SIMULATION, CONTROL AND REMOTE (INTERNET) COMMUNICATION OF AN INDUSTRIAL ROBOT IN A MANUFACTURING ENVIRONMENT

by

Johannes Jacobus Naude

A dissertation submitted to the faculty of Engineering in partial fulfilment of the requirements for the degree of DOCTOR OF ENGINEERING in MECHANICAL AND MANUFACTURING ENGINEERING at the RAND AFRIKAANS UNIVERSITY Johannesburg, South Africa

Supervisor: Professor Z. Katz

February 1998
Abstract

A simulation system of an industrial robot, within a manufacturing environment for its intelligent interaction within the cell as well as its control via the Internet is presented. The simulation verification in an experimental cell in which an ABB IRB 2400 robot operates is discussed. Sensors employed throughout the cell to supply the input for robot action through an expert system are described. The robot interacts with several task groups of the cell, production equipment, materials handling and assembly. The cell use of a PC, directly linked to the robot and other equipment and sensors for, cell control is explained. The PC has full on-line control of all equipment while the simulation runs simultaneously with the experimental set-up. The system incorporates robot and cell control via the Internet. To add additional intelligence to the cell a transponder system, tagging each part in the robotic cell, is also implemented. This enables each part to be identified by the robot, as well as for the robot to interact with each transponder.
Acknowledgements

First of all I would like to thank the Almighty God for giving me the ability and strength to successfully complete this work.

I would also like to thank my supervisor, Prof. Z. Katz, for his guidance and help throughout this work.
List of symbols

Chapter 2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Capacitance</td>
</tr>
<tr>
<td>C_{tune}</td>
<td>Tuning capacitance</td>
</tr>
<tr>
<td>( f )</td>
<td>Frequency</td>
</tr>
<tr>
<td>L</td>
<td>Inductance</td>
</tr>
<tr>
<td>Q</td>
<td>Quality factor</td>
</tr>
<tr>
<td>R_{S}</td>
<td>Effective series resistance</td>
</tr>
<tr>
<td>X_{C}</td>
<td>Reactance of C</td>
</tr>
<tr>
<td>X_{eff}</td>
<td>Effective reactance</td>
</tr>
<tr>
<td>X_{L}</td>
<td>Reactance of L</td>
</tr>
</tbody>
</table>

Chapter 3

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Width/length of a quadratic antenna</td>
</tr>
<tr>
<td>B_{min}</td>
<td>Minimum magnetic flux</td>
</tr>
<tr>
<td>C</td>
<td>Capacitance</td>
</tr>
<tr>
<td>C_{chip}</td>
<td>Capacitance of chip</td>
</tr>
<tr>
<td>cd</td>
<td>Conducting diameter</td>
</tr>
<tr>
<td>C_{p}</td>
<td>Parasitic capacitance</td>
</tr>
<tr>
<td>C_{s}</td>
<td>Specific capacity between two conductors</td>
</tr>
<tr>
<td>C_{w}</td>
<td>Interwinding capacity</td>
</tr>
<tr>
<td>d</td>
<td>Diameter of copper wire</td>
</tr>
<tr>
<td>f_{self}</td>
<td>Self resonant frequency of the coil</td>
</tr>
<tr>
<td>f_{res}</td>
<td>Resonant frequency</td>
</tr>
<tr>
<td>I_{ant}</td>
<td>Antenna current</td>
</tr>
<tr>
<td>L</td>
<td>Inductance</td>
</tr>
<tr>
<td>L_{PC}</td>
<td>Inductance of coil at ( f_{res} )</td>
</tr>
<tr>
<td>N_{A}</td>
<td>Number of windings or turns</td>
</tr>
<tr>
<td>r</td>
<td>Antenna radius</td>
</tr>
<tr>
<td>R_{chip}</td>
<td>Resistance of the chip</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>$R_{DC}$</td>
<td>Direct current resistance</td>
</tr>
<tr>
<td>$R_m$</td>
<td>Resistance per meter</td>
</tr>
<tr>
<td>$R_{ohm}$</td>
<td>Ohmic resistance</td>
</tr>
<tr>
<td>$R_{PC}$</td>
<td>Parallel resistance of coil at $f_{res}$</td>
</tr>
<tr>
<td>$R_S$</td>
<td>Effective series resistance</td>
</tr>
<tr>
<td>$s$</td>
<td>Distance between conductors</td>
</tr>
<tr>
<td>$U$</td>
<td>Circumference of antenna</td>
</tr>
<tr>
<td>$U_{A-B}$</td>
<td>Voltage at A-B</td>
</tr>
<tr>
<td>$x$</td>
<td>Transmission range</td>
</tr>
<tr>
<td>$X_{eff,s}$</td>
<td>Effective series reactance</td>
</tr>
<tr>
<td>$\omega$</td>
<td>$2 \cdot \pi \cdot f$</td>
</tr>
<tr>
<td>$\mu_0$</td>
<td>Permeability</td>
</tr>
</tbody>
</table>

**Chapter 4**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>3-D plane</td>
</tr>
<tr>
<td>$Q$</td>
<td>Quaternion</td>
</tr>
<tr>
<td>$u$</td>
<td>3-D point</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Roll robot link</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Pitch robot link</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Cylindrical robot link</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>3-D</td>
<td>Three dimensional</td>
</tr>
<tr>
<td>ABB</td>
<td>Asea Brown Bovery</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>BCC</td>
<td>Block Check Character</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic-link Library</td>
</tr>
<tr>
<td>DOF</td>
<td>Degree Of Freedom</td>
</tr>
<tr>
<td>DOS</td>
<td>Disk Operating System</td>
</tr>
<tr>
<td>EAS</td>
<td>Electronic Article Surveillance</td>
</tr>
<tr>
<td>EVA</td>
<td>Extravehicular Activities</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocall</td>
</tr>
<tr>
<td>GHz</td>
<td>Giga Hertz</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HITAG</td>
<td>Phillips/Mikron trademark</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocall</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocall</td>
</tr>
<tr>
<td>KHz</td>
<td>Kilo Hertz</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MDI</td>
<td>Multiple Document Interface</td>
</tr>
<tr>
<td>MHz</td>
<td>Mega Hertz</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>N DOF</td>
<td>N Degree Of Freedom</td>
</tr>
<tr>
<td>PC</td>
<td>Personal computer</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>TF</td>
<td>Transformation Matrix</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocall</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocall / Internet Protocall</td>
</tr>
<tr>
<td>TCP&lt;sub&gt;R&lt;/sub&gt;</td>
<td>Tool Center Point</td>
</tr>
<tr>
<td>UDP</td>
<td>User Diagram Protocall</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>VRML</td>
<td>Virtual Reality Modelling Language</td>
</tr>
<tr>
<td>Web</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>WITS</td>
<td>Web Interface for Telesience</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1 System defined speed data values. 18
Table 6.1 Transponder reading distances. 141

List of Figures

Chapter 2

Figure 2.1 Definition of tool co-ordinate system. 20
Figure 2.2 Position control Rapid programme. 24
Figure 2.3 Path control Rapid programme. 27
Figure 2.4 Internet Growth since 1987. 29
Figure 2.5 Hardware layout of transponder system. 39
Figure 2.6 Flat cable antenna circuit diagram. 40

Chapter 3

Figure 3.1 Communication layout. 43
Figure 3.2 Photograph of robotic cell with predefined positions indicated. 47
Figure 3.3 Schematic layout of robotic cell paths and positions. 48
Figure 3.4 Assembly. 49
Figure 3.5 Production Equipment. 50
Figure 3.6 Materials Handling. 51
Figure 3.7 Flow diagram of rule base. 53
Figure 3.8 Antenna radius versus Transmission range for card transponders. 56
Figure 3.9 Antenna radius versus Transmission range for disc transponders. 56
Figure 3.10 Function $f(N_A)$ of the number of turns of the flat cable antenna. 59
Figure 3.11 Tolerated range of the quality factor in dependance of the effective series reactance. 62
Figure 3.12 Antenna fitted on pneumatic gripper. 62
Figure 3.13 Transponder. 63
Figure 3.14 Equivalent transponder circuit. 63
Figure 3.15 Constructed transponder fitted inside a part. 67
### Chapter 4

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>View of robot from the side, front and above.</td>
<td>70</td>
</tr>
<tr>
<td>4.2</td>
<td>Simulation wire frame with dimensions.</td>
<td>71</td>
</tr>
<tr>
<td>4.3</td>
<td>Axis 4, 5 and 6 of simulation with dimensions.</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>Rotation line AB is specified by point A and B.</td>
<td>72</td>
</tr>
<tr>
<td>4.5</td>
<td>The co-ordinate system after translating the original to point A.</td>
<td>73</td>
</tr>
<tr>
<td>4.6</td>
<td>Configuration Quadrants.</td>
<td>77</td>
</tr>
<tr>
<td>4.7</td>
<td>Schematic of classified IRB 2400.</td>
<td>78</td>
</tr>
<tr>
<td>4.8</td>
<td>The two possible positions for u3.</td>
<td>80</td>
</tr>
<tr>
<td>4.9</td>
<td>Angles of $\lambda_1$ together with X and Y-axes.</td>
<td>81</td>
</tr>
<tr>
<td>4.10</td>
<td>The two possible positions for $u_4$.</td>
<td>82</td>
</tr>
<tr>
<td>4.11</td>
<td>Determining two possibilities for a point in a plane.</td>
<td>85</td>
</tr>
<tr>
<td>4.12</td>
<td>Modelled part.</td>
<td>89</td>
</tr>
<tr>
<td>4.13</td>
<td>Flow diagramme of Visual Basic software layout.</td>
<td>94</td>
</tr>
<tr>
<td>4.14</td>
<td>Main form with explanations added.</td>
<td>96</td>
</tr>
<tr>
<td>4.15</td>
<td>Programming form with explanations added.</td>
<td>100</td>
</tr>
<tr>
<td>4.16</td>
<td>Robot Control form.</td>
<td>101</td>
</tr>
<tr>
<td>4.17</td>
<td>Connecting form.</td>
<td>102</td>
</tr>
<tr>
<td>4.18</td>
<td>Robot Paths form</td>
<td>103</td>
</tr>
<tr>
<td>4.19</td>
<td>Expert System form.</td>
<td>104</td>
</tr>
<tr>
<td>4.20</td>
<td>Web Watch form.</td>
<td>107</td>
</tr>
<tr>
<td>4.21</td>
<td>Transponder form implementing the transponder tagging system.</td>
<td>110</td>
</tr>
</tbody>
</table>

### Chapter 5

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Generalised communication layout.</td>
<td>115</td>
</tr>
<tr>
<td>5.2</td>
<td>Java applet from Web page.</td>
<td>124</td>
</tr>
</tbody>
</table>

### Chapter 6

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Antenna testing positions.</td>
<td>140</td>
</tr>
</tbody>
</table>
Simulation, control and remote (Internet) communication of an industrial robot in a manufacturing environment.

Chapter 1

1. Introduction

1.2 Thesis Structure

Chapter 2

2. Relevant Concepts and Components

2.1 Visual Basic

2.2 Java

2.3 Expert Systems and Delphi

2.4 Robot Controller Language: Rapid

2.4.1 Rapid software implementation

2.4.1.1 Position Control

2.4.1.2 Path Control

2.5 Internet, Robots, simulation and VRML

2.5.1 Australia Telerobotics

2.5.2 Nasa Telerobotics

2.5.3 WITS (Web Interface for Telescience)

2.5.4 WorkSpace

2.5.5 VRML (Virtual Reality Modelling Language)

2.5.6 Ideal Remote Controlled System

2.6 Forward / Inverse knematics

2.6.1 Algebraic
Chapter 4

4. Robot simulation (PC)

4.1 Robot simulation key components

4.1.1 Simulation

4.1.2 Transformation

4.1.3 Quartenion to Rotational Matrix

4.1.4 Robot Configuration

4.1.5 Inverse Kinematics of IRB 2400

4.1.6 Determining two possible positions in a plane

4.1.7 Hiding the invisible lines on a 3-D wire frame

4.1.8 Part Manipulation

4.1.9 Collision Detection

4.2 Implementation in Visual Basic

4.2.1 Main form

4.2.2 Programming form

4.2.3 Robot Control form

4.2.4 Connect form

4.2.5 Robot Paths form

4.2.6 Expert form

4.2.7 Web Watch form

4.2.8 Transponder form
<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Robot simulation (Internet)</td>
<td>114</td>
</tr>
<tr>
<td>5.1 Java Applet and Java Server implementation</td>
<td>114</td>
</tr>
<tr>
<td>5.2 Simulation Web site</td>
<td>124</td>
</tr>
<tr>
<td>5.2.1 Communication Check</td>
<td>125</td>
</tr>
<tr>
<td>5.2.2 Access Password</td>
<td>125</td>
</tr>
<tr>
<td>5.2.3 Control Options</td>
<td>126</td>
</tr>
<tr>
<td>5.2.4 Part Gripped</td>
<td>127</td>
</tr>
<tr>
<td>Chapter 6</td>
<td></td>
</tr>
<tr>
<td>6. Testing the system</td>
<td>128</td>
</tr>
<tr>
<td>6.1 Local Testing (Visual Basic Simulation)</td>
<td>129</td>
</tr>
<tr>
<td>6.2 Internet Testing (Java applet)</td>
<td>136</td>
</tr>
<tr>
<td>6.3 Transponder System Testing</td>
<td>139</td>
</tr>
<tr>
<td>Chapter 7</td>
<td></td>
</tr>
<tr>
<td>7. Future developments and conclusion</td>
<td>145</td>
</tr>
</tbody>
</table>

References                                   | 147  |
Product Manuals                                | 151  |
Internet references                            | 152  |

Appendix A Experimental set-up wiring diagram and sensors.
Appendix B Finger extension, modelled robotic cell dimensional layout, part and part locator.
Appendix C Transponder Reader Module, power supply and antenna wire diagram.
Appendix D Communication to Reader Module details.
Appendix E Expert system rule base.
1. **Introduction**

The focus of this study rests upon the remote control and visualisation of a robotic cell, either by a PC connected to the cell or via a Web browser on the Internet. The ability to have an on-line visual representation of the robotic cell together with control over it, especially over the Internet, have many advantages. This could for example enable the manager of industrial plants, which incorporate robotic cells to monitor, control and visually observe on-line the status and current action in the robotic cells from the convenience of his office with no other special equipment other than a PC and an Internet connection.

This study utilises an ABB IRB 2400 industrial robot functioning in a cell which incorporates Assembly, Materials handling and Production equipment sections. This whole cell is also simulated by way of a wire frame on PC. The PC is connected to the Robot Controller by means of a Rap Serial link. This link enables the PC to control the robot as well as the cell on-line. The simulation part of the Visual basic program is translated to Java to serve as an applet on a Web page. This applet communicates with the Web server which in turn, communicates with the PC running the simulation package. In this way control via the Internet was established. Rules for the specific cell set-up were created and utilised in an expert system to enable the cell to perform a certain set of manufacturing actions. This expert system adds additional value to the control of the cell since it enables the user on the Internet to perform certain operations within the cell without having any knowledge of the detail programming language of the robot or computer programming language.

The on-line recognition and identification of parts within a robotic cell are becoming a reality. This project also presents an alternative to existing methods like image recognition and standard bar-coding. Passive miniature transponders are attached to the parts being manipulated within the robotic cell. These transponders are operated from an antenna which is attached to the end effector of the robot arm. These transponders offer read/write capabilities and thus contain a description and history of the part it is attached to. While other systems can detect the position of a part, the transponder can supply the exact identity of the part. This recognition and identification system is integrated into an already intelligent robotic cell, in order to advance the level of automation and intelligence within the robotic cell.

Thus the focus of adding a transponder system rests upon the integration and advancement of intelligent decision making within the robotic cell. Complicated image recognition techniques are often
utilised to identify parts and objects requiring huge amounts of computing power to accomplish this in real time. Transponders present a system that can accomplish the above on-line, as well as being cost effective, while providing the exact identity of the part in question very accurately.

The following research objectives were accordingly specified for this project:

- To create an online, 3-D visual representation (simulation) of the robotic cell, together with control over it, locally as well as from the Internet.
- To integrate intelligence, in the form of an expert system, into the robotic cell with which the remote/Internet user can interact.
- To integrate an on-line part identification system (transponder system) into the robotic cell to also add to the level of intelligence in the cell also available to the user to interact with.

In order to accomplish the above mentioned objectives, the following fundamental research issues are covered in this project:

- To mathematically research a way to present an on-line 3-D representation of the robotic cell in such a way that parts and objects can easily be added to the cell and manipulated within the cell.
- The above researched method needed to be implemented firstly on a local PC connected to the Robot Controller in such a way that on-line control could be established. Fundamental research in this regard included the various communication methods as well as the limitations/specifications imposed by the Robot Controller to accomplish the above mentioned. It also included the researched choice of a PC programming language to make the above possible.
- To research a way the above can be presented on the Internet. This included investigating the TCP/IP protocol and the utilisation thereof to accomplish the above. The only solution found, at the time, was by the use of Java.
- Fundamental mathematical research into a non-computational intensive method of hiding the invisible lines on a simple 3-D wire frame model.
- Detailed fundamental research into various inverse kinematics solutions. The inverse kinematics solution was needed because of restrictions imposed by the Robot Controller. This inverse kinematics solution needed to comply with the information and format presented by the Robot Controller as well as the Robot Controller's way of dealing with singularity points as well as axis configurations.
- Fundamental research on transponder systems in general and specifically regarding reader antenna design and transponder antenna design.
CHAPTER 1: INTRODUCTION

- Mathematical fundamental research into a fast non-computational intensive sufficient way of detecting collisions between the various objects in the simulated robotic cell.

- The integration of the above mentioned points to easily run on the average PC (133 MHz Pentium) and easily be interacted with by any Internet user.

By successfully implementing the above mentioned research objectives a unique knowledge contribution to the research field of robotics and on-line control and simulation are made in the following manner. At the time of investigation, a system as described above did not exist on the Internet. Relevant systems that could be found are discussed. The main knowledge contribution is the above mentioned intelligent on-line remote controlled robotic cell with artificial intelligence components, identification components and all of the above mentioned components included with which the remote/internet user can intelligently interact while having a detailed 3-D visual representation of the real robotic cell as well as all the relevant information regarding the current condition in the cell. This cell, although very simple, presents a basis for any on-line remote/internet controlled manufacturing environment.

1.2 Thesis Structure

This thesis is structured in such a way as to first explain and present the reader with the various components, programming and robot languages, related current activities on the Internet and transponder systems. The above are presented in chapter 2 and providers the reader the necessary background to follow the implementation of the above in the following chapters.

Chapter 3 discusses the experimental set-up. The actual hardware used, configuration of different sections and systems are explained to present the reader with the physical/hardware functioning of the project. Chapter 3 thus covers the robotic cell layout, communication and PC layout as well as the transponder system.

The detail implementation of the PC simulation in Visual Basic is presented in chapter 4. All the necessary components and their detail implementation as well as the software layout are presented. Chapter 5 presents the same as chapter 4 but describes in detail the software implementation, regarding the Internet, in Java. It also presents an explanation of the actual Website, hosting the user interface in the form of a Java Applet and the user usage thereof.

Chapter 6 presents three tests used to verify the claimed abilities of the system. This includes tests regarding the Visual Basic software, Internet software (Java) and the testing of the transponder
system. Finally chapter 7 presents some conclusions as well as the readers view for future development in this field.
2. Relevant Concepts and Components

This chapter presents a discussion of the concepts and components used in this project. Since the Internet was used extensively during the study, various Internet pages are referred to. This will also enable future researchers to acquire updated information on the relevant topics. The rest of this chapter will thus provide the reader with the necessary background to follow the implementation of the concepts and components, discussed in the rest of this chapter, in the following chapters. This chapter will also emphasise the very need to find a variety of different systems and components that can be integrated in such a way to accomplish the set of research objectives. It will thus allow the reader to follow the various serious integration problems that were encountered and also the presented solutions thereof.

2.1 Visual Basic 5

Microsoft Visual Basic is a high level programming language for creating Windows-based programmes on Windows 95 and Windows NT Workstations. Visual Basic 5, the most recent release, offers a very fast way of developing applications. This rapid way of application development eliminates the need to learn C++, as well as the intricacies of how Windows works.

Currently three million developers are now creating applications with unprecedented productivity and unrivalled ease using Microsoft Visual Basic technology. Visual Basic 5 also includes controls for Internet and client/server applications. It also includes an optimising native code compiler which enables programmes to load and run faster, as much as a 20 percent improvement on previous versions of Visual Basic. Each programme can be optimised for speed or for size.

The choice of a development language was between Visual C++ and Visual Basic. Since these are the only two languages supported by the manufacturers of RobComm, the link between the Robot Controller and the PC.

Since Visual Basic provides a fast and effective way of creating graphical user interfaces, Visual Basic is chosen as the programming language. Another reason for choosing Visual Basic is because many programmable applications support Visual Basic. Some examples of such applications include Solid Works, a 3-D modeller with add-on applications like finite element analysis and cnc path generation. Other examples are MathCad and Mathematica. Visual Basic is also fully compatible with other Microsoft products like Access and Excell. The use of Visual Basic thus allows the easy integration with other applications in future.
More information about Microsoft Visual Basic and Microsoft Visual C++ can be found on the Web site http://www.microsoft.com and in [9].

2.2 Java

Java was developed as an object-oriented programming language loosely based on C++ that would be processor-independent with a high priority as being a language for network computing. Sun Microsystems formed the JavaSoft group, which has grown in less than three years to more than four hundred people working on Java and Java-related technologies.

Java's syntax is much the same as C++'s, but unlike C, basically everything is an object. Object oriented programming allows new, complex objects to inherit from parent objects. Java objects, like objects the world over, bundle together code and the data that the code works on.

Java can be implemented as a window driven applications the same as any Windows programme or as an applet. Applets are Java programmes that are embedded into HTML (Hyper Text Markup Language) pages on the Internet. These applets can include client/server capabilities.

In order to develop Java applets or Java applications the developer first creates the Java source code. This source is then compiled to byte code, which is stored on a server (or on the local computer). In order to execute a Java applet/application, the user invokes a Java Virtual Machine (VM), which executes the Java byte code. In the case of an applet being imbedded in a HTML page, the Internet user's browser downloads the Web page together with the applet and then starts executing the applet through the built-in Java Virtual Machine. Unlike most other programming languages, Java byte-code is not a platform-specific code native to any particular processor. All Internet browsers, including Netscape, Internet Explorer, HotJava and Java development environments contain Java Virtual Machines which enable the execution of Java applets or applications in the byte code format.

Java is also specifically designed to limit the harm that a Java programme can cause to the system which it is running on. This is especially important on the Internet, where Java applets are downloaded and executed automatically when a user visits a Web page which contains Java applets. For example, this automatic execution might insert viruses on the system it is running on. To prevent such actions Java only allows the reading and writing to storage devices to very restricted specified areas of the system it is running on. The same applies to reading and writing to memory locations.

Not only is Java designed for downloading over the network, standard Java libraries also specifically support client/server computing. Java includes provisions in the language for multi-threading and for network communications. Compared to other languages (such as C), it is much easier to write a pair of
programmes, one executing locally on the user's computer which is handling the user interaction, and
the other executing remotely on a server, which is performing potentially more sophisticated and
processor intensive work.

Because the Java environment is so rich, with literally hundreds of libraries implementing everything
from complex 3-D graphics to cryptography to distributed computing with persistent objects, there is
very little a programmer might want to do that Java is not capable of.

To summarise, the following features makes Java particularly suited as a programming language for
the Internet.

- Java compiles to a platform-independent byte-code, which is typically interpreted by a local
  Java interpreter, or in the case of Java applets, by a Web browser like Netscape.

- Java has built-in language classes, which make communication via the Internet very simple. For
  example, clients which access a company's Web site may download together with Web
  documents a Java applet which could act as the client applet for a client/server which would
  typically also be written in Java. Java can open and access objects across the Net and has an
  extensive library for TCP/IP (Transmission Control Protocol / Internet Protocol) protocols like
  HTTP (Hyper Text Transfer Protocol) and FTP (File Transfer Protocol).

- Part of the Java package is a platform-independent graphical user interface (GUI) class library.
  The local interpreter maps these platform-independent GUI objects onto the corresponding
  API (Application Program Interface) classes (or function calls) of the current operating system.
  Hence, a Java application will look in many ways like any other native application on your
desktop.

- Java has a built-in security system, which makes it extremely difficult to tamper with the
  system. Since the primary application of Java is as programming language for the Internet, this
  issue is extremely important. If users could corrupt the code (by for example inserting a virus
  into the code) there would be a very limited application for Java. Java's security system is
  implemented on class level (every class automatically has it). Before running a class, Java run-
time system performs a series of checks including source code verification and memory access
verification.
The run-time system is very compact. The interpreter occupies only about 40K of memory while support for multi-threading and the standard class library requires another 175K. This makes it even possible for small systems to be able to run Java applications/applets.

Java is very robust in the sense that it does not allow direct access to memory. Java run-time system uses garbage collection to release memory. Hence applications cannot suffer from memory leaks.

Object orientation makes code exchange and re-use very easy. This together with the platform independence makes it very lucrative to share code across the Internet and to use the Internet as one huge class library.

A complete range of products and literature can be found on Sun Microsystems’ Web site at http://www.sun.com. More information on Java can also be found in [8] and [33].

2.3 Expert system and Delphi
An artificial intelligence (AI) system created to solve problems in a particular domain is called an expert system [27, 1]. Knowledge in the expert system is provided by people who are experts in that domain. A standard programme can do everything an artificial intelligence programme can do, but it cannot be programmed as easily or as quickly. For example, a rule can be added to the rule-base of an AI programme without any changes to the rest of the programme. In both types of programmes, all pieces are interdependent in the way they carry out their designed functions. But an AI programme possesses a notable characteristic, which is equivalent to a vital characteristic of human intelligence. Each minute piece can be modified without affecting the structure of the entire programme. This flexibility provides greater programming efficiency and understandability - in a word, intelligence.

In the specific case of an expert system a knowledge base is built from facts, rules, procedures, and objects which relate to a specific problem. During operation the inference engine performs the reasoning process of the human and assists in solving a problem by drawing conclusions based on perceived facts. This is achieved by using information stored as a knowledge base and comparing it with input from the external environment. In this way a large number of possible conclusions are narrowed to select the appropriate solution.
The expert system software used in this study is obtained from Programming Expert systems in Pascal [26]. In order to utilise this software it had to be incorporated into and be compatible with Visual Basic 5, the programming language chosen for the development of the simulation. However, the expert system software is written in Pascal, and thus a way had to be found to incorporate the Pascal programme into Visual Basic 5. This was done by way of a DLL (dynamic-link library).

In Microsoft Windows, dynamic linking provides a way for a process to call a function which is not part of its executable code. The executable code for the function is located in a dynamic-link library (DLL), containing one or more functions that are compiled, linked, and stored separately from the processes using them. DLL’s can be used by all Windows-based programming languages including Visual Basic, Visual C++ and Delphi. The expert system Pascal software is converted to a DLL by making use of Delphi. Delphi is an object-oriented graphical toolset for developing Windows applications in Pascal. Delphi includes the capability to create DLL from Pascal source code. The original expert system Pascal source code was modified to function as a DLL in a Visual Basic environment and compiled to a DLL using Delphi.

In this way expert system software was generated which could be modified and made to fit its purpose in the Visual Basic 5 simulation software. Therefore the purchase of specific expert system software was unnecessary. More information on Delphi can be obtained from the Web site http://www.borland.com.

2.4 Robot Controller Language: RAPID

The ABB IRB 2400 Robot Controller uses the RAPID programming language to control the robot and peripheral equipment in a specific way. The Rapid programming language is programmed in much the same way as any high level programming language like C, Java or Pascal, incorporating a main routine supported by subroutines and functions. Rapid consists of a very wide range of standard functions, subroutines and data types of which only those used in this study will be explained and discussed. The following explains the data types used:

Data Type robtarget

Robtarget is used to define the position of the robot and external axes by specifying the position and orientation of the tool centre point (TCPₐ). As the robot is able to achieve the same position in several
different ways, the axis configuration is also specified. (See section 4.1 for an explanation of the axes configuration).

This defines the axes values if these are in any way ambiguous, for example:

- if the robot is in a forward or backward position,
- if axis 4 points downwards or upwards,
- if axis 6 has a negative or positive revolution

**Robtarget** has the following components: (All data structures in the remainder of this chapter are presented in the following format, first giving the structure name, followed by the purpose or function of the structure and finally the data type of the structure.)

<table>
<thead>
<tr>
<th>Structure</th>
<th>Purpose</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans</td>
<td>translation</td>
<td>Data type: pos</td>
</tr>
</tbody>
</table>

The position (x, y, and z) of the tool centre point expressed in mm. For the purposes of this study the position is specified in relation to the base or world co-ordinate system at the base of the robot (See chapter 4 and Figure 4.2).

