished by operating the equipment under the control of a “program” of instructions, which has been prepared especially for the given product.
- (iii) **Flexible Automation** – other terms used include FMS and Computer-Integrated Manufacturing Systems.” This type of automation is most suitable

for the mid-volume production range. Flexible automated systems typically consist of a series of workstations that are interconnected by a materials-handling and storage system. A central computer is used to control the various activities that occur in the system.

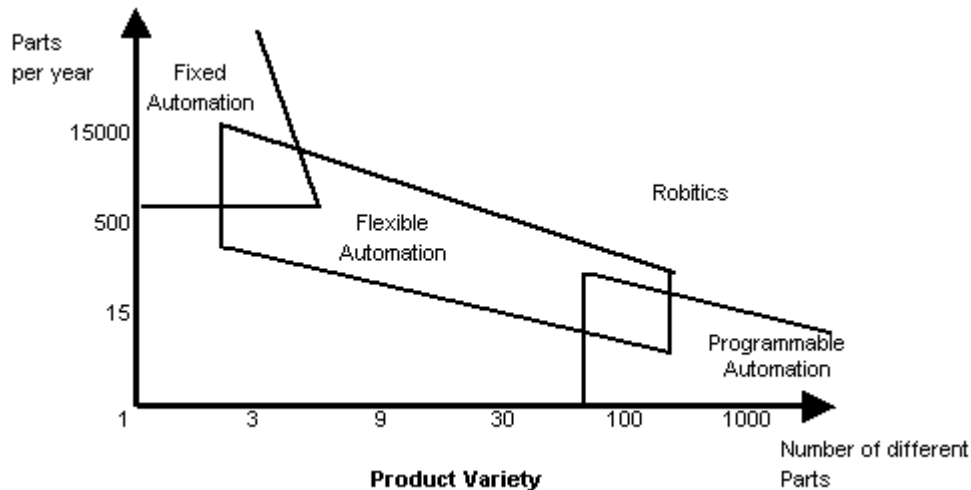


Figure 2.3: Relationship of fixed automation, programmable automation, and flexible automation as a function of production volume and product variety [2]

Of the three types of automation, robotics coincide most closely with programmable automation. An industrial robot is a general-purpose, re-programmable machine, which possesses certain anthropomorphic or humanlike characteristics. The most typical humanlike characteristic of existing robots is their movable arms. The robot can be programmed to move its arm through a sequence of motions in order to perform some useful task. It will repeat that motion pattern over and over until reprogrammed to perform some other task. Hence, the programming feature allows robots to be used for a variety of different industrial operations, many of which involve the robot working together with other pieces of automated or semi-automated equipment [2][3].

2.1.1 Robot Structure

A robot is made up of two main parts:

The *manipulator* is the part of the robot that consists of links connected by revolute or prismatic joints as illustrated in Figure 2.4[22].

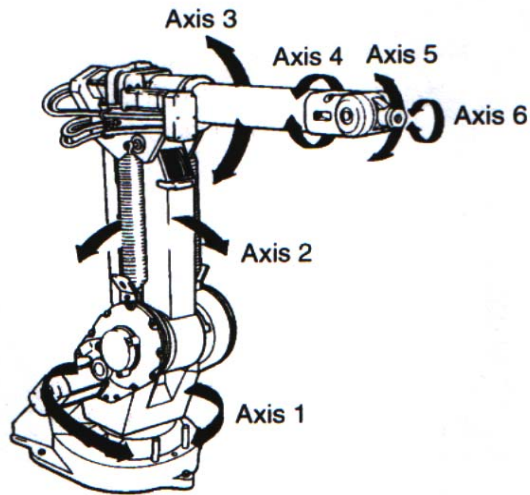


Figure 2.4: ABB 1400 Robot Manipulator, 6 Axis [6 DOF] [22]

The *controller* contains the electronics required to control the manipulator, external axes and peripheral equipment, as illustrated in Figure 2.5[22].

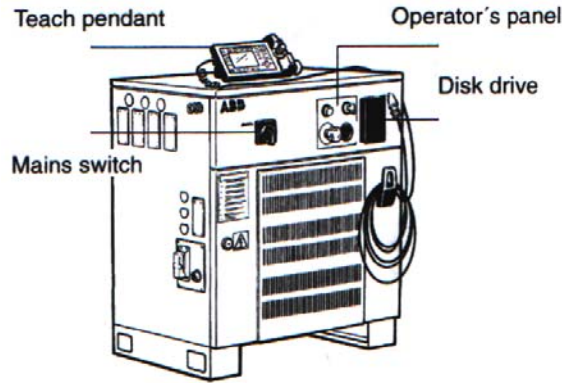


Figure 2.5: ABB 1400 Robot Controller [22]

2.1.2 Robot Arm Kinematics and Dynamics

Robot arm kinematics deals with the analytical study of the geometry of motion of a robot arm with respect to a fixed reference co-ordinate system. This system is without regard to the force / moments that cause the motion. Thus, **kinematics** deals with the analytical description of the spacial 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 [10].

The two fundamental concepts with respect to robot arm kinematics are -

- direct kinematics; and
- inverse kinematics.

Since independent variables in a robot arm are the joint variables, and a task is usually stated in terms of the reference co-ordinate frame, the inverse kinematics problem is used more frequently.

A systematic and generalized approach which utilizing *matrix algebra* to describe and represent the spatial geometry of the links of a robot arm by systematically establishing a co-ordinate system (body-attached frame) to each link of an articulated chain [42]. This method uses a 3×3 homogeneous transformation matrix to describe the special relationship between two adjacent mechanical links and reduces the direct kinematic problem to finding an equivalent 3×3 homogeneous transformation matrix [1]. Thus, through sequential transformations, the end-effector expressed in the “hand co-ordinates” can be transformed and expressed in the “base co-ordinates” which make up the inertial frame of this dynamic system.

Rotation matrices which comprise of a 3×3 rotation matrix can be defined as a transformation matrix which operates on a position vector in a three-dimensional Euclidean space and maps its coordinates expressed in a rotated coordinate system $OUVW$ (body-attached frame) to a reference coordinate system $OXYZ$, as shown in Figure

2.6. [22][3][10]

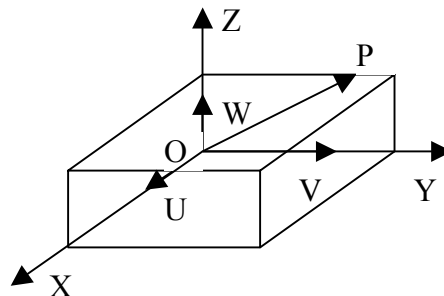


Figure 2.6: Reference and body-attached co-ordinate system [22][1]

Figure 2.7 shows the $OUVW$ coordinate system rotated at an α angle about the OX axis, then rotated an ϕ angle about the OY axis, and then rotated an θ angle about the OY .

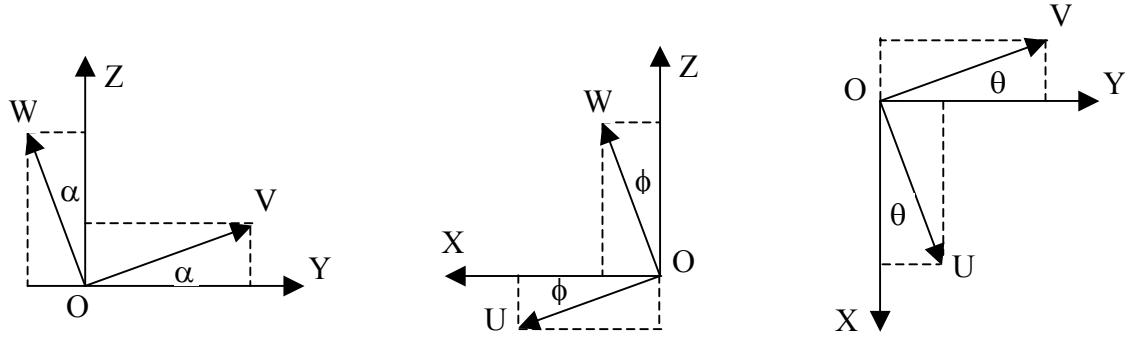


Figure 2.7: Illustration of an OUVW rotating co-ordinate system [22][1]

The rotation matrices can be represented as the following respectively:

$$R_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

$$R_{y,\phi} = \begin{bmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{bmatrix}$$

$$R_{z,\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.8: Illustration of a Rotation 3x3 Matrix for the 3 axes [x,y,z] [1][3]

Composite Rotation Matrix form the basic rotation matrices which can be multiplied together to represent a sequence of finite rotations about the principle axes of the $OXYZ$ coordinate system. Since matrix multiplications do not commute, the order or sequence of performing rotations is important. The rotation matrix representing a rotation of α angle

about the OX axis (*yaw*) followed by a rotation of θ angle about the OZ (*roll*) followed by a rotation of ϕ angle about OY (*pitch*) axis is given by the resultant rotation matrix as:

$$R = R_{y,\phi} R_{z,\theta} R_{x,\alpha} = \begin{bmatrix} C\phi C\theta & S\phi S\alpha - C\phi S\theta C\alpha & C\phi S\theta S\alpha + S\phi C\alpha \\ S\theta & C\theta C\alpha & -C\theta S\alpha \\ -S\phi C\theta & S\phi S\theta C\alpha + C\phi S\alpha & C\phi C\alpha - S\phi S\theta S\alpha \end{bmatrix}$$

The rotation matrix representing a rotation of ϕ angle about OY axis followed by a rotation of θ angle about the OZ axis followed by a rotation of α angle about the OX , the resultant rotation matrix representing these rotations is: [1][4]

$$R = R_{x,\alpha} R_{z,\theta} R_{y,\phi} = \begin{bmatrix} C\theta C\phi & -S\theta & C\theta S\phi \\ C\alpha S\theta C\phi + S\alpha S\phi & C\alpha C\theta & C\alpha S\theta S\phi - S\alpha C\theta \\ S\alpha S\theta C\phi - C\alpha S\phi & S\alpha C\theta & S\alpha S\theta S\phi + C\alpha C\phi \end{bmatrix}$$

2.1.2.1 Direct and Inverse Kinematics

Figure 2.9 is an illustration of direct and inverse kinematics.

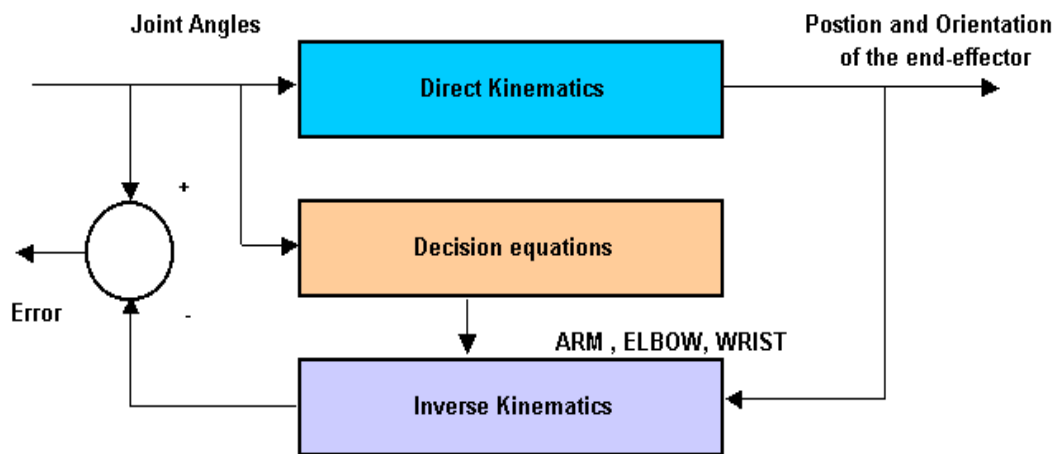


Figure 2.9: A simple diagram indicating the relationship between direct and inverse robot kinematics [1]

Since the links of a robot arm may rotate and / or translate with respect to a reference co-ordinate frame, the total spatial displacement of the end-effector is due to the angular rotations and linear translations of the links.

2.1.2.2 Links, Joints and their parameters

A mechanical manipulator consists of a sequence of rigid bodies, called links, connected by either revolute or prismatic joints [5].

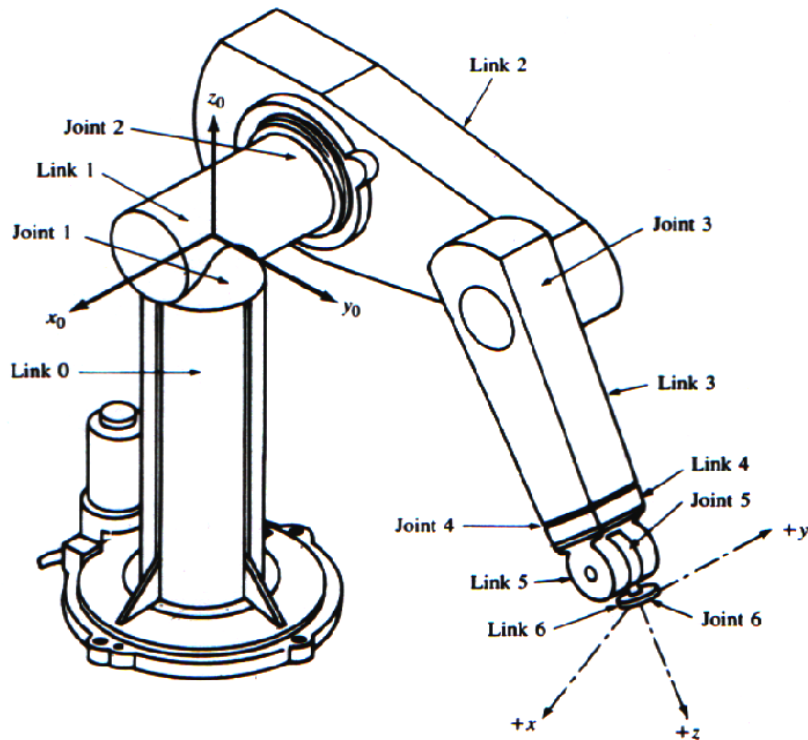


Figure 2.10: A PUMA robot arm illustrating joints and links [1]

Each joint-link pair constitutes one degree of freedom (DOF). For an N degree of freedom manipulator, there are N joint-link pairs with link 0 (not considered part of the robot) attached to a supporting base where an inertial co-ordinate frame is usually established for this dynamic system and the last link is attached with a tool. The joints

and links are numbered outwardly from the base. Thus, joint 1 is the point of connection between link 1 and the supporting base. A joint axis (for joint i) is established at the connection of two links as illustrated in Figure 2.10 and 2.11.

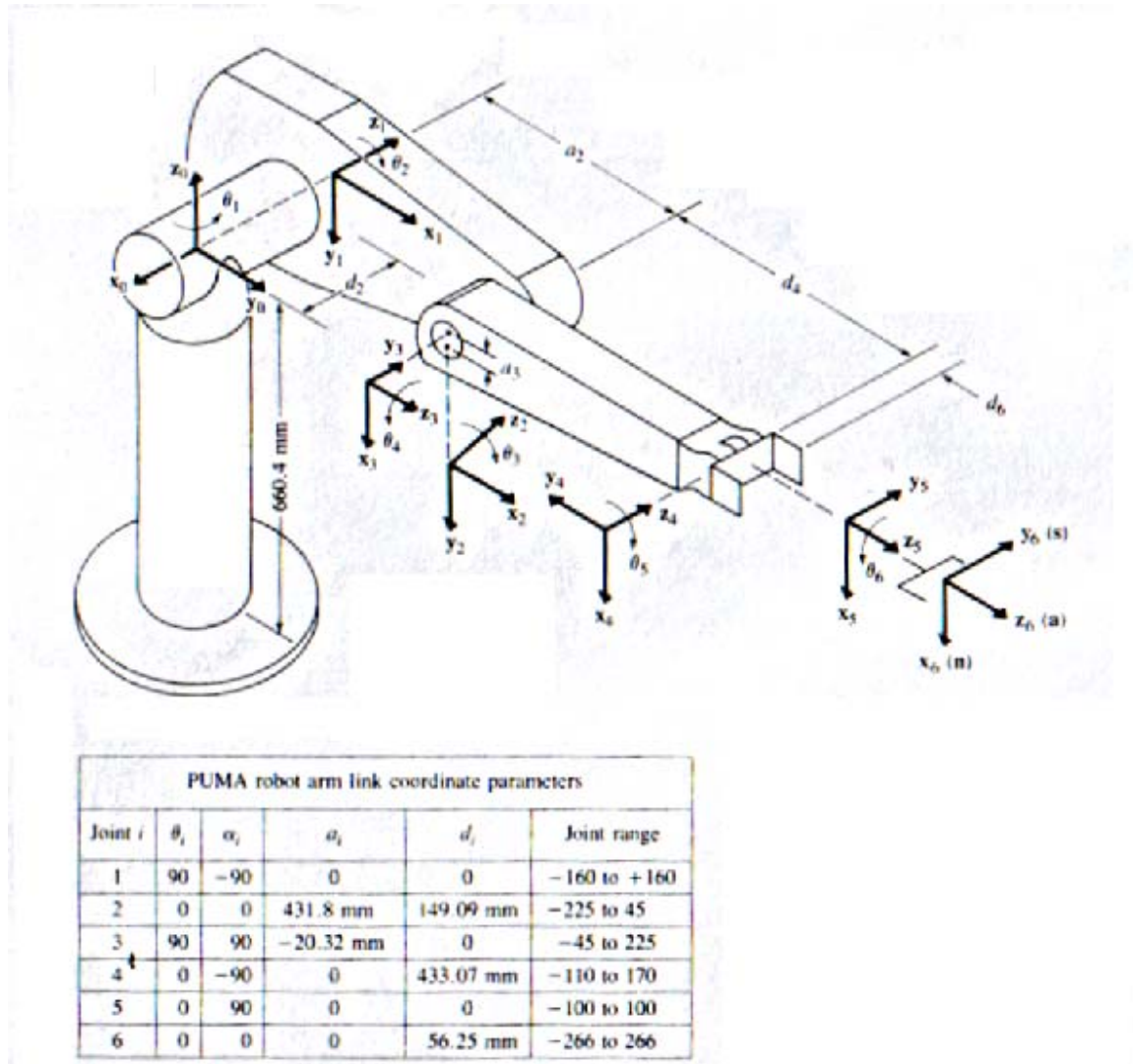


Figure 2.11: Link co-ordinate system and its parameters [1][2]

2.1.3 Robot Motion

Robot motion is sub-divided into the following co-ordinate frames as listed below with respect the co-ordinate chain as illustrated in Figure 2.12: [21][22].

- (i) **The World Co-ordinate System** – defines a reference to the floor, which is the starting point for the other co-ordinate systems.
- (ii) **The Base Co-ordinate System** – is attached to the base mounting surface of the robot.
- (iii) **The Tool Co-ordinate System** – specifies the tool's center point and orientation.
- (iv) **The User Co-ordinate System** – specifies the position of a fixture or work piece.
- (v) **The Object Co-ordinate System** – specifies how a work piece is positioned in a fixture or work piece manipulator.
- (vi) **Program displacement coordinate system** — is set up by robot instructions in RAPID program, and is related to object coordinate system.

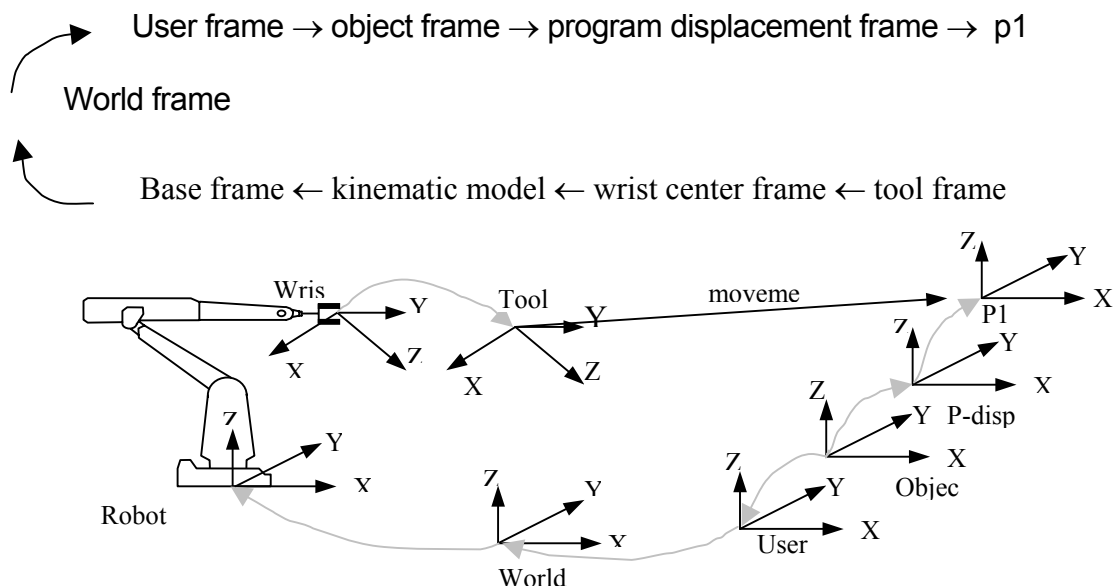


Figure 2.12: Co-ordinate frame chain [21][22]

2.2 Manipulator Trajectory Control

With the knowledge of kinematics and dynamics of a serial link manipulator, one would like to servo the manipulator's joint actuators to accomplish a desired task by controlling the manipulator to follow a desired path. Before moving the robot arm it is of interest to know whether there are any obstacles present in the path that the robot arm has to traverse (obstacle constraints) and whether the manipulator hand needs to traverse along a specified path (path constraints).

The space curve that the manipulator hand moves along from an initial location (position and orientation) to the final location is called the *robot path*. The trajectory planning interpolates and / or approximates the desired path by a class of polynomial functions and generates a sequence of time based "control set points" for the control of the manipulator from the initial location to the destination location.

Control Analysis – the movement of the robot arm is usually performed in two distinct control phases:

1. *Main motion control* is to move the arm from the initial position / orientation to the vicinity of the desired target position / orientation along a planned trajectory.
2. *Fine motion control* is when the end-effector of the arm dynamically interacts with the object using sensory feedback information from the sensors in order to complete the task.

The current industrial approach to robot arm control is – treat each joint of the robot as a simple joint servo mechanism.

The servo mechanism approach models the varying dynamics of a manipulator inadequately because it neglects the motion and configuration of the whole arm mechanism. Robot arm control requires the consideration of more efficient dynamic models, sophisticated control approaches, the use of dedicated computer architectures and parallel processing techniques [1].

2.2.1 Robot Programming Language

One major obstacle in using manipulators as general-purpose assembly machines is the lack of suitable and efficient communication between the user and the robotic system so that the user can direct the manipulator to accomplish a given task. There are several ways to communicate with a robot, such as: Discrete word recognition, teach and playback and a high-level programming language. The most general approach used in order to solve the human-robot communication problem is the use of high-level programming. Robots are commonly used in areas such as arc welding, spot welding and paint spraying.

Robot programming is substantially different from traditional programming. There are several considerations which must be handled by any programming language, such as -

- the objects to be manipulated by a robot are three-dimensional objects which have a variety of physical properties,
- robots operate in a spatially complex environment,
- the description and representation of three-dimensional objects in a computer are imprecise; and

- sensory information has to be monitored, manipulated and properly utilized.

Current approaches to programming can be categorized into two categories, namely -

1. robot-orientated programming; and
2. object-orientated or task-level programming.

In *robot-orientated programming* an assembly task is explicitly described as a sequence of robot motions. The robot is guided and controlled by the program throughout the entire task with each statement of the program roughly corresponding to one action of the robot.

Task-level programming describes an assembly task as a sequence of positional goals of the object rather than the motion of the robot needed to achieve these goals and hence no explicit robot motion is specified [25][2][1].

2.2.2 Characteristics of Robot-level languages

The most common approach taken in designing a robot-level language is to extend an existing high-level language to meet the requirements of robot programming. Its design philosophy is to provide a system environment where different robot programming interfaces may be built. It has a rich set of primitives for robot operations and allows the users to design high-level commands according to their particular needs. The following are needs identified for this project -

(i) Position Specifications

In robot assembly the robot and the parts are generally confined to a well-defined workspace. The parts are usually restricted by fixtures and feeders to minimize

positional uncertainties. Assembly from a set of randomly placed parts requires vision which is not yet common practice in industry.

The most common approach used to describe the orientation and the position of the objects in the workspace is by co-ordinate frames. They are usually represented as 4x4 homogeneous transformation matrices. A frame consists of a 3x3 submatrix (specifying the orientation) and a vector (specifying the position) which are defined with respect to some base frame.

(ii) Motion Specifications

The most common operation in robot assembly is the *pick and place operation*. It consists of moving the robot from an initial configuration to a grasping configuration, picking up an object and moving to final configuration. The motion is usually specified as a sequence of positional goals for the robot to attain. However, only specifying the initial and final configurations is not sufficient. Both constraints must be considered, such as obstacles in the present planned path [1].

(iii) Sensing and Flow of Control

The location and the dimension of the object in the workspace can be identified only to a certain degree of accuracy. For a robot to perform tasks in the presence of these uncertainties sensing must be performed. The sensory information gathered also acts as a feedback from the environment enabling the robot to examine and verify the state of the assembly [30].

Sensing in robot programming can be classified into three types :

1. *Position Sensing* – used to identify the current position of the robot, usually achieved by encoders that measure the joint angle and compute the corresponding hand position (end effector) in the current workspace.
2. *Force and Tactile Sensing* – used to detect the presence of objects in the workspace.
3. *Vision* – used to identify objects and provide a rough estimate of their position, orientation and profile.

(iv) Programming Support

A language without programming support (editor / debugger) is useless to the user. A sophisticated language must provide a programming environment that allows the user to support it. Complex robot programs are difficult to develop and can be difficult to debug. It should be realized by now that programming in a robot-oriented language is tedious and cumbersome.

2.2.3 Characteristics of Task-level languages

A completely different approach in robot programming is by *task-level programming*. An assembly task can best be described in terms of the objects being manipulated rather than by the robot motions.

A task-level programming system allows the user to describe the task in a high level language (task specification). A task planner will then consult a database (world models)

and transform the task specification into a robot-level program (robot program synthesis) that will accomplish the task.

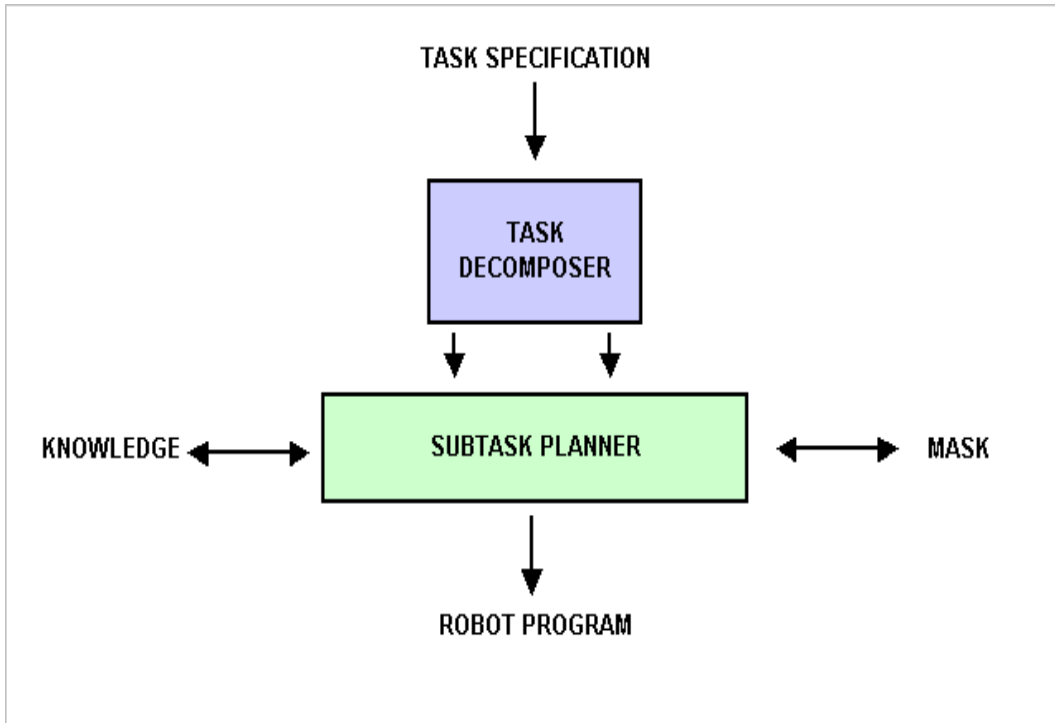


Figure 2.13: Task planner

Architecture for a robot task planner is displayed in Figure 2.13 The task specification is decomposed into a sequence of subtasks by the task decomposer and information is extracted, such as: Initial state, final state, grasping position, operand, specifications and attachment relations. The subtasks then pass through the subtask planner which generates the required robot program [29].

The concept of task planning is quite similar to the idea of automatic program generation in artificial intelligence. The user supplies the input-output requirements of a desired

program, and the program generator then generates a program that will produce the desired input-output behaviour [37].

Task-level programming, like automatic program generation, is in the research stage with many problems still unsolved. The problems encountered in task planning and some of the proposed solutions are discussed below.

(i) World Modeling

This modeling is required to describe the geometric and physical properties of the object (including the robot) and to represent the state of the assembly of the objects in the workspace.

Geometric and Physical Models. For the task planner to generate a robot program that performs a given task, it must possess information about the objects and the robot itself.

A geometric model provides the spatial information (dimension, volume, shape) of the objects in the workspace.

In the AUTOPASS system [38], objects are modeled by utilizing a modeling system called GDP (geometric design processor) [39], which uses a procedural representation to describe objects. Within this procedure, the shape of the object is defined by calls to other procedures representing other objects or set operations.

GDP provides a set of primitive objects (all of them are polyhedra) which can be cuboid, cylinder, wedge, cone, hemisphere, laminum and revolute. These primitives

are internally represented as a list of surfaces, edges and points, which are defined by the parameters in the corresponding procedure. For example,

```
CALL SOLID(CUBOID, "Block", xlen, ylen, zlen) ;
```

will invoke the procedure SOLID to define a rectangular box called Block with dimensions xlen, ylen, and zlen. More complicated objects can then be defined by calling other procedures and applying the MERGE subroutine to them.

In this research project, the same software coding approach is utilized as illustrated below.

```
Call Creat_Robot_robotargetVar("HOME", "PERS", "CBV", "1", x, y,z, q1, q2, q3, q4)
```

RAPID robot target variables can be created on the fly when the robot trajectory program is automatically generated. The call function will ensure that a variable named "HOME" declared as a PERS = persistence, with the following x,y,z coordinate position will be created in the master robot sub-routine RAPID program when requested. This function can generate a number of variables when requested.[22]

(ii) Task Specification

This is done with a high-level language. At the highest level one would like to have natural languages as the input, without having to give the assembly steps. An entire task like building a water pump could then be specified by the command "build

water pump”. However, this level of input is still quite far away. The current approach is to use an input language with a well-defined syntax and semantics, where the assembly sequence is given. An *assembly task* can be described as a sequence of states of the world model [1].

(iii) Robot Program Synthesis

The synthesis of a robot program from a task specification is one of the most important and most difficult phases of task planning.

The major steps in this phase are -

- grasping,
- planning,
- motion planning; and
- plan checking.

Before the task planner can perform the planning, it must first convert the symbolic task specification into a usable form. One approach is to obtain configuration constraints from the symbolic relationships. The RAPT Interpreter extracts the symbolic relationships and forms a set of matrix equations with the constraint parameters of the objects as unknowns. These equations are solved symbolically by using a set of rewrite rules to simplify them. The result obtained is a set of constraints on the configurations of each object that must be satisfied to perform the operation.

2.3 Communication Controls

The advent of the microprocessor created a new class of manufacturing devices – *the digital controller*. Digital devices are now not only commonplace in manufacturing but in many cases are essential to the manufacturing process. The management, maintenance and fully optimized use of these devices are greatly enhanced by communications between the control devices and supervisory computer systems. While in theory this communication should be trivial, it is in practice often so difficult as to be impossible. The evolution of modern industrial control has centered around the microprocessor.

It is the microprocessor that has created the need to communicate and integrate manufacturing in today's modern process plants. This is referred to as CIM.

The flexibility of the microprocessor is best utilized through changes in programming. Communicating those changes to the device is essential if the benefit is to be derived. Short of storing the program variations locally, the only viable alternative is to store them remotely, then communicate them to the device. This is also the only practical way to process and archive the data remotely. To obtain the benefits from the flexibility and the wealth of information generated by the microprocessor communication is essential [8].

2.3.1 Real-Time

The definition of real time is not precise, it is very situational. Within the banking industry applications that deal with ATM's are considered "real time". However, during the four to seven seconds it often takes for an ATM transaction, hundreds of rands of a

product may be mismanufactured or serious safety problems may arise in a typical process.

Yet in the 500 milliseconds that a DCS may take to measure a variable, calculate a response and execute a control action, a racing car can travel over ten times its own length. Real time is then relative to the environment. In batch and continuous processing operations, overall system response times are generally measured in the tens to hundreds of milliseconds or, at the worst, in seconds. Thus, the data communications networks within these systems must have performance characteristics that are consistent with this range.

Most industrial robot controllers have to be configured and programmed via a hand held teach pendant, making programming very tedious and time consuming [3]. By providing the system with an Ethernet communication link between a bridge PC and the robot controllers opens the system to true real-time control application [6]. This ensures that RAPID path programming can be automatically generated remotely and downloaded when robot path position is manipulated due to the sensory feedback devices, such as a vision system.

2.3.2 Local Area Networks (LAN's)

In an effort to address the growing complexity of data communications, networks were developed that allowed these devices to communicate with each other in a simpler fashion than with point-to-point technologies. These networks reflect either the shared usage of media (physical connections) or the software protocols used to communicate. Many factors must be carefully weighed when selecting a LAN.

A variety of systems are available, each of which are unique in operation, hardware and capabilities.

(i) Benefits of LAN's

LAN's are essentially transparent to users and can provide a variety of benefits, depending on the configuration and usage. Some of the benefits could include -

1. reduction / control of cabling cost,
2. user sharing of programs, data, printers and communication links,
3. access to multiple databases,
4. access to remote systems,
5. possibility of linking multivendor machines,
6. improved data integrity and security; and
7. improved staff communications.

