

---

**INDUSTRIAL ROBOTS PROGRAMMING:  
BUILDING APPLICATIONS FOR THE FACTORIES  
OF THE FUTURE**

**INDUSTRIAL ROBOTS PROGRAMMING:  
BUILDING APPLICATIONS FOR THE FACTORIES  
OF THE FUTURE**

J. Norberto Pires  
Mechanical Engineering Department  
University of Coimbra, Portugal

 Springer

*J. Norberto Pires  
Mechanical Engineering Department  
University of Coimbra  
Portugal*

Industrial Robots Programming: Building Applications for the Factories of the Future

Library of Congress Control Number: 2006932582

ISBN 0-387-23325-3                      e-ISBN 0-387-23326-1  
ISBN 9780387233253

Printed on acid-free paper.

© 2007 Springer Science+Business Media, LLC

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

9 8 7 6 5 4 3 2 1

springer.com

Dedicated to the memory of my father Joaquim  
and to Dina, Rita, Beatriz and Olímpia.

---

## Foreword

Robots have traditionally been used to work in industrial environments, as they constitute the most flexible existing automation technology. In the recent years, manufacturing systems are becoming more autonomous requiring less operator intervention and a higher degree of customization and reconfigurability for disparate applications. In this scenario, robot programming is a key factor toward building the applications for the factories of the future.

This book by J. Norberto Pires constitutes a unique and authoritative reference in our professional field, as one of the very few books written by an academic with a strong industrial cut. The focus is on the software interfaces enabling humans and machines to effectively cooperate on the shopfloor. Several sensors and controllers are analyzed in detail, leading to the realization of interface devices using e.g. speech recognition and CAD models, and their versatility for a number of industrial manufacturing systems is enlightened.

Easy to read, rich in worked out examples and case studies, the book is complemented with additional experimental material available on a web site, including code and multimedia files, which the author promises to update regularly.

It is my conviction the book will be appreciated by a wide readership, ranging from technical engineers wishing to learn the foundations of industrial robotics to scholars and researchers wishing to understand the needs and the potential of a new generation of advanced industrial robots to be developed in the next decade.

### **Bruno Siciliano**

Professor of Control and Robotics at the University of Naples  
President-Elect of the IEEE Robotics and Automation Society

---

## Preface

A scientific and technical book is a starting point. A source of information for people browsing for details, a guide for others trying to build similar or related solutions, or a source of inspiration for yet others wondering about how things work.

This book was written by an engineer and university professor which has been active in the field of industrial robotics since 1994. It was planned, designed and built to serve engineers looking for better and more efficient systems, but also to serve academic readers interested in the robotics area. The book focus mainly on industrial robot programming in the beginning of the twentieth first century, namely on the important issues related with designing, building and operating flexible and agile robotic systems. It explores in detail the issue of software interfaces, but also input/output devices and several industrial and laboratory examples. In fact, the book uses several types of fully worked out examples to illustrate and clarify concepts and ideas, enabling the reader to see them working and even to test some of them. Most of the experimental material used in this book can be obtained from:

*<http://robotics.dem.uc.pt/indrobprog>*

This site will be updated regularly by the author constituting a source of information, code and multimedia files which complement the contents of the book.

Finally, the author wants to thank deeply to all the persons that contributed to this book, namely all his undergraduate and graduate students, specially his Ph.D. students Tiago Godinho and Germano Veiga, and his M.Sc. student Ricardo Araújo for their help and support in building and testing some of the solutions presented in the book.

J. Norberto Pires, Coimbra, Portugal, 2006

---

## Contents

<b>1. Introduction to the Industrial Robotics World .....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 A Brief History of the Industrial Robot.....	2
1.3 Using Robotics to Learn .....	9
1.3.1 Constitution of the Robot Nicola.....	12
1.3.2 Nicola Software.....	13
1.3.2.1 PLC Software .....	13
1.3.2.2 Software for the On-board PC.....	18
1.3.2.3 Feedback from the On-board Webcam .....	20
1.4 Using Robotics to Work .....	23
1.4.1 Using an Offline Simulation Environment.....	27
1.5 Statistics of Robotic Workers.....	29
1.6 Overview of the rest of the book .....	32
1.7 References.....	33
<b>2. Robot Manipulators and Control Systems.....</b>	<b>35</b>
2.1 Introduction.....	35
2.2 Kinematics .....	36
2.2.1 Direct Kinematics .....	39
2.2.2 Inverse Kinematics.....	43
2.3 Jacobian.....	47
2.4 Singularities .....	58
2.4.1 Brief Overview: Singularity Approach.....	61
2.5 Position Sensing.....	66
2.6 Actuators: Motors .....	69
2.6.1 Motor Drive System.....	72
2.7 Dynamics .....	75
2.7.1 Inertia Tensor and Mass Distribution .....	76
2.7.1.1 Important Results .....	77
2.7.2 Lagrange-Euler Formulation.....	78

2.7.3 D'Alembert Formulation .....	79
2.7.4 Newton-Euler Formulation .....	79
2.7.5 Dynamic Parameters .....	82
2.8 Matlab Examples .....	84
2.9 Robot Control Systems .....	86
2.9.1 Drive the motors to move the TCP and coordinate the motion for useful work .....	87
2.10 Servo Control .....	90
2.11 IO Control .....	91
2.12 Communication .....	92
2.13 Sensor Interface .....	94
2.13.1 Interfacing Laser 3D Sensor for Seam Tracking .....	94
2.13.2 Interfacing a Force/Torque Sensor .....	96
2.13.2.1 Using a Force/Torque Sensor .....	100
2.14 Programming and Program Execution .....	103
2.15 User Interface .....	104
2.16 References .....	105
<b>3. Software Interfaces .....</b>	<b>109</b>
3.1 Introduction .....	109
3.2 Low Level Interfaces .....	111
3.2.1 IO Digital Signals .....	111
3.2.1.1 Robot Controller Software .....	116
3.2.1.2 PLC Software .....	118
3.2.1.3 PC Software .....	120
3.2.2 Using Fieldbuses .....	122
3.2.2.1 Profibus (Process FieldBus) .....	122
3.2.2.2 CAN (Controller Area Network) .....	122
3.2.2.3 DeviceNet .....	123
3.3 Data Protocols and Connections .....	125
3.3.1 RPC – Remote Procedure Calls .....	131
3.3.2 TCP/IP Sockets .....	135
3.3.2.1 TCP Ports .....	138
3.3.3 UDP Datagrams .....	138
3.3.3.1 UDP Ports .....	139
3.4 Simple Example: Interfacing a CCD Camera .....	139
3.4.1 Robot Controller Software .....	141
3.4.2 Webcam Software .....	143
3.4.3 Remote Client .....	147
3.4.4 Using UDP Datagrams .....	153
3.5 Simple Example: Control Panel .....	156
3.6 Simple Example: S4Misc – Data Access on a Remote Controller .....	158
3.7 Industrial Example: Semi-autonomous Labeling System .....	162
3.7.1 Robotic Labeling System .....	163
3.7.2 System Software .....	164
3.7.3 Robot Controller Software .....	166
3.7.4 TCP/IP Server .....	167



3.7.5 Discussion .....	169
3.7.6 Conclusion.....	170
3.8 References .....	171
<b>4. Interface Devices and Systems .....</b>	<b>173</b>
4.1 Introduction .....	173
4.2 Speech Interfaces .....	178
4.2.1 Introduction .....	178
4.2.2 Evolution .....	180
4.2.3 Technology .....	183
4.2.4 Automatic Speech Recognition System and Strategy .....	184
4.2.5 Pick-and-Place and Robotic Welding Examples .....	186
4.2.6 Pick-and-Place Example .....	187
4.2.7 Robotic Welding Example.....	191
4.2.8 Adjusting Process Variables .....	195
4.2.9 Conclusion.....	197
4.3 VoiceRobCam: Speech Interface for Robotics .....	198
4.3.1 Robot Manipulator and Robot Controller.....	199
4.3.2 PLC Siemens S7-200 and Server.....	200
4.3.3 Webcam and Image Processing Software .....	203
4.3.4 User Client Application .....	205
4.3.5 Speech Interface .....	210
4.4 CAD Interfaces .....	215
4.4.1 Speech Interface for Welding .....	221
4.5 References.....	223
<b>5. Industrial Manufacturing Systems .....</b>	<b>225</b>
5.1 Introduction.....	225
5.2 Helping Wrapping Machines for the Paper Industry .....	226
5.2.1 Layout of the System .....	227
5.2.1.1 Station One – Dimensions and Weight.....	228
5.2.1.2 Station Two – Roll Wrapping and Inner Header .....	228
5.2.1.3 Station Three – External Header.....	229
5.2.1.4 Station Four – Labeling.....	230
5.2.2 EmailWare.....	233
5.2.2.1 EmailWare Application Example .....	237
5.2.3 Conclusions and Discussion .....	241
5.3 Complete Robotic Inspection Line for the Ceramic Industry.....	241
5.3.1 Motivation and Goals.....	242
5.3.2 Approach and Results .....	246
5.3.2.1 Basic Functioning of the De-palletizing System .....	248
5.3.2.2 Basic Functioning of the Palletizing System .....	248
5.3.3 Operational Results and Discussion .....	250
5.4 Handling Production Changes Online.....	251
5.4.1 Robotic Palletizing System.....	252
5.4.1.1 Identify Empty Pallets and Measure Parameters of an Empty Pallet .....	253

5.4.1.2 Pick a Glass from the Production Line .....	254
5.4.1.3 Palletize the Glass .....	254
5.4.2 System Software.....	256
5.4.3 On-line monitoring.....	260
5.4.4 Discussion and Results .....	263
5.4.5 Conclusion.....	265
5.5 References .....	265
<b>6. Final Notes.....</b>	<b>267</b>
6.1 Introduction.....	267
6.2 Operation “ <i>Albert</i> ” .....	268
6.2.1 And “ <i>Albert</i> ” Speaks.....	275
6.3 References .....	280
<b>Index .....</b>	<b>281</b>

## **Introduction to the Industrial Robotics World**

### **1.1 Introduction**

Robotics is a subject that leaves nobody indifferent. No matter if they are used to work in industry or at our homes, mimic some of the human capabilities, or used to access dangerous environments, launched to space, or simply used to play with, robots are always a source of interest and admiration. Here the focus is in robots used to work on industrial environments [1], i.e., robots built to substitute man on certain industrial manufacturing tasks being a mechatronic coworker for humans.

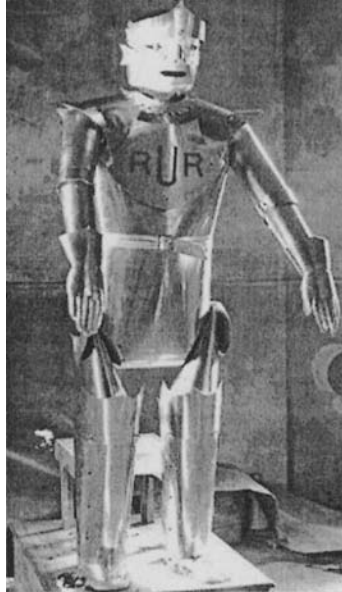
In fact, actual manufacturing setups rely increasingly on technology. It is common to have all sources of equipment on the shop floor commanded by industrial computers or PLCs connected by an industrial network to other factory resources. Also, manufacturing systems are becoming more autonomous, requiring less operator intervention in daily operations. This is a consequence of today's market conditions, characterized by global competition, a strong pressure for better quality at lower prices, and products defined in part by the end-user. This means producing in small batches, never risking long stocks, and working to satisfy existing customer orders. Consequently, concepts like flexibility and agility are fundamental in actual manufacturing plants, requiring much more from the systems used on the shop floor. Flexible manufacturing systems take advantage of being composed by programmable equipment to implement most of its characteristics, which are supported by reconfigurable mechanical parts.

Industrial robots are good examples of flexible manufacturing systems. Using robots in actual manufacturing platforms is, therefore, a decision to improve flexibility and to increase the agility of the manufacturing process. If the manufacturing processes are complex, with a low cycle time, and have a lot of parameterization due to the diversity of products, then using robots is the correct decision, although it isn't enough for a complete solution. In fact, engineers need to

integrate other technologies with the objective of extracting from robots the flexibility they can offer. That means using computers for controlling and supervising manufacturing systems, industrial networks, and distributed software architectures [2,3]. It also means designing application software that is really distributed on the shop floor, taking advantage of the flexibility installed by using programmable equipment. Finally, it means taking special care of the human-machine interfaces (HMI), i.e., the devices, interfaces, and systems that enable humans and machines to cooperate on the shop floor as coworkers, taking advantage of each other's capabilities.

## 1.2 A Brief History of the Industrial Robot

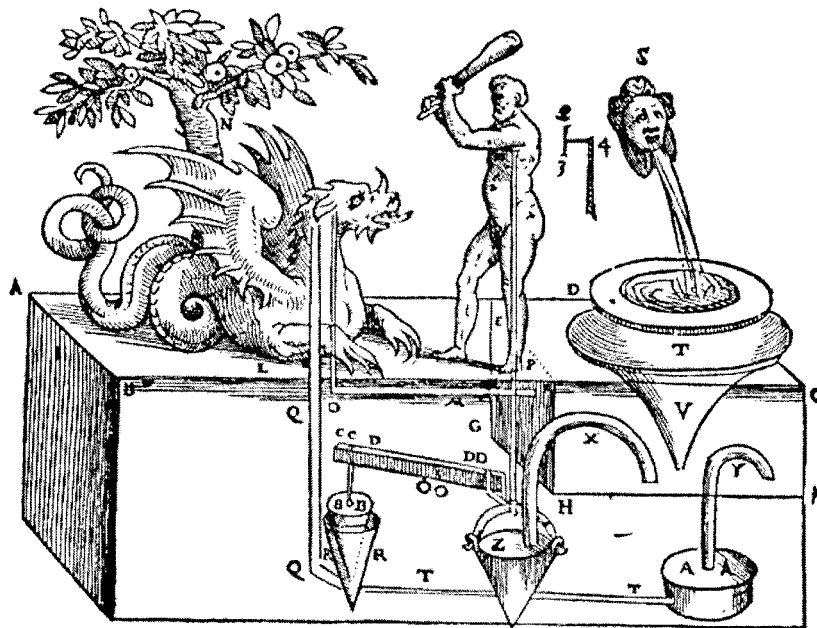
The word “robot” comes from the Czech “robota” which means tireless work. It was first used in 1921 by the novelist *Karel Capek* in his novel “*Rossum's Universal Robots*”. Capek's robots (Figure 1.1) are tireless working machines that looked like humans and had advanced capabilities even when compared with actual robots. The fantasy associated with robotics offered by science fiction movies, and printed and animated cartoons is so far from reality that actual industrial robots seem primitive compared with the likes of *C3PO* and *R2-D2* (from the movie *Star Wars*), *Cyberdyne T1000* (from the movie *Terminator II*) *Bishop* (from the movie *Alien II*) and *Sonny* (from the movie *I Robot*), for example.



**Figure 1.1** A robot from *Karel Capek's* novel “*Rossum's Universal Robots*”

But robotics was a special concern of the most brilliant minds of our common history, since many of them took time to imagine, design, and build machines that could mimic some human capabilities. It is one of the biggest dreams of man, to build obedient and tireless machines, capable of doing man's boring and repetitive work; an idea very well explained by *Nicola Tesla* in his diary [4]:

*"... I conceived the idea of constructing an automaton which would mechanically represent me, and which would respond, as I do myself, but, of course, in a much more primitive manner, to external influences. Such an automaton evidently had to have motive power, organs for locomotion, directive organs, and one or more sensitive organs so adapted as to be excited by external stimuli ..."*



**Figure 1.2** Water clocks designed by *Ctecius* (270 B.C.)

Today's challenge is to consider robots as human coworkers and companions, extending human capabilities to achieve more efficient manufacturing and to increase the quality of our lives. This book focuses on industrial robotic coworkers. The fields of robotics that consider the companion aspects, namely service robotics and humanoid robotics, are not covered in this book. Nevertheless, the social perspective of using robots not only as coworkers, but also as personal assistants, is very promising. In fact, due to several social and economical factors, we are required to work until very late in life: It is common in Europe to only allow

retirement when a person is near seventy years old. Since our physical and mental capabilities decrease with time, the possibility of having mechanical assistants that could help us in our normal routine has some valuable interest.

Robotics can be traced back to 350 B.C., in the ancient *Greece*, to the fabulous philosopher and mathematician *Archytas of Tarentum* (428-347 B.C.) and a demonstration he made in front of the *metropolis* senators. A strange machine that he called "*the pigeon*" was capable of flying more the 200m, using some type of jet propulsion based on steam and compressed air: a great achievement for the time (the invention of the screw and also the pulley are attributed to *Archytas*).

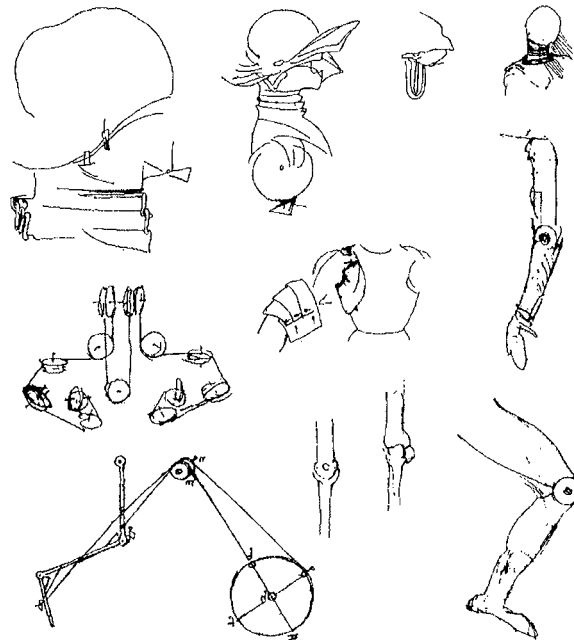


**Figure 1.3** A Greek design adapted by *al-Jazari* for a garden hand-washer

In 270 B.C., also in ancient *Greece*, the civil engineer *Ctecibius* was capable of building water clocks with moving parts (Figure 1.2). His work had followers like *Phylo of Byzantium* author of the book "*Mechanical Collection*" (200 B.C.), and

*Hero of Alexandria* (85 B.C.), and *Marcus Vitruvius* (25 B.C.). In the twelfth century, the Arabian *Badias-zaman al-Jazari* (1150-1220) recollected some of the Greek developments in the book “*The Science of the Ingenious Devices*” [5] (Figure 1.3), and that is how they reached our time. In those early times the problem was about mechanics, about how to generate and transmit motion. So it was mainly about mechanisms, ingenious mechanical devices [5,6].

Then in the fifteenth century, *Leonardo da Vinci* showed indirectly that the problems were the lack of precision and the lack of a permanent power source. He designed mechanisms to generate and transmit motion, and even some ways to store small amounts of mechanical energy [7]. But he didn’t have the means to build those mechanisms with enough precision and there was no permanent power source available (pneumatic, hydraulic, or electric). Maybe that was why he didn’t finish his robot project [5,6], a fifteenth century knight robot (Figure 1.4) intended to be placed in the “*Salle delle Asse*” of the *Sforza* family castle in Milan, Italy. It wasn’t good enough. Or it was so revolutionary an idea for the time that he thought that maybe it was better to make it disappear [5,6].



**Figure 1.4** Leonardo’s studies for a humanoid robot

And then there was the contribution of *Nicola Tesla* at the turn of the nineteenth century. He thought of using *Henrich Hertz’s* discovery of radio waves (following the work of *James Clerk Maxwell* about electromagnetic phenomena) to command

an automata. He built one (Figure 1.5) to demonstrate his ideas and presented it in New York's *Madison Square Garden* in 1898 [4,6]. The problem then was that machine intelligence was missing. Robots should be able to do pre-programmed operations, and show some degree of autonomy in order to perform the desired tasks. When that became available, robots developed rapidly, and the first industrial one appeared in the early 1970s and spawned a multi-million dollar business.

After that, robotic evolution was not as fantastic as it could have been, since there was a lot to do and the available machines were sufficiently powerful to handle the requested jobs. Manufacturers were more or less happy with their robots, and consequently industrial robots remained position-controlled, somehow difficult to program by regular operators, and really not especially exciting machines. Features currently common in research laboratories hadn't reached industry yet because of a lack of interest from robot manufacturers. Nevertheless, there was a considerable evolution that can be summarized as follows.

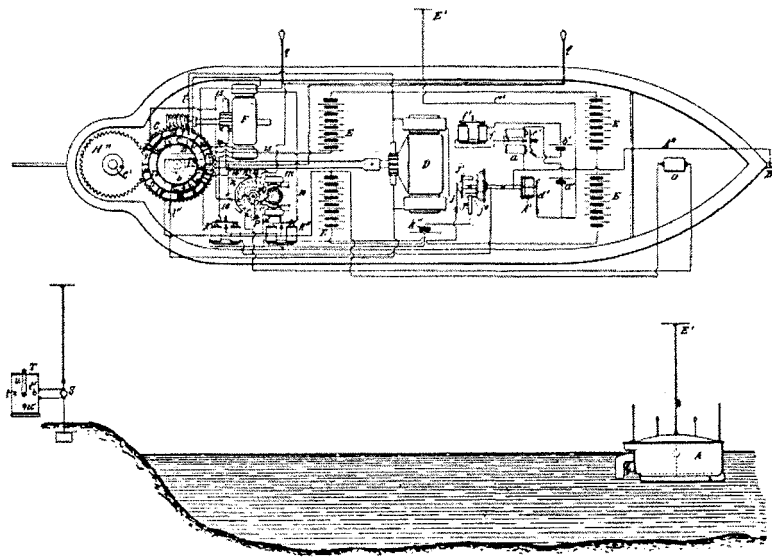


Figure 1.5 Nicola Tesla's remote-controlled miniature submarine

In 1974, the first electrical drive trains were available to use as actuators for robot joints. In the same year, the first microprocessor-controlled robots were also available commercially.

Around 1982, things like Cartesian interpolation for path planning were available in robot controllers, and many of them were also capable of communicating with other computer systems using serial and parallel interfaces. In the same year, some



manufacturers introduced joystick control for easier programming, and the *teach pendant* menu interface.

In 1984, vision guidance was introduced as a general feature for tracking, parts identification, and so on.

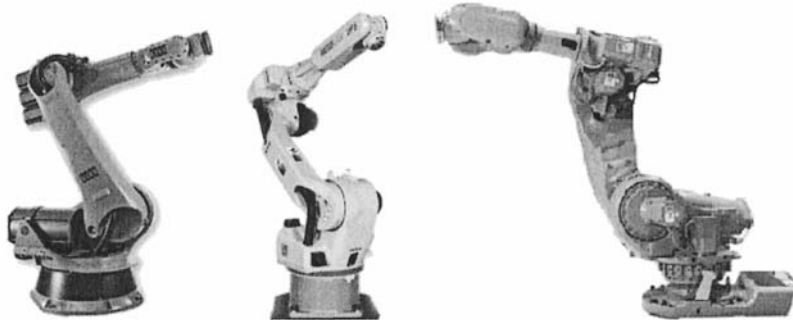
In 1986, the first digital control loops were implemented enabling better actuator control and enabling the use of AC drives.

Networking is a feature of the 1990s, with several manufacturers implementing networking capabilities and protocols.

In 1991, there was the implementation of digital torque control loops, which enabled, for example, the utilization of full dynamical models; a feature only available in the first robots around 1994.

During the period 1992-1994 several manufacturers introduced features like Windows-based graphical interfaces, virtual robot environments for off-line programming, and *fieldbuses*.

Robot cooperation is a feature introduced from 1995 to 1996.



**Figure 1.6** Actual robot manipulators

Around 1998, robot manufacturers started introducing collision detection to avoid damaging robots, and load identification to optimize robot performance. Since then other features include fast pick and place, weight reduction, optimized programming languages, object-oriented programming, remote interfaces using RPC sockets and TCP/IP sockets, *etc.*. Figure 1.6 shows some of the robot manipulators available currently on the market.

So how do we define robotics then? Is it a science? Is it a technique or collection of techniques? If the reader opens a robotics book something like this appears:

*“A robot is a re-programmable multi-functional manipulator designed to move materials, parts, tools, or specialized devices, through variable programmed motions for the performance of a variety of tasks”, from the book Robotics – Control, Sensing, Vision and Intelligence, Fu, Gonzalez, Lee, MacGraw Hill, 1987.*

Although correct, despite being restricted to robot manipulators, this definition doesn't give the correct idea. The common sense image of a robot is usually associated with strong and superb machines, tireless (like *Karel Capek's* machines), obedient (“*yes, noberto san ...*”), but nevertheless, fascinating machines that make us dream. And that fascination is not in that definition.

As with everything, we should look to the past and pick what was fundamental for the history of robotics in terms of ideas and dreams. From the *Greeks* and *Arabs* we should pick their idea of “*ingenious devices*”. In fact, robotics is very much about mechanics, motion, mechanisms to transmit motion, and having the art and the skill to design and build those mechanisms. Yes, “ingenious devices” is really a good start.

Then we should turn to *Leonardo* (sixteenth century) and look to his quest for “... *precision ...*” and “...*permanent power source ...*”. He understood that robots need parts built with very high precision and a permanent power source. That was not available at his time, *i.e.*, machine tools and a permanent power source (electric, hydraulic, or pneumatic).

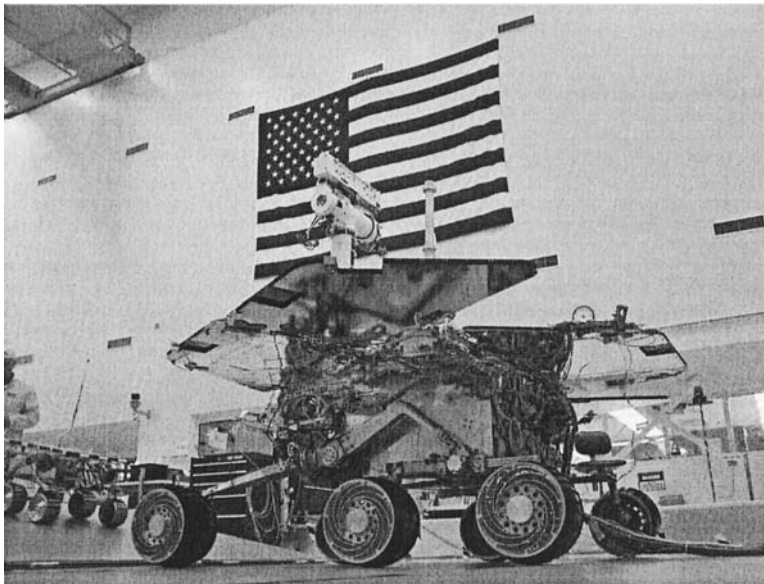
Finally, we should read *Nicola Tesla* and observe his outstanding and visionary work. He understood that robots are a consequence of dreams and neat ideas. Robots need to be controlled and programmed, distinguish situations, *etc.*, have ways of “*understanding*”, and that means using computers, electronics, software, and sensors, in a way to enable machines to be programmed and to sense their environment. Those are the elements that enable us scientists, engineers, and robot users to try different things and new ideas, being a source of fascination. In his own words [4]:

*“... But this element I could easily embody in it by conveying to it my own intelligence, my own understanding. So this invention was evolved, and so a new art came into existence, for which the name “teleautomatics” has been suggested, which means the art of controlling movements and operations of distant automatons.*

Therefore, we can define robotics as a science of generic, ingenious, precise, mechatronic devices, powered by a permanent power source; a science that is open to new ideas and that stimulates the imagination. A stimulus so strong that it attracted many of the best minds of our common history, *i.e.*, authors of the work that constitutes the legacy that we humans leave for the future.

### 1.3 Using Robotics to Learn

Putting robots in space, and in other planets, is a very exciting field of modern robotics. This and other fantastic achievements justify the enormous interest about robots and robotic applications. Only a few engineering fields are as multidisciplinary as robotics, i.e., areas that require knowledge of as many different scientific and technical disciplines. Robotics integrates an extensive knowledge of physics, mechanics, electronics, computer science, data communications, and many other scientific and technical areas important for the design, construction, and operation of machines that execute human-like functions.



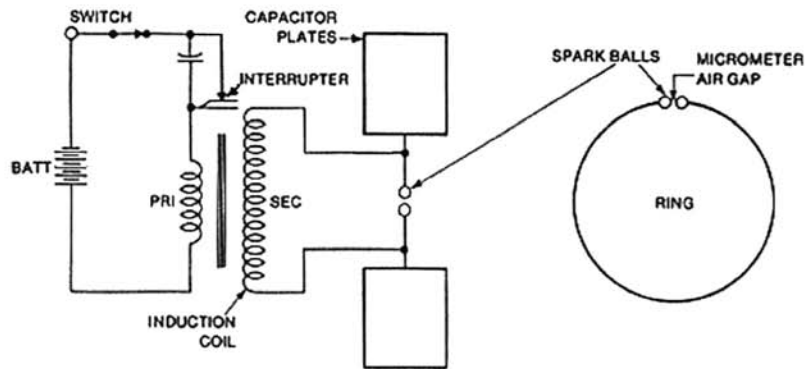
**Figure 1.7** Robot MER-A (*Spirit*) sent to Mars in June of 2003 (from NASA) [8]

In this section a small mobile robot, named *Nicola*, is presented. The robot is constructed, using commonly available industrial equipment, to be commanded from a typical personal computer running standard operating systems and software development tools. The final goal is to demonstrate what is involved in the construction of this type of robot, showing that it is possible to play with science and technology and in the process learn and spend a fantastic time. The robot *Nicola* will be presented step-by-step with enough detail for showing what is involved.

NASA initiated in June 2003 a new mission to further explore Mars, the great red planet of our solar system [8]. The allure of Mars is based on its proximity to Earth, but also on the assumption that it was once like Earth, with water available

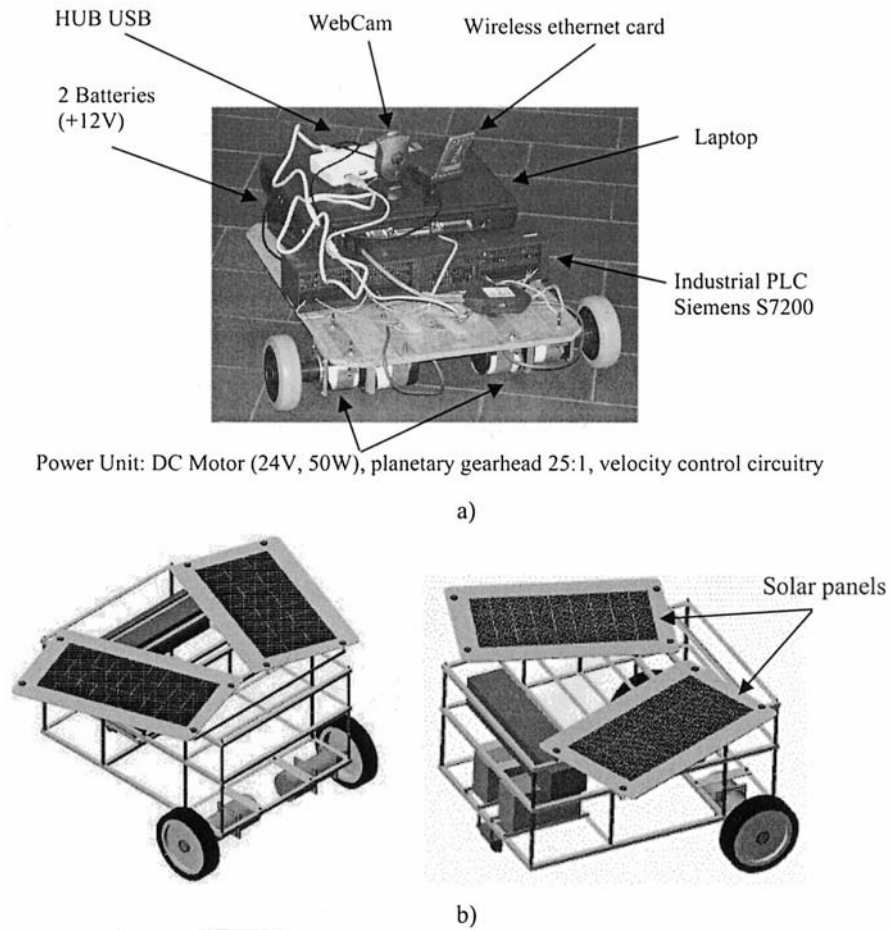
on the surface and life, before changing gradually to a hot and dusty planet. In this mission, NASA used again semi-autonomous mobile robots to explore the planet. These Mars exploration rovers (MER – Figure 1.7), named *Spirit* and *Opportunity*, are capable of navigating the surface of the planet, analyzing rocks and land, and sending back pictures, videos, and results from experiments carried out on the planet's surface. The spaceship that carried *Spirit* was launched on June 10, 2003, and arrived on Mars on January 4, 2004. In turn, the spaceship that carried *Opportunity* left on July 7, 2003, and arrived on Mars on January 25, 2004.