<table>
<thead>
<tr>
<th>Structure</th>
<th>Purpose</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rot</td>
<td>rotation</td>
<td>Data type: orient</td>
</tr>
</tbody>
</table>

The orientation of the tool, expressed in the form of a quaternion (q1, q2, q3 and q4). (See section 4.13 for an explanation of a quaternion). For the purposes of this study the position is specified in relation to the base or world co-ordinate system at the base of the robot (See section 4.1.3).

<table>
<thead>
<tr>
<th>Structure</th>
<th>Purpose</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>robconf</td>
<td>robot configuration</td>
<td>Datatype: confdata</td>
</tr>
</tbody>
</table>

The axis configuration of the robot (cf1, cf2, cf6 and cfx). This is defined in the form of a quarter revolution of axis 1, axis 4 and axis 6. The component cfx is not used at present. (For more details see section 4.1.4).

<table>
<thead>
<tr>
<th>Structure</th>
<th>Purpose</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>extax</td>
<td>external axes</td>
<td>Data type: extjoint</td>
</tr>
</tbody>
</table>
The position of the external axes. Since no external axes exist, this data type is not used.

The Data type structure of robtarget is thus as follows where num indicates a value of type real:

```
< dataobject of robtarget >
  < trans of pos >
    < x of num >
    < y of num >
    < z of num >
  < rot of orient >
    < q1 of num >
    < q2 of num >
    < q3 of num >
    < q4 of num >
  < robconf of confdata >
    < cf1 of num >
    < cf4 of num >
    < cf6 of num >
    < cfx of num >
  < extan of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
```
Data Type `jointtarget`

`jointtarget` is used to define the position of the robot and external axes by specifying the position of each individual axes. Thus to convert from the `robtarget` format to the `jointtarget` format the inverse kinematics algorithm of section 4.1.5 is needed.

`jointtarget` has the following components:

- **robax**
  
  robot axes

  Data type: `robjoint`

  Axis position is defined as the rotation degrees for the respective axis (arm) in a positive or negative direction from the axis calibration position. Thus 6 values for the six axes.

- **extax**
  
  external axes

  Data type: `extjoint`

  The position of the external axes. Since no external axes exist, this data type is not used.

The Data type structure of `jointtarget` is thus as follows where `num` indicates a value of type `real`:

```
< dataobject of jointtarget >
  < robax of robjoint >
    < rax_1 of num >
    < rax_2 of num >
    < rax_3 of num >
    < rax_4 of num >
    < rax_5 of num >
    < rax_6 of num >
  < extax of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
```
Data Type speeddata

Speeddata is used to specify the velocity at which both the robot and the external axes move. Speeddata defines the velocity:

- at which the tool centre point moves,
- of the reorientation of the tool,
- at which linear or rotating external axes move.

When several different types of movement are combined, one of the velocities often limits all movements. The velocity of the other movements will be reduced in such a way that all movements will finish at the same time. The velocity is also restricted by the performance of the robot. This differs, depending on the type of robot and path movement.

Speeddata has the following components:

\[ v_{tcp} \quad \text{velocity TCP}_R \quad \text{Data type: num} \]

The velocity of the tool centre point (TCP\(_R\)) in mm/s

\[ v_{ori} \quad \text{velocity orientation} \quad \text{Data type: num} \]

The velocity of reorientation about the TCP\(_R\) expressed in degrees/s.

\[ v_{leax} \quad \text{velocity linear external axes} \quad \text{Data type: num} \]

The velocity of linear external axes in mm/s
The velocity of rotating external axes in degrees/s.

The Data type structure of speeddata is thus as follows where num indicates a value of type real:

\[
\begin{array}{c}
< \text{dataobject of speeddata} > \\
< \text{v_tcp of num} > \\
< \text{v_ori of num} > \\
< \text{v_leax of num} > \\
< \text{v_reax of num} > \\
\end{array}
\]

A number of speeddata values are already defined in the Robot Controller and are used within this study. These values are presented in Table 2.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>TCP speed</th>
<th>Orientation</th>
<th>Linear ext. axis</th>
<th>Rotating ext. axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>v5</td>
<td>5 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v10</td>
<td>10 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v20</td>
<td>20 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v30</td>
<td>30 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v40</td>
<td>40 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v50</td>
<td>50 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v60</td>
<td>60 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v80</td>
<td>80 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v100</td>
<td>100 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v150</td>
<td>150 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v200</td>
<td>200 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v300</td>
<td>300 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v400</td>
<td>400 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v600</td>
<td>600 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v800</td>
<td>800 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v1000</td>
<td>1000 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v1500</td>
<td>1500 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v2000</td>
<td>2000 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v2500</td>
<td>2500 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>v3000</td>
<td>3000 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
<tr>
<td>vmax</td>
<td>5000 mm/s</td>
<td>500°/s</td>
<td>5000 mm/s</td>
<td>1000°/s</td>
</tr>
</tbody>
</table>

Table 2.1. System defined speed data value.
Data Type `zonedata`

`zonedata` is used to specify how a position is to be terminated, i.e. how close to the programmed position the axes must be before moving towards the next position. A position can be terminated either in the form of a stop point or a fly-by point. A stop point means that the robot and external axes must reach the specified position (stand still) before programme execution continues with the next instruction. A fly-by point means that the programmed position is never attained. Instead, the direction of motion is changed before that position is ever reached. For the purposes of this study a system pre-programmed value of `fine` which corresponds to a stop point is always used.

Data Type `tooldata`

`tooldata` is used to describe the characteristics of a tool, e.g. a welding gun or gripper.

`tooldata` affects the robot movements in the following ways:

The tool centre point (TCP) refers to a point which will satisfy the specific path and velocity performance. If the tool is re-oriented or if co-ordinated external axes are used, only this point will follow the desired path at the programmed velocity.

The load of the tool is used to control the robot's movements in the best possible way. An incorrect load can for example cause overshoot.

Programmed positions refer to the position of the current TCP and the orientation in relation to the co-ordinate system. This means that if for example, a tool is replaced because it is damaged, the old programme can still be used if the tool co-ordinate system is redefined.

`tooldata` has the following components:

- `robhold` (robot hold) Data type: bool
  - Defines whether or not the robot is holding the tool
    - TRUE Robot is holding the tool.
    - FALSE The robot is not holding the tool, i.e. a stationary tool.
The tool co-ordinate system is defined as follows:
- The position of the TCP_R (x, y and z) in mm.
- The tool directions, expressed in the form of a quaternion (q1, q2, q3, and q4).
(See section 4.1.3 for an explanation of quaternion).

Both the position and rotation are defined using the wrist co-ordinate system (See Figure 2.1). If a stationary tool is used, the definition is defined in relation to the world co-ordinate system. If the direction of the tool is not specified, the tool co-ordinate system and the wrist co-ordinate system will coincide. A control hole in axis 6 of the robot indicates the -X direction.

![Figure 2.1 Definition of tool co-ordinate system.](image)

The load of the tool defined as follows:
- The weight of the tool in kg.
- The centre of gravity of the tool (x, y and z) in mm
- The axes of moment of the tool, expressed as a quaternion (q1, q2, q3, q4)
- The moment of inertia of the tool about the axes of moment \( x \), \( y \) and \( z \), expressed in \( \text{kgm}^2 \). If all components are defined as being \( 0 \ \text{kgm}^2 \), the tool is handled as if it was a spot load.

The Data type structure of tooldata is thus as follows where \( \text{num} \) indicates a value of type real and \( \text{bool} \) indicates a value of type boolean:

\[
\begin{align*}
\text{< dataobject of tooldata >} \\
\text{< robhold of bool >} \\
\text{< tframe of pose >} \\
\text{< trans of pos >} \\
\text{< x of num >} \\
\text{< y of num >} \\
\text{< z of num >} \\
\text{< rot of orient >} \\
\text{< q1 of num >} \\
\text{< q2 of num >} \\
\text{< q3 of num >} \\
\text{< q4 of num >} \\
\text{< tload of loaddata >} \\
\text{< mass of num >} \\
\text{< cog of pos >} \\
\text{< x of num >} \\
\text{< y of num >} \\
\text{< z of num >} \\
\text{< axm of orient >} \\
\text{< q1 of num >} \\
\text{< q2 of num >} \\
\text{< q3 of num >} \\
\text{< q4 of num >} \\
\text{< ix of num >} \\
\text{< iy of num >}
\end{align*}
\]
For the purposes of the experimental set-up, the current gripper is defined in the tool variable as follows:

\[
\text{FERS tooldata tool1} := \left[ \text{TRUE}, \left[ [0, 0, 0], [1, 0, 0, 0] \right], \\
[1.2, [0, 0, 50], [1, 0, 0, 0], 0, 0, 0] \right];
\]

- The robot is holding the tool
- The TCP_R and tool co-ordinate system coincide with the wrist co-ordinate system.
- The tool weighs 1.2 kg
- The centre of gravity is located at a point 50 mm along the Z axis.
- The load can be considered a spot load, i.e. without any moment of inertia.

For purposes of positioning the robot, two Rapid positioning instructions, MoveL and MoveAbsj were used.

**MoveL**

MoveL is used to move the tool centre point (TCP_R) linearly (in a straight line) to a given destination. When the TCP_R is to remain stationary, this instruction can also be used to re-orient the tool.

For example, the instruction

\[
\text{MoveL p1, v1000, fine, tool1};
\]

will move the TCP_R of tool1 from the current position in a straight line to position p1 defined in the robotarget format. The speed of the movement defined as speeddata v1000 will be 1000 mm/s and the tool will stop at p1 exactly because of the zonedata setting of fine.

**MoveAbsj**

MoveAbsj is used to move the robot to an absolute position, defined in axes positions. The final position of the robot, during a movement with MoveAbsj, is neither affected by the given tool nor the
current position of the robot. The robot axes moves to the destination position along a non-linear path in which all the axes reach the destination position at the same time.

For example the instruction

\[
\text{MoveAbsJ p2, v1000, fine, tool1}
\]

will move the robot axes from the current positions, simultaneously to the positions defined in \( p2 \) which is defined as a jointtarget variable. The speed of the movement is defined as speeddata \( v1000 \) and the robot axes will stop at \( p2 \) exactly because of the zone data setting of fine.

The Rapid programming language also utilises all the standard IF, FOR and WHILE loops found in all high-level programming languages.

2.4.1 Rapid Software implementation

2.4.1.1 Position Control

While developing the system, one of the main objectives was to have as much control as possible on the PC which could communicate in a fast and efficient way, with as little as possible data transfer, with the Robot Controller.

In order to control the position of the robot from the PC, two possible methods were investigated. The first method involves the generation of complete Rapid programmes on the PC after which these programmes are downloaded to the controller to be executed. Thus for every movement of the robot a programme would be generated and downloaded for execution. This method was implemented and tested. The results were not very favourable for the on-line position control since it produced a delay of between 1.5 to 2.5 seconds, between the command given for a new position and the time the robot starts moving. Thus a more efficient and quicker way had to be found to accomplish the above.

The second method involves the updating of certain variables, existing in a Rapid programme running on the Robot Controller, from the PC. The Rapid programming language makes provision for persistent variables which can be updated while the programme containing these variables is running on the Robot Controller. Thus if a Rapid programme containing only one robot position instruction in which the position variable is defined as persistent, is executed repeatedly, the robot can be positioned
by changing this persistent variable from the PC. This method of control was implemented and produced a robot response time of 0.3 seconds which is much faster than the previous method. This method is used and will be discussed in detail in the following section.

This method involves the use of a single MoveAbsj instruction in a rapid programme which is repeatedly executed. Figure 2.2 presents this programme.

```rapid
VERSION: 1
LANGUAGE:ENGLISH

MODULE PositionControl

PERS extjoint newpos:=[0,0,0,0,0,0];
PERS jointtarget Jnewpos:=[[0,0,0,0,0,0],[0,9E9,9E9,9E9,9E9,9E9]];
PERS speeddata nspeed:=[1000,30,200,15];

PROC main()
  Jnewpos.robax.rax_1 := newpos.eax_a;
  Jnewpos.robax.rax_2 := newpos.eax_b;
  Jnewpos.robax.rax_3 := newpos.eax_c;
  Jnewpos.robax.rax_4 := newpos.eax_d;
  Jnewpos.robax.rax_5 := newpos.eax_e;
  Jnewpos.robax.rax_6 := newpos.eax_f;
  MoveAbsJ Jnewpos,nspeed,fine,tooll;
ENDPROC
ENDMODULE
```

Figure 2.2 Position control Rapid programme.

The MoveAbsj command is chosen to update the robot position to that of the simulation during manual control from the simulation for a very specific reason. When the robot is positioned with any of the other positioning commands like MoveL, it is very sensitive to singularity points (which will be discussed in detail in section 4.1) and the robot often gets stuck in these points. The reason for this is that all the movement instructions, except MoveAbsj, use the robtarget format to specify the position and orientation. The specified positions then need to be converted to six axes positions by the Robot Controller to exactly specify the movement of the robot. If the robtarget position or the path which the robot follows, passes through a point that cannot be ambiguously converted to six robot axis
positions, an error occurs and the robot stops. An example of such a point is when axis 5 is in the 0° position. By definition axis 4 can now be rotated through 90° and axis 6 can be rotated the same degrees in the opposite direction to maintain TCP_R position and orientation and throughout this process the position of the axes still comply with a single specified point in the robtarget format. (See section 4.1.4 for details). With the MoveAbsJ instruction this problem does not exist since the axis positions are specified.

A Rapid programme has the following format:

First the version and language is specified in:

```plaintext
%%%
VERSION:1
LANGUAGE:ENGLISH
%%%
```

The body of the specific module with a specific name is encapsulated with:

```plaintext
MODULE PositionComatrol

END MODULE
```

The main programme is encapsulated with:

```plaintext
PROC main()

ENDPROC
```

and is found inside the MODULE section. Sub-routines, if any, are also included in the MODULE section. Any declarations of variables to be used is found just after the MODULE statement. For the purposes of position control as previously described the following variables are declared and defined with default values as persistent (PERS): (See Figure 2.2)
Persistent variables can be changed during programme execution from elsewhere in the programme or from the PC through the Rap Serial Link as discussed earlier. However, the Rap Serial Link does not support the changing of jointtarget variables from the PC. Therefore a way had to be found to change jointtarget values from the PC. This is accomplished by changing variables in an extjoint variable, newpos, and then extract the changed components to be duplicated in pre-defined locations within a jointtarget variable Jnewpos.

This is accomplished by the first part of the main programme in Figure 2.2 namely:

\[
\begin{align*}
J_{\text{newpos}.\text{robax}.\text{rax}_1} &:= \text{newpos.eax}_a; \\
J_{\text{newpos}.\text{robax}.\text{rax}_2} &:= \text{newpos.eax}_b; \\
J_{\text{newpos}.\text{robax}.\text{rax}_3} &:= \text{newpos.eax}_c; \\
J_{\text{newpos}.\text{robax}.\text{rax}_4} &:= \text{newpos.eax}_d; \\
J_{\text{newpos}.\text{robax}.\text{rax}_5} &:= \text{newpos.eax}_e; \\
J_{\text{newpos}.\text{robax}.\text{rax}_6} &:= \text{newpos.eax}_f;
\end{align*}
\]

Finally the robot is moved to the desired position by specifying the six axes positions by the instruction

\[\text{MoveAbsJ Jnewpos,nspeed, fine, tool1;}\]

which includes a persistent speeddata variable, nspeed that can also be changed from the PC through the link. To prevent the movement of the robot to the default values during the first execution of the programme, the programme is first loaded into the Robot Controller at which point the persistent variables are already active. These variables are then set to the current robot position before the programme is started. Thus while the programme in Figure 2.2 is executed repeatedly the robot position and speed can on-line be changed from the PC through the Rap Serial Link. The manipulation of the persistent variables through the Visual Basic programme is discussed in section 4.2.
2.4.1.2 Path Control
Whereas the Position Control used the non-linear positioning instruction MoveAbsJ, path execution needs to employ linear as well as non-linear movement of the robot.

```
%%%
VERSION:1
LANGUAGE:ENGLISH
%%%

MODULE RobControl

PERS robtarget lnewpos:=[[959.79,361.78,881.54],[0.205449,0.142248,0.951251,0.180770],[0,0,0,1],[0E+09,0E+09,0E+09,0E+09,0E+09,0E+09]];
PERS speeddata nspeed:=[1000,30,200,15];
PERS jointtarget Jnewpos:=[[0,0,0,0],[0,9E9,9E9,9E9,9E9,9E9]];

PROC main()
  IF lnewpos.extax.eax_a = 2 THEN
    lnewpos.extax.eax_a := lnewpos.extax.eax_b;
    MoveL lnewpos,nspeed,fine,tooll;
  ENDIF
  IF lnewpos.extax.eax_a = 1 THEN
    Jnewpos.robax.rax_1:= lnewpos.trans.x;
    Jnewpos.robax.rax_2:= lnewpos.trans.y;
    Jnewpos.robax.rax_3:= lnewpos.trans.z;
    Jnewpos.robax.rax_4:= lnewpos.rot.q1;
    Jnewpos.robax.rax_5:= lnewpos.rot.q2;
    Jnewpos.robax.rax_6:= lnewpos.rot.q3;
    lnewpos.extax.eax_a := lnewpos.extax.eax_b;
    MoveAbsJ Jnewpos,nspeed,fine,tooll;
  ENDIF
ENDPROC
ENDMODULE
```

Figure 2.3 Path Control Rapid programme

To accomplish the above, MoveAbsJ as well as MoveL instructions were incorporated with the accompanying persistent variables employing the same method as used in Position Control. All the pre-programmed paths existing on the PC’s hard drive were constructed using only the instructions
MoveAbsj and MoveL. An interpreter recognising these commands and variables, is written in Visual Basic and incorporated in the simulation software.

After interpreting each instruction line of a path, the corresponding variable values are extracted together with information whether it is a MoveL or MoveAbsj instruction, and transferred to the persistent variables on the programme running on the robot, which then executes the portion of the path contained in the specific instruction. The simulation software then waits for the robot to execute the instruction by continuously reading the robot position until it reaches the endpoint of the MoveL or MoveAbsj instruction. Figure 2.3 presents the Rapid programme utilised for this purpose.

Three persistent variables were used, Inewpos of type robtarget to be used by MoveL, Jnewpos of type jointtarget to be used by MoveAbsj and nspeed of type speeddata to set the speed. As previously mentioned the link does not support the writing of persistent variables of type jointtarget, thus the same method as in Position Control, is employed. The variables for both MoveL and MoveAbsj are written from the PC to the persistent variable Inewpos. The first variable of the currently unused external axes section, Inewpos.extax.eax_a, is checked for a value of 2. If this is the case then the programme knows that it needs to execute a MoveL instruction with Inewpos as position and orientation. Inewpos.extax.eax_a is then reset to the unused default value which is equal to Inewpos.extax.eax_b after which the MoveL instruction is executed.

If the value of Inewpos.extax.eax_a was at first equal to 1 then the programme knows that the MoveAbsj instruction needs to be executed. The corresponding values for Jnewpos are then extracted from Inewpos and Inewpos.extax.eax_a is reset as in the above case after which the MoveAbsj instruction is executed.

Thus by repeatedly executing the above programme and writing the speed and instruction type and position through the corresponding persistent variables nspeed and Inewpos pre-programmed paths on the PC’s hard drive can be executed. The Visual Basic implementation can be found in section 4.2.5.

By using two Rapid programmes, one for Position Control utilising MoveAbsj and one for Path Control utilising MoveAbsj, as well as MoveL instead of only the latter provides the additional advantage that if the robot gets stuck on a singularity point and the programme is automatically stopped, the Position Control programme can be started and the robot can be move out of the singularity.
CHAPTER 2: RELEVANT CONCEPTS AND COMPONENTS


2.5 Internet, Robots, VRML and Simulation

Today the Internet is expanding exponentially, which could never be imagined a few years ago when it originated (http://www4.mids.org/growth/internet/index.html). Figure 2.4 presents a graph of the Internet hosts with IP addresses since 1987.

![Internet Growth since 1987](image)

**Figure 2.4 Internet growth since 1987**

The Internet is also becoming more and more important as an information as well as an advertising medium. Recently the Internet has also been extending towards the remote control of various physical devices. This varies from remote surgery, reading of remote sensors to the remote control of real robots.

This section will focus on the research and some commercial products available regarding current developments in remote control and simulation which also incorporates the Internet. The Web site, http://piglet.cs.umass.edu:4321/robotics.html, **Robotics Internet Resource Page**, contains pointers to robotics-related information from various sources on the Internet. This Web site contains a
comprehensive list including commercial, academic and research Web sites and activities. The following is a brief discussion on some of the important activities on the Internet related to this study.

2.5.1 Australia Telerobotics 

The University of Western Australia has an ABB IRB 1400 industrial robot on-line on the Internet (http://telerobot.mech.uwa.edu.au/). This set-up offers a user from the Internet the ability to build, by the use of building blocks, certain structures and perform certain functions. Feedback to the user is given in the format of 4 grey scaled pictures from 4 different cameras mounted on different positions in the robotic cell. By observing the changes of the last update on the pictures, the next position can be given to the robot. This project is in collaboration with the Nasa Telerobotics Programme. However, the visual feedback requires large amounts of data to be transferred in order to update the pictures. The movement of the robot is relative and subject to judgement which requires a reasonable amount of practice on the user's side.

2.5.2 Nasa Telerobotics 

The NASA Space Telerobotics Programme (http://ranier.oact.hq.nasa.gov/telerobotics.html) is an element of NASA's ongoing research programme, under the responsibility of the Office of Space Science (formerly located in the Spacecraft Systems Division of the Office of Space Access and Technology). The programme is designed to develop telerobotic capabilities for remote mobility and manipulation, by merging robotics and tele-operations and creating new telerobotics technologies. Space robotics technology requirements can be characterised by the need for manual and automated control, non-repetitive tasks, time delay between operator and manipulator, flexible manipulators with complex dynamics, novel locomotion, operations in the space environment, and the ability to recover from unplanned events. To meet these needs, the programme is focused on the following goal:

"To develop, integrate and demonstrate the science and technology of remote manipulation such that by the year 2004, 50% of the EVA(Extravehicular Activities)-required operations on orbit and on planetary surfaces may be conducted telerobotically."

It can thus be seen that the remote control of space related robots and manipulators is currently one of the major researching fields.
2.5.3 WITS

WITS (Web Interface for Telescience) (http://mars.graham.com/wits/) is a simulation system for the Mars Pathfinder Sojourner rover. WITS has been developed by NASA's Jet Propulsion Laboratory to enable scientists to participate in planetary rover missions from anywhere in the world using the Internet. WITS will be used in the year 2001 rover mission to Mars to generate commands to the rover, and is used in this Pathfinder mission as a public outreach tool.

2.5.4 WorkSpace

WorkSpace 4 is the latest release from Robot Simulations Ltd (http://www.rosl.com/index.htm) which offers a PC-based software system for 3-D Simulation, Calibration and Off-line programming which operates on Windows 95 and NT platforms. With this software a whole proposed robotics cell can be simulated and rendered in 3-D. The cell can then be analysed and changed until satisfactory. The software also offers off-line programming and the on-site calibration of robots. The success of this product proves that the analysis and simulation of any cell before realising is becoming the norm throughout the world. Trading in North America the company estimates the simulation market in this region alone to be US $ 30 -35 million.

2.5.5 VRML

VRML (Virtual Reality Modelling Language) a draft specification for defining three-dimensional environments on the World Wide Web and is increasingly being used on the Internet. VRML allows the 3-D modelling of virtual worlds. With the most recent release of VRML 2.0 the user can interact and navigate through these 3-D worlds. 3-D objects in these 3-D worlds can be linked to other media types on the Internet, such as HTML, sound and video.

Unlike programming languages such as C++, VRML does not have to be compiled and run. Rather, VRML files get parsed and then displayed. Since this is a much faster process, the creation of VRML files is much simpler than programming. It also allows for more interactivity and facilitates incremental improvements. With the use of "plugins" users can view .wrl files embedded in Web pages with browsers such as Netscape and Internet Explorer. Whereas HTML provides the user with a 2-D interface, VRML provides a much richer 3-D interface which enables the user to view or interact with a 3-D object from any angle or view. The following Web site gives a list of VRML related sites http://www.vruniverse.com/vrspace.html.
VRML are currently receiving much attention regarding the use as a simulation and integration tool. This is true, especially regarding the field of robotics. The following Web site, http://www.op.dlr.de/FF-DR-RS/Joerg.Vogel/Vrml/lib.html, provides a comprehensive list of everything related to VRML and Robotics around the world.

An example of an interactive VRML robot model can be found on the Web site, http://www.op.dlr.de/FF-DR-RS/STUDENTS/Martin.Rohrmeier/robot/robot.html. This represents the VRML modelling of a robot with which the user can interact in such a way that the robot can be dragged by the end-effector, while updating the different links, to any position in the robot’s workspace. This simulation includes the inverse kinematics to update the different links.

VRML can be integrated with JAVA which produces a very powerful combination. Using this combination a VRML scene can be automatically altered from outside. Thus, in the case of robotics a very realistic 3-D rendered VRML model of a robot can be updated to the real robot’s position or can determine the real robot’s next position. An example of such an attempt can be found on the Web site http://www.vruniverse.com/vrml2/CAreminder.html

Some examples of real robots on the Internet already exist with various functions. A list of real robots on the Internet can be found on the Web site, http://ranier.oact.hq.nasa.gov/telerobotics_page/realrobots.html. It is therefore clear that the integration of robotics, simulation and control on the Internet is becoming a reality

2.5.6 Ideal remote controlled System
The ideal remote control system can be described as a system which can accurately be remote controlled with a user interface which presents the actual remote system as close as possible incorporating real time updates and feedback on any commands and actions performed by the user. Currently products like Workspace offers the representation of a robotic cell, as a 3-D simulation which compares very well with the actual cell. The Australian Telerobotics robot offers direct control of the robot but with slow feedback to update the pictures and with low accuracy regarding the placing of the robot. If the remote system can be 3-D modelled, including the parts in the cell, the user can specify the adjustment of the positions of these objects, the remote system can perform these
adjustments or actions and as feedback only send the adjusted positions of the objects to be updated in the 3-D modelled simulation used by the user.

Since all the positions in the modelled 3-D environment is known the user can specify the exact new location of each object. In addition, if all the parts in the cell can be electronically labelled, each part can be accompanied by its history regarding tolerances, material type, treatment, position etc. All this information can be transferred to the user to give an even more accurate description of the conditions in the remote system. The remote system should also incorporate intelligence in the form of artificial intelligence (AI), expert systems, fuzzy logic[34], neural networks[23] etc. to enable the user to interact with the remote system by instructing it to perform complex actions by specifying simple instructions. Intelligence will also enable the remote system to act on certain requests by drawing conclusions and thus advising the remote user on possible alternatives or explaining why certain requests cannot be met. This should also include actions taken upon unplanned events.

Currently many of these components of the remote control system already exist like real-time four dimensional collision detection [5], fuzzy logic in robot control [34], automated part assembly [29,25] to only mention a few. Thus, the challenge today is to combine all these different tools and components to establish a system which can interact intelligently with a remote user, presenting to the user a close representation of the actual system. The aim of this study is the establishment of a basis for such a system.

2.6 Forward / Inverse manipulator kinematics

Regarding manipulators only having rotational joints, the forward kinematics solution provides the solution to calculate the tool centre point (TCP\textsubscript{R}) and orientation from the given rotational position of each axis of an N degree of freedom (DOF) manipulator. This solution is very simple and can be found in any textbook related to robotics, such as [6,14,17] and [20]. Forward kinematics is simple, because a set of joint angles specifies exactly one TCP\textsubscript{R} position and orientation.

The inverse kinematics solution involves the calculation of the position of each rotational axis from a given TCP\textsubscript{R} position together with its orientation. Inverse kinematics, however, is difficult because most real systems are underconstrained, thus for a given goal position, there could be infinite solutions (i.e. many different joint configurations could lead to the same endpoint).

The field of robotics has developed many inverse kinematics systems which, due to their constraints, have closed-form solutions. Solving the inverse kinematics is one of the major research areas in robotics. This is due to the need for more complex manipulators with N DOF. There are three main
methods for solving inverse kinematics, namely algebraic, geometric and iterative. According to [12], algebraic and geometric methods provide closed-form solution for a given kinematics chain but some manipulators do not have closed-form solutions. Iterative methods provide generalised solutions to inverse kinematics but converges to one solution space only where a kinematics chain usually have multiple solutions. Also, algebraic and geometric methods provide real-time computation of inverse kinematics and find all solution configurations [28].