(ii) LAN characteristics

Local area networking is a critical step in wiring either the office or the factory. By properly planning the network, one can create a system that links a group of different or incompatible computers, workstations, and accessories within one office, an entire building, or group of buildings.

In its simplest form, a local area network utilizes standard cabling, which acts as an electronic highway for transporting data and other information to and from different “workstations” in the same office area.

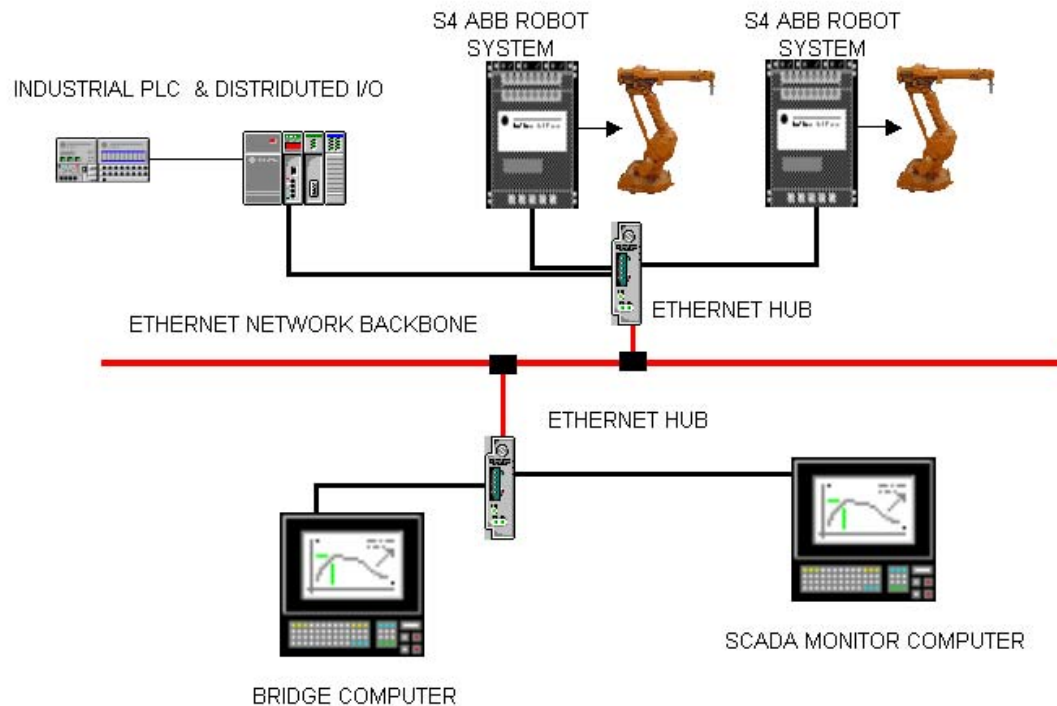


Figure 2.14: Industrial LAN Network Architecture

2.3.2.1 Communication Models

(i) Proprietary Model

DEC had their own model for communications, the DNA. DNA (Figure 2.15) is rich in peer-to-peer services. It first lacked terminal connectivity. DEC remedied that with the addition of LAT protocols, which were optimized for the terminal environment in a network-based system.

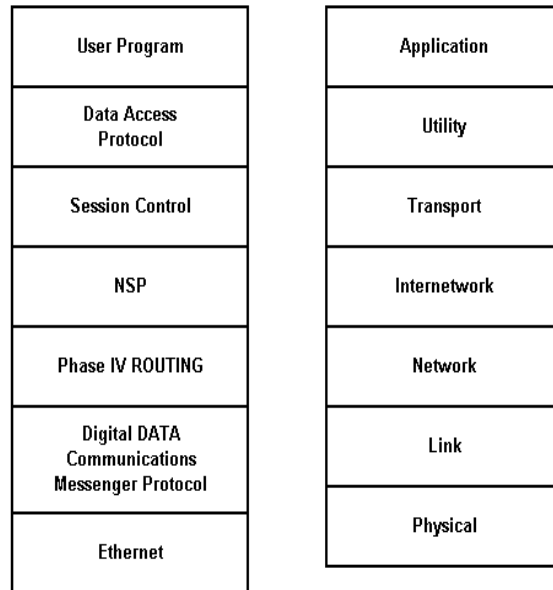


Figure 2.15: DNA by DEC and Internet model

(ii) **ISO Model** – the seven functional communication layers are: [32]

1. **Application Layer** : This layer is concerned with the information in the message and how well it serves the user. This is where application programs call upon the communication services. Typical protocols at this layer include FTAM, and VT.
2. **Presentation Layer** : This layer is used to prepare the information for the application.
3. **Session Layer** : This layer establishes the logical communications link between units and gradually feeds or buffers the information to the devices or the program that performs the Presentation function. The Session layer

also provides the critical identification and authentication functions. It recognizes users and acknowledges both their arrival and departure.

4. **Transport Layer** : This layer functions to provide a common interface to the communications network. It translates whatever unique requirements the other higher layers might have into something the network can understand. It detects and corrects errors in transmission and provides for the expedited delivery of priority messages. It checks the data, puts it into proper order if necessary, and usually sends an acknowledgement back to the originating Transport layer.
5. **Network Layer** : This layer sets up a logical transmission path through a switched or dedicated network. In local networks this path may be only theoretical, since the individual units are almost always electrically connected into the circuit and the paths are defined by the network topology.
6. **Data Link Layer** : This layer does the accounting and traffic control chores needed to transfer information on an electrical link. It forms the information to be moved into strings of characters, or into blocks of bits (characters). The Data Link layer puts every piece of information into the right place and checks it out before releasing it. Similarly, incoming information is broken down and properly routed within the receiving device.

7. *Physical Layer* : This layer describes the electrical and physical connection between the communicating links.

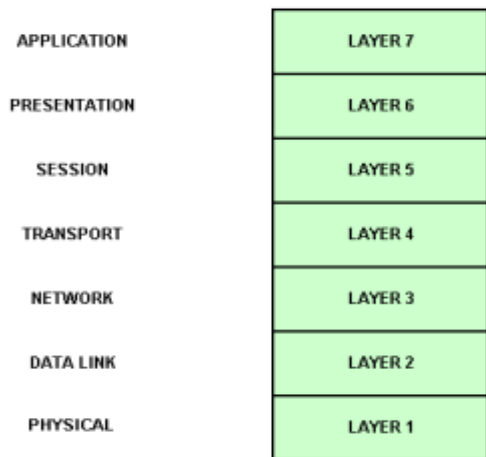


Figure 2.16: OSI seven-layer model

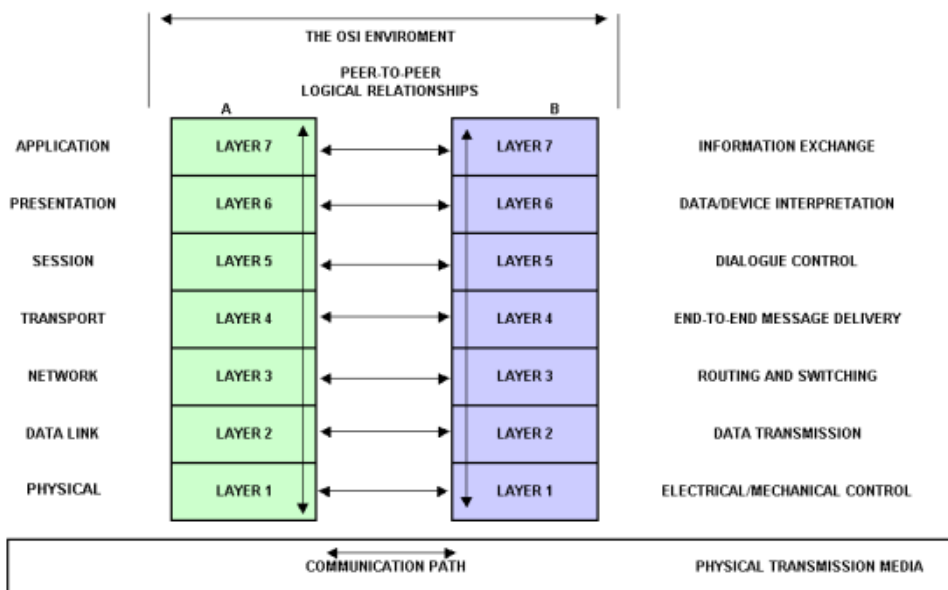


Figure 2.17: Data flow

There are *ISO Standards* for all seven ISO layers. At the lower two layers one of the common ISO/IEEE standards that are in place and use the ISO model as a reference is IEEE 802.3. This refers to CSMA/CD networks. ETHERNET is a common example of CSMA/CD protocol [35].

(iii) Internet Protocols

It appears no other protocol is having as great an impact on real-time networks today as TCP/IP (Transmission Control Protocol / Internet Protocol) – a subset of the collection of protocols employed by the Internet. The Internet evolved from the old US DOD ARPANET (Advanced Research Projects Agency Network). This nationwide network was designed to support the interconnection of local area networks at research institutions working on government funded projects. The initial key protocols had to do with end-to-end message integrity and routing information over the wide area network (WAN). This became the TCP/IP portion of the network. While TCP/IP play a significant part of the Internet, there are other protocols, for example file transfer, mail, and virtual terminal services [35].

(iv) Mid Level Protocols

IP – The services provided by the IP protocol are basically either related to addressing and routing or associated with the segmentation of packets if maximum packet size varies between intermediate segments and the two end networks.

| VERSION | IHL | TYPE OF SERVICE | TOTAL LENGTH | ID | FLAGS | FRAGMENT OFFSET | TIME TO LIVE | PROTOCOL | HEADER CHECKSUM | SOURCE ADDRESS | DESTINATION ADDRESS | OPTIONS | PADDING | DATA |
|---------|------|-----------------|--------------|--------|-------|-----------------|--------------|----------|-----------------|----------------|---------------------|---------|---------|------|
| 4 | 4 | 1 | 2 | 2 | 8 | 13 | 1 | 1 | 2 | 4 | 4 | 3 | | |
| BITS | BITS | OCTET | OCTETS | OCTETS | BITS | BITS | OCTET | OCTET | OCTETS | OCTETS | OCTETS | OCTETS | | |

Figure 2.18: IP Frame

IP Addresses – IP packets use a 4 byte address field for both the source and destination addresses. Within this 32 bit field is network and station address information.

IP Routing – IP routing is accomplished as a shared function. The source station determines if the destination is part of the local network. If it is, the packet is sent directly to the destination. If the destination is not on the local network, the IP layer uses a stored table of routing information and destination addresses to determine which gateway device to send the packet to. The gateway is then responsible for the further transmission of that packet.

TCP – Transport Control Protocol – Because the underlying IP layer does not provide reliable service, being datagram-based, another service is required to ensure end-to-end integrity. This is the function of the transmission control protocol.

Fig 2.19 shows the contents of the TCP packet.

| SOURCE PORT | DESTINATION PORT | SEQUENCE NUMBER | ACK NUMBER | DATA OFFSET | UNUSED | CONTROL | WINDOW | CHECK SUM | URGENT POINTER | OPTIONS | PADDING | DATA |
|-------------|------------------|-----------------|------------|-------------|--------|---------|--------|-----------|----------------|---------|---------|------|
| 2 | 2 | 4 | 4 | 4 | 6 | 6 | 2 | 2 | 2 | 3 | | |
| OCTETS | OCTETS | OCTETS | OCTETS | BITS | BITS | BITS | OCTETS | OCTETS | OCTETS | OCTETS | | |

Figure 2.19: TCP Frame

The use of *ports*, as they are called in a TCP environment, facilitates multiple sessions. The ports serve as a means to establish a virtual connection at this level, promoting guaranteed delivery. The sequence number is used to ensure proper ordering as the packets are received. The WINDOW parameter is used for flow control. If interfacing through TCP, it is advisable to use this full TCP packet as opposed to the lower overhead user datagram protocol (UDP) packets. The UDP packet as shown in Figure 2.20 does not support a connection-orientated service or error recovery and, as such, is of little value as an interfacing protocol for real-time networks. If the information is important enough to burden the real-time system with its handling, it should certainly warrant the additional integrity that the full TCP packet affords [8].

| | | | | |
|----------------|---------------------|--------|----------|------|
| SOURCE PORT | DESTINATION PORT | LENGTH | CHECKSUM | DATA |
| 2 | 2 | 2 | 2 | |
| OCTETS | OCTETS | OCTETS | OCTETS | |

Figure 2.20: UDP Frame

2.3.2.2 Real-Time Issues of TCP/IP

TCP/IP has become a readily accepted networking protocol for general-purpose plantwide networks, particularly where Unix systems are present.

As shown in Figure 2.21, a Client Server approach that uses TCP/IP is popular. Generally, these systems have dedicated serial interfaces to the process control systems from which they extract data. However, as more TCP/IP-based systems are implemented, it is reasonable to expect that TCP/IP network interfaces will become

available. This means, as with so many other developments in networking, that additional communication loads will be placed on those real-time networks as they move data to higher level systems. An understanding of TCP/IP services undoubtedly will be a prerequisite for internetworking in the very near future.

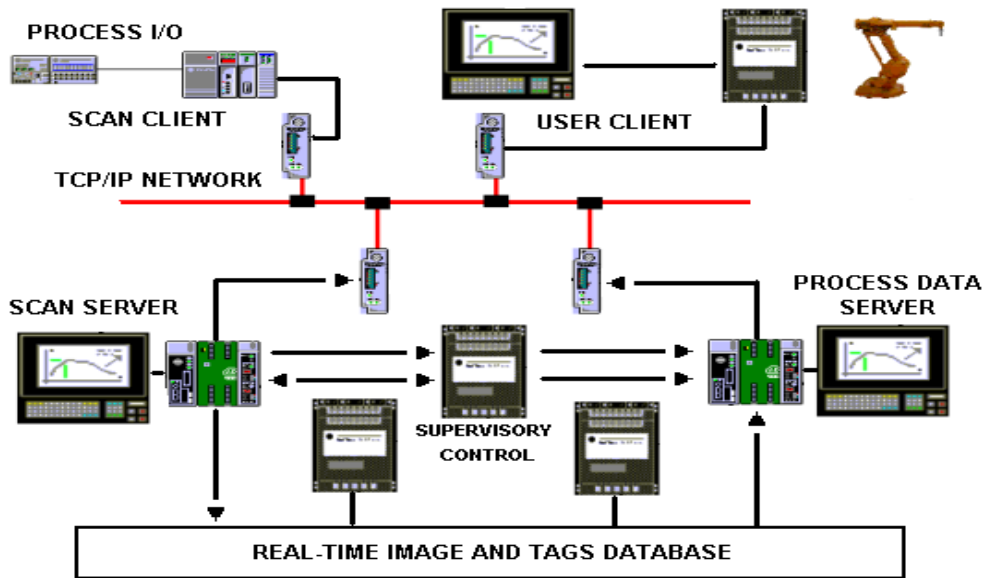


Figure 2.21: Client-server approach using TCP/IP

2.4 Robot Sensing

The use of an external sensing mechanism allows a robot to interact with its environment in a flexible manner. This is in contrast to preprogrammed operations in which a robot is taught to perform repetitive tasks via a set of preprogrammed functions. Although the latter is by far the most predominant form of operation of current industrial robots, the use of sensing technology to endow machines with a greater degree of intelligence in dealing with the environment is indeed an active topic of research and development in the robotic field.

Robot sensing is divided into two functional areas, internal and external states:

1. *Internal State Sensors* – deal with the detection of variables such as arm joint positions, which are used for robot control.
2. *External State Sensors* – deal with the detection of variables, such as range, proximity and touch. Although proximity, touch and force sensing play a significant role in the improvement of robot performance, vision is recognized as the most powerful of robot sensory capabilities [5].

2.4.1 Machine Vision

Machine vision (other names include computer vision and artificial vision) is an important sensor technology with potential applications in many industrial operations. Many of the current applications of machine vision are in inspection, however it is anticipated that vision technology will play an increasingly significant role in the future of robotics.

Vision systems designed to be utilized with robot or manufacturing systems must meet two important criteria which currently limit the influx of vision systems to the manufacturing community. The first of these criteria is the need for a relatively low-cost vision system. The second criterion is the need for relatively rapid response time needed for robot or manufacturing applications, typically a fraction of a second.

Nevertheless, there has been a significant influx of vision systems into the manufacturing world. The systems are used to perform tasks, which include selecting parts that are randomly orientated from a bin or conveyer, parts identification and limited inspection.

These capabilities are selectively used in traditional applications to reduce the cost of part and tool fixturing, to allow the robot program to test for and adapt to limited variations in the environment.

Advances in vision technology for robotics are expected to broaden the capabilities to allow for vision-based guidance of the robot arm, complex inspection for close dimensional tolerances, improved recognition and part location capabilities. These will result from the constantly reducing cost of computational capability, increased speed and new and better algorithms currently being developed.

The field of computer vision was one of the fastest growing commercial areas in the latter part of the twentieth century. Computer vision is a complex and multidisciplinary field and is still in its early stages of development.

Advances in vision technology and related disciplines are expected within the next decade, which will permit applications not only in manufacturing, but also in photo interpretation, robotic operations in hazardous environments, autonomous navigation, cartography and medical image analysis.

2.4.1.1 Background

Vision may be defined as the process of extracting, characterizing, and interpreting information from images of a two dimensional world. This process, also commonly referred to as **machine or computer vision**, may be subdivided into six principal areas, the operation of the vision system consists of six functions as illustrated in Figure 2.22: [1][2][3][5]

1. *Sensing* – Is the process that yields a visual image.
2. *Preprocessing* – Deals with techniques such as noise reduction and enhancement of details.
3. *Segmentation* – Is the process that partitions an image into objects of interest.
4. *Description* – Deals with the computation of features (e.g. size and shape) suitable differentiating one type of object from another.
5. *Recognition* – Is the process that identifies these objects (e.g. wrench, bolt, engine block, Object profiles).
6. *Interpretation* – Assigns meaning to an ensemble of recognized objects.

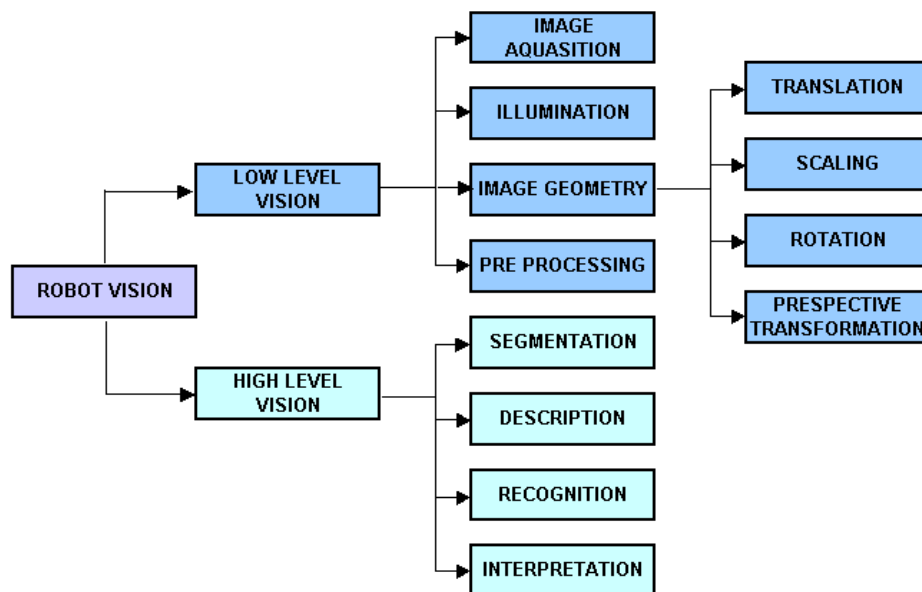


Figure 2.22: Robot Vision basic structure

Machine Vision is concerned with the sensing of vision data and its interpretation by a computer. The typical vision system consists of the camera and digitizing hardware, a digital computer, and hardware and software necessary to interface them. The operation of the vision system consists of three functions as illustrated in Figure 2.23:

1. Sensing and digitizing image data
2. Image processing and analysis
3. Application

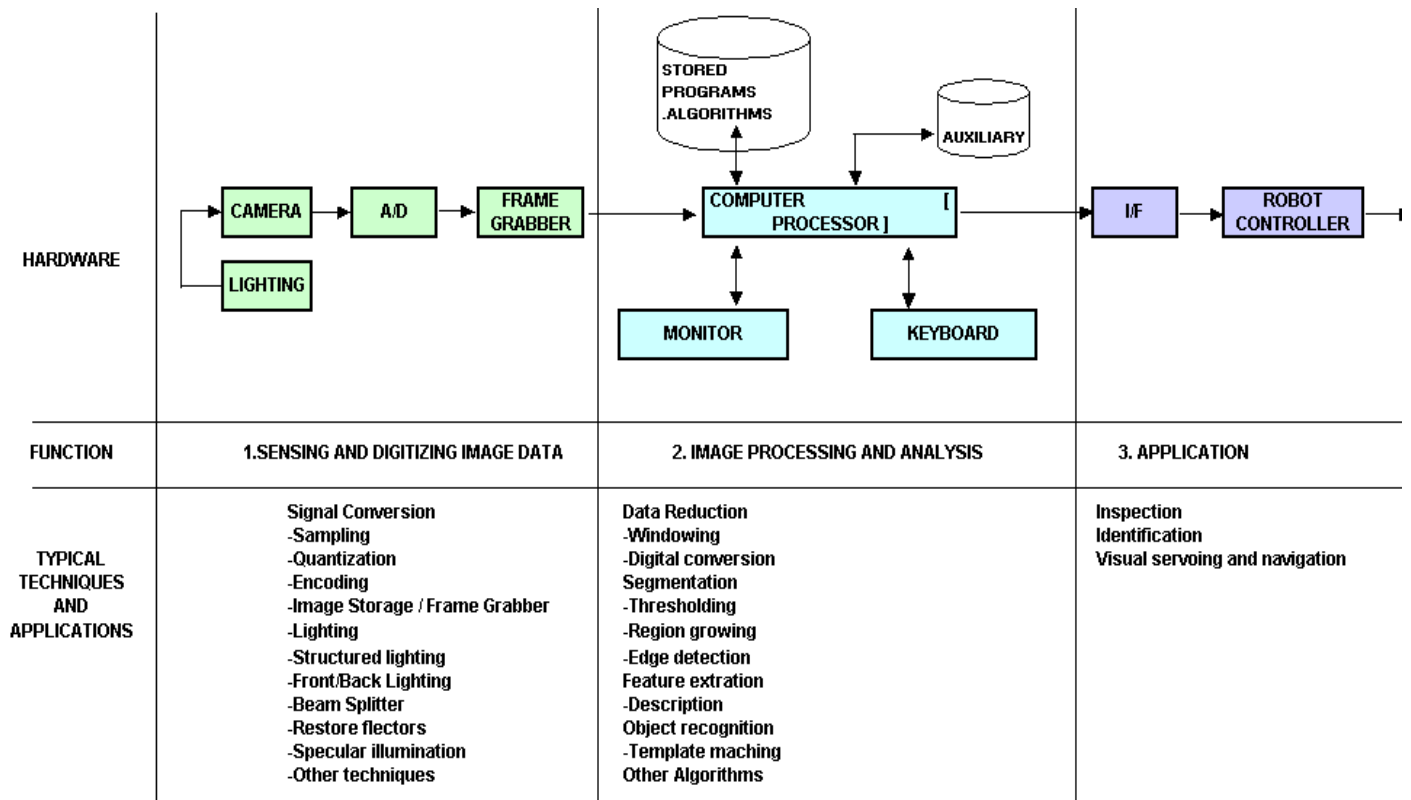


Figure 2.23: This diagram simplifies the relationship between the three functions of machine vision [2]

The sensing and digitizing functions involve the input of vision data by means of a camera focused on the scene of interest. Special lighting techniques are frequently used to obtain an image of sufficient contrast for later processing [12][13]. The image viewed by the camera is typically digitized and stored in computer memory.

The digital image is called a *frame* of vision data and is frequently captured by a hardware device called a grabber [24][32].

The frames consist of a matrix of data representing projections of the scene sensed by the camera. The elements of the matrix are called picture elements, or *pixels*. The number of pixels are determined by a sampling process performed on each image frame. A single pixel is a projection of a small portion of the scene which reduces that portion to a single value. The value is the measure of the light intensity for that element of the scene. Each pixel intensity is converted into a digital value.

The digitized image matrix for each frame is stored and then subjected to image processing and analysis functions for data reduction and interpretation of the image. These steps are required in order to permit real-time application of vision analysis required in robotic applications.

Typically an image frame will be thresholded to produce a binary image, and then various feature measurements will further reduce the data representation of the image. This data reduction can change the representation of a frame from several hundred thousand bytes of raw image data to several hundred bytes of feature value data. The resultant feature data can be analyzed in the available time for action by the robot system.

Various techniques to compute the feature values can be programmed into the computer to obtain feature descriptors of the image which are matched against previously computed values stored in the computer. These descriptors include shape and size characteristics that can be readily calculated from the thresholded image matrix.

To accomplish image processing and analysis, the vision system must be trained frequently. In training, information is obtained on prototype objects and stored as computer models.

The information gathered during training consists of features such as the area of the object, its perimeter length, major and minor diameters and similar features. During subsequent operation of the system, feature values computed on unknown objects viewed by the camera are compared with the computer models to determine if match has occurred.

The final function of a machine vision system is the applications function. The current applications of machine vision in robotics include inspection, part identification, location and orientation. Research is ongoing in advanced applications of machine vision for use in complex inspection, guidance and navigation.

Many two-dimensional vision systems can operate on a binary image which is the result of a simple thresholding technique. This is based on an assumed high contrast between the object(s) and the background. Image contrast can be manipulated by using a controlled lighting system.

Another way of classifying vision systems is according to the number of gray levels used to characterize the image. In a binary image the gray level values are divided into either of two categories, black or white. Other systems permit the classification of each pixel's gray level into various levels, the range of which is called a gray scale.

As is true in humans, vision capabilities endow a robot with a sophisticated sensing mechanism that allows the machine to respond to its environment in an “intelligent” and flexible manner [2][9].

2.4.1.2 Vision Preprocessing

Vision processing has three levels -

1. *low-level vision [sensing and preprocessing]* - use algorithms to compensate noise reduction and then a primitive image can be extracted,
2. *medium-level [subdivision-segmentation, description, and recognition of individual objects]* – refers to those processes that extract, categorize, and label components in an image resulting from low-level vision; and
3. *high-level vision [interpretation]* – refers to processes that attempt to emulate cognition.

Plenty of image preprocessing techniques are available in the field of robot vision. The method used for image preprocessing range from spatial-domain and frequency-domain. Only a subset of them is suited for real-time image processing if the processing speed plays a predominant role in this process.

Convolution technique is one of the spatial-domain techniques used most frequently (also referred to as templates, windows, or filters) [32]. The desired image $f(x, y)$ can be obtained by convoluting the original image $i(x, y)$ with a convolution mask $h(x, y)$.

$$f(x, y) = h(x, y) * i(x, y)$$

In robot vision system, the convolution masks are usually a 3×3 , 5×5 or 7×7 matrices. For computational purposes the 3×3 matrix is widely utilized in real-time systems where system speed is required. A typical convolution mask is given as

$$H_{3 \times 3} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

The implementation of $H_{3 \times 3}$ convolution can be defined as

$$f(x, y) = \sum_{j=1}^3 \sum_{k=1}^3 i(x + j - 2, y + k - 2) \cdot h(j, k)$$

The image preprocessing techniques, which are also employed in this project, are the following:

➤ *Smoothing*

Smoothing operations are used for reducing noise and other spurious effects that may be present in an image as a result of sampling, quantization, transmission, or disturbances in the environment during image acquisition. The following convolution masks are used:

➤ *High Pass Filtering*

High pass filtering is utilized to sharpen images that are out of focus or fuzzy. Its convolution mask is

$$H_{High-pass} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

➤ *Median Filtering*

Median filtering ranks the current set of nine pixel intensities in order of magnitude and places the median intensity value into the destination image at the central point. The whole image is processed in turn by sliding the window over the entire image.

➤ *Low Pass Filtering*

Low pass filtering is exploited to smooth out a sharp image. Its convolution mask is given as:

$$H_{Low-pass} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

➤ *Noise Cleaning*

Noise cleaning is employed to remove random noise spikes on the captured image. Its convolution mask is given as:

$$H_{Noise-cleaning} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

➤ *Averaging*

Averaging can be used to remove random noise spikes and clean edge features in the image. Its convolution mask is given as:

$$H_{Averaging} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

➤ *Thresholding*

Digital image thresholding is a crucial process in robot vision system, which is used to manipulate the captured image. To separate and extract the object from the background in terms of an image array $f(x, y)$, a threshold of T is normally utilized. The thresholding technique is not limited to a fixed value T . A thresholded image can be acquired by

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{otherwise} \end{cases}$$

In the case of dark objects on a light background, thresholding takes the selected grayscale value T and compares each pixel intensity in the image. If the intensity at pixel $f(x, y) < T$ that pixel is replaced by a logic 0 value. If the intensity $f(x, y) > T$ that pixel is replaced by a logic 1 value. A typical image intensity histogram with two peaks is shown in Figure 2.24.

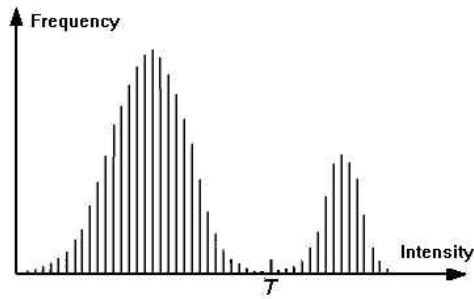


Figure 2.24: Typical two peak intensity histogram

The manipulated image after thresholding is illustrated in Figure 2.25.

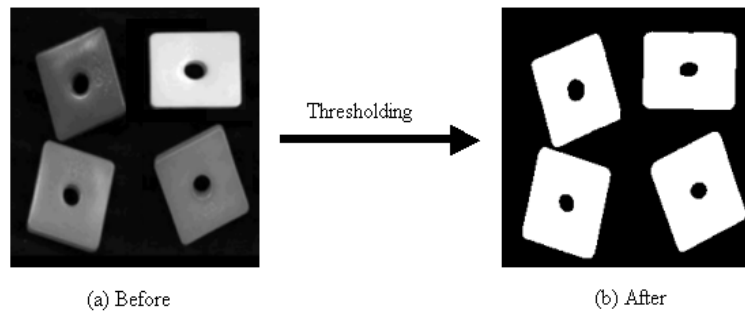


Figure 2.25: Thresholding to gray-level Image

In general, thresholding falls into two categories, which are manual thresholding and adaptive thresholding. Adaptive thresholding takes a histogram of all the pixel intensities in the images, detects the pixel intensity most frequent in the image and follows the histogram curve down to identify the minimum. An adaptive thresholding is capable of figuring out an optimal thresholding value [5].

➤ *Contour Detection*

Contour detection plays a central role in robot vision. Using the information from the contours means a considerable reduction in the volume of data to be processed in image analysis. In addition, using the contours obtained from the image is their relative stability under fluctuations in the lighting of the scene. The standard approaches to contour detection are implicitly based on a very simple model in which the image is regarded as ideally composed of essentially constant region separated by step edges. The classical approach [3] to contour detection makes use of digital (finite-difference) versions of standard isotropic derivative operators, such as the gradient or *Laplacian*.

The first derivative of a contour model is zero in all region of constant intensity. The second derivative is zero in all locations, except at the onset and termination of an intensity transition.

➤ *Laplacian Edge Detection*

The *Laplacian* is a scalar second derivative operator for functions of two dimensions, given by:

$$\nabla^2 f(x, y) = \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$$

The digital *Laplacian* at point (x, y) can be defined as:

$$L[f(x, y)] = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y)$$

2.4.1.3 Industrial Machine Vision

Researchers in the field of Industrial Machine Vision (IMV) concentrate their efforts on problems appropriate to the industrial environment. In such an environment one may be able to control the background, the lighting, the camera position, or other parameters. Such control may allow the use of techniques that would be inappropriate to a general-purpose vision system.

2.4.1.4 Fundamentals – The formation of a Digital Image

The imaging literature is filled with a variety of imaging devices, including dissectors, flying spot scanners, videocons, orthicons, plumbicons, CCD's (charge-couples devices), and others [40][41]. These devices differ both in the ways in which they form images and in the properties of the images so formed. However, all the devices convert light energy to voltage in similar ways.

2.4.1.5 Vision Hardware Components

Vision systems are occasionally supplied by the robot manufacturer and integrated with the controller, but usually are separate, with an interface to the robot controller as illustrated in Figure 2.26[14]

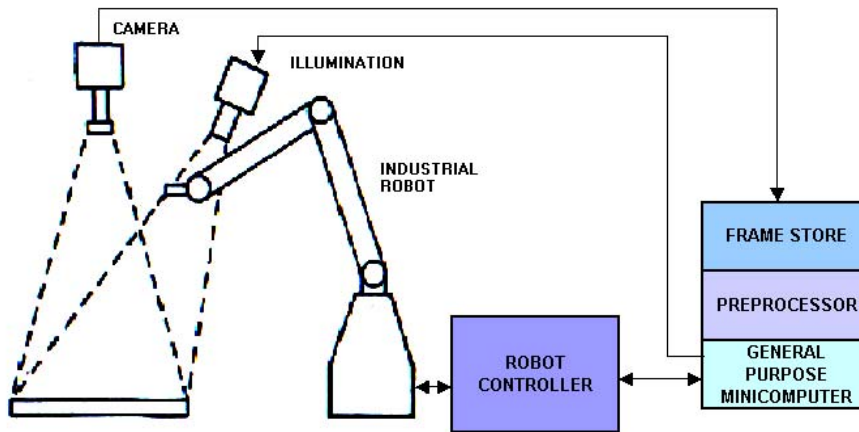


Figure 2.26: The components of a robot vision system

2.4.1.6 Image Acquisition

Visual information is converted to electrical signals by visual sensors. When sampled spatially and quantized in amplitude, these signals yield a *digital image*.

The following is of importance with regard to digital images:

1. The principal imaging techniques used for robotic vision
2. The effects of sampling on spatial resolution
3. The effects of amplitude quantization on intensity resolution

The principal devices used for robotic vision are television cameras, consisting either of a tube or a solid-state imaging sensor, and associated electronics.