The utilization of these robots was also a dream of the great Croatian inventor *Nicola Tesla* (1845-1943), a man that gave a pioneering and visionary contribution for the evolution of robotics. He worked with the legendary *Thomas Edison* and was a tireless, dedicated, and bright inventor. *Tesla* was the archetype of the inventor: solitary, absent minded, abstracted of the normal things of life, with an exclusive dedication to his work and visionary. At the end of the nineteenth century he dreamt (doesn't everything begins like this?!) of *automatons* capable of performing tasks only possible to intelligent living creatures. For that, the *automaton* needed an element equivalent to the human brain. Since that seemed complicated, he thought about using his own brain for commanding the automaton [4].



**Figure 1.8** Heinrich Hertz's first transmitter, 1886 schematic

That capacity of commanding distant automatons was achieved using *Henrich Hertz* waves (published in 1887 in a treatise named "*Optice Elettrica*"). *Tesla* had access to *Hertz's* publications and saw in his radio transmitters and receivers (Figure 1.8) a way to implement his own ideas. To demonstrate the principle, *Tesla* built a model of a submarine (Figure 1.5) controlled remotely using coded hertz impulses (controlled by radio, therefore). He could command the boat to turn to the right or to the left, submerge and emerge, *etc.* Despite the enormous interest of the new invention, which he demonstrated in the *Madison Square Garden* of *New York City* (1898), before an overwhelmed audience, he failed to obtain support to continue his efforts on the subject.



**Figure 1.9** The Robot Nicola: a) Nicola I; b) Nicola II

But it was a fabulous advancement for the time (nineteenth century). How would it be building a system with those characteristics today? Using common industrial equipment, wireless communications, actual operating systems, and well known programming tools?

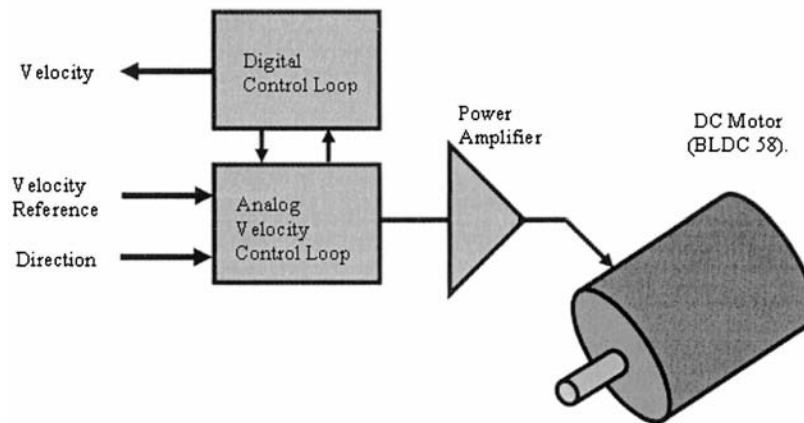
That is the goal of our robot *Nicola*, i.e., to show that *Tesla's* dream is still actual, and that despite the sophistication of those robotic space explorers (Figure 1.8), the technology involved and the concepts are simple, accessible, and fun to learn how it all basically works.

### 1.3.1 Constitution of the Robot Nicola

The robot *Nicola* is very simple. Basically it is a three-wheel robot with two power wheels in front and a free wheel in the back (Figure 1.9). The wheels used are of the same type that can be found in office chairs and other office equipment. Each of the two power wheels are equipped with a power unit composed of:

1. One 24 V DC motor (máx. power 50 W, máx. velocity 3650 rpm, máx. torque 0.17 Nm), model MDLC-58 from *Maclennan Ltd.* [9]
2. One 25:1 gear unit, model IP57-M2 from *Maclennan Ltd.* [9]

The selected DC motor is equipped with a velocity control loop (Figure 1.10), which makes it very simple to linearly control velocity just by feeding the unit with a 0-5 V analog signal. The control circuit is a very simple electronic circuit composed of a velocity control loop and a power amplifier. The velocity control loop makes the motor velocity proportional to the commanding analog signal (0-5 V in magnitude), and the rotating velocity is defined by a digital input (0 – positive direction, 1 - negative direction).



**Figure 1.10** Diagram of the velocity control circuitry [9]

Using this power unit, attached to each wheel, there is no need for a mechanical steering mechanism since the electric differential effect can be used to steer the robot, i.e., varying the speed of each independently wheel it is possible to turn to the right and to the left with high-precision and several curvature radius. For example, if the speed of the left wheel ( $v_l$ ) is equal to the speed of the right wheel ( $v_r$ ), the robot moves forward in a straight line ( $v_l = v_r > 0$ ). If we change the sense of rotation of the wheels ( $v_l = v_r < 0$ ), the robot moves backwards also in a straight line. Making  $v_l > v_r$ , the robot turns to the right, and with  $v_l < v_r$  it turns to the left. Adjusting the value of  $v_l$  and  $v_r$  several curvature radius may be obtained. Finally, making  $v_l = -v_r$  the robot turns about itself.

Furthermore, with the objective of using industrial components, the robot uses a medium class PLC (*Programmable Logic Controller*) to interface with sensors and actuators. The selected PLC is a *Siemens S7-200* (DC model with the 215 CPU), equipped with a 12-bit resolution analog module (module EM235, with three inputs and one output) [10].

To command the robot, a laptop is installed on the robot, connected to the PLC using a serial link (RS-232C channel). The software running on the laptop was built to work as a *TCP/IP socket server*, enabling commands by any authorized remote client. The operating system running on the PC is the *Microsoft Windows XP*, which makes it easy to add advanced services, attach devices (like network devices, Webcams, etc.), and explore them from available software developing tools (*Visual Basic, C++, C#, etc.*).

### 1.3.2 Nicola Software

The software designed to control and operate Nicola is divided into three levels, identified with the programmable hardware components that constitute the robot:

1. The PLC that implements the low-level interface with sensors and actuators
2. The on-board PC used to manage the robot and interface with remote users
3. The commanding PC, i.e., the computer used to command the robot and monitor its operation

In the following sections the software will be presented in detail. The interested reader can download the source code from [11].

#### 1.3.2.1 PLC Software

The mission of the PLC is to interface with analog and digital sensors that could be used with the robot, and to control the two DC motors that move the robot and any other actuator that could be added to the system. Consequently, a PLC is a good choice since this is basically what is required from them in industry, i.e., to work as local and low-level interfaces with sensors and actuators implementing sequential commanding routines. In addition, PLCs are very easy to use and to program, which also justifies the solution. The only difficulty with the PLC is the need to have it working as a server, executing the commands sent by the on-board PC that manages the robot (Figure 1.11). This means that the PLC should implement the services required to operate the robot, namely:

1. The possibility to change any analog or digital output
2. The possibility to access any analog or digital input
3. The possibility to command macros, or batches of functions

4. The possibility to receive events with detailed information about the status of the robot.

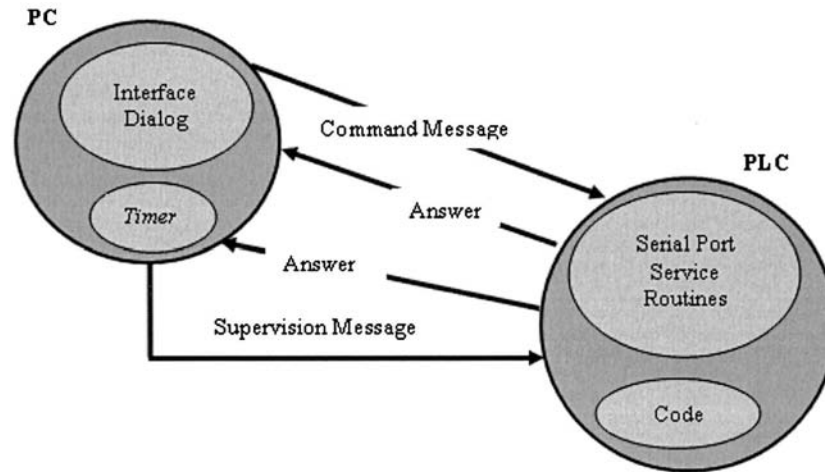


Figure 1.11 Messages between the on-board PC and the PLC

Table 1.1 List of PLC command codes

Command	Parameter 1	Parameter 2	Description
159	120 + output	Valor	Changes the specified analog output.
160	120 + input	-	Reads the actual value of the specified analog input.
200	120 + output 130 + output		Activates the specified digital output of the first output block (120) or of the second output block (130).
201	120 + output 130 + output		Deactivates the specified digital output of the first output block (120) or of the second output block (130).
253	-	-	Supervision message.

This idea is very simple and not different from what is done in more advanced machines, like the industrial robots. From the remote client, properly authorized, the user accesses memory zones to change some predefined variables (bytes, word or double-word variables). If the value of those variables is used in the programmed instructions, it is possible to execute only the intended sequences just by comprehensively changing the values of those variables. The PLC answers to remote commands sent with a pre-defined format and with a maximum length of



100 bytes. The first byte of the commanding message specifies the command, and the following bytes are parameters (see Table 1.1).

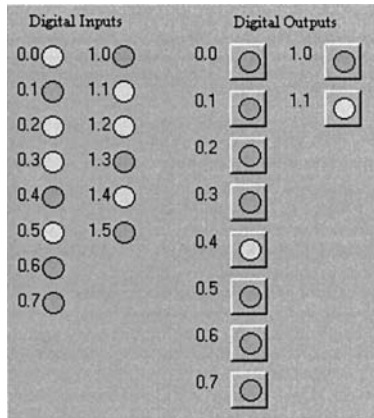
The synchronous answer of any command is a copy of the message received, which enables the commanding PC to check if the command was well received using for example an ACK-NACK (acknowledge – not acknowledge) protocol. Besides that, there is a special command (code = 253) used for monitoring the PLC state. When the PLC receives this command it should answer by sending the state of all its IO inputs and outputs. This message should be sent frequently to track the robot state. In the robot Nicola this message is associated to a 500 ms timer, which means that the robot state is updated at a frequency of 2 Hz.

Any asynchronous answer contains the execution results of one command. For easy identification from the serial port interrupt routine, the first byte of the answer identifies the code of the executed command. The user commands should be associated with user actions like pressing software buttons or keyboard buttons, etc. When the PLC receives a command, it transfers the received data into a pre-defined memory zone starting with register VB90. Consequently, if the command contains  $n$  bytes, with  $n \leq 100$ , the following happens:

Byte **VB90** – contains the number of byte received  
 Byte **VB91** – contains the character (code) that identifies the command  
 Byte **VB92** – contains parameter 1  
 ...  
 Byte **VB90 + n - 1** – contains parameter  $n$

The PLC routine designed to handle the serial port initializes the port in the first SCAN cycle, entering after that into the listen state. When a message is received, the data is transferred to the already mentioned memory zone and a copy is sent back to the calling PC.

For example, the PLC used with *Nicola* (*Siemens S7-200*) has 10 digital outputs in the basic module, labeled from Q0.0 to Q0.7 (output block 0), and from Q1.0 to Q1.1 (output block 1). To access those digital outputs, the command must specify the type of access (write or a read access), the signal number, and the signal value in the case of a write access (check Table 1.1).



```
'Activates/deactivates digital outputs from block 0
Private Sub q0_Click(Index As Integer)
If fq0(Index) = False Then
com.Output = Chr(200)+Chr(120+Index)+Chr(10)
fq0(Index) = True
Else
com.Output = Chr(201)+Chr(120+Index)+Chr(10)
fq0(Index) = False
End If
End Sub
```

```
' Activates/deactivates digital outputs from block 1
Private Sub q1_Click(Index As Integer)
If fq1(Index) = False Then
com.Output = Chr(200)+Chr(130+Index)+Chr(10)
fq1(Index) = True
Else
com.Output = Chr(201)+Chr(130+Index)+Chr(10)
fq1(Index) = False
End If
End Sub
```

```
' Shows IO state
Private Sub rio_Click()
Dim i As Integer
For i = 0 To 7
If (bq00 And 2 ^ i) = 2 ^ i Then
q0(i).Picture = lon
fq0(i) = True
Else
q0(i).Picture = loff
fq0(i) = False
End If
Next i
For i = 0 To 1
If (bq10 And 2 ^ i) = 2 ^ i Then
q1(i).Picture = lon
fq1(i) = True
Else
q1(i).Picture = loff
fq1(i) = False
End If
Next i
For i = 0 To 7
If (bi00 And 2 ^ i) = 2 ^ i Then
i0(i).Picture = lon
Else
i0(i).Picture = loff
End If
Next i
For i = 0 To 5
If (bi10 And 2 ^ i) = 2 ^ i Then
i1(i).Picture = lon
Else
i1(i).Picture = loff
End If
Next i
End Sub
```

The serial port interrupt service routine stores the messages received from the PLC in the variables:  
 bq00 – digital output signals of block 0  
 bq10 – digital output signals of block 0  
 bi00 – digital input signals of block 0  
 bi10 – digital input signals of block 0

The routine **rio\_click** represents the received information at the user panel using colors: yellow (activated), gray (deactivated).

**Digital Outputs**

**Digital Inputs**

When a message arrives, the service routine calls "rio\_click" to present the information:

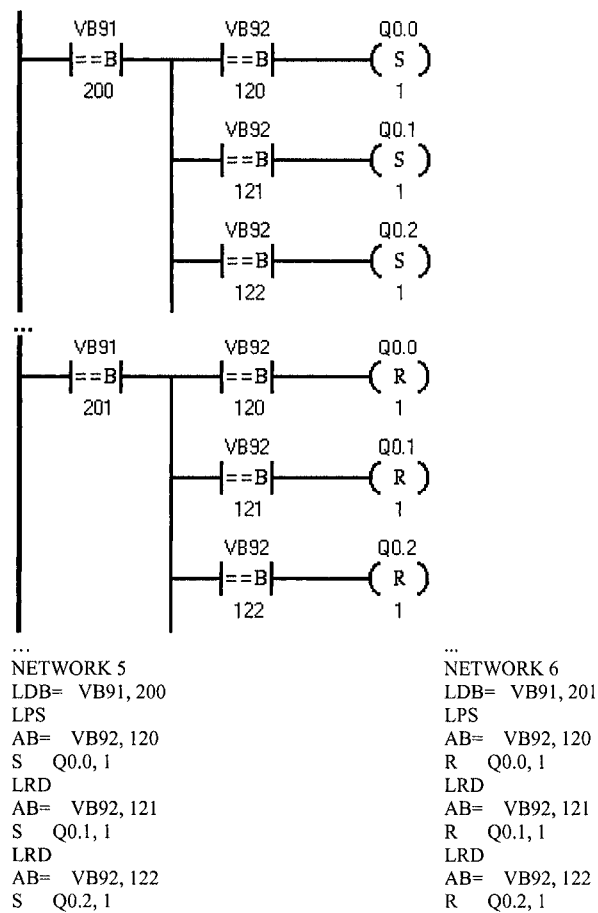
```
Private Sub com_OnComm()
get_com_message
rio_Click
End Sub
```

Figure 1.12 PC software designed to access IO signals

Consequently, to change the state of Q1.1 to 1 the following command should be sent (Table 1.1):

200 131 255 255 10

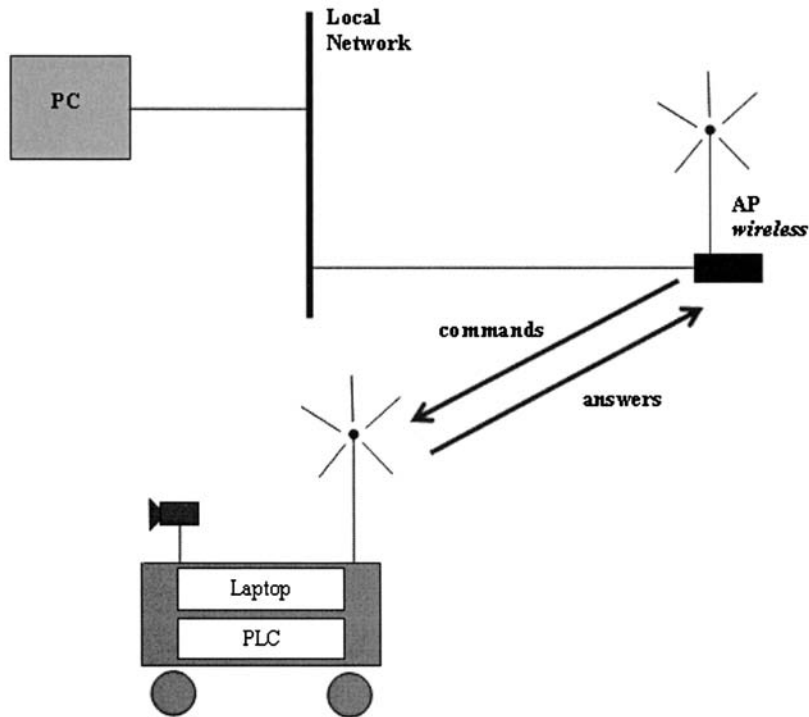
where “200” specifies a digital write access, “131” specifies the output Q1.1, “255” is a null command/parameter and “10” is the end-of-message character. The software for this example, including the both the PLC and the PC side, is presented in Figures 1.12 and 1.13 (the PC part was coded using *Microsoft Visual Basic .NET2003*, and the PLC part was coded using the *Siemens PLC S7-200* programming tool called *Microwin 3.2*).



**Figure 1.13** PLC code to activate/deactivate digital outputs. Due to space limitations, only the code for the first three outputs of the digital block 0 is presented.

### 1.3.2.2 Software for the On-board PC

The software for the on-board PC was designed to control the robot, and to interface with the remote user connected to the robot's on-board computer using a wireless network connection (Figure 1.14).



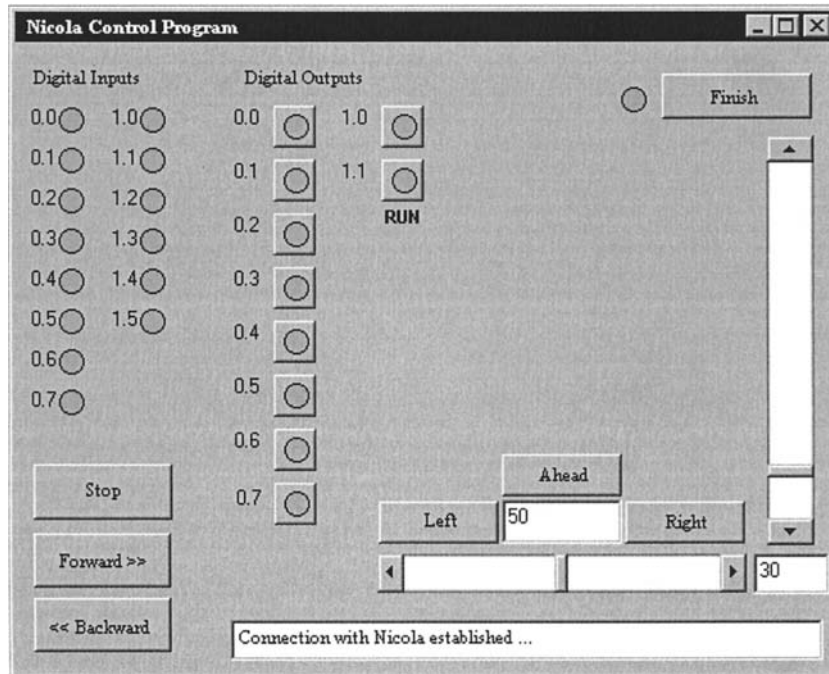
**Figure 1.14** Overview of the system used to operate the robot *Nicola*

The on-board user interface software is a TCP/IP socket server that listens on a specific port, accepts and validates user connections, and processes the commands sent by the remote client. Those commands have the following basic syntax:

$$rx \text{ command } parameter_1 \text{ parameter}_2 \dots parameter_n$$

where,  $rx$  specifies the robot (for example,  $r1$ ),  $command$  is a string that specifies the command to be executed (Table 1.2), and  $parameter_i$  is the set of parameters associated with the particular command.

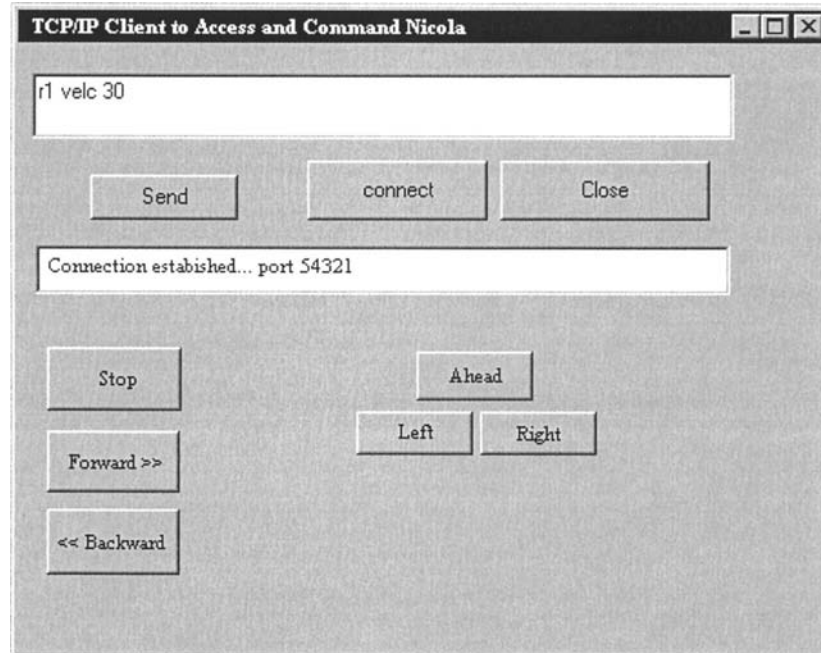
Figure 1.15 shows the shell of the TCP/IP server developed for the on-board computer. The panel functions enable the user to quickly access the local robot functions, and the TCP/IP server included in the application implements the interface for remote users.



**Figure 1.15** TCP/IP server used to operate the robot *Nicola*: listens to connections on port 54321, validates connections, and process commands

**Table 1.2** List of commands available from the on-board TCP/IP server

Command	Parameter	Description
INFO	-	Supervision message.
VELC	Valor (0-255)	Commands the robot velocity: 0 (min.) to 255 (máx.).
STOP	-	Stop command.
AVAN	-	Commands the motors to move in the positive (forward) direction.
RECU	-	Commands the motors to move in the negative (backward) direction.
FRNT	-	Commands the motors to move straight ahead/backward, i.e., clears any steering direction.
DIRT	-	Turns right at 50%, i.e., the actual velocity of the left motor is kept and the velocity of the right motor is reduced by 50%.
ESQD	-	Turns left at 50%, i.e., the actual velocity of the right motor is kept and the velocity of the left motor is reduced by 50%.
DIRD	Valor (0-100)	Turns right by the specified amount.
DIRE	Valor (0-100)	Turns left by the specified amount.



**Figure 1.16** TCP/IP client used to operate the robot *Nicola* from any remote PC

Figure 1.16 shows a simple TCP/IP client example that can be used to command remotely the robot *Nicola*. This example offers to the user the possibility to execute simple commands like start and stop, move forward or backward, turn left and right (with a specified steering angle), or move straight ahead and regulate the robot's speed.

#### 1.3.2.3 Feedback from the On-board Webcam

The robot *Nicola* is equipped with a *webcam* to register images of its operation and to help the remote user command it in situations where the robot is not in sight. It's very easy to get images and video streams from a *webcam* and there are a lot of software packages and tools to do that. Here the *Microsoft Visual SDK 1.2* is used because it is an open source *SDK*, and because it integrates well with the development environment used to write the software: the *Microsoft Visual Studio .NET2003*.

Since the video feed is installed on the robot, there's also the problem of sending the obtained images from the on-board computer to the remote computer, using the data rate more adjusted to the capacity of the wireless link.

Again we opted to build a TCP/IP server to work as the image service. Basically this server is able to capture images and save those images in the hard disk of the on-board computer. These files can then be shared with the remote computer using an FTP connection or simply by sharing the directory. Using a mechanism like a semaphore it is possible to avoid having the two computers accessing the file at the same time, i.e., by the on-board computer that generates the file and by the remote computer that reads the file and presents it to the user. The image refresh rate depends on the communication speed and availability, but also on the size of the image. Nevertheless, it is possible to have rates up to 10 frames per second. Live streams, of about 30 to 40 frames per second, are only possible for the on-board computer since it was decided to avoid sending streams over the TCP/IP connection. This was a decision for simplicity, but also a practical decision: Live streams are really not necessary for this application.

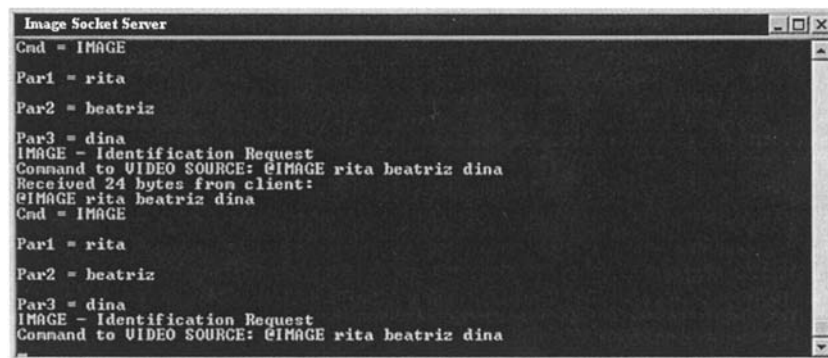
The TCP/IP image server implements the following basic services:

1. Specify the vision provider, namely the driver that will be used to capture the image. In this example the Webcam uses a *Video for Windows* (VFW) driver
2. Start/stop the acquisition service
3. Obtain the actual image and save it to the on-board hard disk

The image server (Figure 1.17) listens at the port 54322 for messages starting with the character “@” and ending with the character “#”. For example, the command message to obtain the actual image is:

`@IMAGE rita beatriz dina#`

where, *IMAGE* is the command, *rita* is the username, *beatriz* is the password and *dina* is the name of the file where the image should be saved. The TCP/IP client will present the image only if the answer from the server matches exactly the command sent. Any other situation is considered an error.



```

Image Socket Server
Cmd = IMAGE
Par1 = rita
Par2 = beatriz
Par3 = dina
IMAGE - Identification Request
Command to VIDEO SOURCE: @IMAGE rita beatriz dina
Received 24 bytes from client:
@IMAGE rita beatriz dina
Cmd = IMAGE
Par1 = rita
Par2 = beatriz
Par3 = dina
IMAGE - Identification Request
Command to VIDEO SOURCE: @IMAGE rita beatriz dina

```

Figure 1.17 Output window of the on-board TCP/IP image server

Basically, the TCP/IP image client (Figure 1.18) has one button for each available service and shows the obtained image and the refresh rate. The method used to avoid simultaneous access to the image file between the two computers was a 50ms timer. The timer interrupt service routine performs alternatively the call to acquire the image and the call to get the file from the on-board computer, avoiding the simultaneous access to the image file. This means that a new image is obtained every 100 ms. Consequently, the only limitation to the refresh rate is the throughput of the communication link.

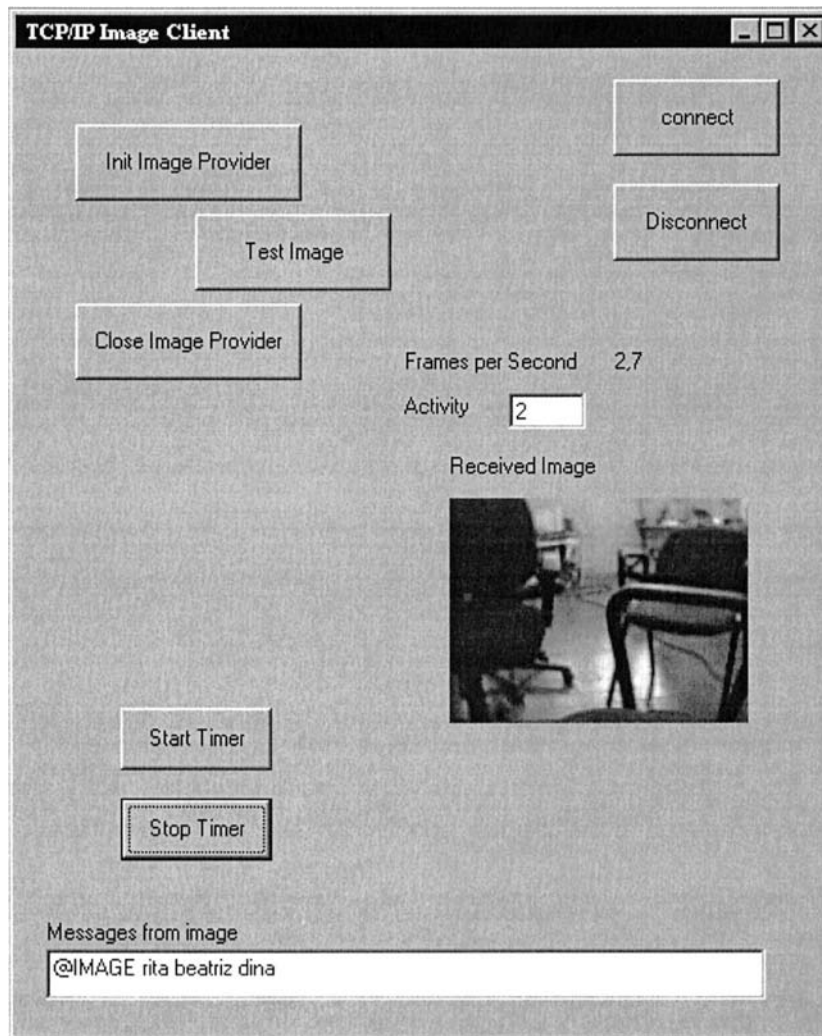


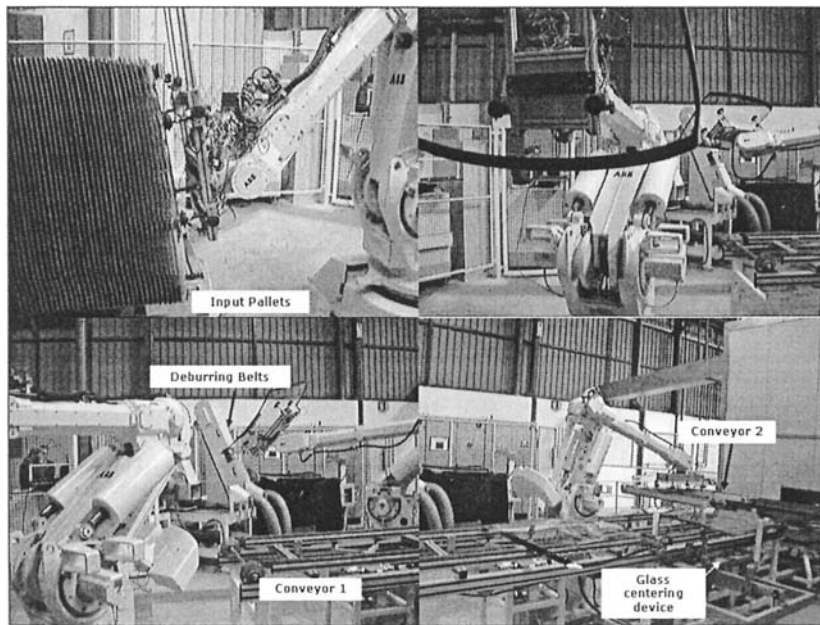
Figure 1.18 TCP/IP image client used on the remote PC



This simple example, which explores industrial equipment to build a useful mobile robot, shows clearly that robotics is a very interesting subject to learn and play with science and technology. It also shows that the concepts are accessible and can be explored by any student or engineer. The main objective of this section was to motivate readers to explore this book, because it'll show how things work and can be implemented in a practical way, with enough detail for those readers who want to explore further.

#### 1.4 Using Robotics to Work

The industrial robotic system presented in this section was designed to execute the task of removing the excess of PVC material from automobile glasses, which accumulates during the glass manufacturing cycle. In fact, most of the automobile glasses, namely front, rear, and roof glasses, are composed of two sheets of glass joined by a layer of PVC. For proper assembly, and to ensure proper joining of the PVC to the glass while maintaining transparency, the glass goes through a heating process, followed by a considerable period inside a pressure chamber. This process generates a very stiff excess of PVC on the borders of the glass that must be carefully removed because it alters the dimensions of the glass, causing difficulties in assembling it in the car body, not to mention the aesthetic implications.



**Figure 1.19** Robotic glass deburring system

Traditionally, this excess of PVC is removed by hand using small cutting devices. Nevertheless, for highly efficient plants, this is not desirable since it slows down production, and requires very high concentration from operators so they don't touch and damage the glass with the cutting device. Consequently, the process is very risky for the quality of the final product. Furthermore, with recent car designs, some glasses are glued directly in the chassis without any exterior rubber, mainly with roof, front, and rear glasses. Here the requirements for perfect PVC removal are even higher, which demands an automatic procedure to execute it.