2.6.1 Algebraic
Detailed steps toward an algebraic solution to the PUMA 500 manipulator can be found in [28,21] and [6]. To solve inverse kinematics algebraically, it is necessary to solve equations for $\theta_1...\theta_N$ for N DOF.

The problem can be formulated as in equation 5.1, given the end-effector frame (the desired TCP_R position and orientation),

$$A_{i-1}^{i} (\theta_i) = \begin{bmatrix} N_x & O_x & A_x & P_x \\ N_y & O_y & A_y & P_y \\ N_z & O_z & A_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the right hand side describes the required position and orientation of the end-effector. The problem comes down to solving N equations for N unknowns [21].

This method does not guarantee a closed-form solution for a given manipulator. Thus, mechanical engineers usually design simple manipulators where closed-form solution exist. [6,15] and [16] proposed a generalised closed-form solution which can be derived for 6 (or less) DOF kinematics chain.

2.6.2 Geometric
As opposed to an algebraic method, a closed-form solution using the geometry of the manipulator is derived. [14] used theorems in co-ordinate geometry which can be found in [2] to derive the closed-form solution for a six DOF manipulator. This involves projecting of the link co-ordinate frames onto the $x_{i-1}$ and $-y_{i-1}$ frame. This method can be applied to any manipulator with known geometry. One limitation of this method is that the closed-form solution for the first three joints must exist.
CHAPTER 2: RELEVANT CONCEPTS AND COMPONENTS

geometrically [7]. Apart from that, the closed-form solution for one class of manipulators cannot be used in other manipulators of a different geometry [6].

2.6.3 Iterative

This method solves inverse kinematics by iteratively solving for the joint angles. This method is another research area in robotics. Generally this method is slower and converges to only one solution as opposed to the two methods already presented [12]. There are three components which constitute iterative methods, namely the Jacobian, pseudoinverse and minimization methods (for non-linear system).

2.7 Transponder Tagging and System Selection

A relatively new concept for identifying items by miniature radio frequency (RF) tags, called transponders, are being implemented extensively. Interrogators placed in the proximity of these tags read the information contained within the tag and can even write to the tag in order to change certain portions of the memory contents in the tag. Multiple tags can be co-located in the RF field, generated by the interrogator. Thus, performing high-speed inventories of multiple items within a RF field becomes a reality.

The following are the major properties by which a transponder can be classified:

- transponder power source
- electrical or magnetically field operated
- operating frequency
- transponder size
- transponder operating distance
- reader antenna size
- ability to read multiple transponders at the same time
- transponder cost
- transponder functions

Transponders can receive power, in order to operate, from two possible sources. First the transponder can have an onboard power source, such as a battery. This however, is not a convenient option since it influences the size of the transponder. Depleting is also a problem, therefore an unreliable power
source arises. However, an onboard power source provides a greater operating range. The other manner of generating power is by using the interrogating signal to energise the transponder. Radio frequency energy is “beamed” at the transponder. The radio frequency energy is sufficient to power the transponder and cause the transponder to modulate its contents back to the interrogator emitting the radio frequency energy. This option makes the transponder completely passive when not in use and therefore more reliable but with a smaller operating range.

Transponders are operated either by a magnetic or electric field. The former method is used for the popular 100 - 135 kHz frequency passive transponders with operating distances up to 1 metre. The electric field is used for operating distances of up to 13 m when used with a power source (battery) on the transponder. Other frequencies in use range from 915 to 2450 MHz (http://electronics.pnl.gov:2080/rftag/rfcdetails.html) and 1.9 MHz to 8.2 MHz (http://rapidttp.com/transponder/sulpcros.html) and soon in the 2.45 to 5.6 GHz range (http://rapidttp.com/transponder/trends.html)

Transponder size/volume is closely related to the operating range. Usually the size/volume is determined by the transponder antenna and not by the microchip present on the antenna. Transponder antennas can be in the form of a coil, for example in pet identification (http://www.dfw.net/tqg/electronicid/tx14001.html) or as a flat antenna like in the credit card type.(http://207.87.1.43/identification/). Various other types encapsulated in different shapes for different purposes are available and growing daily.

The reader antenna size is also very closely related to the operating range. The larger the antenna, the larger the reading range. For example, to obtain a reading range of 1 metre with a passive 100 to 135 KHz transponder, an antenna of 50 cm by 90 cm is needed. In the above case the reader antenna needs to create a magnetic field which surrounds the transponder, for the transponder to modulate its response.

Currently many of the transponders available commercially include a system which can simultaneously read multiple transponders within the range of the reader. This adds a great advantage since, from a variety of transponders in the reading range, the transponders can be identified and interaction/processing can be done with each transponder individually. An example of such an
application is in the case of bus fare collection. The entrance to the bus is equipped with a reader antenna and each passenger is in possession of a transponder mounted on a credit card. Without taking the card from wherever the passengers are carrying them, the busfare is automatically deducted from the transponder card as the passengers passes the reader antenna.

Currently transponders are produced that not only supply an identification number when interrogated by the reader, but also allows the multiple reading and writing of information from and to the transponder. The communication between the reader and transponder is also encrypted to avoid unauthorised interference from outside. This function opens many doors for application in the marketplace. Examples include the electronic purse, the adding of history and other information on transponder tagged manufactured goods to only name a few.

Transponder cost is certainly the most important factor since it determines the economic viability of the transponder. This is even more important when the transponder is not re-used, thus sold with the article it is identifying. Currently small, low cost transponders range in price from $0.25 to $0.50 when bought in large volumes, which makes them very attractive to implementation. Transponders are believed to replace the old bar-coding system. In comparison with standard barcode identification transponders allows the identification of an object on which the transponder is not visible. An example of this is the Trolleyponder system (http://www.trolleyscan.co.za/) in which a whole supermarket trolley full of articles can be scanned once to obtain the identity of all the articles in the trolley. Up to 400 articles can be read at the same time. This identification is done up to a very high accuracy and in a very short time. It is thus clear that transponders have a distinct advantage over bar-code systems in which one visible article can be scanned at a time.

Transponders are currently used extensively in the following fields, to only name a few:

- Electronic Article Surveillance (EAS)
- Shipping Container and railcar tracking
- Animal tracking
- Vehicle access
- Personnel access
- Production control
2.7.1 System selection

The on-line recognition and identification of parts within a robotic cell is becoming a reality. This study incorporates an alternative to existing methods, such as image recognition and standard bar-coding. Passive (dormant) miniature transponders are attached to the parts being manipulated within the robotic cell. These transponders are operated from an antenna which is attached to the end effector of the robot arm. These transponders offer read/write capabilities and thus contain a description and history of the part it is attached to. While other systems can detect the position of a part, the transponder can supply the exact identity of the part. This recognition and identification system is integrated into the already intelligent robotic cell in order to advance the level of automation and intelligence within the robotic cell. A detailed description of the integration of the transponder system into the robotic cell can be found in section 3.8.

Various types of transponder systems are available for all kinds of applications as mentioned above. For the purposes of the study a system which offers read/write, identification abilities, as well as local support from the suppliers, was needed. Phillips's new HITAG range
(http://207.87.1.43/identification/products/contactless/) was chosen and the reader module together with the application development software and transponders were bought from them. An antenna was built by implementing the manufacturer’s specifications on the desired size of the antenna. See section 3.8 for antenna parameter calculations. Figure 2.5 presents the layout of the transponder system.

The Phillips/Mikron HITAG system uses a magnetic field, generated by the antenna, operating at 125 kHz to communicate with the transponders. The HITAG1 transponders offers 2 kbit of multiple read/write memory as well as an unchangeable unique serial number. Depending on the antenna size, the transponders can be operated from a maximum distance of 1 meter. The Reader Module communicates with the transponders through the antenna and to the PC via a RS 232 serial communication line. The Reader Module offers certain functions to be used by the PC to operate the transponder through the Reader Module. Functions available include the detecting of various types of transponders in the antenna field, the selecting of transponders for read/write operations, as well as the handling of communication when multiple transponders in the antenna field exist to prevent communication collisions. Phillips claims to be able to sell these transponders in large volumes at $0.50 which makes them commercially very attractive. The system is accompanied by C libraries which facilitates the development of customised user software. All communication, from the PC to the Reader Module as well as from the Reader Module to the transponders is checked to ensure a very high data transfer accuracy.

All the different types of antennas proposed by the manufacturer were investigated which included the frame antenna, flat cable antenna, aluminium tube antenna and transformer antenna. The flat cable antenna was chosen because of the ease of construction from ordinary computer flat cable. The advantages of the flat cable antenna can be summarised as follows:
2.7.1.1 Resonance

A very important principle in the construction of antennas is the principle of resonance [11,38]. Figure 2.6 presents the series circuit diagram of the flat cable antenna and cable, tuning capacitance and reader module. In the below series circuit the inductance L is generated by the winding of the antenna coil. The resistance R present the resistance found in the internal resistance of the cable used to generate the antenna coil, as well as the internal resistance in the cable, connecting the reader module and antenna.

![Antenna coil and cable diagram](image)

Figure 2.6 Flat cable antenna circuit diagram.

To achieve resonance in the above series circuit the reactance of $C_{\text{tune}}$ and the reactance of L must be equal. At this point the voltage drop across L and $C_{\text{tune}}$ respectively is equal and 180° out of phase and is characterised by a specific resonant frequency. Therefore, the voltages cancel each other completely and the current flow is determined wholly by the resistance, R. At this point the current also has the largest possible value. In the antenna circuit of Figure 2.6, the frequency generated by the reader module is fixed at 125 kHz. Also fixed is the physical construction of the antenna and cable is fixed. In order to make the above circuit resonate, the only variable left, $C_{\text{tune}}$ needs to be varied until resonance is reached. Thus for resonance the following applies:
\[ X_L = 2 \cdot \pi \cdot f \cdot L = X_C = \frac{1}{2 \cdot \pi \cdot f \cdot C_{tune}} \]  

(2.1)

where

\[ f \]  = frequency in hertz

\[ L \]  = inductance in henrys

\[ C_{tune} \]  = tuning capacitance in farads

\[ X_L \]  = reactance of L

\[ X_C \]  = reactance of \( C_{tune} \)

From equation 2.1 it can be verified that the tuning capacitance, \( C_{tune} \), can be calculated by:

\[ C_{tune} = \frac{1}{2 \cdot \pi \cdot f \cdot X_L} \]  

(2.2)

In reality it is not only the inductance of \( L \) which needs to be taken into account in the calculation of the reactance of the antenna, but also factors including the interwinding capacity and proximity effect of the interwinding cables in the antenna coil. The cable from the antenna to the reader module also adds an additional internal capacitance, known as the \( C_P \) value, which normally varies between 40 to 100 pF/m for different kinds of Coaxial-Cable. To aid the estimation of \( C_{tune} \), the manufacturers of the transponder system provides guidelines, for each type of antenna, to calculate an effective series reactance, \( X_{eff} \), which include some of the above-mentioned relevant factors. These calculations for the specific flat cable antenna used in the experimental set-up appears in section 3.8. \( C_{tune} \) can then be estimated, by the use of equation 2.2, as follows:

\[ C_{tune} = \frac{1}{2 \cdot \pi \cdot f \cdot X_{eff}} \]  

(2.3)

This estimation does not guarantee resonance and the antenna needs to be tuned manually by varying the value of \( C_{tune} \) around the calculated value. In practice this means that an oscilloscope must be corected across points A and B of Figure 2.6 (as close as possible to the reader module) and the voltage must be measured. The current flowing through the circuit also needs to be measured and can be measured via a current probe connected to the same oscilloscope. If no current probe is available a
1 Ω resistor can be added into the circuit in series and the voltage across this resistor can be measured. The voltage displayed on the oscilloscope represents the current in the circuit since \( V = I \cdot R \). Since at resonance the voltages over the inductor and capacitor cancels each other and the current flow is determined by the resistor alone, the voltage across points A and B and the current in the circuit should be in phase at resonance. This point should also present the largest current possible. Thus, the value of \( C_{\text{tune}} \) is varied while the Reader Module is operational until the current and voltage across points A and B are in phase.

The manufacturers of the specific Reader Module specifies a value of 200mA and 7 V at resonance.

2.7.1.2 Quality Factor \( Q \)

When maximum sharpness or selectivity of antenna signals is needed, the objective of design is to reduce the inherent resistance to the lowest possible value. The resistance in the antenna is equivalent to the energy loss. The effective reactance, \( X_{\text{eff}} \) in a series antenna circuit divided by the series resistance, \( R_S \) in the same circuit, is called the quality factor, \( Q \), or:

\[
Q = \frac{X_{\text{eff}}}{R_S} \tag{2.4}
\]

where

\( Q \) = quality factor

\( X_{\text{eff}} \) = effective reactance of the antenna circuit without \( C_{\text{tune}} \)

\( R_S \) = effective series resistance of antenna circuit without \( C_{\text{tune}} \)

The manufacturers also supply the tolerable \( Q \) ranges for the different antennas. See section 3.8 for the calculations. More information can be found in the Phillips Antenna Design manual [38].
3. Experimental Set-up

Figure 3.1 Communication layout.
3.1 Communication layout

This chapter presents the physical implementation of the robotic cell. The detail communication methods between the different components and PC are discussed in detail. Also included in this chapter is the implementation of the reader and transponder antennas with the corresponding calculations. The pre-programmed robot paths and the integration of these paths into the expert system are also presented. Three sections within the experimental set-up, namely Production Equipment, Assembly and Material Handling, is defined and explained. The aim of this chapter is thus to give the reader a detailed overview on the physical components and operation of the experimental set-up. Figure 3.1 presents the communication layout of the whole system.

The robot and controller used in the set-up is a six axis ABB IRB 2400 industrial robot. The controller contains the programming language, Rapid, used to programme the robot and functionalities to connect to outside digital inputs and outputs, as well as facilities to interface to a remote PC through the Rap Serial Link. This remote link enables the remote user to perform the following:

- Downloading and Uploading programmes from and to the PC.
- Executing Rapid programmes
- Stoping and starting the robot.
- Update Persistent Variables in a Rapid programme while executing.
- Receiving error messages and task messages from the robot.
- Read the status of the robot in detail.
- Read the position of the robot in detail.
- Reading and writing digital input and output values.

The robot is equipped with a Barry Wright Corporation ASTEK GTP-45-3X S/N 011 three finger pneumatic gripper. Each finger is equipped with an extension which incorporates a limit switch which switches on if pressure is applied to the specific finger indicating that the specific finger did grip something. The finger extensions also incorporates an adjustment mechanism to be adjusted for different parts. Detail drawings of the finger extensions and gripper appears in Appendix B. Figure 3.6 presents a photograph of the gripper.

The controller allows for 16 digital inputs and 16 digital outputs to which the sensors and other equipment were connected. The sensors consists of 3 infra-red beams and 9 limit switches. The infra-
red beam units are used to detect the presence of parts in certain locations. Circuit diagrams and specifications of the infra-red units appear in Appendix A. 6 of the 9 limit switches are used to determine the presence of parts in certain locations. The other three serve on the gripper as described above. Two digital outputs were used. One to open and close the gripper, by way of a pneumatic valve and the other to stop and start the simulated lathe (electric motor).

A full wiring and connecting diagram of the integration of the digital inputs and outputs, as well as the Rap Serial link to the controller are presented in Appendix A. The transponder system is discussed later.

3.2 PC configuration

Each PC connected to a network and which has access to Internet is identified by a unique IP address. Accompanying this IP address is a common name for the PC. For example, the controlling PC’s IP address is 152.106.20.211 with the common name of DIESEL. See Figure 3.1 which includes a layout of the PC configuration. In order to operate and test the complete system three PC’s were used. The first, the Controlling PC (Diesel) is directly connected to the Robot Controller via the Rap Serial Link. This PC contains the simulation software which has complete on-line status and control over the robotic cell. It also incorporates the expert system, as well as handles the requests and commands from users on the Internet. A serious problem encountered with the Rap Serial Link was that, as soon as the Controlling PC is connected to the Robot Controller, any TCP and UDP connections from elsewhere to the Controlling PC is cut off. This poses a very critical problem since this isolates the Controlling PC and Robot Controller from communicating through a network. However, it was discovered that the hard drive of the controlling PC can still be shared by other computers on the same local area network (LAN). Thus, another computer on the LAN can gain read and write access to the controlling PC’s hard drive by mapping this hard drive as a network drive. The PC chosen to connect to Diesel, Nagasaki with IP address 152.106.20.187, also served as a Web server on which the Web page with the robotic cell interface is stored. In order for Nagasaki to communicate to Diesel the following system is devised. Two files on Nagasaki’s hard drive, which is mapped as a network drive to Diesel, is used to communicate. The first file, s_read.dat, is continuously read by Diesel to detect any changes in the content, which would indicate incoming information from Nagasaki, and the second s_write.dat is used by Diesel to write new information which needs to go to Nagasaki. On the other side Nagasaki is continuously reading s_write.dat to detect changes which would indicate incoming information from
Diesel, and is writing any information which needs to be sent to Diesel in s_read.dat. Collisions while reading the same file from two different PC simultaneously, are automatically prevented by the PCs. This method of communication proved to work very well. Microsoft Front Page is used as a Web server on Nagasaki.

By running Microsoft Front Page Web server on Nagasaki, Nagasaki can post Web pages on the Internet. However, a system that would present the Internet user with an interface and capabilities similar to the simulation software running on Diesel is needed. This interface should also communicate with the simulation software which serves as the brain of the robotic cell and the whole system.

The only way the above could currently be accomplished is by using Java. Java provides the ability to write programmes, called applets, which can be put inside a Web page and which can also communicate directly to the Web server (Nagasaki). The visual simulating part of the simulation running on Diesel was translated to Java accompanied by certain predetermined control abilities to the user. The above was compiled into an applet and placed inside a Web page accompanied by instructions on how to use the interface. This Web page is posted on Nagasaki with Internet address http://nagasaki.rau.ac.za/rob_web.

Also running on Nagasaki is server software receiving the information from the applet on the Web page and writing it to s_read.dat, as well as reading s_write.dat for changes in the content, in which case the contents are sent to the applet.

This server software is also written in Java, since Java is platform independent, not to limit the operating system of the Web server to Windows 95 or Windows NT. The protocol used for the applet communication, as well as for the Web page retrieval by the user is the standard TCP/IP protocol. Details regarding the server/applet software is presented in section 5.1.

By implementing the above set-up, communication, integration and interfacing from the Internet to the simulation software then to the robotic cell, was made possible. Also connected to Diesel, via a RS 232 communication line, is the Phillips/Mikron transponder system. Details of this system is discussed later in this chapter.

3.3 Robotic Cell layout

The experimental robotic cell layout will be discussed in sections regarding Assembly, Production Equipment, Materials handling and Transponder Tagging. This specific set-up was chosen as a practical specific case of a more general generic concept incorporating the above four sections. A
specific expert system to deal with the specific experimental set-up was created to work towards a goal of producing finished assembled parts within the cell. A set of rules were drawn up to work towards the goal which represents single actions in the robotic cell. This rule base is used in the expert system to conclude an action towards the goal. The expert system receives its inputs from the sensors, limit switches and infra-red beams. The expert system is run repeatedly, while executing an action with each repetition, until the goal is accomplished in which case all the parts are machined and assembled in the finished positions. The expert system can be started with the parts in any configuration or position. The experimental set-up uses pre-defined positions with pre-defined paths connecting these positions. These positions are schematically indicated in Figure 3.3. A photograph indicating the positions can be found in Figure 3.2. While at a certain position, the robot is only allowed to move to the next position connected by a path which would correspond to a single action commanded by the expert system. Therefore, only one path can be executed at a time provided the current position is at the beginning or end of the path. In Figure 3.3 the directions of the paths are indicated. Paths can therefore also be executed in reverse.

Figure 3.2 Photograph of robotic cell with predefined positions indicated.
The cell can also be manipulated manually in a pre-programmed path fashion or by way of specifying the different axis angles of the robot. Specific detail is presented in section 4.2.

![Diagram of robotic cell paths and positions.](image)

Numbers 1 to 11 indicate the path numbers with the direction included.

Specific positions in the experimental set-up are indicated by A to HT.

**Figure 3.3 Schematic layout of robotic cell paths and positions.**

### 3.4 Assembly

Figure 3.4 presents more detail of the Assembly section. Four identical parts are used in the experiment. For the purpose of the expert system, which will be discussed later, the parts are initially in positions C, D, E and F, where their presence is detected by limit switches. If activated the expert system guides the cell through a process where each part is taken from its initial position, through a machining process and brought back to the Assembly section. At the Assembly section the first finished part is placed in position G, the second is assembled on top of the first, the third is placed in
CHAPTER 3: EXPERIMENTAL SET-UP

position H and the fourth is assembled on top of the third. Limit switches detect the presence of parts at positions C to G. The presence of assembled parts two (GT) and four (HT) is detected by infra-red beams as indicated on Figure 3.4. Position B serves as an entry position to the Assembly section.

![Assembly Diagram](image)

3.5 Production Equipment

Figure 3.5 presents more detail of the Production Equipment section. The indicated electric motor simulates a machining process. The parts are brought here, put on the shaft of the motor, released, gripped again and then taken away. An infra-red beam detects the presence of a part on the motor shaft. Position I indicates the part position on the motor shaft and position J serves as the entry position to the Production Equipment section.
3.6 Material Handling

This section includes the robot and gripper as indicated in Figure 3.6. The gripper consists of three fingers. Each finger includes a limit switch which would indicate when a force is applied to that finger. A part is properly gripped if all three limit switches are closed. Position A is used as a starting position for the robot. Figure 3.6 indicates the rubber band which is used to keep the limit switches in the open position if no part is gripped.
3.7 Expert System

An expert system is incorporated in order for the cell to work towards a goal. As previously stated, the goal of the expert system is to produce, through the robotic cell, two assembled parts from the input of four unassembled and not machined parts. The way in which the expert system operates, is by collecting information from all the sensors, positions and conditions after which the rule base is consulted to produce a conclusion. This conclusion is implemented in the robotic cell as a single action. Due to this action, some of the sensors, positions or conditions will change. The expert system is then run again with the new values to yet conclude another conclusion and implementing the corresponding action in the cell. This process is repeated until the goal, of producing two assembled parts, is met or until an error occurs. Figure 3.7 presents a flow diagram of the process to obtain the goal. This flow diagram is implemented in the form of 75 rules in a rule base used by the expert system. By referring to the diagram, a conclusion is reached if a value for sysresult can be found.
Figure 3.7 Flow diagram of rule base.
The following is an explanation of the actions resulting from the corresponding value for `sysresult`. Refer to Figure 3.3 for an explanation of path and position locations.

- **RunPathX**: Moves the robot, currently at the start of Path X to the end of Path X.
- **RunPathXR**: Moves the robot currently at the end of Path X to the start of Path X, thus Path X in reverse.
- **Close_GripperN**: Closes the Gripper to grip a new unmachined part.
- **Close_GripperF**: Closes the Gripper to grip a machined part.
- **Open_Gripper**: Opens the Gripper.
- **errorX**: Stops the robot and expert system and display error X.

The following are the sensors positions and conditions, which serve as variables in the expert system, with an explanation of each:

- **Robot_At**: The robot position is read and matched with predetermined positions A to J. (See Figure 3.3) If the robot is not within a certain tolerance from any of these positions, Robot_At receives the value of `nowhere`.
- **PROD_PartInMachine**: The value of infra-red sensor 1. (See Appendix A and Figure 3.4) The value of `yes` indicates the presence of a part in the Lathe in the Production Equipment section. `No` indicates that there is no part in the Lathe.
CHAPTER 3: EXPERIMENTAL SET-UP

Gripper_Closed

Reads the value of the current signal to the Gripper’s Pneumatic valve. A value of yes indicates a closed Gripper and a value of no, an open one.

Part_Gripped

Reads the value of Gripper Limit Switch 1 to 3. If all limit switches are closed the value of yes is given to Part_Gripped, else the value of no is allocated.

ASSEM_PartInPlaceX

Reads the X corresponding Limit switches C to H or Infra-red HT or GT. If the corresponding limit switch is closed or the corresponding Infra-red relay is closed, a value of yes is allocated. Else a value of no is allocated.

PartIn_Gripper

PartInGripper value is set by previous actions concluded by the expert system. The action Close_GripperN changes PartInGripper to the value of new and the action Close_GripperF changes it to finished. PartIn_Gripper starts with the value of no.

The complete rule base, as used by the expert system, is found in Appendix E.
3.8 Part Tagging and Antenna design

As discussed in section 2.7, the antenna type chosen is the common flat cable antenna, generated from ordinary flat computer cable with the following properties:

- Conducting diameter \( cd = 0.32 \text{ mm} \)
- Resistance per meter \( R_m = 0.21 \Omega/\text{m} \)
- Distance between two conductors \( s = 1.27 \text{ mm} \)
- Specific capacity between two conductors \( C_s = 50 \text{ pF/m} \)

The following procedure is used to calculate the various antenna parameters in order to ensure the correct and effective operation of the antenna as described by the Phillips manual [38]:

- First the operating range and size of the antenna are chosen and determined by using Figure 3.8 and Figure 3.9.
- Next the number of turns or windings within the antenna are determined with the aid of equation 3.1.
- The following is then calculated:
  - Antenna Inductance with the aid of equation 3.2
  - Antenna direct current resistance with the aid of equation 3.3
  - Antenna interwinding capacity with the aid of equation 3.4
  - Antenna effective series resistance with the aid of equation 3.5
  - Antenna effective series reactance with the aid of equation 3.6
- The above values are then used to calculate the quality factor of the antenna with the aid of equation 3.7
- The values for the quality factor and effective series reactance are checked against the tolerated range for these values presented on the graph in Figure 3.11.
- If the above values are within the tolerable range, an estimation of the tuning capacitance is calculated with the aid of equation 3.8.
- The antenna is manually tuned, while operating, starting at the above tuning capacitance value until the resonating conditions are met.
3.8.1 Antenna size versus Transmission range

Figure 3.8 and 3.9 present graphs of the transmission range $x$ versus antenna radius $r$ implicating performance of the HITAG card and disc transponders respectively. In these graphs the antenna is assumed to be round.

**Figure 3.8 Antenna radius versus Transmission range for card transponders.**

**Figure 3.9 Antenna radius versus Transmission range for disc transponders.**
A conversion formula is available to relate the chosen radius to a rectangular antenna. However, for the specific experimental set-up, a round antenna, which would fit around the pneumatic gripper, is chosen. In order not to interfere with the other equipment, the antenna radius is chosen as 0.12 m which, according to Figure 3.8 and Figure 3.9, would give a transmission/operating range of 0.36 m and 0.3 m respectively for the card and disc transponders which should be sufficient for detecting transponders in the vicinity of the gripper.

The next parameter to be selected is the density of the magnetic flux generated by the antenna. A minimum value, \( B_{\text{min}} \), is needed to activate the various transponder types. For the card type transponders the value is 740 nT and 1300 nT for the disc type transponders. To make the system more versatile, the value of 1300 nT is chosen to allow the antenna to be able to operate both types.

Thereafter, the number of turns or windings in the antenna are calculated to give the above magnetic flux with the following formula:

\[
N_A = \frac{2 \cdot B_{\text{min}} \cdot (r^2 + x^2)^{3/2}}{\mu_0 \cdot I_{\text{Ant}} \cdot r^2}
\]  

(3.1)

where

\( N_A \) = Number of turns

\( B_{\text{min}} \) = minimum magnetic flux density - 1300 nT

\( r \) = antenna radius - 0.12 m

\( x \) = transmission range - 0.3 m

\( I_{\text{Ant}} \) = antenna current - 200 mA

\( \mu_0 \) = permeability = \( 4\pi \times 10^{-7} \) As/Vm

thus

\[
N_A = \frac{2 \cdot 1300 \cdot 10^{-9} (0.12^2 + 0.3^2)^{3/2}}{4 \cdot \pi \cdot 10^{-7} \cdot 0.2 \cdot 0.12^2}
\]

\[= 24.23 \equiv 25 \text{ turns} \]

### 3.8.2 Antenna Inductance

The inductance of the antenna is then calculated according to the following formula.

\[
L = 8 \cdot N_A^2 \cdot a \cdot \left[ \ln\left(\frac{a}{N_A \cdot s} \right) + 0.224 \cdot \frac{s \cdot N_A}{a} + 0.726 \right] + 4 \cdot N_A \cdot a \quad \text{nH, cm} \quad (3.2)
\]
where

\[ N_A = \text{number of turns} - 25 \text{ turns} \]

\[ a = \text{width/length of a quadratic antenna} \]

In the case of this round antenna

\[ a = \sqrt{\text{Antenna Area}} = \sqrt{\pi \cdot r^2} = \sqrt{\pi \cdot 0.12^2} = 0.21 \text{ m} = 21 \text{ cm} \]

\[
L = 8 \cdot 25^2 \cdot 21 \cdot \left[ \ln\left( \frac{21}{25 \cdot 0.127} \right) + 0.224 \cdot \frac{0.127 \cdot 25}{21} + 0.726 \right] + 4 \cdot 25 \cdot 21 \quad \text{nH, cm}
\]

\[ = 0.28 \cdot 10^6 \text{ nH} \]

### 3.8.3 Direct current resistance

The direct current resistance of the antenna can be calculated by multiplying the length of conducting cable in the antenna by the resistance per meter value \( R_m \).