As far as a color CCD camera is concerned, the captured image, which is made of a pixel array $I_k(x, y)$ (where $k = 1, 2, 3; x = 1, 2, \dots, m; y = 1, 2, \dots, n$)

where size is $m \times n$, contains the colour information and intensity of three colour channels. This colour pixel array can be represented as

$$I_k(x, y) = \begin{bmatrix} I(1,1) & I(1,2) & \dots & I(1,n) \\ I(2,1) & I(2,2) & \dots & I(2,n) \\ \dots & \dots & \dots & \dots \\ I(m,1) & I(m,2) & \dots & I(m,n) \end{bmatrix} \quad (k = 1, 2, 3)$$

The representation of pixel varies from the purposes of image processing. RGB (red, green, and blue) method, HSB (hue, saturation, and brightness) method, and CMYK (cyan, magenta, yellow, and black) method are commonly used. In the field of machine vision, RGB method is generally employed. In this case, each pixel is represented by three channels which denote red, green, and blue intensity respectively[32].

2.4.1.7 Applications of Vision include:

1. Detecting object presence or type
2. Determining object location and orientation before grasping
3. Feedback during grasping
4. Feedback for path control in welding and other continuous processes
5. Feedback for fitting a part during assembly
6. Reading identity codes
7. Object counting

8. Inspection, e.g. of printed circuit boards to detect incorrectly inserted components

2.5 Conclusion

A large aspect of industrial robotics has been discussed as a whole, such as how robots are categorized according to their arm movement configuration, how they are utilized in automation applications, robot manipulator axis structure, robot kinematics and dynamics for both direct and inverse kinematics, the parameters of robot arm links and joints as defined and a 4×4 homogeneous transformation matrix is introduced to describe the location of the link with respect to the fixed co-ordinate frame. Another area of discussion is robot manipulator trajectory path planning, robot programming languages and methods, robot sensing and flow control methods which will play an important role in later chapters.

In order to provide an industrial robot with remote flexibility, different communication protocols were discussed, focusing on the real-time issues as well as the Ethernet communication protocols, and protocol layers, which will ensure that the real-time system issues will be overcome.

Robot movement flexibility can be enhanced, by providing the robot with a sensing device, such as a vision system.

The fundamental concepts covered in this chapter will be used extensively in the chapters to follow, for deriving the equations of motion of an industrial ABB robot manipulator that describes the dynamic behaviour of a robot arm.

CHAPTER 3

SYSTEM SETUP : HARDWARE AND SOFTWARE ARCHITECTURE

This chapter provides a brief overview of the system architecture in terms of hardware and software components required and developed to achieve the integrated vision-based trajectory control system for an industrial ABB Robot. The system architecture provides the system platform overview of how the components were implemented in this research project.

3.1 Introduction

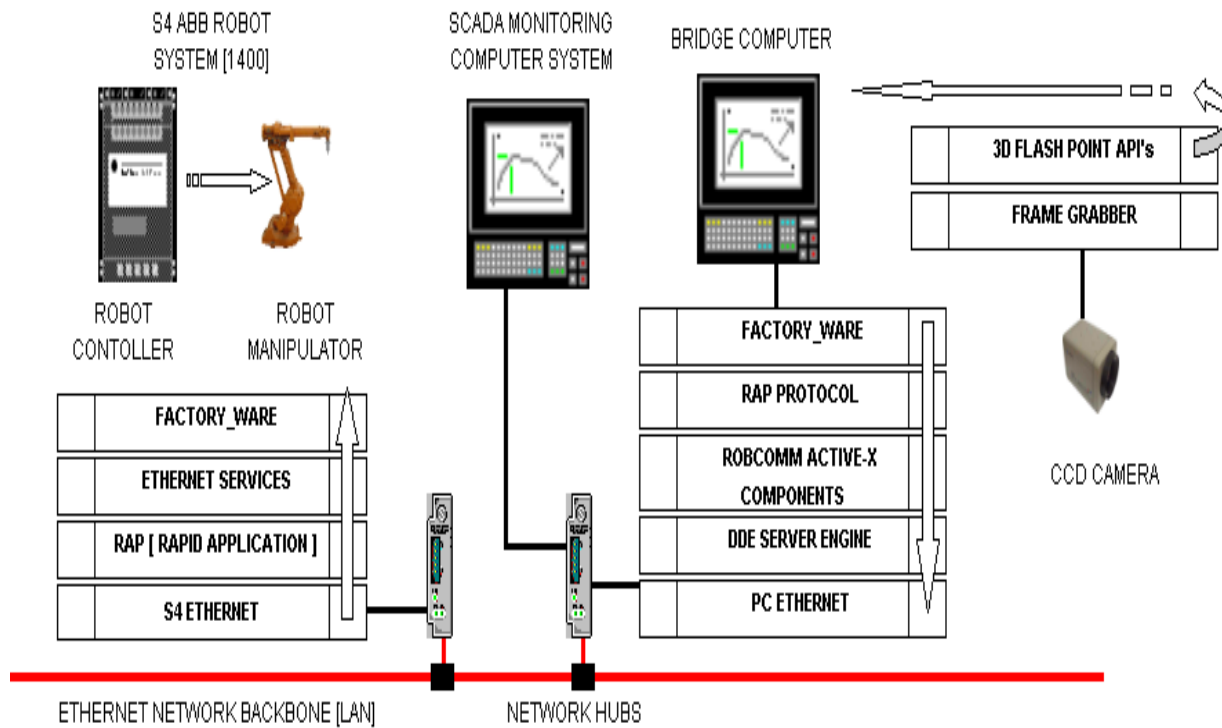


Figure 3.1: Communication Interfaces and Software development components for hardware sub-systems

In this research project to perform profile recognition and integrated robot control for industrial applications the following principle modules, as illustrated in Figure 3.1, were established:

- Robot Vision, which consists of a CCD camera [23] and an AGP bus frame grabber [24]. The camera is fixed to the robot arm and captures an image of the object. It is interfaced to the frame grabber hardware, which provides the live images, this is controlled through 2D Flash Point API's, the components are utilized to ensure that the image processing can be fulfilled.
- Bridge PC-Based control system, which provides the system with an on-line robot trajectory motion control mechanism. The PC-Based control is equipped with FactoryWare Software, which provides the system with powerful communication components that ensure a stable Ethernet communication platform such as:
 - RobComm - equipped with a software toolkit for simplified communication control between PC control system and S4 robot controllers via Ethernet.
 - Rapid Application Protocol (RAP) - handles the synchronization services between the Bridge computer and the S4 robot controller. These services are provided by the FactoryWare interface options.
 - RobComm consists of ActiveX Components that are common to all applications, thereby enabling rapid application development.

- RobComm Server provides a status monitoring display, which provides current status of all the defined aliases and a tuning interface to adjust how RobComm communicates with each alias.
- DDE Sever Engine provides a software building block that takes care of the communication with the robot and presents the robot data in a standard DDE format. Examples of applications that handle DDE communication are Microsoft “Excel”, which can be utilized for system visualization.
- An ABB industrial robot IRB-1400 and controller, which can be accessed through S4 Ethernet or serial communication. S4 robot Controller has been equipped with a standard Ethernet hardware, enabling the system to communicate with other remote Ethernet devices, which utilized standard TCP/IP Ethernet protocols. This hardware enables the Ethernet integration between the Bridge Computer and the S4 robot controller. In order to initiate communication the following software components need to be established:
 - FactoryWare Services install all communication protocols available for Ethernet communication.
 - Ethernet Services, ensure that all relevant components become available in the S4 controller configuration environment, which will in turn initiate the Ethernet hardware in the robot controller hardware rank.
 - Rapid Application Protocols, handle the synchronization services between the S4 robot controller and the Bridge PC.

The aim of this PC-based experimental setup is *to provide a platform for the manufacturing environment where continuous change in product, orientation and environment has a major effect on system setup. This will reduce the loss of production.*

The robot with vision sensory feedback is used to manipulate the robot trajectory path when there is a change in object profile and orientation. The robot is given this ability by means of a PC-Based Control system with vision and path manipulation software. A CCD camera provides a video image to a video frame grabber, which captures a 2D image of the environment of the robot. The Bridge PC software will process the image data and generate an image map which contains the object profile co-ordinates. The co-ordinates are then used to generate the RAPID Robot trajectory path. The RAPID program is downloaded to the robot controller via an Ethernet serial communication link (TCP/IP Protocol). All robot control command events are managed via the Ethernet link. The Ethernet communication link is set up for peer-to-peer communication for this project setup.

The Bridge PC is set up as the backbone and provides the link between the camera and robot controller. It has been set up with a host name “S4” and the Ethernet IP address 100.100.100.1. The RobComm Server manages the Ethernet protocol between the Bridge PC and robot controller. The second Ethernet card manages the communication with the LAN. This provides the additional system flexibility.

When the system software starts up so does the RobComm Server and a dedicated communication link with the robot controller is established. This allows the ActiveX component to access all the relevant components needed for the research project.

Before the installation of the relevant software the ABB IRB 1400 Industrial Robot at PE Technikon, did not have the capability of remote access from a remote server via an Ethernet link. The Ethernet hardware and all relevant Ethernet software protocols were installed. This enabled the robot with remote communication access.

The Bridge PC was equipped with 2D-Video Frame Grabber hardware, which provides the platform for image processing for the robotic environment.

3.2 Implementation Aspects for the Communications between hardware sub-systems

In this project the three major hardware sub-systems, namely: Industrial Robot Controller and the vision sensory system were integrated with a PC-Based control system to facilitate real-time response capability.

Figure 3.1 illustrates the hardware components and the software interface modules that was developed and integrated. The following sub-sections will give a brief outline on the hardware components which is illustrated Figures 3.2a-c, and from an implementation aspect of the communication modules and software development.

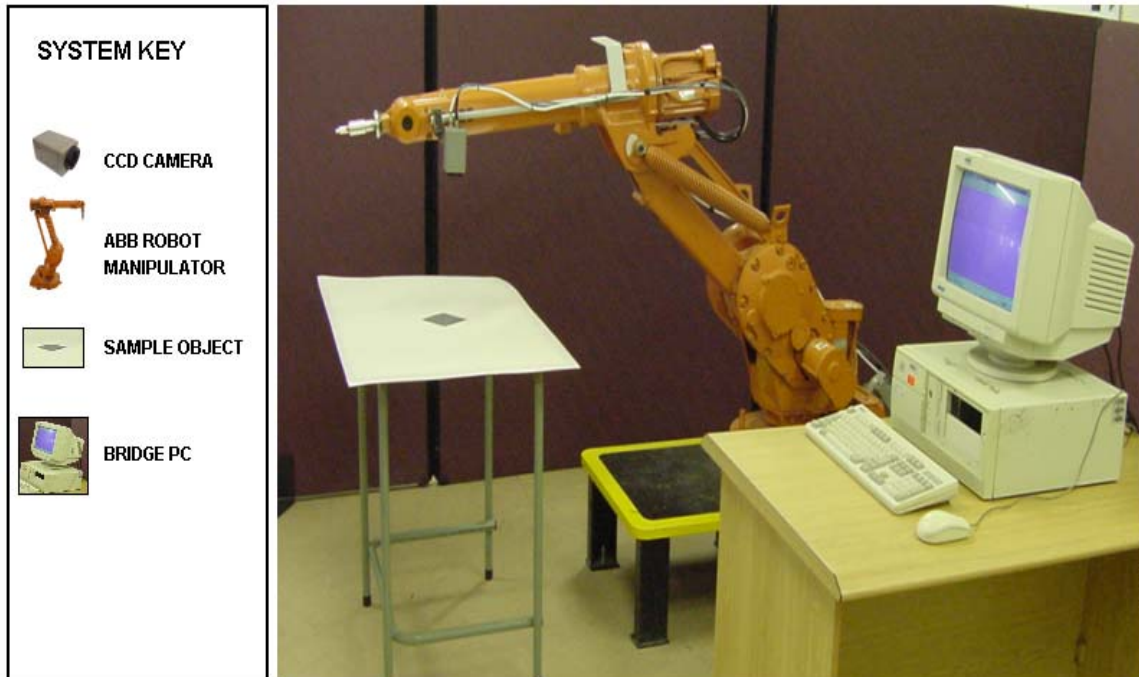


Figure 3.2a: 1400 ABB Robot, Vision Feedback Camera & Bridge PC -System Setup



Figure 3.2b: 1400 ABB Robot Controller & Bridge PC – System Setup

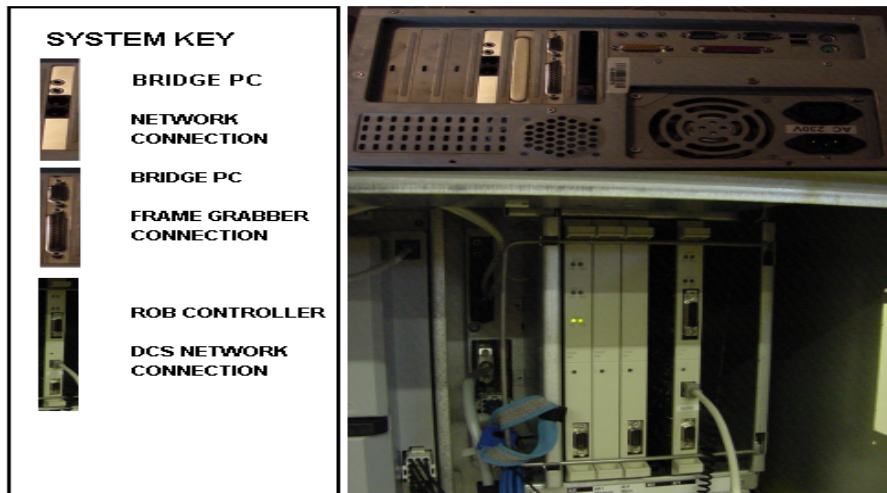


Figure 3.2c: 1400 ABB Robot Controller & Bridge PC Hardware – System Setup

3.2.1 Industrial Robot

The robot being utilized in this research is an ABB IRB 1400 Industrial Robot, which was installed at the Manufacturing Technology Research Centre (MTRC) at PE Technikon. The robot consists of two major components, namely the manipulator and the controller.

The *robot manipulator* has six axes, with spherically-jointed geometry. This provides the robot with six degrees of freedom (6DOF). The manipulator has three axes for positioning and three axes for orientation. The robot was designed for a manufacturing environment, specifically for flexible robot-based automation.

The *controller* has a variety of flexible hardware enabling it to communicate with remote hardware using one of the following methods, e.g. I/O, Analog, Serial RS232, Serial Ethernet, etc. All of the methods use standard industrial interface mediums. The robot has

a work envelope of approximately 1.444 m radius with a repeatable position accuracy of 0.05 mm.

The S4C Controller is a standard controller and is used as the platform for twenty-two different ABB robots. The robot controller consists of three onboard computers to achieve control of the robot, its axis and to perform I/O. A dedicated Ethernet hardware controller manages all data transfer between the remote peripherals (under the TCP/IP protocol). The robot utilizes Baseware as the operating kernel system. This kernel provides complete control of the robot in order to control program execution, communication and motion. Application programs run on top of the operating kernel system. The ABB Robot programming language RAPID is used to program the robot for motion functionality. The programming environment consists of a number of dedicated functions, e.g. MOVEJ, MOVEL, etc.

A *module* contains all data and functions. A *program* may consist of a number of modules, which include user-defined modules and system modules. Only one module will implement a main function, which is the entry point of the program.

A *teach pendant* is connected to the controller and is used as the control and programming user interface. Pull-down menus, dialogues and windows are used to display information to the user. Touch keys and a joystick are provided as input devices.

The controller is provided with Ethernet hardware, which allows the remote user to access robot information and status at high speed. This opens the window to the true sense of real-time control. The Ethernet link can be connected directly to the PC via peer-to-peer interface or via a network hub. The ABB Ethernet hardware and software

require four IP addresses for remote access, which increases the speed of access. The IP address of the remote device must not overlap the IP address of the robot.

The network platform of the robot controller is managed via Factoryware software that is loaded on top of the baseware software and it utilizes the RAP protocol. The RAP protocol manages all data flow between the robot controller and the remote device.

Figure 3.3 illustrates the robot controller Ethernet configuration sequence that is required in order for the Bridge-PC to establish a communication link via the FactoryWare software. Without this setup sequence the communication link would be unavailable.

The following robot controller component entities are required, which will provide the Ethernet communication link:

- Physical Protocol – Ethernet hardware controller card
- Transmission protocol – providing the Industrial robot with a unique IP network address on the Ethernet LAN, IP Address utilized is 100.100.100.102, with a Sub-Net of 255.255.255.0
- Application Protocol – protocol language which will be utilized by the robot, to handle critical data information about the robot controller and manipulator such as, controller status, manipulator TCP position, controller program position, etc

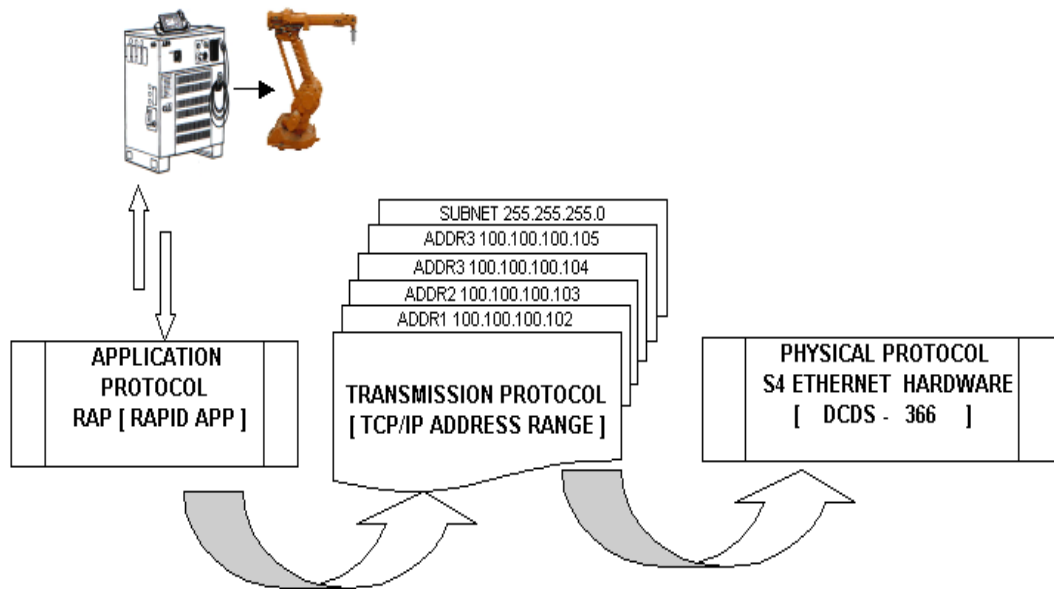


Figure 3.3: Robot Controller Ethernet Communication Configuration

3.2.2 Bridge PC

The PC utilized for this research is a standard office PC with an AMD processor and 512MB of onboard memory, which enhances the overall processing speed of the simulation software. Windows 98 operating system was selected due to the constraints of the software components needed for the robot communication. The PC was equipped with two standard 10 Base / 100 Ethernet hardware cards, which are used for robot communication and LAN. An enhanced video frame grabber 3D graphic hardware card (Flash Point) was installed to process image data.

The Ethernet hardware utilized for the peer-to-peer robot communication was set up with a static IP address 100.100.100.101, subnet 255.255.255.0 while the LAN IP address is dynamic and configured via the remote server, which is illustrated in Figure 3.4.

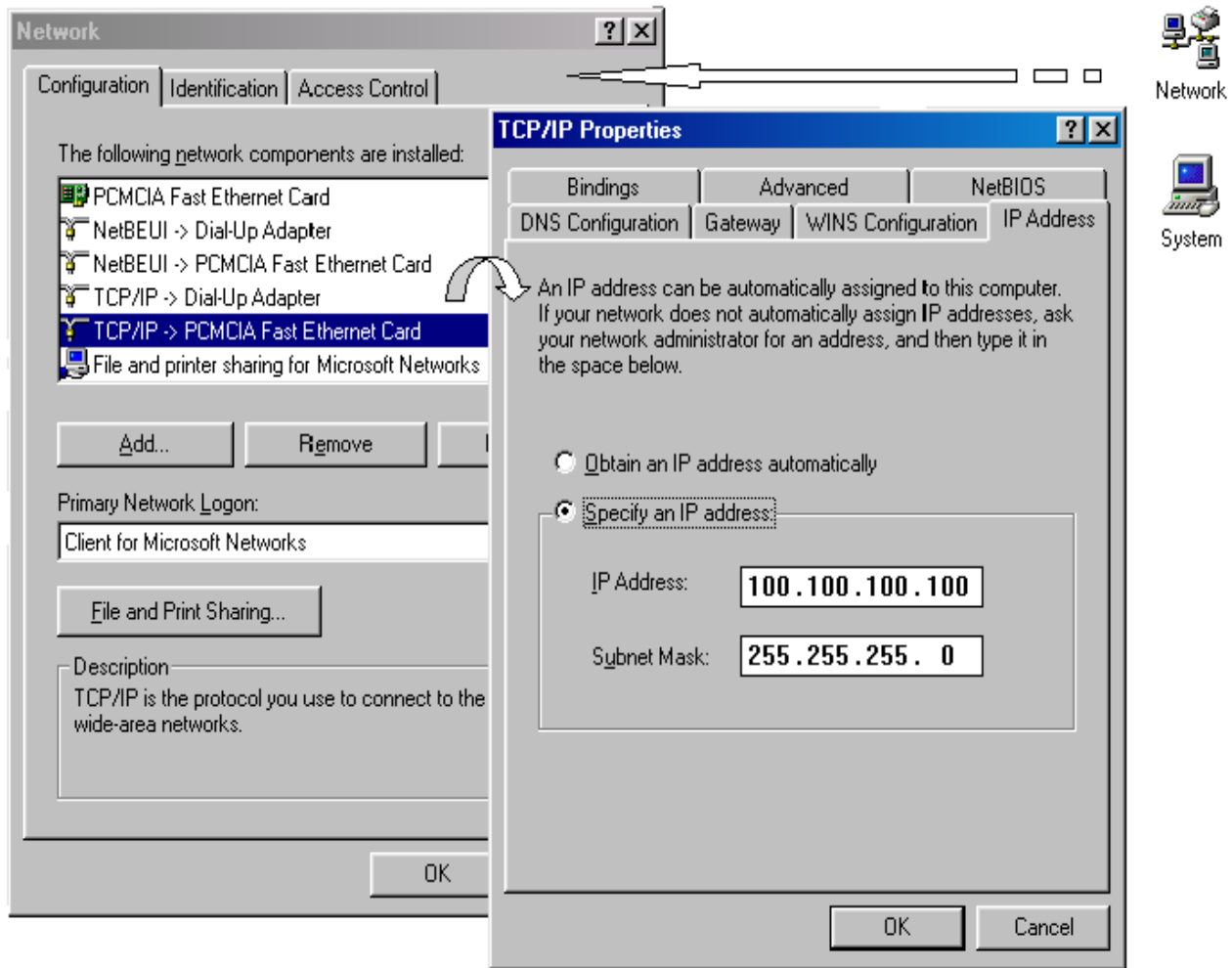


Figure3.4: Experimental Ethernet TCP/IP Configuration Robot & LAN Connection

3.2.3 Robot Vision System

The robot vision system consists of a CCD camera and a Flash Point 3D frame grabber was implemented. The following architecture consists as:

3.2.3.1 CCD Camera Control

CCD camera is the hardware core of vision system. SRC-503HP CCD camera is utilized. [23] It is a high performance CCD camera. It supports high-resolution output

up to 752×582V (440,000 pixels) and its scanning system provides 15,625 kHz (H) and 50 Hz (V). In addition, this camera model supports auto backlight compensation, auto white balance (AGC) and zoom [23].

3.2.3.2 Flash Point 3D Frame Grabber Control

Composite video image (Image resolution of 640×480) is captured by the frame grabber. The Flash point 3D video image brightness, contrast, saturation, etc are manipulating the image processing software to ensure that the best possible image will be processed correctly. Flashpoint 3D frame grabber with PCI bus is employed in this application. This is a high-performance, low-cost PCI frame grabber which is able to capture and display full-frame color and video in real time to VGA display memory. It supports pixel format of 8/16/32 bits per pixel. It supports non-destructive overlay on live video [24].

3.3 Software Architecture

Figure 3.5 shows the software architecture indicating all the software components developed and how these components were integrated to establish an integrated robot vision control system. The software components were logically divided into the following modules:

- Vision Recognition Classes
- Robot Trajectory Control
- RobComm Active X Components and DDE Engine

➤ S4 Robot RAPID Program Structure

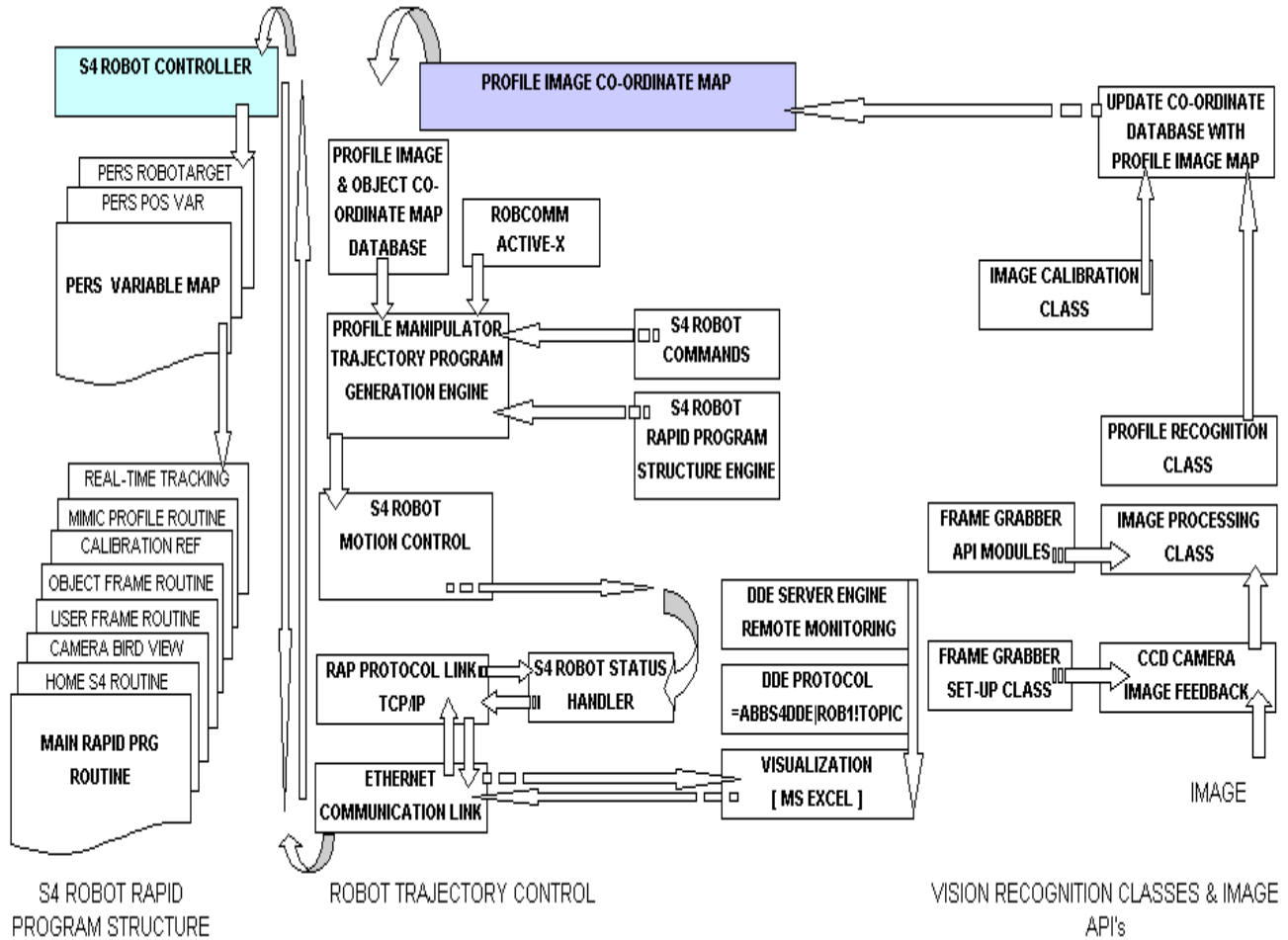


Figure 3.5: Software architecture – Integrated Robot Vision Control system

The following sub-sections describes the implementation aspects of the software components developed:

3.3.1 Vision Recognition Classes and Image API's

FP3D header is the primary library, which contains many vital type definition constants, structure definitions, and prototypes used when calling the Flash Point 3D API. Initialization of the Flash Point 3D library functions must be called to put a live image video window on the VGA display. This is illustrated in Figure 3.6.

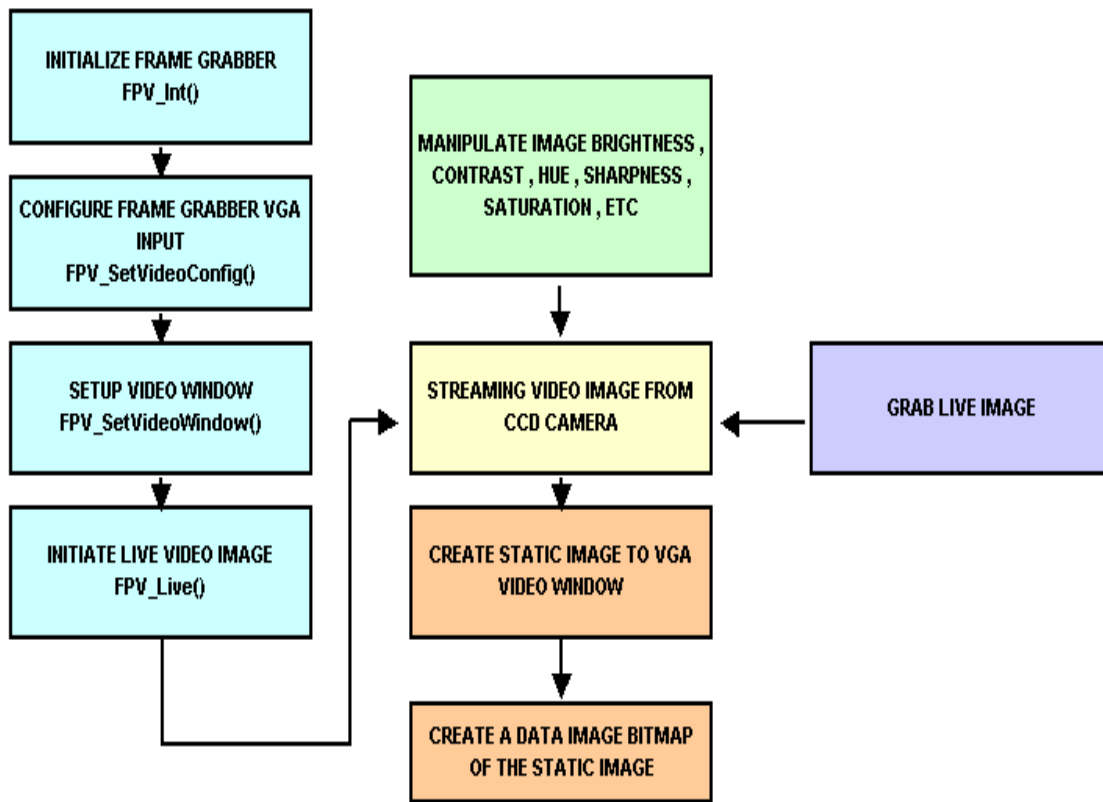


Figure 3.6: Illustrates frame grabber configuration sequence

In general, four FlashPoint 3D library functions must be called to put a live video window on the VGA display, which is listed as follows: [24]

- FPV_Int function initializes FlashPoint VGA to the current loaded configuration values.
- FPV_SetVideoConfig function sets the FlashPoint VGA's video input configuration.
- FPV_SetVideoWindow function sets the size and location of the video window on the FlashPoint VGA's display.
- FPV_VideoLive function starts or stops the incoming video.

The grabbing video image is performed by using the following API functions: [24]

- FPV_CopyVGArect, copies a rectangle of pixel data to or from the VGA frame buffer or offscreen buffer.
- FPV_ScreenToDIB, create a DIB from a screen rectangle.
- FPV_Savefile, saves an image bitmap from memory to disk. The file type is determined from the file extension.

The default mode of grabbing the Flash Point 3D image assumes that the video is always on top. This means that if the video is partially covered by a window or graphics all of the video image is still copied. Once the image is grabbed into video memory, the image is processed and analyzed via algorithms. These algorithms were created in a Microsoft Visual C environment. The profile image map is extracted and processed to create a robot motion trajectory path, which is illustrated in Figure 3.7.

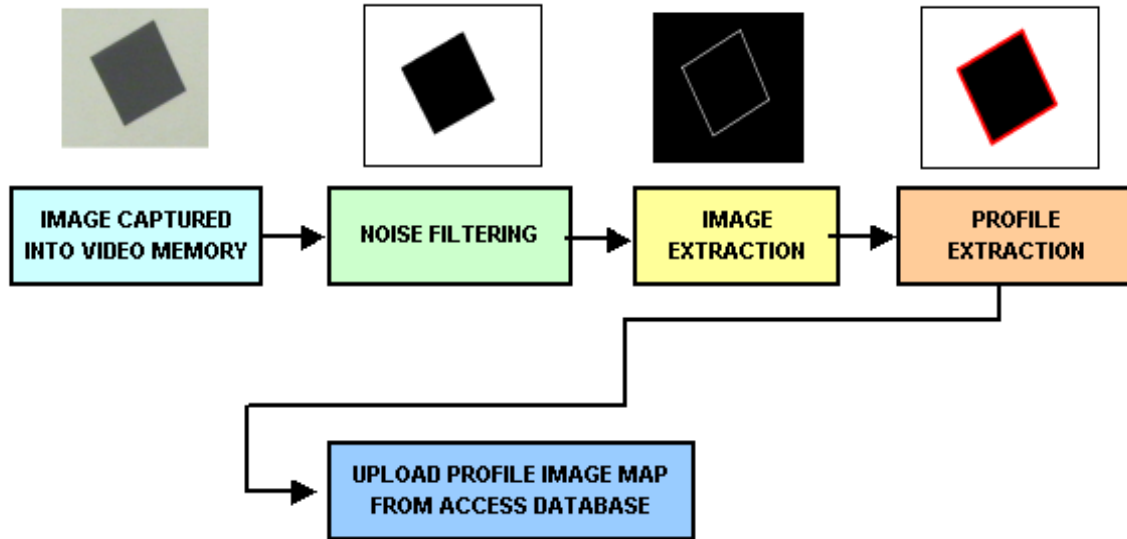


Figure 3.7: Vision Profile Extraction Architecture

The Profile Image Extraction Architecture is composed of the following components:

- Object profile image - captured into the frame grabber's video memory, via the FPV_ScreenToDIB API image function.
- Image Noise filtering - performs the image pre-processing module, which makes use of different filters, such as: Low pass, high pass, median, noise cleaning, averaging, smoothing, etc.
- Image Extraction - performs from the thresholding algorithm, which produces a consistent binary image.
- Profile Extraction - performs edge thinning to produce a one pixel thick boundary and then extracts the geometric features from the boundary.