The system (Figure 1.19) designed to handle the operation described above is composed of [12]:

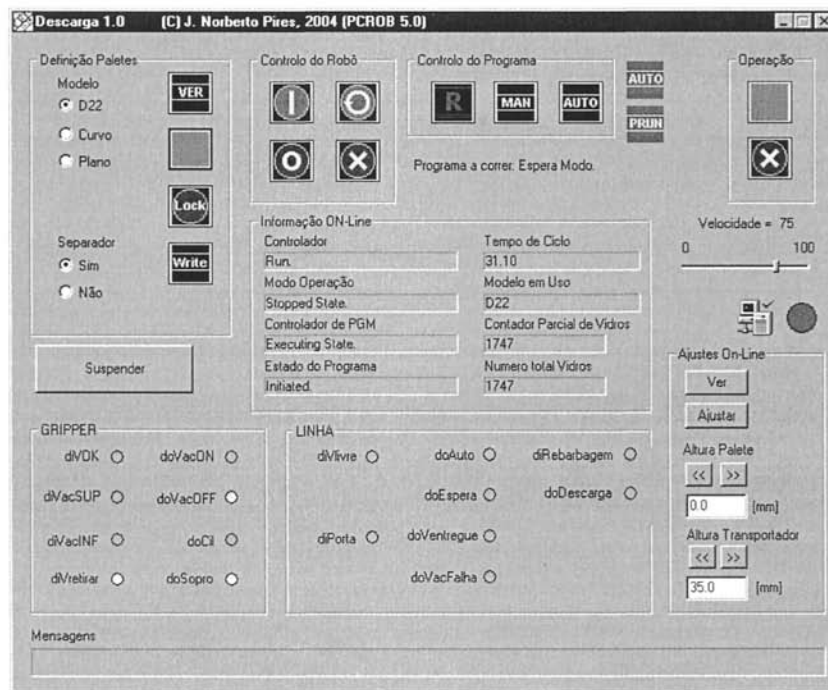
1. Two industrial robots ABB IRB6400 equipped with the S4C+ controllers
2. Specially designed electric-pneumatic grippers to hold firmly the glasses
3. Two automatic deburring belts controlled by the robot's controller IO system
4. One industrial PLC (*Siemens S7-300*) that manages the cell logic and the interface to the adjacent industrial systems, providing to the robot controllers the necessary state information and the interface to the factory facilities
5. One personal computer to command, control and monitor the cell operation

The system works as follows: The first robot verifies if conveyor 1 (Figure 1.19) is empty and loads it with a glass picked from the pallet in use. The system uses a rotating circular platform to hold three pallets of glasses, enabling operators to remove empty pallets and feed new ones without stopping production. After releasing the glass, the robot pre-positions to pick another glass, which it does when the conveyor is again empty. If the working glass model requires deburring, then the centering device existing in the conveyor is commanded to center the glass so that the second robot could pick up the glasses in the same position. With the glass firmly grasped, the deburring robot takes it to the deburring belts and extracts the excess PVC by passing all the glass borders on the surface of the deburring belt. When the task is finished, the robot delivers the glass on conveyor 2, and proceeds to pick another glass.

The deburring velocity, pressure, trajectory, *etc.*, is stored in the robot system on a database sorted by the glass model, which makes it easy to handle several models. Programming a new model into the system is also very simple and executed by an authorized operator. There is a collection of routines that take the robot to pre-defined positions, adjusted by the given dimensions of the glass, allowing the operator to adjust and tune positions and trajectories. He can then "play" the complete definition and repeat the teaching procedure until the desired behavior is obtained. This means being able to control the robot's operation with the controller in automatic mode, which is obtained by including some teach-pendant features in the process for operator interface.

Another important feature of this robotic system is the ability to adjust production online, adapting to production variations. This objective is obtained by using a client-server architecture, which uses the cell computer (*client*) to parameterize the software running on the robot controller (*server*). That can be achieved by offering the following services from the robot server to the clients:

1. All planned system functionalities by means of general routines, callable from the remote client using variables that can be accessed remotely
2. Variable access services that can be used remotely to adjust and parameterize the operation of the robotic system



**Figure 1.20** Operator interface for de-palletizing robot

With these features implemented and with a carefully designed operator interface (Figure 1.20 and Figure 1.21) and robot server software, it's possible to achieve a system that requires limited human intervention related with adjustment tasks to cope with production variations. Since a remote interface is used (Figures 1.20 and 1.21), the necessary adjustments are executed *online* without stopping production. Those operations include:

1. Adjusting the deburring angle, *i.e.*, the angle between the border of the glass and the deburring belt. The angle introduced is added to the programmed one, so that zero degrees means keeping the programmed angle unchanged

2. Adjusting the force on the belt during the deburring operation (adjusted by position). The commanded value is entered in millimeters and updates the actual position in the direction perpendicular to the belt and parallel to the surface of the glass
3. Adjusting the deburring speed
4. Maintenance procedures necessary to change the belts after the planned deburring cycles

The de-palletizing robot requires less parameterization because it executes a very simple operation. Other than that, the gripper adapts to the surface of every model of glass, using presence sensors strategically placed near two suction cups (see Figure 1.19), with the objective of having an efficient de-palletizing operation. Nevertheless, the operator is able to change the velocity of the process by stating a slow, fast, or very fast cycle to adjust to production needs, suspend and resume operations, adjust the way the robot approaches the surface of the glass, *etc.*. These adjustments are necessary to obtain the most efficient operation in accordance with the observed production conditions, to solve daily problems, and to cope with production variations.

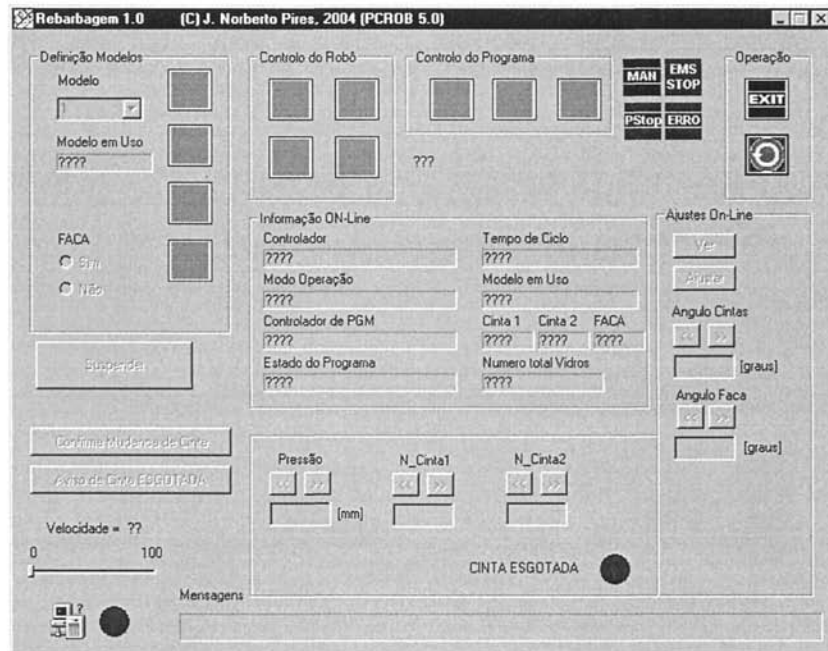


Figure 1.21 Operator interface for deburring robot

Finally, it is important to mention that the robot is equipped with a force/torque sensor mounted on the wrist. The objective is to adjust automatically the model setup introduced by the operator, correcting the points where the measured force

between the belt and the glass exceeds the recommended values, attempting to avoid damage to the glass and to increase the deburring efficiency. This procedure is active during the process of applying a new model, and also during production, if explicitly activated by the operator, constituting an automatic correcting feature.

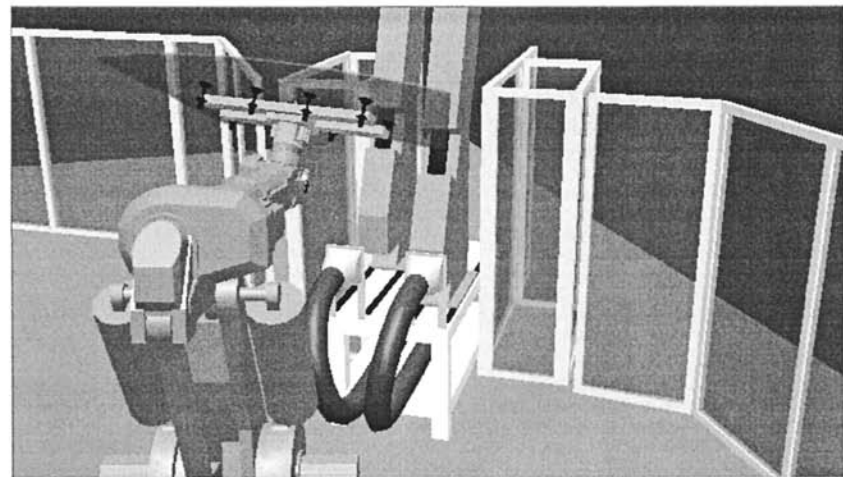
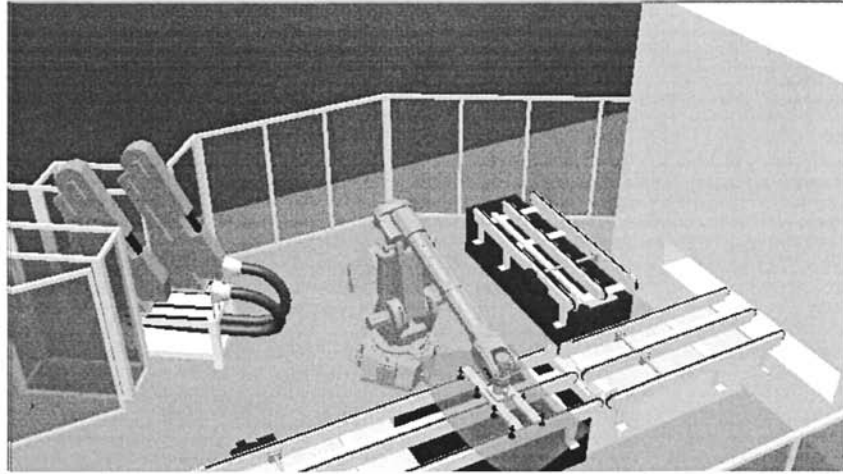
The system has worked for some time and proved to be very simple to operate, showing also quick adaptation from operators [12,18]. The adjusting features added to the system proved to be very helpful, allowing the company to respond in a timely fashion to production changes, avoiding variations in the quality of the final product, and to introduce quickly new models into the production database. Since the models are identified automatically, using barcode readers placed on the pallet platform, the system works continuously without operator intervention. The only thing needed is to feed the system with pallets full of glasses, removing the empty ones. That operation is done periodically with the help of electro-mechanical fork lift trucks.

Most of the features presented in this example will be explored in this book for robotic welding applications, namely the capacity to simulate the procedure, the capacity to adjust online and change parameterization, the capacity to monitor the system, and specify the sequence of operations, and so on.

This example shows clearly the advantages of using robots with actual manufacturing platforms and the importance of carefully designing the manufacturing systems, and integrating intelligent sensors, actuators, and the human factor. This final aspect related with HMI (human-machine interface) is fundamental in any manufacturing system and somehow a measure of its success, since these systems need a very efficient way to operate with humans in a way to expose system features and allow the users to explore the system capabilities to the maximum extent [12-18].

#### **1.4.1 Using an Offline Simulation Environment**

Using offline programming and simulation environments may be useful to develop and especially to optimize industrial manufacturing systems. Frequently the system is not available for online utilization, which calls for the possibility to work with graphical models of the manufacturing cell under study. The industrial deburring system presented in this section (Figures 1.22 and 1.23) was optimized using a graphical offline tool (*RobotStudio 5* from *ABB Robotics*), although the 3D drawings of several components of the cell were designed using *SolidWorks*.



**Figure 1.23** Analyzing the glass deburring process on the graphical environment

The utilization of offline packages has some advantages:

- If carefully designed, the graphical model constitutes a powerful tool to continuously develop the system without stopping production

- It allows the system engineer to simulate and optimize the solutions before testing them on the real cell for final implementation
- It constitutes a powerful tool to analyze new production scenarios, with new products, new production sequences, etc., before testing them or even before proposing them to the production team
- It constitutes a nice environment to demonstrate to customers the viability of certain type of production, cycle time, *etc*
- Since this type of environment runs a virtual robot controller, it allows the user to develop software and try it on the graphical model

The only disadvantage is the correlation between the graphical model and the real system. This means that the system engineer needs to carefully calibrate the system using precise data from the cell. This will allow him to export code directly to the cell and have it working with only minor calibration and routine checking.

## 1.5 Statistics of Robotic Workers

There are at least 800 000 robots working in industry worldwide (Table 1.3), but since statistics are very difficult to obtain in several countries, the real number should be over 1 million units operating all over the world [23]. Considering the statistics from 2003 [23], the lead country pushing its economy using robots is Japan, with around 350 000 robots operating, followed by the European Union, with around 250 000 robots in action, and the United States with around 112 000 robots. In Europe, Germany is the lead country with 112 700 units operating (matching the United States), followed by Italy (50 000 robots), France (26 000 robots) and Spain (20 000 robots).

**Table 1.3** Robot operational stock at the end of the year (2001-2003) with a forecast for the period 2004-2007

Country	Operational Stock at the End of the Year			
	2002	2003	2004	2007
Japan	350 169	348 734	352 200	349 400
USA	103 515	112 390	121 300	145 100
EU	233 769	249 200	266 100	325 900
Germany	105 212	112 693	121 500	151 400
Italy	46 881	50 043	53 100	151 400
France	24 277	26 137	28 400	35 900
Spain	18 352	19 847		
Portugal	1 282	1 367		

Source: IFR – International Federation of Robotics [23]

In 1990, the installation of new industrial robots in the European Union was only 20% of the new installations reported from Japan. The USA had only 7% of new installations when compared with Japan. Nevertheless, this gap was reduced

significantly and currently both EU and USA grow at approximate rates when compared with Japan, being sometimes higher than the Japanese rates. For example, in the period 2001-2002, the European Union installed more robots than Japan, but in 2003 the Japanese recovered the first place. This evolution of the European and North American robot installations reveals itself in the operational stock. The European stock evolved from 23% of that of Japan in 1990 to almost 72% in 2003. The figures for the USA show an evolution from 12% in 1990 to 32% in 2003, respectively.

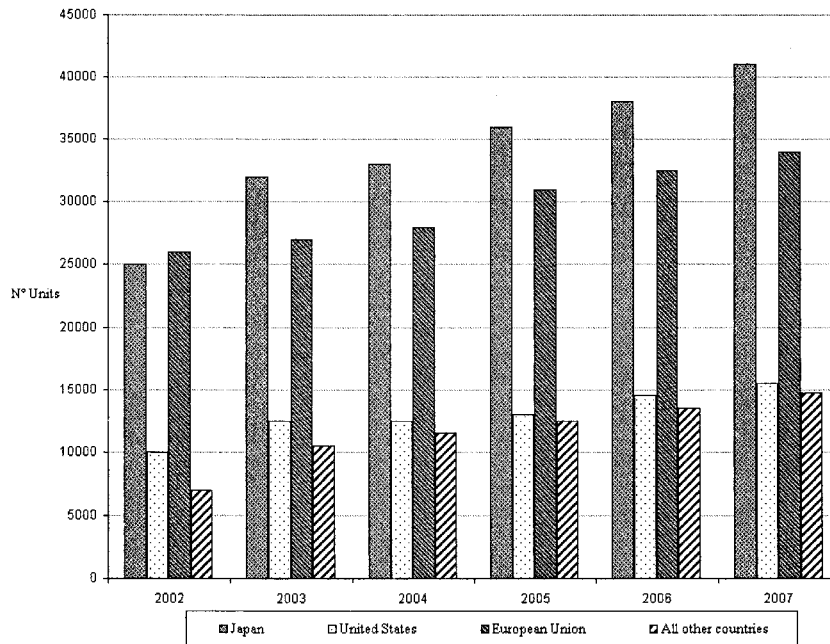


Figure 1.24 New robot installations per year [23]

The IFR forecast for 2007 expects a steady growth of robot installations in the European Union (6.1% per year) and in the United states (5.8% per year). Although Japan's new installations experienced different growth rates in the period 1999-2001, a significant recovery started in 2002 and a steady growth rate is expected at least until 2007 (5.7% per year).

Robots are becoming very common in any industrial installation (Figure 1.23 shows the number of robots per 10 000 workers for the motor vehicle industry, one of the most successful areas of robot operation) where they cooperate with human workers to achieve better efficiency and productivity. The pressure to invest in robots, namely regarding cost savings, increases in productivity and quality, and transferring dangerous tasks from humans to machines, i.e., to remain competitive in the global market, configures a scenario where humans and robots share the working space. In fact, in the beginning of the 21<sup>st</sup> century, robots are already



human coworkers and successful installations must consider carefully the human-robot interaction and handle it as efficiently as possible.

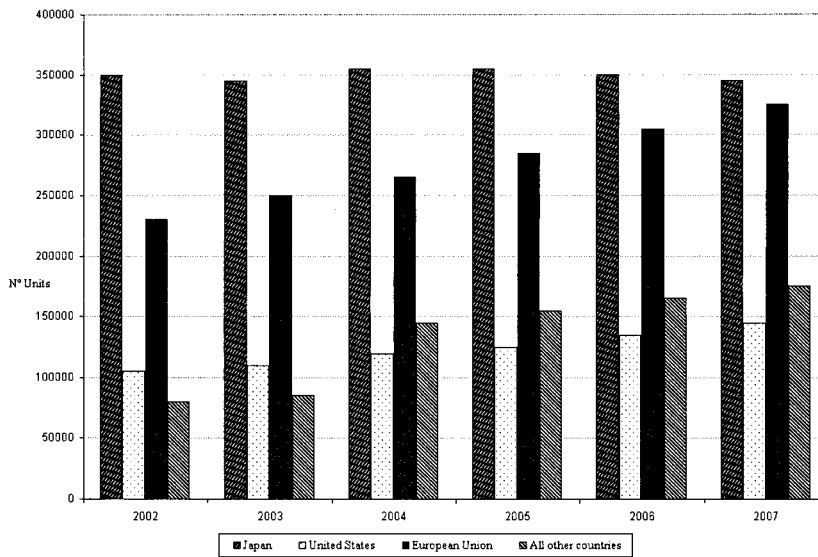


Figure 1.25 Operational stocks at the end of the year [23]

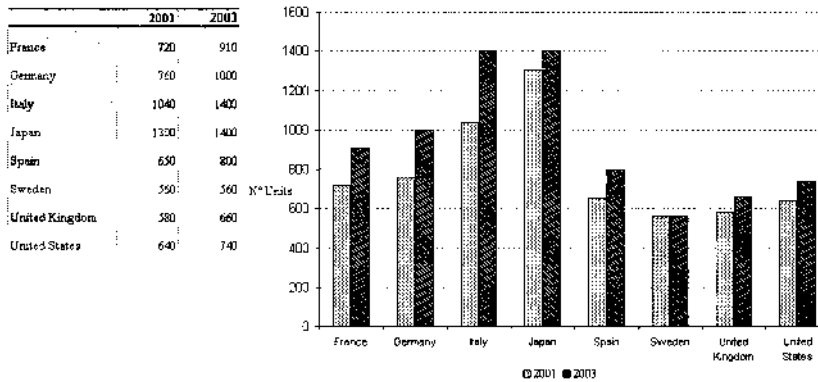


Figure 1.26 Number of robots per 10 000 workers in the car industry [23]

Consequently, industrial robots fit well with the two main challenges faced currently by modern manufacturing: more quality at lower prices and the need to improve productivity. Those are the requirements to keep manufacturing plants in developed countries, rather in the low-salary regions of the world. Other very important characteristics of manufacturing systems are flexibility and agility since companies need to respond to a very dynamic market with products that have low life-cycles due to fashion tendencies and worldwide competition.

So, manufacturing companies need to respond to market needs efficiently, keeping their products competitive. This requires a very efficient and controlled manufacturing process, where focus is on automation, computers and software.

The final objective is to achieve semi-autonomous systems, i.e., highly automated systems that require only minor operator intervention. In many industries, production is closed tracked in any part of the manufacturing cycle, which is composed by several in-line manufacturing systems that perform the necessary operations to transform the raw materials into a final product. In many cases, if properly designed, those individual manufacturing systems require simple parameterization to execute the tasks they are designed to execute. If that parameterization can be commanded remotely by automatic means from where it is available, then the system becomes almost autonomous in that operator intervention is reduced to the minimum and essentially needed for error and maintenance situations. Human and machines can cooperate doing their own tasks, more or less autonomously, and interface more closely when required by the manufacturing process.

A system like this will improve efficiency and agility, since it is less dependent on human operators. Also, since those systems are built under distributed frameworks, based on client-server software architectures that require a collection of functions that implement the system functionality, it is easier to change production by adjusting parameterization (a software task now) which also contributes to agility. Furthermore, since all information about each item produced is available in the manufacturing tracking software, it is logical to use it to command some of the shop floor manufacturing systems, namely the ones that require simple parameterization to work properly. This procedure would take advantage of the available information and computing infrastructure, avoiding unnecessary operator interfaces to command the system. Also, further potential gains in terms of flexibility and productivity are evident.

## 1.6 Overview of the rest of the book

This book is about industrial robot programming in the beginning of twentieth first century. It focuses on the important aspects of designing and building robotic manufacturing cells, which explore the capabilities of the actual industrial equipment, and the available computer and software technologies. Special attention will be paid to exploring the available input devices and systems that can be used to create more efficient human-machine interfaces, namely to the programming, control, and supervision tasks performed by non-technical personnel.

**Chapter Two** (“Robot Manipulators and Control Systems”) introduces most of the industrial robotic equipment currently available, namely aspects related with industrial robotic manipulators, their control systems and programming

environments. In the process, two specific manipulators will be considered closely since both will be used in many examples presented in the rest of the book.

**Chapter Three** (“Software Interfaces”) discusses software interfaces that can be used to develop distributed industrial manufacturing cells. It covers the mechanisms and techniques used to interface robots with computers, as well as intelligent sensors, actuators, other factory resources, production management software, and so on. The software discussed in this chapter is used in all the examples presented in the book, and is the core of several industrial and laboratory applications.

**Chapter Four** (“Interface Devices and Systems”) presents an overview of several available devices and systems that can be used to program, control, and supervise industrial robotic manufacturing cells. The intention here is to show that these interfaces and systems are available and to demonstrate, with application examples, how they can be explored to design solutions easier to use and program by non-technical operators.

**Chapter Five** (“Industrial Manufacturing Systems”) is dedicated to a few application examples designed and implemented recently by the author of this book. The applications are described in detail to enable the interested reader to explore further. Although the selected examples were designed for specific applications, and carefully tuned for the industry in which they are currently used, the discussion is kept general since most of the problems addressed are common to many industries.

Finally, **chapter six** (“Final Notes”) presents a brief summary of the concepts and ideas presented in this book, and lists a few possible actions that the interested reader can follow to learn more about this important area of modern engineering.

A good collection of references is also presented at the end of each chapter to enable the reader to explore further.

## 1.7 References

- [1] Pires, JN, “Welding Robots. Technology, systems issues and applications”, Springer, 2005.
- [2] Kusiak, A, “Computational Intelligence in Design and Manufacturing”, John Wiley & Sons, 2000.
- [3] Halsall F., "Data Communications, Computer Networks and Open Systems", Third Edition, Addison-Wesley, 1992.
- [4] Tesla, N, “My Inventions: Autobiography of Nicola Tesla”, Willinston, VT: Hart Brothers, 1983.
- [5] Rosheim, M, “Robot Evolution: The Development of Anthrobots”, New York: John Willey & Sons, 1994.

- [6] Rosheim, M, "In the Footsteps of Leonardo", IEEE Robotics and Automation Magazine, June 1997.
- [7] Pedretti, C, "Leonardo Architect", Rizzoli International Publications, New York, 1981.
- [8] Mars Exploration WebSite (NASA), <http://mars.jpl.nasa.gov>
- [9] Mclennan Ltd., Precision Motion Control, <http://www.mclennan.co.uk/>
- [10] Siemens, Micro Automation SIMATIC S7-200, [www.siemens.com/s7-200](http://www.siemens.com/s7-200)
- [11] Robot Nicola WebSite, <http://robotics.dem.uc.pt/norberto/nova/nicola.htm>
- [12] Pires, JN, "Semi-autonomous Manufacturing Systems: the role of the HMI software and of the manufacturing tracking software", IFAC Journal on Mechatronics, accepted for publication on Vol. 15, to appear in 2005.
- [13] Pires, JN, Sá da Costa JMG, "Object Oriented and Distributed Approach for Programming Robotic Manufacturing Cells", IFAC Journal on Robotics and Computer Integrated Manufacturing, February 2000.
- [14] Pires, JN, Paulo, S, "High-efficient de-palletizing system for the non-flat ceramic industry", Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, 2003.
- [15] Pires, JN, "Object-oriented and distributed programming of robotic and automation equipment", Industrial Robot, An International Journal, MCB University Press, July 2000.
- [16] Pires, JN, "Interfacing Robotic and Automation Equipment with Matlab", IEEE Robotics and Automation Magazine, September 2000.
- [17] Pires, JN, "Force/torque sensing applied to industrial robotic deburring", Sensor Review Journal, MCB University Press, July 2002.
- [18] Pires, JN, Godinho, T, Ferreira, P, "CAD interface for automatic robot welding programming", Sensor Review Journal, MCB University Press, July 2002.
- [19] Bloomer, J, "Power Programming with RPC", O'Reilly & Associates, Inc., 1992.
- [20] Box, D, "Essential COM", Addison-Wesley, 1998
- [21] Rogerson, D, "Inside COM", Microsoft Press, 1997.
- [22] Visual C++ .NET 2003 Programmers Reference, Microsoft, 2003 (reference can be found at Microsoft's Web site in the Visual C++ .NET location)
- [23] "World Robotics 2004 – Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment", International Federation of Robotics and the United Nations, 2004.

## Robot Manipulators and Control Systems

### 2.1 Introduction

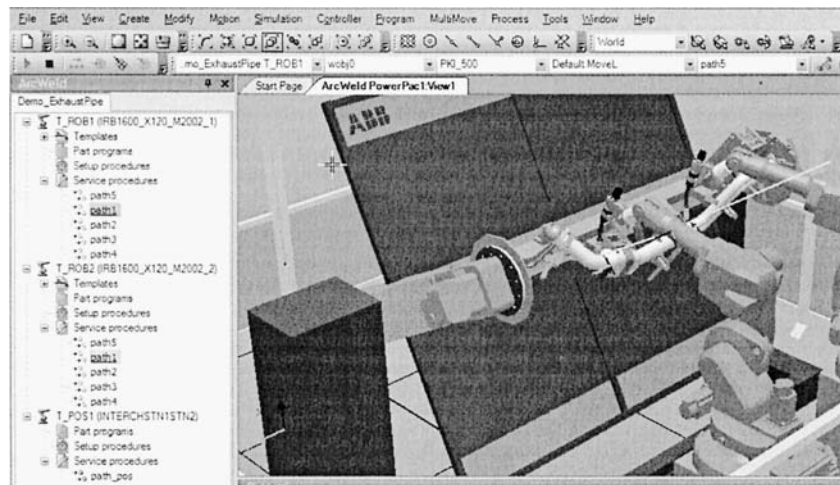
This book focuses on industrial robotic manipulators and on industrial manufacturing cells built using that type of robots. This chapter covers the current practical methodologies for kinematics and dynamics modeling and computations. The kinematics model represents the motion of the robot without considering the forces that cause the motion. The dynamics model establishes the relationships between the motion and the forces involved, taking into account the masses and moments of inertia, i.e., the dynamics model considers the masses and inertias involved and relates the forces with the observed motion, or instead calculates the forces necessary to produce the required motion. These topics are considered very important to study and efficient use of industrial robots.

Both the kinematics and dynamics models are used currently to design, simulate, and control industrial robots. The kinematics model is a prerequisite for the dynamics model and fundamental for practical aspects like motion planning, singularity and workspace analysis, and manufacturing cell graphical simulation. For example, the majority of the robot manufacturers and many independent software vendors offer graphical environments where users, namely developers and system integrators, can design and simulate their own manufacturing cell projects (Figure 2.1).

Kinematics and dynamics modeling is the subject of numerous publications and textbooks [1-4]. The objective here is to present the topics without prerequisites, covering the fundamentals. Consequently, a real industrial robot will be used as an example which makes the chapter more practical, and easier to read. Nevertheless, the reader is invited to seek further explanation in the following very good sources:

1. *Introduction to Robotics*, JJ Craig, John Willey and Sons, Chapters 2 to 7.

2. *Modeling and Control of Robotic Manipulators*, F. Sciavicco and B. Siciliano, McGraw Hill, Chapters 2 to 5.
3. *Handbook of Industrial Robotics*, 2<sup>nd</sup> edition, Shimon Nof, Chapter 6 written by A. Goldenberg and M. Emani.



**Figure 2.1** Aspect of a graphical simulation package (*RobotStudio* – ABB Robotics)

Another important practical aspect is the way how these topics are implemented and used by actual robot control systems. This chapter also reviews the fundamental aspects of robot control systems from the perspective of an engineer and of a system integrator. The objective is to introduce the main components and modules of modern robot control systems, by examining some of the control systems available commercially.

## 2.2 Kinematics

Actual industrial robot manipulators are very advanced machines exhibiting high precision and repeatability. It's common to have medium payload robots (16 to 20kg of payload) offering repeatability up to 0.1 mm, with smaller robots exhibiting even better performances (up to 0.01 mm). These industrial robots are basically composed by rigid links, connected in series by joints (normally six joints), having one end fixed (base) and another free to move and perform useful work when properly tooled (*end-effector*). As with the human arm, robot manipulators use the first three joints (arm) to position the structure and the remaining joints (wrist, composed of three joints in the case of the industrial manipulators) are used to orient the *end-effector*. There are five types of arms commonly used by actual industrial robot manipulators (Figure 2.2): *cartesian*, *cylindrical*, *polar*, *SCARA* and *revolution*.

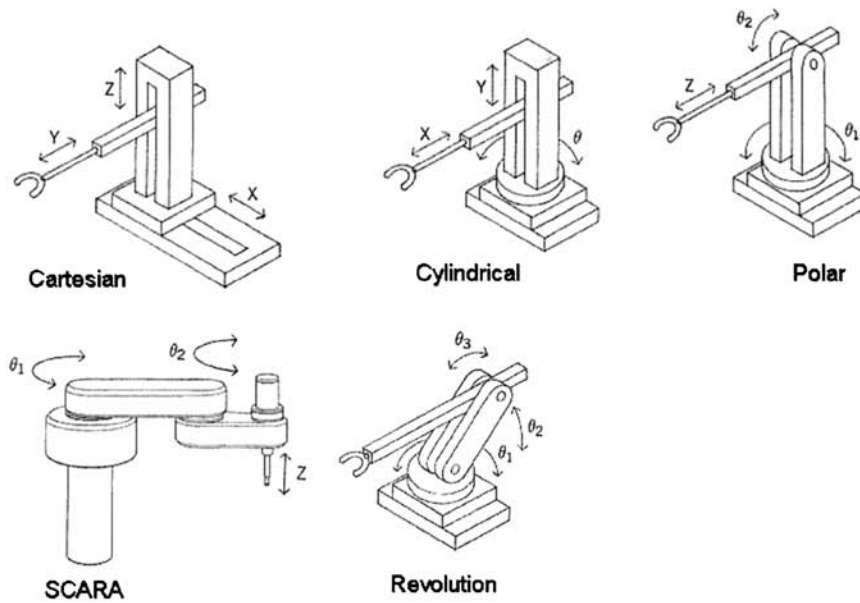


Figure 2.2 Types of arms used with actual robot manipulators

In terms of wrist designs, there are two main configurations (Figure 2.3):

1. pitch-yaw-roll (XYZ) like the human arm
2. roll-pitch-roll (ZYZ) or spherical wrist

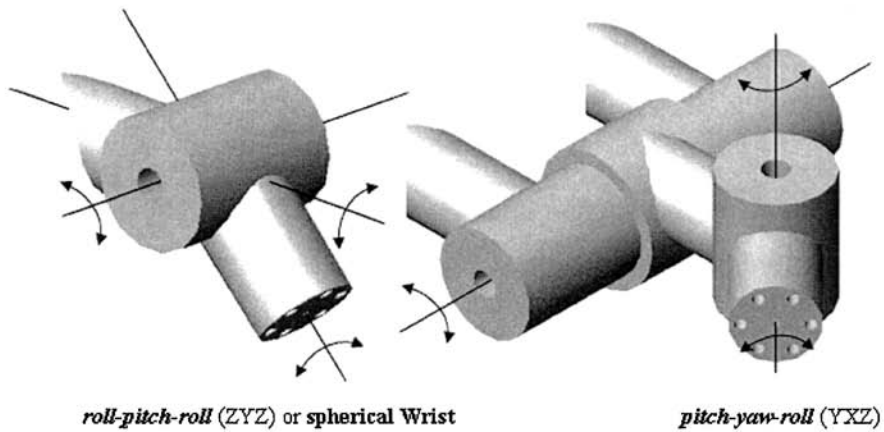


Figure 2.3 Wrist design configurations

The spherical wrist is the most popular because it is mechanically simpler to implement. Nevertheless, it exhibits singular configurations that can be identified

and consequently avoided when operating with the robot. The trade between simplicity of robust solutions and the existence of singular configurations is favorable to the spherical wrist design, and that is the reason for its success.