The following formula describes the direct current resistance \( R_{DC} \).

\[
R_{DC} = 2 \cdot \pi \cdot r \cdot N_A \cdot R_m = 2 \cdot \pi \cdot 0.12 \cdot 25 \cdot 0.21 \Omega = 3.96 \Omega \quad (3.3)
\]

As mentioned earlier, the flat cable antenna shows no proximity effect. Therefore, the direct current resistance of the antenna is equal to the ohmic resistance of the antenna. Thus,

\[ R_{DC} = R_{ohm} \]

### 3.8.4 Interwinding capacity

The interwinding capacity of the antenna \( C_w \) is now calculated with the following formula:

\[
C_w \equiv \frac{U}{N_A^2} \cdot \frac{C_s}{3} \cdot f(N_A) \quad (3.4)
\]

where

- \( C_w \) = interwinding capacity of a flat cable antenna in pF
- \( U \) = circumference of antenna in m
- \( C_s \) = specific capacity between two conductors of the flat cable in pF/m given in the manufacturers specification of the flat cable.
- \( N_A \) = number of turns in the antenna
$f(N_A) = \text{function of the number of turns plotted in Figure 3.10. This function is needed to estimate the parasitic capacity of flat cable antennas. The function can be calculated by } f(N_A) = \sum_{n=1}^{N_A} n^{1.25} \cdot n^{0.0095 \cdot n}$

$C_w \equiv \frac{2 \cdot \pi \cdot 0.12}{25^2} \cdot \frac{50}{3} \cdot 1100 \ \text{pF}$

$= 22.1 \ \text{pF}$

Figure 3.10 Function $f(N_A)$ of the number of turns of the flat cable antenna.
3.8.5 Effective series resistance

The effective series resistance, $R_s$, can now be calculated with the following formula:

$$ R_s = \frac{R_{\text{ohm}} \cdot \frac{L}{C_w} - \frac{R_{\text{ohm}}}{\omega \cdot C_w} \cdot (\omega \cdot L - \frac{1}{\omega \cdot C_w})}{(R_{\text{ohm}}^2 + (\omega \cdot L - \frac{1}{\omega \cdot C_w})^2)} $$

where

$R_s =$ effective series resistance in $\Omega$

$R_{\text{ohm}} =$ ohmic resistance in $\Omega$

$C_w =$ interwinding capacity of antenna in pF

$L =$ antenna inductance in nH

$\omega = 2 \cdot \pi \cdot \text{Freq} = 785398$ where $\text{Freq}$ is the resonant frequency of 125 kHz

Thus

$$ R_s = \frac{3.96 \cdot 0.28 \cdot 10^{-3}}{22.1 \cdot 10^{-12}} - \frac{3.96}{785398 \cdot 22.1 \cdot 10^{-12}} \cdot (785398 \cdot 0.28 \cdot 10^{-3} - \frac{1}{785398 \cdot 22.1 \cdot 10^{-12}}) $$

$$ = 3.99 \Omega $$

3.8.6 Effective series reactance

The effective series reactance $X_{\text{eff},S}$ can also be calculated with the following formula:

$$ X_{\text{eff},S} = -\frac{R_{\text{ohm}}^2 + \frac{L}{\omega \cdot C_w} \cdot (\omega \cdot L - \frac{1}{\omega \cdot C_w})}{R_{\text{ohm}}^2 + (\omega \cdot L - \frac{1}{\omega \cdot C_w})^2} $$

where

$X_{\text{eff},S} =$ effective series reactance in $\Omega$

$R_{\text{ohm}} =$ ohmic resistance in $\Omega$

$C_w =$ interwinding capacity of antenna in pF

$L =$ antenna inductance in nH

$\omega = 2 \cdot \pi \cdot \text{Freq} = 785398$ where $\text{Freq}$ is the resonant frequency of 125 kHz
Thus

\[
X_{\text{eff},S} = \frac{3.96^2}{785398 \cdot 22.1 \cdot 10^{-12}} + \frac{0.28 \cdot 10^{-3}}{22.1 \cdot 10^{-12}} \cdot \left( \frac{785398 \cdot 0.28 \cdot 10^{-3}}{785398 \cdot 22.1 \cdot 10^{-12}} - \frac{1}{785398 \cdot 22.1 \cdot 10^{-12}} \right) \Omega
\]

\[
= 220.7 \Omega
\]

### 3.8.7 Quality factor

The quality factor of the antenna can now be calculated by the following formula as derived in section 2.7

\[
Q_L = \frac{X_{\text{eff},S}}{R_S} = \frac{220.7}{3.99} = 55.3
\]

(3.7)

According to the graph in Figure 3.11, it is verified that the quality factor, 55.3, and effective series inductance, 220.7 Ω are within the tolerable range.

All that remain is to tune the antenna. The following equation is used to calculate an estimate of the tuning capacitance, \(C_{\text{tune}}\), to ensure a resonating antenna circuit:

\[
C_{\text{tune}} = \frac{1}{\omega \cdot X_{\text{eff},S}} = \frac{1}{785398 \cdot 220.1} = 5.78 \text{ nF}
\]

(3.8)

The above calculated capacitance is connected in series with the antenna circuit of Figure 2.5 and is manually varied, while the antenna is operational, until the antenna current and voltage, as observed on an oscilloscope, are in phase. The antenna is tuned while attached to 3 m of RG 174-50Ω co-axial cable with \(C_p = 101 \text{ pF/m}\). The above resonating condition is reached with a capacitance of 6.744 nF. Figure 3.12 presents a photograph of the antenna fitted to the pneumatic gripper.
Figure 3.11 Tolerated range of the quality factor in dependence of the effective series reactance.

Figure 3.12 Antenna fitted on pneumatic gripper.
3.8.8 Chip Antenna

The above designed antenna is tested with transponders obtained from a local manufacturer. Although these transponders served as a testing method for the antenna, they are moulded into a credit card size card which is too large to be attached to the parts in the robotic cell. Thus, a transponder had to be manufactured which could be used on the parts.

Each transponder consists of two parts, a micro-chip and an antenna. The same principles of a resonant circuit, which was applied to the antenna-reader circuit, applies to the transponder circuit. The circuit of the transponder, including the micro-chip and antenna, Figure 3.13, can be modelled to the circuit diagram presented in Figure 3.14.

\[
\begin{align*}
\text{Micro-chip} & \\
\text{Coil or Antenna} & \\
\end{align*}
\]

**Figure 3.13 Transponder.**

\[
\begin{align*}
L_{pc} & \quad R_{pc} \\
C_{pc} & \quad U_{A-B} \\
C_{chip} & \quad R_{chip} \\
\end{align*}
\]

**Figure 3.14 Equivalent transponder circuit.**

Where

- \( U_{A-B} \): voltage at A-B
- \( f_{\text{res}} \): resonant frequency of transponder (125 kHz)
- \( L_{pc} \): inductance of the coil at \( f_{\text{res}} \)
CHAPTER 3: EXPERIMENTAL SET-UP

**RPCE**  parallel resistance of the coil as \( f_{res} \)

**C_{PC}**  parasitic capacitance of the package

**C_{chip}**  capacitance of the chip

**R_{chip}**  resistance of the chip

**f_{self}**  self resonant frequency of the coil

The following values are supplied by the manufacturer:

\[
\begin{align*}
\text{f}_{\text{self}} & > 750 \text{ kHz} \\
\text{R}_{\text{PC}} & > 45 \text{ k}\Omega \\
\text{C}_{\text{chip}} & = 210 \text{ pF} \pm 10 \% \\
\text{C}_{\text{PC}} & = 1.5 \text{ pF} \text{ hot laminated cards} \\
\text{C}_{\text{PC}} & = 1.5 \text{ pF} \text{ moulded tags}
\end{align*}
\]

From equation 2.1 in section 2.7 it can be verified that the following is true for a series inductor capacitor resonant circuit:

\[
2 \cdot \pi \cdot f \cdot L = \frac{1}{2 \cdot \pi \cdot f \cdot C} \tag{3.8}
\]

where

\[
\begin{align*}
\text{f} & \quad \text{resonant frequency in hertz} \\
\text{L} & \quad \text{inductance in henrys} \\
\text{C} & \quad \text{capacitance in farads}
\end{align*}
\]

From the above equation it can be verified that:

\[
L = \frac{1}{(2 \cdot \pi \cdot f)^2 \cdot C} \tag{3.9}
\]

In the particular circuit of Figure 3.14 the following is applicable:

\[
\begin{align*}
\text{L} & = \text{L}_{\text{PC}} \\
\text{C} & = \text{C}_{\text{chip}} + \text{C}_{\text{PC}}
\end{align*}
\]
Substituting the above in equation 3.9 gives:

\[
L_{pc} = \frac{1}{(2 \cdot \pi \cdot f)^2 \cdot (C_{chip} + C_{pc})}
\]  

(3.10)

The above formula can therefore be used to estimate the inductance of the antenna or coil connected to the micro-chip. Since the completed transponders are not moulded or laminated in plastic, thus \(C_{pc} = 0\);

Thus the coil inductance can be calculated as:

\[
L_{pc} = \frac{1}{(2 \cdot \pi \cdot 125000)^2 \cdot 210 \cdot 10^{-12}} \text{ H}
\]

\[
= 7.72 \text{ mH}
\]

Next the size, wire diameter and number of turns in the coil need to be selected and determined. The following formula is suggested by the manufacturers in calculating the above:

\[
N_A = \left[ \frac{L}{2 \cdot U \cdot (\frac{U}{d} - 1.07)} \right]^{0.5405}
\]

(3.11)

Where

\begin{align*}
N_A & \quad \text{Number of turns} \\
L & \quad \text{inductance of coil (nH)} \\
\text{d} & \quad \text{Diameter of copper wire (cm)} \\
U & \quad \text{average coil circumference (cm)}
\end{align*}

The value of \(L\) is already calculated as 7.72 mH. In order to fit inside the parts (see Appendix B for detailed drawings of the parts) a round antenna with a diameter of 2.5 cm is chosen. This antenna is realised with copper wire having a diameter of 0.1 mm. Therefore, the number of turns is calculated as follows with the aid of equation 3.11:
\[ N_A = \left[ \frac{7720000}{2 \cdot 7.85 \cdot (\ln(\frac{7.85}{0.01}) - 1.07)} \right]^{0.5405} \]

\[ \approx 470 \text{ turns} \]

Where

- \( N_A \): Number of turns
- \( L = 7720000 \) nH
- \( d = 0.01 \) cm
- \( U = \pi \cdot 2.5 = 7.85 \) cm

Figure 3.15 presents a photograph of the realised antenna, constructed around a plastic centre, attached to the micro-chip and fitted inside a part. The above chip/antenna combination suffers from many assumptions in the calculation of the number of turns. This will cause the transponder circuit to be out of resonance. Due to the very small current present in the transponder circuit it is very difficult to measure the phase angle. Thus, the transponder circuit is tuned by adding or subtracting turns on the coil, starting at the above calculated value. The transponder is then repeatedly tested in the antenna field until the maximum reading range is obtained. When tested the transponder seemed to work better if the number of turns are decreased to 460.
Figure 3.15 Constructed transponder fitted inside a part.
4. Robot simulation (PC)

This chapter considers the software development in Visual Basic which is implemented on the PC (Diesel), connected to the Robot Controller (See Figure 3.1). The first section of this chapter, section 4.1, deals with the mathematical side of presenting the 3-D wireframe, hiding the invisible lines, detecting collisions and calculation the inverse kinematic solution. It also explains the Robot Controller’s use of axis configurations and the use of quartenions to indicate orientation. Section 4.2 thus provides the reader with all the mathematical fundamentals in order to follow the software implementation in section 4.2. The rest of the chapter, section 4.2, deals with the Visual Basic implementation of the above mathematical principles. The software is divided into eight forms which each implements a section of the required objectives. The last form also presents the details between the PC and the reader module of the transponder system.

The following describes the software developed to simulate and control an ABB IRB 2400 six axes industrial robot from a PC. The purpose of this software is the following:

1. To create a 3-D simulation which represents the real robot.
2. The simulation should have the same degree of freedom and movement as the real robot in order to simulate the real robot as close as possible.
3. The user must be able to view the 3-D simulation from any position and also be able to zoom in on any part of the simulation.
4. The simulation must be linked to the real robot, so that the real robot follows the movement of the simulation in real time.
5. The opposite must also be possible, that is if the robot is running any programme the simulation should follow the robot’s movements in real time.
6. The user interface should also allow the user to perform all tasks, normally performed at the robot console. This includes operations like:

   - Downloading programmes from the PC.
   - Stop and starting the robot.
   - Creating programmes for the robot with the aid of the simulation.
   - Receiving error messages and task messages from the robot.
   - To read the status of the robot in detail at any time.
From the above it is clear to see that this software attempts to "move" the real robot to the PC from which it can be viewed in 3-D, monitored and controlled. This gives the user the ability to remotely, without any visual contact with the real robot, operate the robot incorporating as many of the real robot's abilities as possible.

In order to establish the communication link between the real robot and the PC, a communication link called RobComm 2.1, was bought from the suppliers of the robot. This link supports Microsoft Visual Basic 5 as well as Microsoft Visual C++. RobComm 2.1 communicates with the Robot Controller via a Rap Serial Link also bought from the suppliers of the robot.

4.1 Robot simulation key components

4.1.1 Simulation

In order to create a realistic 3-D simulation, a way had to be found to represent the real robot as simple as possible with the best visual representation as possible. Since this simulation should function in real time, the speed of updating the simulation is very important and therefore the way in which the simulation is implemented.

Figure 4.1 provides the three different views of the ABB IRB 2400 industrial robot, including the dimensions. Using these dimensions a 3-D wire frame containing 70 points to which 91 lines are connected, was created to represent the IRB 2400. This wire frame is shown in Figure 4.2 and Figure 4.3. The base co-ordinates, which are the same as the world co-ordinates, were chosen as shown in Figure 4.2. together with the choice of the tool co-ordinates. This wire frame was built from 7 building blocks, each containing 10 points or 13 lines (Each block includes two points representing the rotation line of the corresponding axis). The first block represents the fixed base of the robot and then a block for each axis. In order to accomplish the movement of the simulation, a function had to be developed which uses as input the six axes position values in degrees and as output gives the new positions of the 70 points so that the new positions of the 91 lines could be established.

Since all the joints of the IRB 2400 are rotational joints, a function was developed which rotates a specified group of points around a line, specified by 2 points, by a specified amount of degrees. By repeatedly executing this function around the rotation line of each axis, starting at axis 1, the movement of the simulation is accomplished. This function acts as the main part of the simulation movement. The above procedure serves as the forward kinematics solution. The above method of creating the forward kinematics solution is chosen because of its simplicity, which will be demonstrated in the next section, as well as the ease of adding functionalities like the manipulation of
modelled parts and adding a second modelled robot. The following explains the functioning of the above procedure:

Figure 4.1 View of robot from the side, front and above (Measurements in mm).
Figure 4.2 Simulation wire frame with dimensions.
4.1.2 Transformation

In Figure 4.4 the rotational line, around which the specified group of points are to be rotated, is specified by point A and B.

Figure 4.4 Rotation line, AB, is specified by point A an B.
The original co-ordinate system is moved to point A by subtracting A from all the points to be rotated. Figure 4.4 is therefore translated to the representation in Figure 4.5.

A transformation matrix is required which will transform the x-axis to rest on line AB with the positive direction in the direction of point B. Then a transformation rotating the specified points around line AB can be performed. After this the inverse of the first transformation matrix is performed to restore the co-ordinate system to the one in Figure 4.5. From Figure 4.5 it can be seen that:

\[ AC = L_1 = (x_2^2 + y_2^2)^{0.5} \]
\[ AB = L_2 = (L_1^2 + z_2^2)^{0.5} \]
\[ \sin(\theta_2) = y_2/L_1 \]
\[ \cos(\theta_2) = x_2/L_1 \]
\[ \sin(\theta_y) = z_2/L_2 \]
\[ \sin(\theta_y) = L_1/L_2 \]

\[ \theta_x = \text{Specified angle around x-axis} \]

Therefore the following would describe a transformation matrix, \( T_F \), which would use as input the co-ordinate system in Figure 4.5 and would produce a specified rotation around line AB of a specified group of points.
\[ TF = \text{Rot}(z, \Theta_z) \cdot \text{Rot}(y, \Theta_y) \cdot \text{Rot}(x, \Theta_x) \cdot \text{Rot}(y, -\Theta_y) \cdot \text{Rot}(z, -\Theta_z) \]

Where

\[
\text{Rot}(z, \Theta_z) = \begin{bmatrix}
\cos(\Theta_z) & \sin(\Theta_z) & 0 \\
-s\sin(\Theta_z) & \cos(\Theta_z) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\text{Rot}(y, \Theta_y) = \begin{bmatrix}
\cos(\Theta_y) & 0 & \sin(\Theta_y) \\
0 & 1 & 0 \\
-s\sin(\Theta_y) & 0 & \cos(\Theta_y)
\end{bmatrix}
\]

\[
\text{Rot}(x, \Theta_x) = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\Theta_x) & -\sin(\Theta_x) \\
0 & \sin(\Theta_x) & \cos(\Theta_x)
\end{bmatrix}
\]

After multiplication with the aid of MathCad:

\[
\begin{bmatrix}
T11 & T21 & T31 \\
T12 & T22 & T32 \\
T13 & T23 & T33
\end{bmatrix}
\]

Where

\[
T11 = \cos(\Theta_z) \cdot \cos(\Theta_y) \cdot \cos(\Theta_x) - \cos(\Theta_z) \cdot \cos(\Theta_x) \cdot \cos(\Theta_y) + \cos(\Theta_y) \\
T12 = \sin(\Theta_z) \cdot \cos(\Theta_y) \cdot \cos(\Theta_x) - \cos(\Theta_z) \cdot \sin(\Theta_x) \cdot \cos(\Theta_y) \cdot \cos(\Theta_y) \\
T13 = \cos(\Theta_z) \cdot \sin(\Theta_y) - \cos(\Theta_y) \cdot \sin(\Theta_x) \cdot \cos(\Theta_z) - \cos(\Theta_y) \cdot \sin(\Theta_z) \cdot \sin(\Theta_x)
\]

\[
T21 = \sin(\Theta_z) \cdot \cos(\Theta_y) \cdot \sin(\Theta_x) - \cos(\Theta_z) \cdot \sin(\Theta_x) \cdot \sin(\Theta_y) - \cos(\Theta_x) \cdot \cos(\Theta_y) + \cos(\Theta_z) \cdot \cos(\Theta_y)
\]

\[
T22 = \cos(\Theta_y) \cdot \cos(\Theta_x) + \cos(\Theta_z) \cdot \cos(\Theta_y) \cdot \cos(\Theta_x) - \cos(\Theta_y) \cdot \cos(\Theta_x) \cdot \cos(\Theta_z)
\]

\[
T23 = \sin(\Theta_z) \cdot \sin(\Theta_y) - \cos(\Theta_y) \cdot \sin(\Theta_x) \cdot \cos(\Theta_z) - \cos(\Theta_y) \cdot \cos(\Theta_x) \cdot \sin(\Theta_z)
\]

\[
T31 = \cos(\Theta_z) \cdot \cos(\Theta_x) - \cos(\Theta_z) \cdot \sin(\Theta_x) \cdot \sin(\Theta_y) + \cos(\Theta_y) \cdot \cos(\Theta_x)
\]

\[
T32 = \sin(\Theta_z) \cdot \cos(\Theta_x) - \cos(\Theta_z) \cdot \sin(\Theta_x) \cdot \sin(\Theta_y) - \cos(\Theta_z) \cdot \cos(\Theta_y)
\]

\[
T33 = \cos(\Theta_z) \cdot \sin(\Theta_x) + \cos(\Theta_x) \cdot \cos(\Theta_z) \cdot \cos(\Theta_y) - \cos(\Theta_z) \cdot \cos(\Theta_y)
\]
T31 = \cos(\theta_z) \cdot \cos(\theta_y) \cdot \sin(\theta_y) + \cos(\theta_y) \cdot \sin(\theta_z) \cdot \sin(\theta_x) - \cos(\theta_y) \cdot \cos(\theta_x) \\
T32 = \sin(\theta_z) \cdot \cos(\theta_y) \cdot \sin(\theta_y) - \cos(\theta_y) \cdot \cos(\theta_z) \cdot \sin(\theta_x) - \cos(\theta_y) \cdot \sin(\theta_x) \\
T33 = 1 - \cos(\theta_y) \cdot \cos(\theta_x) + \cos(\theta_y) \cdot \cos(\theta_x) \\
\sin(\theta_z) \cdot \sin(\theta_x)

After multiplication by TF the translation is reversed by adding Point A to the specified points.

The above is implemented in the Visual Basic simulation as the function Axisrotate2.

4.1.3 Quaternion to Rotational Matrix

In practice, homogenous transformations are inefficient because computations on matrices require more operations than other representations, and, since their representation of rotations is highly redundant, numerical inconsistencies can be a problem. Thus, quaternions are often used to represent rotations [31].

In general a quaternion Q consists of a scalar s and a vector part v.

\[ Q = [s + v] \]

If two quaternions are multiplied, the result is a new quaternion:

\[ Q_1 \times Q_2 = [s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 + s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2] \]

From this definition, it follows that if

\[ S = \sin(\theta/2) \quad \text{and} \quad C = \cos(\theta/2) \]

Then the quaternion

\[ 0 + \Rot(n, \theta) \times \mathbf{u} = [C + S \cdot \mathbf{n}] \times [0 + \mathbf{u}] \times [C - S \cdot \mathbf{n}] \]

Thus

\[ Q = \Rot(n, \theta) = [C + S \cdot \mathbf{n}] \]

which corresponds to a rotation by an angle \( \theta \) about an axis \( n \), where \( u \) is a unit vector.

The rotational matrix representing the quaternion can be obtained by rotating the points \((1,0,0)\), \((0,1,0)\) and \((0,0,1)\), each a unit vector located on a different axis, around the axis \( n \) by an angle \( \theta \). A transformation matrix, TF, for rotating points around a line was developed in the previous section. Since the above three points each, when multiplied by the transformation matrix represents a column vector in the rotational matrix and since the three points put together as the columns of a matrix is equal to the identity matrix, the rotational matrix is equal to the transformation matrix TF.
The value of \( \sin(\theta_x) \) and \( \cos(\theta_x) \) \((\theta_x = \theta \text{ in matrix } TF)\) is calculated from \( C \) and \( S \).

From the definition of the quaternion \( Q = [s + v] = \text{Rot}(n, \theta) = [C + S \cdot n] \) the following applies:

\[
\sin(\theta_x/2) = S = \sqrt{v_1^2 + v_2^2 + v_3^2}
\]

\[
\cos(\theta_x/2) = C = s
\]

The following procedure is followed to determine \( \theta_x \):

1. \( \theta_x \) is first calculated using \( \theta_x = 2 \sin^{-1}(\sqrt{v_1^2 + v_2^2 + v_3^2}) \)

This does not specify if \( \theta_x \) lies in 0-90° or 90-180°. To determine the correct quadrant, the following is done:

\[
\theta_{x1} = 2 \cos^{-1}(s)
\]

If \( \theta_{x1} < 0 \) then \( \theta_x \) lies in 90-180° else in 0-90°.

The following four variables in matrix \( TF \) still need to be determined:

- \( \sin(\theta_y) \)
- \( \cos(\theta_y) \)
- \( \sin(\theta_z) \)
- \( \cos(\theta_z) \)

Referring to Figure 4.5 where, in this case point \( B = v \) the following is calculated as:

\[
\sin(\theta_z) = y_2/L_1
\]

\[
\cos(\theta_z) = x_2/L_1
\]

\[
\sin(\theta_y) = z_2/L_2
\]

\[
\sin(\theta_y) = L_1/L_2
\]

The rotational matrix, in this case, \( TF \) is now calculated with the above values.

This concludes the Quaternion to Rotational matrix transformation. The above procedure is contained in the sub-routine \( \text{QuartToRot} \) in the Visual Basic simulation.

### 4.1.4 Robot Configuration

All positions of the robot are defined and stored using rectangular co-ordinates. When calculating the corresponding axes positions, there will often be two or more possible solutions. This means that the robot is able to achieve the same position, i.e. the tool is in the same position and with the same orientation, with several different positions or configurations of the robot axes.

To unambiguously denote one of these possible configurations, the robot configuration is specified using three axes values. These values define the current quadrant of axis 1, axis 4 and axis 6. The
quadrants are numbered 0, 1, 2, etc. (they can also be negative). The quadrant number is connected to the current joint angle of the axis. For each axis, quadrant 0 is the first quarter revolution, 0-90°, in a positive direction from zero position; quadrant 1 is the next revolution, 90-180°, etc. Quadrant -1 is the revolution 0°-(-90°), etc. Figure 4.6 represents the quadrant classification graphically.

![Figure 4.6 Configuration Quadrants.](image)

4.1.5 Inverse Kinematics of the IRB 2400

In order to determine the six different axes positions from a given tool centre point (TCP_R) and orientation, it is necessary to have the inverse kinematics solution of the IRB 2400. To develop the inverse kinematics solution, the geometrical method is used. The reason for the choice of the geometrical method is to incorporate in the solution the use of the defined values for the robot configuration as explained in section 4.1.4. In this way the inverse kinematics solution produces a single axis solution to all given TCP_R, orientation and robot configuration values, except at singularity points. Solutions/definitions of the singularity points of the IRB 2400 robot are yet to be defined by the robot manufacturers and will be treated as such for this project.

The following describes the algorithm developed to solve the inverse kinematics:
The IRB 2400 Robot is classified as \( \lambda_0 \phi_0 \lambda \) [20], and schematically represented by Figure 4.7. The lengths of the six different robot links are as follows:

- \( L_1 = 750 \) mm
- \( L_2 = 710 \) mm
- \( L_3 = 125 \) mm
- \( L_4 = 850 \) mm
- \( L_5 = 50 \) mm
- \( L_6 = 50 \) mm

For the remainder of this section all 3-D points are denoted by \( u \) and all 3-D planes by \( p \). The position space of the 6 DOF orientable robot \( \lambda \theta \phi \theta \lambda \) shown in Figure 4.7 is \( u_6, u_7 \) and \( p_4 \). The robot specifies the tool centre point, orientation and robot configuration as follows:

\[
P_1 = [(950.00,0.00,1585.00), (0.707107,0.006060,0.707107,0.0110000), [0,0,0,0,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
\]
Where:

The first three values indicate the TCP_R and are represented by u7.
The next four values represent the quaternion for specifying the tool orientation.
The next four values specify the configuration of axes 1, 4, 6 and an external axis respectively.
The rest is currently unused information for a possible external axis.

First the quaternion is converted, with the procedure QuartToRot, to a rotational matrix. The last column of this matrix, representing a unit vector in the new z-direction, is stored in u8.

The following can be calculated:

\[
\begin{align*}
u_6 &= u_7 - L_6^* u_8 \\
u_5 &= u_6 + L_5( u_6 - u_7 )/L_6 \\
u_2 &= (0,0,750)
\end{align*}
\]

Plane p2 is calculated by specifying point u5, u2 and Origin to lie in the plane by using the function P3_Plane.

\[ p_2 = P3\textunderscore Plane(u5, u2, \text{Origin}) \]

\( \lambda_1 \) is calculated by using u5 in the following equation.

\[
\lambda_1 = \cos^{-1}(u_5x / (u_5x^2 + u_5y^2)^{0.5})
\]

Without taking the physical constraints of the real robot into account, there will always exist two possibilities for \( \lambda_1 \).
The first is if u4 is above the line between u3 and u5, this is defined as Upper u4, and the second is if u4 is below the line between u3 and u5, defined as Lower u4.

The correct configuration can be determined by looking at the configuration of axis 1, O1. If the current value of \( \lambda_1 \) corresponds to O1 then \( \lambda_1 \) is correct, or else if \( \lambda_1 \) is smaller than 0 then \( \lambda_1 = \lambda_1 - 180^0 \) or if \( \lambda_1 \) is greater than 0 then \( \lambda_1 = \lambda_1 + 180^0 \). If \( \lambda_1 \) was not altered, the robot is in the
Upper u4 configuration else in the Lower u4 configuration (See Figure 4.10 for the explanation of Upper and Lower u4). The value of λ1 is checked against the configuration value of axis 1 by using the function Orientation. The above will also ensure that $-180 < \lambda_1 < 180$ which corresponds to the physical limitations of the robot.