- Image profile co-ordinate map is uploaded to an access database, which will be utilized to produce a profile trajectory path.

3.3.2 Robot Trajectory Control

The environment for an intelligent robot handling system is always regarded as unpredictable. The path trajectory engine must be planned online, the engine must receive a continuous flow of information about occurring events and generate new controls, while previously controlled motions are being executed. All relevant path trajectory co-ordinate positions must be generated on-line, transferring them to their respective goals.

An architecture of trajectory program generation engine is shown in Figure 3.8 For robot handling system, the relationship between the robot coordinate system (tool coordinate system) and object coordinate system must be created. Transformation matrices are usually employed for this purpose.

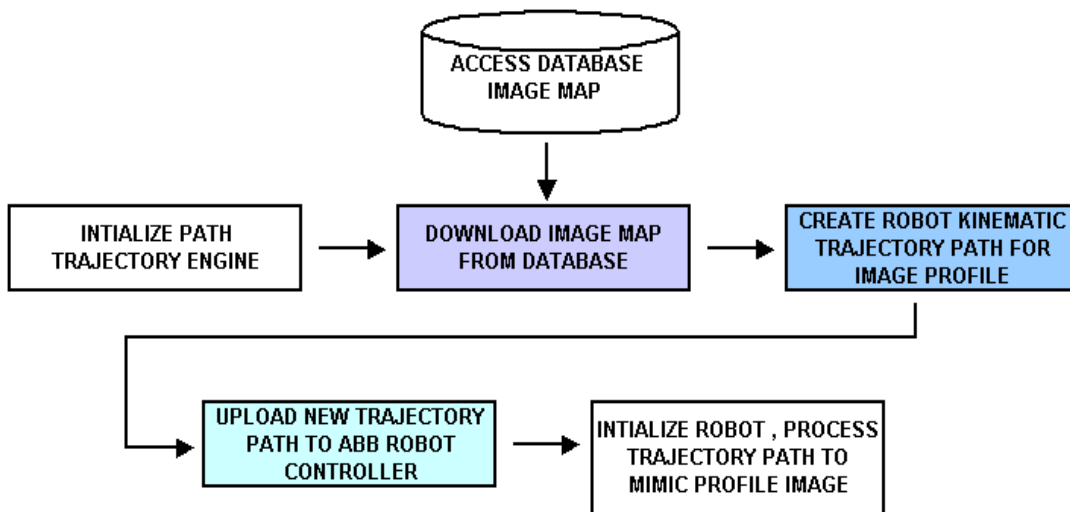


Figure 3.8 Robot trajectory generation engine

Robot trajectory generation engine architecture is composed of the following components:

- Initialize Robot Path trajectory engine, which will automatically generate the robot trajectory RAPID program for an ABB robot controller.
- The image map is downloaded from the Access Database, which contains the image data such as, profile position with respect to the object co-ordinate frame, size, orientation, and the pixel image ratio.
- The image map is used to generate a kinematic trajectory path RAPID robot program for the current image profile, which will be mimicked by the robot.
- The trajectory profile RAPID program is uploaded into the ABB control via an Ethernet communication link. Data transfer is synchronized by the software I/O mechanism to ensure that the robot controller does not react erratically during data transfer.
- Motion Control, which handle the robot motion synchronization.

3.3.3 RobComm ActiveX Components and DDE Engine

3.3.3.1 RobComm ActiveX Components

The Factoryware Interface option enables the robot system to communicate with a PC using RobComm. The RobComm Ethernet configuration is illustrated in Figure 3.9, whereby a unique Alias Name and IP address has to be created which is mapped to

the Bridge PC's static IP address. This will provide the crucial link for the ActiveX component, to access parameter information from the robot controller:

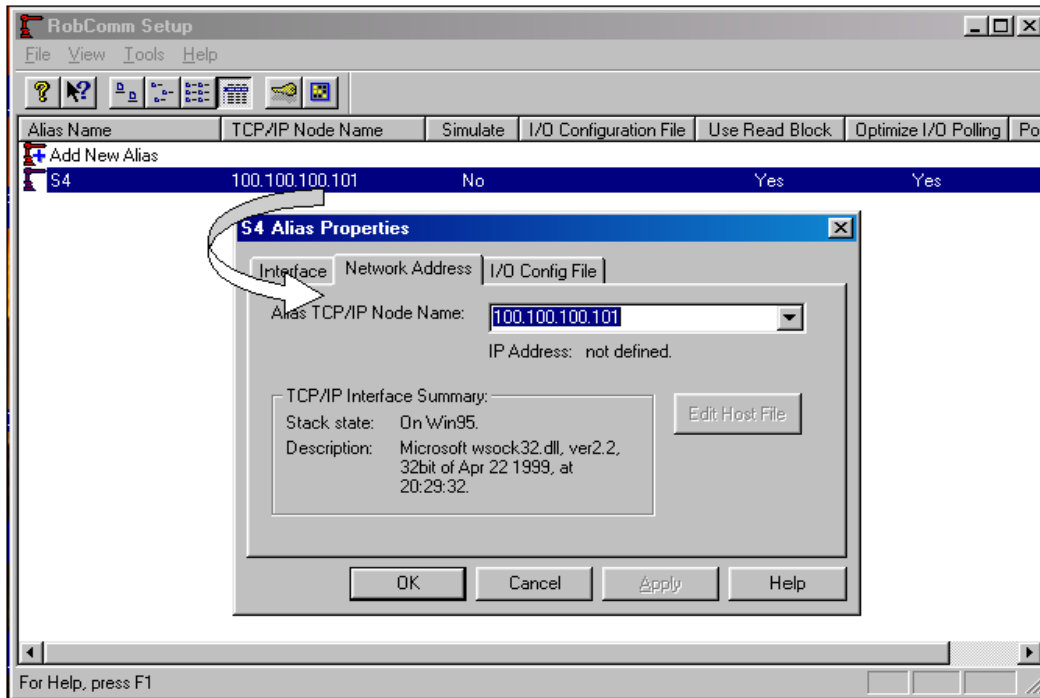


Figure 3.9: FactoryWare Configuration

RobComm requires RAP communication services, which are uploaded to the robot controller. This will enable bi-directional communication flow. RobComm is a collection of ActiveX Controls (OCX). The operation of the OCX controls is configured via the control properties. The OCX components provide a flexible, comprehensive communication interface to the S4 robot controller.

RobbComm ActiveX controls support a 32 bit windows application created with Microsoft Visual Basic, Visual C, or Wonderware InTouch version 7.0.

RobComm is designed to run multiple applications, including multi-threaded applications. It can communicate with multiple S4 controllers without conflict. Applications developed with RobComm work via an Ethernet link to multiple robots.

The Factoryware Interface includes RAP, based on MMS functionality. RAP is used for robot communication, and provide the following functions :

- Start and stop execution
- Transfer program to / from the robot
- Transfer system parameters to / from the robot
- Transfer files to / from the robot
- Read the robot status
- Read or write data
- Read error messages
- Change robot mode
- Read logs

RAP communication is available in both serial and network links as illustrated by Figure 3.10, the Ethernet network configuration has been utilized for this experimental setup, which adds an advantage to the real-time processing for this application [22].

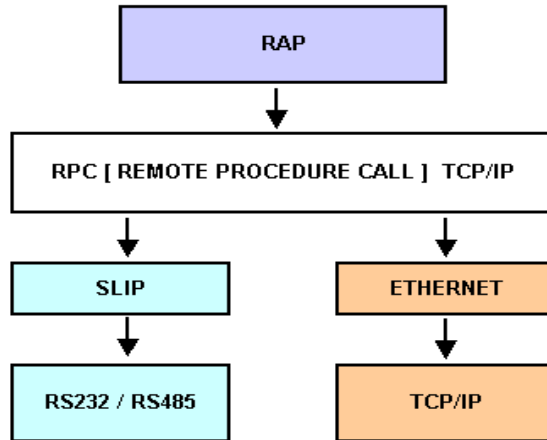


Figure 3.10: S4 Robot Controller communication protocols

3.3.3.2 S4 DDE Server

ABB S4 DDE Server is designed to utilise the robot communication protocols and make you more productive by combining the power of PCs with that of the robots. The ABB S4 DDE Server is a software building block that takes care of the communication with the robot and presents the robot data in the standard DDE format. Any application that can “talk” the DDE “language” can communicate with the ABB robots via the ABB S4 DDE Server. Examples of applications that do DDE communication are Microsoft “Excel” and “InTouch” from Wonderware. With InTouch the user can build his own custom user interface, visualizing his production process. InTouch then needs the ABB S4 DDE Server in order to communicate with the robots. The DDE Server communicates with the robots using the RAP protocol. It maintains a database of the relevant variables in the robot and makes sure that these DDE variables are kept updated all the time.

If new RAPID variables are introduced in the robot program, the DDE Server will create corresponding DDE variables “on-the-fly”. The application using the DDE Server can therefore concentrate on the user interface and rely on the updated DDE variables . The ABB S4 DDE Server provides reading and writing of I/O, RAPID variables and robot system variables. It supports spontaneous messages from the robot (SCWrite), as well as file operations. DDE Server DDE stands for Dynamic Data Exchange. It is a communication protocol designed by Microsoft to allow Windows applications to send and receive data from each other. It is implemented as a client/server mechanism. The server application (like the ABB S4 DDE Server) provides the data and accepts request from any other application that is interested in its data. Requesting applications (like InTouch) are called clients. To obtain data from another application the client program opens a channel to the server application by specifying three things: Figure 3.12 illustrates the ABB S4 DDE Server environment, the application name is “ABBS4DDE”. A topic represents a logical connection to a robot, and identifies the individual robot. The topic names are defined when you configure the robots in the DDE server. Figure3.11 illustrates the format for addressing the DDE items.

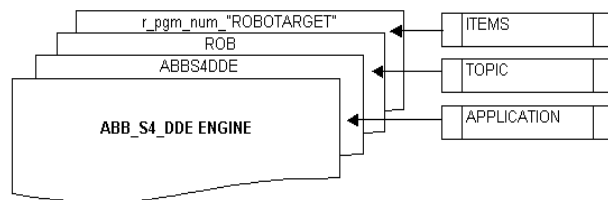


Figure 3.11: DDE addressing structure

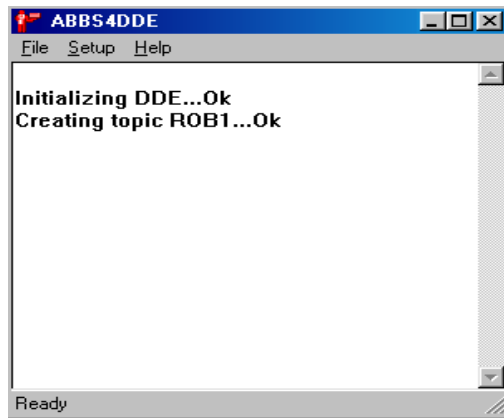


Figure 3.12: DDE Server Engine

3.3.4 S4 Robot RAPID Program Structure

A RAPID program consists of instructions and data. The program is usually made up of three different parts:

- A main routine
- Several subroutines
- Program data

The program memory contains system modules. The main routine is the routine from which program execution starts. Subroutines are used to divide the program up into smaller parts in order to obtain a modular program that is easy to read and maintain. Data is used to define positions, numeric values (registers, counters) and co-ordinate systems, etc. The structure of RAPID program is shown in Figure 3.13.

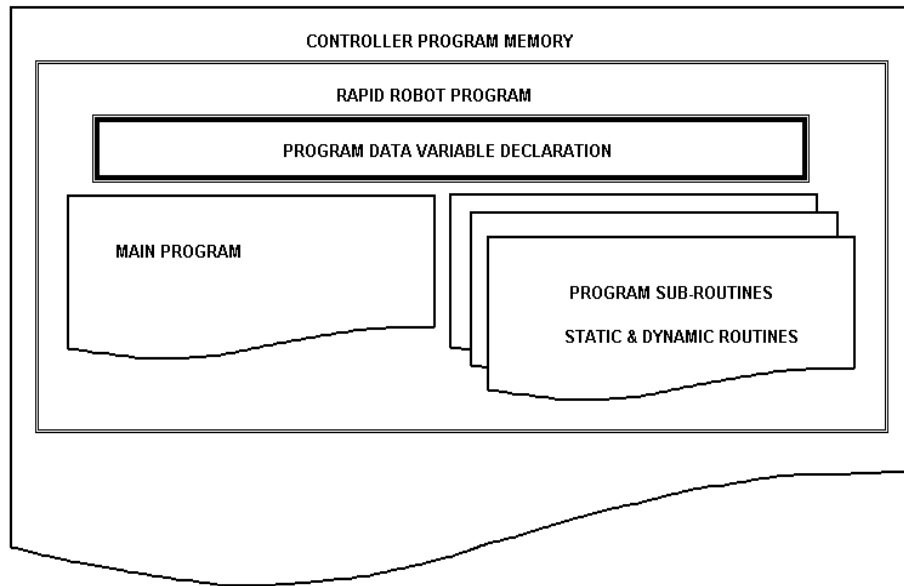


Figure 3.13: S4 ABB Controller RAPID program structure

System modules are programs that are always present in the memory. Routines and data related to the installation rather than the program, such as tools and service routines, are stored in system modules.

ABB Robot RAPID trajectory path structure is composed of the following components. This program structure will be utilized to form the main robot motion components in the experimental setup. The RAPID program structure, which is utilized in the experimental setup is illustrated in Figure 3.14, it comprises of the following components:

- Main Program Routine – which handles the sub-routine call synchronization.
- Persistence Robot Target Variables declaration.
- Static Sub-Routines – this handles the static robot target co-ordinate routines.
- Dynamic Sub-Routine – this handles the dynamic object co-ordinate frame.

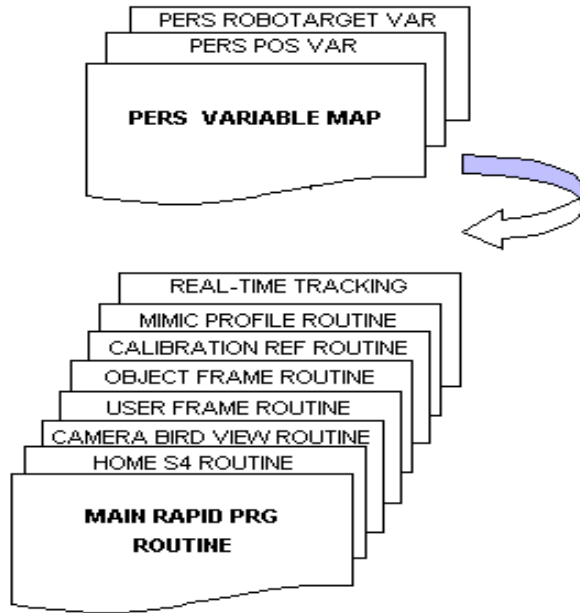


Figure 3.14: ABB Robot RAPID trajectory path structure

3.4 Conclusion

This chapter focused on the integrated hardware and software architecture for the communication and software development modules that will enable real-time robotic vision guided control.

This architecture allows an industrial robot system to respond to changes of object displacement and orientation, which would in turn possibly provide the system with additional flexibility. This can be achieved by a robot vision sensory feedback system and a PC-based robot control system. Object-oriented techniques are employed in the software development. PC-bus interface cards (frame grabber, PMAC card, data acquisition card, and Ethernet card) are utilized, as well as ActiveX techniques are widely exploited to build up a configurable system. The proposed architecture in Figure 3.1 puts

forward a generic framework for a remote PC-Based robot control system, which organizes the system hardware and software components and reveals their relationships.

Chapter 4 will describe the development of the software modules required to control the robot motion based on the visual information. The software components to perform object recognition is discussed in Chapter 5.

CHAPTER 4

PC-BASED ROBOT TRAJECTORY PATH CONTROL SYSTEM

To achieve this control software algorithms were developed using the robotic RAPID motion fundamental instruction set, RAPID programming environment, direct and inverse kinematics path trajectory control fundamentals, RAP Communication and Robot DDE Server aspects described in Chapter 2 and Chapter 3.

Figure 4.1 illustrates the control architecture that was constructed to achieve the PC-Based robot control mechanism as well as the interface to the vision sensory system described in Chapter 5. In order to achieve the above control mechanism, robot motion fundamentals were developed into algorithms, which will be discussed in detail. The programming architecture provides a platform for further development of a remote robot control. It also considers all software components implemented in the development of robot control via an Ethernet communication to an ABB industrial Robot Controller, communication aspects although indicated, were covered in Chapter 3.

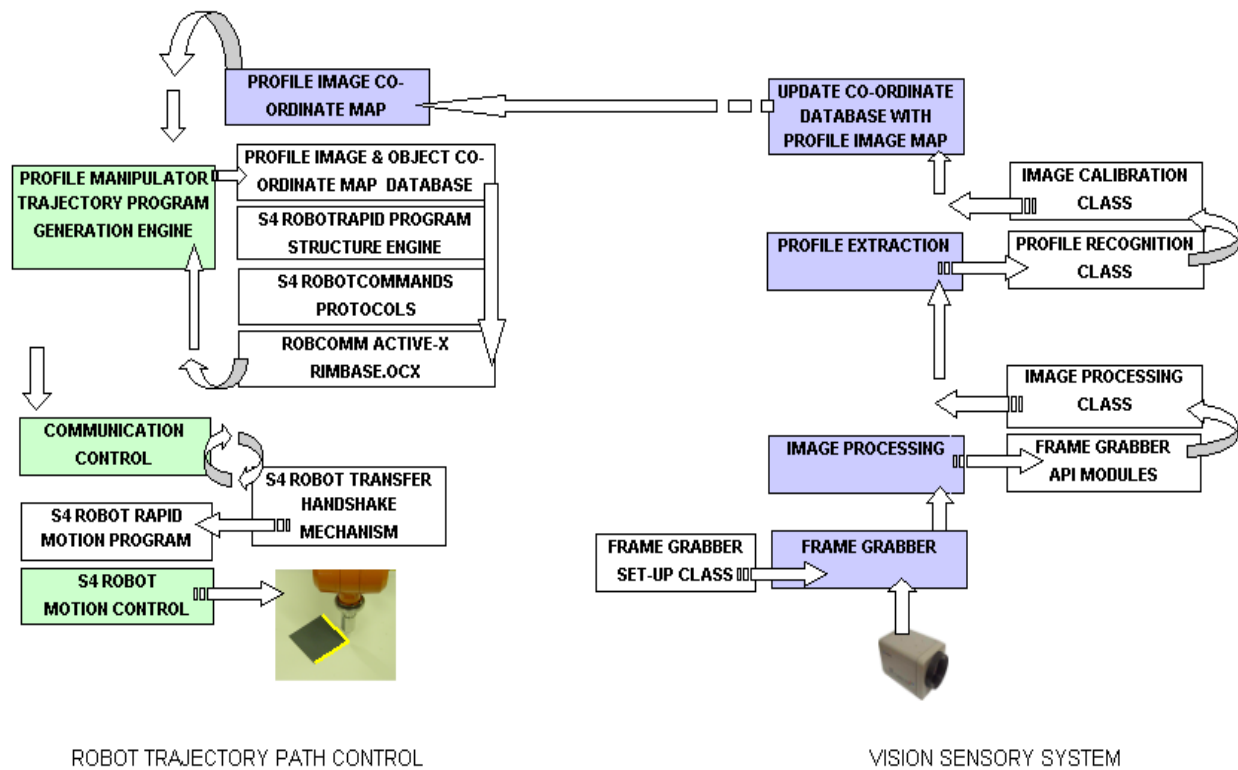


Figure 4.1: PC-Based Robot Trajectory Path Control Architecture

The PC-Based Robot Trajectory control system receives a profile map from the vision sensory system, it consists of the following components:

➤ **Frame Grabber**

The CCD Camera provides the vision sensory feedback system for this research project, which provides the system with greater movement flexibility.

➤ **Image Processing**

Image processing Class – A raw image is captured into dynamic memory where algorithms analyze the binary map of the image by filtering and cleaning random

noise. Threshold is utilized to provide a gray-scale of the image. The object can be clearly distinguished from the background.

➤ **Profile Extraction**

Profile Extraction Class – Utilizes close chain vectors, which utilizes standard mathematic algorithms. This ensures that the chain profile segments contain the magnitude and position of the object.

➤ **Profile Manipulation and Trajectory program generation Engine**

This uses the extracted profile to automatically create a robot trajectory path that will track the image.

➤ **Image Database**

This provides a storage medium that handles the image profile co-ordinate map. The co-ordinates will be utilized by the robot trajectory program generation engine to create a RAPID trajectory program.

➤ **Profile Manipulation and Trajectory Program Generation Engine**

The PC-Based RAPID program engine was developed to provide a platform for the trajectory path control. This engine provides an environment for basic RAPID motion program development and control. This environment utilizes direct and inverse kinematics path modeling algorithms to control TCP position of the robot manipulator. The RobComm ActiveX components form the most important building blocks for the research platform, which provides the tools for the system

integration. The vision feedback system is integrated via a PC-Based robot control to an industrial robot controller. Without this ActiveX component the research would not have been possible.

➤ **Communication Control**

The control mechanism establishes and maintains the communication link between the motion control engine and the RobComm server.

➤ **Motion Control**

The control mechanism synchronizes robot motion control.

➤ **DDE Server Engine**

The DDE server engine provides additional programming flexibility for industrial Visualization software environments. [Intouch Wonderware SCADA Software]

The following sub-sections provide the fundamentals for the RAPID programming environment, which are used to construct software algorithms that will transform the image profile to actual trajectory motion commands in order for the system to trace the image profile on-line.

4.1 Robot RAPID Motion Control Commands

Path motion forms the main component of any industrial robot. Figure 4.2 illustrates the move command structure for an ABB robot controller, which will servo the robot manipulator to specific path co-ordinates that have been predefined or calculated from

robot kinematics algorithms. The Industrial robot MOVE command is structured as illustrated in Figure 4.2.

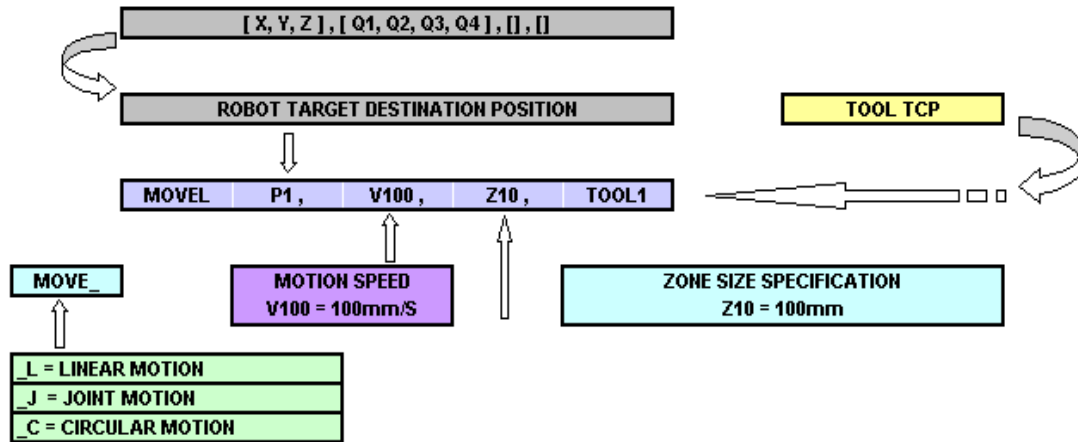


Figure 4.2: MOVE command architecture

MOVE Command consist of the following components:

- MOVE_L = Linear path motion
- P1 = the destination position to which the robot is to move
- V100 = Motion Speed
- Z10 = Zone size (accuracy) , i.e. how close the robot must be to the destination position before it can start to move towards the next position
- TOOL1 = Current Tool Position (TCP)

Motion MOVE Instruction

The robot manipulator position control managed in this research project is illustrated in Figure 4.3.

Linear Motion MOVE[L]

The linear motion functionality

Joint Motion MOVE[J]

The JOINT motion functionality

Joint Motion MOVE[C]

This JOINT motion was not utilized in the research project, but is available to the robot system.

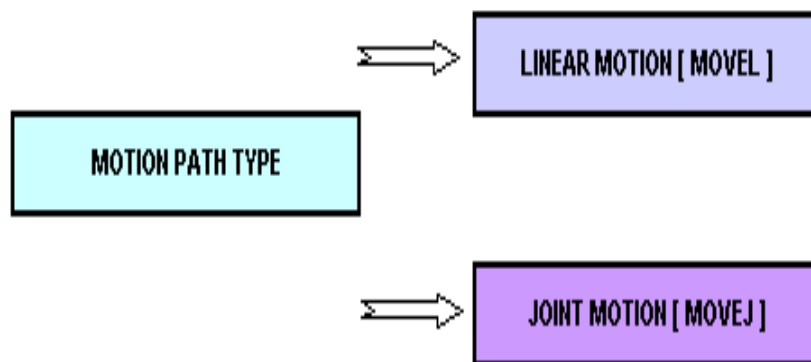


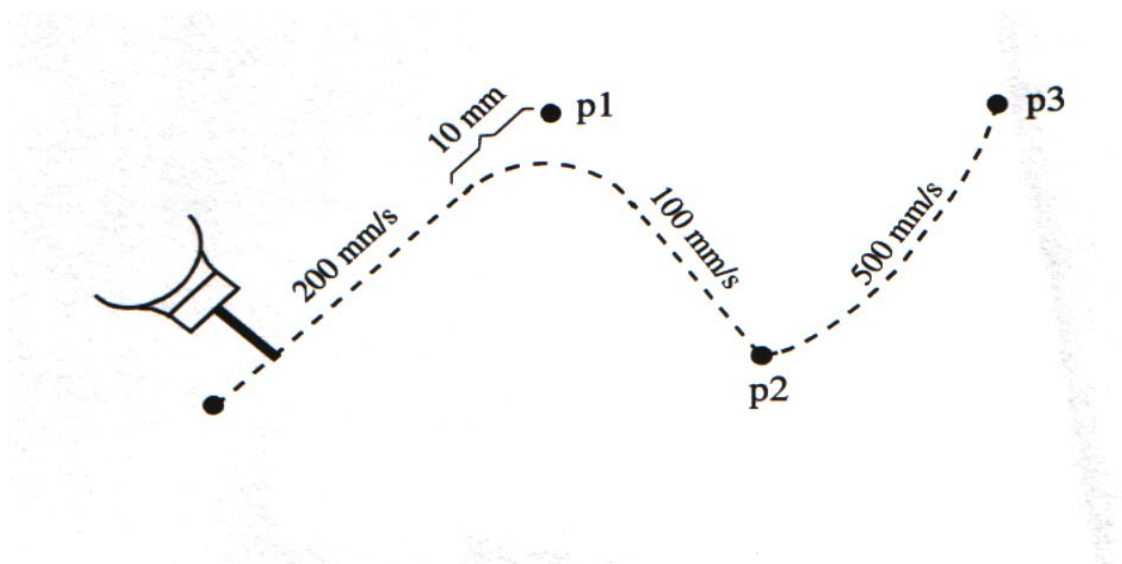
Figure 4.3: Motion path type commands

Motion SPEED and ZONE size specification

The speed and zone size refer to different data fields, which include the desired speed in mm/s, zone size in mm, etc. You can create and name these data fields yourself, but the most commonly used values are already available.

Motion TOOL TCP

One must then specify the tool, its dimensions and weight, in the tool data. The TCP of the tool is moved to the specified destination position when the instruction is executed, as illustrated in Figure 4.4



```
MoveL p1, v200, z10, tool1  
MoveL p2, v100, fine, tool1  
MoveL p3, v500, fine, tool1
```

Figure 4.4: Positioning the robot

4.2 RAPID Program Data Types

Figure 4.5 displays the variable declaration for a robot target position which will be utilized in the remote manipulation/construction software for robot position manipulation.

```
START_POS = robtarget
```

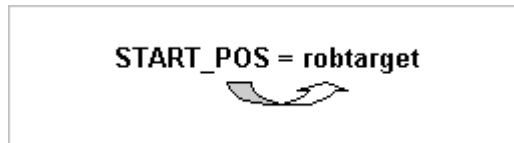


Figure 4.5: Robtarget variable declaration

The robot target variables consist of the following structures. It contains a (x, y, z) coordinate frame with (q1, q2, q3, q4) which indicates the robot orientation position. The data type “**Robtarget**” is used for the robot’s position, which includes the orientation of the tool and the configuration of the axes.

```
Var POS position (x,y,z)  
POS START_POS  
START_POS:=[500,0,940]
```

The data type “**Orient**” is used for orientation (such as the orientation of a tool) and rotation (such as the rotation of a co-ordinate system).

```
Var ORIENT Orient  
ORIENT ORIENT1  
ORIENT1:=[1,0,0,0]
```

The orientation must be normalized and the sum of the system must be equal to one.

$$1 = q^2_1 + q^2_2 + q^2_3 + q^2_4$$

A *quaternion* describes the rotational matrix. Quaternions are calculated based on the elements of the rotational matrix. Figure 4.6 illustrates the orientation algorithms.

$$\begin{aligned}
 q_1 &= \frac{\sqrt{x_1 + y_2 + z_3 + 1}}{2} \\
 q_2 &= \frac{\sqrt{x_1 - y_2 - z_3 + 1}}{2} & \text{sign } q_2 &= \text{sign } (y_3 - z_2) \\
 q_3 &= \frac{\sqrt{y_2 - x_1 - z_3 + 1}}{2} & \text{sign } q_3 &= \text{sign } (z_1 - x_3) \\
 q_4 &= \frac{\sqrt{z_3 - x_1 - y_2 + 1}}{2} & \text{sign } q_4 &= \text{sign } (x_2 - y_1)
 \end{aligned}$$

Figure 4.6: Quaternions algorithms

4.3 ABB Robot RAPID Program and Motion Control

In this research project the robot motion is controlled from the RAPID program, which has been automatically generated from the PC-Based Trajectory application. There is numerous programming structures that can be utilized, these structures are selected depending on required motion tasks. The current programming structure that was developed creates a flexible approach to achieve the end goal of the research project. The program has been segmented into a main program with a dynamic sub-routine and numerous static sub-routines.

The *static components* are position co-ordinates that are utilized for calibration, object viewing, and object co-ordinate marker positions. The *dynamic component* manipulates the robot manipulator with respect to the object profile position co-ordinate map. Figure 4.7 illustrates the program components architecture (static and dynamic). These components will be discussed in detailed below.

This co-ordinate map contains the orientation and position of the object profile. The CCD camera sensory feedback provides a closed loop system for the robot.

The object image is viewed via a CCD camera. Image processing software manipulates the captured object profile, which extracts the image co-ordinate map. This co-ordinate map is fed back to the robot controller which servo's the robot manipulator to trace the object profile.

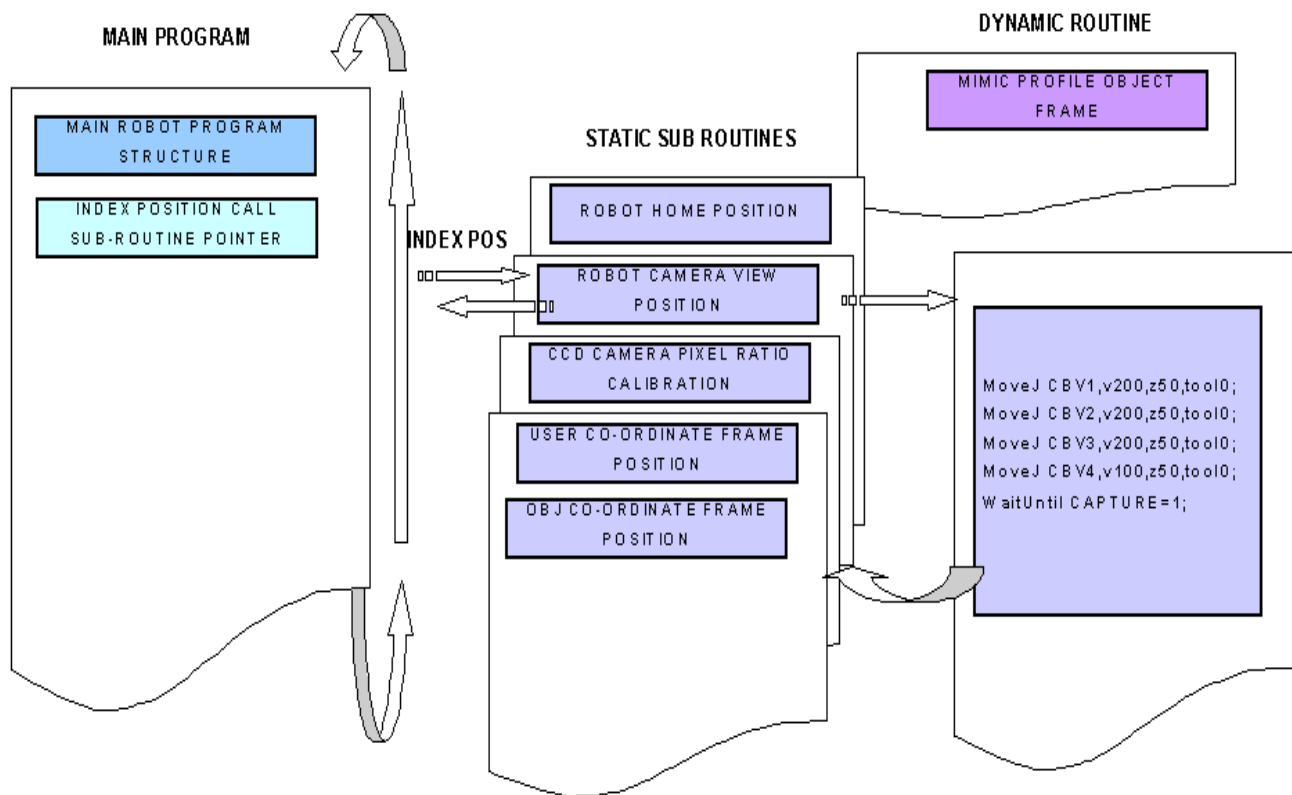


Figure 4.7: RAPID Robot program architecture

The RAPID program consist of the following components:

The program architecture is illustrated in Figure 4.7.