The position and orientation of the robot's *end-effector* (tool) is not directly measured but instead computed using the individual joint position readings and the kinematics of the robot. Inverse kinematics is used to obtain the joint positions required for the desired *end-effector* position and orientation [1]. Those transformations involve three different representation spaces: *actuator space*, *joint space* and *cartesian space*. The relationships between those spaces will be established here, with application to an ABB IRB1400 industrial robot (Figure 2.4). The discussion will be kept general for an anthropomorphic<sup>1</sup> manipulator with a spherical wrist<sup>2</sup>.

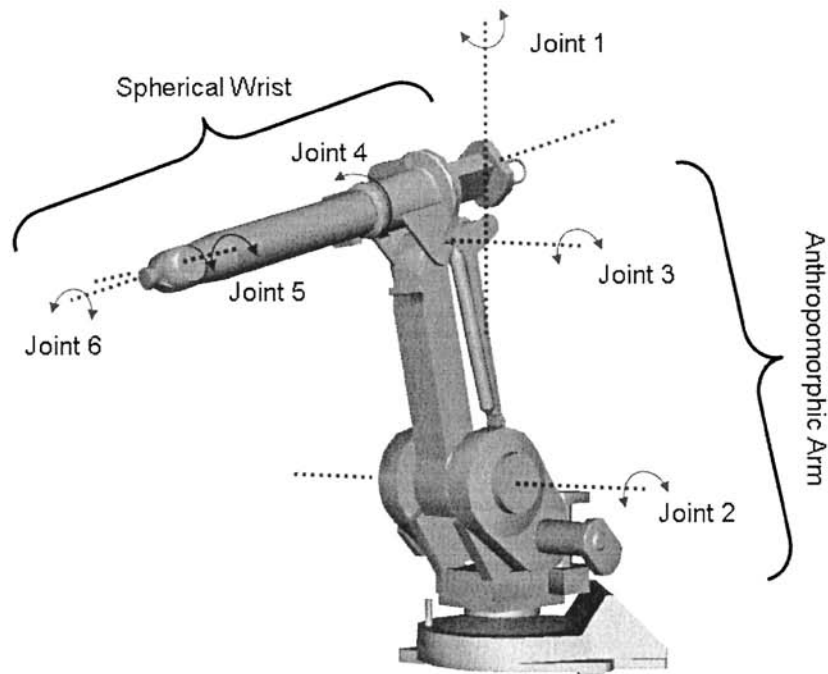


Figure 2.4 ABB IRB1400 industrial robot

<sup>1</sup> An anthropomorphic structure is a set of three revolute joints, with the first joint orthogonal to the other two which are parallel

<sup>2</sup> A spherical wrist has three revolute joints whose axes intersect at a single point



**Table 2.1** Denavit-Hartenberg parameters for the IRB1400

Link	$\theta_i$ (°)	$\alpha_{i-1}$ (°)	$a_{i-1}$ (mm)	$d_i$ (mm)
1	$\theta_1$ (0°)	0°	0	475
2	$\theta_2$ (90°)	90°	150	0
3	$\theta_3$ (0°)	0°	600	0
4	$\theta_4$ (0°)	90°	120	720
5	$\theta_5$ (0°)	-90°	0	0
6	$\theta_6$ (0°)	90°	0	85 + d

where d is an extra length associated with the *end-effector*

**Table 2.2** Workspace and maximum velocities for the IRB1400

Joint	Workspace (°)	Maximum Velocity (°/s)
1	+170° to -170°	110°/s
2	+70° to -70°	110°/s
3	+70° to -65°	110°/s
4	+150° to -150°	280°/s
5	+115° to -115°	280°/s
6	+300° to -300°	280°/s

Figure 2.5 represents, for simplicity, the robot manipulator axis lines and the assigned frames. The *Denavit-Hartenberg* parameters, the joint range and velocity limits are presented in Tables 2.1 and 2.2. The represented frames and associated parameters were found using Craig's convention [1].

### 2.2.1 Direct Kinematics

By simple inspection of Figure 2.5 it is easy to conclude that the last three axes form a set of *ZYZ Euler* angles [1,2] with respect to frame 4. In fact, the overall rotation produced by those axes is obtained from:

1. rotation about  $Z_4$  by  $\theta_4$
2. rotation about  $Y'_4=Z'_5$  by  $\theta_5$
3. rotation about  $Z''_4=Z''_6$  by  $\theta_6$ .<sup>3</sup>

which gives the following rotation matrix,

---

<sup>3</sup>  $Y'_4$  corresponds to axis  $Y_4$  after rotation about  $Z_4$  by  $\theta_4$  and  $Z''_4$  corresponds to  $Z_4$  after rotation about  $Y'_4=Z'_5$  by  $\theta_5$

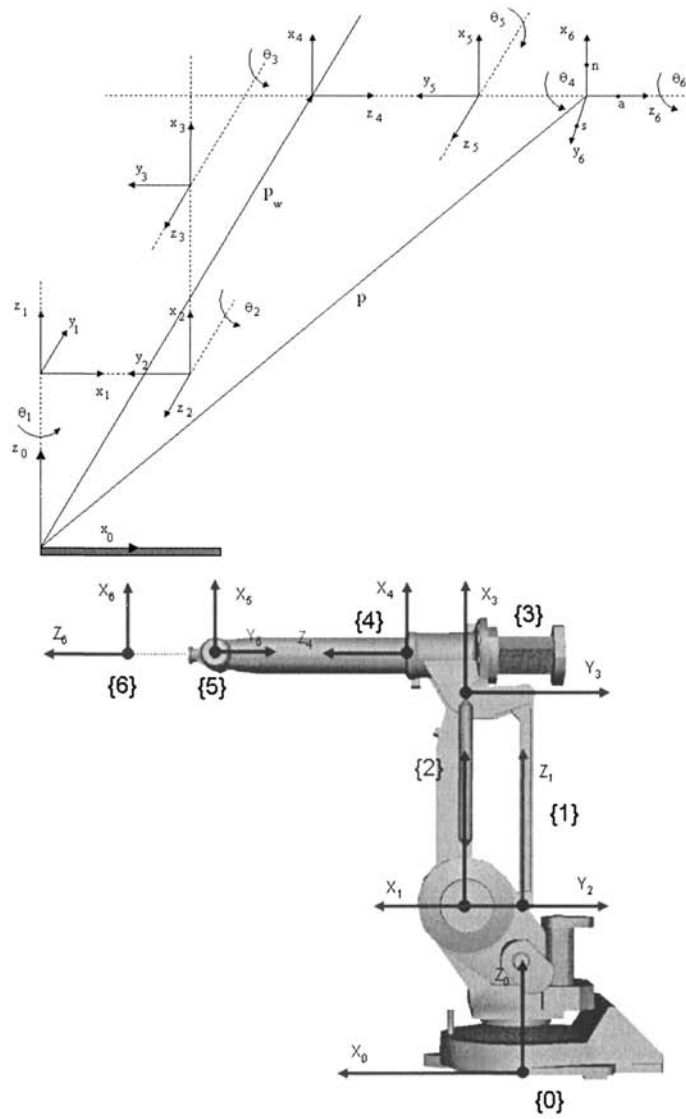


Figure 2.5 Link frame assignment

$$\begin{aligned}
 R_{\text{Euler}} &= R_z(\theta_4).R_{y^4}(\theta_5).R_{z^4}(\theta_6) = \\
 &= \begin{bmatrix} c_4 & -s_4 & 0 \\ s_4 & c_4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & 0 & s_5 \\ 0 & 1 & 0 \\ -s_5 & 0 & c_5 \end{bmatrix} \begin{bmatrix} c_6 & -s_6 & 0 \\ s_6 & c_6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} c_4 c_5 c_6 - s_4 s_6 & -c_4 c_5 s_6 - s_4 c_6 & c_4 s_5 \\ s_4 c_5 c_6 + c_4 s_6 & -s_4 c_5 s_6 + c_4 c_6 & s_4 s_5 \\ -s_5 c_6 & s_5 s_6 & c_5 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = R
 \end{aligned} \tag{2.1}$$

The above rotation matrix  $R$ , in accordance with the assigned frame settings, should verify the following two equations:

$$\begin{aligned}
 R_6^3 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} R \\
 R(\theta_4 = 0) &= R_6^4
 \end{aligned} \tag{2.2}$$

The values of  $\theta_4$ ,  $\theta_5$  and  $\theta_6$  can be now obtained. Comparing  $r_{13}$  with  $r_{23}$  (considering  $s_5 \neq 0$ ) results in,

$$\theta_4 = A \tan 2(r_{23}, r_{13}) \tag{2.3}$$

Squaring and summing  $r_{13}$  and  $r_{23}$  and comparing the result with  $r_{33}$  gives,

$$\theta_5 = A \tan 2(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \tag{2.4}$$

if a positive square-root of  $r_{13}^2 + r_{23}^2$  is chosen: this assumption limits the range of  $\theta_5$  to  $[0, \pi]$ .

Using the same argument now considering elements  $r_{31}$  and  $r_{32}$  the following is obtained for  $\theta_6$ :

$$\theta_6 = A \tan 2(r_{32}, -r_{31}) \tag{2.5}$$

For  $\theta_5 \in [-\pi, 0]$  the solution is:

$$\begin{aligned}
 \theta_4 &= A \tan 2(-r_{23}, -r_{13}) \\
 \theta_5 &= A \tan 2(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \\
 \theta_6 &= A \tan 2(-r_{32}, r_{31})
 \end{aligned} \tag{2.6}$$

The IRB1400 is an anthropomorphic manipulator with spherical wrist. The anthropomorphic structure of the first three joints is the one that offers better

dexterity to the robot manipulator. The first three joints are used to position the wrist. The orientation of the wrist is managed by the wrist spherical structure, which is also the one that gives higher dexterity. Using the link transformation matrix definition derived at [1],

$$T_i^{i-1} = \begin{bmatrix} c_i & -s_i & 0 & a_{i-1} \\ s_i c \alpha_{i-1} & c_i c \alpha_{i-1} & -s \alpha_{i-1} & -s \alpha_{i-1} d_i \\ s_i s \alpha_{i-1} & c_i s \alpha_{i-1} & c \alpha_{i-1} & c \alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

the direct kinematics of the ABB IRB1400 robot manipulator can be easily obtained (as presented in Figure 2.6).

$$T_1^0 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2^1 = \begin{bmatrix} -s_2 & -c_2 & 0 & a_1 \\ 0 & 0 & -1 & 0 \\ c_2 & -s_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & a_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^3 = \begin{bmatrix} c_4 & -s_4 & 0 & a_3 \\ 0 & 0 & -1 & -d_4 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_5^4 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_5 & -c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^5 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & -1 & -d_6 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2^0 = \begin{bmatrix} -c_1 s_2 & -c_1 c_2 & s_1 & a_1 c_1 \\ -s_1 s_2 & -s_1 c_2 & -c_1 & a_1 s_1 \\ c_2 & -s_2 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^0 = \begin{bmatrix} -c_1 s_{23} & -c_1 c_{23} & s_1 & -a_2 c_1 s_2 + a_1 c_1 \\ -s_1 s_{23} & -s_1 c_{23} & -c_1 & -a_2 s_1 s_2 + a_1 s_1 \\ c_{23} & -s_{23} & 0 & a_2 c_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = \begin{bmatrix} -c_1 s_{23} c_4 + s_1 s_4 & c_1 s_{23} s_4 + s_1 c_4 & c_1 c_{23} & d_4 c_1 c_{23} - a_3 c_1 s_{23} - a_2 c_1 s_2 + a_1 c_1 \\ -s_1 s_{23} c_4 - c_1 s_4 & s_1 s_{23} s_4 - c_1 c_4 & s_1 c_{23} & d_4 s_1 c_{23} - a_3 s_1 s_{23} - a_2 s_1 s_2 + a_1 s_1 \\ c_{23} c_4 & c_{23} s_4 & s_{23} & d_4 s_{23} + a_3 c_{23} + a_2 c_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^3 = \begin{bmatrix} c_4 c_5 c_6 - s_4 s_6 & -c_4 c_5 s_6 - s_4 c_6 & c_4 s_5 & d_6 c_4 s_5 + a_3 \\ s_5 c_6 & -s_5 s_6 & -c_5 & -d_6 c_5 - d_4 \\ s_4 c_5 c_6 + c_4 s_6 & -s_4 c_5 s_6 + c_4 c_6 & s_4 s_5 & d_6 s_4 s_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^4 = \begin{bmatrix} c_5 c_6 & -c_5 s_6 & s_5 & d_6 s_5 \\ s_6 & c_6 & 0 & 0 \\ -s_5 c_6 & s_5 s_6 & c_5 & d_6 c_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } T_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x^0 \\ r_{21} & r_{22} & r_{23} & p_y^0 \\ r_{31} & r_{32} & r_{33} & p_z^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ with,}$$

$$\begin{aligned} r_{11} &= ((s_1 s_4 - c_1 s_{23} c_4) c_5 - c_1 c_{23} s_5) c_6 + (c_1 s_{23} s_4 + s_1 c_4) s_6 \\ r_{12} &= ((-s_1 s_4 + c_1 s_{23} c_4) c_5 + c_1 c_{23} s_5) s_6 + (c_1 s_{23} s_4 + s_1 c_4) c_6 \\ r_{13} &= (-c_1 s_{23} c_4 + s_1 s_4) s_5 + c_1 c_{23} c_5 \\ r_{21} &= ((-s_1 s_{23} c_4 - c_1 s_4) c_5 - s_1 c_{23} s_5) c_6 + (s_1 s_{23} s_4 - c_1 c_4) s_6 \\ r_{22} &= ((s_1 s_{23} c_4 + c_1 s_4) c_5 + s_1 c_{23} s_5) s_6 + (s_1 s_{23} s_4 - c_1 c_4) c_6 \\ r_{23} &= (-s_1 s_{23} c_4 - c_1 s_4) s_5 + s_1 c_{23} c_5 \\ r_{31} &= (c_{23} c_4 c_5 - s_{23} s_5) c_6 - c_{23} s_4 s_6 \\ r_{32} &= (-c_{23} c_4 c_5 + s_{23} s_5) s_6 - c_{23} s_4 c_6 \\ r_{33} &= c_{23} c_4 s_5 + s_{23} c_5 \\ p_x^0 &= ((-c_1 s_{23} c_4 + s_1 s_4) s_5 + c_1 c_{23} c_5) d_6 + d_4 c_1 c_{23} - a_3 c_1 s_{23} - a_2 c_1 s_2 + a_1 c_1 \\ p_y^0 &= ((-s_1 s_{23} c_4 - c_1 s_4) s_5 + s_1 c_{23} c_5) d_6 + d_4 s_1 c_{23} - a_3 s_1 s_{23} - a_2 s_1 s_2 + a_1 s_1 \\ p_z^0 &= d_6 (c_{23} c_4 s_5 + s_{23} c_5) + d_4 s_{23} + a_3 c_{23} + a_2 c_2 + d_1 \end{aligned}$$

Figure 2.6 Direct kinematics of an ABB IRB 1400 industrial robot

Having derived the direct kinematics of the IRB 1400, it's now possible to obtain the *end-effector* position and orientation from the individual joint angles  $(\theta_1, \theta_2, \theta_3, \theta_4; \theta_5, \theta_6)$ .

### 2.2.2 Inverse Kinematics

Inverse kinematics deals with the problem of finding the required joint angles to produce a certain desired position and orientation of the *end-effector*. Finding the inverse kinematics solution for a general manipulator can be a very tricky task. Generally they are non-linear equations. Close-form solutions may not be possible and multiple, infinity, or impossible solutions can arise. Nevertheless, special cases have a closed-form solution and can be solved.

The sufficient condition for solving a six-axis manipulator is that it must have three consecutive revolute axes that intersect at a common point: *Pieper* condition [5]. Three consecutive revolute parallel axes is a special case of the above condition, since parallel lines can be considered to intersect at infinity. The ABB IRB 1400 meets the *Pieper* condition due to the spherical wrist.

For these types of manipulators, i.e. manipulators that meet the *Pieper* condition, it is possible to decouple the inverse kinematics problem into two sub-problems: position and orientation. A simple strategy [1,2] can then be used to solve the inverse kinematics, by separating the position problem from the orientation problem. Consider Figure 2.5, where the position and orientation of the *end-*

*effector* is defined in terms of  $\mathbf{p}$  and  $R_6^0 = [n \ s \ a]$ . The wrist position ( $\mathbf{p}_w$ ) can be found using

$$\mathbf{p}_w = \mathbf{p} - d_6 \mathbf{a} \quad (2.8)$$

It is now possible to find the inverse kinematics for  $\theta_1, \theta_2$  and  $\theta_3$  and solve the first inverse kinematics sub-problem, i.e, the position sub-problem. Considering Figure 2.7 it is easy to see that

$$\theta_1 = A \tan 2(p_{wy}, p_{wx})^4 \quad (2.9)$$

Once  $\theta_1$  is known the problem reduces to solving a planar structure. Looking to Figure 2.7 it is possible to successively write

$$p_{wx1} = \sqrt{p_{wx}^2 + p_{wy}^2} \quad (2.10)$$

$$p_{wz1} = p_{wz} - d_1 \quad (2.11)$$

$$p_{wx1'} = p_{wx1} - a_1 \quad (2.12)$$

$$p_{wy1'} = p_{wy1} \quad (2.13)$$

$$p_{wz1'} = p_{wz1} \quad (2.14)$$

and

$$p_{wx1'} = -a_2 s_2 + a_x c_{23} \quad (2.15)$$

$$p_{wz1'} = a_2 c_2 + a_x s_{23} \quad (2.16)$$

---

<sup>4</sup> Another possibility would be  $\theta_1 = \pi + A \tan 2(p_{wy}, p_{wx})$  if we set  $\theta_2 \rightarrow \pi - \theta_2$

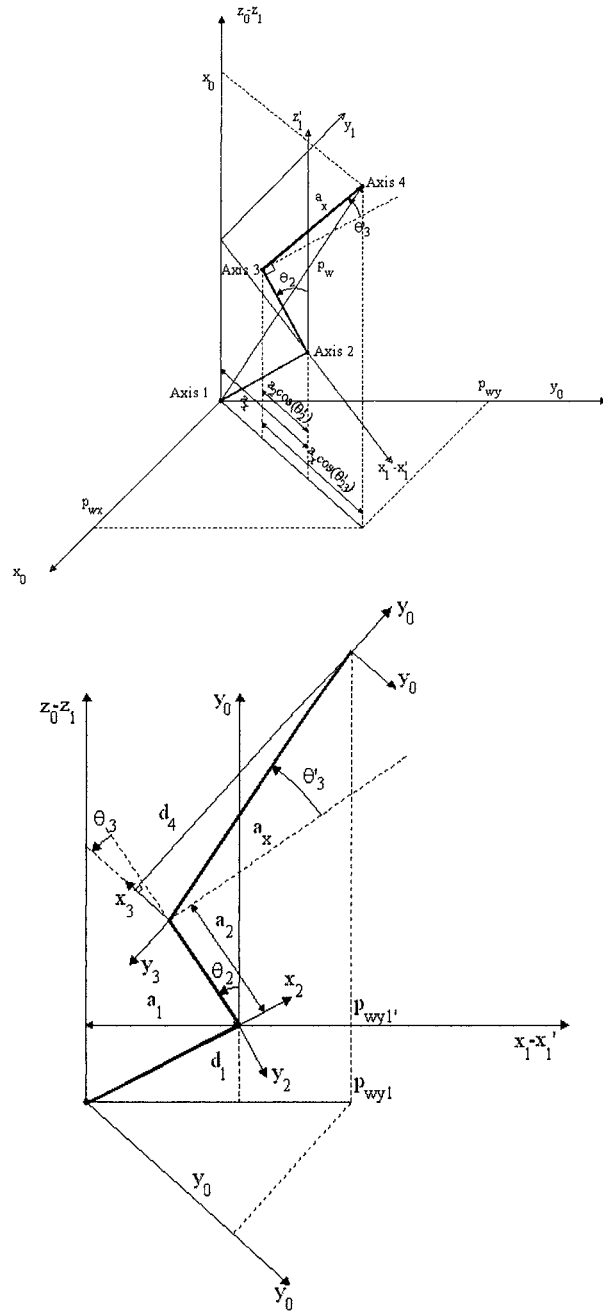


Figure 2.7 Anthropomorphic structure

Squaring and summing equations (2.15) and (2.16) results in

$$p_{wxl'}^2 + p_{wzl'}^2 = a_2^2 + a_x^2 + a_2 a_x s_3' \quad (2.17)$$

which gives

$$s_3' = \frac{p_{wxl'}^2 + p_{wzl'}^2 - a_2^2 - a_x^2}{2a_2 a_x} \quad (2.18)$$

Setting  $c_3' = \pm\sqrt{1-s_3'^2}$  the solution for  $\theta_3'$  will be

$$\begin{aligned} \theta_3' &= A \tan 2(s_3', c_3') \\ \theta_3 &= \theta_3' - A \tan(a_3 / d_4) \end{aligned} \quad (2.19)$$

Now, using  $\theta_3'$  in (2.15)-(2.16) results in a system with two equations with  $s_2$  and  $c_2$  unknowns:

$$\begin{aligned} p_{wxl'} &= a_2 c_2 + a_x (c_2 c_3' - s_2 s_3') \\ p_{wzl'} &= a_2 s_2 + a_x (s_2 c_3' + s_3' c_2) \end{aligned} \quad (2.20)$$

Solving for  $s_2$  and  $c_2$  gives

$$s_2 = \frac{-(a_2 + a_x s_3') p_{wxl'} + a_x c_3' p_{wzl'}}{a_2^2 + a_x^2 + 2a_2 a_x s_3'} \quad (2.21)$$

$$c_2 = \frac{(a_2 + a_x s_3') p_{wzl'} + a_x c_3' p_{wxl'}}{a_2^2 + a_x^2 + 2a_2 a_x s_3'} \quad (2.22)$$

and the solution for  $\theta_2$  will be

$$\theta_2 = A \tan 2(s_2, c_2) \quad (2.23)$$

To solve the second inverse kinematics sub-problem (orientation), i.e., to find the required joint angles  $\theta_4$ ,  $\theta_5$  and  $\theta_6$  corresponding to a given *end-effector* orientation  $R_6^3$ , we simply take advantage of the special configuration of the last three joints. Because the orientation of the *end-effector* is defined by  $R_6^0$ , it's simple to get  $R_6^3$  from,

$$R_6^3 = (R_3^0)^{-1} \cdot R_6^0 = (R_3^0)^T \cdot R_6^0 \quad (2.24)$$

which gives



$$\mathbf{R}_6^3 = \begin{bmatrix} -c_1 s_{23} & -s_1 s_{23} & c_{23} \\ -c_1 c_{32} & -s_1 c_{23} & -s_{23} \\ s_1 & -c_1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (2.25)$$

with

$$\begin{aligned} r_{11} &= -c_1 s_{23} a_{11} - s_1 s_{23} a_{21} + c_{23} a_{31} & r_{12} &= -c_1 s_{23} a_{12} - s_1 s_{23} a_{22} + c_{23} a_{32} \\ r_{13} &= -c_1 s_{23} a_{13} - s_1 s_{23} a_{23} + c_{23} a_{33} & r_{23} &= -c_1 c_{23} a_{13} - s_1 c_{23} a_{23} - s_{23} a_{33} \\ r_{33} &= s_1 a_{13} - c_1 a_{23} \\ r_{21} &= -c_1 c_{23} a_{11} - s_1 c_{23} a_{21} - s_{23} a_{31} & r_{22} &= -c_1 c_{23} a_{12} - s_1 c_{23} a_{22} - s_{23} a_{32} \\ r_{31} &= s_1 a_{11} - c_1 a_{21} & r_{32} &= s_1 a_{12} - c_1 a_{22} \end{aligned}$$

It is now possible to use the previous result for the ZYZ Euler angles to obtain the solutions for  $\theta_4$ ,  $\theta_5$  and  $\theta_6$ .

For  $\theta_5 \in [0, \pi]$  the solution is

$$\begin{aligned} \theta_4 &= A \tan 2(r_{33}, r_{13}) \\ \theta_5 &= A \tan 2(\sqrt{r_{13}^2 + r_{33}^2}, -r_{23}) \\ \theta_6 &= A \tan 2(-r_{22}, r_{21}) \end{aligned} \quad (2.26)$$

For  $\theta_5 \in [-\pi, 0]$  the solution is

$$\begin{aligned} \theta_4 &= A \tan 2(-r_{33}, -r_{13}) \\ \theta_5 &= A \tan 2(-\sqrt{r_{13}^2 + r_{33}^2}, r_{23}) \\ \theta_6 &= A \tan 2(r_{22}, -r_{21}) \end{aligned} \quad (2.27)$$

## 2.3 Jacobian

In this section, the equations necessary to compute the jacobian of the ABB IRB1400 industrial robot are presented and the jacobian is obtained. Nevertheless, the discussion will be kept general for an anthropomorphic robot manipulator. In the process, the equations that describe the linear and angular velocities, static forces, and moments of each of the manipulator links are also presented and the corresponding developments applied to the selected robot.

The jacobian of any robot manipulator structure is a matrix that relates the *end-effector* linear and angular Cartesian velocities with the individual joint velocities:

$$V = \begin{bmatrix} v \\ w \end{bmatrix} = J(\theta) \cdot \dot{\theta} \quad (2.28)$$

where  $J(\theta)$  is the jacobian matrix of the robot manipulator,  $\dot{\theta} = [\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_n]^T$  is the joint velocity vector,  $v = [v_1, v_2, v_3]^T$  is the *end-effector* linear velocity vector, and  $w = [w_1, w_2, w_3]^T$  is the *end-effector* angular velocity vector.

The jacobian is an  $n \times m$  matrix, where  $n$  is the number of degrees of freedom of the robot manipulator and  $m$  is the number of joints. Considering an anthropomorphic robot manipulator with a spherical wrist, the corresponding jacobian will be a  $6 \times 6$  matrix. Basically there are two ways to compute the jacobian:

1. By direct differentiation of the direct kinematics function with respect to the joint variables. This usually leads to the so-called *analytical jacobian*,

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{\phi} \end{bmatrix} = J_A(\theta) \cdot \dot{\theta} \quad (2.29)$$

where  $\dot{p}$  is the time derivative of the position of the *end-effector* frame with respect to the base frame,  $\dot{\phi}$  is the time derivative of the orientation vector expressed in terms of three variables (for instance, ZYZ Euler angles). Obviously,  $\dot{p}$  is the translational velocity of the *end-effector* and  $\dot{\phi}$  is the rotational velocity.

2. By computing the contributions of each joint velocity to the components of the *end-effector* Cartesian linear and angular velocities. This procedure leads to the *geometric jacobian*.

Generally, the analytical and geometrical jacobian are different from each other. Nevertheless, it is always possible to write

$$w = T(\phi) \cdot \dot{\phi} \quad (2.30)$$

where  $T$  is a transformation matrix from  $\dot{\phi}$  to  $w$ . Once  $T(\phi)$  is given, the analytical jacobian and geometric jacobian can be related by

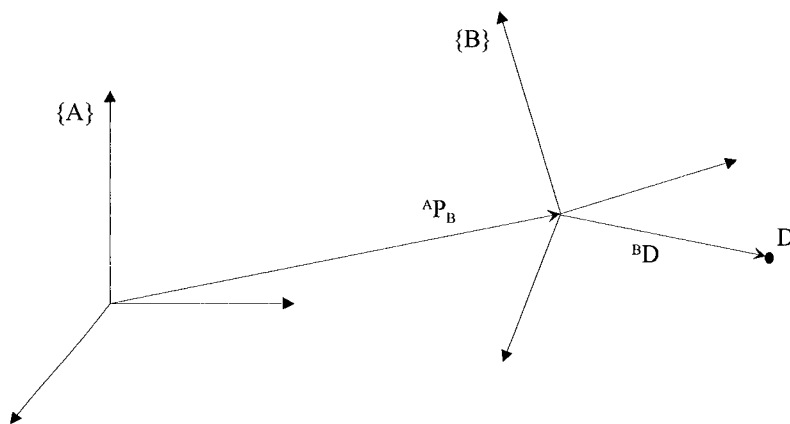
$$V = \begin{bmatrix} I & 0 \\ 0 & T(\phi) \end{bmatrix} \cdot \dot{x} = T_J(\phi) \cdot \dot{x} \quad (2.31)$$

which gives

$$J = T_J(\phi) \cdot J_A \quad (2.32)$$

Here the geometric jacobian will be calculated, because in the process the linear and angular velocities of each link will also be obtained. Nevertheless, the analytical jacobian should be used when the variables are defined in the operational space.

First the equations for the link linear and angular velocities and accelerations [1,2] will be obtained. Associating a frame to each rigid body, the rigid body motion can be described by the relative motion of the associated frames. Consider a frame {B} associated with a point D (Figure 2.8).

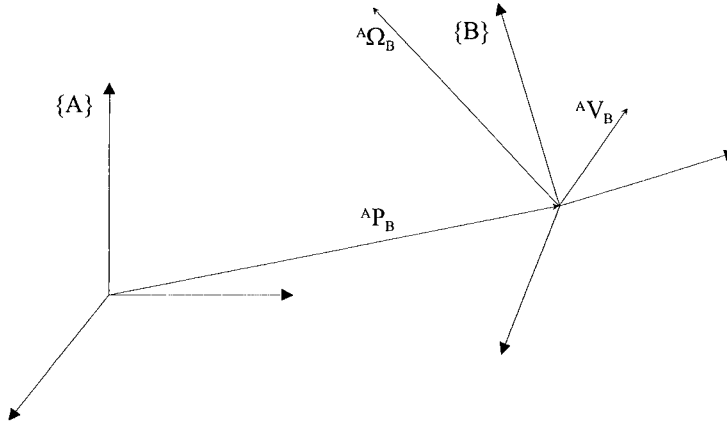


**Figure 2.8** Describing point D relative to a stationary frame

The position vector of point D in frame {B} is  ${}^B D$  and the relative velocity of D described about an arbitrary stationary frame {A} is [6],

$${}^A V_D = {}^A V_B + {}^A_B R {}^B V_D \quad (2.33)$$

If the relative motion between {A} and {B} is non-linear then (2.33) is not valid. The relative motion between two frames {A} and {B} has generally two components: a linear component  ${}^A V_B$  and a non-linear component (the angular or rotational acceleration)  ${}^A \Omega_B$  as in (Figure 2.9).



**Figure 2.9** Relative motion between two frames {A} and {B}

In that general case it can be written [1,6,7],

$${}^A V_D = {}^A V_B + {}^A_B R {}^B V_D + {}^A \Omega_B \times {}^A_B R {}^B D \quad (2.34)$$

where  ${}^A V_D$  is the linear velocity of the origin of frame {B} about frame {A},  ${}^A_B R {}^B V_D$  is the linear velocity of point D about frame {B} expressed in terms of {A} (i.e.,  ${}^A_B R {}^B V_D = {}^A ({}^B V_D)$ ),  ${}^A \Omega_B \times {}^A_B R {}^B D = {}^A \Omega_B \times {}^A D$  is the linear velocity of point D about {A} expressed in terms of {A} as the result of the angular velocity  ${}^A \Omega_B$  of {B} about {A}.

If D is stationary in {B} ( ${}^B V_D = 0$ ) and the origins of {A} and {B} are coincident, i.e., the relative motion of D about {A} is only due to the rotation motion of {B} about {A} described by  ${}^A \Omega_B$ , then  ${}^A V_D = {}^A \Omega_B \times {}^A_B R {}^B D$ . This equation can also be obtained by differentiation of

$${}^A D = {}^A_B R {}^B D \quad (2.35)$$

which yields

$${}^A \dot{D} = {}^A \dot{R} {}^B D + {}^A_B R {}^B \dot{D} \quad (2.36)$$

or since in this special case  ${}^A_B R {}^B \dot{D} = 0$ ,

$${}^A V_D = {}^A \dot{R} {}^B D \quad (2.37)$$

Substituting in (2.37)  ${}^B D = {}^A R^{-1} {}^A D$  results in

$${}^A V_D = {}^A \dot{R} {}^A R^{-1} {}^A D \quad (2.38)$$

Because  ${}^A R$  is an orthonormal matrix, we can write [1,7],

$${}^A \dot{R} {}^A R^{-1} = {}^A S \quad (2.39)$$

where  ${}^A S$  is a *skew-symmetric* matrix associated with  ${}^A R$ .

Using (2.39) in (2.38) gives

$${}^A V_D = {}^A S {}^A D \quad (2.40)$$

The skew-symmetric matrix  ${}^A S$  defined in (2.39) is called *angular velocity matrix*.