Next the position of u3 is determined. Without taking the physical constraints of the real robot into account, there will always exist two possibilities for u3.

![Figure 4.8](image)

Figure 4.8 The two possible positions for u3.

Figure 4.8 represents the two possible positions for u3. The criteria for these two possible points are that both lie in p2 and that the values for L2 and Lt2 are 710 mm and 859.142 mm respectively. ( Lt2 = (L4² + L3²)¹/² ) The function ThreePointInPlane uses this criteria and calculates the two possibilities u3a and u3b. The values for θ2a is determined by calculating the angles between vector u3a - u2 and vector u2 - Origin and for θ2b by calculating the angles between vector u3b - u2 and vector u2 - Origin. These angles are calculated with the function VecAngle.
These angles do not specify if $u_3$ lies in the same quadrant as $\lambda_1$ or in the opposite quadrant, since they are measured from the z-axis. To specify this, the value of $\lambda_1$ is used as follows:

If $u_{3y} < 0$, that is $u_{3y}$ lies in $-180^\circ < u_{3y} < 0^\circ$, and $0^\circ < \lambda_1 < 180^\circ$ then $\theta_2$ is corrected to $-\theta_2$.

Also, if $u_{3y} > 0$, that is $u_{3y}$ lies in $0^\circ < u_{3y} < 180^\circ$, and $-180^\circ < \lambda_1 < 0^\circ$ then $\theta_2$ is corrected to $-\theta_2$.

The above corrections are repeated for $\theta_{2a}$ and $\theta_{2b}$.

Next an elimination process, which consists of two parts, is started. The first part deals with the physical limitations of the robot regarding $\theta_2$. The second part uses the z values of $u_{3a}$ and $u_{3b}$ together with the Upper $u_4$ or Lower $u_4$ configuration of the robot.

The physical limitations for $\theta_2$ are $-109.5 < \theta_2 < 99.5$. The values of $\theta_{2a}$ and $\theta_{2b}$ are checked with this limitation and if they do not fall in the range, a value of $500^\circ$ is allocated to indicate an invalid angle. This completes the first elimination step for $\theta_2$.

For the second part of the elimination process for $\theta_2$ the following criteria is used:

If the robot is in the Upper $u_4$ configuration and $u_{3az} > u_{3b}$ then $\theta_{2b}$ is eliminated.

If the robot is in the Upper $u_4$ configuration and $u_{3az} < u_{3b}$ then $\theta_{2a}$ is eliminated.

If the robot is in the Lower $u_4$ configuration and $u_{3az} > u_{3b}$ then $\theta_{2a}$ is eliminated.

If the robot is in the Lower $u_4$ configuration and $u_{3az} < u_{3b}$ then $\theta_{2b}$ is eliminated.
The angles are eliminated by setting them equal to 500° as was done previously. The above procedure ensured that only one valid value for θ2 remains, which corresponds to one position for u3. The valid angle and position are copied to θ2 and u3 respectively.

With θ2 and u3 determined, the next step is to determine the position of u4. Without taking the physical limitations of the robot into account, two possible positions for u4 exist, taking u3 as determined. Figure 4.10 shows the two possible positions for u4.

The criteria for these two points are that both lie in p2 and that the values for L3 and L4 are 125 mm and 850 mm respectively. The function ThreePointInPlane uses this criteria and calculates the two possibilities u4a and u4b.

To calculate the two angles, the angle, θF, between vector u2 - u3 and vector u5 - u3 is first calculated with the function VecAngle. From Figure 4.10 it is clear that the two angles θ3a and θ3b can be calculated as follows:

\[ θ3a = 180° - θF - θT \]

\[ θ3b = 180° - θF + θT \]
Where \( \theta_T \) is a constant and equal to \( 81.63^\circ \).

The physical limitations of the robot for axis 3 are now applied. The range of axis 3 is \(-50.5^\circ < \theta_3 < 50.5^\circ\).

If the value of \( \theta_3a \) or \( \theta_3b \) fall outside this range the value is set to \( 500^\circ \) to indicate an invalid value, as done before.

\( \theta_3 \) is now set to the valid value of \( \theta_3a \) or \( \theta_3b \).

The choice of \( u_4a \) or \( u_4b \) creates a problem, since \( u_4a \) does not always indicate \( \theta_3a \) and \( u_4b \) does not always indicate \( \theta_3b \). To choose the correct one, another two angles are used.

The angle \( \theta u_4 \) between vector \( u_3-u_2 \) and vector \( u_4a-u_3 \) is calculated with the function \text{VecAngle}.

The following criteria is used to determine if \( u_4a \) is the correct point:

If the absolute value of \( \theta u_4 - \text{absolute value of } \theta_3 \) is smaller than a certain tolerance, \( T \), then

\( u_4 = u_4a \)

The above procedure is repeated for \( u_4b \). This concludes the determination of \( \theta_3 \) and \( u_4 \).

The next step is to calculate \( p_4 \) by using points \( u_7, u_9 \) and \( u_6 \) by using the function \text{P3_Plane}. Next the function \text{InALine} is used to check if points \( u_4, u_5 \) and \( u_6 \) lie in a straight line.

If points \( u_4, u_5 \) and \( u_6 \) lie in a straight line, then the following is done:

\( \phi_4 \) is set to \( 0^\circ \) in the quadrant represented by the configuration of axis 4, \( O_4 \), by use of function \text{AngleWithOrient}.

\( \theta_5 \) is set to zero

The angle \( \theta_6 \) is calculated by calculating the angle between planes \( p_2 \) and \( p_4 \). (See Figure 4.7).

If \( \theta_6 > 90^\circ \) then \( \theta_6 = \theta_6 - 90^\circ \). By doing this, the angle for \( \theta_6 \) in the first quadrant is established.
\( \theta_6 \) is now put in the correct quadrant by using the configuration of axis 6, \( \theta_6 \), and the function \text{AngleWithOrient}.

Else

Calculate \( p_3 \) by using points \( u_4, u_5 \) and \( u_7 \) and the function \text{P3 Plane}.

Calculate \( \phi_4 \) as the angle between \( p_2 \) and \( p_3 \).

\( \phi_4 \) is put in the correct quadrant by using \( \phi_4 \) and function \text{AngleWithOrient}.

\( \theta_5 \) is determined from the angle between vector \( u_5 - u_4 \) and \( u_6 - u_5 \) by using the \text{VecAngle}.

This however, always gives \( \theta_5 \) as positive angle. In order to determine the negative side, point \( u_3 \) is rotated around vector \( u_4 - u_5 \) by \( \phi_4 \) with the aid of function \text{SinglePointRot}. The angle \( \theta_5T \) is calculated as the angle between vector \( u_5 - u_3 \) and vector \( u_6 - u_5 \). The following criteria is used to determine if \( \theta_5 \) is negative:

If absolute value of \((\phi_4 + \theta_5 - \theta_5T)\) is larger than a certain tolerance, then \( \theta_5 \) is negative.

Where \( \phi_4 = \text{constant} = 8.365^\circ \) as indicated on Figure 4.6.

\( \theta_6 \) is calculated as the angle between planes \( p_3 \) and \( p_4 \) by using function \text{VecAngle}.

\( \theta_6 \) is put in the correct quadrant by using the configuration of axis 6, \( \theta_6 \), and the function \text{AngleWithOrient}.

End

This concludes the conversion from TCP\(_R\) and orientation with axis configurations to the six axes values of the robot. The above procedure is implemented in the Visual Basic software as function \text{InverseKin}. 


4.1.6 Determining two possible points in a plane

Figure 4.11 Determining two possibilities for a point in a plane.

The following problem needs to be solved:

Two points \( u_1 \) and \( u_2 \), situated in plane \( p \) are known. The two possible points \( u_3a \) and \( u_3b \) need to be calculated in such a way that \( u_3a \) and \( u_3b \) are at a distance \( L_1 \) from \( u_1 \) and a distance \( L_2 \) from \( u_2 \), as presented in Figure 4.11. To solve the above problem mathematically involves many difficulties if the solution is to be software implemented. The following graphical method is devised to solve the above:

First the length, \( L_3 \), between \( u_1 \) and \( u_2 \) is determined by:

\[
L_3 = |u_1 - u_2|
\]

\( \theta \) can be determined from the cosine rule:

\[
\theta = \cos^{-1}(L_1^2 + L_3^2 - L_2^2)/(2*L_1*L_3)
\]

A point \( T \) is defined as shown on Figure 4.11. \( L_{21} \) is now calculated from

\[
L_{21} = |u_1 - T| = \cos(\theta)*L_2
\]

By using the value of \( L_{21} \), point \( T \) is calculated by

\[
T = u_2 + L_{21}(u_1 - u_2)/L_3
\]
The point $T_v$, anywhere on line $T \cdot u_2$, is defined. Therefore the vector $T_v - T$ must be in plane $p$ and orthogonal to vector $u_5 - u_2$. Let $u_5 - u_2 = U$ and $p = [a, b, c, d]$. The following then need to be satisfied:

\begin{align*}
U_x(T_v x - T_x) + U_y(T_v y - T_y) + U_z(T_v z - T_z) &= 0 \quad (4.1) \\
 a \cdot T_v x + b \cdot T_v y + c \cdot T_v z + d &= 0 \quad \text{or} \quad T_v x = -(b \cdot T_v y + c \cdot T_v z + d)/a \quad (4.2)
\end{align*}

Substituting equation 4.2 in equation 4.1 gives:

\begin{align*}
U_x(-(b \cdot T_v y + c \cdot T_v z + d)/a - T_x) + U_y(T_v y - T_y) + U_z(T_v z - T_z) &= 0 \\
\text{or} \quad T_v y = [T_v z(U_x c/a - U_z) + U_x d/a + U_x T_x + U_y T_y + U_z T_z ]/(-U_x b/a + U_y) \quad (4.3)
\end{align*}

$T_v z$ is chosen as $T_v z = T_v z + 100$ and $T_v y$ and $T_v x$ is calculated by equation 4.3 and equation 4.2 respectively. $T_v u$ can now be calculated and normalised from:

$$T_v u = T_v - T$$

Next $L_{t2}$ is calculated from the relation:

$$L_{t2} = L_2 \sin(\theta)$$

The two values for $u_3$ can now be calculated as follows:

$$u_3b = T - L_{t2} \cdot T_v u \quad \text{or} \quad u_3a = T + L_{t2} \cdot T_v u$$

The above is implemented in the subroutine $\text{ThreePointInPlane}$. The source code can be found on the disks included with this document.
4.1.7 Hiding the invisible lines on a 3-D wire frame

In order to give a more realistic presentation of the wire frame, a function, *HideLine* was written which hides the invisible lines of the wire frame. The following describes the algorithm employed to accomplish the above:

The following procedure is repeated for each line of the wire frame:

The intersections of all the other lines with the current line in the X-Z plane is calculated with the function *LineIntersectLine*. The intersections are checked by determining if the intersection point lies on both the current line and the tested line with the aid of function *IsOnLinePiece*. All the intersections which comply to the above criteria, are stored in a matrix.

The intersection points within the matrix are now filtered to eliminate points which lie very close to each other. Thus, all points which lie within a certain tolerance of each other, are reduced to one intersection point.

Up to now the lines of the wire frame have been analysed in the X-Z plane only, thus the 2-D plane in which the 3-D wire frame is presented. To determine if a line piece between two intersection points is visible or not, the third dimension, the Y axis, need to be employed.

Therefore, the next section determined the equations for the current line in the X-Z plane, as well as the X-Y plane.

The following is repeated until all the intersections in the matrix have been processed:

The first intersection point of the matrix is taken (which represents the starting point of the current line). The closest other intersection point from the matrix is calculated. Next the midpoint, in 3-D, of the line between the current intersection point and the closest one is determined.

This midpoint is tested to determine if it is visible or not by the function *IsVisible*. This function will be discussed at the end of this section.
If it is visible, the line from the current intersection point to the closest one in the X-Z plane is drawn. The current point is eliminated from the matrix and the closest point is set equal to the current point and the loop is repeated.

The above completes the algorithm for hiding the invisible lines of the wire frame. The IsVisible function, which is utilised in the above algorithm, will now be discussed.

IsVisible

As previously discussed the wire frame consists of 7 blocks, the base and one block for each axis. Each block consists of 6 surfaces or planes, therefore in total $7 \times 6 = 42$ surfaces exist. Each surface or plane is specified by four points.

The point being tested for being visible or not is referred to as the testpoint.

For each of the 42 surfaces the following is done:

The testpoint is first tested if it lies within the current polygon specified by the four points of the current surface with the aid of function IsIn4P_Polygon.

If the above is true, the testpoint is tested if its Y-value is greater than the Y-value of the current surface at the same X and Z co-ordinates. If this is not true, then the point is not visible, else it is visible.

If the testpoint was invisible in any of the surfaces the function returns a value of invisible, else a value of visible is returned. Each of the functions mentioned can be found on the attached disks containing all the source code.
4.1.8 Part Manipulation

Figure 4.12 Modelled Part.

Figure 4.12 presents a modelled part as incorporated in the simulation. Appendix B holds the detail drawings of the physical parts used in the physical robotic cell. For the purpose of the simulation four of the above parts are modelled with the indicated dimensions. The parts were modelled as rectangular cubes to be as simple as possible and require as little as possible processing when manipulated. The actual physical parts are round and can be viewed in chapter 3 (See Figure 3.15). As already mentioned the transformation matrix TF, developed in section 4.1.2, serves a very versatile purpose. In the manipulation of the simulated parts this matrix is used again in a very simple way.

As already explained, the robot is simulated as seven blocks, one for the base and one for each axis. Each part is also simulated as a block. As soon as the robot grips a part, the block simulating that specific part is “attached” to the simulated robot as an eighth block. As the simulated robot is manipulated the eighth block, also manipulated in the same way by the transformation matrix TF, is manipulated as a seventh axis of the robot, but with no relative motion between the seventh and sixth axis. When the part is released, the above condition is cancelled. With the above method the simulated part in a specific position can be gripped by the simulated robot in any position, as well as any simulated part can be released in any position, without for example falling to the ground. This however, is not the case in reality and therefore the necessity for sensors in the physical robotic cell to keep track of the various part positions. The software implementation of the above can be found in the sub-routine chkM_Grip_Click() and in the sub-routines called from it on the Main form of the simulation software.
4.1.9 Collision Detection

In order to detect the collision of different wire frame objects with each other a function, Collision_Sub is generated to notify the user of collisions. Since the whole wire frame simulation is generated using blocks (see section 4.1.1), each specified by 8 points in 3-D, the task of collision detection is simplified to the collision detection of specified blocks with each other. However, each block contains 6 planes, therefore the collision detection is further simplified to the intersection detection between the planes of the specified blocks. For example, collision detection between the wire frame robot and a wire frame object in the robotic cell is accomplished by checking for any intersections between the polygons contained in the robot wire frame blocks and the polygons contained in the wire frame blocks of the object. The function Collision_Sub receives as parameters two ranges of polygons to test against each other for intersections. In the development of the software, all the blocks and polygons are numbered to easily identify the polygons belonging to a certain block.

The following describes the algorithm implemented in Collision_Sub, employed to accomplish the above:

As already mentioned, the function Collision_Sub receives two ranges of polygon numbers, say polygons A to B and polygons C to D as input parameters. The main part of the algorithm consists of two loops, the second nested in the first. The algorithms are as follows:

For all polygons from A to B the following is done:

- Retrieve the 3-D co-ordinates of the current polygon, say $A_{Temp}$, by use of the function NextPoly
- Calculate the equation of the plane in which $A_{Temp}$ lie with use of the function P3_pla}

For all polygons from C to D the following is done:

- Retrieve the 3-D co-ordinates of the current polygon, say $C_{Temp}$, by use of the function NextPoly
- Calculate the equation of the plane in which $C_{Temp}$ lie with use of the function P3_pla}
- Calculate the intersection line, if any, between the plane in which $A_{Temp}$ and $C_{Temp}$ lie by use of the function PlaneIntersectPlane.
Next, the intersections of all eight of the extended lines of $A_{\text{Temp}}$ and $C_{\text{Temp}}$ with the intersection line between the two planes of the polygons is determined with the use of the function `Det_Intersect`.

The intersection points of the extended lines of $A_{\text{Temp}}$ is checked with the function `IsOnLinePieceE` to determine if the specific point is on the specific line piece of $A_{\text{Temp}}$. Since all the polygons are rectangulars, either two intersections, say $C_1$ and $C_2$ or no intersections would occur. The very unlikely cases where the intersection line of the two planes touches the one corner, or crosses through both corners of $A_{\text{Temp}}$ are ignored.

The intersection points of the extended lines of $C_{\text{Temp}}$ is now checked with the function `IsOnLinePieceB` to determine if the specific point is on the specific line piece of $C_{\text{Temp}}$ as well as if they are situated between $C_1$ and $C_2$.

**Collision is detected if the following are all true:**

Two points could be found on the line pieces of $A_{\text{Temp}}$ which intersect the intersection line between the plane of $A_{\text{Temp}}$ and $C_{\text{Temp}}$.

Any points could be found on the line pieces of $A_{\text{Temp}}$ which intersect the line between point $C_1$ and $C_2$.

The next polygon in the C to D range is analysed.

The next polygon in the A to B range is analysed.

The above implementation can be found in the sub-routine `cmdCollision_Click` which implements the function `Collision_Sub`. The above simple collision detection system is possible since the complete wire frame model of the robotic cell consists of rectangular polygons. The Visual Basic implementation proved very successful in determining collisions between specified objects, given a specific configuration of all the objects in the robotic cell. The above function is able to check for any
collisions in a certain cell configuration within 0.54 seconds. The latter time is achieved on a 166 MHz PC with 32 MB RAM. All the source code regarding the above can be found on the disks attached to this document.
4.2 Implementation in Visual Basic

An explanation of the actual user interface created with Visual Basic is discussed below. Visual Basic version 5, the most recent release at the time, is used in the development of the software as discussed in section 2.1. The software is implemented as the Visual Basic project file Robmove.vbp. In order to communicate with the Robot Controller the Visual Basic module S4api.bas, which contains the declarations of all the functions communicating with the Robot Controller, as supplied by the robot manufacturers, is included into the project. Also supplied by the manufacturers is an OCX control, SponMsg.ocx which enables the receiving of spontaneous messages from the Robot Controller. This control is also included in the project. The disks attached to this document contain all the source code, as well as the installation disks.

Visual Basic offers a programming method called MDI (Multiple Document Interface) An MDI programme contains several documents (child windows or forms) within a parent window (a container window or form). Almost all the popular professional Windows based applications are MDI-based. These applications allow the user to jump around in the child windows without closing any of them. However with the implementation of this project, MDI is not used. The reason for this is to simplify the software flow to minimise any potential damage to the robot resulting from too many instructions from different child windows to the Robot Controller. The software is developed by using a main form with various other forms which opens from the main form. Before accessing another form, the current form must first be hidden or closed. In this way, very strict control is ensured regarding the execution of commands to the robot from different forms. Figure 4.13 presents a flow diagram of the complete software package. Following is a brief explanation of the functions of each control on the different windows or forms. The software consists of the following forms:

- Main form
- Programming form
- Robot Control form
- Connect form
- Robot Paths form
- Expert form
- Web Watch form
- Transponder form
Figure 4.13 Flow diagramme of Visual Basic software layout (Continue).
Figure 4.13 Flow diagramme of Visual Basic software layout.
The programme also employs timers. Timers are sub-routines which incorporate an internal clock which allows the user to set a certain time between the automatic repetition of the sub-routines. Figure 4.13 also indicates all the timers and their function in the respective forms. In the remainder of this chapter all mentioned forms, buttons, labels, checkboxes, scrollbars and listboxes are presented in bold. Each of the above-mentioned forms will now be discussed separately:

4.2.1 Main form

Figure 4.14 represents the Main form with explanations added. The wire frame manipulation part of the software is divided into three parts. The first part deals with the Scaling, point of Viewing and relative Position of the simulation on the Main form.

The second part deals with the Update of the simulation movement and the third part deals with the manipulation of the Simulated Parts.

Scaling, Viewing and Position

With the start-up of the software an original global matrix OBlock is created containing the points of the zero position simulation with co-ordinate system as presented in Figure 4.2 with the form displaying the X-Z plane.

This matrix is used as the basis and is never changed during the execution of the programme. For the purpose of Scaling and Viewing a global matrix, Block, equal to OBlock is created. Whenever any scaling or viewing is done OBlock is used as basis and the new positions, as a result of scaling or viewing are stored in Block. The relative position of the simulation is moved by changing the values of the zero point, the zero X and Y positions, of the Main form.

The Scaling scrollbar can be moved to up or down scale the wire frame. The wire frame can be viewed from any direction by adjusting the X Axis Rotation scrollbar and Z Axis Rotation scrollbar. This adjustment causes rotation around the X and Z axis's according to the Base co-ordinate system in the left-hand corner. By adjusting the Horizontal Position scrollbar and Vertical Position scrollbar the position of the wire frame is moved horizontally and vertically respectively about the screen.
Update

Two wire frame robots exist in the simulation and the selection of a certain robot is determined by the value of the Robot checkbox in the upper left corner. Robot1 refers to the robot on the right and Robot2 to the one on the left.

The update or movement of any of the two robot wire frames can be accomplished in three ways:

1. Adjusting any of the six Axes scrollbars. The speed or incremental size update can also be adjusted in the upper right-hand corner. These settings take effect when the arrows at the end of the Axis scrollbars are used. The following gives the size of the incremental update for the various Speed settings:

   - Fast - 10 degrees
   - Medium - 1 degree
2. Typing the required axis value in the corresponding Axis textboxes. After a value or values are adjusted in this manner the Move button is clicked to update the wire frame.

3. Movement of the wire frame by using the Arrow -up, -down, -left, -right, a and s keys. Whenever this mode of movement is required the cursor must be placed in the Axis Keycontrol textbox. The Active Axis box displays the current active axes. To change the current active axes the Active Axis box can be clicked or the 0 (Ins) key on the left-hand number pad of the keyboard can be pressed. This movement control also incorporates the Speed setting under point 1.

Simulated Parts Update
The simulation, as the experimental set-up, contains four identical parts. These parts can be manipulated by any of the two simulated robots. The selection of a certain part to be manipulated by the selected robot is done by selecting a part from the Part listbox in the upper left corner. The value in the Grip Off checkbox, in the upper left corner, determines if the selected part in the Part listbox is connected with axis 6 of the selected robot in order for the part to be manipulated as if gripped by the robot in question. The parts can be gripped from any position, as well as released in any position, which obviously would not be possible in reality. When a part is gripped, irrespectively of its position, it forms part of axis 6 and accordingly moves with axis 6 as the corresponding robot is manipulated.

Other movement related functions
The Reset button restores the wire frame to the original zero position when the Full Reset checkbox is checked else, only the position, scaling and viewing are changed to the original.
If the Joint checkbox is checked, axis 3 stays parallel with the previous position when axis 2 is moved. The Program button and Degree checkbox will be discussed in the Programming form.

Robot control on Main form
This section consists of two functions. The first, Robot to Simulation, is where the wire frame or simulation follows the real robot movements. The second, Simulation to Robot, is where the real
robot follows the simulation. For both these functions to operate the Not Connected label should display Connected with a green colour. Only one of these functions can operate at a time.

When the Robot Control Not Initiated label is clicked the PC downloads the pre-programmed programme of Figure 2.2 to the robot and starts the execution thereof. This programme runs repeatedly, on the robot, with a position variable that can be changed from the PC during runtime. The Robot Control Not Initiated label will display Robot Control Initiated in green if the operation was successful, else an error code will be displayed in the Error textbox. To engage the robot to follow the simulation the VarWrite AbsJ checkbox must be checked. When this is done the simulation can be moved in any of the above described 3 ways and the robot will follow. Clicking on the Robot Control Not Initiated label will de-activate the operation.

When the Not Active label, in the Robot to Simulation section, is clicked the timer tmrRobToSimul is activated and reads the robot position repeatedly and updates the simulation accordingly. If no errors occurred the Not Active label will display Active in green. Clicking again on the Not Active label will stop the timer and indicate Not Active.

In the Robot Link section the DisConnect button will disconnect the robot if connected. The Connect button displays the Connect form and will be discussed later. The Not Connected label indicates if the robot is connected or not. The Stop button stops the current programme running on the robot.

Clicking on the HideLines button will redraw the simulation without the hidden lines. The full wire frame can be obtained again by clicking the ZDrawRob button. The Remove ZDrawRob button removes the full wire frame. Clicking the Cls button clears the simulation drawn by the HideLines button.

The TopView button presents a view from the top (down the Z-axis) of the simulation. It also includes working circles of both the robots. The Coll Off checkbox is used to switch the collision detection system on and off. If switched on, a message would indicate the collision of a robot with another robot or surroundings. The Collision button checks for any collisions in the current robotic cell configuration.
4.2.2 Programming form

Figure 4.15 represents the Programming form as displayed on the computer screen with explanations added. When the Programming button on the Main form is clicked the Programming form is displayed. The Newposition textbox then contains the current position of the simulation. This position is presented in one of two formats. The first format specifies the TCP, as shown on the Main form, and orientation, as a quaternion, with the configuration of axes 1, 4 and 6 as discussed in section 4.1.4. The following is an example:

```
MoveJ[[1034.98,185.56,1226.21],[0.480802,0.066762,0.862730,0.141667],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,ne,tool0;
```

The above format appears in the Newposition textbox if the Degree checkbox on the Main form is unchecked. If it is checked, the position is specified by giving the exact position of each of the six axes in degrees. The following is an example:

```
MoveAbsJ[10.00, 10.00, 20.00, 10.00, 10.00, 10.00],[0,9E9,9E9,9E9,9E9,9E9]],v200,fine,tool0;
```

Once the new position is in the Newposition textbox it can be added or inserted into the Programming textbox by the Add and Insert buttons respectively. The Speed textbox is used to specify the speed the robot must move to the specified position which is also included in each position line. The speed values are as described in section 2.4. (See Table 2.1)

The Clear button clears the Programming textbox. The Save To: button is used to save the contents of the Programming form, with the specific Rapid language pre- and post-fixes, in order to save this as a Rapid file for execution by the robot. The file is saved to the filename specified in the Save To textbox. The Done button hides the Programming form. The Programming form also incorporates the standard Copy, Cut and Paste functions which can be activated by right clicking with the mouse or using the Edit menu on the menu bar.
4.2.3 Robot Control form

The Connect option on the menu bar displays the Connect form which will be discussed later.

The Robot Control form consists of three parts. (See Figure 4.16):

- File Services
- Program Execution
- System Status

File Services

By clicking on the Download button the file in the Local textbox on the PC is transferred to the remote location on the Robot specified in the Remote textbox. The opposite action is performed when the Upload button is clicked. By clicking the Program Load button the file in the Remote textbox is loaded into the Robot Controller. Any errors which might occur are displayed in the Errors textbox.
Figure 4.16 Robot Control form.

Program Execution

By clicking the Program Prepare button the current programme in the Robot Controller, loaded by the Program Load button, is checked for any errors and is prepared for execution. The Start button starts the execution of the current prepared programme in the Robot Controller. Depending on the value of the Repeat checkbox, the execution is continuous or is executed only once. The Ping button is used to check the communication link to the robot. The Program Delete button deletes the current programme in the Robot Controller. The Stop button stops the current execution of a programme and thus stops the robot. Any errors which might occur are displayed in the Errors textbox.

System Status

By checking the Continuous Update, checkbox the System Status of the robot is read continuously. If the System Status only need to be read once, the Read System Status button can be clicked. By clicking the Motors On/Off button the robot motors can be switched on and off respectively. The System Status, as displayed, is according to the robot specifications. Any errors which might occur are displayed in the Errors textbox.
4.2.4 Connect form

The Connect form is displayed by clicking the Connect menu on the Robot Control form. The host name or IP address of the Robot Controller is entered in the textbox as indicated on Figure 4.17. Two options exist. OK, to connect to the Robot Controller and Cancel to cancel the connection and go back to the Robot Control form. The progress on checking the Rap Serial Link is also indicated as a percentage if the OK button is clicked. This procedure takes about 3 minutes.

4.2.5 Robot Paths form

The Robot Paths form is displayed by clicking the Paths button on the Main form and is represented by Figure 4.18. This form contains all the pre-programmed paths for use in the robotic cell. The specific location of the different paths is given in Figure 3.2 and Figure 3.3. The desired path can be chosen by clicking the Path listbox. Depending on the status of the Forward checkbox the chosen path will be executed in the forward or reverse direction. The MsgBox checkbox, if checked, will pause the execution of the specific path at each programming point on the path and give the position details. The Update Simulation checkbox, if checked, will update the simulation as the path is executed.
To initiate the robot for path execution, the Robot Control Not Initiated label is clicked. After a while the label will turn green with the label displaying Robot Control Initiated. If the Run Robot checkbox is checked and the Run button is clicked the robot will execute the current path. The speed of execution can be set in the Speed listbox. Any errors which might occur will be displayed in the Errors textbox. The Stop button will stop the robot. To exit the form click the Hide button.