Static RAPID Sub-routines

➤ Main Robot program routine

Main RAPID Program routine handler ensures that the correct routines are manipulated when requested by the remote trajectory application. The program sub-routines CALL synchronization utilizes a software handshake routine that has been developed. This handshake has been configured in such that the software “INDEX_MARKER” provides an index. This index ensures that the correct sub-routine is called at the appropriate time. Due to the constraints of the RobComm ActiveX components a task program structure was developed whereby the numeric “INDEX_MARKER” variable was utilized to CALL the correct program sub-routine, for example when the “INDEX_MARKER=4” the system would jump to the “MIMIC_PROFILE” sub-routine. This would in turn trace the image profile, once the routine has been completed the “INDEX_MAKER=0” and return the main program routine.

This “INDEX_MARKER” has been configured to be a persistent variable, which can be manipulated from the remote application.

The Main RAPID program routine is illustrated below.

```

//*****
PROC main()
  WHILE(TRUE)

    //SUB ROUTINE THAT SERVO'S THE MANIPULATOR
    //TO A ZERO DEGREE REFERENCE POSITION.

    IF INDEX_MARKER=1 THEN
      HOME;
    ENDIF

    //SUB ROUTINE THAT SERVO'S THE MANIPULATOR
    //TO THE OBJECT VIEWING POSITION IO ORDER TO CAPTURE
    //THE VIEWED IMAGE.

    IF INDEX_MARKER=2 THEN
      CAMERA_BIRD_VW;
    ENDIF

    //SUB ROUTINE THAT PROVIDES THE SYSTEM WITH A
    //CALIBRATION REFERENCE LINE.

    IF INDEX_MARKER=3 THEN
      CAL_OBJ;
    ENDIF

    //DYNAMIC SUB ROUTINE THAT IS UTILIZED TO CONTROL
    //THE MOTION WITH RESPECT TO THE OBJECT PROFILE.

    IF INDEX_MARKER=4 THEN
      MIMIC_PROFILE;
    ENDIF

    //SUB ROUTINE THAT PROVIDES A SIMPLE MECHANISM
    //TO CAPTURE THE OBJECT CO-ORDINATE REFERNECE
    //FRAME

    IF INDEX_MARKER=5 THEN
      OBJ_CFRAME;
    ENDIF

  ENDPROC
//*****

```

➤ **Robot Home Position Sub-routine**

The robot HOME position routine forms part of the static movement commands. Once the command has been called, the system will request the robot controller to rotate and move all robot manipulator axes to their HOME position, which is the zero degree state. Figure 4.8 graphically illustrates the RAPID sub-routine call procedure.

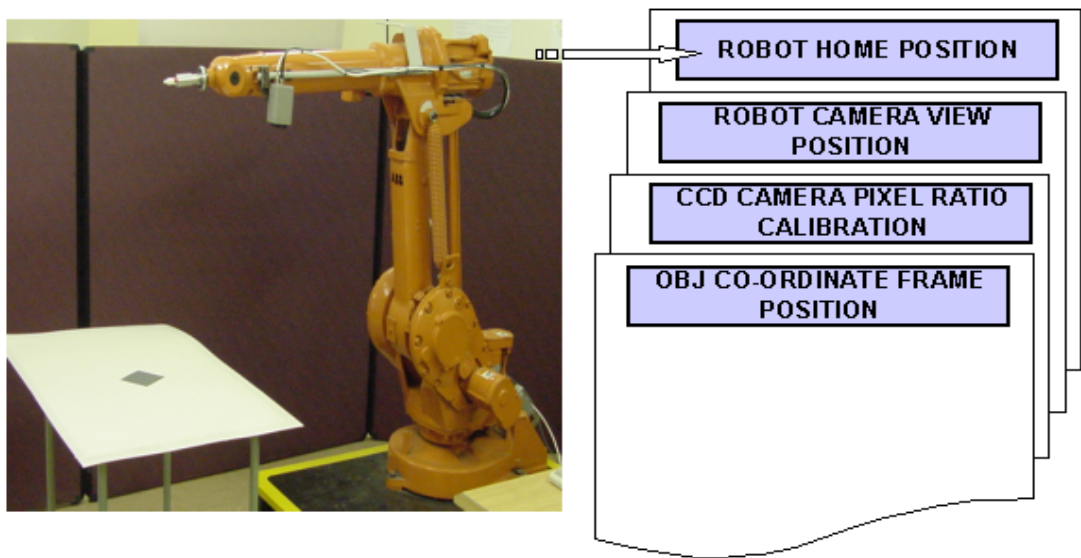


Figure 4.8: Home sub-routine function

The software source code is illustrated below:

```

//*****
SUB Routine RAPID program
PROC HOME()
  MoveAbsJ[[0,0,0,0,0,0],
            [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
            ,v200,z50,tool0;
ENDPROC

//*****

```

➤ **Robot Camera view position Sub-routine**

The robot “CAMERA_BIRD_VW” position routine forms part of the static movement commands .The robot controller will move the robot manipulator to the relevant position where the test object can be viewed correctly. A software BOOL flag “CAPTURE” is utilized for the system to be synchronized when the image has been correctly captured. The position will be maintained until the system request the robot controller to return the manipulator to the “HOME” position. Figure 4.9 graphically illustrates the RAPID sub-routine call procedure.

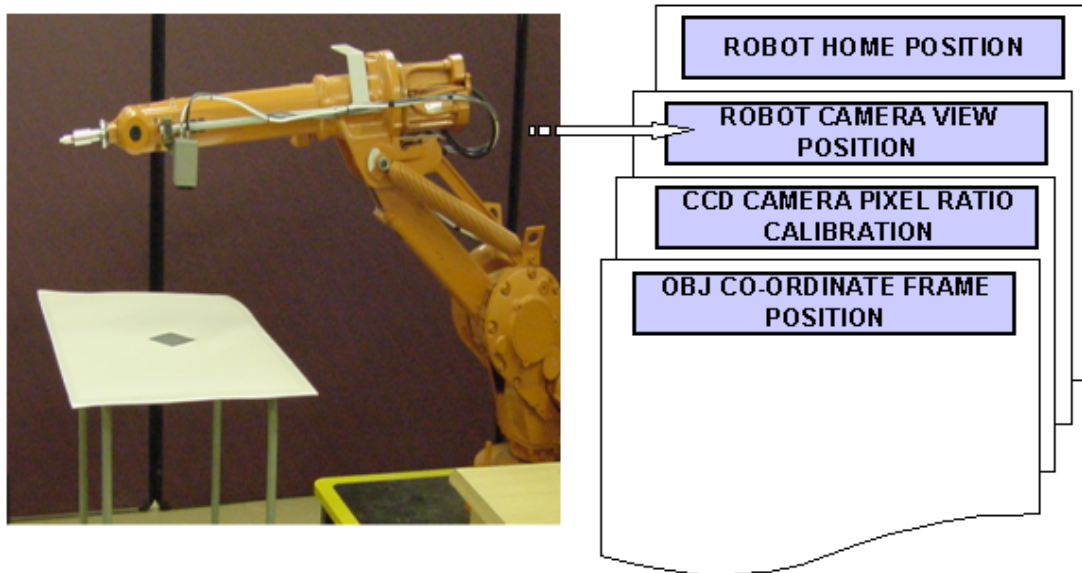


Figure 4.9: Robot camera view sub-routine

The software source code is illustrated below :

```
//*****  
PROC CAMERA_BIRD_VW()  
  MoveJ CBV1,v200,z50,tool0;  
  MoveJ CBV2,v200,z50,tool0;  
  MoveJ CBV3,v200,z50,tool0;
```

```

MoveJ CBV4,v100,z50,tool0;
WaitUntil CAPTURE=1;
MoveAbsJ[0,0,0,0,0,0],
          [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
          ,v200,z50,tool0;
INDEX_MARKER:=0;
ENDPROC
//*****

```

➤ **CCD Camera Pixel Ratio Calibration Sub-routine**

The robot “CAL_OBJ” position routine forms part of the static movement commands. This routine is utilized for system calibration where the robot manipulator will be commanded to mark off a specific calibration length that has been pre-programmed, for the vision system which will view the refer calibration marker and the object image pixel ratio can be calculated, to ensure that the correct object view is with in correct proportions. Figure 4.10 graphically illustrates the RAPID sub-routine call procedure.

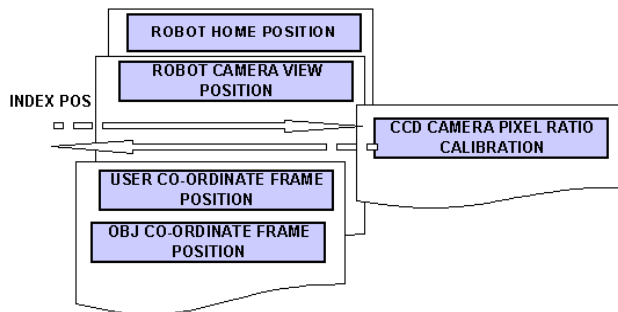


Figure 4.10: CCD camera pixel ration calibration

The software source code is illustrated below:

```

//*****
PROC CAL_OBJ()

MoveJ CO1,v150,z50,tool0;
MoveJ CO2,v150,z50,tool0;

```

```

MoveL Offs(CO2,0,200,-30),v50,z1,tool0;
MoveAbsJ[[0,0,0,0,0,0],
          [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
          v200,z50,tool0;
INDEX_MARKER:=0;

ENDPROC
//*****

```

Dynamic RAPID Sub-routine

➤ MIMIC Profile Object Frame Sub-routine

The robot “MIMIC_PROFILE” position routine forms the main part of the dynamic movement routines. This routine is utilized for the object profile, to trace the captured image, which is used to manipulate the robot manipulator to the required trajectory movement profile. Figure 4.11a-b graphically illustrates the RAPID dynamic sub-routine call procedure which traces the object profile.

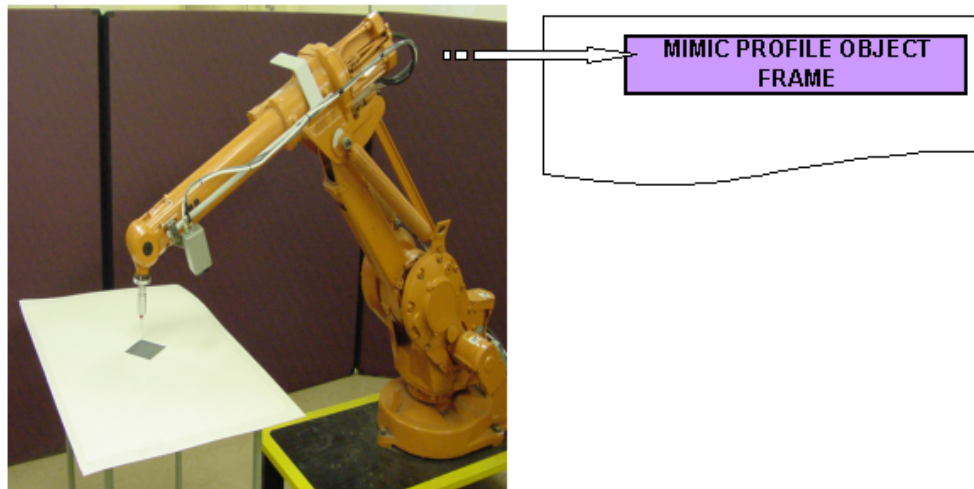


Figure 4.11a: Mimic profile object sub-routine

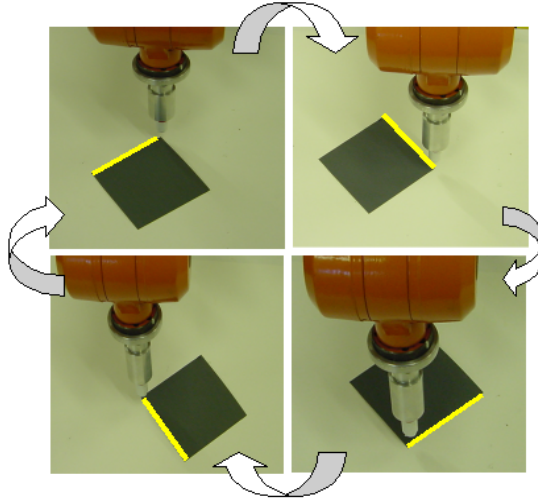


Figure 4.11b: Mimic profile object sequence

The software source code is illustrated below :

```

//*****
PROC MIMIC_PROFILE()

PROC MIMIC_PROFILE()
  MoveJ PRP1,v150,z50,tool0;
  MoveJ PRP2,v150,z50,tool0;

  START_POS:=PRP2;
  MoveJ START_POS,v150,z50,tool0;
  MoveL Offs(START_POS,0,0,0,Z_OFFSET),v150,z50,tool0;
  MoveL Offs(START_POS,X_POS1,Y_POS1,Z_OFFSET),v150,z50,tool0;
  MoveL Offs(START_POS,X_POS2,Y_POS2,Z_OFFSET),v150,z50,tool0;
  MoveL Offs(START_POS,X_POS3,Y_POS3,Z_OFFSET),v150,z50,tool0;
  MoveL Offs(START_POS,X_POS4,Y_POS4,Z_OFFSET),v150,z50,tool0;
  MoveJ PRP3,v150,z50,tool0;

  MoveAbsJ
  [[0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,z50,tool0;
  ENDPROC
  MoveAbsJ[[0,0,0,0,0],
           [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
           v200,z50,tool0;

  ENDPROC
//*****

```

In order for the remote system to servo the robot manipulator to a required target position, the robtarg variables have to be configured as “PERSISTENCE = PERS“, this ensures that the RobComm handler can re-configure the target positions on the fly.

The RAPID variable structure is illustrated below:

```
//*****  
PERS robtarg PRP4:=[[955.01,0,1195],[0.707106,2E-06,0.707108,2E-06],  
[0,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
Constant robtarg PRP3:=[[1146.26,-187.55,576.24],[0.008364,0.118848,-0.992864,-  
0.005275],[-1,0,-,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];  
//*****
```

4.4 Robot Motion Control using Kinematics

A mechanical manipulator can be modeled as an open-loop articulated chain with several rigid bodies (links) connected in series by either revolute or prismatic joints driven actuators. One end of the chain is attached to a supporting base while the other end is free and attached with a tool (end-effector) to manipulate objects or perform assembly tasks. The relative motion of the joints results in the motion of the links that positions the hand in a desired orientation. In most robotic applications one is interested in the spatial description of the end-effector of the robot manipulator with respect to a fixed reference co-ordinate system.

Robot arm kinematics deals with the analytical study of the geometry of the motion of a robot arm with respect to a fixed reference co-ordinate system. In this research project two modeling methods are dealt with which form the basis in which the ABB IRB 1400 robot is controlled.

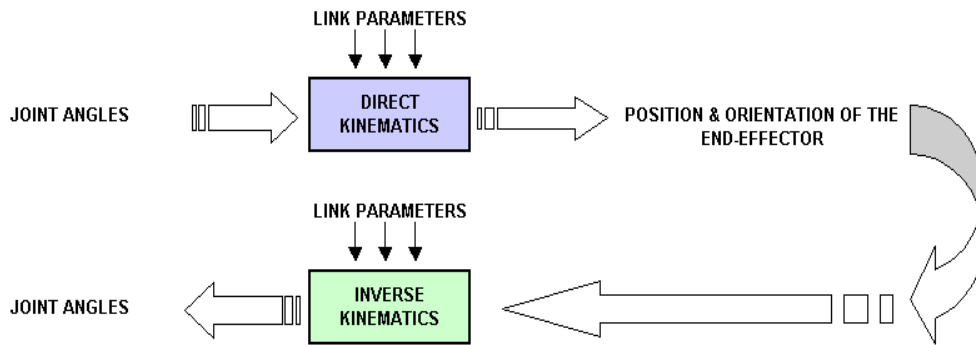


Figure 4.12: Direct and inverse kinematics relationship

Since the independent variables in a robot arm are the joint variables and a task is usually stated in terms of a reference co-ordinate frame, the inverse kinematics problem is used more frequently. Figure 4.12 illustrates the relationship between these two modeling methods.

4.4.1 Direct Kinematics Solution

There are numerous methods and algorithms used to determine and display the tool center position value of the robot. The method used in this project, the Denavit-Hartenberg (D-H) theory, states that a 4x4 homogeneous transformation matrix represents each link co-ordinate system at the joint with respect to the previous link co-ordinate system. Thus, through sequential transformations, the end-effector expressed in the “hand co-ordinates” can be transformed and expressed in the “base co-ordinate” which makes up the inertial frame of the dynamic system.

4.4.2 Direct Kinematic Computation for an IRB 1400 Robot

An *orthonormal cartesian co-ordinate system* (x_i, y_i, z_i) can be established for each link at its joint axis, where $i = 1, 2, \dots, n$ ($n =$ number of degrees of freedom) plus the base co-ordinate frame. Since a rotary joint has only one degree of freedom, each (x_i, y_i, z_i) co-ordinate frame of the robot arm corresponds to joint $i + 1$ and is fixed to link i . When the joint actuator activates joint i , link i will move with respect to $i - 1$.

The base co-ordinates are defined as the 0^{th} co-ordinate frame (x_0, y_0, z_0), which is also the *inertial co-ordinate frame*. With respect to the ABB IRB 1400 robot, the *tool co-ordinate system* is referenced by (x_6, y_6, z_6).

Co-ordinate frames are established based on the following rules:

- The z_{i-1} axis lies along the axis of motion of the i^{th} joint.
- The x_i axis is normal to the z_{i-1} axis, and points away from it.
- The y_i axis completes the right-handed co-ordinate system.

| Joint (i) | angle | | a(i) | d(i) | Joint Range |
|-----------|-------|------|------|------|----------------|
| 1 | 0 | - 90 | 150 | 475 | - 170 to + 170 |
| 2 | - 90 | 0 | 600 | 0 | - 70 to + 70 |
| 3 | 0 | - 90 | 120 | 0 | - 65 to + 70 |
| 4 | 0 | + 90 | 0 | 720 | - 150 to + 150 |
| 5 | 0 | -90 | 0 | 0 | - 115 to + 115 |
| 6 | 180 | - 91 | 0 | 85 | - 300 to + 300 |

Figure 4.13: Link parameters and joint angle range for the ABB 1400 Industrial Robot

These link robot parameters, which are illustrated in Figure 4.13, are the true offset values for a ABB robot which is utilized in the forward kinematics algorithms. This, ensure that the correct TCP of the robot will be calculated.

Kinematic Equations for an ABB Industrial Manipulator

The homogenous matrix, ${}^R T_H$ which provides the robot tool centre position and orientation with respect to the base co-ordinate system, needs to be calculated by a matrix chain product of successive co-ordinate transformation matrices of ${}^{i-1} A_i$, and is expressed as

$${}^0 T_i = {}^0 A_1 \cdot {}^1 A_2 \cdot {}^2 A_3 \cdot {}^3 A_4 \cdot {}^4 A_5 \dots \dots \dots {}^{i-1} A_i \quad \text{for } i = 1, 2, \dots, n$$

$$= \begin{bmatrix} x_i & y_i & z_i & p_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$\begin{bmatrix} x_i & y_i & z_i & p_i \end{bmatrix}$ = orientation matrix of the i th co-ordinate system established at the link $[i]$ with respect to the base co-ordinate system. It is the upper left 3x3 partitioned matrix of ${}^0 T_i$

p_i = position vector which points from the origin of the base co-ordinate system to the origin of the i th co-ordinate system. It is the upper right 3x1 partitioned matrix of ${}^0 T_i$

$${}^0A_1 = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & a_1 \cdot \cos\theta_1 \\ \sin\theta_1 & 0 & \cos\theta_1 & a_1 \cdot \sin\theta_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1A_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_2 \cdot \cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & a_2 \cdot \sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^2A_3 = \begin{bmatrix} \cos\theta_3 & 0 & -\sin\theta_3 & a_3 \cdot \cos\theta_3 \\ \sin\theta_3 & 0 & \cos\theta_3 & a_3 \cdot \sin\theta_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^3A_4 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & \cos\theta_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^4A_5 = \begin{bmatrix} \cos\theta_5 & 0 & -\sin\theta_5 & 0 \\ \sin\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^5A_6 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ \sin\theta_6 & \cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3$$

$$= \begin{bmatrix} c_1 & 0 & -s_1 & a_1 \cdot c_1 & c_2 & -s_2 & 0 & a_2 \cdot c_2 & c_3 & 0 & -s_3 & a_3 \cdot c_3 \\ s_1 & 0 & c_1 & a_1 \cdot s_1 & s_2 & c_2 & 0 & a_2 \cdot s_2 & s_3 & 0 & c_3 & a_3 \cdot s_3 \\ 0 & -1 & 0 & d_1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1 & 0 & -s_1 & a_1 \cdot c_1 & c_2 \cdot c_3 - s_2 \cdot s_3 & 0 & -c_2 \cdot s_3 - s_2 \cdot c_3 & a_3 \cdot c_2 \cdot c_3 - a_3 \cdot s_2 \cdot s_3 + a_2 \cdot c_2 \\ s_1 & 0 & c_1 & a_1 \cdot s_1 & s_2 \cdot c_3 + c_2 \cdot s_3 & 0 & -s_2 \cdot s_3 + c_2 \cdot c_3 & a_3 \cdot s_2 \cdot c_3 - a_3 \cdot c_2 \cdot s_3 + a_2 \cdot s_2 \\ 0 & -1 & 0 & d_1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1(c_2 \cdot c_3 - s_2 \cdot s_3) & s_1 & -c_1(c_2 \cdot s_3 + s_2 \cdot c_3) & c_1(a_3 \cdot c_2 \cdot c_3 - a_3 \cdot s_2 \cdot s_3 + a_2 \cdot c_2 + a_1) \\ s_1(c_2 \cdot c_3 - s_2 \cdot s_3) & -c_1 & -s_1(c_2 \cdot s_3 + s_2 \cdot c_3) & s_1(a_3 \cdot c_2 \cdot c_3 - a_3 \cdot s_2 \cdot s_3 + a_2 \cdot c_2 + a_1) \\ -(s_2 \cdot c_3 + c_2 \cdot s_3) & 0 & s_2 \cdot s_3 - c_2 \cdot c_3 & -(a_3 \cdot s_2 \cdot c_3 + a_3 \cdot c_2 \cdot s_3 + a_2 \cdot s_2) + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
T_2 &= {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6 \\
&= \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & 0 & -s_5 & 0 \\ s_5 & 0 & c_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 \cdot c_6 & -c_5 \cdot s_6 & -s_5 & -d_6 \cdot s_5 \\ s_5 \cdot c_6 & -s_5 \cdot s_6 & c_5 & -d_6 \cdot c_5 \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_4 \cdot c_5 \cdot c_6 - s_4 \cdot s_6 & -(c_4 \cdot c_5 \cdot s_6 + s_4 \cdot c_6) & -c_4 \cdot s_5 & -d_6 \cdot c_4 \cdot s_5 \\ s_5 \cdot c_5 \cdot c_6 + c_4 \cdot s_6 & -c_5 \cdot s_4 \cdot s_6 + c_4 \cdot c_6 & -s_4 \cdot s_5 & -d_6 \cdot s_4 \cdot s_5 \\ s_5 \cdot c_6 & -s_5 \cdot s_6 & c_5 & d_6 \cdot c_5 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Figure 4.14: ABB Industrial Robot link co-ordinate transformation matrices

Specifically, for [i]=6, we obtain the T matrix, $T = {}^0A_6$, which specifies the position and orientation of the endpoint of the manipulator with respect to the base co-ordinate system. The final robot arm matrix T for an ABB robot manipulator is given below with the equations for each of the matrix structures. The calculations equation was utilized to calculate the joint solution for a given set of joint angles.

$$T = \begin{bmatrix} x_i & y_i & z_i & p_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

n = normal vector of the hand

s = sliding vector of the hand

a = approach vector of the hand

p = position vector of the hand

The direct kinematics solution of the six-link ABB robot manipulator is, simply a matter of calculating $T = {}^0A_6$, by chain multiplying the six ${}^{i-1}A_i$, matrices and evaluating each element of T matrix.

For an ABB1400 Industrial robot, arm matrix T is found as follows:

$$T = T_1 T_2 = {}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6$$

$$= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$\begin{aligned}
n_x &= c_1(c_2.c_3 - s_2.s_3)(c_4.c_5.c_6 - s_4.s_6) + s_1(s_4.c_5.c_6 + c_4.s_6) - c_1.s_5.c_6(c_2.s_3 + s_2.c_3) \\
n_y &= s_1(c_2.c_3 - s_2.s_3)(c_4.c_5.c_6 - s_4.s_6) - c_1(s_4.c_5.c_6 + c_4.s_6) - s_1.s_5.c_6(c_2.s_3 + s_2.c_3) \\
n_z &= -(s_2.c_3 + c_2.s_3)(c_4.c_5.c_6 - s_4.s_6) + s_5.c_6(s_2.s_3 - c_2.c_3) \\
s_x &= -c_1(c_2.c_3 - s_2.s_3)(c_4.c_5.s_6 + s_4.c_6) - s_1(s_4.c_5.s_6 - c_4.c_6) + c_1.s_5.s_6(c_2.s_3 + s_2.c_3) \\
s_y &= -s_1(c_2.c_3 - s_2.s_3)(c_4.c_5.s_6 + s_4.c_6) + c_1(s_4.c_5.s_6 - c_4.c_6) + s_1.s_5.s_6(c_2.s_3 + s_2.c_3) \\
s_z &= (s_2.c_3 + c_2.s_3)(c_4.c_5.s_6 + s_4.c_6) - s_5.s_6(s_2.s_3 - c_2.c_3) \\
a_x &= -c_1.c_4.s_5(c_2.c_3 - s_2.s_3) - s_1.s_4.s_5 - c_1.c_5(c_2.s_3 + s_2.c_3) \\
a_y &= -s_1.c_4.s_5(c_2.c_3 - s_2.s_3) + c_1.s_4.s_5 - s_1.c_5(c_2.s_3 + s_2.c_3) \\
a_z &= c_4.s_5(s_2.c_3 + c_2.s_3) + c_5(s_2.s_3 - c_2.c_3) \\
p_x &= -c_1.c_4.s_5.d_6(c_2.c_3 - s_2.s_3) - s_1.s_4.s_5.d_6 \\
&\quad - c_1(c_2.s_3 + s_2.c_3)(d_6.c_5 + d_4) + c_1(a_3.c_2.c_3 - a_3.s_2.s_3 + a_2.c_1.c_2 + a_1) \\
p_y &= -s_1.c_4.s_5.d_6(c_2.c_3 - s_2.s_3) + c_1.s_4.s_5.d_6 \\
&\quad - s_1(c_2.s_3 + s_2.c_3)(d_6.c_5 + d_4) + a_3.s_1.c_2.c_3 - a_3.s_1.s_2.s_3 + a_2.s_1.c_2 + a_1.s_1 \\
p_z &= c_4.s_5.d_6(s_2.c_3 + c_2.s_3) + (s_2.s_3 - c_2.c_3)(d_6.c_5 + d_4) - (a_3.s_2.c_3 + a_3.c_2.s_3 + a_2.s_2) + d_1
\end{aligned}$$

All calculations were implemented in the RAPID program generation engine for an industrial ABB IRB1400 manipulator, shown in section 4.4.4. The robot manipulator was positioned in specific locations and the robot controller TCP was correlated with the matrix calculation to verify the position accuracy [7].

4.4.3 Analytical computation of the inverse kinematic model

Computing the inverse kinematic position model (IKPM) of a robot arm is explained in terms of the basic *trigonometric method* for simple plane robot arms[27][28]. This approach has been utilized on the ABB IRB 1400 robot with respect to the plane joint axis.

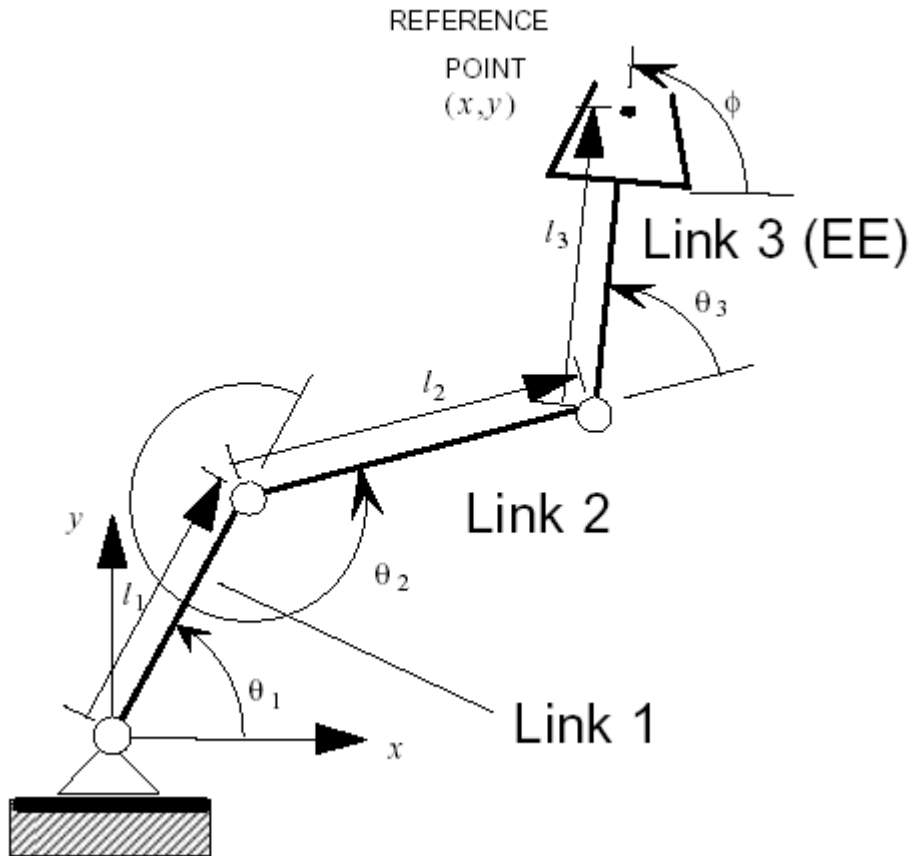


Figure 4.15: Planar 3-R Manipulator with the three reference joint angles

Transform joint co-ordinates to the end effector co-ordinates:

$$\begin{aligned}x &= l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\y &= l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ \phi &= (\theta_1 + \theta_2 + \theta_3)\end{aligned}$$

Solving nonlinear trigonometric equations using a 'atan2' equation. The following non-trigonometric equations steps, illustrate how joint manipulator angles can be calculated as follows:

STEP 1: The calculation of Angle1:

$$\begin{aligned}x - l_3 \cos \phi &= l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) \\y - l_3 \sin \phi &= l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2)\end{aligned}$$

$$\begin{aligned}x' &= x - l_3 \cos \phi, \\y' &= y - l_3 \sin \phi.\end{aligned}$$

STEP2:

$$\begin{aligned}(x' - l_1 \cos \theta_1) &= (l_2 \cos(\theta_1 + \theta_2)) \\(y' - l_1 \sin \theta_1) &= (l_2 \sin(\theta_1 + \theta_2))\end{aligned}$$

$$(-2l_1 x') \cos \theta_1 + (-2l_1 y') \sin \theta_1 + (x'^2 + y'^2 + l_1^2 - l_2^2) = 0$$

STEP3:

$$(-2l_1x')\cos\theta_1 + (-2l_1y')\sin\theta_1 + (x'^2 + y'^2 + l_1^2 - l_2^2) = 0$$

P Q R

$$P \cos \alpha + Q \sin \alpha + R = 0$$

$$\gamma = \text{atan2} \left(\frac{Q}{\sqrt{P^2 + Q^2}}, \frac{P}{\sqrt{P^2 + Q^2}} \right)$$

$$\theta_1 = \gamma + \sigma \cos^{-1} \left(\frac{-(x'^2 + y'^2 + l_1^2 - l_2^2)}{2l_1 \sqrt{x'^2 + y'^2}} \right)$$

STEP4: The calculation of Angle2:

$$\theta_2 = \text{atan2} \left(\frac{y' - l_1 \sin \theta_1}{l_2}, \frac{x' - l_1 \cos \theta_1}{l_2} \right) - \theta_1$$

STEP5: The calculation of Angle3:

$$\theta_3 = \phi - (\theta_1 + \theta_2)$$

4.4.4 Software Implementation of Robot Kinematics

During the operational setup of this research project all software implemented for robot kinematics was constructed using Microsoft Visual Basic 6, instead of Visual C. This was due to the limitations of the RobComm ActiveX components. A kinematic function was created, which managed and manipulated the robot co-ordinate frame. This was dependent on the axis orientation angles. This function provides the full functional calculation of direct kinematics as well as the quarternions value of (q1 – q4).

The *quarternion value* represents the rotational matrix of the tool co-ordinate system angle with respect to the robot base co-ordinate system. As seen in the previous section, the ABB robot positional structure is divided into the tool centre position (x, y, z) and the orientation of this TCP with respect to the quarternions (q1 – q4). This is the reason why the robot orientation is expressed in quarternion instead of axis angle position.

The *final arm matrix* ${}^R T_H$ is the relationship between the base and the tool as illustrated below.

$$\begin{aligned} {}^R T_H &= T_1 T_2 \\ &= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Rotational Matrix

$$= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix is structured in the following manner. Three 3D vectors, vectors n, s, a, correspond to the rotated x, y, z, axis with respect to the tool co-ordinate system and the robot base co-ordinate system.

The value of n_x will then be the x component of the x vector. The quaternion values can be calculated using this matrix value as illustrated by the following equation:

$$\begin{aligned} q_1 &= \frac{\sqrt{x_1 + y_2 + z_3 + 1}}{2} \\ q_2 &= \frac{\sqrt{x_1 - y_2 - z_3 + 1}}{2} & \text{sign } q_2 &= \text{sign } (y_3 - z_2) \\ q_3 &= \frac{\sqrt{y_2 - x_1 - z_3 + 1}}{2} & \text{sign } q_3 &= \text{sign } (z_1 - x_3) \\ q_4 &= \frac{\sqrt{z_3 - x_1 - y_2 + 1}}{2} & \text{sign } q_4 &= \text{sign } (x_2 - y_1) \end{aligned}$$

Figure 4.16 illustrates the ABB robot remote kinematic programming environment, which provides a more flexible approach to remote RAPID robot programming environment.