Writing  $S$  as

$$S = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} \quad (2.41)$$

and the vector  $\Omega$  ( $3 \times 1$ ) as

$$\Omega = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \quad (2.42)$$

results in

$$S D = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} \cdot \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} = \begin{bmatrix} -\Omega_z D_y + \Omega_y D_z \\ \Omega_z D_x - \Omega_x D_z \\ -\Omega_y D_x + \Omega_x D_y \end{bmatrix} = \Omega \times D \quad (2.43)$$

where  $D = (D_x, D_y, D_z)^T$  is a position vector. The vector  $\Omega$  associated with the angular velocity matrix is called an *angular velocity vector*. Using (2.43) and (2.40) gives

$${}^A V_D = {}^A \Omega_B \times {}^A D \quad (2.44)$$

Considering now the linear and angular accelerations of each link, it's possible to write by direct differentiation of (2.34),

$${}^A\dot{V}_D = {}^A\dot{V}_B + ({}^A_B R \ {}^B V_D)' + {}^A\dot{\Omega}_B \times {}^A_B R \ {}^B D + {}^A\Omega_B \times ({}^A_B R \ {}^B D)' \quad (2.45)$$

or since,

$$({}^A_B R \ {}^B V_D)' = {}^A_B R \ {}^B \dot{V}_D + {}^A\Omega_B \times {}^A_B R \ {}^B V_D$$

and

$$\begin{aligned} ({}^A_B R \ {}^B D)' &= {}^A_B R \ {}^B \dot{V}_D + {}^A\Omega_B \times {}^A_B R \ {}^B D, \\ {}^A\dot{V}_D &= {}^A\dot{V}_B + {}^A_B R \ {}^B \dot{V}_D + 2 {}^A\Omega_B \times {}^A_B R \ {}^B V_D + \\ &+ {}^A\dot{\Omega}_B \times {}^A_B R \ {}^B D + {}^A\Omega_B \times ({}^A\Omega_B \times {}^A_B R \ {}^B D) \end{aligned} \quad (2.46)$$

The above equation is the general equation for the linear acceleration of point D about {A} and expressed in terms of {A}. If  ${}^B D$  is a constant vector (like in robotics applications) then equation (2.46) simplifies to

$${}^A\dot{V}_D = {}^A\dot{V}_B + {}^A\dot{\Omega}_B \times {}^A_B R \ {}^B D + {}^A\Omega_B \times ({}^A\Omega_B \times {}^A_B R \ {}^B D) \quad (2.47)$$

because  ${}^B V_D = {}^B \dot{V}_D = 0$ .

If we consider a third frame {C}, with  ${}^A\Omega_B$  being the angular velocity of {B} about {A} and  ${}^B\Omega_C$  the angular velocity of {B} about {C}, then the angular velocity of {C} about {A} is,

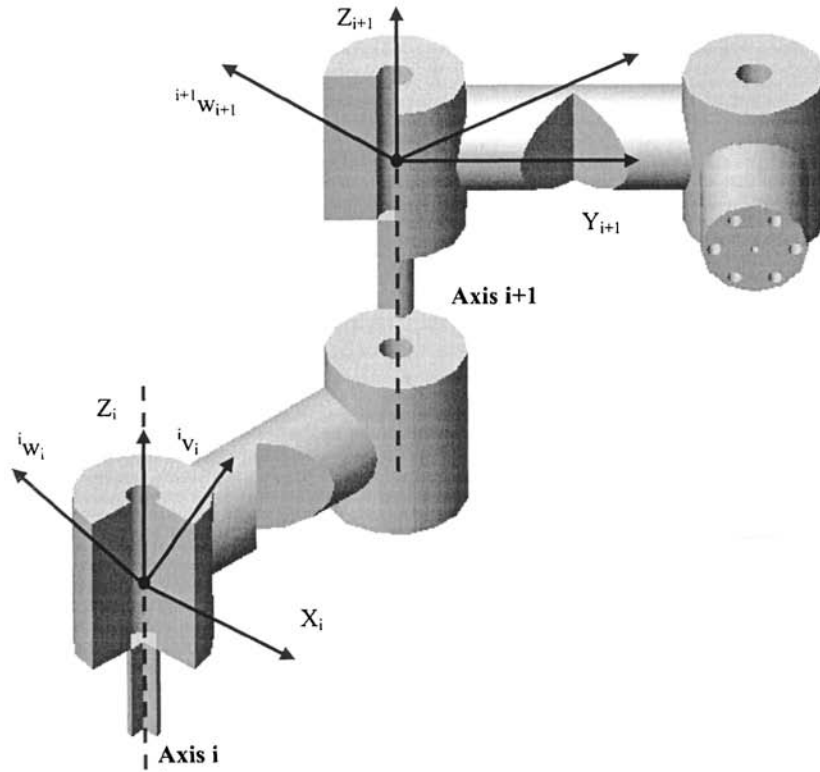
$${}^A\Omega_C = {}^A\Omega_B + {}^A_B R \ {}^B\Omega_C \quad (2.48)$$

Taking the derivative of (2.48) results in

$${}^A\dot{\Omega}_C = {}^A\dot{\Omega}_B + ({}^A_B R \ {}^B\Omega_C)' = {}^A\dot{\Omega}_B + {}^A_B R \ {}^B\dot{\Omega}_C + {}^A\Omega_B \times {}^A_B R \ {}^B\Omega_C \quad (2.49)$$

This is a very useful equation to compute the angular acceleration propagation from link to link.

Let's apply this to a robot manipulator. As mentioned before we will consider only rigid manipulators with revolutionary joints, with the base frame as the reference frame.



**Figure 2.10** Linear and angular velocity vectors of adjacent links

The angular velocity of link (i+1), expressed in terms of {i}, is given by<sup>5</sup>

$${}^i w_{i+1} = {}^i w_i + {}^i_{i+1} R \dot{\theta}_{i+1} {}^{i+1} Z_{i+1} \quad (2.50)$$

It is equal to the angular velocity of link (i) plus the angular velocity of joint (i+1) about  $Z_{i+1}$  expressed in terms of {i}.

Multiplying both sides of (2.50) by  ${}^{i+1}_i R$  results in the angular velocity of link (i+1) expressed in terms of {i+1},

$${}^{i+1} w_{i+1} = {}^{i+1}_i R {}^i w_{i+1} = {}^{i+1}_i R {}^i w_i + \dot{\theta}_{i+1} {}^{i+1} Z_{i+1} \quad (2.51)$$

<sup>5</sup> Note that  $w_{i+1} = \Omega_{i+1}$  and that  ${}^i w_{i+1}$  is the same quantity expressed in terms of {i}.

The linear velocity of the origin of  $\{i+1\}$ , expressed in terms of  $\{i\}$ , is given by

$${}^iV_{i+1} = {}^iV_i + {}^iW_i \times {}^iP_{i+1} \quad (2.52)$$

It is equal to the linear velocity of the origin of  $\{i\}$  plus a term that results from the rotation of the link  $(i+1)$  about  $Z_{i+1}$ . The same solution can be obtained from (7) by making  ${}^iP_{i+1}$  constant in  $\{i\}$ , i.e., by making  ${}^iV_{i+1} = 0$ .

Multiplying both sides of (2.52) by  ${}^{i+1}R_i$  we get the linear velocity of link  $(i+1)$  expressed in terms of  $\{i+1\}$

$${}^{i+1}V_{i+1} = {}^{i+1}R_i ({}^iV_i + {}^iW_i \times {}^iP_{i+1}) \quad (2.53)$$

Applying (2.51) and (2.53) from link to link, the equations for  ${}^nW_n$  and  ${}^nV_n$  (where  $n$  is the number of joints) will be obtained. The equations for  ${}^0W_n$  and  ${}^0V_n$  can be obtained by pre-multiplying  ${}^nW_n$  and  ${}^nV_n$  by  ${}^0R_n$  :

$${}^0W_n = {}^0R_n {}^nW_n \quad (2.54)$$

$${}^0V_n = {}^0R_n {}^nV_n \quad (2.55)$$

It's also important to know how forces and moments distribute through the links and joints of the robot manipulator in a static situation, i.e., how to compute the forces and moments that keep the manipulator still in the various operating static configurations. Considering the manipulator at some configuration, the static equilibrium is obtained by proper balancing of the forces and moments applied to each joint and link, i.e., by cancelling the resultant of all the forces applied to the center of mass of each link (static equilibrium). The objective is to find the set of moments that should be applied to each joint to keep the manipulator in static equilibrium for some working configuration (Figure 2.11).

Considering,

$f_i$  = force applied at link  $(i)$  by link  $(i-1)$

$n_i$  = moment in link  $(i)$  due to link  $(i-1)$

the static equilibrium is obtained when

$${}^iF_i - {}^iF_{i+1} = 0 \quad \text{and} \quad {}^iN_i - {}^iN_{i+1} - {}^iP_{i+1} \times {}^iF_{i+1} = 0 \quad (2.56)$$

i.e., when,

$${}^iF_i = {}^iF_{i+1} \quad (2.57)$$

and



$${}^i n_i = {}^i n_{i+1} + {}^i P_{i+1} \times {}^i f_{i+1} \quad (2.58)$$

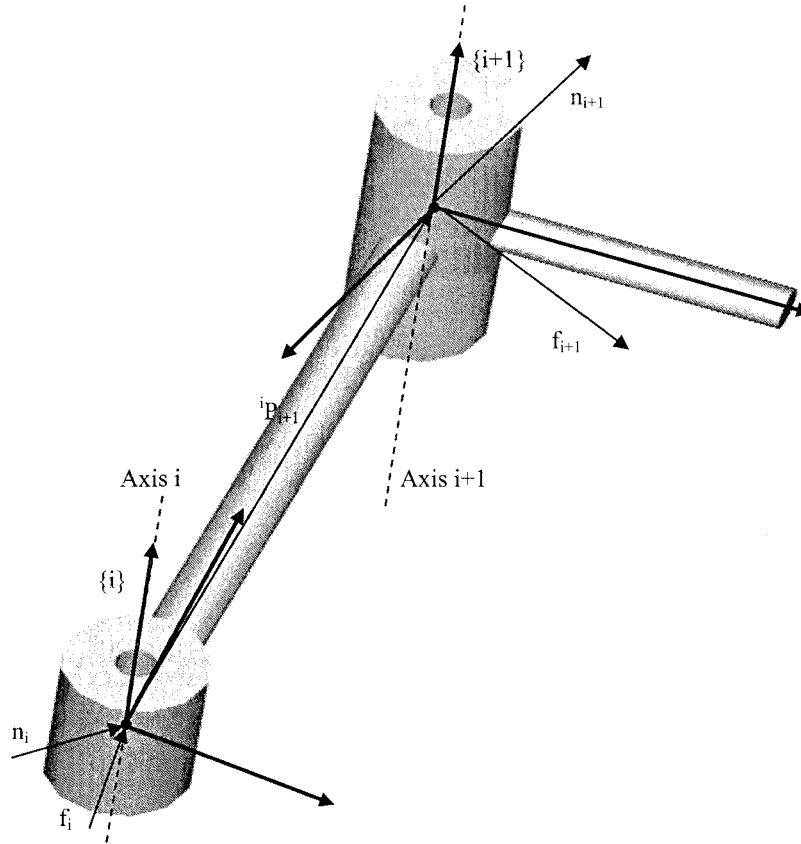


Figure 2.11 Static equilibrium: force balancing over link (i)

Writing the above equations in their own reference frame gives

$${}^i f_i = {}^i_{i+1} R \quad {}^{i+1} f_{i+1} \quad (2.59)$$

$${}^i n_i = {}^i_{i+1} R \quad {}^{i+1} n_{i+1} + {}^i P_{i+1} \times {}^i f_i \quad (2.60)$$

To compute the set of joint moments that hold the manipulator in static equilibrium we must obtain, for each joint (i), the projection of  ${}^i n_i$  over the joint axis

$$\tau_i = {}^i n_i^T \quad {}^i Z_i \quad (2.61)$$

Returning to the jacobian, from (2.54)-(2.55) it's possible to write

$${}^0\mathbf{w}_{i+1} = {}^0\mathbf{w}_i + \dot{\theta}_{i+1} R(\theta_{i+1}) \mathbf{z}_{i+1} \quad (2.62)$$

$${}^0\mathbf{v}_{i+1} = {}^0\mathbf{v}_i + {}^0\mathbf{w}_i \times {}^0\mathbf{P}_{i+1}^i \quad (2.63)$$

Using (1) and (2.62)-(2.63) the  $i^{\text{th}}$  column of the jacobian can be found to be

$${}^0\mathbf{J}_i = \begin{bmatrix} \mathbf{z}_i \times {}^0\mathbf{P}_n^i \\ \mathbf{z}_i \end{bmatrix} \quad (2.64)$$

Applying (2.62), (2.63), and (2.64) to the IRB1400 industrial robot, the equations presented in Figure 2.12 are obtained.

---


$$\begin{aligned}
{}^0\mathbf{v}_0 = 0 \quad {}^0\mathbf{w}_0 = 0 \quad {}^0\mathbf{v}_1 = 0 \quad {}^0\mathbf{w}_1 &= \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \quad {}^0\mathbf{v}_2 = \begin{bmatrix} -a_1 s_1 \dot{\theta}_1 \\ a_1 c_1 \dot{\theta}_1 \\ 0 \end{bmatrix} \quad {}^0\mathbf{w}_2 = \begin{bmatrix} s_1 \dot{\theta}_2 \\ -c_1 \dot{\theta}_2 \\ \dot{\theta}_1 \end{bmatrix} \\
{}^0\mathbf{v}_3 &= \begin{bmatrix} (a_2 s_1 s_2 - a_1 s_1) \dot{\theta}_1 - a_2 c_1 c_2 \dot{\theta}_2 \\ (a_1 c_1 - a_2 c_1 s_2) \dot{\theta}_1 - a_2 c_2 s_1 \dot{\theta}_2 \\ -a_2 s_2 \dot{\theta}_2 \end{bmatrix} \quad {}^0\mathbf{w}_3 = \begin{bmatrix} s_1 (\dot{\theta}_2 + \dot{\theta}_3) \\ -c_1 (\dot{\theta}_2 + \dot{\theta}_3) \\ \dot{\theta}_1 \end{bmatrix} \\
{}^0\mathbf{v}_4 &= \\
&\begin{bmatrix} (a_2 s_2 - a_1 + a_3 s_{23} - d_4 c_{23}) s_1 \dot{\theta}_1 - (a_2 c_2 + d_4 s_{23} - a_3 c_{23}) c_1 \dot{\theta}_2 - (d_4 s_{23} + a_3 c_{23}) c_1 \dot{\theta}_3 \\ (a_1 - a_2 s_2 + d_4 c_{23} - a_3 s_{23}) c_1 \dot{\theta}_1 - (a_2 c_2 + d_4 s_{23} + a_3 c_{23}) s_1 \dot{\theta}_2 - (d_4 s_{23} + a_3 c_{23}) s_1 \dot{\theta}_3 \\ (d_4 c_{23} - a_3 s_{23} - a_2 s_2) \dot{\theta}_2 + (d_4 c_{23} - a_3 s_{23}) \dot{\theta}_3 \end{bmatrix} \\
{}^0\mathbf{w}_4 &= \begin{bmatrix} s_1 (\dot{\theta}_2 + \dot{\theta}_3) + c_1 c_{23} \dot{\theta}_4 \\ -c_1 (\dot{\theta}_2 + \dot{\theta}_3) + s_1 c_{23} \dot{\theta}_4 \\ \dot{\theta}_1 + s_{23} \dot{\theta}_4 \end{bmatrix} \\
{}^0\mathbf{v}_5 &= \\
&\begin{bmatrix} (a_2 s_2 - a_1 + a_3 s_{23} - d_4 c_{23}) s_1 \dot{\theta}_1 - (a_2 c_2 + d_4 s_{23} - a_3 c_{23}) c_1 \dot{\theta}_2 - (d_4 s_{23} + a_3 c_{23}) c_1 \dot{\theta}_3 \\ (a_1 - a_2 s_2 + d_4 c_{23} - a_3 s_{23}) c_1 \dot{\theta}_1 - (a_2 c_2 + d_4 s_{23} + a_3 c_{23}) s_1 \dot{\theta}_2 - (d_4 s_{23} + a_3 c_{23}) s_1 \dot{\theta}_3 \\ (d_4 c_{23} - a_3 s_{23} - a_2 s_2) \dot{\theta}_2 + (d_4 c_{23} - a_3 s_{23}) \dot{\theta}_3 \end{bmatrix} \\
{}^0\mathbf{w}_5 &= \begin{bmatrix} s_1 (\dot{\theta}_2 + \dot{\theta}_3) + c_1 c_{23} \dot{\theta}_4 + (c_1 s_{23} s_4 + s_1 c_4) \dot{\theta}_5 \\ -c_1 (\dot{\theta}_2 + \dot{\theta}_3) + s_1 c_{23} \dot{\theta}_4 + (s_1 s_{23} s_4 - c_1 c_4) \dot{\theta}_5 \\ \dot{\theta}_1 + s_{23} \dot{\theta}_4 - c_{23} s_4 \dot{\theta}_5 \end{bmatrix} \quad {}^0\mathbf{v}_6 = \begin{bmatrix} {}^0\mathbf{v}_6(x) \\ {}^0\mathbf{v}_6(y) \\ {}^0\mathbf{v}_6(z) \end{bmatrix} \\
{}^0\mathbf{v}_6(x) &= ((a_2 s_2 - a_1 + a_3 s_{23} - d_4 c_{23}) s_1 + d_6 ((s_1 s_{23} c_4 + c_1 s_4) s_5 - s_1 c_{23} c_5)) \dot{\theta}_1 + \\
&+ ((-a_2 c_2 - d_4 s_{23} - a_3 c_{23}) - d_6 (c_{23} c_4 s_5 + s_{23} c_5)) c_1 \dot{\theta}_2 + (c_1 (-d_4 s_{23} - a_3 c_{23}) - d_6 (c_{23} c_4 s_5 + \\
&+ s_{23} c_5)) \dot{\theta}_3 + d_6 (s_1 c_4 s_5 + c_1 s_4 s_5 s_{23}) \dot{\theta}_4 + d_6 (s_1 c_5 s_4 - c_1 c_{23} s_5 - c_1 c_4 c_5 s_{23}) \dot{\theta}_5
\end{aligned}$$

$${}^0\mathbf{v}_6(y) = ((a_1 - a_2s_2 + d_4c_{23} - a_3s_{23})c_1 + ((-c_1*s_{23}*c_4 + s_1*s_4)*s_5 + c_1*c_{23}*c_5)*d_6)\dot{\theta}_1 - ((a_2c_2 + d_4s_{23} + a_3c_{23}) + d_6(c_{23}c_4s_5 + s_{23}c_5))s_1\dot{\theta}_2 - ((d_4s_{23} + a_3c_{23})s_1 + d_6s_1(c_{23}c_4s_5 + s_{23}c_5))\dot{\theta}_3 + d_6(s_{23}s_1s_4s_5 - c_1c_4s_5)\dot{\theta}_4 - d_6(c_{23}s_1s_5 + c_1s_4c_5 + s_1c_4c_5s_{23})\dot{\theta}_5$$

$${}^0\mathbf{v}_6(z) = ((c_{23}c_5 - s_{23}c_4s_5)d_6 + d_4c_{23} - a_3s_{23} - a_2s_2)\dot{\theta}_2 + ((c_{23}c_5 - s_{23}c_4s_5)d_6 + d_4c_{23} - a_3s_{23})\dot{\theta}_3 - s_5s_4c_{23}d_6 + (c_{23}c_5c_4 - s_5s_{23})d_6\dot{\theta}_5$$

$${}^0\mathbf{w}_6 =$$

$$\begin{bmatrix} s_1(\dot{\theta}_2 + \dot{\theta}_3) + c_1c_{23}\dot{\theta}_4 + (c_1s_{23}s_4 + s_1c_4)\dot{\theta}_5 + ((-c_1s_{23}c_4 + s_1s_4)s_5 + c_1c_{23}c_5)\dot{\theta}_6 \\ -c_1(\dot{\theta}_2 + \dot{\theta}_3) + s_1c_{23}\dot{\theta}_4 + (s_1s_{23}s_4 - c_1c_4)\dot{\theta}_5 - ((s_1s_{23}c_4 + c_1s_4)s_5 - s_1c_{23}c_5)\dot{\theta}_6 \\ \dot{\theta}_1 + s_{23}\dot{\theta}_4 - c_{23}s_4\dot{\theta}_5 + (c_{23}c_4s_5 + s_{23}c_5)\dot{\theta}_6 \end{bmatrix}$$

$${}^0\mathbf{J} =$$

$$\begin{bmatrix} (a_2s_2 - a_1 + a_3s_{23} - d_4c_{23})s_1 & -(a_2c_2 + d_4s_{23} - a_3c_{23})c_1 & -(d_4s_{23} + a_3c_{23})c_1 & 0 \\ (a_1 - a_2s_2 + d_4c_{23} - a_3s_{23})c_1 & -(a_2c_2 + d_4s_{23} + a_3c_{23})s_1 & -(d_4s_{23} + a_3c_{23})s_1 & 0 \\ 0 & d_4c_{23} - a_3s_{23} - a_2s_2 & d_4c_{23} - a_3s_{23} & 0 \\ 0 & s_1 & s_1 & c_1c_{23} \\ 0 & -c_1 & -c_1 & s_1c_{23} \\ 1 & 0 & 0 & s_{23} \end{bmatrix}$$

$${}^0_3\mathbf{J} = \begin{bmatrix} a_2s_1s_2 - a_1s_1 & -a_2c_1c_2 & 0 \\ a_1c_1 - a_2c_1s_2 & -a_2c_2s_1 & 0 \\ 0 & -a_2s_2 & 0 \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \quad {}^0_6\mathbf{J} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{bmatrix}$$

$$\mathbf{J}_{11} = (a_2*s_2 - a_1 + a_3*s_{23} - d_4*c_{23})*s_1 + d_6*((s_1*c_4*s_{23} + c_1*s_4)*s_5 - s_1*c_{23}*c_5);$$

$$\mathbf{J}_{12} = ((-a_2*c_2 - d_4*s_{23} - a_3*c_{23}) - d_6*(c_{23}*c_4*s_5 + s_{23}*c_5))*c_1;$$

$$\mathbf{J}_{13} = c_1*((-d_4*s_{23} - a_3*c_{23}) - d_6*(c_{23}*c_4*s_5 + s_{23}*c_5));$$

$$\mathbf{J}_{14} = d_6*(s_1*c_4*s_5 + c_1*s_4*s_5*s_{23});$$

$$\mathbf{J}_{15} = d_6*(s_1*c_5*s_4 - c_1*c_{23}*s_5 - c_1*c_4*c_5*s_{23});$$

$$\mathbf{J}_{16} = 0;$$

$$\mathbf{J}_{21} = (a_1 - a_2*s_2 + d_4*c_{23} - a_3*s_{23})*c_1 + ((-c_1*s_{23}*c_4 + s_1*s_4)*s_5 + c_1*c_{23}*c_5)*d_6;$$

$$\mathbf{J}_{22} = -((a_2*c_2 + d_4*s_{23} + a_3*c_{23}) + d_6*(c_{23}*c_4*s_5 + s_{23}*c_5))*s_1;$$

$$\begin{aligned}
\mathbf{J}_{23} &= -(d4*s23 + a3*c23)*s1 - d6*s1*(c23*c4*s5 + s23*c5); \\
\mathbf{J}_{24} &= d6*(s23*s1*s4*s5 - c1*c4*s5); \\
\mathbf{J}_{25} &= -d6*(c23*s1*s5 + c1*s4*c5 + s1*c4*c5*s23); \\
\mathbf{J}_{26} &= 0; \\
\\
\mathbf{J}_{31} &= 0; \\
\mathbf{J}_{32} &= (c23*c5 - s23*c4*s5)*d6 + d4*c23 - a3*s23 - a2*s2; \\
\mathbf{J}_{33} &= (c23*c5 - s23*c4*s5)*d6 + d4*c23 - a3*s23; \\
\mathbf{J}_{34} &= -s5*s4*c23*d6; \\
\mathbf{J}_{35} &= (c23*c5*c4 - s5*s23)*d6; \\
\mathbf{J}_{36} &= 0; \\
\\
\mathbf{J}_{41} &= 0; \\
\mathbf{J}_{42} &= s1; \\
\mathbf{J}_{43} &= s1; \\
\mathbf{J}_{44} &= c1*c23; \\
\mathbf{J}_{45} &= c1*s23*s4 + s1*c4; \\
\mathbf{J}_{46} &= (-c1*s23*c4 + s1*s4)*s5 + c1*c23*c5; \\
\\
\mathbf{J}_{51} &= 0; \\
\mathbf{J}_{52} &= -c1; \\
\mathbf{J}_{53} &= -c1; \\
\mathbf{J}_{54} &= s1*c23; \\
\mathbf{J}_{55} &= s1*s23*s4 - c1*c4; \\
\mathbf{J}_{56} &= -((s1*s23*c4 + c1*s4)*s5 - s1*c23*c5); \\
\\
\mathbf{J}_{61} &= 1; \\
\mathbf{J}_{62} &= 0; \\
\mathbf{J}_{63} &= 0; \\
\mathbf{J}_{64} &= s23; \\
\mathbf{J}_{65} &= -c23*s4; \\
\mathbf{J}_{66} &= c23*c4*s5 + s23*c5;
\end{aligned}$$

**Note:** These calculations were made in *MatLab* using the symbolic Toolbox.

---

**Figure 2.12** Linear and angular velocities, jacobian matrices  ${}^0_3J$ ,  ${}^0_4J$  and  ${}^0_6J$

## 2.4 Singularities

If the objective is to use the differential kinematics equation (2.28) for simplicity and efficiency, then it's necessary to deal with the singularities of the jacobian. The differential kinematics equation maps the vector of joint velocities  $\dot{q} = [\dot{q}_1 \ \dot{q}_2 \ \dot{q}_3 \ \dot{q}_4 \ \dot{q}_5 \ \dot{q}_6]^T$  with the *end-effector* twist vector  $\mathbf{v} = \begin{bmatrix} v^T & w^T \end{bmatrix}^T$ . This mapping is seriously affected when the jacobian is rank-deficient (kinematics

singularities), because in those situations the mobility of the robot is reduced, the inverse kinematics may show infinite solutions, and (because the jacobian determinant may take very small values near singularities) small task space velocities may cause very large joint velocities [2]. So, to control the robot manipulator it is necessary to find all singular configurations and design a scheme to identify a singular configuration approach.

In order to find all the singular points of the ABB IRB 1400 anthropomorphic industrial robot, which has a very simple kinematic structure, a scheme will be used that separates the arm singularities and the wrist singularities. By dividing the jacobian into four 3x3 blocks it can then be expressed as

$${}^0_6J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \quad (2.65)$$

Now, looking to all the elements of  $J_{12}$  (Figure 2.12) it is clear that  $\det(J_{12})$  vanishes making  $d_6=0$ . That is equivalent to choosing the origin of the *end-effector* frame coincident with the origin of axis 4 and 5, i.e., making  $\mathbf{p}_w = \mathbf{p}$ . Since singularities are a characteristic of the robot structure and do not depend on the frames chosen to describe kinematically the robot, this procedure is allowed. It's possible then to write

$$\det(J) = \det(J_{11}) * \det(J_{22}) \quad (2.66)$$

The robot's singular configurations are the ones that make  $\det(J) = 0$  which means from (2.66)

$$\det(J_{11}) = 0 \quad \text{or} \quad \det(J_{22}) = 0 \quad (2.67)$$

Solving the first equation leads to the so called *arm singularities* and solving the second leads to the *wrist singularities*.

#### Wrist Singularities

The wrist singularities can be found just by analyzing the structure of  $\det(J_{22})$ :

$$\det(J_{22}) = \det \begin{pmatrix} z_4 & z_5 & z_6 \end{pmatrix} = \det \begin{pmatrix} c_1c_{23} & c_1s_{23}s_4 - c_1c_4 & (s_1s_4 - c_1s_{23}c_5)s_5 + c_1c_{23}c_5 \\ s_1c_{23} & s_1s_{23}s_4 - c_1c_4 & -(s_1s_{23}c_4 + c_1s_4)s_5 + s_1c_{23}c_5 \\ s_{23} & -c_{23}s_4 & c_{23}c_4s_5 + s_{23}c_5 \end{pmatrix} \quad (2.68)$$

The above determinant is non-null if the column vectors of  $J_{22}$  (which correspond to  $z_4$ ,  $z_5$ , and  $z_6$ ) are linearly independent, i.e., the singular configurations are the ones that make at least two of them linearly dependent. Now, vectors  $z_4$  and  $z_5$  are linearly independent in all configurations, and the same occurs between  $z_5$  and  $z_6$ . This conclusion is easy to understand looking to (2.68) and/or remembering that  $z_4$

is perpendicular to  $z_5$ , and  $z_5$  is perpendicular to  $z_6$  in all possible robot configurations. A singular configuration appears when  $z_4$  and  $z_6$  are linearly dependent, i.e., when those axis align with each other, which means  $s_5=0$  from (2.68). Consequently the wrist singular configurations occur when,

$$\theta_5 = 0 \quad \text{or} \quad \theta_5 = \pi \quad (2.69)$$

The second condition ( $\theta_5 = \pi$ ) is out of joint 5 work range, and because of that is of no interest, i.e., the wrist singularities will occur whenever  $\theta_5 = 0$ .

#### Arm Singularities

The arm singularities occur when  $\det(J_{11}) = 0$  making again  $\mathbf{p} = \mathbf{p}_w \Rightarrow d_6 = 0$ , i.e., when

$$\det \begin{pmatrix} (a_2 s_2 - a_1 + a_3 s_{23} - d_4 c_{23}) s_1 & -(a_2 c_2 + d_4 s_{23} - a_3 c_{23}) c_1 & -(d_4 s_{23} + a_3 c_{23}) c_1 \\ (a_1 - a_2 s_2 + d_4 c_{23} - a_3 s_{23}) c_1 & -(a_2 c_2 + d_4 s_{23} + a_3 c_{23}) c_1 & -(d_4 s_{23} + a_3 c_{23}) c_1 \\ 0 & d_4 c_{23} - a_3 s_{23} - a_2 s_2 & d_4 c_{23} - a_3 s_{23} \end{pmatrix} = 0 \quad (2.70)$$

Solving (2.70) gives

$$-a_2 (d_4 c_3 - a_3 s_3) (a_3 s_{23} - d_4 c_{23} + a_2 s_2 - a_1) = 0 \quad (2.71)$$

which leads to the following conditions:

$$-a_3 s_3 + d_4 c_3 = 0$$

and/or

$$a_3 s_{23} - d_4 c_{23} + a_2 s_2 - a_1 = 0 \quad (2.72)$$

The first condition leads to  $\theta_3 = \arctg \left( \frac{d_4}{a_3} \right)$ . The elbow is completely stretched out

and the robot manipulator is in the so called *elbow singularity*. This value of  $\theta_3$  is out of joint 3's work range, so it corresponds to a non-reachable configuration, and because of that is of no interest.

The second condition corresponds to configurations in which the origin of the wrist (origin of axis 4) lies in the axis of joint 1, i.e., lies in  $z_1$  (note that  $z_1$  is coincident with  $z_0$ ). In those configurations, the position of the wrist cannot be changed by rotation of the remaining free joint  $\theta_1$  (remember that an anthropomorphic manipulator with a spherical wrist uses the anthropomorphic arm to position the spherical wrist, which is then used to set the orientation of the *end-effector*). The manipulator is in the so called *shoulder singularity*.

In conclusion, the arm singularities of the ABB IRB 1400 industrial robot are confined to all the configurations that correspond to a shoulder singularity, i.e., to configurations where  $a_3s_{23} - d_4c_{23} + a_2s_2 - a_1 = 0$ .

#### 2.4.1 Brief Overview: Singularity Approach

As already mentioned, the solutions of the inverse kinematics problem can be computed from

$$\dot{q} = J^{-1}(\theta)V \quad (2.73)$$

solving (2.28) in order to  $\dot{q}$ . With this approach it's possible to compute the joint trajectories  $(q, \dot{q})$ , initially defined in terms of the *end-effector* wrist vector  $V$  and of the initial position/orientation. In fact, if  $q(0)$  is known it's possible to calculate:

$$\dot{q}(t) \text{ from: } \quad \dot{q}(t) = J^{-1}(\theta)V(t)$$

and

$$q(t) \text{ from: } \quad q(t) = q(0) + \int_0^t \dot{q}(\alpha) d\alpha \quad (2.74)$$

Nevertheless, this is only possible if the jacobian is full rank, i.e., if the robot manipulator is out of singular configurations where the jacobian contains linearly dependent column vectors. In the neighborhood of a singularity, the jacobian inverse may take very high values, due to small values of  $\det(J)$ , i.e., in the neighborhood of a singular point small values of the velocity in the task space ( $V$ ) can lead to very high values of the velocity in the joint space ( $\dot{q}$ ).