4.2.6 Expert from

The Expert form, presented by Figure 4.19, is displayed by clicking the Expert button on the Main form. The expert system can be operated manually or automatically. The basic expert system software is obtained from [26]. This software in Pascal, is altered to serve as a DLL file which could be used by the Visual Basic simulation software as discussed in section 2.3. Delphi is used to compile the DLL. A rule base is generated for the experimental set-up. These rules contain the intelligence related to a certain set of variables existing in the experimental set-up in the form of sensors, positions and conditions. The flow diagram of Figure 3.7 presents the rule base. It is important at this stage to note the difference between an expert system and simple software logic. If the rule base, presented in Figure 3.7, is implemented in simple software logic without an expert system the given system would perform in the same way as the expert system.
Because:
part_gripped = yes
and
gripper_clased = yes
and
partin_gripper = new

Since no rules are able to conclude a value for part_gripped.
The user must be asked.

Is a part gripped?
1. yes
2. no
Enter a number from 1 to 2

The crucial difference however is that at any time additional rules can be added to the rule base of the expert system without changing anything else. These added rules are automatically incorporated into the reasoning of the expert system and thus would have an immediate affect on the output. In the simple software logic case the whole structure of the logic needs to be restructured in order to accommodate any added rules. The same applies if changes to the rules need to be implemented.

At start-up of the Expert form this rule base is loaded into the expert system. Only the first simulated robot or Robot1 (see section 4.2.1) can interact with the expert system and can be controlled from the Internet. This expert system presents the following:

1. Explanations while reaching a conclusion. (Explain How).
2. Explanation of why more input is needed. (Explain Why).
3. Questions to the user to obtain more input. (P Question).
4. A selection from which the user can select a response to a question. (Select Option)

5. The result after a conclusion is reached. (Result)

6. The option to load facts. (Load Facts)

7. Manual operation of the expert system

8. Reading of sensors, positions and conditions in the robotic cell.

9. Controlling the Gripper and Machine (Lathe) in the robotic cell.

10. Taking action in the robotic cell on the concluded results.

11. Automatic operation of the expert system

1. **Explanations while making a conclusion. (Explain How).**
   In the process of concluding a result, the rules which are used to draw the conclusion, are displayed in the Explain How textbox. This feature is very useful since the exact reasoning can be observed by the user and also aids the development and refinement of a new rule base.

2. **Explanation of why more input is needed. (Explain Why).**
   If no conclusion could be reached, due to the lack of sufficient information, the expert system explains why more information is needed. This explanation is presented in the Explain Why textbox.

3. **Questions to the user to obtain more input. (P Question).**
   When a condition arises where more information is needed to conclude a result, as described above, the P Question textbox presents the actual question to the user.

4. **A selection from which the user can select a response to a question. (Select Option)**
   The Select Option textbox presents the user with the possible answers to the question presented in P Question.

5. **The result after a conclusion is reached. (Result)**
   The result after a conclusion is reached is presented in the Result textbox. For the specific case of the experimental set-up the result contains the next action in the robotic cell.
6. The option to load facts. (Load Facts)
All the sensor values, conditions and positions which exist in the experimental set-up, are presented as facts and grouped under 1. Production Equipment, 2. Materials Handling and 3. Assembly. If the Load Facts checkbox is checked then these facts are automatically loaded when the expert system is started. If not checked the expert system will ask the user the value of all the needed facts manually.

7. Manual operation of the expert system
By clicking the Start button in the Manual Control section of the Expert form, the expert system is manually started. If the Auto checkbox is checked the single digit response of the user is automatically processed without having to click the Start button to process the response. In cases where the Select Option textbox offers more than 9 options the Auto checkbox first need to be unchecked in order for the user to respond with a 2 digit answer after which the Start button is clicked to process the response. A history of previous answers are accumulated in the Answer history section. This history can be cleared (expert system can be restarted) by clicking the Clear History button.

8. Reading of sensors, positions and conditions in the robotic cell.
By clicking the Read Sensors button the facts under 1. Production Equipment, 2. Materials Handling and 3. Assembly are updated to the current ones in the robotic cell. This function is only operational if the robot is connected.

9. Controlling the Gripper and Machine in the robotic cell.
If the robot is connected, the Machine Stop and Gripper Closed checkboxes are enabled. The user can then open/close the Gripper and stop/start the machine (Lathe) by clicking the respective checkboxes.

10. Taking action in the robotic cell on the concluded results.
By clicking the Take Action button, the user converts the text in the Result textbox to the appropriate action in the robotic cell. These actions can only be implemented if the robot is connected.
The following is a list of possible actions:

- open_gripper - Opens the Gripper
- close_gripperX - Close Gripper to grip part X
• runpathX - Executes path X in the forward direction.
• runpathXr - Executes path X in the reverse direction.
• no_parts - All the parts in the cell have been machined and assembled. Stops the expert system.

11. Automatic operation of the expert system
By clicking the Start Auto button the Load Facts button is automatically checked, the Read Sensors procedure is automatically performed and the expert system is started. Every time a result is concluded, the Take Action procedure is automatically performed and the system waits until the action specified by the result is completed. The Read Sensors procedure, expert system and Take Action procedure is repeatedly run until the no_parts or error result occurs.

4.2.7 Web Watch form

The Web Watch form is displayed by clicking the Web button on the Main form. This form is used to interface with the simulation applet on the Internet. The detailed communication set-up to the Internet is already discussed in chapter 3. By clicking the Start button, a procedure is invoked which continuously check for new information from the applet. A running Counter which runs up to 1000 and then starts at 0 again indicates that the system is checking for incoming information. The Stop
button stops this procedure. Any new information is displayed in the Incoming textbox. The Previous textbox displays the previously received information from the applet. If the Take Action checkbox is checked, immediate action is taken on any arrived information. To authorise the action taken on received information, all information received is preceded with a password, supplied by the user on the Internet. If this password matches the one in the Password textbox, action appropriate to the received information is authorised. The following are commands on which action is taken:

- MoveAbsj[10,10,10,10,10,10] - Move the robot to the specified position
- open_gripper - Opens the Gripper.
- close_gripperX - Gripping part no X
- runpathX - Runs Path X in the forward direction
- runpathXr - Runs Path X in the reverse direction
- robot_control - Initiates the manual control to position the robot.
- path_control - Initiates the path control to enable the robot to execute a certain path.
- stop_robot - Stops the robot.
- start_expert - Starts the expert system.
- StartTest - Does a communication check at start-up of the applet.

To send information to the applet the information is entered in the Outgoing textbox and the Send button is clicked. During the normal, as well as the remote operation of the simulation software the following commands are automatically sent to be received if a user exists on the Internet:

- MoveAbsj[10,10,10,10,10,10] PlaceX - Move the robot to the specified pre-definition position X.
- close_gripperX - Grip part X.
- open_gripper - Opens the Gripper.

To exit the Web Watch form the Hide button is clicked.
4.2.8 Transponder form

The last form deals with the integration of the transponder system into the rest of the software. As already mentioned in section 3.8, a round flat cable antenna, diameter 0.12 m, is built and attached to the pneumatic gripper. See Figure 3.12 for a photograph. The purpose of this antenna is to identify the parts in the robotic cell, to which the transponders are attached. As previously mentioned, the on-line recognition of parts within a robotic cell is becoming a reality. This method of transponder tagging is introduced to replace complicated image recognition techniques, requiring very large computing power and equipment. While other systems can detect the position of a part, the transponder can give its identity accompanied by relevant information and history of the part. The above increases the available information regarding the current conditions in the robotic cell and therefore having more information, more intelligent and powerful conclusions and actions can be generated by artificial intelligent components like the expert systems. Thus, by implementing the above, the robotic cell becomes more intelligent.

An example of the above in the specific experiment can be found in the Assembly section. (See Figure 3.4 and chapter 3) When the robot picks up a part from the Assembly section to be machined in the Production Equipment section, the expert system assumes, because of the part’s position, that the part has not been machined. With the addition of the transponder system the expert system will know for sure that the specific part has not been machined because it can be read from the memory of the transponder attached to the specific part. After the part has been machined in the Production Equipment section, the memory of the transponder is altered to indicate that the specific machining process has been completed. Whenever, at a later time, this specific part needs to be manipulated, the robot will know through the transponder system that this part has already been machined by reading the information on the transponder attached to it. The software implementation of the transponder system on the Transponder form is discussed below.

Since all the data transfer details from the transponder Reader Module through the RS 232 communication line to the PC are given in the manual provided by the manufacturers, any serial communication software can be used to communicate with the Reader Module. This is very helpful since the C libraries supplied with the Reader Module employs the commands, inportb and outportb, which are limited to C applications developed for DOS. This complicates the integration of these libraries into the Windows environment, since these two commands are not available in Visual C++ or the equivalent commands in Visual Basic.
However, Visual Basic 5 provides a communication control, called MSComm, which supports communication through commport 1 to 4 (Comm 1 to Comm 4) that can interface with the RS 232 communication line to the Reader Module. Therefore by setting the various properties of the MSComm control to the following data transfer details, communication to and from the Reader Module is established:

**Data Transfer Details**
- **Communication port**: Comm 3
- **Baud rate**: 9600 baud
- **Data bits**: 8
- **Stop bit**: 1
- **Parity bit**: No

Since commport 1 is already used by the mouse and commport 2 by the link to the Robot Controller an additional series card is inserted into the PC to supply commport 3 and 4. Commport 3 is therefore used to connect to the Reader Module. Figure 4.21 presents the Transponder form implementing the transponder system.
All information to the Reader Module is sent by using the MSComml.Output command. In order to receive information from the Reader Module, the OnComm event is used. This event occurs as soon as the number of characters specified in the RThreshold property of the MSComm control has been received from the Reader Module. On the occurrence of this event, the information received from the Reader Module, which is stored in the InBuffer of the MSComm control, is added to the contents of the Received textbox. The RThreshold property in this specific case is set to 1 and combined with a timer which detects the time between two consecutive characters received from the Reader Module. If a delay of more than 50 ms occur before the next character is received, the end of the current reply from the Reader Module is acknowledged and the complete reply accumulated in the Received textbox is processed after which the Received textbox is cleared to await another reply from the Reader Module.

During processing of the received data from the reader, the data is divided, according to the manufacturer’s specification, in four sections, namely Block 1, Status, Data and BCC. Block 1 presents the hexadecimal value of the first character received from the Reader Module. The Status represents the decimal value of the second received character. BCC presents the last received character and is used to verify the accuracy of the received data from the Reader Module. Appendix D presents the calculation method of BCC. Data presents any reply from the reader and can vary in length according to the specific reply.

The Transponder form is displayed by clicking the Transp button on the Main form. All the data received from the Reader Module is displayed in the Receive textbox. After each transmission from the Reader Module the contents of the Receive textbox is processed into the Block1, Status, Data and BCC textboxes respectively as explained earlier. The Get Version button instructs the Reader Module to supply its version, serial number and date of manufacturing. The Power Down button switches the antenna current off, therefore no interaction with any transponder can take place. To switch the antenna current back on the Power Up button can be clicked. The Stop Command button cancels all current instructions being processed by the Reader Module. The Poll Tags button sets the Reader Module into polling mode for HITAG1 transponders. As soon as a HITAG1 transponder is detected in the antenna field, the Reader Module will send a response to the PC. The Get_Snr button instructs the Reader Module to check the current antenna field for HITAG1 transponders and to return the first found transponder’s serial number and also to indicate if more transponders exist in the transponder field.
The continuous section employs a timer which continuously polls or requests the serial number of transponders in the antenna field depending on the status of the Snr and Poll check buttons. This continuous action is activated by clicking the Con - Stop button. After activating this continuous action, any results will be displayed in the No Tag label. The caption of this label changes to Tag Found if a transponder is found while in Poll mode, and changes to Serial No Found if a serial number of a transponder is detected while in Snr mode. The continuous action can be stopped by clicking the Con - Stopped button again.

The On_Comm textbox is used to display any messages regarding events which occurred while receiving data from the Reader Module. The above is currently not used for any specific purpose.

The following is a list of commands used in the above explained form instructing the Reader Module to perform certain functions:

<table>
<thead>
<tr>
<th>Implementation Button</th>
<th>Command</th>
<th>Description of action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Version</td>
<td>Chr(2) + &quot;V&quot; + &quot;T&quot;</td>
<td>Reader Module returns its version, serial number and date.</td>
</tr>
<tr>
<td>Power Down</td>
<td>Chr(3) + &quot;D&quot; + Chr(0) + &quot;G&quot;</td>
<td>Sets the Reader Module in power down mode.</td>
</tr>
<tr>
<td>Power Up</td>
<td>Chr(3) + &quot;D&quot; + Chr(1) + &quot;F&quot;</td>
<td>Sets the Reader in power up mode.</td>
</tr>
<tr>
<td>Poll Tags</td>
<td>Chr(3) + &quot;I&quot; + Chr(3) + &quot;I&quot;</td>
<td>The Reader Module is set into polling mode and polls for transponders in the antenna field. If a transponder replies, this reply is sent to the PC and the polling mode is cancelled.</td>
</tr>
<tr>
<td>Get Snr</td>
<td>Chr(2) + &quot;G&quot; + &quot;E&quot;</td>
<td>Instructs the reader to read the serial number of any HITAG1 transponder in</td>
</tr>
</tbody>
</table>
the antenna field. The reply from the reader may include that no tags in the antenna field exist, the serial number of a tag in the antenna field and if there are more than one tag in the antenna field.

Stop Command \texttt{Chr(2)+Chr(166) + Chr(164)} Stops all current activated reader
functions like for example polling.

Appendix D holds the manufacturer's specifications to calculate the above commands.

In developing the above software a problem regarding the received data is encountered. When receiving information from the reader some of the characters seems to be lost during the transmission. This however, is not the case with the DOS sample software received from the manufacturers. Unfortunately, due to time limitation this problem could not be solved in the Windows environment. Due to the above, the exact serial number, memory contents or any other information available from the transponders could not be read accurately. However, the above developed software could detect the presence of a transponder in the antenna field, as well as detect the presence of a transponder serial number. Chapter 6 provides more detail on the testing of the transponder system.
5. Robot Simulation (Internet)

This chapter presents the communication between the Java server on Nagasaki and the applet on the Web page through which the Internet user interfaces with the Java server. As stated in section 3.2, all network connections from the PC connected to the Robot Controller is disabled as soon as the Rap Serial Link between the Robot Controller and PC, connected to the robot, is engaged. This link also only supports Windows 95 or Windows NT operating systems with programming interfaces to only Microsoft Visual Basic and Microsoft Visual C++. To establish the communication from the PC, connected to the Robot Controller, to the PC running the Java server software, a non-standard method of mapping a network hard drive and using two shared files, as described in section 3.2 is used to overcome the disabled TCP/IP connection. This non-standard procedure is not necessary in the general case where the link to the Robot Controller is more versatile and does not limit the usage thereof. In such a case the specific communication layout, presented in Figure 3.1, can be reduced to the use of only two PC’s in the whole system. Figure 5.1 presents this more general layout. The simulation and server software can be written in any language which supports the TCP/IP protocol, as well as the protocol required by the Robot Controller link. Currently the latest versions of all common programming languages like Visual Basic, Visual C++, C++, Java, Delphi etc. supports the easy use of TCP and TCP/IP in programming and Robot Controller links should be open to any type of implementation without any restrictions.

Therefore, considering the above, the communication between the Web server and applet form a main part of the general communication layout which, together with the relevant software, is discussed in detail in the following section. However, the simulation part of the applet is not discussed since it is almost identical to the Visual Basic version with alterations made only to accommodate the Java language.

5.1 Java Applet and Java Server implementation

This section describes the communication part of the Java applet, Roblix2.java, as well as the Java server software, server3.java, running in the specific experimental set-up on the PC, Nagasaki. (See Figure 3.1 ) As discussed above, this section of the software can be implemented in any client/server application. In this case the client is in the form of an applet, Roblix.java, on a Web page and the server, server3.java, is Java software running on Nagasaki. These Java source code files are compiled to Roblix.class and server3.class and appear on the disks attached to this document.
SeFitlsors and Eta ipment

Robot Controller

Sensors and Equipment

Other systems like a transponder system.

PC User on Internet

Link to Robot Controller

TCP/IP connection between Web page and Web server

TCP/IP connection between Java Applet on Web page and Java server on Nagasaki

PC serving as a Web server, as well as running the simulation software containing Client/Server abilities, 3-D modelling of the robotic cell and intelligent components like for example an Expert system

The User PC connects to the Web server and retrieves the Web page containing the Java applet which connects to the Simulation software also containing server software to communicate to the Applet on a Client/Server basis.

Figure 5.1 Generalised Communication layout.

The following is the source code found in roblix5.java, on the attached disks, dealing with the communication to the server. All comments are indicated with //.
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;

public class roblix5 extends Applet
{
    public void init()
    {
        ...
        ...
        ...
        // Initialising code resides here. Full source code is available in the file roblix5.java in the disks
        // attached to this document
        ...
        ...
        ...
        // The following creates a socket to communicate with a server on port 6789 of the host (Nagasaki)
        // that the applet's code is stored on. It also creates streams to use with the socket. User input to the
        // server is via a TextArea, outputarea, and output from the server is received in a TextArea,
        // outputareas.
        CONNECTED = true;
        // Set the CONNECT flag to true
        try
        {
            s = new Socket(getCodeBase().getflosto, PORT);
            // The host, nagasaki.rau.ac.za and PORT, 6789 is defined in the HTML page index.html
            in = new DataInputStream(s.getInputStream());
            out = new PrintStream(s.getOutputStream());
            listener = new streamlistener(this, in, outputarea, CONNECTED);
            //Start the streamlistener thread to listen for responses from the host
            showStatus("Connected to " + s.get inetAddress().getHost Name()
            + ":" + s.getPort());
            // If the connection is successful show the above message on the status bar
        }
        catch (IOException e) { showStatus("**** Server Down ****");
            CONNECTED = false;
        }
    }
}
outputarea.setText("SERVER NOT RESPONDING - NOT OPERATIONAL");
outputarea_p.setText("SERVER NOT RESPONDING - NOT OPERATIONAL");

// If a connection with the host was not successful the above messages are shown on the
// status bar and TextAreaes respectively. The CONNECT flag is set to false
// Test communication by sending the test “StartTest” to the host

inputfield.setText("StartTest");
Send();

// Wait for output from the server on the specified stream, and display
// it in the specified TextArea.
// After a connection was successfully established between the applet and host the streamlistener class
// is activated to continuously listen for responses from the host which is posted in the output
// TextArea

class streamlistener extends Thread {
    DataInputStream in;
    TextArea output;
    roblix5 rob;
    boolean Connected;

    public streamlistener(roblix5 rob,DataInputStream in, TextArea output,boolean Connected) {
        this.rob = rob;
        this.in = in;
        this.output = output;
        this.Connected = Connected;
        this.start();
    }

    public void run() {
        String line;
        output.setText("TOO MANY CONNECTIONS OR SERVER DOWN");
        rob.TakeAction();
        output.setText("TOO MANY CONNECTIONS OR SERVER DOWN ");
        rob.TakeAction();
        // The above sets the output TextArea to the above message. If the test signal of StartTest is
        // successfully returned from the host it will override the above message

        // Continuously wait for responses from the host until an error occurs or until the connection is
        // closed by the host
        try {
            // Save the rest of the code here...
        } catch (Exception e) {
            System.out.println("Exception caught: "+e.getMessage());
            // Handle exception...
        }
    }
}
// In an endless loop wait for host response
for(;;) {
    line = in.readLine();
    if (line.trim() == null) break;

    // If the host's response is longer than 5 characters send it to the output TextArea
    // and act on it. The function TakeAction in the roblix5 class is used to take action on
    // server responses.
    if (line.length() > 5 )
    {
        output.setText(line);
        rob.TakeAction();
    }
}

// Any communication errors is reported in the output TextArea

finally {Connected = false;output.setText("CONNECTION CLOSED BY SERVER");}
// When breaking the communication to the host the output TextArea displays the above message.
}

The following is the complete Java source code for the server running on Nagasaki.

import java.io.*;
import java.net.*;

public class server3 extends Thread {  
    public final static int DEFAULT_PORT = 6789;
    protected int port;
    protected ServerSocket listen_socket;
    int CONNUMBER;

    // Exit with an error message, when an exception occurs.
    public static void fail(Exception e, String msg) {
        System.err.println(msg +": " + e);
        System.exit(1);
    }

    // Create a ServerSocket to listen for connections on port 678
    public server3(int port)
    {
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
    }
}
try { listen_socket = new ServerSocket(port); }
catch (IOException e) {fail(e, "Exception creating server socket");}
// Display the above message when a communication error occurs
// Display the following message if the server is successfully listening to port 6789 for possible
// connections from applets.
System.out.println("Server: listening on port " + port);

// Start the listening thread
this.start();
}

// The body of the server thread, looping forever, listening for and accepting connections from
// clients. For each connection it creates a connection object to handle communication through the
// new Socket.
public void run() {
    CONNUMBER = 0;
    boolean Flip = true;
    try {
        while(true) {
            if (CONNUMBER == 0 ) {
                System.out.println("No Users Connected ! Listening on port " + port);
                Socket client_socket = listen_socket.accept();
                CONNUMBER = CONNUMBER + 1;
                connection c = new connection(client_socket,CONNUMBER,this);
                file_watch d =new file_watch(client_socket,CONNUMBER,this);
            }
            if (Flip) {
                System.out.println(CONNUMBER + " User Connected !");
            }
            else {
                System.out.println(CONNUMBER + " USER CONNECTED !");
            }
            Flip = !Flip;
            try{ sleep(200); } catch(InterruptedException e3) {} 
        }
    }
    catch (IOException e) { fail(e, "Exception while listening for connections");}
}

// Start the server up, listening on an optionally specified port
public static void main(String[] args) {
    int port = 0;
    if (args.length == 1) {
        try { port = Integer.parseInt(args[0]); } 
        catch (NumberFormatException e) {port = 0;}
        catch (NumberFormatException e) {fail(e, "Exception while parsing argument ");}
    }
}
new server3(port);

public void Close_Connection()
{
    CONNUMBER = 0;
}

// This class is the thread which handles all communication with a client
class connection extends Thread {

    protected Socket client;
    protected DataInputStream in;
    protected PrintStream out;
    int CONNUMBER;
    server3 serv;

    // Initialise the streams and start the thread
    public connection(Socket client_socket, int CONNUMBER, server3 serv) {
        client = client_socket;
        this.CONNUMBER = CONNUMBER;
        this.serv = serv;
        try {
            in = new DataInputStream(client.getInputStream());
            out = new PrintStream(client.getOutputStream());
        }
        catch (IOException e) {
            CONNUMBER = 0; client.close();} catch (IOException e2) {CONNUMBER = 0;}
        System.err.println("Exception while getting socket streams: "+ e);
    return;
        this.start();
    }

    // Provide the service.
    // Read a line, and send it to a file.
    public void run() {
        String line;
        StringBuffer revline;
        int len;
        try {
            for(;;) {
                // read in a line
                line = in.readLine();
                // send it
                out.println(line);
            }
        }
    }
}
if (line == null) break;
len = line.length();

System.out.println("Receiving from Applet client !!!");

// If the file d:\ser_dat\s_write.dat is accessible write to it.
try {
    File outputFile = new File("d:\ser_dat\s_write.dat");
    FileOutputStream fos = new FileOutputStream(outputFile);

    for (int i = 0; i<len; i++)
    {
        fos.write(line.charAt(i));
    }
    fos.close();

    // Print the message below if the file could not be found.
    catch(FileNotFoundException e){System.out.println("File not found !!!
d:\ser_dat\s_write.dat");};
    revline = new StringBuffer(len);
    for(int i = len-1; i >= 0; i--)
    {revline.insert(len-l-i, line.charAt(i));
    // and write out the line
    out.println("Server Echo: " + line);
}

    catch (IOException e){CONNUMBER = 0;serv.Close_Connection(); }

} finally {try {CONNUMBER = 0;serv.Close_Connection(); client.close();} catch (IOException e2){CONNUMBER = 0;serv.Close_Connection(); }
}

// This class is the thread that repeatedly reads a file "s_read.dat" and if
// this file has changed since the last read the contents are sent to the client

class file_watch extends Thread {
    protected Socket client;
    protected PrintStream out;
    int CONNUMBER;
    server3 serv;
}
// Initialise the streams and start the thread
public file_watch(Socket client_socket, int CONNUMBER, server3 serv) {
    client = client_socket;
    this.CONNUMBER = CONNUMBER;
    this.serv = serv;
    try {
        out = new PrintStream(client.getOutputStream());
    } catch (IOException e) {
        try {client.close(); CONNUMBER=0; } catch (IOException e2){CONNUMBER=0; }
        System.err.println("Exception while getting socket streams: "+ e);
        return;
    }
    this.start();
}

// Provide the service.
// Repeatedly reads the file "s_read.dat", if it changed sent the contents to the client. Pause 0.5 sec
//after the file is read.
public void run() {

    StringBuffer prevline = new StringBuffer(205);
    String prev_str="Start",
    try {
        while(true) {
            try {
                String revline = new StringBuffer(205);
                File input = new File("d:\ser_dat\s_read.dat");
                FileInputStream fis = new FileInputStream(input);
                int c;
                revline = new StringBuffer(205);
                int char_count = 0;
                while ((c = fis.read()) != -1)
                {
                    char_count=char_count + 1;
                    char char_read = (char)c;
                    if (char_count <= 200) revline.insert(0, char_read);
                }
                fis.close();
                revline.reverse();
            }
        }
    }
}
String rev_str=revline.toString();

if (prev_str.equals(rev_str)==false)
{
    prev_str=rev_str;
    out.println(revline);
    System.out.println("Sending to Applet client !!!");
}

//Finished reading file and sending to applet.

} catch(FileNotFoundExceptione) {System.out.println("File not found !!!! d:\ser_dat\s_read.dat");}
// Pause 500 ms.
try{ sleep(500);} catch(InterruptedException e3) {}

catch (IOException e){CONNUMBER = 0;serv.Close_Connection(); }

finally {try {CONNUMBER = 0;serv.Close_Connection(); client.close();} catch (IOException e2){CONNUMBER = 0;serv.Close_Connection(); } }

The above code is derived from [8]. The complete 2270 line source code can be found on the disks attached to this document.
5.2 Simulation Web site

Figure 5.2 presents the Java applet as it appears at the bottom of the Web page with address http://nagasaki.rau.ac.za/rob_web. The rest of the Web page contains explanations on how to use the applet, as well as photographs of the actual robotic cell. The file containing the above, index.html, as well as all the photographs and compiled applet are available on the disks supplied with this document.

It is important to note that the above applet runs best on Netscape Navigator Gold Version 3.01 and 4.01/3. Users using Internet Explorer may experience problems with the positioning of the different objects. The use of the applet together with the different functionalities will now be discussed under the following headings:

- Communication Check
- Access Password
- Control Options
- Part Gripped
The operation of the applet will be explained under the assumption that the user is accessing it through the above-mentioned HTTP address. All data received by the applet is displayed in the **Receive from Robot Cell textbox**. Any previous data is displayed in the **Previous from Robot Cell textbox**. Any data sent from the applet is typed into the **Sending to Robot Cell textbox** after which the **Send button** is clicked. In some of the functions or commands, activated by the user, the above is done automatically, as will be discussed in the following sections. All communication between the applet and the Visual Basic simulation is handled, on the Visual Basic side, by the **Web Watch form**, discussed in section 4.2.7. See this section for all the valid commands recognised by the **Web Watch form** as well as the applet. The **Reset button** can be clicked to reset the applet to start-up status.

### 5.2.1 Communication Check

If the applet was loaded successfully, after accessing http://nagasaki.rau.ac.za/rob_web, the **Receiving from Robot Cell textbox** should display "Communication Check - OK". If this is not the case type "Restart" in the **Sending to Robot Cell textbox** and click the **Send button**. Next type "StartTest" in the **Sending to Robot Cell textbox** and click the **Send button**. If the "Communication Check - OK" message is not displayed in the **Receiving from Robot Cell textbox**, close your Web browser and re-open it. Do not just Reload or Refresh it. If two attempts are still unsuccessful the server communicating with the applet is down. If all the commands sent are returned in the **Receiving from Robot Cell textbox** with a "Server Echo--" prefix then the applet can be operated in a limited way but with no effect on the physical robotic cell, therefore the Java server software on Nagasaki is running but the Visual Basic simulation is not running (See Figure 3.1).