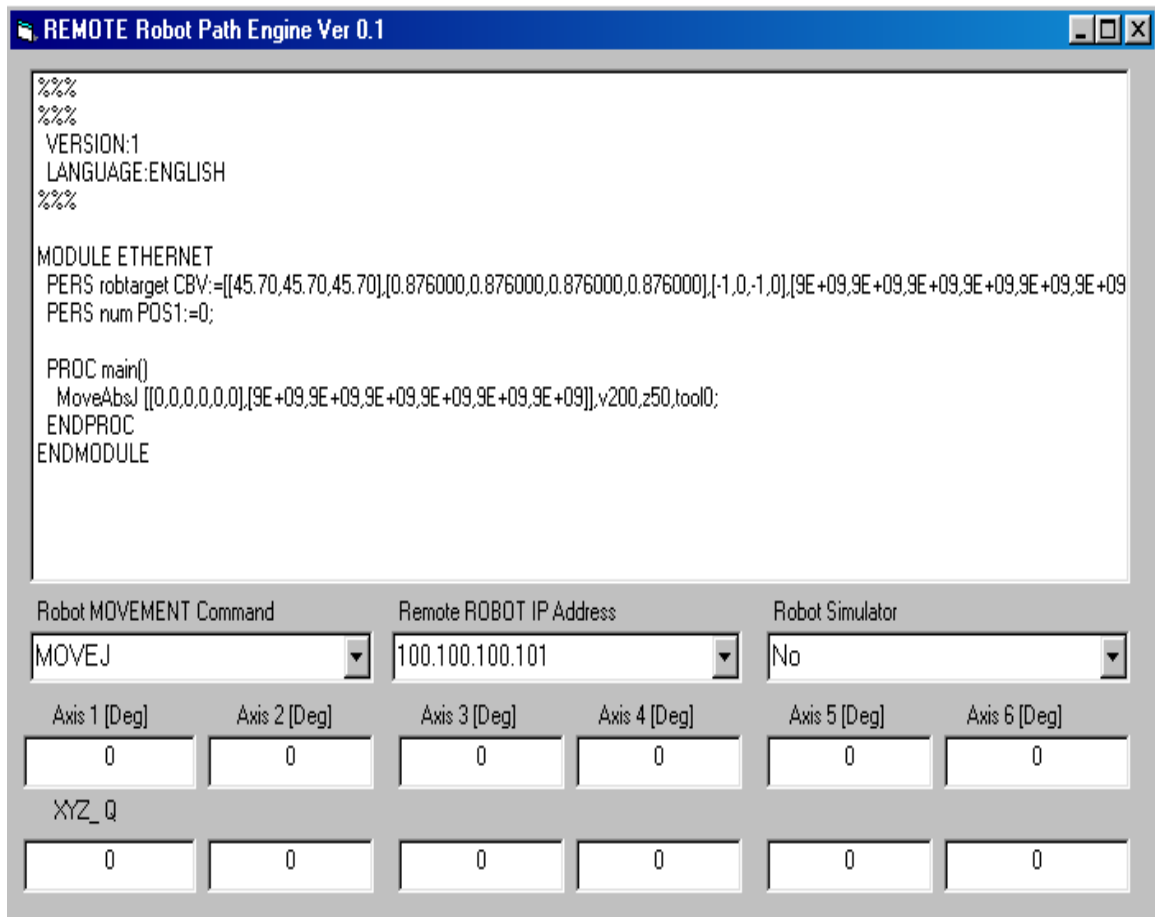


Figure 4.16: Robot Path Engine environment

The Visual Basic source code for the direct kinematics computation is illustrated below which is used to manipulate the robot TCP and orientation:

```

//*****
// Convert the Angle Degree value to RAD value for computation requirements
A1 = AD1 * (PI / 180)
A2 = AD2 * (PI / 180)
A3 = AD3 * (PI / 180)
A4 = AD4 * (PI / 180)
A5 = AD5 * (PI / 180)
A6 = AD6 * (PI / 180)

ROBOT_ANGLE1 = AD1
ROBOT_ANGLE2 = AD2
ROBOT_ANGLE3 = AD3

```

```

ROBOT_ANGLE4 = AD4
ROBOT_ANGLE5 = AD5
ROBOT_ANGLE6 = AD6

```

```
//Link parameters
```

```

ROBOT_OFFSET_ANGLE1 = Val(AO1) // 0
ROBOT_OFFSET_ANGLE2 = Val(AO2) // -90
ROBOT_OFFSET_ANGLE3 = Val(AO3) // 0
ROBOT_OFFSET_ANGLE4 = Val(AO4) // 0
ROBOT_OFFSET_ANGLE5 = Val(AO5) // 0
ROBOT_OFFSET_ANGLE6 = Val(AO6) // 180

```

```
//Correct angles for correct robot operations
```

```

SN1 = Sin((ROBOT_ANGLE1 + ROBOT_OFFSET_ANGLE1) * PI / 180)
CS1 = Cos((ROBOT_ANGLE1 + ROBOT_OFFSET_ANGLE1) * PI / 180)

```

```

SN2 = Sin((ROBOT_ANGLE2 + ROBOT_OFFSET_ANGLE2) * PI / 180)
CS2 = Cos((ROBOT_ANGLE2 + ROBOT_OFFSET_ANGLE2) * PI / 180)

```

```

SN3 = Sin((ROBOT_ANGLE3 - ROBOT_ANGLE2 + ROBOT_OFFSET_ANGLE3)
* PI / 180)
CS3 = Cos((ROBOT_ANGLE3 - ROBOT_ANGLE2 +
ROBOT_OFFSET_ANGLE3) * PI / 180)

```

```

SN4 = Sin((ROBOT_ANGLE4 + ROBOT_OFFSET_ANGLE4) * PI / 180)
CS4 = Cos((ROBOT_ANGLE4 + ROBOT_OFFSET_ANGLE4) * PI / 180)

```

```

SN5 = Sin((ROBOT_ANGLE5 + ROBOT_OFFSET_ANGLE5) * PI / 180)
CS5 = Cos((ROBOT_ANGLE5 + ROBOT_OFFSET_ANGLE5) * PI / 180)

```

```

SN6 = Sin((ROBOT_ANGLE6 + ROBOT_OFFSET_ANGLE) * PI / 180)
CS6 = Cos((ROBOT_ANGLE6 + ROBOT_OFFSET_ANGLE) * PI / 180)

```

```
//Calculated arm matrix values
```

```

Nx = CS1 * (CS2 * CS3 - SN2 * SN3) * (CS4 * CS5 * CS6 - SN4 * SN6) + SN1 *
(SN4 * CS5 * CS6 + CS4 * SN6) - CS1 * SN5 * CS6 * (CS2 * SN3 + SN2 * CS3)

```

```
Nxx = Val(Nx)
```

```

Ny = SN1 * (CS2 * CS3 - SN2 * SN3) * (CS4 * CS5 * CS6 - SN4 * SN6) - CS1 *
(SN4 * CS5 * CS6 + CS4 * SN6) - SN1 * SN5 * CS6 * (CS2 * SN3 + SN2 * CS3)

```

```
Nyy = Val(Ny)
```

```

Nz = -(SN2 * CS3 + CS2 * SN3) * (CS4 * CS5 * CS6 - SN4 * SN6) + SN5 * CS6 *
(SN2 * SN3 - CS2 * CS3)

```

```
Nzz = Val(Nz)
```

```

Sx = -CS1 * (CS2 * CS3 - SN2 * SN3) * (CS4 * CS5 * SN6 + SN4 * CS6) - SN1 *
(SN4 * CS5 * SN6 - CS4 * CS6) + CS1 * SN5 * SN6 * (CS2 * SN3 + SN2 * CS3)

```

```
Sxx = Val(Sx)
```

```

Sy = -SN1 * (CS2 * CS3 - SN2 * SN3) * (CS4 * CS5 * SN6 + SN4 * CS6) - CS1 *
(SN4 * CS5 * SN6 - CS4 * CS6) + SN1 * SN5 * SN6 * (CS2 * SN3 + SN2 * CS3)
Syy = Val(Sy)
Sz = (SN2 * CS3 + CS2 * SN3) * (CS4 * CS5 * SN6 + SN4 * CS6) - SN5 * SN6 *
(SN2 * SN3 - CS2 * CS3)
Szz = Val(Sz)
Ax = -CS1 * CS4 * SN5 * (CS2 * CS3 - SN2 * SN3) - SN1 * SN4 * SN5 - CS1 *
CS5 * (CS2 * SN3 + SN2 * CS3)
Axx = Val(Ax)
Ay = -SN1 * CS4 * SN5 * (CS2 * CS3 - SN2 * SN3) - CS1 * SN4 * SN5 - SN1 *
CS5 * (CS2 * SN3 + SN2 * CS3)
Ayy = Val(Ay)
Az = CS4 * SN5 * (SN2 * CS3 + CS2 * SN3) + CS5 * (SN2 * SN3 - CS2 * CS3)
Azz = Val(Az)

Px = (-CS1 * CS4 * SN5 * d6 * (CS2 * CS3 - SN2 * SN3)) - (SN1 * SN4 * SN5 *
d6) - (CS1 * (CS2 * SN3 + SN2 * CS3) * (d6 * CS5 + d4)) + (CS1 * (aa3 * CS2 *
CS3 - aa3 * SN2 * SN3 + aa2 * CS1 * CS2 + aa1))
Py = -SN1 * CS4 * SN5 * d6 * (CS2 * CS3 - SN2 * SN3) + CS1 * SN4 * SN5 * d6 -
SN1 * (CS2 * SN3 + SN2 * CS3) * (d6 * CS5 + d4) + aa3 * SN1 * SN2 * SN3 - aa3
* SN1 * SN2 * SN3 + aa2 * SN1 * CS2 + aa1 * aa1 * SN1
Pz = CS4 * SN5 * d6 * (SN2 * CS3 + CS2 * SN3) + (SN2 * SN3 - CS2 * CS3) * (d6
* CS5 + d4) - (aa3 * SN2 * CS3 + aa3 * CS2 * SN3 + aa2 * SN2) + d1

//Calculations of Quaternion values of a direct kinematics
q1 = (Sqr(Nxx + Syy + Azz + 1) / 2)
q2 = (Sqr(Nxx - Syy - Azz + 1) / 2)
q3 = (Sqr(Syy - Nxx - Azz + 1) / 2)
q4 = (Sqr(Azz - Nxx - Syy + 1) / 2)
'q1 = Sqr(1.75) / 2

//*****

```

4.5 Software Modules for robot motion control

It was discovered during the research that programming a robot with a teach pendant was very time consuming. A Visual Basic programming platform was developed to minimize the development time. In future applications, key functions could be utilized to demonstrate the remote programming capability.

Figure 4.17 illustrates the remote RAPID program architecture for motion control.

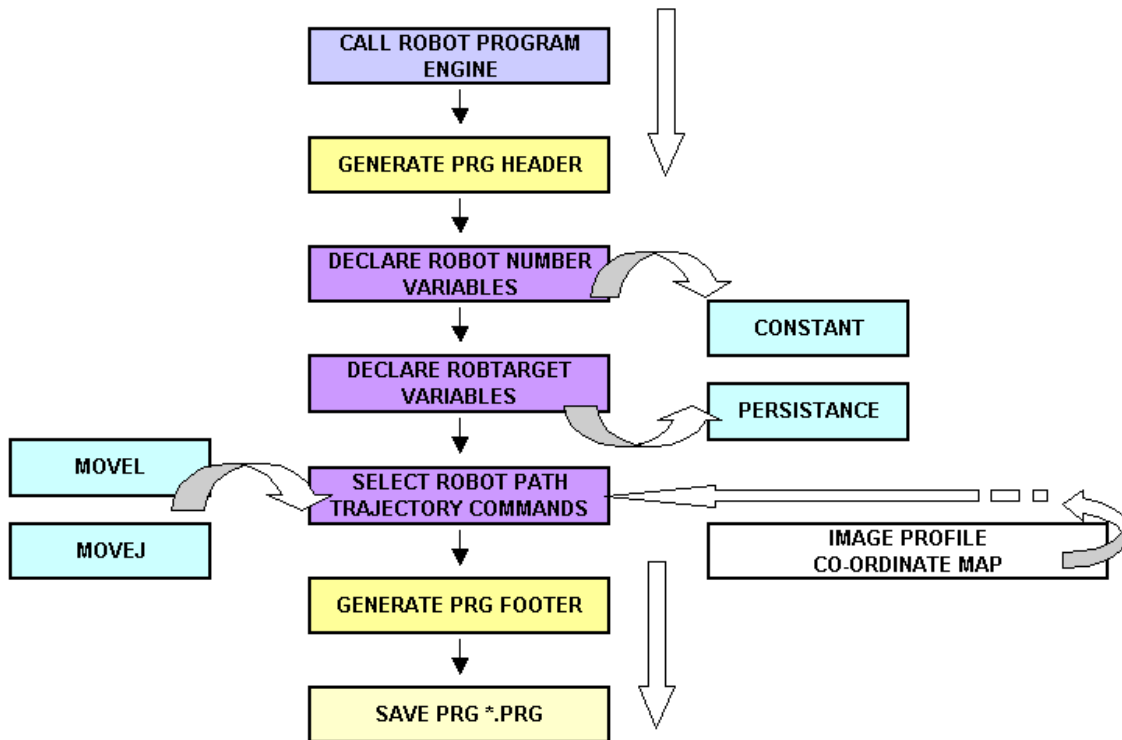


Figure 4.17: Program Modules and architecture for motion control

➤ **Program Header**

To ensure that the ABB robot controller will respond correctly to the automatically generated robot program, it is critical the correct program structure is generated via the remote robot program engine as illustrated in Figure 4.17. The following source code below displays the source code required to generate the ABB robot path trajectory program.

The “RobotPrgMAINHeader function will structure the robot program with the correct file header and footer.


```

//*****
// Robot Program HEADER
Sub RobotPrgMAINHeader(FileName_X As String)
Dim FileName As String
Dim FileString As String

FileName = FileName_X
'Robot File *.prg
    Open "c:\\" + FileName + ".prg" For Append As #1 ' Open file for output.
    FileString = ""
    Print #1, FileString
    List1.AddItem FileString
    FileString = " PROC main()"
    Print #1, FileString
    List1.AddItem FileString
Close #1 ' Close file.
End Sub
//*****

```

➤ **Program Footer**

```

//*****
Sub RobotPrgFooter(FileName_X As String, PRG_Name As String)
Dim FileName As String
Dim FileString As String
FileName = FileName_X
'Robot File *.prg
    Open "c:\\" + FileName + ".prg" For Append As #1 ' Open file for output.
    FileString = "ENDMODULE"
    List1.AddItem FileString
    Print #1, FileString
Close #1 ' Close file.
End Sub
//*****

```

➤ **Robot_RobTarget Variable declaration**

Once the user has generated the robot trajectory header structure, manipulator movement or Number variables can be generated via the Robot_robtargetVar and Creat_Robot_NUMVar function. The type structure items can be requested via the software function as discussed above. The software source code is illustrated below:

```

//*****
Sub Creat_Robot_robtargetVar(FileName_X As String, VAR_Type As String,
VAR_Name As String, VAR_Name_ArrayCNT As String, x As Variant, y As
Variant, z As Variant, q1 As Variant, q2 As Variant, q3 As Variant, q4 As
Variant)
' format
' PERS robtarget START_POS:=[[1054.66,-320.13,571.33],[0.002583,-
0.053022,0.998589,0.001637],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

Dim FileName As String
'config var format
x = Format(x, "0.00")
y = Format(y, "0.00")
z = Format(z, "0.00")

q1 = Format(q1, "0.000000")
q2 = Format(q2, "0.000000")
q3 = Format(q3, "0.000000")
q4 = Format(q4, "0.000000")

FileName = FileName_X
'Robot File *.prg
Open "c:\\" + FileName + ".prg" For Append As #1 ' Open file for output.
FileString = " " + VAR_Type + " robtarget " + VAR_Name + ":[[" + x + "," + x
+ "," + z + "],[[" + q1 + "," + q2 + "," + q3 + "," + q4 + "],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];"
List1.AddItem FileString
Print #1, FileString

Close #1 ' Close file.

End Sub
//*****

```

➤ **Robot_ Number Variable declaration**

```
//*****  
Sub Creat_Robot_NUMVar(FileName_X As String, VAR_Type As String,  
VAR_Name As String, VAR_Name_ArrayCNT As String, NUM As Variant)  
' format  
' PERS num Z_POS8:=0;  
  
Dim FileName As String  
'config var format  
NUM = Format(NUM, "0")  
  
FileName = FileName_X  
'Robot File *.prg  
Open "c:\\" + FileName + ".prg" For Append As #1 ' Open file for output.  
FileString = " " + VAR_Type + " num " + VAR_Name + " :=" + NUM + ";"  
Print #1, FileString  
List1.AddItem FileString  
Close #1 ' Close file.  
  
End Sub  
//*****
```

➤ **Robot MOVEMENT Command Function**

Robot MOVEMENT commands can be generated via the Create_InstructionMOVE function. The type structure items can be requested via the software function as discussed above. The software source code is illustrated below:

```
//*****  
Sub Create_InstructionMOVE(FileName_X As String, INSTRU As String,  
Pos_Var As String, SPEED As String, ACC As String, x As Variant, y As  
Variant, z As Variant, q1 As Variant, q2 As Variant, q3 As Variant, q4 As  
Variant, cf1 As Variant, cf4 As Variant, cf8 As Variant, cfx As Variant)  
  
Dim FileName As String  
Dim FileString As String  
Dim INST As String  
'ENDMODULE  
' MoveJ PRP1,v150,z50,tool0;
```

```

INST = INSTRU

FileName = FileName_X
'Robot File *.prg
Open "c:\\" + FileName + ".prg" For Append As #1 ' Open file for output.

If (INST = "MOVEJ") Then
  FileString = "  MoveJ " + Pos_Var + "," + SPEED + "," + ACC + ",tool0;"
End If
If (INST = "MOVEL") Then
  FileString = "  MoveL " + Pos_Var + "," + SPEED + "," + ACC + ",tool0;"
End If

Print #1, FileString
Close #1 ' Close file.

End Sub
//*****

```

➤ **RAPID Static Position Command Function**

Robot Static sub-routines are generated via the HOME_POS function. The type structure items can be requested via the software function as discussed above. The software source code is illustrated below:

```

//*****
Sub HOME_POS(FileName_X As String)
'  MoveAbsJ
[[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,z50,tool0;

Dim FileName As String
Dim FileString As String
FileName = FileName_X
'Robot File *.prg
Open "c:\\" + FileName + ".prg" For Append As #1 ' Open file for output.
FileString = "  MoveAbsJ
[[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,z50,tool0;"
Print #1, FileString
List1.AddItem FileString
Close #1 ' Close file.

End Sub
//*****

```

4.6 OFF-LINE Development Environment to program robot

Figure 4.18 illustrates the ABB robot remote programming environment, which provides a more flexible approach to remote RAPID robot programming environment.

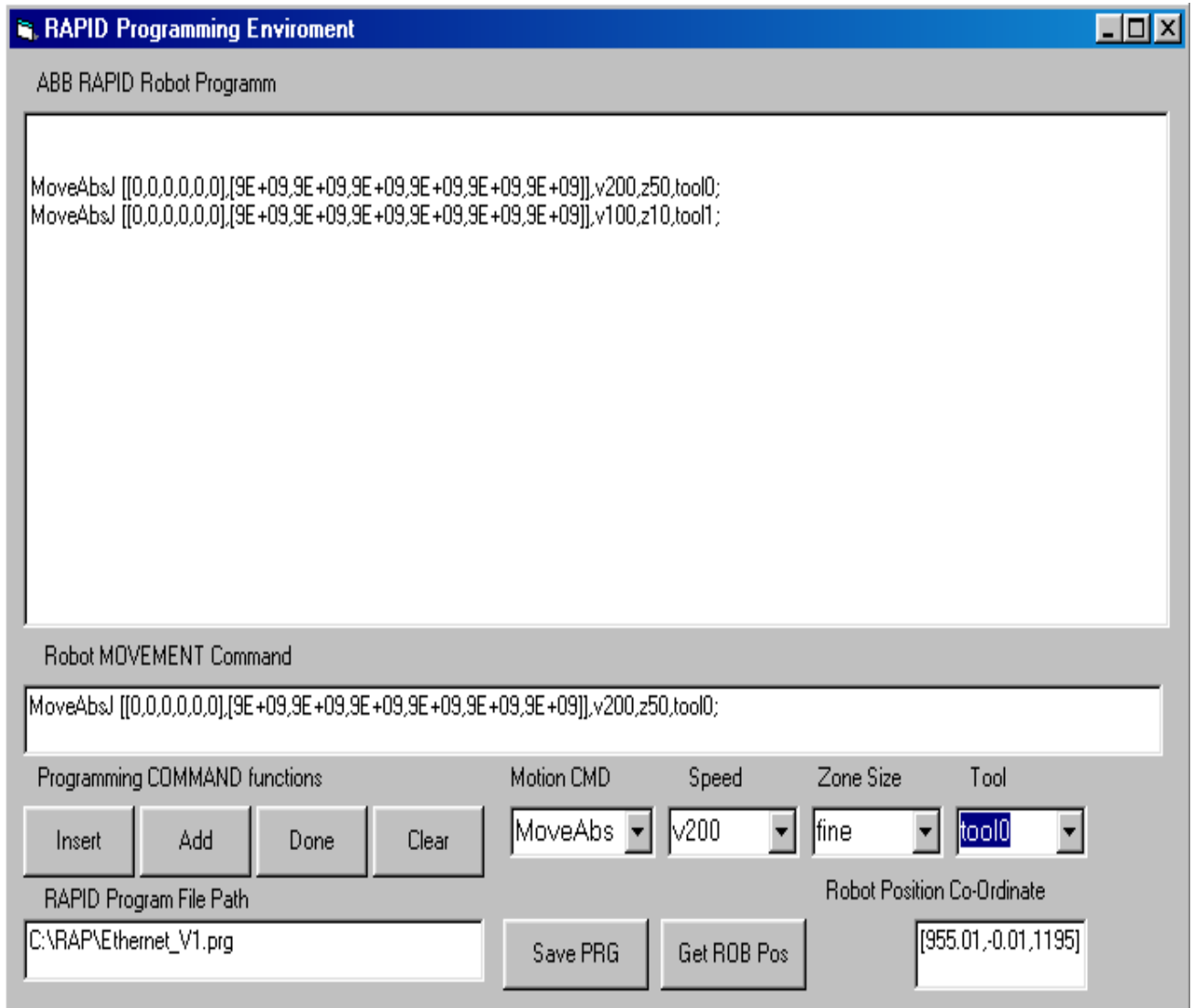


Figure 4.18: Remote RAPID programming environment

The ABB RAPID programming environment consists of the following components functionality:

➤ Motion CMD ComboBOX

This provides a selection of motion instructions available for the ABB robot controller such as, MOVEJ, MOVEL, MOVEABSJ.

➤ GetROB_Pos Command Button

GetROB_Pos Button allows the user to get the current robot manipulator coordinate position, which can be utilized with the development of a new RAPID robot program. This is a useful tool to reduce RAPID robot programming time. Should the base program variables be configured correctly, certain variables may be updated on the fly. The GetRobPos position source code is illustrated below.

```
Sub GetRobPos_click(Index As Integer)
'This button updates the robot position fields
Dim status As Integer 'return status variable

Dim robpos As S4RobPosData ' dimension robot position object
' create the object
Set robpos = CreateObject("S4RobPosData")
' get the position from the robot
status = robotHelper(Index).S4CurrentPositionGet(robpos)
If status <> 0 Then ' did the function work?
MsgBox "status = " + Str(status) ' no,
Else ' yes, it worked
lblPosDatX(Index) = Format$(robpos.x)
lblPosDatY(Index) = Format$(robpos.y)
lblPosDatZ(Index) = Format$(robpos.z)
lblPosDatQ1(Index) = Format$(robpos.q1)
lblPosDatQ2(Index) = Format$(robpos.q2)
lblPosDatQ3(Index) = Format$(robpos.q3)
lblPosDatQ4(Index) = Format$(robpos.q4)
lblPosDatTool(Index) = Format$(robpos.ToolObj)
```

```
lblPosDatWobj(Index) = Format$(robpos.WObj)
lblPosDatEx1(Index) = Format$(robpos.eaxA)
lblPosDatEx2(Index) = Format$(robpos.eaxB)
lblPosDatEx3(Index) = Format$(robpos.eaxC)
lblPosDatEx4(Index) = Format$(robpos.eaxD)
lblPosDatEx5(Index) = Format$(robpos.eaxE)
End If

End Sub
```

➤ Speed ComboBOX

This provides a selection of speed setting available for the ABB robot controller move instruction set such as, v100, v200, v500 ms.

➤ Zone Size ComboBOX

This provides a selection of Zone Size setting available for the ABB robot controller move instruction set such as, fine, 10, 20, etc.

➤ Tool ComboBOX

This provides a selection of Tool setting available for the ABB robot controller move instruction set such as, 0, 1, etc.

➤ Add Command Button

Once all the motion instruction variables have been selected, the add command button will update the program window with the correct MOVE instruction which has been configured.

- Save PRG Command Button, this will save all the robot motion sequences to RAPID program file with the correct file format that has been illustrated in Chapter 2.

4.7 Communication Control

While developing the system application, one of the main objectives was to establish the amount of flexibility the remote server would allow the remote user to access the robot controller functionality and at what possible update rate. This would establish if it was possible to create a real-time remote communication interface application.

It was established that in order to control the position of the robot, there were two possible areas of analysis.

1. This first approach was to generate a complete RAPID program on the PC after which this was uploaded via the remote Ethernet link. Thus for every new position change the robot requires the system to regenerate an entirely new RAPID program structure. The new program must then be uploaded to the robot. A large amount of processing time was wasted uploading information to the robot.

This method certainly does not create a true real-time system and should only be utilized in areas where there is low volume production.

2. The second approach was to create a base RAPID program with persistent variables. These variables are utilized for robot target positions. The persistent variables can be updated during the auto-processing cycle on the fly. This

provides the robot with the capability of real-time position versatility within the robot work co-ordinate frame.

To ensure that the remote Ethernet programming environment can be achieve a stable and reliable communication link had to be established a verified. The communication link was establish as verified via windows DOS command , the

“PING IP Address” command.

IP Address “ 100.100.100.102 “ as follows :

```
C:\ping 100.100.100.102  
Ping 100.100.100.102 with 32 bytes of data:
```

REMOTE device REPIES with a the following system packet information, if available :

```
Reply from 100.100.100.102: bytes=32 time<10ms TTL=128  
Reply from 100.100.100.102: bytes=32 time<10ms TTL=128  
Reply from 100.100.100.102: bytes=32 time<10ms TTL=128  
Reply from 100.100.100.102: bytes=32 time<10ms TTL=128
```

Ping statistics from 100.100.100.102:

```
    Packets: Sent = 4, Received = 4 , Lost = 0 ( 0% loss) ,  
Approximate round trip times in milli-seconds:  
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

RobComm Server Configuration

Once the Ethernet network has been correctly established as demonstrated above, the RobComm server can be configured to provide the vital link between the Bridge PC and the Robot Controller. Figure 4.19 illustrates the RobComm Ethernet server setup environment. In this environment an Alias Name is established such as 'S4' which provides the link for the RobComm ActiveX. With this Alias Name, a network address has to be manually configured which is mapped to the network setting of the robot controller. The Network Address was configured as follows: TCP/IP Node Name 100.100.100.101.

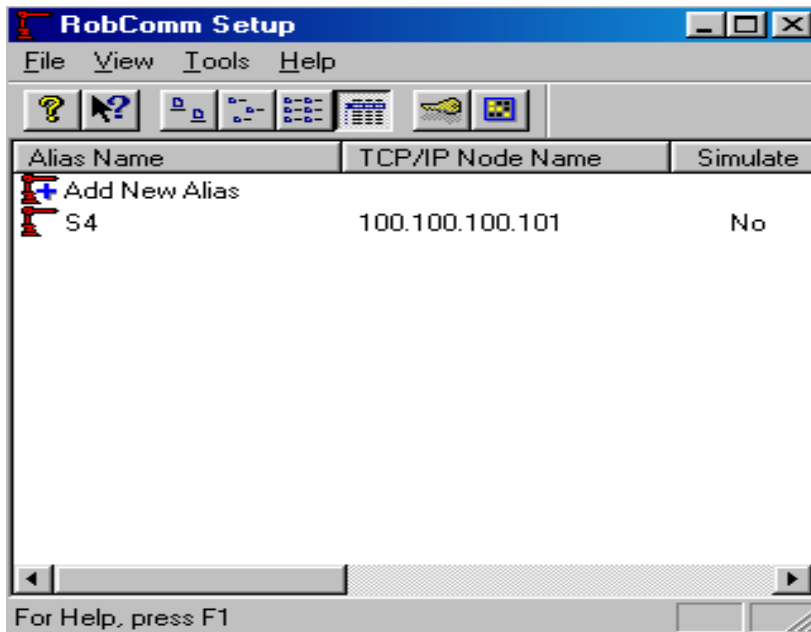
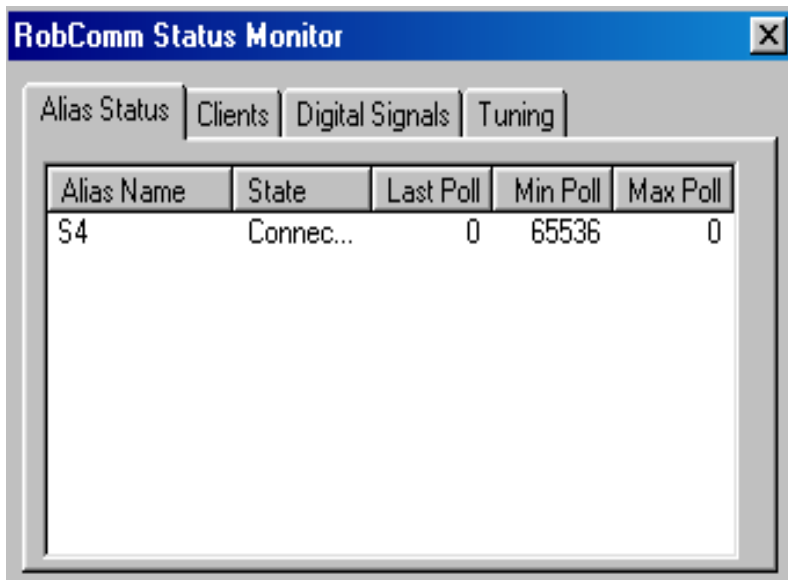


Figure 4.19: RobComm Ethernet communication setup “ IP:100.100.100.101”

RobComm Status Monitor

The RobComm server provides a monitoring status window that allows the user to confirm that the server has correctly established a connection with the robot controller. Figure 4.20 Illustrates the RobComm connectivity environment server, to provide the user with communications activity status.



| Alias Name | State | Last Poll | Min Poll | Max Poll |
|------------|-----------|-----------|----------|----------|
| S4 | Connec... | 0 | 65536 | 0 |

Figure 4.20: RobComm Active Server establishing communication with ABB Robot Controller

4.8 Software Modules to Initiate Motion Control

This user environment provides the trajectory path manipulation and control mechanism software. The robot system integration is handled via the RobComm ActiveX components and the software algorithms, which are illustrated below.

Figure 4.21 illustrates the ABB robot remote trajectory application, which provides an automated flexible approach to a remote manipulation environment.

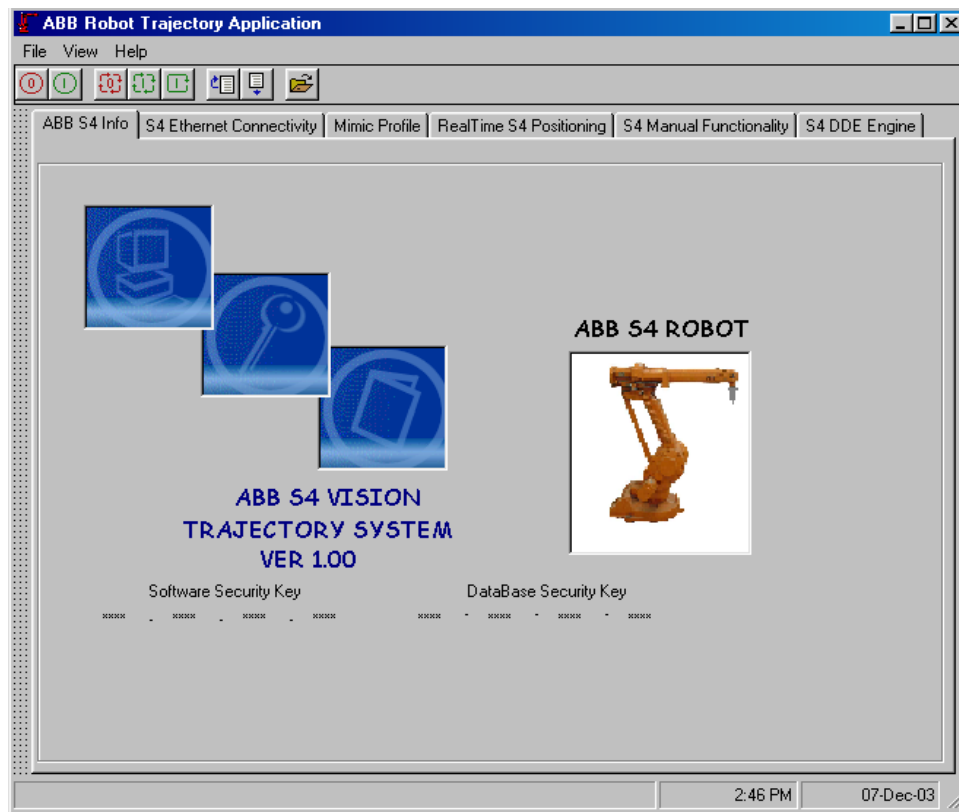


Figure 4.21: ABB Robot trajectory Motion Control

RobComm Object Configuration

In order to ensure that the RobComm ActiveX components are initialized correctly the following software object has to be initialized and a software alias created as follows:

```
//*****  
//Create an RobotHelper Object  
robotHelper(0).Robot = "S4" 'change this to your robot alias  
setUpRobots  
//*****
```

Manual robot motions commands

This user environment provides the manual functionality to manipulation and control to robot manipulator to pre-define co-ordinate positions. Figure 4.22 illustrates the ABB robot remote manual motion environment for the dynamic and static RAPID sub-routines which were developed for research testing purposes.

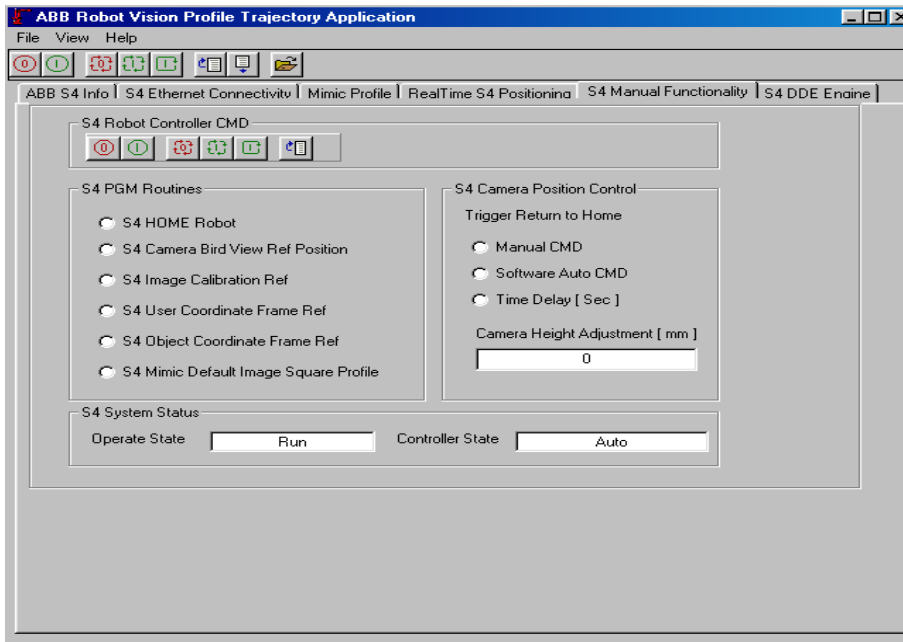


Figure 4.22: Software environment for manual robot control commands

The following visual basic source code below demonstrates the manual system functionality required to servo the robot manipulator to pre-defined coordinate positions, this was utilized while testing the remote communication link. This code provides the control mechanism platform to automated robot trajectory path system.