The *singular value decomposition* (SVD) of the jacobian [3,8-10] is maybe the most general way to analyze what happens in the neighborhood of a singular point; also it is the only general reliable method to numerically determine the rank of the jacobian and the closeness to a singular point. With the insight given by the SVD of the jacobian, a *Damped Least-Square* scheme [9] can be optimized to be used in near-singular configurations. The *Damped Least-Square* (DLS) scheme trades-off accuracy of the inverse kinematics solutions with feasibility of those solutions: this trade-off is regulated by the damping factor  $\xi$ . To see how this works, let's define the DLS inverse jacobian by rewriting (2.28) in the form

$$(JJ^T + \xi^2 I)\dot{q} = J^T V \quad (2.75)$$

where  $\xi$  is the so-called damping factor. Solving (2.75) in order to  $\dot{q}$  gives

$$\dot{q} = (JJ^T + \xi^2 I)^{-1} J^T V = J_{\text{dis}}^{-1} V \quad (2.76)$$

with  $J_{\text{dis}}$  being the *damped least-square* jacobian inverse. The solutions of (2.76) are the ones that minimize the following cost function [2,9,11]:

$$g(\dot{q}) = \frac{1}{2} (V - J\dot{q})^T (V - J\dot{q}) + \frac{1}{2} \xi^2 \dot{q}^T \dot{q} \quad (2.77)$$

resulting from the condition

$$\min_{\dot{q}} \left( \|V - J\dot{q}\|^2 + \xi^2 \|\dot{q}\|^2 \right) \quad (2.78)$$

The solutions are a trade-off between the *least-square* condition and the minimum norm condition. It is very important to select carefully the damping factor  $\xi$ : small values of  $\xi$  lead to accurate solutions but with low robustness to the singular or near-singular occurrences (= high degree of failure in singular or near-singular configurations), i.e., low robustness to the main reason to use the scheme. High values of  $\xi$  lead to feasible but awkward solutions.

To understand how to select the damping factor  $\xi$ , in the following the jacobian will be decomposed using the SVD technique. The SVD of the jacobian can be expressed as

$$J = U \Sigma V^T = \sum_1^6 \sigma_i u_i v_i^T \quad (2.79)$$

where  $\sigma_1 > \sigma_2 > \dots > \sigma_r > 0$  ( $r = \text{rank}(J)$ ) are the jacobian *singular values* (positive square roots of the eigenvalues of  $J^T J$ ),  $v_i$  (columns of the orthogonal matrix  $V$ ) are the so-called *right or input singular vectors* (orthonormal eigenvectors of  $J^T J$ ) and  $u_i$  (columns of the orthogonal matrix  $U$ ) are the so-called *left or output singular vectors* (orthonormal eigenvectors of  $J J^T$ ). The following properties hold:

$$\begin{aligned} R(J) &= \text{span} \{u_1, \dots, u_r\}^6 \\ N(J) &= \text{span} \{v_{r+1}, \dots, v_6\} \end{aligned}$$

The range of the jacobian  $R(J)$  is the set of all possible task velocities, those that could result from all possible joint velocities:  $R(J) = \{V \in \mathfrak{R}^6: V = J\dot{q} \text{ for all possible } \dot{q} \in \mathfrak{R}^6\}$ . The first  $r$  input singular vectors constitute a base of  $R(J)$ . So, if in a singularity the rank of the jacobian is reduced then one other effect of a singularity will be the decrease of  $\dim[R(J)]$  by eliminating a linear combination of

---

<sup>6</sup> The span of  $\{a_1, \dots, a_n\}$  is the set of the linear combinations of  $a_1, \dots, a_n$ .



task velocities from the space of feasible velocities, i.e., the reduction of the set of all possible task velocities.

The null space of the jacobian  $N(J)$  is the set of all the joint velocities that produce a null task velocity at the current configuration:  $N(J) = \{\dot{q} \in \mathfrak{R}^6: J\dot{q} = 0\}$ . The last  $(6-r)$  output singular vectors constitute a base of  $N(J)$ . So, in a singular configuration the dimension of  $N(J)$  is increased by adding a linear combination of joint velocities that produce a null task velocity.

Using the SVD of the jacobian (2.78) in the DLS form of the inverse kinematics (2.75) results in

$$\dot{q} = \sum_1^6 \frac{\sigma_i}{\sigma_i^2 + \xi^2} v_i u_i^T V \quad (2.80)$$

The following properties hold:

$$\begin{aligned} R(J_{lds}) &= R(J^\dagger)^7 = N^\perp(J)^8 = \text{span} \{u_1, \dots, u_r\} \\ N(J_{lds}) &= R(J^\dagger) = R^\perp(J)^9 = \text{span} \{v_{r+1}, \dots, v_6\} \end{aligned} \quad (2.81)$$

which means that the properties of the *damped least-squares* inverse solution are analogous to those of the pseudoinverse solution (remember that the inverse pseudoinverse solution gives a *least-square* solution with a minimum norm to equation (2.28)).

The damping factor has little influence on the components for which  $\sigma_i \gg \xi$  because in those situations

$$\frac{\sigma_i}{\sigma_i^2 + \xi^2} \approx \frac{1}{\sigma_i} \quad (2.82)$$

i.e., the solutions are similar to the pure *least-square* solutions.

Nevertheless, when a singularity is approached, the smallest singular value (the  $r$ -th singular value) tend's to zero, the associated component of the solution is driven to zero by the factor  $\frac{\sigma_i}{\xi^2}$  and the joint velocity associated with the near-degenerate

components of the commanded velocity  $V$  are progressively reduced, i.e., at a singular configuration, the joint velocity along  $v_r$  is removed (no longer remains in the null-space of the jacobian as in the pure *Least-Square* solution) and the task

---

<sup>7</sup>  $J^\dagger$  is the pseudoinverse jacobian.

<sup>8</sup> Orthogonal complement of the null space joint velocities.

<sup>9</sup> Orthogonal complement of the feasible space task velocities.

velocity along  $u_r$  becomes feasible. That is how the damping factor works; as a measure or indication of the degree of approximation between the damped and pure *least-square* solutions. Then a strategy [8], initially presented by [12], can be used to adjust  $\xi$  as a function of the closeness to the singularity. Based on the estimation of the smallest singular value of the jacobian, we can define a singular region and use the exact solution ( $\xi=0$ ) outside the region and a damped solution inside the region. In this case, a varying  $\xi$  should be used (increasing as we approach the singular point) to achieve better performance (as mentioned the damped solutions are different from the exact solutions). The damping factor  $\xi$  can then be defined using the following law modified from [9]

$$\xi^2 = \begin{cases} 0 & \hat{\sigma}_6 \geq \varepsilon \\ \left(1 - \left(\frac{\hat{\sigma}_6}{\varepsilon}\right)^{2\eta}\right) \xi_{\max}^2 & \hat{\sigma}_6 < \varepsilon \end{cases} \quad (2.83)$$

where  $\xi_{\max}^2$  and  $\eta$  are defined by the user to shape the solution to his needs,  $\varepsilon$  defines the size of the region and  $\hat{\sigma}_6$  is the estimate of the smallest singular value. The estimate is done using a recursive algorithm originally presented at [13] and later extended by [14] to estimate not only the smallest singular value but also the second smallest singular value. This procedure avoids estimation inaccuracy due to the cross of the two smallest singular values, when the manipulator approaches both the wrist and the shoulder singularity. The algorithm is as follows:

Suppose we have estimates of the two last input singular vectors  $\hat{v}_5$  and  $\hat{v}_6$  with

$$\begin{aligned} \hat{v}_5 &\approx v_5 \text{ and } \|\hat{v}_5\| = 1 \\ \hat{v}_6 &\approx v_6 \text{ and } \|\hat{v}_6\| = 1 \end{aligned} \quad (2.84)$$

The estimate  $\hat{v}_6$  is then use to compute  $\hat{v}'_6$  from

$$(J^T J + \xi^2 I) \hat{v}'_6 = \hat{v}_6 \quad (2.85)$$

Then the estimate  $\hat{\sigma}_6^2$  is computed from

$$\hat{\sigma}_6^2 = \frac{1}{\|\hat{v}'_6\|} - \xi^2 \quad (2.86)$$

and the initial estimate  $\hat{v}_6$  is updated using

$$\hat{v}_6 = \frac{\hat{v}'_6}{\|\hat{v}'_6\|} \quad (2.87)$$

The second smallest singular value is computed using the estimate  $\hat{v}_6$  from,

$$\left[ J^T J + \xi^2 I - (\hat{\sigma}_6^2 + \xi^2) \hat{v}_6 \hat{v}_6^T \right] \hat{v}'_5 = \hat{v}_5 \quad (2.88)$$

Then the estimate  $\hat{\sigma}_5^2$  is computed from

$$\hat{\sigma}_5^2 = \frac{1}{\|\hat{v}'_5\|} - \xi^2 \quad (2.89)$$

and finally the initial estimate  $\hat{v}_5$  is updated using

$$\hat{v}_5 = \frac{\hat{v}'_5}{\|\hat{v}'_5\|} \quad (2.90)$$

Special care should be taken with the numerical implementation of the DLS inverse kinematics solutions, to correct the numerical drift. Basically a feedback term can be used [2,9,15] by making

$$V = V_d + K \cdot e = V_d + K \begin{pmatrix} \mathbf{p}_d - \mathbf{p} \\ \frac{1}{2}(\mathbf{n} \times \mathbf{n}_d + \mathbf{s} \times \mathbf{s}_d + \mathbf{a} \times \mathbf{a}_d) \end{pmatrix} \quad (2.91)$$

where  $K$  is a positive definite diagonal  $6 \times 6$  matrix,  $\mathbf{p}_d$  and  $\mathbf{p}$  are the desired and actual position, and the orientation is defined in terms of the desired and actual  $(\mathbf{n}, \mathbf{s}, \mathbf{a})$  vectors of the *end-effector* frame.

Due to the increase of *end-effector* errors [11] in the neighborhood of a singularity by means of the near-degenerate components of *end-effector* velocity, the matrix  $K$  should be corrected using  $K = \rho K_0$ , where  $K_0$  is a diagonal constant matrix and  $\rho$  is the correcting factor. Now, inside a singular region we should use  $K=0$  because in some situations the resulting joint velocities can drive the manipulator to reach the joint limits, even if eventually the error will approach zero. When the manipulator is sufficiently away from a singularity, we should have  $\rho=1$ . So, generally we define  $\rho$  as

$$\rho = \begin{cases} 0 & \sigma_6 \leq \varepsilon \\ \frac{(\sigma_6 - \varepsilon)^2}{(n-1)^2 \varepsilon^2} & \varepsilon < \sigma_6 < n\varepsilon \\ 1 & \text{otherwise} \end{cases} \quad (2.92)$$

where  $n$  is defined by the user based on self-experience and on test results with a particular robotic manipulator setup.

## 2.5 Position Sensing

The IRB1400 uses resolvers [16-19] as position sensors. The drive unit used at this robot (manufactured by *ELMO AB* for *ABB Robotics*), includes a PM AC synchronous motor, both current feedback devices, a brake, and a brushless resolver, all assembled at factory ,i.e., they come in one piece [20].

A brushless resolver consists of a stator, a rotor and a rotary transformer. The stator and rotor windings are distributed in a way that the magnetic flux is distributed as a sine wave of the angle of rotation (perfect resolver). The output of a resolver is therefore an AC voltage in accordance with the angular position of the shaft. This type of position sensor is characterized by its high accuracy output, maintenance free brushless design, and immunity to noise, vibration, and shock. Other characteristics introduced by highly automated manufacturer production facilities include homogeneity in accuracy, transformation ratio, phase-shift, etc.

These characteristics significantly reduce major sources of error such as:

1. Amplitude imbalance due to different amplitudes of the resolver output signals
2. Imperfect quadrature due to phase-shift
3. Inductance harmonic error due to imperfect inductance profiles, i.e., the inductance profiles do not follow perfect sine wave as consequence of imperfect sinusoidal winding

Two types of resolvers can be considered (Figure 2.13): *Brushless Amplitude Output Resolvers* (BAOR) and *Brushless Phase-Shift Output Resolvers* (BPOR)<sup>10</sup>. Resolvers of type BAOR are excited by an AC voltage to the rotor winding and the output is obtained from the stator windings in the form sine and cosine voltages proportional to the rotation angle  $\theta$ . Resolvers of the type BPOR are excited by sine and cosine voltages to the stator windings and the output is obtained from the

---

<sup>10</sup> *Tamagawa Seiki Co. LTD.* names these resolvers as BRX and BRT, respectively.

rotor winding in the form of a sine voltage with phase-shifted in proportion to the rotation angle  $\theta$ .

The IRB 1400 uses BAOR type resolvers from the Japanese manufacturer Tamagawa Seiki Co. LTD. [19,20].

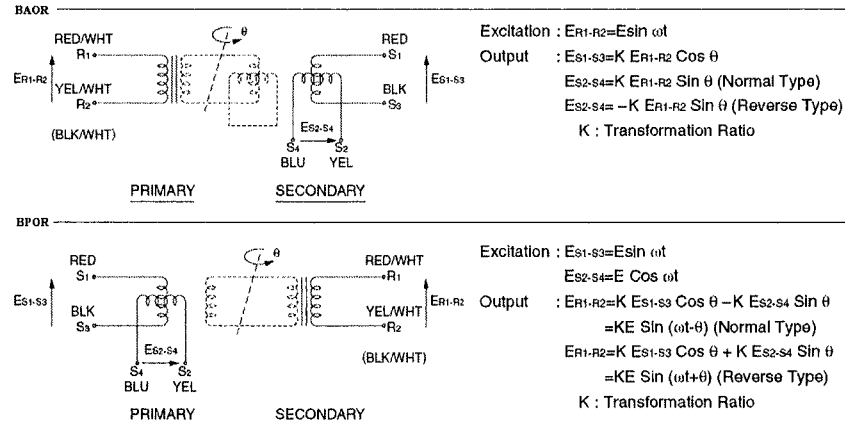


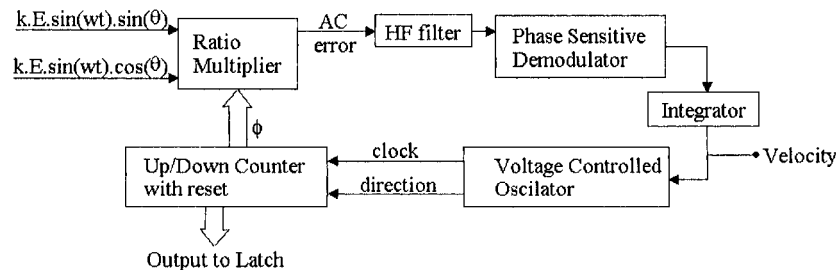
Figure 2.13 Types of resolvers

The use of a resolver implies the availability of a *resolver to digital converter* (RDC) and processing circuit [21-23]. The RDC is used to track and convert resolver signals to a digital parallel binary word, generally using a ratiometric tracking conversion method that improves noise immunity and tolerance to lead length (important when the converter is remote from the resolver). The RDC circuit uses an RDC along with the necessary interface and signal conditioning circuitry. Because noise can degrade significantly the accuracy of the measurement, special care must be taken with the driving lines from and to the resolvers: the use of shielded twisted pair cabling and isolation amplifiers may be needed.

The basic functional diagram of an RDC is presented in Figure 2.14, where it is used data relative to the *Analog Devices* RDC model AD2S80A. The converter works as a type II closed-loop system with the angle  $\phi$  as a control variable (this angle is the current converter estimate of the angle  $\theta$ ).

Generally, the converter’s functioning can be described as follows: First the inputs (resolver outputs  $E_{S2-S4}$  and  $E_{S1-S3}$ ) are multiplied by  $\cos(\phi)$  and  $\sin(\phi)$ , respectively, at the *ratio multiplier*. Then the difference between the signals is computed giving the ratio multiplier output AC error signal  $E_{ac} = A_1 \cdot K \cdot E \cdot \sin(\theta - \phi) \cdot \sin(\omega t)$ , where  $A_1$  is the ratio multiplier gain (fixed at 14.5 for AD2S80A). Second, this error signal is synchronously demodulated at the *phase sensitive demodulator* (PSD), using the resolver excitation frequency as a demodulation reference, leaving the error signal

$E_{PSD} = A_1 K.E \sin(\theta - \phi)$ . The output of the demodulator is a DC voltage proportional to the RMS value of the demodulator input:  $\frac{\pm\sqrt{2}}{\pi} * \text{Demodulator\_Input}_{RMS}$  (for sinusoidal input signals in phase or antiphase with the reference signal). Before entering the PSD, the signal passes over an *HF filter* (with components selected by the user) to remove any DC offset voltage. Then the PSD output passes through the *integrator* (with components selected by the user), whose output signal (proportional to the velocity of the resolver) is fed to the *voltage controlled oscillator* (VCO). The VCO integrates the velocity signal and compares the resulting signal with the minimum DC voltage resolution (uses two comparators for positive and negative voltages, meaning rotation in the positive direction or in the negative direction) and updates the up/down counter by producing the counter clock and direction signal. The value of the internal latch used to interface with the user is also updated with the counter value. An RDC works similarly to a *successive approximation type analog to digital converter*.



**Figure 2.14** Resolver to digital converter basic functional diagram

The RDC returned digital value is generally a 12, 14, or 16-bit binary number containing the actual rotation angle. This angle should be mapped to the robot's joint space. For that, the following guidelines should be used:

1. Choose an angle data format, i.e., degrees, radians
2. Account for the resolver offset<sup>11</sup>, i.e., the resolver reading when the manipulator is in the home position. At that point, we should have  $number\_of\_rotations = 0$  and  $actual\_angle = 0$

A complete RDC circuit implementation should also save the total number of rotations in an 8-bit *up-down* counter/register. In essence, the circuit should give the rotation angle of the motor in the actual rotation and the total number of rotations already performed.

<sup>11</sup> Usually these values are measured by the robot manufacturer and printed on the robot or in the robot documentation.

## 2.6 Actuators: Motors

Generally the actuators used to move the joints of any industrial robot are motors, usually DC permanent magnet (PM) motors or AC PM motors. Other motors can be used, including pneumatic or hydraulic servo motors. The IRB 1400 uses three-phase synchronous AC PM motors, with six poles (axes 1-3) and four poles (axes 4-6), manufactured by *Elmo AB* – Sweden.

The three-phase synchronous AC PM motor rotating magnetic field is obtained by making a three-phase current to flow in the stator coil (Figure 2.15), which has a sinusoidal distribution. So, a brushless sine wave PM AC synchronous motor is obviously not mechanically commutated (there are no brushes) but instead the commutation is done by acting on the three-phase current signals. Nevertheless, the commutation position of the motor should be retained, i.e., the resolver reading when the motor is at the electrical home position (electrical  $0^\circ$  position) - this value is called the commutation offset (COMMOFF).

The usual procedure to find the commutation offsets is as follows:

1. Turn the motor to the commutating position by feeding a positive constant current to the motor
2. Feed the resolver with the necessary excitation signal (4kHz and 5 V<sub>rms</sub> for IRB 1400 drives)
3. Adjust the resolver to  $+90^\circ$  ( $\pm 0,5^\circ$ ), i.e, turn to the maximum value on coil Y of the resolver with the same phase as the 5V feeding signal. At that point we should have:

$$\underline{\text{Voltage across coil X}} = 0V$$

and

$$\underline{\text{Voltage across coil Y}} = \text{input voltage} * \text{transformation ratio}$$

The value of the rotation angle (90 degrees) is the commutation offset. This procedure is used with the IRB 1400 drives, so that is why the COMMOFFS are constant for all drives (1.570800 radians). For some older robots, like the ABB IRB 2000 (up to model M90), the motor and the resolver are separate parts, assembled together by the manufacturer without following the above referred procedure. So, the COMMOFFS are different for all drives. The values are obtained at factory and printed on the robot or in the documentation; nevertheless, these values can be updated using the robot controller.

A full description of a three-phase synchronous sine wave PM motor can be found in:

1. *Design of Brushless Permanent-Magnet Motors*, Herdershot Jr., Magna Physics Publishing and Clarendon Press, Oxford, 1995, Chapters 6 and 7
2. *Electric Drives and their Controls*, R.M. Crowder, Clarendon Press, Oxford, 1995, Chapter 5, section 5.3

Nevertheless, a brief overview is presented here.

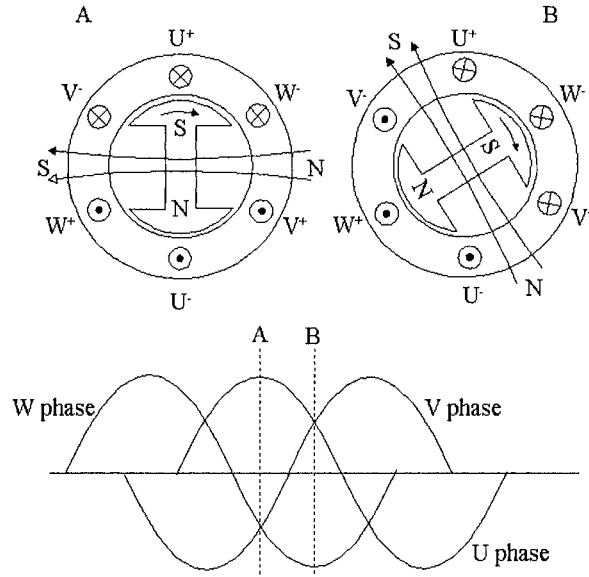


Figure 2.15 Three-phase synchronous motors and current signals

Considering  $\beta$  as the angle between rotor magnet north axis and the stator windings axis, it can be shown [17] that the motor torque is

$$T \propto \sin(\beta) \tag{2.93}$$

Consequently, the angle  $\beta$  must be kept at  $90^\circ$  in order to maximize the torque, which is done by phasing the current waveforms relative to the actual rotor position. To ensure that the ampere-conductor distribution remains in synchronism with the rotor's magnetic field, the stator supply frequency ( $f$ ) must be equal to the rotor angular velocity ( $w_s$ ),  $w_s = 2\pi.f$ , which is related to the mechanical angular velocity of the motor ( $w_m$ ) by  $w_m = w_s/p$ , where  $p$  is the number of the motor pole pairs. In order to keep the torque angle constant, i.e., to keep the ampere-distribution north axis in synchronism with the rotor north axis (displaced by  $90^\circ$ ), a high-performance and precise sensor should be used (generally a resolver).

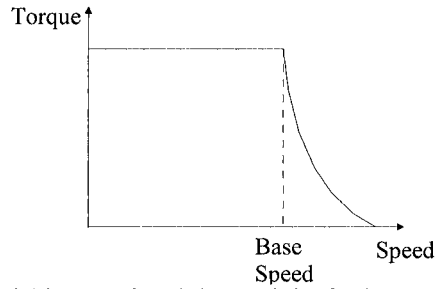
With this type of control action the motor follows the equation

$$\text{Torque} = \text{Flux} * \text{Current} \tag{2.94}$$

For this type of motors, the flux is constant, sinusoidally distributed in space, and the generated EMF varies sinusoidally in each phase. The overall torque-speed characteristic is presented in Figure 2.16. The maximum torque can be maintained



up to the base speed. After that, it is still possible to increase the velocity by changing  $\beta$  but the motor enters the field-weakening mode and any increase in speed is done at the expense of the peak torque.



**Figure 2.16** Torque-Speed characteristic of a sine wave motor

The “natural” relations for the back-EMF ( $E$ ) and for the torque ( $T$ ), used for a DC square wave motor still hold for a sine wave motor, i.e.,

$$\begin{aligned} T &= k_t * I \\ E &= k_e * w_m \end{aligned} \tag{2.95}$$

but now with  $\frac{k_t}{k_e} = \frac{\sqrt{3}}{2} \neq 1$ .

The torque constant ( $k_t$ ) and the back-EMF constant ( $k_e$ ) can be measured using the following equations:

$$k_e = \frac{\hat{e}_{LL}}{w_m} \text{ (V-s/rad)} \tag{2.96}$$

where  $\hat{e}_{LL}$  is the peak line-line voltage and  $w_m$  is the mechanical angular velocity.

$$k_t = \frac{T}{\hat{i}} \text{ (Nm)} \tag{2.97}$$

where  $\hat{i}$  is the peak line current when the motor is in normal operation, measured using a current sensor connected to measure the phase current directly and then displayed in an oscilloscope.

It is also possible to write

$$\begin{aligned}
 T \cdot \omega_m &= k_t \cdot \hat{i} \cdot \frac{\hat{e}_{LL}}{k_e} = \frac{\sqrt{3}}{2} \cdot \hat{e}_{LL} \cdot \hat{i} = \frac{\sqrt{3}}{2} \cdot \sqrt{2} \cdot E_{RMS} \cdot \sqrt{2} \cdot I_{RMS} \\
 &= \sqrt{3} \cdot E_{RMS} \cdot I_{RMS} = \text{Electrical--Mechanical Power Conversion}
 \end{aligned}
 \tag{2.98}$$

and,

$$T = \frac{\sqrt{3} \cdot E_{RMS} \cdot I_{RMS}}{\omega_m} = k_t \cdot I_{RMS} \Rightarrow k_t = \frac{\sqrt{3} \cdot E_{RMS}}{2\pi \cdot \frac{\text{VolRPM}}{60}}
 \tag{2.99}$$

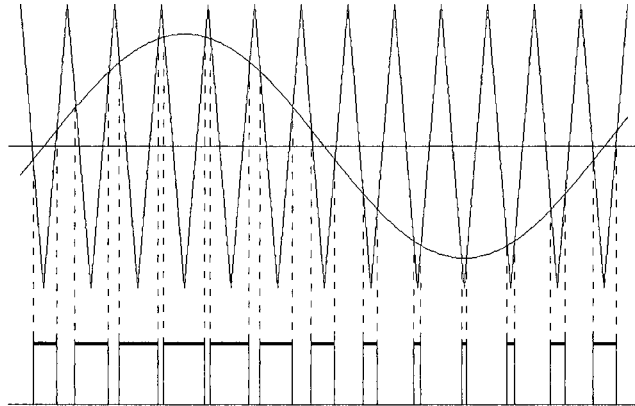
### 2.6.1 Motor Drive System

In this section, the main circuits necessary to drive a three-phase AC synchronous PM motor are briefly presented. As already mentioned, a brushless AC PM motor requires alternating sine wave phase currents, because the motor is designed to generate sinusoidal back-EMF. The power electronic control circuit is very simple and uses some control strategy<sup>12</sup> to achieve torque, smooth speed, and accurate control, keeping the current to a safe value. In order to obtain sine wave phase currents, the power supply (DC voltage) must be switched on and off at high frequency, under the control of a current regulator that forces the power transistors to switch on and off in a way that the average current is a sine wave. Basically, the sine wave reference signals could just be applied directly to the power transistors, after appropriate power amplification. However, that means using the power transistors in the proportional or linear region, which will increase the operating temperature due to the high power loss. The power loss is reduced by switching the transistors on and off by comparing the sine wave reference with a high frequency triangular carrier wave (PWM - *pulse width modulation* circuit). The frequency and amplitude of the triangular wave are kept constant. The comparator switches on the transistors when the values of the reference sine wave exceed those of the triangular wave; and switches them off when the inverse situation occurs (Figure 2.17). The duty ratio is then increased and decreased by the sine wave, centered by 50%. This procedure leads to a average sine wave output, because the output of the inverter feeding the power transistors is 0V when the duty ratio is 50%.

Special care should be taken in selecting the carrier frequency, because the power loss increases with increasing frequency and the motor speed response decreases with decreasing frequency. Torque and current ripples appear more frequently at higher frequencies as well.

---

<sup>12</sup> A set of rules that determine when the power transistors are switched on and off

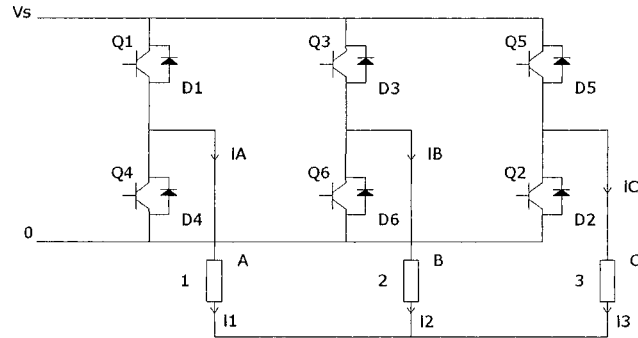


**Figure 2.17** PWM basic functioning

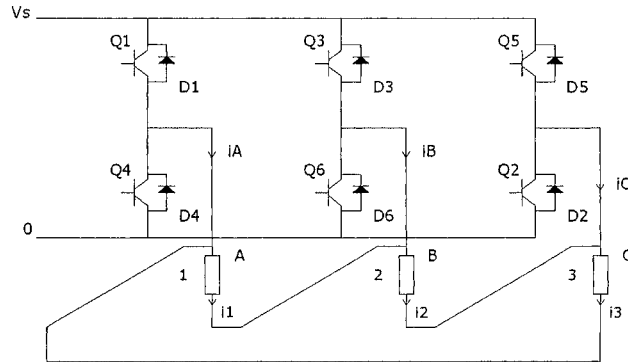
The basic power electronic circuit to control a sine wave three-phase AC PM motor is the full-bridge circuit. The transistors used in the circuit must have very low turn-on and turn-off switching times (of the order of nanoseconds) and some other properties summarized as follows:

1. Zero on-state forward voltage drop, to minimize losses and maximize available “voltage” to force current into the motor
2. Zero leakage current in the off state, to minimize losses because a power transistor usually has high voltages across it when it is off, so even a small leakage current can produce high losses in the transistor’s off state
3. High forward-blocking capability that should be higher than the supply voltage by a safety margin (usually 30%). The reverse-blocking capability is generally a margin of the forward-blocking, usually because the power transistors are reverse-protected by appropriately connected diodes
4. High  $dv/dt$  capability, because modern power transistors are MOS-gated, with capacitive input impedance at the gate, which make’s them sensitive to spurious turn-on when the gate is subjected to a high  $dv/dt$ . High  $dv/dt$  immunity is then desirable, but nevertheless a safe procedure is to drive the gate from a low impedance source/sink
5. High  $di/dt$  capability, to prevent current-crowding effects and second breakdown the  $di/dt$  capability must be high
6. High-speed switching, from transistors to minimize switching losses and also from the power diodes, because the commutation of inductive current from a transistor branch to a diode branch is the most important way to protect against destructive transient voltages

The full bridge circuit is presented in Figure 2.18 for two popular phase windings: eye and delta [17]. Figure 2.19 shows line current waveforms for three-phase sine wave motors, including transistor states and current paths.

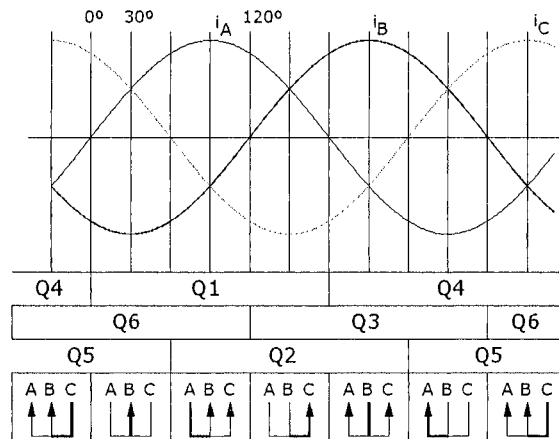


**Eye connected**



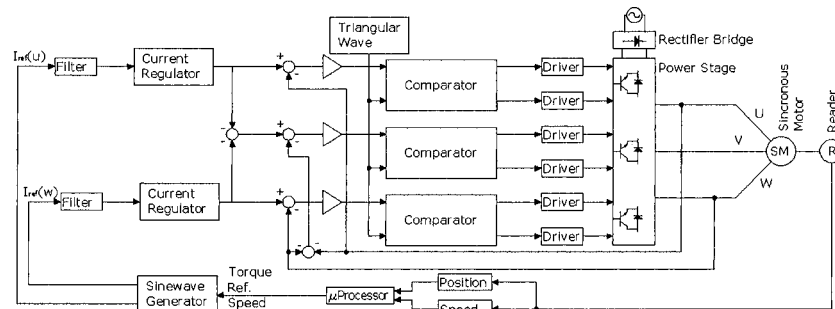
**Delta connected**

**Figure 2.18** Full bridge circuit for eye and delta connected windings



**Figure 2.19** Line current waveforms for a sine wave motor, including transistor states and current paths

A general control system for a sine wave three-phase brushless motor is presented in Figure 2.20: includes a PWM circuit, over current (due to motor stall or short circuits) protection, a filter to damp DAC steps, a current controller (usually a PI controller designed to drive the motor current to the desired value) and a sine wave generator. Synchronization is achieved by changing current references in accordance with motor position.



**Figure 2.20** Block diagram of a general control system for a brushless synchronous three-phase sine wave motor

## 2.7 Dynamics

Dynamics deals with mapping forces exerted on the robot's parts as well as with the motion of the robot, i.e., its joint positions, velocities, and accelerations. This mapping is achieved using a set of mathematical equations, based on some specified dynamic formulation that describes the dynamic behavior of the robot manipulator, i.e., its motion. Those sets of equations constitute the dynamic model of the robot manipulator. The dynamic model can be used to simulate and control the robot manipulator, i.e., the dynamic model provides the means to compute the joint positions, velocities, and accelerations starting from the joint torques (*direct dynamics*), and the means to compute the joint torques using the joint positions, velocities, and accelerations (*inverse dynamics*).