### 5.2.2 Access Password

A password is needed to gain access to the Visual Basic simulation package which in turn is connected to the actual robotic cell. If the robotic cell is not connected the simulation package will still respond with a message that the simulation is updated but that the robot is not connected. Type the password in the **Access Password textbox**. This password should be the same as the password entered on the **Web Watch form** on the Visual Basic simulation (See section 4.2.7) This password is send with each command to be recognised by the simulation package. The applet can be operated in a limited way without the password but with no effect on the robotic cell.
5.2.3 Control Options

Three Control Options exist namely Control R, Control P, Expert and None. After any of these controls have been activated, the None checkbox must first be activated before any other control is activated. Each of the above controls will now be discussed.

Control R
This control is used to manually control the robot with the scroll bars in the top left corner. To activate this control, click the Control R checkbox and wait for one of the following messages in the Receiving from Robot Cell textbox.

- "Robot Ready" means that the robot can manually be controlled.
- "Robot Not Connected" means just that.

Use the scroll bars to move the robot to the desired position. Click one of the Send buttons and the robot will update to the same position if the "Robot Ready" message is displayed. This command corresponds to the Simulation to Robot section on the Main form of the Visual Basic simulation. See section 4.2.1 and Figure 4.14.

Control P
This control is used to move the robot to pre-programmed positions A to HT. (See Figures 3.2 and 3.3) To activate this control click the Control P checkbox and wait for one of the following messages in the Receiving from Robot Cell textbox.

- "Robot Ready" means that the robot can be positioned at positions A to HT
- "Robot Not Connected" means just that.

Use the checkboxes under Move Robot to: section to select the desired position as presented by Figures 3.2 and 3.3. After selecting a position wait for a message "ROBOT AND SIMULATION UPDATED" or "SIMULATION UPDATED - Robot Not Connected". The possible destinations positions depend on the current position. All the possible destinations will
be clickable, the other ones will not react. This control corresponds to the Robot Paths form in section 4.2.5 of the Visual Basic simulation.

**Expert**

This control is used to start the expert system. To activate this control, click the Expert checkbox and wait for the "Expert Running" message that would appear briefly in the Receiving from Robot Cell textbox. The expert system takes a while to execute an action after the "Expert Running" message was shown. To stop the expert system, click the None checkbox. This control corresponds to the Expert form in section 4.2.6 of the Visual Basic simulation. While the expert system is running the activities in the robotic cell can be observed by the user.

**5.2.4 Part Gripped**

As previously discussed, four identical parts exist in the robotic cell. The Part Gripped section is used to grip and release the different parts. After loading the applet Part1 to 4 are situated at positions C to F with Part1 corresponding to position F, Part2 to position E etc. In order to grip a part click the corresponding checkbox (Part1 to Part4). Before the next part can be gripped the None checkbox need to be clicked. Important to note is that any part can be gripped or released at any time. This however does not correspond to reality in the robotic cell. Care must be taken, to make sure that the robot is at a position where a part exists before it is actually gripped. The same applies when moving to a position where a part already exists while having a part in the robot gripper. When a part is released anywhere accept at positions C to HT and I it will be dropped by the robot while the simulation will show it hanging where it was left.

At all times, wait for a readable/clear message to appear in the Receiving from Robot Cell textbox before interacting with the applet again. The above section presented the Internet user with access to the actual robotic cell. This example with limited functions and abilities is used in this specific study to demonstrate the remote on-line control and monitoring of the specific robotic cell used in this project.
6. Testing the System

In order to present an evaluation of the system, certain tasks are set up and the detailed steps and actions to complete this task are then given. Only a few of the important tasks are presented as tests. The tests are divided into two sections. The first section deals with the evaluating of the Visual Basic 5 simulation package, running on the computer connected to the Robot Controller, referred to as local testing. The second section deals with test performed via the Internet from the Java applet referred to as Internet testing. Before the local testing can be done, the PC (Diesel) connected to the Robot Controller, first needs to establish a communication connection with the Robot Controller. This is done by activating the virtual modem connection by double clicking the Dial Roblix shortcut on the desktop. A window will appear and the Connect button is clicked. After a while another window will appear on which the Continue button is clicked. To verify that the communication connection to the Robot Controller was established successfully, the following command is typed in a DOS window:

```
ping 152.106.20.237
```

If the communication was successfully established the following reply would appear:

```
Pinging Roblix[152.106.20.237] with 32 bytes of data:
Reply from 152.106.20.237: bytes=32 Time=...ms TTL=255
```

If the connection failed, check all the cable connections and repeat the above procedure. Next the Visual Basic 5 simulation package, called Robmove, can be started from the Robmove shortcut on the desktop. The Robmove programme is connected to the Robot Controller through the virtual modem by following the following steps: (See section 4.2 for a description of each form)

- Click the Connect button on the Main form
- Click the Connect item on the menu bar of the Robot Control form.
- Click the OK button on the Connect form

A progressing red bar would appear presenting the progress in establishing communication to the Robot Controller. A Connection Successful ! message box will appear when the connection is successfully established. The previous procedure takes about 3 minutes to complete. Click the OK
button on the message box. After the above message, the Robmove simulation package is ready to interact and communicate with the Robot Controller.

Make sure that the key on the Robot Controller with positions for

- **Automatic Mode**
- **Manual Mode slower than 250 mm/s**
- **100 % Manual Mode**

is switched to **Automatic Mode**.

Click the **Read System Status button**. If the **Controller State** displays a status of "Stand-by(Motors Off)" then click the **Motors On/Off button** and make sure that the status changes to "Run (Motors On)".

The **Hide button** on the **Robot Control form** can be clicked to return to the **Main form**. At the start of the tests the robot is assumed to be in axis position A \([10, 10, 10, 10, 10, 10]\) with parts located at positions C, D, E and F. (See chapter 3 for a full layout and explanation of the experimental set-up.) Also ensure that the air compressor tank, which the pneumatic gripper is operated from, is at a pressure of 4 bar.

### 6.1 Local Testing (Visual Basic 5 simulation)

For the local testing the following tasks are chosen:

- The robot is to be moved from its current axis position \([10, 10, 10, 10, 10, 10]\) to
- axis position \([50, -10, 0, 10, 10, 10]\) to
- axis position \([60, 10, 30, 10, -30, 10]\) to
- axis position \([20, 10, 30, 10, -30, 10]\) to
- axis position \([10, 10, 10, 10, 10, 10]\)

- While the robot is moved to the above positions generated a Rapid programme to move in a straight line from each position to the next with a speed of 150 mm/s
CHAPTER 6: TESTING THE SYSTEM

- Repeat the above but generate a Rapid programme which moves in joint movement from each position to the next with a speed of 150 mm/s.

- Downloading the above two generated programmes to the Robot Controller.

- Executing the above two programmes from the Robot Controller to verify the automatic programme generation while the simulation is following the physical robot.

- Manually retrieve a part from position F and place it in position I, on the Lathe, by using the pre-programmed paths.

- Read all the sensors in the system to verify the position of all the parts.

- Start the expert system and wait until it finishes or produces an error message.

- Verify that all the parts are assembled in positions G, H, HT and GT in the physical robotic cell. Verify the same for the simulated parts in the Visual Basic 5 simulation.

The above test is implemented as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Initiate manual robot control and move to the specified positions.</strong> Click the Robot Control Not Initiated label on the Main form and wait until the label turns green with the caption of Robot Control Initiated. The Axis textboxes and the simulation will update to the current physical robot axis position of [10, 10, 10, 10, 10, 10]</td>
</tr>
<tr>
<td>2</td>
<td>Set the speed to 150 mm/s by setting the Speed listbox, underneath the above label, to v150.</td>
</tr>
<tr>
<td>3</td>
<td>Check the Var Write checkbox.</td>
</tr>
<tr>
<td>4</td>
<td>Uncheck the Degree checkbox.</td>
</tr>
<tr>
<td>5</td>
<td>Click the Program button.</td>
</tr>
</tbody>
</table>
On the Programming form type v150 in the Speed textbox.

Click the Add button to add the current robot position in the position/orientation format to the current new programme.

Click the Done button.

**Move to next position**

Type the next robot position [50,-10,0,10,10,10] into the various Axes textboxes and click the Move button. The physical robot and simulation will update to this position. Alternatively used the six Axes scrollbars to position each axis.

**Add to the programme**

Click the Program button.

On the Programming form click the Add button to add the current robot position in the position/orientation format to the current programme.

Click the Done Button.

Repeat steps 8 to 12 with the robot position of [60,10,30,10,-30,10]

Repeat steps 8 to 12 with the robot position of [60,10,30,10,-30,10]

Repeat steps 8 to 12 with the robot position of [20,20,30,10,-30,10]

Repeat steps 8 to 12 with the robot position of [10,10,10,10,10,10]

Click the Program button.

On the Programming form click the Add button to add the current robot position in the position/orientation format to the current programme.

**Save the programme**

Enter the location and name of the current programme in the Save To: textbox as c:\test_1.prg.

Click the Save To: button.

Click the Clear button.

Click the Done button.

**Repeat above procedure with joint movement**

On the Main form check the Degree checkbox.

Repeat steps 5 to 17
CHAPTER 6: TESTING THE SYSTEM

Save the programme

24 Enter the location and name of the current programme in the Save To: textbox as C:\Test_J.prg.
25 Click the Save To: button.
26 Click the Done button.

Initialises robot to simulation control

27 Click the Not Active label in the Robot to Simulation section on the Main form.
28 Click the OK button on the Robot Stopped ! message box and wait for the label to turn green with a caption of Active.

Download the programmes

29 Click the Connect button on the Main form.
30 On the Robot Control form enter the following in the Local textbox: C:\Test_L.prg
31 Enter the following in the Remote textbox: ram1 disk:\Test_L.prg
32 Click the Download button.
33 Click the OK button on the Program Downloaded ! message box.
34 Repeat steps 30 to 33 with C:\Test_J.prg and ram1 disk:\Test_J.prg respectively.

Run the programme Test_L.prg

35 Enter the following in the Remote textbox: ram1 disk:\Test_L.prg
36 Click the Program Load button.
37 Click the OK button on the Program Loaded ! message box.
38 Click the Program Prepare button.
39 Click the OK button on the Program Prepared ! message box.
40 Click the Start button.
41 Click the OK button on the Robot Started ! message box.
42 Click the Hide button and check that the simulation follows the physical robot. Wait for the programme to finish.
Run the programme Test_I.prg

43 Click the Connect button on the Main form.
44 Repeat steps 35 to 42 with ram1disk\Test_I.prg.

Initialise robot path control

45 Click the Paths button on the Main form.
46 On the Robot Paths form click the Robot Control Not Initiated label and wait until the label turns green with the caption of Robot Control Initiated label.
47 Check the Run Robot and Update Simulation checkboxes.
48 Uncheck the MsgBox checkbox.

Manipulate the robot via the pre-programmed paths

49 Referring to Figure 3.3 the robot is currently at position A, thus use the Path listbox to select Path 7 in order to move to position B.
50 Make sure the Forward/Reverse checkbox is checked to indicate Forward
51 Click the Run button and wait for the robot to reach position B.
52 Click the Hide button.

Open the gripper

53 On the Main form click the Expert button.
54 On the Expert form check the Gripper Open/Closed checkbox and make sure the physical gripper opens.
55 Click the Hide button.

Manipulate the robot via the pre-programmed paths

56 On the Main form click the Paths button.
57 On the Paths form use the Path listbox to select Path 6 in order to retrieve the part at position F
58 Make sure the Forward/Reverse checkbox is checked to indicate Forward
59 Click the Run button and wait for the robot to reach position F.
60 Click the Hide button.
CHAPTER 6: TESTING THE SYSTEM

Close the gripper

61 Check the Grip Off checkbox to display Part 1 Gripped.
62 On the Main form click the Expert button.
63 On the Expert form uncheck the Gripper Open/Closed checkbox and make sure the physical gripper closes.
64 Click the Hide button.

Manipulate the robot via the pre-programmed paths

65 On the Main form click the Paths button.
66 On the Paths form use the Path listbox to select Path 6 in order to move the part at position F to position B
67 Make sure the Forward/Reverse checkbox is unchecked to indicate Reverse
68 Click the Run button and wait for the robot to reach position B
69 Repeat steps 66 to 68 with Path 7 to move the robot to position A
70 Repeat steps 66 to 68 with Path 8 in the forward direction to move the robot to position J
71 Repeat steps 66 to 68 with Path 9 in the forward direction to move the robot to position I
72 Click the Hide button.

Open the gripper

73 On the Main form uncheck the Part 1 Gripped checkbox.
74 Click the Expert button.
75 Release the part in the gripper by checking the Gripper Open/Close checkbox.

Manipulate the robot via the pre-programmed paths

76 Repeat steps 66 to 68 with Path 9 in the reverse direction to move the robot to position J.
77 Repeat steps 66 to 68 with Path 8 in the reverse direction to move the robot to position A
78 Click the Hide button.

Read all the sensors

79 On the Main form click the Expert button.
80 Click the Read Sensors button and check the various sensors have the respective values as follows:
Also check that the simulated parts are in the correct positions to correspond to the physical parts. Click the Hide button to view the Main form.

**Start the Expert system**

81 On the Main form click the Expert button.

82 Check the Load Facts checkbox.

83 Click the Start Auto button and observe the reasoning being performed in the various Expert form textboxes. The robot processing all the parts from the current configuration in the cell can be viewed by clicking the Hide button. Each action is delayed by more or less 10 seconds. A message box displaying Robot Stopped! will indicate that the expert system has finished. After the expert system has stopped, check that the physical parts are located in positions G, H, GT and HT. Also check that the simulated parts are in the same positions.

**Activating control from the Internet**

84 On the Main form click the Reset button.

85 Click the Web button.

86 On the Web form check the Take Action checkbox.

87 Select any password longer than 5 characters and enter it in the Password textbox.
Click the Start button and wait until the Start button's caption changes to Running.

Click the Hide button.

The above test was performed while having visual contact with the robotic cell. All the actions were completed successfully with all the physical and simulated parts being in all the correct respective positions during the test.

6.2 Internet Testing (Java applet)

Move the parts in the robotic cell to the start-up positions, C, D, E and F.

Before the Internet test is started, ensure that the Microsoft FrontPage Personal Web Server is running on Nagasaki. If not, it can be started by clicking the following in sequence:

```
Start - Programs - Microsoft FrontPage - Personal Web Server
```

Also ensure that the Java server on Nagasaki is running. If not, type the following in a DOS window in the c:\FrontPage Webs\Content\rob_web directory:

```
java server3
```

The following will be replied:

```
Server: Listening on port 6789
No Users Connected! Listening on port 6789
```

At the start of the tests, the robot is assumed to be in axis position [10,10,10,10,10,10] with parts located at positions C, D, E and F. This test is done with no visual contact with the actual robotic cell. This test can be done from anywhere in the world on a PC connected to the Internet. This test was done on a PC running Netscape Navigator Gold Version 3.01, as an Internet browser.

For the Internet testing the following tasks are chosen:

- Go to the Internet address http:nagasaki.rau.ac.za\rob_web
• Using the pre-programmed paths, move the part in place F to place I therefore position it on the Lathe.

○ Start the expert system and wait until it finishes.

• Manual move the robot to axis position \([60,10,30,10,-30,10]\)

○ Manual move the robot to axis position \([10,10,10,10,10,10]\)

• Verify that all the parts are assembled in positions G, H, HT and GT in the physical robotic cell. Verify the same for the simulated parts on the Java applet, as well as in the Visual Basic 5 simulation.

The above test is implemented as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Viewing the Web page containing the Java Applet</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Type <a href="http://nagasaki.rau.ac.za/rob_web">http://nagasaki.rau.ac.za/rob_web</a> in the location textbox of Netscape. The page contains some background and the main objective of the study. It also includes a detailed description on the use of the applet including photo’s of the physical robotic cell.</td>
</tr>
<tr>
<td>2</td>
<td>Go to the bottom of the page where the applet is situated and confirm the message “Communication check OK” in the Receiving from Robot Cell textbox.</td>
</tr>
<tr>
<td><strong>Enter password</strong></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Type the previously selected password into the Access Password textbox.</td>
</tr>
<tr>
<td><strong>Activate robot path control</strong></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Click the Control P checkbutton and wait for a “Robot Ready” message in the Receiving from Robot Cell textbox</td>
</tr>
</tbody>
</table>
Move to position F via A and B

5 The applet starts with position B as default. Click the checkbutton A in the Move Robot to: section to synchronise the applet and physical robot. Wait for the “Simulation and Robot Updated” message to appear in the Receiving from Robot Cell textbox.
6 Repeat step 5 with checkbutton B.
7 Repeat step 5 with checkbutton F.

Grip Part 1

8 Click the Part1 checkbutton in the Part Gripped section to grip the part in position F and wait for the “Simulation and Robot Updated Gripper Closed” message to appear in the Receiving from Robot Cell textbox

Move to position I, the Lathe, via B, A and J

9 Repeat step 5 with checkbutton B.
10 Repeat step 5 with checkbutton A.
11 Repeat step 5 with checkbutton J.
12 Repeat step 5 with checkbutton I.

Release Part 1

13 Click the Part1 checkbutton in the Part Gripped section to release the part in position I and wait for the “Simulation and Robot Updated Gripper Opened” message to appear in the Receiving from Robot Cell textbox
14 Repeat step 5 with checkbutton J.
15 Repeat step 5 with checkbutton A.

Activate the expert system

16 Click the None checkbutton in the Control Options section and wait for the “Robot Stopped!” message to appear in the Receiving from Robot Cell textbox
17 Click the Expert checkbutton in the Control Options section. The expert system will start and the progress will be visible in the applet. Wait for the “No Parts Left! Expert Stopped” message to appear in the Receiving from Robot Cell textbox
CHAPTER 6: TESTING THE SYSTEM

Activate the robot manual control

18 Click the None checkbutton in the Control Options section and wait for the “Robot Stopped!” message to appear in the Receiving from Robot Cell textbox.

19 Click the Control checkbutton in the Control Options section and wait for the “Robot Ready” message to appear in the Receiving from Robot Cell textbox.

Manipulate the robot manually

20 Type the axis position \([60,10,30,10,-30,10]\) into the respective six Axes textboxes.

21 Click the Move button.

22 Click one of the Send buttons and wait for the “Simulation and Robot Updated” message to appear in the Receiving from Robot Cell textbox.

23 Repeat steps 20 to 22 with axis position \([10,10,10,10,10,10]\).

24 Click the None checkbutton in the Control Options section and wait for the “Robot Stopped!” message to appear in the Receiving from Robot Cell textbox.

Verify visually that the configuration on the applet, Visual Basic 5 simulation and physical robotic cell are the same. Also verify that the various part positions correspond.

The above test is performed while having no visual contact with the robotic cell. All the actions were completed successfully with all the physical and simulated parts being in all the correct respective positions after the test.

The above two tests, although implementing only some of the functions available in the software, have proven that the robotic cell can be remotely, on-line manipulated and controlled. The non-visual contact test performed on the applet, further proves that the simulation on the applet supplied sufficient information regarding the physical robotic cell to remotely control it by having observed a simulated robotic cell.

6.3 Transponder System Testing

The purpose of this test is to prove that the robot can find and identify a certain part with an attached transponder within the robotic cell. The above would then serve as an example to show the increase in the intelligence within the robotic cell by adding the transponder system. This test is performed
manually in the sense that the robot arm is moved throughout the robotic cell by way of the pre-programmed paths until the specified part is detected and identified by the transponder system. Due to the loss in received data from the Reader Module, as explained in section 4.2.8, the above search and identification function is not included into the expert system and only simulate how such a function would operate if included into the expert system. As stated in section 4.2.8 the transponder system enables the robot arm to search, identify, as well as read relevant information from a certain part’s transponder where, without the transponder system some information about a certain part is assumed because of the parts position. If included in the expert system, the transponder system would therefore eliminate certain assumptions and replace the assumptions with detailed knowledge, obtained from the transponder system, to use in making more useful and decisive conclusions and therefore resulting in a higher level of intelligence within the robotic cell.

This test is divided into two parts. The first part is used to test the performance of the manufactured antenna, attached to the pneumatic gripper, as well as the manufactured transponder attached to one of the four parts. The second part of the test deals with verifying and demonstrating the recognition and identification of a specific part within the robotic cell.

For the second part of the test the following tasks are chosen:

- By using the bought transponder cards, determine the maximum distance in the 5 directions X, -X, Y, -Y and Z, indicated on Figure 6.1, that the card transponder can still be read.

- Repeat the above with the manufactured transponder.
Also measure the antenna current and voltage, as well as the phase angle between antenna voltage and current. Also indicate the tuning capacitance of the antenna.

Table 6.1 holds the results of the above test. The antenna current is measured by inserting a 1 Ω resistor in series in the antenna circuit and therefore measuring the voltage across the resistor to indicate the current as explained in section 2.7.1. Both voltage and current were measured by an oscilloscope and is also given.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Transponder Card</th>
<th>Manufactured Transponder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance cm</td>
<td>X</td>
<td>-X</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 6.1 Transponder reading distances.

The following antenna parameters are measured:

- Antenna Current: 191 mA (Peak Current)
- Antenna Voltage: 6.2 V (Peak Voltage)
- Antenna Phase angle: Voltage 10° before Current.
- Tuning Capacitance: 6.744 nF.

Important to note, is that during the above test, the orientation of the transponder relative to the orientation of the antenna, is not taken into account. Areas exist where the antenna is unable to communicate with the transponder, although within the range of Table 6.1, due to the specific orientation of the transponder. The values of the Transponder Card, given in Table 6.1 compare very well with the design range of 36 cm, presented in section 3.8. In the Z direction the reading range was 2.7 % more than designed for. However, the other directions were 20 % less.

The Manufactured Transponder did not perform very well, since it has a reading range of about 50 % of what it was designed for. This can be due to the fact that relatively thick copper wire is (0.01 cm) used in the construction of the transponder coil (See section 3.8.8) whereas the Transponder Cards used 0.007 cm diameter copper wire. Another reason is that the coil might not have been properly tuned, to 460 turns, by the procedure in section 3.8.8.

The calculated tuning capacitance of the antenna, $C_{\text{tune}}$, as calculated in section 3.8.7, as 5.78 nF was only out by 16.6 % compared to the manually tuned capacitance of 6.744 nF.
For the second part of the test the following tasks are chosen:

- One of the four parts within the robotic cell is equipped with the manufactured transponder. (See section 3.8.8 for the transponder construction)

- This part is placed at position E in the robotic cell. The three other parts are placed in positions I, C and F.

- The robot is required to manually be moved from the starting position A, by way of the pre-programmed paths, until the part containing the transponder is gripped in the gripper with the robot arm back in position A.

The second part of the test is performed as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the Paths button on the Main form.</td>
</tr>
<tr>
<td>2</td>
<td>On the Robot Paths form click the Robot Control Not Initiated label and wait until the label turns green with the caption of Robot Control Initiated.</td>
</tr>
<tr>
<td>3</td>
<td>Check the Run Robot and Update Simulation checkboxes.</td>
</tr>
<tr>
<td>4</td>
<td>Uncheck the MsgBox checkbox.</td>
</tr>
</tbody>
</table>

**Initialise robot path control**

**Manipulate the robot via the pre-programmed paths**

Referring to Figure 3.3 the robot is currently at position A, thus use the Path listbox to select Path 7 in order to move to position B.

Make sure the Forward/Reverse checkbox is checked to indicate Forward.

Click the Run button and wait for the robot to reach position B.

Click the Hide button.

Click the Transp button on the Main form.

On the Transponder form click the GetVersion button and check that the system returns “16-12-9612101000152” in the Data textbox.
Make sure the **Poll checkbutton** is checked and click the **Con - Stop button** to start polling for transponders in the antenna field.

Check that the red **No Tag label** turns to green displaying **Tag Found** indicating that a tag exists in the antenna field.

Click the **Hide button**

On the **Main form** click the **Paths button**.

---

**Check position C for transponder**

On the **Paths form** use the **Path listbox** to select Path 6 in order to move to position F.

Make sure the **Forward/Reverse checkbox** is checked to indicate **Forward**

Click the **Run button** and wait for the robot to reach position F.

Click the **Hide button**.

On the **Main form** click the **Expert button**.

On the **Expert form** check the **Gripper Closed checkbox** to close the gripper.

Click the **Hide button**.

On the **Main form** click the **Paths button**.

On the **Paths form** use the **Path listbox** to select Path 6 in order to move to position B.

Make sure the **Forward/Reverse checkbox** is checked to indicate **Reverse**

Click the **Run button** and wait for the robot to reach position B.

Use the **Path listbox** to select Path 7 in order to move to position A.

Make sure the **Forward/Reverse checkbox** is checked to indicate **Reverse**

Click the **Run button** and wait for the robot to reach position A.

Click the **Hide button**.

On the **Main form** click the **Transp form**

On the **Transponder form** check that the **No Tag label** displays **No Tag in red**.

Click the **Hide button**.

On the **Main form** click the **Paths button**.

Repeat steps 13 to 21 but retrieve the part at position E via Path 4.

On the **Transponder form** check that the **No Tag label** displays **Tag Found in green**.

Therefore during the above test, the part containing the transponder was searched for and found among the other parts. This part was successfully gripped and taken to position A. The above
manually performed search procedure can be implemented as rules and added to the expert system's rule base. The search procedure can then be executed automatically and would also be integrated into the rest of the expert system.
7. Future Developments and Conclusion

Since the start of this study many developments regarding the use of Java and VRML in the field of robotics as a simulation tool have evolved. The interest in the Internet as a simulation tool seems to continue to grow very fast. A very good example of an interactive Java/VRML robot model can be found at the Web site http://www.vruniverse.com/. Future continuation of this study should definitely include the replacement of the wire frame model in the Java applet, with a combined VRML and Java model in which the robotic cell can be modelled very close to reality as a 3-D solid colour model as in the above-mentioned Web site. Similar to the above, a robot manufacturer can supply the Internet user with the ability to build a robotic cell as well as manipulate the robots in the cell. This will enable the Internet user or prospective customer to virtually test the robotic cell according to his/her requirements. The same applies to the manager of several already established robotic cells, already simulated on the Internet.

The remote control of robots or tele-robotics is also becoming very important as discussed in chapter 2. NASA is investing large amounts of money in various projects related to tele-robotics and the visualisation and simulation of remote environments. There is therefore very little doubt that the remote control, visualisation and real-time simulation of remote environments will continue to grow and eventually become the standard elements for remote control.

In addition to the recommendation of incorporating VRML into the simulation, the addition of additional AI components, besides expert systems, is strongly recommended. This could, for example, include automatic path generation and automatic part assembly by neural network [32],[4], to only name two. Very sophisticated sensors[24] are also currently available to increase the level of input into a robotics system that would enable AI components to deliver more intelligent and advanced suggestions and results.

This study presents a basis for any real-time remote control and visualisation project also incorporating the identification of various objects through transponder tagging. A specific physical experimental set-up, as a specific case of the above-mentioned general concept, is used as well as the simulation thereof, accessible locally, as well as from the Internet. The system, also incorporating an expert system, performed very well and lived up to the specific goals specified for the on-line control, simulation, visualisation and identification regarding the objects in the physical and simulated system. The transmission of data between the different PC’s worked very well, even though a non-standard communication method had to be developed to overcome certain dedicated software limitations (See
A more generalised communication layout, which eliminates the non standard communication method, together with the Java source code, is presented which can be applicable to any applet/server application. The transponder system also performed very well, although some problems regarding the Visual Basic implementation were encountered. As a conclusion it can be stated that the system seen as a whole succeeded to “move” the physical robotic cell in the form of a simulation to the user to interact on-line with the physical robotic cell through the simulated robotic cell and the AI component, the expert system, as well as the transponder system.

The findings and accomplishments of this project can thus be summarised as follows:

- The successful development of an on-line real time 3-D simulation of a remote robotic cell.

- The successful integration of different systems to increase the level of intelligence in the robotic cell (expert system, transponder system and collision detection system).

- The successful generation of a user-friendly interface to the robotic cell.

- The successful integration of the above into the Internet to provide access to any user with an Internet connection.

- Successfully implementing the above on the average PC (Pentium 166 32MB RAM).

The problems and shortfalls experienced within this project can be summarised as follows:

- Insufficient accesses and control over the Robot Controller resulted in non-standard and inferior methods of communication.

- Insufficient information available on the detail operation of the Robot Controller especially regarding singularity points and inverse kinematic solution.
The main knowledge contribution to the field of remote controlled robotics is the above mentioned intelligent on-line remote controlled robotic cell with artificial intelligence components, identification components and all of the above mentioned components included with which the remote/internet user can intelligently interact while having a detailed 3-D visual representation of the real robotic cell as well as all the relevant information regarding the current condition in the cell. This cell, although very simple, presents a basis for any on-line remote/internet controlled manufacturing environment.