```

//*****
Private Sub ManRoutines1_Click(Index As Integer)

Dim status As Integer 'return status variable
Dim i As Integer
Dim resultid As Long

'S4 MAUNAL ROUTINE FUNCTIONS
Select Case Index:
Case 0: 'HOME ROBOT
Call MANUAL_S4_HOME
ManRoutines1(0).value = False
Case 1: 'CAMERA BIRD VIEW POSITION
Call MANUAL_S4_CBV
ManRoutines1(1).value = False
Case 2: 'IMAGE CALIBRATION REF LINE
Call MANUAL_S4_CRL
ManRoutines1(2).value = False
Case 3: 'USER CO-ORDINATE FRAME REF
Call MANUAL_S4_UFC
ManRoutines1(3).value = False
Case 4: 'OBJECT CO-ORDINATE REF FRAME
Call MANUAL_S4_OFC
ManRoutines1(4).value = False
Case 5: 'MIMIC DEFAULT CO-ORDINATE
Call MANUAL_S4_MIMIC
ManRoutines1(5).value = False
Case 6: 'MIMIC DEFAULT CO-ORDINATE
ManRoutines1(6).value = False

End Select

End Sub
//*****

```

The OptionButton, “ManRoutines1(x)” triggers the manual sub-routine via the appropriate INDEX_MARKER value, this will call the correct robot controller sub-routine that has been pre-configured in the RAPID robot program. Figure 4.23 illustrates when the HOME_ROBOT manual function has been selected. This will cause the INDEX_MARKER value to equal 1. The “S4ProgramNumVarWrite” command will request the robot controller RAPID program variable to CALL the HOME_ROBOT sub-routine which will in turn servo the robot manipulator to the HOME position. The robot HOME position forces all the arm linkages to a zero degree state.

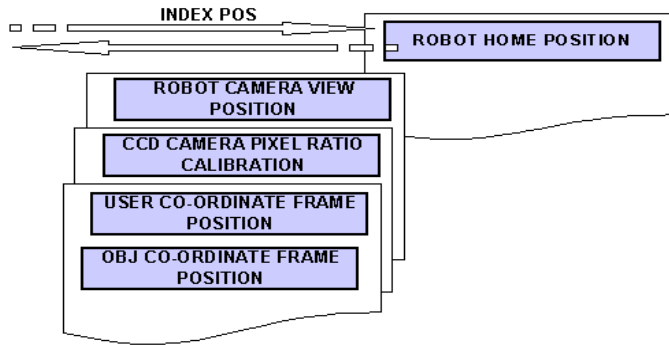


Figure 4.23: RAPID Sub-Routine function

```
//*****
```

```
Sub MANUAL_S4_HOME()
Dim S4_VarNum_Name As String
Dim S4_VarNum As Single
Dim S4_VarNumResult As Integer
Dim ResultSpec As Integer
Dim resultid As Long

'iINDEX MARKER = 1
S4_VarNum_Name = "INDEX_MARKER"
S4_VarNum = 1
```

```
S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,  
S4_VarNum, ResultSpec, resultid)
```

```
End Sub
```

```
//*****
```

Object Vision Manual Position control

In most industrial applications the robot vision system is located in a fixed viewing position, which limits the object to a specific viewing position. In this research environment the CCD camera has been attached to the end of the robot arm, providing the system with additional viewing flexibility. This means, that a number of objects can be viewing in different positions within the robot workcell. For this application only one viewing position has been utilized, a pre-defined position has been programmed, the 'z' co-ordinate can be manipulated via the remote application. This 'z' co-ordinate displacement, references back to the camera viewing height. This software sub-routine is triggered in the same manner as the HOME_ROBOT sub routine, the only difference is that the software has a CAPTURE software flag that has been added to provide a viewing wait time period which synchronizes the robot controller from return to the HOME position before the image has been captured correctly. These software algorithms provide the control mechanism platform for automatic robot control. Software source code has been illustrated below.

```
//*****
```

```
Sub MANUAL_S4_CBV()
```

```
'CAPTURE FLAG = 0  
S4_VarNum_Name = "CAPTURE"
```



```
S4_VarNum = 0
```

```
S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,  
S4_VarNum, ResultSpec, resultid)
```

```
'iINDEX MARKER = 2
```

```
S4_VarNum_Name = "INDEX_MARKER"
```

```
S4_VarNum = 2
```

```
S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,  
S4_VarNum, ResultSpec, resultid)
```

```
If Trig1(1).value = True Then  
    CAPTURE_TIMER1.Enabled = True  
End If
```

```
End Sub
```

```
//*****
```

Robot Trajectory Co-ordinate Position control

The following visual basic source code below is the vital component that handles the download routine for the object profile co-ordinate map to the relevant persistent robot variable which will be utilized to track the captured object profile. Figure 4.24 illustrates the software environment for object profile tracking.

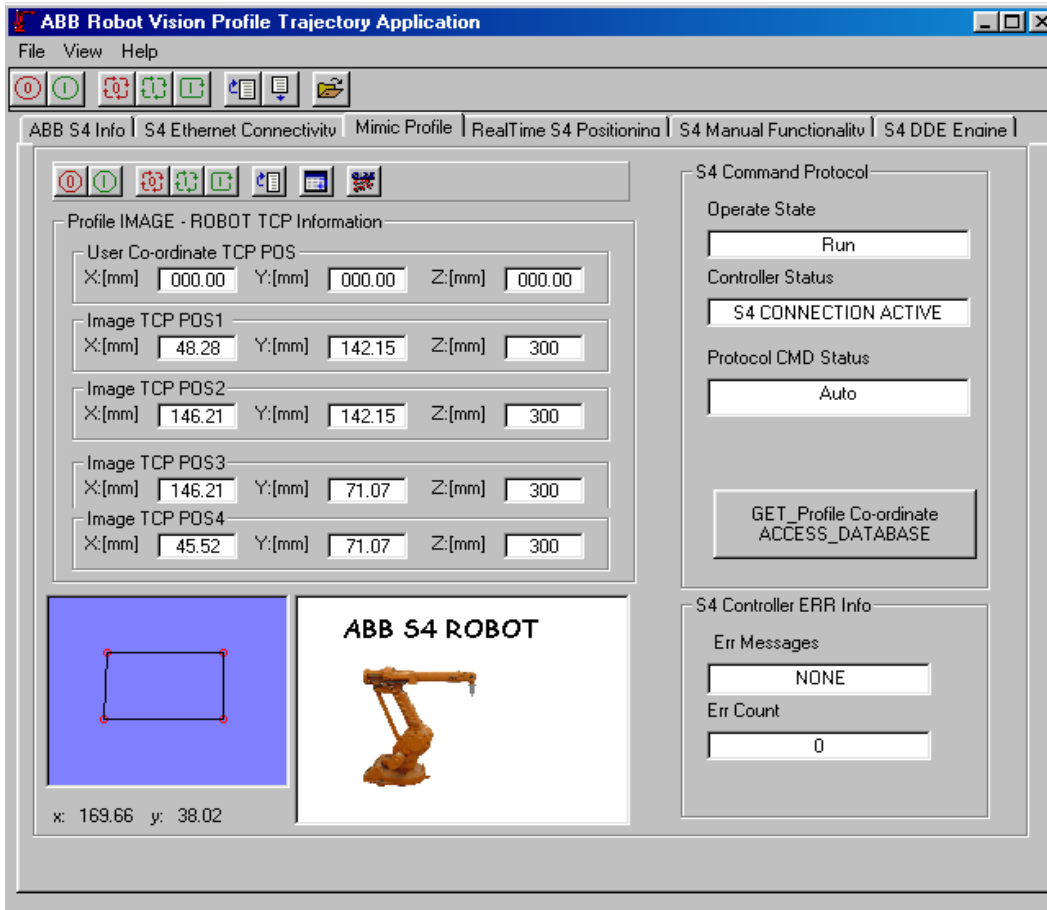


Figure 4.24: Software environment for object profile tracking

The download routine is synchronized via the data handshake routine that is discussed in later Chapters. Software source code which manages the robot trajectory position download transaction mechanism, has been illustrated below.

```
//*****
Sub S4_PERS_VARIABLE_DOWNLOAD()

Dim S4RobotAlias As String
Dim S4_VarNum_Name As String
Dim S4_VarNum As Single
Dim S4_VarNumResult As Integer
Dim ResultSpec As Integer
Dim resultid As Long
Dim CNT As Integer
```

ROB_INDEX_MARKER = 0

S4RobotAlias = "S4"

S4_VarNum_Name = "INDEX_MARKER"

'GET CURRENT INDEX MARKER VALUE

S4_VarNumResult = Helper3.S4ProgramNumVarRead(S4_VarNum_Name, 0,
S4_VarNum)

'IF INDEX MARKER <> 0 FORCE = 0

If S4_VarNumResult <> 0 Then

 S4_VarNum = 0

 S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,
S4_VarNum, ResultSpec, resultid)

End If

'HOME ROBOT CMD INDEX_MARKER = 1

S4_VarNum_Name = "INDEX_MARKER"

S4_VarNum = 1

S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,
S4_VarNum, ResultSpec, resultid)

'MIMIC OBJECT PROFILE CMD INDEX_MARKER = 4

'DOWNLOAD PROFILE -> CO-ORDINATE [X,Y,Z ,Q1,Q2,Q3,Q4] TO S4 ROBOT

'X_POSITION PROFILE REF

CNT = 1

For CNT = 1 To 4

 S4_VarNum_Name = "X_POS" + Trim(Str(CNT))

 S4_VarNum = X_POS(CNT)

 S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,
S4_VarNum, ResultSpec, resultid)

Next CNT

'Y_POSITION PROFILE REF

CNT = 1

For CNT = 1 To 4

 S4_VarNum_Name = "Y_POS" + Trim(Str(CNT))

 S4_VarNum = Y_POS(CNT)

 S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,
S4_VarNum, ResultSpec, resultid)

Next CNT

'Z_POSITION PROFILE REF

CNT = 1

```

For CNT = 1 To 4
  S4_VarNum_Name = "Y_POS" + Trim(Str(CNT))
  S4_VarNum = Z_POS(CNT)
  S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,
S4_VarNum, ResultSpec, resultid)
Next CNT

```

```

S4_VarNum_Name = "INDEX_MARKER"
S4_VarNum = 1
S4_VarNumResult = Helper3.S4ProgramNumVarWrite(S4_VarNum_Name, 0,
S4_VarNum, ResultSpec, resultid)

```

End Sub

```

//*****

```

Robot Controller Hardware Control Mechanism

Figure 4.25 illustrates the remote control tool bar mechanism utilized for the ABB robot controller to initiate hardware/software initialization sequences, eg START MOTOR, STOP MOTOR, RUN PROGRAM, LOAD PROGRAM etc.

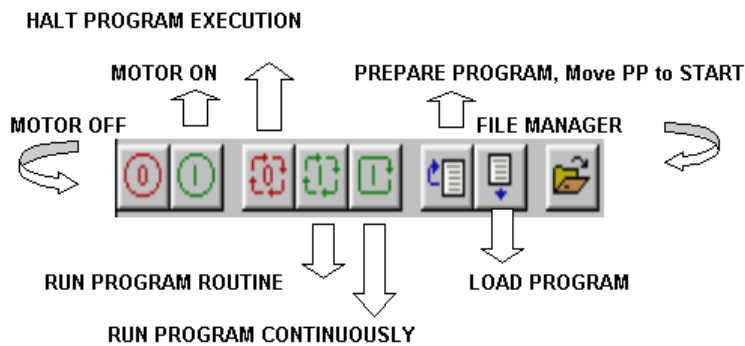


Figure 4.25: Software - Robot control tool bar

The following visual basic source code below will become a vital link for the automated robot control, this code provides the control mechanism to control the robot controller via the Ethernet serial link, establish and manipulate control status of the robot controller.

```

//*****
Dim status As Integer 'return status variable
Dim i As Integer
Dim resultid As Long

For i = 0 To (numRobots - 1)
  Select Case button.Index:
    Case 1: 'motors off button
      status = robotHelper(i).S4Standby(NOTIFY_IF_ERROR, resultid)
    Case 2: 'motors on button
      status = robotHelper(i).S4Run(NOTIFY_IF_ERROR, resultid)
    Case 4: 'Stop program cycle
      status = robotHelper(i).S4Stop(0, 3, NOTIFY_IF_ERROR, resultid)
    Case 5: 'run program 1 cycle
      status = robotHelper(i).S4Start(0, "", 1, 1, NOTIFY_IF_ERROR, resultid)
    Case 6: 'run program continuous
      status = robotHelper(i).S4Start(0, "", -1, 1, NOTIFY_IF_ERROR, resultid)
    Case 8: 'move to start of main, could be start of routine by emptying quotes
      status = robotHelper(i).S4ProgramPrep(0, "main", -1, 1, NOTIFY_IF_ERROR,
resultid)
    Case 9: 'load program
      status = robotHelper(i).S4Standby(0, resultid)
    Case 11: 'file manager
      robotExplorer.Show
  End Select
Next i

If status <> 0 Then ' Was function successful ??
  MsgBox "status = " + Str(status) ' no, message to user
End If
//*****

```

Robot Controller Hardware Control Mechanism

The following visual basic source code below will become a vital synchronization mechanism link. This code will ensure the current robot controller event status, ensuring that the control mechanism has manipulated the correct system functionality during upload or download to control the robot controller.

```

//*****
Private Sub robotHelper_StatusChanged(Index As Integer, OprState As Integer, CtlState
As Integer, PgmCtlState As Integer, PgmState As Integer)
Dim msg As String

```

```

If OprState <> prevOprState(Index) Then
  'set up the info software versions data whenever opr state changes
  lblS4Boot(Index).Caption = robotHelper(Index).BootVersion
  lblS4Sys(Index).Caption = robotHelper(Index).SysVersion
  lblRC(Index).Caption = robotHelper(Index).RAPVersion
  lblApp(Index).Caption = robotHelper(Index).ControlId
  Select Case OprState

    Case 0
      'msg = "CommLink Down"
      S4OPERSTATE = "CommLink Down"
      OprSTATEZ = "CommLink Down"
    Case 1
      'msg = "Initialization"
      S4OPERSTATE = "Initialization"
      OprSTATEZ = "Initialization"
    Case 2
      'msg = "Test < 250 mm/s"
      S4OPERSTATE = "Test < 250 mm/s"
      OprSTATEZ = "Test < 250 mm/s"
    Case 3
      'msg = "Going to Auto"
      S4OPERSTATE = "Going to Auto"
      OprSTATEZ = "Going to Auto"
    Case 4
      'msg = "Auto"
      S4OPERSTATE = "Auto"
      OprSTATEZ = "Auto"
    Case 5
      'msg = "Going Test 100%"
      S4OPERSTATE = "Going Test 100%"
      OprSTATEZ = "Going Test 100%"
    Case 6
      'msg = "Test 100%"
      S4OPERSTATE = "Test 100%"
      OprSTATEZ = "Test 100%"
    Case Else
      'msg = "Unknown "
      S4OPERSTATE = "Unknown "
      OprSTATEZ = "Unknown "
  End Select
  lblOprState(Index) = msg
  prevOprState(Index) = OprState
End If

```

```

If CtrlState <> prevCtrlState(Index) Then
  Select Case CtrlState
    Case 1
      'msg = "Initialization"
      CtlSTATEZ = "Initialization"
    Case 2
      'msg = "Stand-By"
      CtlSTATEZ = "Stand-By"
    Case 3
      'msg = "Power On"
      CtlSTATEZ = "Power On"
    Case 4
      'msg = "Run"
      CtlSTATEZ = "Run"
    Case 5
      'msg = "Power Off"
      CtlSTATEZ = "Power Off"
    Case 6
      'msg = "Guard Stop"
      CtlSTATEZ = "Guard Stop"
    Case 7
      'msg = "Emergency Stop"
      CtlSTATEZ = "Emergency Stop"
    Case 8
      'msg = "Guard E-Stop"
      CtlSTATEZ = "Guard E-Stop"
    Case 9
      'msg = "Stand-By E-Rst"
      CtlSTATEZ = "Stand-By E-Rst"
    Case Else
      'msg = "Unknown"
      CtlSTATEZ = "Unknown"
  End Select
  lblCtrlState(Index) = msg
  prevCtrlState(Index) = CtrlState
End If

```

```

If PgmCtrlState <> prevPgmCtrlState(Index) Then
  Select Case PgmCtrlState
    Case 1
      msg = "Uninitialized"
    Case 2
      msg = "Ready"
    Case 3
      msg = "Executing"
    Case 4

```

```

        msg = "Stopped"
    Case 5
        msg = "Full"
    Case Else
        msg = "Unknown"
    End Select
    lblPgmCtrlState(Index).Caption = msg
    prevPgmCtrlState(Index) = PgmCtlState
End If

If msg <> "" Then robotFrame(Index).Caption = robotHelper(Index).Robot + " (" +
msg + ")"

If PgmState <> prevPgmState(Index) Then
    Select Case PgmState
        Case 1
            msg = "Empty"
        Case 2
            msg = "Loaded"
        Case 3
            msg = "Linked"
        Case 4
            msg = "Initiated"
        Case Else
            msg = "Unknown"
    End Select
    lblPgmState(Index).Caption = msg
    prevPgmState(Index) = PgmState
End If

End Sub
//*****

```

Robot system Co-Ordinate Frame Database

Access database structure below provides vital information with regard to the robot co-ordinate frame system. This will be utilized to map the captured object profile co-ordinate system to the correct object with respect to the world co-ordinate system. Figure 4.26 illustrates the Access data file configuration.

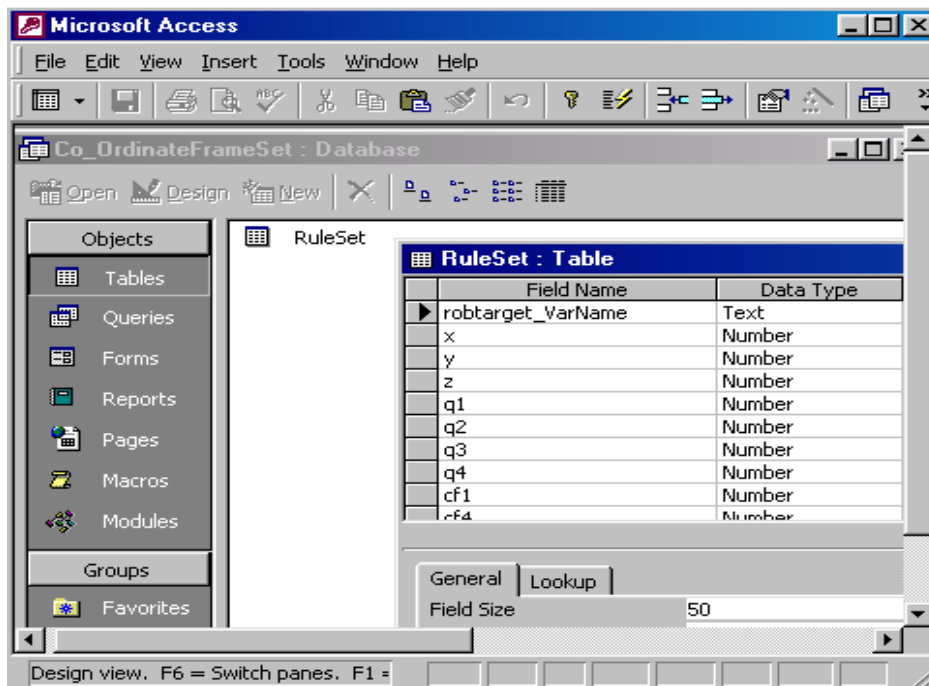


Figure 4.26: Co-ordinate Access Database Structure

The Access Database structure source code is illustrated below [32][33].

```
//*****
'Setup DataBase File Structure
AccessFile = "c:\\" + "Co_OrdinateFrameSet" + ".mdb"

FileCheck = Dir(AccessFile)

If FileCheck = "" Then

'open workspace and database dbEngine
Set WB_Ws = DBEngine.Workspaces(0)
Set WBDData = WB_Ws.CreateDatabase(AccessFile, dbLangGeneral, dbVersion30)

'create new table name
Set WB = WBDData.CreateTableDef("RuleSet")

'creat table field and there field formats

Set WBFlds(0) = WB.CreateField("robtarget_VarName", dbText) ' counter field
WBFlds(0).Size = 50
```

```
Set WBFlds(1) = WB.CreateField("x", dbLong)
WBFlds(1).Size = 50
```

```
Set WBFlds(2) = WB.CreateField("y", dbLong)
WBFlds(2).Size = 50
```

```
Set WBFlds(3) = WB.CreateField("z", dbLong)
WBFlds(3).Size = 50
```

```
Set WBFlds(4) = WB.CreateField("q1", dbLong)
WBFlds(4).Size = 50
```

```
Set WBFlds(5) = WB.CreateField("q2", dbLong)
WBFlds(5).Size = 50
```

```
Set WBFlds(6) = WB.CreateField("q3", dbLong)
WBFlds(6).Size = 50
```

```
Set WBFlds(7) = WB.CreateField("q4", dbLong)
WBFlds(7).Size = 50 .....
```

```
.....
.....
```

```
//*****
```

Robot RAPID Program File transfer Mechanism

In order for the robot controller to receive/transmit program configuration file information between the remote terminal, a file handler needed to be generated to synchronize the file information transfer to the robot controller. This file transfer is only possible when the robot controller has been stopped and in a manual control state. The source code below illustrates the file transfer mechanism, which handles the crucial file information.

```
//*****
```

```
Public Sub copyFiles()
'this routine copies files between devices
'the from list is the selection in the file list window
'the to device and directory are found in the treeview highlight fullpath
Dim status As Integer
Dim resultid As Long
```

```

Dim RobotName As String
Dim fromEquip As String
Dim fromDev As String
Dim fromName As String
Dim toEquip As String
Dim toDev As Strings
Dim toName As String
Dim tag As String
Dim strg As String
Dim i As Integer, j As Integer
Dim flist As ListItem
    If InStr(1, tvwPlant.DropHighlight.FullPath, ":") > 0 Then 'legal drop point
        For i = 1 To FileList.ListItems.Count
            Set flist = FileList.ListItems(i)
            If flist.Selected Then
                'Text1.Text = Text1.Text + flist.Text + ", "
                tag = flist.tag
                'first on the from side...
                strg = Mid$(tag, 13)
                fromEquip = Left$(strg, InStr(1, strg, "\", 1) - 1)
                strg = Mid$(tag, InStr(1, tag, fromEquip, 1) + Len(fromEquip) + 1)
                j = InStr(1, strg, "\", 1)
                If j > 1 Then
                    fromDev = Left$(strg, j - 1)
                    fromName = Mid$(strg, j) + "\" + flist.Text
                Else
                    fromDev = strg
                    fromName = flist.Text
                End If
                'now the to side
                strg = Mid$(tvwPlant.DropHighlight.FullPath, 13)
                toEquip = Left$(strg, InStr(1, strg, "\", 1) - 1)
                strg = Mid$(tvwPlant.DropHighlight.FullPath, InStr(1,
tvwPlant.DropHighlight.FullPath, toEquip, 1) + Len(toEquip) + 1)
                j = InStr(1, strg, "\", 1)
                If j > 1 Then
                    toDev = Left$(strg, j - 1)
                    toName = Mid$(strg, j) + "\" + flist.Text
                Else
                    toDev = strg
                    toName = flist.Text
                End If
                If Not ((fromEquip = "This PC") Or (toEquip = "This PC")) Then
                    j = MsgBox("Can't copy from ramdisk to ramdisk (yet)", vbCritical, "Copy
Error")
                Else

```

```

'set up the helper to work with the proper robot
If (fromEquip <> "This PC") Then
    FMHelper.Robot = fromEquip
ElseIf (toEquip <> "This PC") Then
    FMHelper.Robot = toEquip
End If
j = vbYes
If mnu_confirm.Checked Then
    j = MsgBox("Do you want to copy: " + flist.Text + " to directory: " +
tvwPlant.DropHighlight.FullPath, vbYesNo + vbQuestion, "Confirmation")
End If
If j = vbYes Then
    StatusBar.SimpleText = "Copying: " + flist.Text + " to directory: " +
tvwPlant.DropHighlight.FullPath
    status = FMHelper.S4FileCopy(fromDev, fromName, toDev, toName, 3,
resultid)
    If status <> 0 Then
        j = MsgBox("Copy failed, status=" + Str(status), vbCritical, "Copy
Error")
        StatusBar.SimpleText = ""
    Else
        StatusBar.SimpleText = "Copied: " + flist.Text + " to directory: " +
tvwPlant.DropHighlight.FullPath
    End If
End If
End If
End If
Next i
Set tvwPlant.DropHighlight = Nothing
indrag = False

End If
End Sub
//*****

```

4.9 Remote RAPID DDE Robot Programming Environment

DDE items are utilized as a placeholder for the different variables in the S4 robot controller. To address those variables, the item naming must follow certain rules.

To connect a cell in an MS Excel worksheet to a digital output (ex: do1) in the S4 robot controller, you type: =ABBS4DDE|ROB1!a_digio_raplong_do1 in the formula bar in

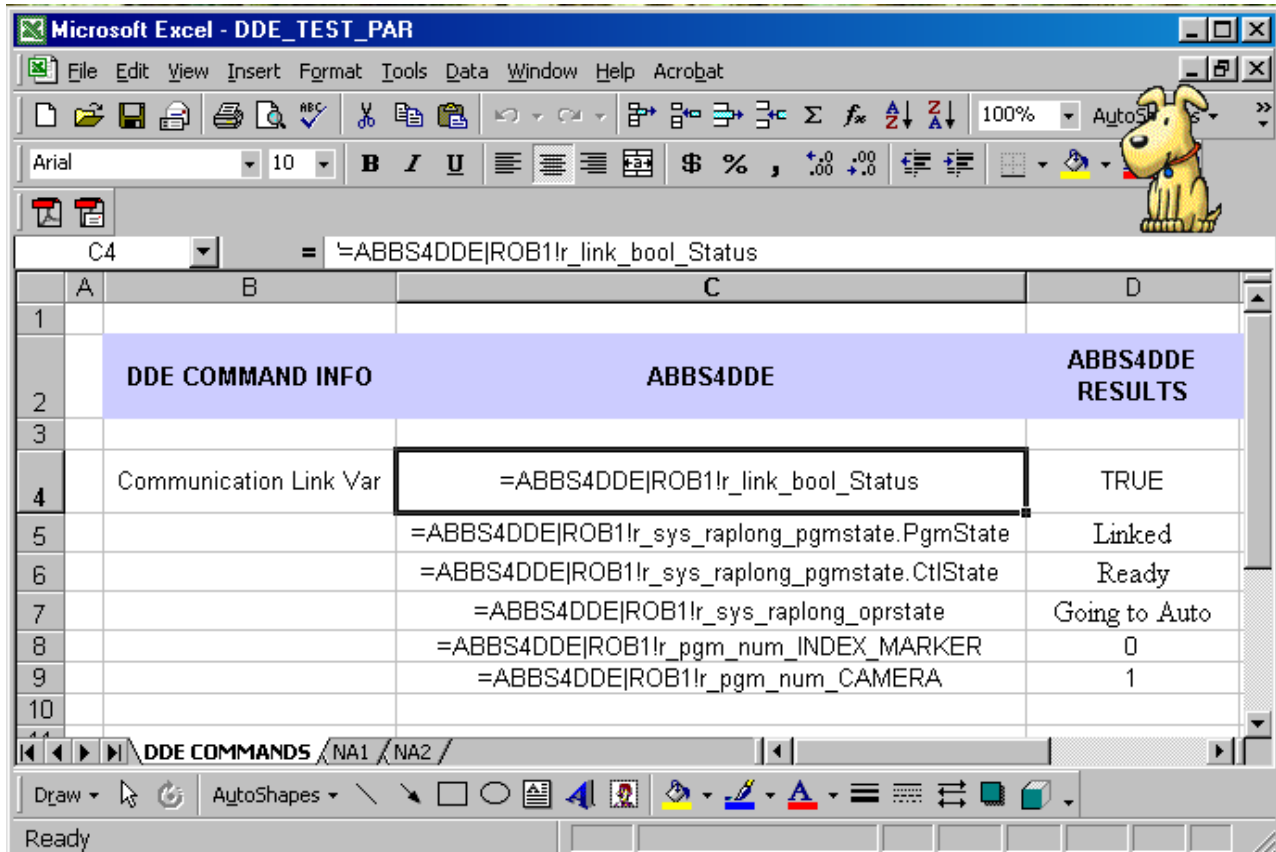


Figure 4.27: Microsoft Excel DDE data simulation with DDE RobComm Server

Robot DDE variable configuration

To connect to a digital output (ex: do1) from Visual Basic, write: LinkTopic =

ABBS4DDE|ROB1, LinkItem = a_digio_raplong_do1 and LinkMode = Automatic.

4.9.1 General DDE item syntax

Connection to variables in the DDE Server are achieved by specifying the name of the variable. There are pre-defined variables and variables defined by the user. The variable names used to connect to the S4 robots are built up using the following system:

<access method> _<functional group> _<variable type> _<variable name>

4.9.2 Access method

Some variables can only be read, some can only be written to, and others can be both read and written to. The name of the variable will indicate this in the first character:

r_ read only
w_ write only
a_ read and write (automatic update variables)

4.9.3 Functional group

The variables are grouped according to their function.

digio_ digital i/o
anio_ analog i/o
rpvar_ rapid program variables
scwri_ superior computer write variables (spontaneous messages)
sys_ system variables
pgm_ program variables
file_ file operation variables
link_ communication link variables

4.9.4 Variable type

The variables have different types that must be specified. The two most used RAPID variable types are:

num_number (single float)
string_string (text)

Other variable types used in the DDE Server are: (as well as complex types like)

raplong_number (long integer)
bool_boolean (0 or 1) (i.e. true or false)
wobjdata_work object data
pos_position data
speeddata_speed data
tooldata_tool data

There are many more data types supported by the DDE Server. Although you have to address complex variables with their correct type, they are reported back as a string with each field separated by a comma. Only RAPID variable types that you can reach from the DDE Server are those that are declared as persistent in your RAPID program.

4.9.5 Variable name

This last field in a complete variable name is the name of the variable as it appears in the S4 controller: an I/O name, a RAPID variable name, a system variable name, etc.

Examples of complete names:

a_digio_raplong_dil
a_rpvar_string_Message
a_rpvar_num_Counter
r_sys_raplong_pgmstate.PgmState

4.9.6 Digital I/O variables

<access method>_DIGIO_<variable type>_<variable name>

Access method: A_ or W_
Functional group: DIGIO_
Variable type: RAPLONG
Variable name: User defined

4.9.7 Digital I/O name example

A_DIGIO_RAPLONG_di1
A_DIGIO_RAPLONG_do1
A_DIGIO_RAPLONG_ingroup1
A_DIGIO_RAPLONG_outgroup1
W_DIGIO_RAPLONG_do1
raplong_ number (long integer)
bool_ boolean (0 or 1) (i.e. true or false)
wobjdata_ work object data
pos_ position data
speeddata_ speed data
tooldata_ tool data

4.9.8 Rapid program variables

Persistent (PERS) Rapid variables defined and declared in your program modules.

<access method>_RPVAR_<variable type>_<variable name>

Access method: A_ or W_
Functional group: RPVAR_

Variable type: STRING or NUM, as well as: wobjdata, pos, speeddata, tooldata, etc.

Variable name: User defined.

Rapid variable name example

A_RPVAR_STRING_Message
A_RPVAR_NUM_Counter
W_RPVAR_STRING_Message

Addressing the DDE items ABB S4 DDE Server

4.10 Conclusion

The discussion in this chapter emphasizes the robot motion programming structure, and different approaches of robot motion, different formulations for robot arm dynamic, RAPID program structure and remote RAPID programming environment. These components ensure that the trajectory application is capable of generating on-line robot RAPID program with reference to the object profile co-ordinate frame.

Trajectory control generates robot RAPID programs in terms of data from object profile planning, kinematics of robot manipulator, and transformation matrices between relevant robot coordinate systems and vision coordinate system. Practical equations that build up the relationship between robot tool frame (represented by TCP of the three-fingered gripper) and object frame have been derived from the transformation matrices. An appropriate quaternion computation method derived in this project guarantees automated generation of robot programs. On this basis, modular RAPID programs are successfully generated and tested with regard to object profiles.

Software I/O variables flags are employed as handshaking signal between execution of robot program and manipulation of the TCP so as to coordinate, these two relatively independent processes.

The discussion in this chapter emphasizes the problem-solving or planning aspect of a robot. A robot planner attempts to find a path from our initial robot world to a final robot

world. The path consists of a sequence of operations that are considered primitive to the system. A solution to a problem could be the basis of a corresponding sequence of physical actions in the physical world.

CHAPTER 5

VISION SENSORY SYSTEM FOR PROFILE RECOGNITION

The advances in vision technology for robotics are expected to broaden the capabilities of robotic vision systems to allow for vision-based guidance of the robot arm, complex inspection for dimensional tolerances, and improved recognition and part location capabilities. These will result from the constantly reducing cost of computational capability, increased speed and new better algorithms currently being developed.

Robot vision plays a critical role in robot intelligence. In robot vision systems, geometric feature extraction and representation are the two most important issues to which we must find solution in terms of the application requirements. To make the present robot vision systems suitable for various eye-hand applications, further researches and improvement still needs to be done. To implement a profile-oriented robot vision system, the following issues must be taken into consideration:

- Image acquisition — includes selection of visual sensors in terms of image resolution of which the vision system must be in possession, illumination and other specifications.
- Image processing techniques — encompasses methodology for image processing in terms of the requirements for real-time, complexity of computation and precision.
- Profile Extraction of object's geometric features — is dependent on the application to which the vision system is going to be applied.

This research focused on developing a profile that is used by the automated robot control application to facilitate on line dynamic response to changes in object profile.

With an automated PC-Based Robot control system, the object position and orientation as well as the object's profile must be identified and represented accurately. A high resolution CCD camera is utilized to acquire visual information of the object profile to be tracked. A real-time system needs to overcome the bottleneck of intensive computation of a vision system. To represent the objects efficiently and effectively, edge vector expression method is developed such that computational efficiency is increased dramatically. A closed chain of vectors is generated. Thereof, profile traversing and feature extracting are conducted with respect to the chain.

This chapter discusses the software components implemented to extract an object's profile as illustrated in Figure 5.1.

5.1 Software Components for Vision

The robot vision system is segmented into two distinct areas.

The first sub-section is the *low-level vision*. This is where the image acquisition and preprocessing function takes place.

The second sub-section is the *high-level vision*. This includes extraction, modeling and profile recognition.

In this research project, the objective of the vision feedback system is to produce an image data co-ordinate map of the image profile captured. The software components developed to implement the vision system is shown in Figure 5.1.

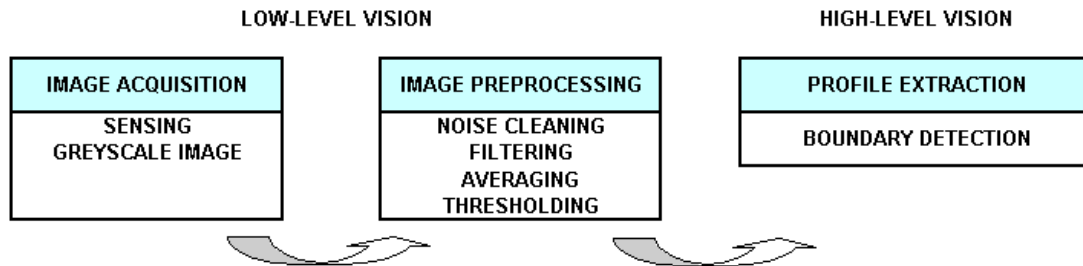


Figure 5.1: Software components to implement a vision sensory system

In this research project, high-level vision provides an image profile co-ordinate map for the robot trajectory application, which will in turn be utilized by the robot manipulation and trajectory program generation engine to direct the robot to its final positions.

5.2 Image Acquisition

The raw captured image needs to be processed to ensure that an accurate profile can be extracted. The mechanism utilized for the preprocessing of the image uses various filters and templates. These are used to eliminate distortion of the original image and will ensure a clear and accurate image [14].

To ensure that a high quality image is captured, a high resolution CCD camera is utilized to view the digitized image. The CCD camera is interfaced to a high-speed frame grabber card, which processes the digital image. The 3D flash point frame grabber card is equipped with programmable hardware in order to improve the viewed image. The

following are examples of programmable features, moderate brightness, contrast, sharpness, etc. To facilitate image processing and increased precision, a clear contrast between the object and the background is needed. In this research project, a dark object was placed on a white background. The vision system views a flat object from a vertical position in order to eliminate shadows, non-uniform illumination and all distortions that cannot be compensated by adjusting the parameters of the frame grabber. Figure 5.2 illustrates the test image captured of an object.

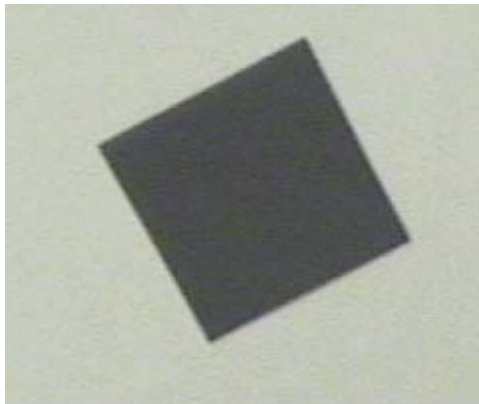


Figure 5.2: Image capture via frame grabber

The digital image captured is stored in the VGA system memory of the computer. The RGB colour image is transferred into the program buffer for further processing. The source code declaration illustrates the memory allocation for the captured image.