The dynamic model is obtained starting from well known physical laws like the *Newtonian* mechanics and the *Lagrange* mechanics [6,24]. Several different dynamic formulations for robot manipulators were developed: *Lagrange-Euler*, *Newton-Euler*, *D'Alembert*, ... [1-3,7]. Nevertheless, they are equivalent to each other because they define the same physical phenomenon, i.e., the dynamics of rigid bodies assembled together to constitute a robot. Obviously, the structure of the motion equations is much different because each formulation was developed to achieve different objectives such as computation efficiency, simplicity to analyze and/or to simulate the structure, etc.

In this section, the dynamic model of the ABB IRB 1400 industrial robot will be briefly summarized using the Newton-Euler dynamic formulation. In the process, the other dynamic formulations are presented and briefly discussed.

### 2.7.1 Inertia Tensor and Mass Distribution

The mass distribution of a rigid body may be characterized by its inertial mass, for the case of one degree of freedom motions, and by its first moment of inertia, for simple rotations, i.e., rotations about a single axis. If there is more than one axis of rotation, the above properties are no longer suitable to characterize the mass distribution of the moving rigid body [6,24]. This is the case of a rigid robot manipulator, which is made by a series of rigid bodies, whose motion is 3-dimensional and therefore an infinite number of rotation axes is possible. The concept of *inertia tensor* is used in this case, which can be considered as a generalization of the concept of moment of inertia. If  $\rho(x,y,z)$  is the mass density of a rigid body, then the inertia tensor may be defined as

$$I = \iiint \rho(r^2 \mathbf{1} - \mathbf{r}\mathbf{r}) dv \quad (2.100)$$

where  $\mathbf{1}$  is a unity tensor. The inertia tensor is a 3×3 matrix expressed in terms of some frame {A}

$${}^A I = \begin{bmatrix} I_{xx} & I_{yx} & I_{zx} \\ I_{xy} & I_{yy} & I_{zy} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \quad (2.101)$$

where the diagonal elements are the moments of inertia about the axes x, y and z of frame {A}

$$\begin{aligned} I_{xx} &= \iiint \rho(y^2 + z^2) dv \\ I_{yy} &= \iiint \rho(z^2 + x^2) dv \\ I_{zz} &= \iiint \rho(x^2 + y^2) dv \end{aligned} \quad (2.102)$$

and the other elements (non-diagonal) are the products of inertia

$$\begin{aligned} I_{xy} &= I_{yx} = -\iiint \rho xy dv \\ I_{yz} &= I_{zy} = -\iiint \rho yz dv \\ I_{zx} &= I_{xz} = -\iiint \rho zx dv \end{aligned} \quad (2.103)$$

### 2.7.1.1 Important Results [6]

Next some important results will be presented, considering that the frame associated to the rigid body is  $\{B\}$  and the inertial frame is  $\{A\}$ .

Suppose that  $I$  is the inertia tensor of the rigid body expressed in terms of some reference frame. The moment of inertia about any axis of rotation  $\mathbf{n}$  (different from any of the rigid body symmetry axes) with the same origin of the reference frame is

$$I_n = \mathbf{n}^T I \mathbf{n} \quad (2.104)$$

Extension of the Parallel Axis Theorem This theorem is used here to compute the inertia tensor variation with linear motions of the reference frame. Suppose that  $\{C\}$  is the frame associated with the rigid body center of mass,  $\{G\}$  is some frame obtained from  $\{C\}$  by linear motion, and  $CP$  is the position vector of the center of mass expressed in terms of  $\{G\}$ . Then

$$I_G = I_C + M ({}^C P^T {}^C P I_3 - {}^C P {}^C P^T) \quad (2.105)$$

where  ${}^C P = (x_c, y_c, z_c)^T$  and  $I_3$  is a  $3 \times 3$  identity matrix.

If the rigid body is rotating, the inertia tensor expressed in terms of  $\{A\}$   ${}^A I$  is also varying with time, but the inertia tensor expressed in terms of  $\{B\}$   ${}^B I$  remains constant (remember that  $\{B\}$  is the frame associated with the rigid body). If the inertia tensor  ${}^B I$  is known then

$${}^A I = {}^A H {}^B I {}^A H^T \quad (2.106)$$

where  ${}^A H$  is the transformation matrix from  $\{B\}$  to  $\{A\}$ .

The reference frame associated with each rigid body must be set to in a way that the products of inertia become null. The axes of that frame are named *primary axes* of the rigid body. The eigenvalues of the inertia tensor are the so-called rigid body *primary moments of inertia*. There are some systematic methods to compute the primary axis of inertia of any rigid body [6,24].

Any rigid body plane of symmetry is perpendicular to one primary axis.

Each symmetry axis of the rigid body is a primary axis. The plane of symmetry perpendicular to that axis is a *primary plane* associated with a degenerated primary moment of inertia.

### 2.7.2 Lagrange-Euler Formulation

Here we briefly introduce the *Lagrange-Euler* formulation. To use this formulation, it is required to develop equations for the robot manipulator's kinetic energy and potential energy. The kinetic energy of link (i) is given by

$$k_i = \frac{1}{2} m_i V_{C_i}^T V_{C_i} + \frac{1}{2} w_i^T \cdot {}^{C_i} I_i \cdot w_i \quad (2.107)$$

where the first term results from the linear velocity of the center of mass of link (i), and the second term is due to the angular velocity of the same link. The robot manipulator's total kinetic energy is then given by

$$K = \sum_{i=1}^6 k_i \quad (2.108)$$

The potential energy of link (i) may be written as

$$u_i = m_i \cdot {}^0 g^T \cdot {}^0 P_{C_i} + u_{ref_i} \quad (2.109)$$

where  ${}^0 g$  is the gravity acceleration vector,  ${}^0 P_{C_i}$  is the position vector of the center of mass of link (i) expressed in terms of frame {0} and  $u_{ref_i}$  is a constant that expresses the potential energy in terms of an arbitrary origin. The total potential energy of the robot manipulator is given by

$$U = \sum_{i=1}^6 u_i \quad (2.110)$$

The *Lagrange* equation is then

$$L = K - U \quad (2.111)$$

where K and U are obtained from (2.100) and (2.110). It follows that the motion equations of the robot manipulator can then be obtained using the Lagrange equation

$$\tau = \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} \quad (2.112)$$

where  $\tau$  is the joint torque vector.

Recently [4], recursive equations based on the *Lagrange-Euler* equations have been developed. The resulting equations are computationally more efficient. Nevertheless, the recursive nature destroys the equation's structure which is a



major drawback for the design and development of new control laws, and the *Newton-Euler* recursive equations remain the most efficient.

### 2.7.3 D'Alembert Formulation

This is basically a *Lagrange* dynamic formulation based on the *D'Alembert* principle. As mentioned before, the *Lagrange-Euler* formulation is simple but computationally inefficient, and the *Newton-Euler* formulation is compact with a recursive non-structured nature and is computationally very efficient. To obtain a recursive and computationally efficient set based on the Lagrange mechanics, a vector representation along with the use of rotation matrices is used to develop the kinetic and potential energy equations. The same procedure used in the *Lagrange-Euler* formulation is then used to compute the motion equations. This procedure is known as *D'Alembert* formulation, and is a generalization of the *Lagrange-Euler* and *Newton-Euler* formulations [7].

### 2.7.4 Newton-Euler Formulation

The *Newton-Euler* formulation will be used to obtain the dynamic equations of the ABB IRB 1400 industrial robot and in the process explained in some detail. We will also compare this to the other dynamic formulations.

If the joint positions, velocities, and accelerations of the robot manipulator are known, along with the kinematics and mass distribution, then we should be able to compute the required joint moments. On the other hand, if the joint torques is known, along with the inverse kinematics and the robot mass distribution, we should be able to compute the joint positions.

The *Newton-Euler* dynamic formulation is a set of recursive equations, divided in two groups: *forward recursive equations* and *inverse recursive equations*.

#### Forward Recursive Equations

This set of equations is used to compute (“*propagate*”) link velocities and accelerations from link to link, starting from link 1 (the first link).

#### Angular Acceleration Computation

Using equations (2.50) and (2.51) gives

$${}^{i+1}w_{i+1} = {}^{i+1}R_i \cdot {}^i\dot{w}_i + {}^{i+1}R_i \cdot {}^i w_i \times \dot{\theta}_{i+1} \cdot {}^{i+1}Z_{i+1} + \ddot{\theta}_{i+1} \cdot {}^{i+1}Z_{i+1} \quad (2.113)$$

for the angular acceleration of link (i+1) expressed in terms of (i+1).

Linear Acceleration Computation

Using equations (2.52) and (2.53) gives

$${}^{i+1}v_{i+1} = {}^{i+1}R_i \left[ {}^i\dot{w}_i \times {}^iP_{i+1} + {}^i w_i \times ({}^i w_i \times {}^iP_{i+1}) + {}^i\dot{v}_i \right] \quad (2.114)$$

for the linear acceleration of link (i+1) expressed in terms of (i+1).

Linear Acceleration Computation at the Link Center of Mass

Using again equations (20) and (25) results,

$${}^i\dot{v}_{C_i} = {}^i\dot{w}_i \times {}^iP_{C_i} + {}^i w_i \times ({}^i w_i \times {}^iP_{C_i}) + {}^i\dot{v}_i \quad (2.115)$$

where  $\{C_i\}$  is the reference frame associated with the center of mass of link (i), and having the same orientation of  $\{i\}$ .

Gravity effects

The gravity effects can be included in the above equations by making

$${}^0\dot{v}_0 = G \quad (2.116)$$

where  $G = \{g_x, g_y, g_z\}^T$  is the gravity acceleration vector with  $|G| = 9,8062 \text{ m/s}^2$ . This is equivalent to consider that the robot manipulator has a linear acceleration of one G, pointing up, which produces the same effect on the robot links as the gravity acceleration.

Using the above equations (2.113)-(2.115), the *Newton* equation (2<sup>nd</sup> law) and the *Euler* equation, it's possible to compute the total force and moment at the center of mass of each link:

$${}^{i+1}F_{i+1} = m_{i+1} \cdot {}^{i+1}\dot{v}_{C_{i+1}} \quad (2.117)$$

$${}^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1} \cdot {}^{i+1}\dot{w}_{i+1} + {}^{i+1}w_{i+1} \times {}^{C_{i+1}}I_{i+1} \cdot {}^{i+1}w_{i+1} \quad (2.118)$$

Note:

Newton Equation (2<sup>nd</sup> law) - The total force applied to a rigid body of mass m and centre of mass acceleration  $\dot{v}_C$ , is given by  $F = m \cdot \dot{v}_C$ .

Euler Equation - Consider a rigid body of mass m, angular velocity w, and angular acceleration  $\dot{w}$ . The total moment N starting the body in motion is given by  $N = {}^C I \dot{w} + w \times {}^C I w$ , where  ${}^C I$  is the rigid body inertia tensor expressed in terms of the reference frame associated with the body's center of mass.

### Backward Recursive Equations

This set of equations is used to compute (“propagate”) link forces and moments from link to link, starting at the last link.

### Computation of Links Forces and Moments

Taking

$f_i$  = force applied at link (i) by link (i-1);  
 $n_i$  = moment in link (i) due to link (i-1);

the force balancing on link (i) can be expressed as

$${}^i F_i = f_i - {}^{i+1}R_i \cdot {}^{i+1} f_{i+1} \quad (2.119)$$

and the moment balancing in the center of mass of link (i) can be expressed as

$${}^i N_i = n_i - {}^i n_{i+1} + (-{}^i P_{C_i}) \times f_i - ({}^i P_{i+1} - {}^i P_{C_i}) \times f_{i+1} \quad (2.120)$$

Using (2.119) in (2.120) gives

$${}^i N_i = n_i - {}^{i+1}R_i \cdot {}^{i+1} n_{i+1} + {}^i P_{C_i} \times f_i - {}^i P_{i+1} \times {}^{i+1}R_i \cdot f_{i+1} \quad (2.121)$$

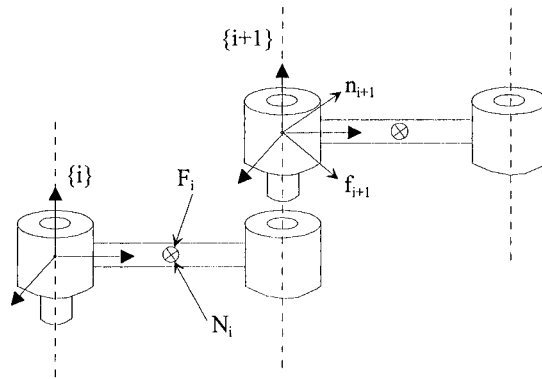


Figure 2.21 Forces and torques applied to the joints

Rewriting (2.119) and (2.121) in a way that their recursive nature becomes more evident results in

$${}^i f_i = {}^{i+1}R_i \cdot {}^{i+1} f_{i+1} + {}^i F_i \quad (2.122)$$

$${}^i n_i = {}^i N_{i+1} + {}^{i+1}R_i \cdot {}^{i+1} n_{i+1} + {}^i P_{C_i} \times f_i + {}^i P_{i+1} \times {}^{i+1}R_i \cdot f_{i+1} \quad (2.123)$$

To obtain the joint moments we just need to project over the Z axis the already computed moment  ${}^i n_i$ , i.e.,

$$\tau_i = {}^i n_i^T \cdot {}^i Z_i \quad (2.124)$$

#### Contact Forces

The contact forces and moments (contact wrench) can be included in the model by putting,

$$\begin{pmatrix} {}^{N+1} f_{N+1} \\ {}^{N+1} n_{N+1} \end{pmatrix} = \text{Contact wrench} \neq 0 \quad (2.125)$$

where N is the number of degrees of freedom of the robot manipulator.

### 2.7.5 Dynamic Parameters

There is a number of parameters that are needed to compute the dynamic model (dynamic parameters). The minimum set of parameters is called the base dynamic parameters, and its identification can reduce significantly the computational load of the dynamic model (by 50%). If we take a closer look at the equations developed for the kinematics energy and for the potential energy of link (i), it is easy to verify that they are linear with respect to some dynamic parameters: the link mass, the six elements of the link inertia tensor, and the three components of the link's first moments of inertia. Some other dynamic parameters must also be included, namely the ones related with joint actuation. The joint torque is given by

$$\tau = \tau_m + \tau_v + \tau_f + \tau_g + \tau_\alpha + \tau_e \quad (2.126)$$

where  $\tau_m = M(\theta)\ddot{\theta}$  is the torque due to the inertia of the robot manipulator,  $\tau_v$  is the torque due to the centrifugal and coriolis forces,  $\tau_f$  is the torque due to the friction forces,  $\tau_g$  is the torque due to the gravity force,  $\tau_\alpha$  is the torque resulting from non-modeled forces and  $\tau_e$  is the torque due to external contact forces.

Now,  $\tau_m$  can be written as  $\tau_m = \tau_{mr} + \tau_{mm}$ , where  $\tau_{mr}$  is the torque due to the robot manipulator inertia (not including the motor drive) and  $\tau_{mm}$  is the torque due to the motor inertia itself. We may express  $\tau_{mm}$  as

$$\tau_{mm} = I_m \cdot \ddot{\theta} = \begin{bmatrix} I_{m_1} & 0 & \dots & 0 \\ 0 & I_{m_2} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & I_{m_n} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \dots \\ \ddot{\theta}_n \end{bmatrix} \quad (2.127)$$

where  $I_m$  is the rotor's moment of inertia and  $n$  is the number of degrees of freedom.

The friction torque may be given by

$$\tau_f = F_s \cdot \text{sgn}(\dot{\theta}) + F_v \cdot \dot{\theta} = \begin{bmatrix} F_{s_1} & 0 & \dots & 0 \\ 0 & F_{s_2} & \dots & \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & F_{s_n} \end{bmatrix} \cdot \text{sgn} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dots \\ \dot{\theta}_n \end{bmatrix} + \begin{bmatrix} F_{v_1} & 0 & \dots & 0 \\ 0 & F_{v_2} & \dots & \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & F_{v_n} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dots \\ \dot{\theta}_n \end{bmatrix} \quad (2.128)$$

where the first term refers to the coulomb friction and the second to the viscous friction.

In conclusion,  $I_{m_i}$ ,  $F_{s_i}$  and  $F_{v_i}$  are also dynamic parameters to take into account, i.e., the all number of dynamic parameters is thirteen:

$$\pi = (I_{xx_i} \quad I_{yy_i} \quad I_{zz_i} \quad I_{xy_i} \quad I_{xz_i} \quad I_{yz_i} \quad m_i r_{ix} \quad m_i r_{iy} \quad m_i r_{iz} \quad m_i \quad I_{m_i} \quad F_{s_i} \quad F_{v_i}) \quad (2.129)$$

The basic Newton-Euler recursive algorithm resumed in the following form:

#### Forward recursive equations

##### Initial conditions

$${}^0 w_0 = 0; \quad {}^0 \dot{w}_0 = 0; \quad {}^0 \dot{v}_0 = {}^0 \ddot{p}_0 = {}^0 \hat{g} = (0 \quad 0 \quad g)^T, \quad \text{with } g = -9,8062 \text{ m/s}^2.$$

For  $i = 1$  to 5,

$${}^{i+1} w_{i+1} = {}^{i+1} R^i \dot{w}_i + {}^{i+1} R^i w_i \times \dot{\theta}_{i+1} \cdot {}^{i+1} Z_{i+1} + \ddot{\theta}_{i+1} \cdot {}^{i+1} Z_{i+1}$$

$${}^{i+1} v_{i+1} = {}^{i+1} R^i \left[ \dot{w}_i \times {}^i P_{i+1} + w_i \times ({}^i w_i \times {}^i P_{i+1}) + \dot{v}_i \right]$$

$${}^i \dot{v}_{C_i} = {}^i \dot{w}_i \times {}^i P_{C_i} + w_i \times ({}^i w_i \times {}^i P_{C_i}) + \dot{v}_i$$

$${}^{i+1} F_{i+1} = m_{i+1} \cdot {}^{i+1} \dot{v}_{C_{i+1}}$$

$${}^{i+1} N_{i+1} = {}^{C_{i+1}} I_{i+1} \cdot {}^{i+1} \dot{w}_{i+1} + {}^{i+1} w_{i+1} \times {}^{C_{i+1}} I_{i+1} \cdot {}^{i+1} w_{i+1}$$

#### Backward recursive equations

##### Initial conditions

$$\text{End-effector wrench} = \begin{pmatrix} {}^{N+1} f_{N+1} \\ {}^{N+1} n_{N+1} \end{pmatrix}$$

For  $i = 6$  to 1,

$${}^i f_i = {}^i R^{i+1} \cdot {}^{i+1} f_{i+1} + {}^i F_i$$

$${}^i n_i = {}^i N_{i+1} {}^i R. {}^{i+1} n_{i+1} + {}^i P_{C_i} \times {}^i F_i + {}^i P_{i+1} \times {}^{i+1} R. {}^{i+1} f_{i+1}$$

$$\tau_i = {}^i n_i^T \cdot {}^i Z_i$$

The generalized force at joint (i) is then

$$\mu_i = {}^i n_i^T \cdot Z_i + I_{m_i} \ddot{\theta}_i + F_{s_i} \operatorname{sgn}(\dot{\theta}_i) + F_{v_i} \dot{\theta}_i + \tau_{v_i} \quad (2.130)$$

## 2.8 Matlab Examples

Taking advantage of the preceding discussion, namely the application to the specific manipulator used for demonstration, along with the particularities of *Matlab*, a few functions were built to show how the above presented results could be used to simulate and operate a robot from *Matlab*. The functionality of this collection of functions is extended by the developments presented in chapter's 3 and 4 of this book, which enable the user to command the real robot from the *Matlab* shell.

Several functions were implemented to compute the direct and inverse kinematics, any rotation or transformation matrix, the jacobian (using the method presented here or the differential method presented in [25]), the DLS jacobian, trajectories in the Cartesian or in the joint space, simulate the operation of the robot, etc. The functions developed are related with the robot used for demonstration (ABB IRB1400), i.e., there was no effort to make them compatible with any other type of industrial robot. Consequently, the presented functions were optimized for anthropomorphic robots with a spherical wrist, with the direct and inverse kinematics obtained symbolically using *Matlab* and further optimized.

To demonstrate the functionality of the developed functions, a few examples will be given below.

### Jacobian

**Functions:** jacobian.m and jacobdls.m

**Parameters:** jacobian (dh, q, type) and jacobdls(dh, q, type) where,

'dh' - Denavit-Hartenberg parameters of the robot

'q' - vector or array of vectors containing the joint angles representing a configuration or a sequence of configurations of the robot

'type' - method used to compute the jacobian:

'a' - returns the base jacobian and the *end-effector* jacobian of using differential method presented in [25]

'b' - returns the base jacobian using the same method [25]

'e' - returns the base jacobian using the kinematics developed in this book

'd' - returns the both jacobians using the kinematics developed in this book

'f' - returns the *end-effector* jacobian using the kinematics developed in this book

Figure 2.22 shows the utilization of the above functions to compute the jacobian of the robot for the configuration  $q_1 = (0\ 0\ 0\ 0\ 0)$ .

```

» f1ops(0)
» J=jacobian(dh,q1,'e')

J =

    0   -720   -120    0    0    0
   955    0    0    0    0    0
    0   805   805    0   85    0
    0    0    0    1    0    1
    0   -1   -1    0   -1    0
    1    0    0    0    0    0

» f1ops

ans =

    188

» f1ops(0)
» J=jacobian(dh,q1,'b')

J =

    0.0000  -720.0000  -120.0000    0    0.0000    0
   955.0000    0.0000    0.0000    0    0.0000    0
    0.0000   805.0000   805.0000    0   85.0000    0
    0    0    0    1.0000    0    1.0000
    0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000

» f1ops

ans =

    3412

```

**Figure 2.22** Computing the jacobian: note the reduction of floating point operations when the optimized kinematics is used.

### Inverse Kinematics

**Function:** irb14ink.m

**Parameters:** irb14ink(dh, t06, quad) where,

'dh' - Denavit-Hartenberg parameters of the robot

't06' - Transformation matrix  $T_6^0$  that describes the position/orientation of the terminal element in terms of the base frame

'quad' - indication of the working quadrant. If nothing is given, the routine admits that the working quadrant is equal to the quadrant of  $\theta_1$

Figure 2.23 shows the function running applied to a singular configuration with indication of the working quadrant.

```

» qc
qc =
    0.7854    1.0472    0.7854     0     0     0
» t06=irb14mtr(qc',0,0,6)
t06 =
    1.0e+003 *
    -0.0007    0.0007    -0.0002    -0.4906
    -0.0007    -0.0007    -0.0002    -0.4906
    -0.0003    0.0000    0.0010    1.5215
         0         0         0         0.0010
» irb14link(dh,t06,'q1')
Singular Point -> sin(q5)=0
Resolving Singular Point ...
ans =
    45.0000    45.0000    45.0000    0.7854    0.7854    0.7854
    60.0000    60.0000    60.0000    1.0472    1.0472    1.0472
    45.0000    45.0000    45.0000    0.7854    0.7854    0.7854
         0    -90.0000    90.0000         0    -1.5708    1.5708
         0         0         0         0         0         0
         0    90.0000   -90.0000         0    1.5708   -1.5708
    57.2958    57.2958    57.2958    1.0000    1.0000    1.0000

```

**Figure 2.23** Computing the inverse kinematics (initial robot configuration expressed in radians)

## 2.9 Robot Control Systems

Robot control systems (Figure 2.24) are electronic programmable systems responsible for moving and controlling the robot manipulator, providing also the means to interface with the environment and the necessary mechanisms to interface with regular and advanced users or operators.

In this section, a brief overview of actual industrial robot control systems is presented, pointing out the important factors that must be addressed either by the advanced user (programmer or system integrator) or by the simple operator. Although the discussion is kept general and valid for any robot controller, a particular robot control system (the ABB IRC5 robot controller [26]) will be used for demonstration.

The robot controller has some important tasks it should perform in order to move and control the robot manipulator, provide means for inter-controller and computer communications, enable a sensor interface, and offer the necessary mechanisms and features that allow robot programming, a robot-user interface and program execution.



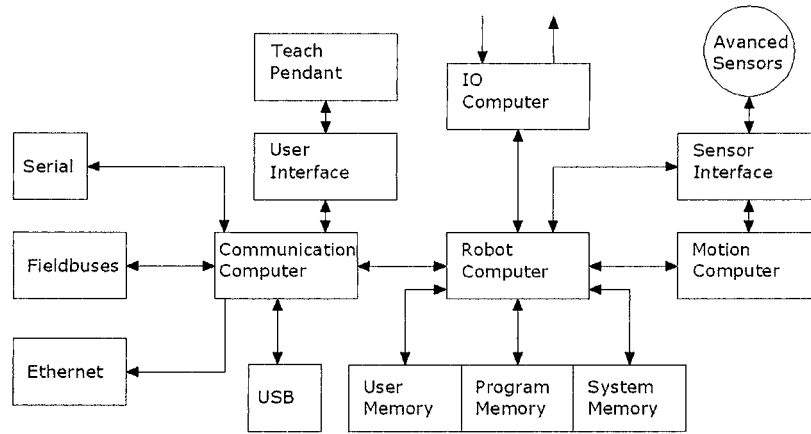


Figure 2.24 Basic architecture of a robot control system

**2.9.1 Drive the motors to move the TCP and coordinate the motion for useful work**

Motion control involves several different tasks, as already mentioned and resumed in Figure 2.25.

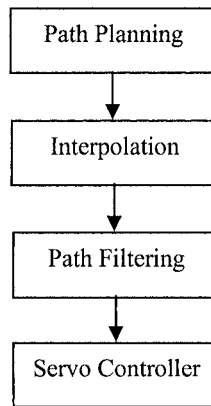


Figure 2.25 Basic tasks involved in motion control

The path planner's basic task is to prepare the robot's path and feed the relevant data to the path interpolator. Moving a robot means specifying an origin position/orientation  $\{T_i\}$  and a final position/orientation  $\{T_f\}$  of the robot's TCP

(*tool center point*). The path interpolator takes the planner data and computes the intermediate points in each interpolation interval, using the specified velocity and acceleration. The outputs of the interpolator are the basic inputs for the servo loops, i.e., they constitute the target points (references) that must be achieved by the servo controllers. The data from the interpolator is filtered by the path filter, before being passed to the servo controllers, in order to provide smoother accelerations/decelerations and keep the motor torques in the range of the servomotor.

A complete definition of the motion parameters, including velocities and accelerations, is also necessary. Sometimes it is necessary to define intermediate position/orientation points (also called “*via points*”) between the initial and final configurations. This procedure will better define the requirements and contribute for the final path. Furthermore, to obtain smooth paths the path planner must be a continuous function, with a continuous first derivative and hopefully also a continuous second derivative [1]. For example, the path generator can be implemented by a 5<sup>th</sup> order polynomial. The use of a high-order polynomial here is motivated by the fact that a quintic polynomial is needed to be able to specify the position, velocity, and acceleration at the beginning and end of each path segment.

Considering a 5<sup>th</sup> order polynomial in the form

$$\theta(t) = a_0 + a_1t + a_2t^2 + a_3t^3 + a_4t^4 + a_5t^5 \quad (2.131)$$

with the following constraints

$$\begin{aligned} \theta_0 &= a_0 \\ \theta_f &= a_0 + a_1t_f + a_2t_f^2 + a_3t_f^3 + a_4t_f^4 + a_5t_f^5 \\ \dot{\theta}_0 &= a_1 \\ \dot{\theta}_f &= a_1 + 2a_2t_f + 3a_3t_f^2 + 4a_4t_f^3 + 5a_5t_f^4 \\ \ddot{\theta}_0 &= 2a_2 \\ \ddot{\theta}_f &= 2a_2 + 6a_3t_f + 12a_4t_f^2 + 20a_5t_f^3 \end{aligned} \quad (2.132)$$

Results in a linear system of six equations with six unknowns whose solutions are

$$\begin{aligned} a_1 &= \theta_0 \\ a_1 &= \dot{\theta}_0 \\ a_2 &= \frac{\ddot{\theta}_0}{2} \\ a_3 &= \frac{20\theta_f - 20\theta_0 - (8\dot{\theta}_f + 120\dot{\theta}_0)t_f - (3\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^3} \end{aligned}$$

$$\begin{aligned}
 a_4 &= \frac{30\theta_0 - 30\theta_f + (14\dot{\theta}_f + 16\dot{\theta}_0)t_f - (3\ddot{\theta}_0 - 2\ddot{\theta}_f)t_f^2}{2t_f^4} \\
 a_5 &= \frac{12\theta_f - 12\theta_0 - (6\dot{\theta}_f + 6\dot{\theta}_0)t_f - (\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^5} \quad (2.133)
 \end{aligned}$$

There are several methods in the literature to compute smooth paths that pass to a given set of “*via points*” [27, 28]. Nevertheless, the function presented above gives a good indication and can be used for that objective, running the function between the intermediate points.

The following *Matlab* functions (Figure 2.26) calculate the robot’s trajectory in the joint space using the 5<sup>th</sup> order polynomial presented above. As already mentioned, with this trajectory planner it is possible to compute the trajectory between two configurations, defining the initial and final velocities and accelerations. The trajectory is represented using a small function that animates the motion of the robot.

#### Trajectory generation and robot animation

**Functions:** irb14trj.m and irb14plt.m

**Parameters:** [qt, qdt, qddt] = irb14trj(q0, q1, nt, qd0, qd1, qdd0, qdd1) and irb14plt(dh, q, opt, number, azm, elv, vgap, vgay) where,

‘q0’ – initial position

‘q1’ – final position

‘nt’ – number of intermediate points of the trajectory to obtain

‘qd0’ and ‘qd1’ – initial and final values of the velocity

‘qdd0’ and ‘qdd1’ – initial and final values of the acceleration

‘dh’ – *Denavit-Hartenberg* parameters of the robot

‘q’ – matrix holding the computed trajectory

‘opt’ – type of representation of the motion

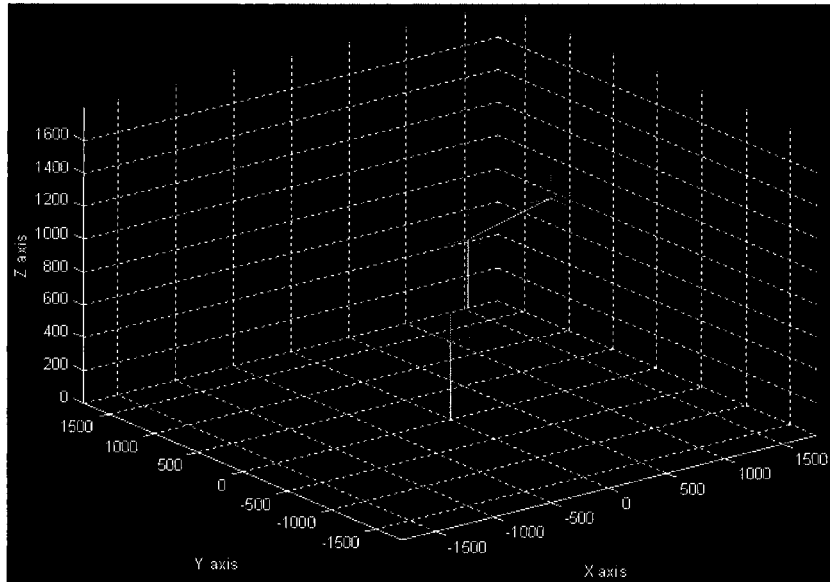


Figure 2.26 Robot's animation using the obtained trajectory

## 2.10 Servo Control

The servo controllers utilize the data from the path planner and interpolator, properly filtered, to drive the robot manipulator axis. As already mentioned the dynamics of the robot is very complex with a huge number of effects, forces and moments to account for, which puts a considerable challenge to the task of controlling a servo-motor. A detailed and complete description of a servo-controller, namely about the control algorithms and circuitry used, is out of the scope of this book, but a brief overview will be given. Generally, the control loop of an industrial robot joint (or axis) has the components presented in Figure 2.27.

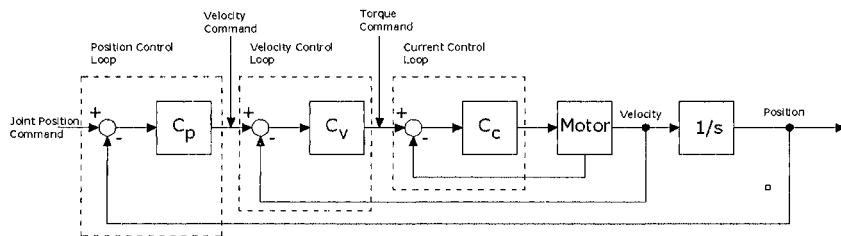


Figure 2.27 Typical robot joint control loop

A brief overview of the AC motors used with industrial robots was already presented, and a typical current control loop was also already sketched in Figure 2.20. Basically, the current control loop implements a PI (proportional and integral) controller [29], having the I component of the controller ( $C_c$ ) with the objective of eliminating the steady-state error and achieving the best possible control. The velocity control loop is built around the current control loop and also uses a PI controller ( $C_v$ ).