The above system can be extended to a much larger scale by using the same principles and components used in this project. The software can be extended to accommodate huge existing expert systems packages to provide the knowledge to control much more complicated robotic cells. This project can also be used to develop a system in which similar robotic cells in different locations are controlled, via the Internet, from one central location. Thus the production and error diagnostics can be closely controlled and monitored.

The following suggestions about future development of this specific project can be summarised as follows:

- The incorporation of VRML as a 3-D modelling tool into the Internet (Java) interface.
- The incorporation of more AI components, especially fuzzy logic and neural networks to be able to do automatic path generation and optimum scheduling.
- Extending the current limited transponder system to one with a larger reading range.
- Developing the ability to determine the location of all transponders in the robotic cell.

The continuation of this project is strongly recommended and the reader is urged to install the Visual Basic simulation and interact with it. The install disks are supplied with this document.
References


REFERENCES, PRODUCT MANUALS AND INTERNET REFERENCES

Product Manuals

ABB (Asea Brown Boveri) IRB 2400 Industrial Robot and Controller
(http://www.abb.se/robotics/)


Phillips/Mikron HT RM800 Long Range Reader Module
(http://207.87.1.43/identification/products/contactless/)


Internet References

Animal Identification
http://www.electronicidinc.com/

Australia's Telerobot on the Web
http://telerobot.mech.uwa.edu.au/

Borland International

Crosspoint manufacturers of EAS transponders
http://rapidttp.com/transponder/suplicros.html

Interactive Cincinnati and Adept robots
http://www.vruniverse.com/index.html

Internet Statistics
http://www4.mids.org/topics/index.htm

Microsoft
http://www.microsoft.com

NASA Space Telerobotics Program
http://ranier.oact.hq.nasa.gov/telerobotics.html

Phillips/Mikron Identification.
http://207.87.1.43/identification/

Real Robots on the Internet
http://ranier.oact.hq.nasa.gov/telerobotics_page/realrobots.html
REFERENCES, PRODUCT MANUALS AND INTERNET REFERENCES

Robotics Internet Resource Page
http://piglet.cs.umass.edu:4321/robotics.html,

Robotics Technology Catalogue
http://ranier.oact.hq.nasa.gov/telerobotics_page/technologies.html

Sun Microsystems
http://www.sun.com

Transponder News
http://rapidttp.com/transponder/

Trolleyscan
http://www.trolleyscan.co.za

VRML and Robotics
http://www.op.dlr.de/FF-DR-RS/Joerg.Vogel/Vrml/lib.html

VRML related sites
http://www.vruniverse.com/

VRML Robot Model

WITS (Web Interface for Telescience)
http://mars.graham.com/wits/

WorkSpace 4 Robot Simulations Ltd
http://www.rosl.com/index.htm
Appendix A

Experimental set-up wiring diagram and sensors

This appendix contains the wiring diagram for the experimental set-up including the technical information regarding the connections to the Robot Controller including the wiring diagram to connect the Robot Controller to the PC via a RS 232 port. It also includes the details of the infra red sensors.
Wiring Diagram of Production Equipment, Materials Handling and Assembly sections

Figure A1. Wiring diagram for the Production Equipment, Materials Handling and Assembly sections
**Technical specification**

<table>
<thead>
<tr>
<th>Type of board</th>
<th>Digital</th>
<th>Analog</th>
<th>Max. boards¹ of each type</th>
<th>Power supply</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In</td>
<td>Out</td>
<td>Voltage output</td>
<td>Current output</td>
</tr>
<tr>
<td>System board</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital I/O</td>
<td>16</td>
<td>16</td>
<td>6²</td>
<td>Internal/External³</td>
</tr>
<tr>
<td>Analog I/O</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>Internal/External</td>
</tr>
<tr>
<td>AD Combi I/O</td>
<td>16</td>
<td>16</td>
<td>1</td>
<td>Internal/External³</td>
</tr>
<tr>
<td>Remote I/O Allen Bradley</td>
<td>128</td>
<td>128</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

1. A total of up to six I/O boards (but max 256 channels totally excluding system board channels), in addition to the system board, can be installed.
2. The small cabinet limits the number of terminal units and external connectors.
3. The digital signals are supplied in groups, each group having 8 inputs or outputs.
4. A maximum of one board with analog signals.
5. Takes up two board slots.

**Signal data**

**Digital inputs** (options 20x/238 + 31x/33x)
- Optically-isolated
- Rated voltage supply, 19-35 V,
- Logical voltage levels: "1" 15-35 V
- "0" 0-5 V
- Input current at rated input voltage: 5.5 mA
- Maximum potential difference: 500 V
- Time intervals:
  - System board time intervals: ≤ 8 ms (hardware) + 1-11 ms (software)
  - ≤ 1.5 ms (hardware) + < 2 ms (software)

**Digital outputs** (options 20x/238 + 31x/33x)
- Optically-isolated, short-circuit protected
- Voltage supply, 19-35 V, nominal 24 V DC
- Minimum voltage drop on output: 2 V
- Load per output: 200 mA
- Load per group of 8 outputs: 1 A
- Maximum potential difference: 500 V
- Time intervals:
  - < 150 μs (hardware) + < 2 ms (software)

**Digital outputs via relay unit** (options 20x/238 + 37x)
- Load per output: 4 A
- Load per group of 8 outputs: 6.3 A
- Voltage range (source): 250 V AC

**Digital inputs via 120 V AC modules** (options 20x/238 + 35x)
- Voltage range: 90-140 V
- Input current: < 8 mA

---

Product Specification IRB 2400 M94A
Figure A1 presents the wiring diagram for the Production Equipment, Materials Handling and Assembly sections. The mechanical limit switches 1 to 3 and C to H switch if pressure is applied to the triggering part. When this happens the wires connected to the switch is connected. The infra red sensors connects the two wires by way of a relay which is discussed in detail below.

**Wiring Diagram of Robot Controller to PC connection via RS 232 line**

The communication line at the Robot Controller is connected to terminal SIO 1 and on the other end connected to a 25 Pin serial socket. The exact layout is given in Figure A2.

![Wiring Diagram of Robot Controller to PC connection via RS 232 line](image)

**Figure A2. Wiring diagram for PC to Robot Controller link**

**Infra red Sensors**

The sensor consists of a transmitter, transmitting an infrared beam, and a receiver, receiving the beam and switching a relay. If the beam is broken, the relay will switch. The relay contacts is then used to connect the two wires connected to the Robot Controller.

The transmitter is based on a 555 oscillator driving an infra red transmitting diode. (See Figure A3) The frequency of this signal is about 5 kHz. The pulsed beam is encoded in a manner that allows the receiver to differentiate between the signal and interference that may be emitted from lights etc. in the area. The high frequency also allows the receiver to be AC coupled. This will make it immune to slow changing light levels. Power in the transmitter is saved by pulsing the signal.
Floppy disc unit, data ports, batteries
In the receiver, LED1 is an infra red receiving diode which is reversed biased by R1. (See Figure A4) Infra red light hitting the diode causes it to conduct. This results in a voltage drop at the junction of R1 and LED1. The signal is AC coupled to the op-amp by C1. This gives the signal a high degree of immunity to interference. The op-amp IC1 is connected as a non-inverting AC amplifier, R2 and R3 bias the input to mid-rail. The gain of the amplifier is set by the combination of R4, R5 and C2. At the frequency of about 5 kHz the gain of the op-amp is a few thousand. The output of the op-amp is AC coupled to the voltage doubler circuit of D1 and D2, thus charging the capacitor C4 sufficiently to switch transistor TR1 and relay RL1. When the beam is interrupted C4 discharges via R7 and the relay is turned off. Resistor R5 and capacitor C5 provide additional power supply decoupling to the amplifier. Diode D3 provides the back EMF protection for TR1.

Figure A2. Receiver circuit diagram

All the receivers and transmitters is powered by a 9 V power supply.

The following is a list of all the parts:
Transmitter

R1, R3  470E
R2  2.2K
C1  47 nF 50 V ceramic (473)
C2  0.1 uF 16 V radial
RV1  4.7 K preset (horizontal)
IC1  NE 555 + base
LED1  I/R transmitting diode (blue tint)

Receiver

R1, R4, R5  47K
R2, R3, R7  100K
R6  100E
R8  5.6K
C1  10 nF 50 V ceramic (103)
C2  100 nF 50 V ceramic (105)
C3, C4  2.2 uF 16 V radial
C5  100 uF 16 V radial
LED1  I/R receiving diode (clear)
IC1  LM 471 + base
D1, D2, D3  1N4148
RLA  LZ12 relay
Appendix B


This appendix contains the detail and assembled drawings for the Finger Extension of the pneumatic Gripper as well as the Modelled Robotic Cell Dimensional layout, Parts and Part locator.
Finger Extension

2 x M4 nuts
Non threaded Connector
2 x M4 nuts
M4 threaded rod
Threaded Connector
M4 Locknut
2 x Side Plates

Finger Base
Finger
Finger adjustment holes

The Finger Extension is assembled with 5 M4 x 25 mm bolts (Not Shown)

Figure B1. Assembled Finger Extension.

All holes Ø 4 mm

2 x Side Plates
2 x Connectors
1 x Finger

Figure B2. Aluminium components for each Finger Extension
Holes for fastening Finger Base to pneumatic Gripper with 2 x M4 x 20mm bolts

Figure B3. PVC Finger Base
Modelled Robot Cell Dimensional layout

Base and World co-ordinate system situated at the base of the robot

Figure B4. Dimensional layout of Modelled Robot Cell
Part 35 x 35 x 55

Figure B5. Enlargement of Assembly section
Part Dimensions

Figure B6. Part Dimensions and cutaway.

Figure B7. Part Locator on Lathe.
Figure B8. Part located on Part Locator
Appendix C

Transponder Reader Module to PC, Power Supply and Antenna wire diagramme.

This appendix contains the wiring diagram for connecting the transponder Reader Module to the PC via a RS 232 serial line as well as to the Power Supply and Antenna. The above is presented in Figure C1.
Figure C1. Transponder Reader Module to PC, Power Supply and Antenna wire diagram.
Appendix D

Communication to Reader Module details

This appendix contains an explanation and example as well as details concerning the commands send to the reader module as well as the security calculations used in this study.
Communication to Reader Module details

Whenever commands are send to the Reader Module, it is send in a certain format as explained in the attached part of the manufacturers manual on page 10 (104) and page 11 (104).

An example of the operation of the Poll Tags command, polling for HITAG1 Standard Protocol Mode as well as Miro/HITAG2 Public Mode A, is given below.

From page 39 (104) of the manufacturers manual it can be seen that the command to set the Reader Module in Polling mode consists of the following four parts:

\[
\text{0x03} \ 'I' \ \text{mode} \ \text{BCC}
\]

Each of the above parts represent 8 bits or alternatively an ASCII character. The only variable in the above command is "mode". According to page 39 (104) the "mode" setting to poll for HITAG1 Standard Protocol Mode as well as Miro/HITAG2 Public Mode A transponders correspond to the following 8 bit number:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 1 1</td>
<td>3</td>
</tr>
</tbody>
</table>

From any ASCII table it can be seen that the value of character "I" corresponds to a hexadecimal number of 0x6C or in binary as 01101100. According to the definition on Page 10 (104), BCC (Block Check Character) can be calculated by the XOR addition of all the previous binary representations of the respective characters as follows:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Character</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Length</td>
<td></td>
<td>0x03</td>
<td>0000 0011</td>
</tr>
<tr>
<td>Block Title</td>
<td>I</td>
<td>0x6C</td>
<td>0110 1100</td>
</tr>
<tr>
<td>mode</td>
<td></td>
<td>0x03</td>
<td>0000 0011</td>
</tr>
<tr>
<td>BCC</td>
<td>I</td>
<td>0x6C</td>
<td>0110 1100</td>
</tr>
</tbody>
</table>
If the supplied C-libraries are used the above command can be send to the Reader Module by calling the function below with mode = 0x03:

    void proloc_PollTags (BYTE_T mode, char *data);

The response from the Reader Module is returned in the character string “data”.

In the Visual Basic program used in this study the MSComm control, MSComm1, is used. Any data send to the Reader Module is send through the MSComm1.Output command. Thus the above polling command is send to the Reader Module as:

    MSComm1.Output = Chr(3) + "1" + Chr(3) + "1"

Where the Chr() command in Visual Basic returns the ASCII character representation of the decimal value in brackets.

The response from the Reader Module in this case is intercepted through the On_Comm event as described in section 4.2.8.
3 Communication Reader-Host

3.1 Introduction

The host (e.g. PC) communicates with the contactless 125 kHz read/write device via a serial interface using a baud rate of 9600 baud. Data transfer details are: 1 start bit, 8 data bits, 1 stop bit and no parity bit, the Least Significant Bit is sent first. Each communication sequence consists of a block of bytes sent by the host, and a block of bytes answered by the reader. All bytes are transmitted transparently, i.e. you can use any character between 0x00 and 0xFF.

Block Length:
Block Length is the sum of all transferred bytes including Block Length but excluding BCC.

Block Title:
The Command Byte if sent from host to reader.
The Status Byte if sent from reader to host.

Data:
Data bytes are only transmitted if data is transferred.

BCC:
The BCC (Block Check Character) is calculated by bytes 1 to n-1 (n=number of bytes of the whole communication sequence).

A different BCC calculation in Operating Mode (mode of the reader for using standard commands) and in KeyInit Mode (mode of the reader device for using personalization commands) helps to avoid the overwriting of secret data accidentally.

BCC calculation in Operating Mode of the reader:
The BCC is computed by EXOR-operation of all block data bytes including Block Length.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>EXOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Example for command GetSnr:

- Byte 1: Block Length 0000 0010 0x02
- Byte 2: Command Byte 0100 0111 0x77
- Byte 3: BCC 0100 0101 0x45

BCC calculation in KeyInit Mode of the reader:
The BCC is computed by adding all block data bytes including Block Length. The least significant eight bits are used as BCC.
3.2 Ordinary Protocol

If only a single read/write device with a node address equal to zero is connected to the host (e.g. on a RS232 serial line) the Ordinary Protocol is used to address this reader.

Format of the Ordinary Protocol (HOST→READER and READER→HOST):

<table>
<thead>
<tr>
<th>Byte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>......</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Block Length</td>
<td>Block Title</td>
<td>data</td>
<td>data</td>
<td>......</td>
<td>BCC</td>
</tr>
</tbody>
</table>

3.3 Extended Protocol

If more than one read/write devices with node addresses different from zero are connected to the host (e.g. on a RS485 serial line) the Extended Protocol is used to address a single reader.

Format of the Extended Protocol (HOST→READER and READER→HOST):

<table>
<thead>
<tr>
<th>Byte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>......</th>
<th>n-1</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Block Length + 0x80</td>
<td>Block Title</td>
<td>data</td>
<td>data</td>
<td>......</td>
<td>Node Address</td>
<td>BCC</td>
</tr>
</tbody>
</table>

Differences to Ordinary Protocol: Bit 7 of Block Length is set, and the Node Address is inserted just before BCC.

If a reader’s node address is different from zero, the reader enters net-mode. In this mode the reader expects all commands from the host to be sent in Extended Protocol including the right Node Address (except SetModuleAdr). If the host transmits a string that does not meet these conditions, the command is ignored, and there will be no answer from the reader (whereas a reader being not in net-mode - with node address equal to zero - would at least answer with a SERIAL ERROR message).

The command SetModuleAdr is used to assign a unique node address to a device whose serial number is known. This command should be sent in Ordinary Protocol. If the right serial number was sent, there will be an answer from the read/write device. This answer is sent in Ordinary Protocol if the former node address of the reader was zero, otherwise the answer is sent in Extended Protocol.

For communication in Extended Protocol use commands with 'Proloc_M'-prefix. For further information see Header File PROLBMU6.h.
3.4 Transfer Timeout Intervals

Character Delay:

Character Delay is the maximum time permitted to elapse between sending two consecutive characters of a block.

\[ \text{Character Delay} \leq 150 \text{ ms} \]

Block Delay:

Block Delay is only necessary if an error has occurred in the serial communication. To allow for re-synchronization in that case of malfunction there must be a minimum interval defined as Block Delay - until sending the next block.

\[ \text{Block Delay} \geq 160 \text{ ms} \]
If polling for HITAG 1 Advanced Protocol Mode was successful:

| 0x08 | Status | 0x80 | SNR-LSB | ---- | SNR-MSB | more | BCC |

*more*: Proximity Reader: *more* is always 0.
Long Range Reader: *more* equal to one indicates that there is at least one additional transponder in the reading area of the read/write device.

If polling for Miro / HITAG 2 Public Mode A was successful:

| 0x08 | Status | 0x02 | data[0] | ---- | data[4] | BCC |

If polling for PIT was successful:

| 0x13 | Status | 0x04 | data[0] | ---- | data[15] | BCC |

If polling for HITAG 2 Password Mode was successful:

| 0x08 | Status | 0x08 | SNR-LSB | ---- | SNR-MSB | config | BCC |

If polling for HITAG 2 Crypto Mode was successful:

| 0x08 | Status | 0x10 | SNR-LSB | ---- | SNR-MSB | config | BCC |

If polling for HITAG 2 Public Mode C was successful:

| 0x13 | Status | 0x20 | data[0] | ---- | data[15] | BCC |

If polling for HITAG 2 Public Mode B was successful:

| 0x13 | Status | 0x40 | data[0] | ---- | data[15] | BCC |

Status:  
0 ... no error  
-1 ... SERIAL ERROR
3.7.26 GetVersion

This command retrieves the serial number of the read/write device, the version number of the HITAG Communication Controller software and its date of creation.

**C-Function:** void proloc_GetVersion (char *data);

**Header-File:** PROLIB6.H

**Serial protocol:**

*HOST - READ/WRITE DEVICE*

0x02  "V"  BCC

*READ/WRITE DEVICE - HOST*

<table>
<thead>
<tr>
<th>0x1D.</th>
<th>Status</th>
<th>data[0]</th>
<th>data[26]</th>
<th>BCC</th>
</tr>
</thead>
</table>

data[0] ... data[7]: Version (format: Vx.yy.zz)
data[8]  ... data[15]: Date (format: dd-mm-yy)
data[16] ... data[26]: Serial number (11 characters)

Status:

- 0 ... no error
- -1 ... SERIAL ERROR
3.7.29 StopCommand

The command StopCommand interrupts the Permanent Reading Mode (activated after ReadMiro, ReadPit, ...) or the permanent writing mode (e.g. activated after WritePit) of the read/write device.

You should not use the command Reset instead, since Reset can cause undesirable side effects (resetting output pins).

C-Function: void proloc_StopCommand (void);
Header-File: PROLIB6.H

Serial protocol:

HOST - READ/WRITE DEVICE

| 0x02 | 0xA6 | BCC |

READ/WRITE DEVICE - HOST

| 0x02 | Status | BCC |

Status:
- 0 ... no error
- -1 ... SERIAL ERROR
3.7.45 SetPowerDown

This command turns the Long Range Reader into Standby Mode.

The byte \textit{-mode-} is set to zero for Standby Mode. To activate the amplifier again this byte must be set to one.

By default the read/write device is in Active Mode.

\textbf{C-Function:} \texttt{void proloc\_SetPowerDown (BYTE\_T mode);} \\
\textbf{Header-File:} PROLIB6.H

\textbf{Serial protocol:}

\textit{HOST - READ/WRITE DEVICE}

\begin{tabular}{c|c|c}
0x03 & 'D' & mode & BCC \\
\end{tabular}

\textit{mode:}

0x00 ... Standby Mode \\
0x01 ... Active Mode

\textit{READ/WRITE DEVICE - HOST}

\begin{tabular}{c|c|c}
0x02 & Status & BCC \\
\end{tabular}

\textit{Status:}

0 ... no error \\
-1 ... SERIAL ERROR
Appendix E

Expert system rule base.

This appendix contains the rule base of the expert system as implemented according to Figure 4.13. The following is a printout of the file robrule.txt, used by the expert system. This file can also be found on the disks attached to this document.
rule1: if
  Part_Gripped=no &&
  PROD_PartInMachine=yes
then
  actionT4=action2.

rule2: if
  Part_Gripped=no &&
  PROD_PartInMachine=yes
then
  actionT3=no.

rule3: if
  Part_Gripped=no &&
  PROD_PartInMachine=no
then
  actionT4=action1.

rule4: if
  Part_Gripped=no
then
  actionT=no.

rule5: if
  Part_Gripped=no
then
  actionT2=no.
rule6: if
   Part_Gripped=yes
then
   actionT3=no.

rule7: if
   Part_Gripped=yes
then
   actionT4=no.

rule8: if
   Part_Gripped=yes &&&
   Gripper_Closed=yes &&&
   PartIn_gripper=new
then
   actionT=action1.

rule9: if
   Part_Gripped=yes &&&
   Gripper_Closed=yes &&&
   PartIn_gripper=finished
then
   actionT=action2.

rule10: if
   Part_Gripped=no &&&
   Gripper_Closed=yes
then
   action=error1.

rule11: if
   Part_Gripped=yes &&&
   Gripper_Closed=no
then
   action=error1.

rule12: if
   actionT=action1 &&&
   Robot_at=I
then
   action=open_gripper.

rule13: if
   actionT=action1 &&&
   Robot_at=J &&&
   PROD_PartInMachine=no
then
   action=RunPath9.

rule14: if
   actionT=action1 &&&
   Robot_at=J &&&
   PROD_PartInMachine=yes
then
   action=error6.
rule14: if
    actionT = action1 &&
    Robot_at = A
then
    action = RunPath8.

rule15: if
    actionT = action1 &&
    Robot_at = B
then
    action = RunPath7R.

rule16: if
    actionT = action1 &&
    Robot_at = C
then
    action = RunPath1R.

rule17: if
    actionT = action1 &&
    Robot_at = D
then
    action = RunPath3R.

rule18: if
    actionT = action1 &&
    Robot_at = E
then
    action = RunPath4R.

rule19: if
    actionT = action1 &&
    Robot_at = F
then
    action = RunPath6R.

rule20: if
    actionT = action1 &&
    Robot_at = G
then
    action = error5.

rule21: if
    actionT = action1 &&
    Robot_at = H
then
    action = error5.

rule22: if
    actionT = action1 &&
    Robot_at = GT
then
    action = error5.
rule23: if
  actionT = action1 &&&
  Robot_at = HT
then
  action = error5.

rule24: if
  actionT = action2 &&&
  Robot_at = I
then
  action = RunPath9R.

rule25: if
  actionT = action2 &&&
  Robot_at = J
then
  action = RunPath8R.

rule26: if
  actionT = action2 &&&
  Robot_at = A
then
  action = RunPath7.

rule27: if
  actionT = action2 &&&
  Robot_at = B
then
  actionT2 = action1.

rule28: if
  actionT = action2 &&&
  Robot_at = C
then
  action = RunPath1R.

rule29: if
  actionT = action2 &&&
  Robot_at = D
then
  action = RunPath3R.

rule30: if
  actionT = action2 &&&
  Robot_at = E
then
  action = RunPath4R.

rule31: if
  actionT = action2 &&&
  Robot_at = F
then
  action = RunPath6R.
rule32: if
  actionT = action2 &&
  Robot_at = G
 then
  action = open_gripper.

rule33: if
  actionT = action2 &&
  Robot_at = H
 then
  action = open_gripper.

rule34: if
  actionT = action2 &&
  Robot_at = GT
 then
  action = open_gripper.

rule35: if
  actionT = action2 &&
  Robot_at = HT
 then
  action = open_gripper.

rule36: if
  actionT2 = action1 &&
  ASSEM_PartInG = no
 then
  action = RunPath5.

rule37: if
  actionT2 = action1 &&
  ASSEM_PartInG = yes &&
  ASSEM_PartInGT = no
 then
  action = RunPath10.

rule38: if
  actionT2 = action1 &&
  ASSEM_PartInG = yes &&
  ASSEM_PartInGT = yes &&
  ASSEM_PartInH = no
 then
  action = RunPath2.

rule39: if
  actionT2 = action1 &&
  ASSEM_PartInG = yes &&
  ASSEM_PartInGT = yes &&
  ASSEM_PartInH = yes &&
  ASSEM_PartInHT = yes
 then
  action = error4.
rule40: if
  actionT2=action 1 &&&
  ASSEM_PartInG=yes &&&
  ASSEM_PartInGT=yes &&&
  ASSEM_PartInH=yes &&&
  ASSEM_PartInHT=no
then
  action=RunPath11.

rule41: if
  actionT4=action 1 &&&
  Robot_at=I
then
  action=RunPath9R.

rule42: if
  actionT4=action 1 &&&
  Robot_at=J
then
  action=RunPath8R.

rule43: if
  actionT4=action 1 &&&
  Robot_at=A
then
  action=RunPath7.

rule44: if
  actionT4=action 1 &&&
  Robot_at=B
then
  actionT3=action 1.

rule45: if
  actionT4=action 1 &&&
  Robot_at=C &&&
  ASSEM_PartInC=yes
then
  action=close_gripperN.

rule46: if
  actionT4=action 1 &&&
  Robot_at=C &&&
  ASSEM_PartInC=no
then
  action=RunPath1R.

rule47: if
  actionT4=action 1 &&&
  Robot_at=D &&&
  ASSEM_PartInD=yes
then
  action=close_gripperN.
rule48: if
    actionT4=action1 &&&
    Robot_at=D &&&
    ASSEM_PartInD=no
then
    action=RunPath3R.

rule49: if
    actionT4=action1 &&&
    Robot_at=E &&&
    ASSEM_PartInE=yes
then
    action=close_gripperN.

rule50: if
    actionT4=action1 &&&
    Robot_at=E &&&
    ASSEM_PartInE=no
then
    action=RunPath4R.

rule51: if
    actionT4=action1 &&&
    Robot_at=F &&&
    ASSEM_PartInF=yes
then
    action=close_gripperN.

rule52: if
    actionT4=action1 &&&
    Robot_at=F &&&
    ASSEM_PartInF=no
then
    action=RunPath6R.

rule53: if
    actionT4=action1 &&&
    Robot_at=G
then
    action=RunPath5R.

rule54: if
    actionT4=action1 &&&
    Robot_at=H
then
    action=RunPath2R.

rule55: if
    actionT4=action1 &&&
    Robot_at=GT
then
    action=RunPath10R.
rule56: if 
  actionT4 = action1 &&
  Robot_at = HT
then
  action = RunPath11R.

rule57: if 
  actionT3 = action1 &&
  Gripper_Closed = yes
then
  action = Open_Gripper.

rule58: if 
  actionT3 = action1 &&
  Gripper_Closed = no &&
  ASSEM_PartInF = yes
then
  action = RunPath6.

rule59: if 
  actionT3 = action1 &&
  Gripper_Closed = no &&
  ASSEM_PartInF = no &&
  ASSEM_PartInE = yes
then
  action = RunPath4.

rule60: if 
  actionT3 = action1 &&
  Gripper_Closed = no &&
  ASSEM_PartInF = no &&
  ASSEM_PartInE = no &&
  ASSEM_PartInD = yes
then
  action = RunPath3.

rule61: if 
  actionT3 = action1 &&
  Gripper_Closed = no &&
  ASSEM_PartInF = no &&
  ASSEM_PartInE = no &&
  ASSEM_PartInD = no &&
  ASSEM_PartInC = yes
then
  action = RunPath1.

rule62: if 
  actionT3 = action1 &&
  Gripper_Closed = no &&
  ASSEM_PartInF = no &&
  ASSEM_PartInE = no &&
  ASSEM_PartInD = no &&
  ASSEM_PartInC = no
then
  action = No_Parts.
rule63: if
    actionT4 = action2 &&&
    Robot_at = I
then
    action = close_gripperF.

rule64: if
    actionT4 = action2 &&&
    Robot_at = J &&&
    Gripper_Closed = no
then
    action = RunPath9.

rule65: if
    actionT4 = action2 &&&
    Robot_at = J &&&
    Gripper_Closed = yes
then
    action = open_gripper.

rule66: if
    actionT4 = action2 &&&
    Robot_at = A
then
    action = RunPath8.

rule67: if
    actionT4 = action2 &&&
    Robot_at = B
then
    action = RunPath7R.

rule68: if
    actionT4 = action2 &&&
    Robot_at = C
then
    action = RunPath1R.

rule69: if
    actionT4 = action2 &&&
    Robot_at = D
then
    action = RunPath3R.

rule70: if
    actionT4 = action2 &&&
    Robot_at = E
then
    action = RunPath4R.

rule71: if
    actionT4 = action2 &&&
    Robot_at = F
then
    action = RunPath6.
rule72: if
   actionT4=action2 &&
   Robot_at=G
then
   action=RunPath5R.

rule73: if
   actionT4=action2 &&
   Robot_at=H
then
   action=RunPath2R.

rule74: if
   actionT4=action2 &&
   Robot_at=GT
then
   action=RunPath10R.

rule75: if
   actionT4=action2 &&
   Robot_at=HT
then
   action=RunPath11R.