```
LPBYTE AllocateMemory() {  
  
    return LPImage = new BYTE[size.cx * size.cy * 3L +  
        sizeof(BITMAPINFOHEADER)];  
  
}
```

Figure 5.3: Memory allocation for the 3D Flash Point Frame Grabber [23]

```
FPV_ScreenToDIB((VIDEOWIDTH - m_IS.size.cx) / 2, (VIDEOHEIGHT -  
m_IS.size.cy) / 2, (SHORT) m_IS.size.cx, (SHORT) m_IS.size.cy,  
(SHORT) STD_OFFSCREEN, (LPSTR) m_image.lpImage);
```

Figure 5.4: Mechanism utilized to transfer the image into the allocated memory

DIB - Device-Independent Bitmaps

Bitmaps that contain a *color table* are device-independent. A color table describes how pixel values correspond to *RGB* color values. RGB is a model for describing colors that are produced by emitting light. A DIB contains the following color and dimension information:

- The color format of the device on which the rectangular image was created
- The resolution of the device on which the rectangular image was created
- The palette for the device on which the image was created
- An array of bits that maps red, green, blue (RGB) triplets to pixels in the rectangular image
- A data-compression identifier that indicates the data compression scheme (if any) used to reduce the size of the array of bits

BITMAPINFOHEADER

The BITMAPINFOHEADER structure contains information about the dimensions and color format of a device-independent bitmap (DIB), which is illustrated below [35][25].

```

typedef struct tagBITMAPINFOHEADER{ // bmih
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;

```

5.3 Image Preprocessing

In order for the object profile to be captured correctly, the image needs to be preprocessed by applying standard matrix filter algorithms, which extract unwanted noise and image distortion. The algorithms utilized sequentially on the captured image are -

- image filtering,
- noise cleaning,
- averaging; and
- image thresholding.

The spatial convolution technique of the digital image is the platform basis for the software.[26]

```

void CImageProfileProcessing::ConvolutionMech(const int nConV[3][3], BOOL style,
int nConstant)
{
    int i, j;
        int nRow, nCol;

```



```

int nTemp;

for(i = 1; i < m_IS.size.cy - 1; i++)
{
for(j = 1; j < m_IS.size.cx - 1; j++)
{
nTemp = 0;
for(nIR = -1; nIR <= 1; nIR++)
{
for(nIC = -1; nIC <= 1; nIC++)
{
nTemp += *(m_lpImage + (i + nIR) * m_IS.size.cx + j +
nIC) * nConV [nIR + 1][nIC + 1];
}
}
}
nTemp /= nConstant;
if(style == GREYSCALE) {
if(nTemp > WHITE_POINT) nTemp = WHITE_POINT;
if(nTemp < BLACK_POINT) nTemp = BLACK_POINT;
}
else {
if(nTemp > 1) nTemp = 1;
if(nTemp < 0) nTemp = 0;
}
*(m_lpImageTemp + i * m_imageSize.cx + j) = nTemp;
}
}
}
}
}

```

5.3.1 Image Filtering

The three filters utilized are -

- high-pass filters,
- median filters; and
- low-pass filters.

The three filters are combined to ensure the ideal image is achieved.

The **high-pass filter** is utilized to sharpen the image that is out of focus or fuzzy. The function uses a 3x3 convolution matrix filter, which emphasizes differences in grey pixel levels in the 3x3 neighborhood about the central pixel window. Figure 5.5 illustrates the convolution filter window mechanism.

$$H_{High-pass} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 5.5: 3x3 matrix high-pass convolution filter

```
const int nConV [3][3] = {{-1, -1, -1}, {-1, 9, -1}, {-1, -1, -1}};  
ConvolutionMech (nConV);
```

Figure 5.6: Convolution filter mechanism

Once the filtering process has been completed, the image is sharpened and the low frequency component in the image is removed. The template matrix used in the function is described in chapter 3.

Image random noise spikes on a noisy image. The noise is removed by utilizing a **median filtering** mechanism. The median filtering mechanism utilizes a non-linear sorting of pixel intensity levels in a 3x3 matrix window. This is used to smooth out ‘salt and pepper’ noise effects. Random noise becomes less apparent after a median filter has been applied to the image. This creates a smoother image. The implementation of median filtering selects windows of pixel data from a 3x3 array of pixels. The pixel set consists of nine pixels. The intensity of the pixel set is analyzed in order of grey scale magnitude.

The central matrix point is dependent on its current intensity. A 3x3 matrix window is moved over the entire video memory image grid, which is used to analyze the entire image pixel map.

Low-pass filtering is utilized to reduce or eliminate high frequency noise components as well as smooth or soften sharp images. A 3x3 matrix convolution filter is utilized to provide a filtering mechanism to eliminate the high frequency components from the captured video image.

$$H_{Low-pass} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 5.7: 3x3 matrix low-pass convolution filter

```
const int nConV [3][3] = {{1, 1, 1}, {1, 2, 1}, {1, 1, 1}};
ConvolutionMech (nConV, GREYSCALE, 10);
```

Figure 5.8: 3x3 matrix convolution filter

5.3.2 Noise Cleaning

Random noise signals, which occur in the captured image are reduced or eliminated with noise cleaning techniques. A 3x3 matrix convolution is utilized to smooth out this noise.

$$H_{Noise-cleaning} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 5.9: 3x3 matrix convolution noise smoothing filter

```
const int nConV [3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
ConvolutionMech (nConV, GREYSCALE, 12);
```

Figure 5.10: 3x3 matrix convolution filter

5.3.3 Averaging

The averaging technique is utilized to eliminate noise spikes and clean edge features in the image. The averaging function uses a 3x3 matrix sliding convolution window to smooth out the noisy image.

$$H_{Averaging} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 5.11: 3x3 matrix averaging convolution filter

```
const int nConV [3][3] = {{1, 1, 1}, {1, 0, 1}, {1, 1, 1}};  
ConvolutionMech (nConV, GREYSCALE, 8);
```

Figure 5.12: 3x3 matrix convolution filter implemented as a software call function

5.3.4 Image Thresholding

This technique is the main mechanism, which provides the image building block for establishing boundaries in images. The image contains a solid object resting on a contrasting background. The process of thresholding is that a threshold greylevel intensity range is established. The entire pixel image is analyzed and each pixel is altered according to its level of intensity. Should the current pixel fall below the greylevel range, the pixel is assigned to the background, while the other pixels that are equal or above the greylevel range are considered to be the object and are assigned to an object reference pixel. The greylevel ranges from 0 – 255. The manual threshold value is determined by calculating the difference of the intensity level between the object and the background.

Figure 5.13 illustrates an accurate binary image achieved through filtering, noise cleaning, averaging, and thresholding.

```
void CImageProfileProcessing::TH()
{
    int i, j;

    for(i = 0; i < m_IS.size.cy; i++)
    {
        for(j = 0; j < m_IS.size.cx; j++)
        {
            if(*(m_lpImage + i * m_IS.size.cx + j) > m_nThreshold)
            {
                *(m_lpImage + i * m_IS.size.cx + j) = TRUE;
            }
            else
            {
                *(m_lpImage + i * m_IS.size.cx + j) = FALSE;
            }
        }
    }
}
```

Threshold is one of the most commonly used image preprocessing tools utilized in image recognition system. This method has been illustrated below.

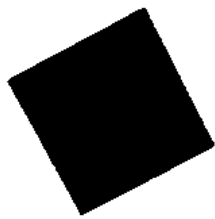


Figure 5.13: Image after threshold mechanism has been applied

5.4 Boundary Detection

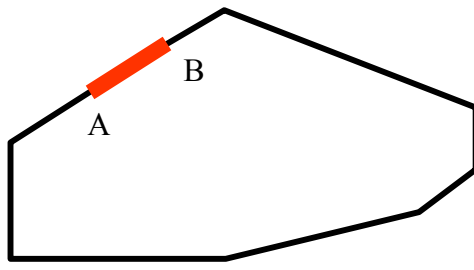
Boundary detection provides an improved mechanism to accelerate the process of image processing. The boundary features a sharp grayscale transition. If the edges are reliably strong, and the noise level is low, one can establish the edge magnitude of the image with a close chain mechanism. Close chain discontinuities occur at boundaries, which result from noise and other interferences. Edge linking and edge thinning must be applied after edge detection.

5.4.1 Edge Detection

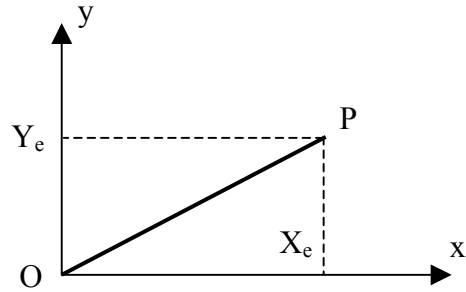
Edge detection is implemented by analyzing each pixel neighborhood and quantifying the slope and direction of the object. In this research project, edge detection utilizes gradient slope detection operators.

5.4.2 Edge Linking

This technique analyses the video pixel binary image and manipulates any pixels, which lie on the boundary between the object and the background. In this application, linear interpolation is employed to create a continuous boundary. Figure 5.14 (a) shows the discontinuity from point A to point B on the boundary, while figure 5.14 (b) shows the principle of linear interpolation.



(a)



(b)

Figure 5.14: (a) Discontinuity from A to B (b) Principle of linear interpolation

The discontinuity from A and B in figure 5.14 (a) is expressed with line segment OP in figure 5.14 (b). Assume the distance from origin O to P is X_e pixels in the x axis and Y_e pixels in the y axis. The linear interpolation method is employed to join point O and point P. The current co-ordinates of the point being interpolated is saved in a co-ordinate variable.

X, Y1, Xr and Yr stand for registers that store the interpolation variables in the x and y axis respectively. Figure 5.15 illustrates the interpolation principle.

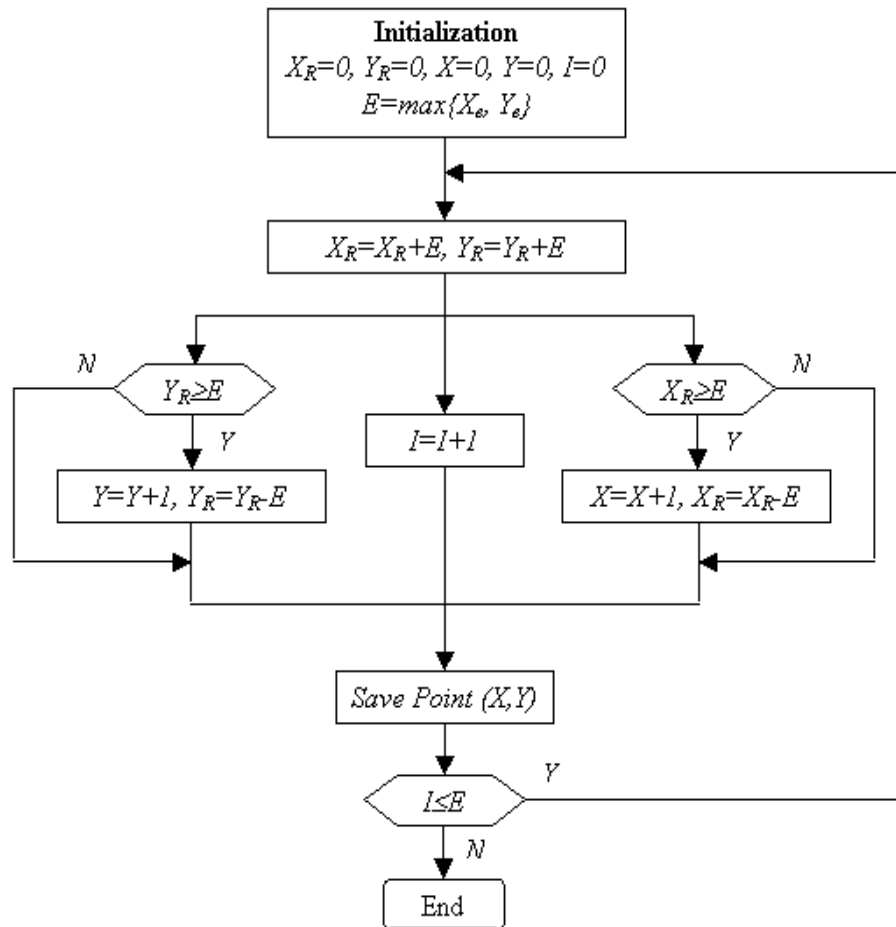


Figure 5.15: Block diagram of interpolation principle

This method allows the profile to be traced and thinned to a single pixel-wide. This facilitates extraction of the object profile.

5.4.3 Edge Following and Thinning

The chain code mechanism is utilized to represent the object profile. The chain codes that are utilized in this research project comprise sets of straight-line segments of specified length and direction, which correlate to the object boundary sides. Edge thinning is performed while edge following is being executed. A single pixel-wide profile is then

generated. The chain code contains the start pixel address followed by a string of code words. Figure 5.16 shows the profile of the object after edge linking, edge following, and edge thinning.

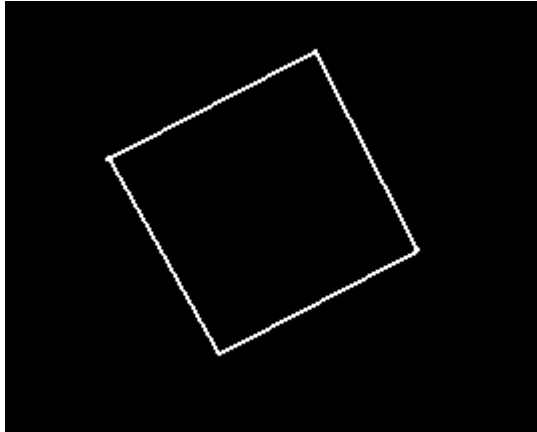


Figure 5.16: Profile of the object

5.5 Extraction of the Image Profile

Extraction of the profile from the object becomes a crucial mechanism for robot path manipulation. Closed chain vectors represent the object profile, which is illustrated in Figure 5.17. A continuous vector tracing method was developed, which utilizes standard mathematic algorithms. This ensures that the chain profile segments contain the orientation and magnitude of the object.

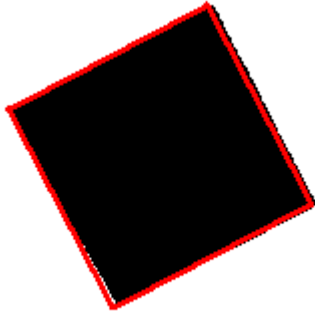


Figure 5.17: Object Profile

This mechanism was developed to represent the profile of the object by means of a closed chain of vectors. The profile is represented by a square of different sizes. This vector tracing method locates the starting point of the vector chain within a given tolerance.

Vector algebra fundamental

Length of vector $\mathbf{v} = ix + jy$ is defined as

$$|\mathbf{v}| = \sqrt{x^2 + y^2}$$

Unit vector \mathbf{v}_0 of vector \mathbf{v} is defined as

$$\mathbf{v}_0 = \frac{\mathbf{v}}{|\mathbf{v}|} = \frac{ix + jy}{\sqrt{x^2 + y^2}}$$

Angle θ between two vectors \mathbf{v}_1 and \mathbf{v}_2 can be calculated by

$$\theta = \cos^{-1}(\mathbf{v}_{10} \cdot \mathbf{v}_{20})$$

$\mathbf{v}_{10} \cdot \mathbf{v}_{20}$ stands for the dot product (or inner product) of vector \mathbf{v}_{10} and vector \mathbf{v}_{20} which are the unit vectors of vectors \mathbf{v}_1 and \mathbf{v}_2 respectively.

Discriminator of two orthogonal vectors \mathbf{v}_1 and \mathbf{v}_2 is defined as

$$\mathbf{v}_{10} \cdot \mathbf{v}_{20} = 0$$

Discriminator of two parallel vectors \mathbf{v}_1 and \mathbf{v}_2 sharing the same direction is defined as

$$v_{10} \cdot v_{20} = 1$$

Discriminator of two parallel vectors v_1 and v_2 , having opposite directions, is defined as

$$v_{10} \cdot v_{20} = -1$$

This algorithm is to locate each corner (or node) within given tolerance.

Figure 5.18 illustrates the profile Image co-ordinate MAP which is utilized the generate a robot trajectory motion path.

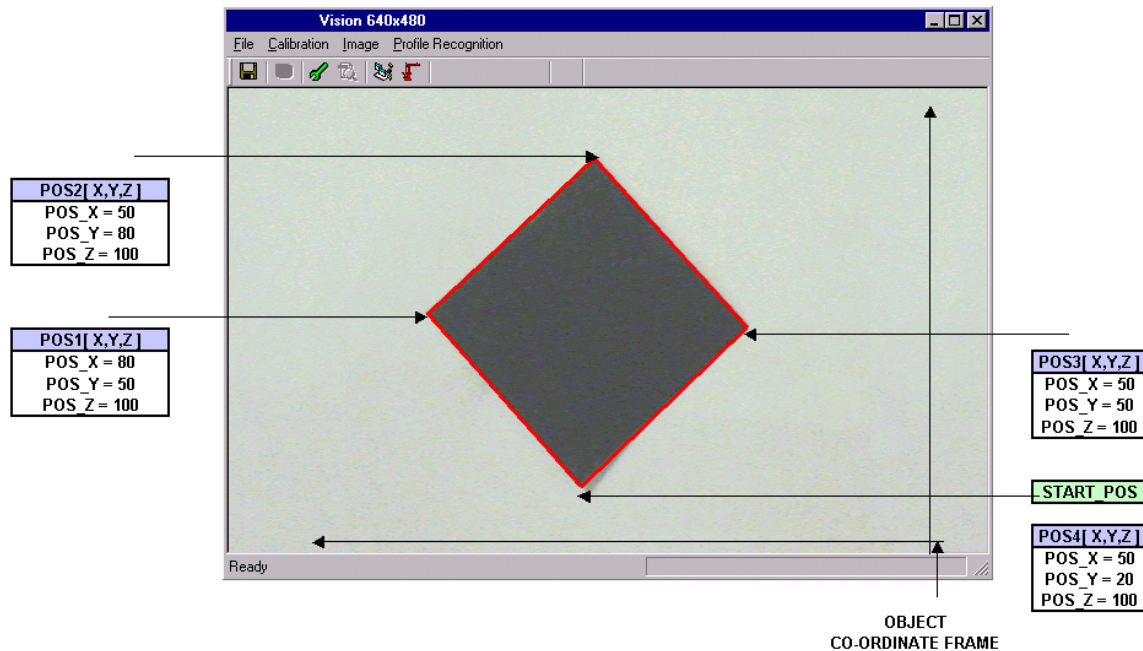


Figure 5.18: Object Profile Image Co-ordinate MAP

5.6 Conclusion

Vision feedback systems play a crucial role in robot trajectory manipulation. This system is by far the most complex feedback system that can be utilized by a robot controller.

This complex system ensures that the robot controller gains maximum movement

flexibility. Real-time image processing is highly dependent on the relevant algorithms. The overhead of the algorithm processing time influences the real-time close loop feedback system. The real-time time frame can be analyzed by calculating the time from which the vision system views the object to the time the robot controller reacts to the trajectory co-ordinate plan.

Therefore in this research project it was proved that real-time robot manipulation can be achieved by utilizing the vision feedback system.

CHAPTER 6

CONCLUSION

This chapter discusses the final project results. It will also illustrate the accomplishments and contributions of this study with regards to the objectives set out in Chapter 1. Problems that were encountered during the project's development as well as future extensions to the project will be highlighted.

6.1 Project Results

Profile recognition and an automated robot trajectory system was created to provide industry with a more flexible approach to remote automated robot systems.

A standard industrial ABB robot and bridge PC were used to test the project results. The two entities were linked together with a standard industrial LAN (Ethernet protocol). The system network link was configured in a peer-to-peer format. The bridge PC was equipped with two Ethernet cards as discussed in Chapter 3. One of the network cards was used for the network link between the robot controller and the bridge PC, the other network card was used for the LAN which enabled remote communication from the internet. An integrated software application was divided into two main areas, vision sensory system and robot trajectory path control which is illustrated in Chapter 4. All the image processing and extraction is handled by the first component while the second component manages the motion control. Software to enable communication between the bridge PC and robot controller was also established.

The following was performed and illustrated:

- The object was extracted and managed via the CCD camera and frame grabber, which was integrated in the Visual C application.
- The square flat black object was placed on a white background to create a guaranteed contrast between the object and the background.
- The image was grabbed into the vision extraction application, which analyzed and processed the image. A profile of the object was then generated.
- The profile object was manipulated so that all the crucial co-ordinates were extracted such as the marker position of all the corners of the square object. The co-ordinates were utilized to create a robot trajectory path for the ABB industrial robot controller.
- The ABB industrial robot was utilized to provide position manipulation from the results.
- The Ethernet cards were installed into both devices (bridge PC and robot controller).
- The Ethernet protocol was successfully established via the configuration of the ABB controller and bridge PC. This was done via the RobComm server.
- The ActiveX RobComm component was configured to ensure a stable and flexible communication platform was achieved.

- The remote robot position manipulation was successfully achieved by creating a real-time position control.
- The DDE engine ensures that a standard SCADA application can be integrated with a robot controller ensuring flexible remote applications.
- The profile recognition system was successfully integrated with the robot controller to provide an integrated feedback system.
- A remote robot programming platform was developed to manipulate kinematics path motion.

6.2 Accomplishments and Contributions of final results

The experimental validation of the profile recognition and automated PC-Based Robot control system was made on a variety of distinguished objects. The experiment result is proven to be successful and fulfill the desired objectives.

The accomplishments and contributions of this study can be summarized as follows:

- A vision profile recognition and automated PC-Based robot control system integrated with ABB IRB1400 industrial robot is established and implemented successfully.
- The generic algorithms developed in the robot vision system give practical and effective solutions to machine vision.
- The robot manipulator trajectory planning algorithm succeeded in generating RAPID programs automatically with respect to the object profile and orientation.

- Seamless integration between individual modules. Serial data flow supports each module effectively.

6.3 Problems encountered

Problems that were encountered during this research project were of a software nature. All hardware was available, but the software limited full control of the hardware. The software problems, which were encountered, were overcome within time. This ensured that all testing and development was achieved.

No LAN was available therefore a peer-to-peer Ethernet communication was established for the remote communication between the robot controller and bridge PC. The second bridge PC Ethernet card was connected to a second remote PC to simulate a remote LAN communication. The IP address utilized was of a static nature and had to be manually configured. This was successfully established.

The current Ethernet hardware for the ABB industrial robot controller required specific firmware and system services to be installed in order for the Ethernet service to be activated. Initially these were not available. A large amount of time was wasted sourcing the correct version of firmware for the robot baseware software. This problem was later solved.

The factoryware software that was available was utilized to bridge the Ethernet communication gap. This software was found to be unstable at times due to the software license development environment. This caused major constraints in the development environment as the ActiveX component of RobComm could not be correctly utilized in

the Visual C environment. Therefore a Visual Basic platform had to be developed and was used to manage the remote communication platform between the robot controller and the bridge PC. This second platform had an impact on the real-time environment, which wasted unnecessary processing time.

6.4 Possible extensions and conclusions

There are many possible extensions from this basic setup. This research project proved to be a vital platform for a remote automated robot system.

- The current vision system setup only allows a 2D vision environment. An additional non-contact displacement measurement device (laser) can be incorporated to provide the profile depth. This will provide a 3D virtual image, which would create additional system flexibility.
- A 3D robot simulation software environment such as ‘Deneb’ can be integrated with the robot RAPID programming environment. After the entire robot environment has been simulated around the product tooling a base robot software program can be automatically generated. This will reduce robot programming time. Only fine tuning of position can be corrected during project system commissioning.
- The Ethernet communication environment provides the ideal platform for real-time position control. By adding additional intelligence software to the current

system the robot can automatically follow a moving object and make intelligent decisions dependent on the vision feedback.

The aim of this research project was to provide a platform for vision feedback and an automated remote robot environment. During the research it was proved that a robot is able to react on vision feedback sensory information. This platform can prove to be a vital component of the motor manufacturing industry.

References

- [1] Fu K. S. 1987. Robotics: control, sensing, vision, and intelligence, New York: McGraw-Hill.
- [2] Groover M.P., Weiss M. et al. 1986. Industrial Robotics Technology, Programming, and Applications. New York: McGraw-Hill.
- [3] Todd D. J. 1986. Fundamentals of robot technology: An introduction to industrial robots, teleoperators and robot vehicles. USA. Halsted Press.
- [4] Megahed M. 1993. Principles of robot modeling and simulation. King Saud University Saudi Arabia. USA. John Wiley & Sons, Inc.
- [5] Wesley E. Snyder. 1985. Industrial robots: computer interfacing and control. Prentice-Hall International Editions. New Jersey.
- [6] Sayers C. 1998. Remote control robotics. University of Pennsylvania.
- [7] Richard M. Murray., Zexiang Li., S. Shankar Sastry. 1993. A mathematical introduction to Robotic manipulation. CRC Press. New York, Washington, D.C.
- [8] Miklovic D. 1993. Real-Time control networks, resources for measurement and control services. USA.
- [9] Martins J.G., Svensson M. 1988. Profitability and industrial robots. Springer-Verlag. New York.
- [10] Richard P. P. 1981. Robot manipulation: mathematics, programming, and control, the computer control of robot manipulators. MIT Press. London England.
- [11] The Australian National University. Vision System Calibration, Experiment instruction.

- [12] Hames B. 2000. Image Processing and Analysis. Oxford University Press Inc. New York.
- [13] Toh T., Ching W. 1992. Automatic Optimization of Machine Vision Lighting. Proceedings of the 2nd Singapore International Conference on Image Processing.
- [14] Gonzalez R., Wintz P. 1987. Digital Image Processing. Second Edition, Addison-Wesley Publishing Company.
- [15] Nof S. 1985. Handbook of Industrial Robotics. USA. John Wiley & Sons, Inc.
- [16] Groover M. 1986. Industrial Robotics — Technology, Programming, and Applications. McGraw-Hill, Inc. USA.
- [17] MuKai T., Ohnishi N. 1999. Sensor Fusion of a CCD Camera and an Acceleration-Gyro Sensor for the Recovery of Three-Dimensional Shape and Scale, IEEE Proceedings of the Second International Conference of Information Fusion, pp.221-228.
- [18] Li T. and Latombe J. On-Line Manipulation Planning for Two Robot Arms in a Dynamic Environment, Research Report. Robotics Laboratory. Stanford University.
- [19] Choset H. Path Planning between Two Points for a Robot Experiencing Localization Error in Known and Unknown Environments. Research Report.
- [20] Zeller M. 1997. Motion planning of a pneumatic robot using a neural network. IEEE Control Systems Magazine, vol.17, No.3, pp.89-98.
- [21] Production Manual IRB 1400. ABB Flexible Automation.
- [22] ABB User's Guide. ABB Flexible Automation.

- [23] Color CCD Camera Operating manual. Eagle Technology.
- [24] Flashpoint 3D User Manual. Integral Technologies, Inc. September 1999.
- [25] RAPID Reference Manual. ABB Flexible Automation.
- [26] Kruglinski D. 1997. Inside Visual C++. Washington. Microsoft Press.
- [27] Schildt H. MFC Programming from the Ground Up. California. McGraw-Hill.
- [28] Kreyszig E. 1999. Advanced Engineering Mathematics. 8th Edition. New York. John Wiley & Sons, Inc.
- [29] Jordan D. Smith P. 1997. Mathematical Techniques — An Introduction for the Engineering Physical, and Mathematical Sciences. 2nd Edition. Oxford University Press.
- [30] Colombo C., Allotta B. 1999. Image-Based Robot Task Planning and Control Using a Compact Visual Representation. IEEE Transactions On Systems, Man, And Cybernetics – part A: Systems and Humans. vol.29, No.1, pp.92-100.
- [31] Lu T. F. 1996. CAD, vision and sensor based intelligent robot server. Computer Integrated Manufacturing Systems. Vol. 9, No. 2, 91-100, 1996.
- [32] Boston Technical Books. 1994. PC Instrumentation for the 90s. 4th Edition.
- [33] Microsoft Corporation. 1997. Visual Basic: Component Tools Guide. USA.
- [34] Microsoft Corporation. 1997. Visual Basic: Guide to Data Access Objects. USA.
- [35] Tanenbaum A.S. 1996. Computer Networks. 3rd Edition. Prentice-Hall International. USA.

- [36] Chapman D. 1998. Visual C++ 6. Sams Publishing. USA.
- [37] Barr A., Cohen P., Feigenbaum E. 1981-1982. The Handbook of Artificial Intelligence. William Kaufmann, Inc. California. Vols1,2,3.
- [38] Liebermann L.I., Wesley M.A. 1977. AUTOPASS: An Automatic Programming System for computer controlled Mechanical Assembly. IBM J. Research Development. Vol 21, no. 4. pp321-333.
- [39] Wesley M.A. 1980. A geometric Modeling system for Automated Mechanical Assembly. IBM J. Research Development. Vol 24, no. 4. pp64-74.
- [40] Castleman R.R. 1977. Digital image processing. Englewood Cliffs N.J. Prentice-Hall.
- [41] Chein R.T., Snyder W.E. 1975. Hardware for Visual Image Processing. IEEE Transactions on Circuits and systems.
- [42] Denavit J., Hartenberg R.S. 1955. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. J. App. Mech., Vol 77, pp 215-221.