Finally, around both of the previous controllers there is the position control loop. This controller takes the position commands as input, generates an error signal by subtracting the actual position (obtained from the joint position sensors) from the commanded reference, and generates the control signal using some selected control law ( $C_p$ ). Typically, the position controller is a simple proportional controller, since the objective is to obtain a good responsive control of the motor position to follow the desired joint command with zero steady-state error and zero overshoot. And that objective is obtained with the combined effect of the position (generally a P controller), velocity (generally a PI controller), and current (generally a PI controller) control loops.

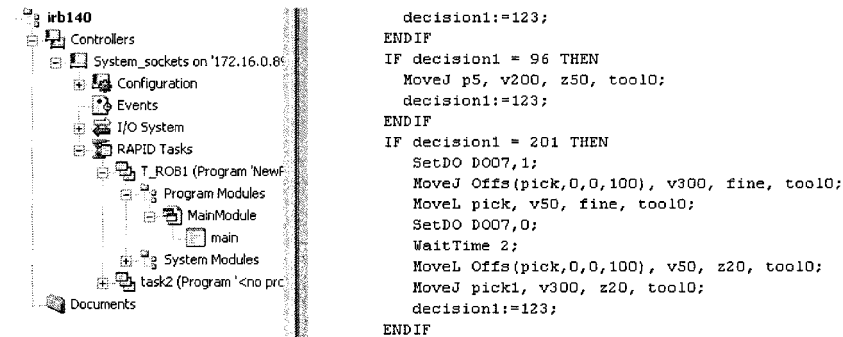
## 2.11 IO Control

One of the most basic things that a robot control system must do is to implement PLC-like functions. Robots are used in manufacturing cells where digital/analog IO and logic controllers govern the way things happen, namely controlling the systems responsible for material handling, transportation, detection, etc.. To interface with those systems, the robot controller needs to “speak” the same language and act as a logic controller, or at least have the same functionality available. Consequently, the robot controller must be able to:

1. Accommodate digital IO signals with variable and configurable electric levels. The robot must be able to read from digital input lines (with different electric levels) and implement basic logic functions on the obtained data: block reading, logic functions, shifting, counters, timers, edge detection, etc. The robot controller must also be able to act on digital IO outputs changing their state (ON/OFF), applying timed pulses, etc.
2. Accommodate analog IO signals. The robot must be able to read from analog inputs, providing the necessary electronic circuits for multiplexing and analog-to-digital conversion, the mathematical functions to handle the results, and the necessary circuits and digital-to-analog converters to act on analog output signals.
3. Implement IO manipulating functions.

The robot controller programming language must implement advanced mathematical functions, and data structures, that can be used within the robot's

program to enable the user to coordinate the robot's motion with IO actions (Figure 2.28), like reading IO information or acting on IO lines (open/close grippers, regulate pressure of pneumatic actuators, regulate the velocity of external motors driven by power inverters or external servo controllers, start/stop equipment, etc.)



**Figure 2.28** Part of a robot program written in RAPID (ABB Robotics programming language)

## 2.12 Communication

Robots are to be used in networks with other robots and computers organized into manufacturing cells that also connect to each other constituting manufacturing lines. This type of manufacturing organization corresponds to one of the most recent developments in the area of industrial automation, i.e., the concept of *flexible manufacturing systems* (FMS). These are highly computerized systems composed by several types of equipment, usually connected through a local area network (local network using MAP<sup>13</sup> protocols [30]) under some hierarchical *computer integrated manufacturing* (CIM) structure [31-33]. The available factory (*shop floor*) equipment is organized into *flexible manufacturing cells* (FMCs) with transportation devices connecting the FMCs. In some cases, functionally related FMCs are organized into *flexible manufacturing lines* (FMLs). Each FML may include several FMCs with different or equal basic capabilities. The organization proposed in Figure 2.29 is a hierarchical structure [33,34] where each FMC has its own controller. Therefore, if the manufacturing process is conveniently organized as a FML, then several controllers will exist on the shop floor level, e.g., one controller for each FML. With this setup, an intelligent and distributed job dispatching and awarding may be implemented, taking advantage of the installed industrial network [33,35-37].

<sup>13</sup> *Manufacturing automation protocol* (MAP).

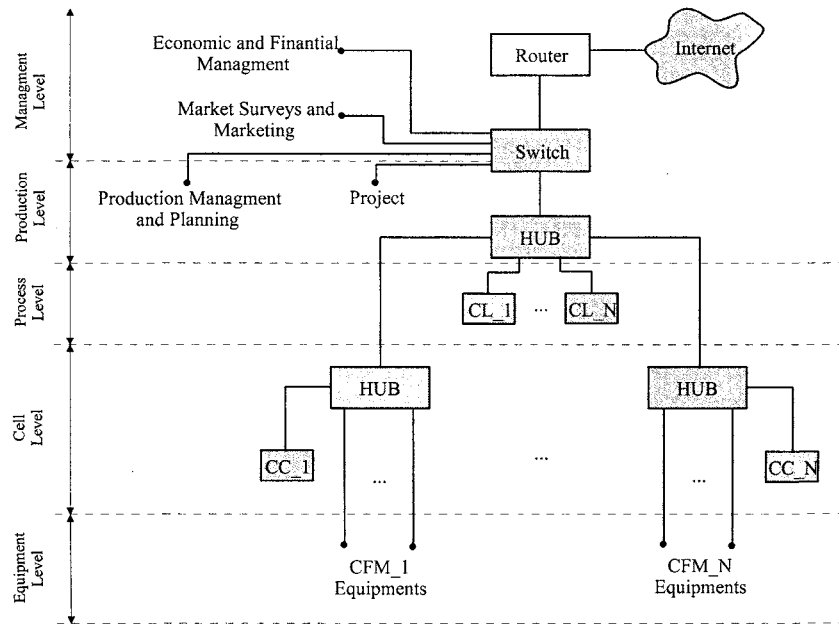


Figure 2.29 Typical CIM hierarchical organization

The best characteristic of an FMC is its flexibility, i.e., its adaptability to new manufacturing requirements that can go from a modified product to a completely new product. The flexibility results from the fact that FMC equipment is programmable and easily reconfigured: that is the case of industrial robot manipulators, mobile robots for parts handling and transportation, *programmable and logic controllers* (PLC), CNC machines, vision systems, conveyors, etc.

Considering the communication between commanding and supervising computers and the robot controllers, and even the communication between robot controllers itself, it is usually supported through a TCP/IP *Ethernet* based network. The functions associated with this type of communication include the exchange of files and programs, the execution of remote operations like backup and system maintenance, etc. In many advanced applications, this network is also used to command and supervise each manufacturing cell operation, with several levels of functionality depending on the type of access: operator access, supervisor access, or information retrieval access from the production planning levels of the network. These types of advanced features will be extensively explored in this book.

Many manufacturers offer robot services through this type of network to support these advanced applications, in the form of RPC servers [38], TCP/IP socket servers [26], or UDP datagram servers [39]. These servers and associated services can be used by the system developer/integrator to provide functionality to the user through the application.

Furthermore, the communication links between the controller and the manufacturing cell can be as follows:

1. Computer network – to interface with commanding and supervising computers, from several levels of the network
2. Fieldbuses – to interface with other robot controllers, but also with PLCs and other cell equipment commanded by programmable controllers. The most common options are *DeviceNet*, *ProfiBus*, *Ethernet IP*, etc. Several robot controllers also use a fieldbus network (*CAN* or *DeviceNet*, for example) to connect some of its internal components (the drive boards to the main computer, etc.)
3. Serial IO – to interface with sensors, or with several types of IO equipment or process equipment like welding power sources, to interface locally with a computer or laptop using a point-to-point occasional connection, and so on

## 2.13 Sensor Interface

Interfacing advanced sensors is a fundamental aspect of any robot control system. In fact, to successfully perform several actual industrial tasks, the robots need special sensors that could be used to help them get the relevant information and use it efficiently through the process. Many of these sensors require high-performance, non-perturbed communication links, and/or need to interface directly to the path planners and motion controllers so that the robot can be guided and instructed in real-time. Consequently, the robot controllers should provide special interfaces for these types of sensors, at least for the most common ones, which can be programmed and explored by the advanced user.

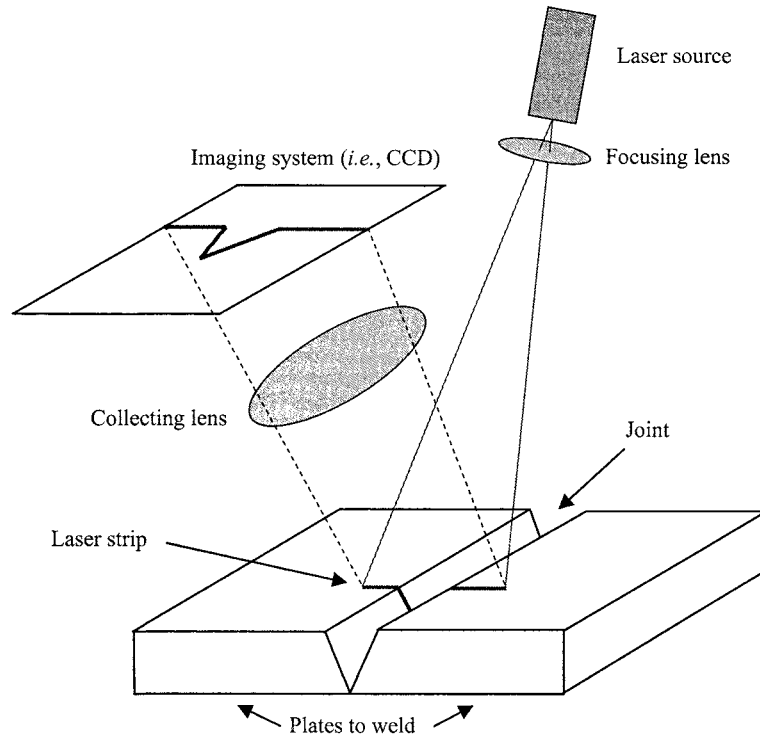
### 2.13.1 Interfacing Laser 3D Sensor for Seam Tracking

Good examples are the laser sensors used in robotic welding for seam finding and tracking during the welding operation. These types of sensors provide signals (analog or through high-speed digital interfaces) that can be used to guide the robot during the welding operation. These sensors work in a simple way, based on the principle of laser triangulation. A low power laser source is used to generate a laser beam that is projected onto the surface of the joint to weld. The reflected light is picked up by a lens that feeds the imaging system, composed usually of a CCD or CMOS sensor. The laser-reflected signals are extracted using filters and image processing software, which is a simple task since the laser signal has a very precise wave length and power (Figure 2.30).

In fact, these laser cameras and related processing hardware and software, with some customization to the selected application, are useful for evaluating most of the geometric parameters other than the mentioned joint detection and seam



tracking features. Since they are available with powerful APIs for general use, with standard interfaces for robot controllers and current computer hardware, these types of sensors constitute a powerful tool for robotic welding.



**Figure 2.30** Explanation of the laser vision principle

Basically, the outputs obtained from these sensors are position accommodations, or position corrections, that should be sent to the robot controller to adapt the current motion. They can also monitor certain variables and provide the means to generate interrupts in the robot controller in order to respond to significant variable changes. For example, the seam volume or the welding gap can be monitored by this sensor. When changes are detected, the corresponding events can be used to trigger an internal interrupt that will adapt the welding parameters (voltage, wire feed and velocity) accordingly. For example, the following would be the procedure to adapt the welding parameters in function of the measured welding gap:

**Variables**

```
Matrix Numeric Adapted_voltage = {1, 1.1, 1.2, 1.4, 1.6, 2, 2.2, ...};
Matrix Numeric Adapted_wire_feed = {2, 2.2, 2.4, 2.6, 2.8, 3, 3.2, ...};
Matrix Numeric Adapted_velocity = {10, 12, 14, 16, 18, 20, 22, ...};
Numeric gap_value;
```

```

Numeric index;

Program
Set Interrupt 1 when gap_value changes;
Start Welding, tracking;
When target point achieved
  Stop welding, tracking;
EndWhen
EndProgram

Interrupt Service Routine
index = scale(gap_value);
voltage = adapted_voltage(index);
wire_feed = adapted_wire_feed(index);
velocity = adapted_velocity(index);
refresh welding parameters;
EndRoutine

```

The position of the sensor can also be read and used to accommodate the position references sent to the motion controller, guiding in this way the robot's motion.

The next example shows how to interface other type of intelligent sensors for which there is no special interface at the robot controller.

### 2.13.2 Interfacing a Force/Torque Sensor

As already mentioned, robot manipulators are good examples of equipment for *flexible manufacturing systems*, due to their flexibility. In fact, flexibility is the major reason for robot utilization and popularity in actual manufacturing plants. In this framework, the majority of the robot's tasks require contact with the surrounding environment, i.e., in the process of fulfilling the task, the robot tool interacts physically with the working objects and surfaces. That interaction generates contact forces that should be controlled in a way to finish the task correctly, not damaging the robot tools and working objects. Those contact forces depend on the stiffness of the tool and working objects/surfaces and should be properly controlled. The option for a particular control technique depends on identifying if [40]:

1. The contact forces should be controlled to achieve task success, but are sufficient to keep them inside some safety domain: *passive force control* [40].
2. The contact forces should be controlled because they contribute directly to the success of the task: *active force control* [40-53].

In the first case, contact forces are an undesirable effect of the task and it is generally sufficient to keep them inside some safety domain. They are not necessary for the task, so usually the strategy is adding flexibility to the *end-effector* with the object of damping all the possible impacts and increasing the

tolerance to positioning errors, complemented with detailed and careful planning of flying trajectories and object approach. There are many solutions in the market to add flexibility to the *end-effector*, and in fact this is currently the standard approach in industry.

In the second case, the contact forces are necessary to finish the task correctly, i.e., controlling the contact forces to make them assume some particular value or, more generally, to follow some force profile.

For industrial robotics applications, force/torque sensors are usually placed near the working tool, generally in the manipulator wrist. This means that the sensor must be reasonably small, built in several sizes to adapt to different robot bolt patterns and load capacities, and mechanically resistant. Considering these restrictions, it is easy to understand why measuring the strain imposed on a selected strain gauge material, just by reading the voltage across the resistance of the material, is still the most used sensing technique.

There are several ways and materials to design sensing gauges, metal wire, metal-foil and semiconductor gauges being the most common. From those, the metal-foil gauges show some interesting features. The strain induced change in resistance is due to length and sectional area changes as well as a small piezo-resistive effect. With the developments in etching processes, metal-foil gauges became a very interesting possibility. They are manufactured in very thin foils (less than 10  $\mu\text{m}$ ), with sizes down to 200  $\mu\text{m}$ , etched by photographic methods. Consequently, there are virtually no limits to the variety of possible geometries. This gives greater flexibility to design geometries, but also to the type of stressing at the surface of the elastic material component where the gauge will be attached. Metal-foil gauges have very high linearity, with very low transverse sensibility (less than 0.3%), and great dynamic range. Also, their thermal characteristics are better than their semiconductor and metal-wire counterparts. All these arguments explain why metal-foil gauges are ideal for force/torque sensing elements. Force/torque sensors manufactured by JR3 (the ones we use in this book) use metal-foil gauges bound to elastic rings as sensing elements, which explain their superior behavior. Figure 2.31 shows the composition of these sensors.

**The sensing part.** It is composed of elastic rings at the outer perimeter between the mounting plates. The monolithic design eliminates hysteresis that would occur from slippage at highly stressed internal joints. The use of elastic rings produces a very stiff device, resulting in minimal deflection under load and better performance at higher frequencies. The rings and their strain gauges are positioned so that the local strain measures can be used to deduce the forces and moments, in all cartesian directions (X, Y, Z), passing through the sensor. The internal cavity between the mounting plates contains the front-end electronics where signals are amplified, digitized, and transmitted to the host receiver board. If the amplification and digitization electronics are inside the sensor, preferable for noisy or industrial environments, there is no analog signal being transmitted and high sampling rates can be achieved (8Khz).

**Table 2.3** Functions available in the MATJR3PCI *Matlab* Mex file

Functions	Brief description
init_jr3	This function opens a handle to the JR3PCI driver, checks memory, and downloads DSP code to the board.
read	Reads from a receiver board memory address.
write	Writes to a receiver board memory address.
system_warnings	Reads system saturation warnings (board memory address WARNINGS).
system_errors	Reads system errors (board memory address ERRORS).
command	Commands JR3 receiver board.
get_threshold_status	Gets the value of the threshold bits (board address THRESHOLD).
reset_threshold	Resets the threshold bits.
read_ftdata	Reads force/torque data from receiver board.
set_transforms	Sets a new transformation definition.
use_transforms	Selects the transformation to use.
read_offsets	Read offsets in use.
set_offsets	Sets actual offsets, using the current offset index.
change_offset_num	Changes actual offset index (num).
reset_offsets	Sets actual offsets to the current values read from FILTER_2.
use_offset	Changes actual offsets to the one defined.
peak_data	Sets address to watch for peaks.
peak_data_reset	Sets address to watch for peaks and resets internal values to current data.
read_peaks	Reads current peak values.
bit_set	Sets bits on defined bit-map.
set_full_scales	Sets JR3 Full Scales.
get_full_scales	Reads actual full_scales.
get_recommended_full_scales	Reads recommended full_scales.
sensor_info	Reads information from the sensor and from the receiver board. Use this function to test your setup.

**Note:** all these functions address a specific sensor, even if a multi-channel board is used.

**DSP receiver board.** Based on the same basic architecture, several interfaces can be used. If the issue is high access rates, then fast IO buses must be used and a shared memory mechanism must be implemented to exchange data and program the sensor. JR3 offers several interface buses like VME, PCI (up to four channels per board), CPCI (also up to four channels) and ISA. The receiver boards are basically DSP boards that implement digital filters and dispose sensor information to users. Also they parameterize readings (offsets, full scales, geometrical transformations, etc.) and implement a few interesting functions such as maximum

and minimum values (peaks) and, warning and error bits, etc. A full description of these functions can be found in [54], and a brief summary can be found in Table 2.3.

**Interface software and drivers.** For *Win32*-based operating systems, we developed a complete set of tools that can be used to build applications using force/torque sensors. These tools include kernel drivers designed for *Win32* operating systems, i.e., *Windows*. Basically, when we want to use some kind of equipment from a computer we need to write code and define data structures to handle all its functionality. We can then pack the software into libraries, which are not easy to distribute being language dependant, or build a software control using one of the several standard languages available. Having in mind that force/torque sensors can be used by persons with different programming capabilities, and from several types of programming languages and environments, the collection of functions that access the sensor capabilities are offered in several packages: C++ Library, *ActiveX* software component, *Matlab* toolbox and *LabView Virtual Instruments* [55].

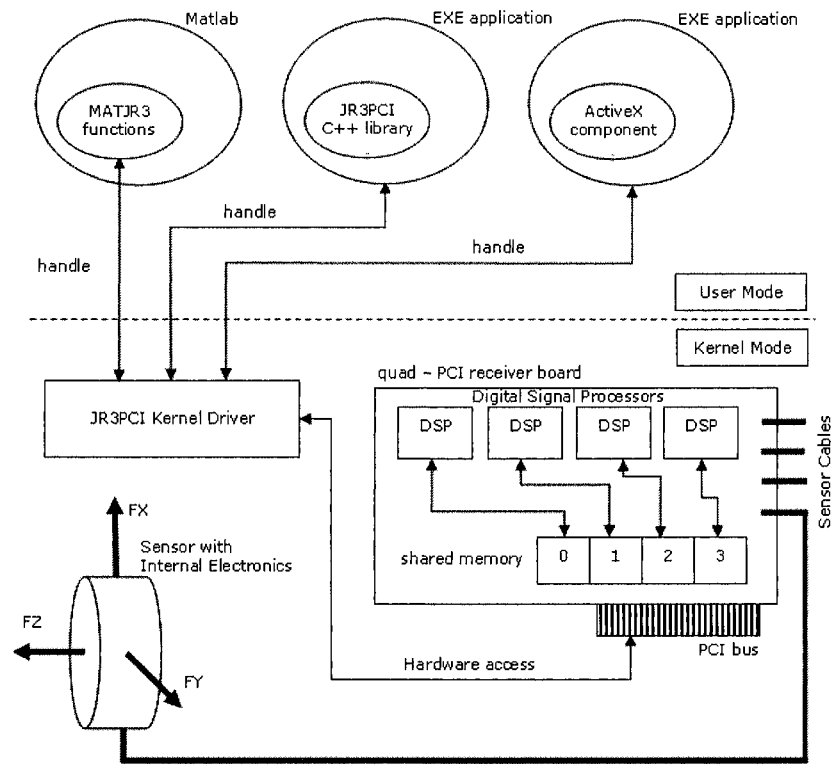


Figure 2.31 Force/torque sensor overview (using PCI receiver board)

With this organization, the sensor works like a server, offering a collection of services to the advanced user, who can use the available programming tools cited above to tailor the sensor behavior. The next section demonstrates the sensor capabilities using the popular application *Matlab*.

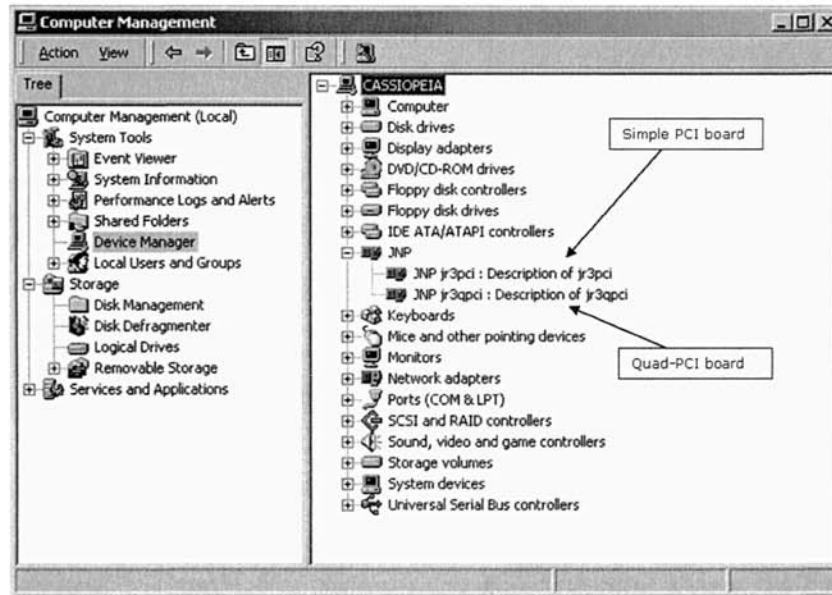


Figure 2.32 Boards reported by Windows device manager

#### 2.13.2.1 Using a Force/Torque Sensor

There are several applications of force/torque sensors, but generally a user just wants to install the sensor on his computer (after installing the sensing part on the mechanical system he is using), and then be able to parameterize it and get the sensor readings at selected rate from within the selected environment he chose to use. The basic software [54] was prepared to be used with virtually any application or programming language under *Win32* operating systems, by any type of user: from computer experts to regular users. Here we use two different environments to explore the sensor capabilities. In this section, *Matlab* is used. *Matlab* is a widely used software environment for research and teaching applications on robotics and automation, mainly because it is a powerful linear algebra tool, with a very good collection of toolboxes that extend its basic functionality, and because it is an interactive open environment. So, it is really a good environment to demonstrate how to use this type of intelligent sensor.

From all the available receiver board models, the quad-PCI receiver model was used. This board is capable of handling four force/torque sensors at the same time on a single PCI slot. It will be used step-by-step.

After having the board installed and correctly reported by the operating system (Figure 2.32), with sensor cables attached, the user is ready to start using the sensor. The first thing to do is open a handle to the sensor receiver board, check if the board is OK, and download the DSP code to the receiver's board program memory.

The command is

```
>> matjr3pci('init_jr3', vendor_ID, device_ID, n_board, n_proc, download);
```

where *vendor\_ID* and *device\_ID* are the PCI ID's of the selected board, *n\_board* is the board number (there can be several in the PCI bus), *n\_proc* is the number of DSP units in the board, and *download* specifies if the user wants to download (1) the DSP code to the program memory or not (0). Nevertheless, DSP code must be downloaded once after each computer power-up, but after that the command can be used simply to open a handle to the board. The command returns zero if successful, or an error code [45]. Consequently, to a quad-PCI board, the command with DSP code download should be:

```
>> matjr3pci('init_jr3', 0x1762, 0x3114, 0, 4, 1);
```

or without download:

```
>> matjr3pci('init_jr3', 0x1762, 0x3114, 0, 4, 0);
```

If the return value is zero (0) then the user can start using the sensor, otherwise the user must solve the problem reported by the software (error code).

The first command could be a query to the system to find out what sensor is attached to each channel. The command is

```
>> matjr3pci('sensor_info', 2);
```

to get information about the force/torque sensor handled by DSP number 2. The returned information includes model and serial numbers, software version, number of ADC bits, etc.

To read offsets from the force/torque sensor handled by DSP number zero (remember we are using a board with 4 DSP: numbered from 0 to 3),

```
>> offset_matrix = matjr3pci('read_offsets', 0);
```

To set offsets of the force/torque sensor handled by DSP number 2,

```
>> matjr3pci('set_offsets', matrix_off, 2);
```

where *matrix\_off* is a matrix with the offset values.

To reset offsets,

```
>> matjr3pci('reset_offsets', n_dsp);
```

where  $n\_dsp$  is the DSP number. With this function, the offsets are zeroed using the actual values reported by FILTER\_2 [56].

The offsets are stored in the memory available for each DSP. It is possible to store 16 independent tables of offsets for each DSP. Consequently, before any of the previous operations, the user should define the table currently in use. If the definition is not performed, all operations are referent to the actual table. To set a table for offset reading the command is,

```
>> matjr3pci('change_offset_num', 12, 1);
```

to specify that all subsequent offset operations for the sensor handled by DSP number 1 are to be addressed to Table 12. Table 12 is also used on any subsequent force/torque reading for that sensor.

To specify a table for actual force/torque readings the command is,

```
>> matjr3pci('use_offset', 10, 2);
```

where table 10 was selected for sensor handled by DSP number 2.

Another important operation on this type of sensor is setting the full-scales to properly scale the readings. This operation is similar to the operations of setting and reading offsets, so it will not be mentioned explicitly.

Each DSP has an address space [56]. To read, write, and issue commands relative to those address spaces the user should use the *read*, *write*, and *command\_jr3* commands. For example, to read the serial number (address 0x00f8 of each DSP address space) of the force/torque sensor attached to DSP number 2 the command is,

```
>> serial_2 = matjr3pci('read_jr3', 248, 2);
```

Finally, to read data from any sensor the command is,

```
>> ft_data = matjr3pci('read_ftdata', n_filter, n_dsp);
```

where  $n\_filter$  is the filter number (from 0 to 6, where 0 means unfiltered data), and  $n\_dsp$  is the DSP number.

The collection of functions available from this *Matlab* toolbox can be found in [54] and the correspondent functions of the C++ library or *ActiveX* control can be found in [57]. The same basic function prototypes have been kept between all the



software packages, which makes the above *Matlab* demonstration a good way to show how the other packages work (C++ library, *ActiveX* control, etc).

This example demonstrates how to interface an intelligent sensor to a computer. If the same facilities were available from the robot controller, then it would be equally easy to make the interface available directly from the controller, enabling in this way the programmer to directly use its readings to influence the robot's motion. Nevertheless, with most of the commercial robot controllers, this type of advanced access is not available or isn't accessible. Consequently, these types of sensors must be used from personal computers feeding the data to the robot using the available communication channels. This type of indirect approach slows down the possible performance, but it's an alternative way to implement the interface to the force/torque sensors.

## 2.14 Programming and Program Execution

Robot controllers should provide a programming language and a library of functions to enable users to explore the functionalities of the robot and of the robot's controller. Most of the manufacturers offer advanced *PASCAL-like* structured programming languages, including a language interpreter within the controller. Consequently, users can write code using any ASCII editor, download it to the controller, and run it immediately without the need for any type of file conversion. Those programming environments also offer simple debugging tools that make the process of developing software easy.

The most advanced manufacturers also offer online and offline PC-based programming tools, which enable users to develop code directly in the controller (online) using a remote PC. Alternatively, the code can be developed offline and downloaded to the controller when ready.

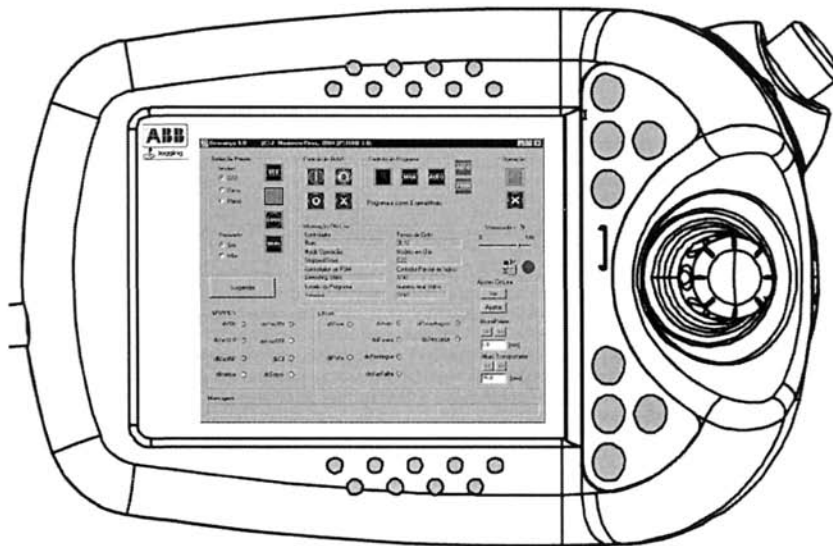
The Teach Pendant Unit (TPU) can also be used to program and parameterize the system. These devices are basically computer units running a local operating system (*Windows CE*, for example) that offer to several types of users the possibility to program, parameterize, and operate the robot manipulator.

The actual robot controllers are also multitasking systems, which enable the user to develop and run multiple tasks simultaneously. This allows new levels of functionality, offering new possibilities to the system developer. Using the available and common inter-task communication mechanisms, along with the ability to regulate task priorities (percentage of CPU time), it's possible to set up applications to handle all the challenges posed by the industrial manufacturing cells.

## 2.15 User Interface

The user interface is basically defined by the system developer, because there are a lot of possibilities. The developer can use the available communication links and the robot controller's remote servers to set up a PC interface to command and monitor the robot operation (see for example Figures 1.20 and 1.21). Alternatively, he can use the controller TPU to design the user interface. Since most of the current teach pendants are advanced computers, running powerful operating systems, the possibilities for developing advanced interfaces are enormous and flexible.

For example, the TPU that comes with the new ABB IRC5 controller [26] is a *Windows CE* system (Figure 2.33), equivalent to any portable CE based consumer device, which can be programmed remotely from a PC using common programming tools like the *Microsoft Visual Studio .NET* programming suite.



**Figure 2.33** Teach Pendant Unit showing a graphical user interface

This book explores several examples that use a remote PC to implement the user interface, examples that use mainly the TPU, and examples that use both possibilities. The idea is to demonstrate that the possibilities are there and that it's up to the system developer to pick the best options for the specific application he's building.

## 2.16 References

- [1] Craig, J.J., "Introduction to Robotics, Mechanics and Control", 2<sup>a</sup> Edition, Addison-Wesley, 1989.
- [2] Sciavicco, L., and Siciliano, B., "Modeling and Control of Robot Manipulators"-2<sup>nd</sup> Edition, McGraw-Hill, 1996.
- [3] De Wit, C.C., Siciliano, B., Bastin B., "Theory of Robot Control", Springer-Verlag, London, 1996.
- [4] Hollerbach, J.M., "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity", IEEE Transactions on Systems, Man and Cybernetics, Novembro de 1980.
- [5] Pieper, D.L., "The Kinematics of Manipulators Under Computer Control", memo. AIM72, Stanford Artificial Intelligence Laboratory, 1968.
- [6] Symon, K.R., "Mechanics", 3<sup>a</sup> Edition, Addison-Wesley, 1971.
- [7] Fu, K., Gonzalez, R., Lee, C.S.G., "Robotics: Control, Sensing, Vision and Intelligence", McGraw-Hill, 1987.
- [8] Klema, V.C., Laub, A.J., "The Singular Value Decomposition: Its Computation and Some Applications", IEEE Transactions on Automatic Control, Vol. AC-25, N<sup>o</sup> 2, April 1980.
- [9] Chiaverini, S., Siciliano, B., Egeland, O., "Review of the Damped Least-Squares Inverse Kinematics with Experiments on an Industrial Robot Manipulator", IEEE Transactions on Control Systems Technology, Vol.2, N<sup>o</sup>2, June 1994.
- [10] Golub, G.H., Van Loan, C.F., "Matrix Computations", The Johns Hopkins University Press, Baltimore, Maryland, 1983.
- [11] Wampler, C.W., Leifer, L.J., "Applications of Damped Least-Squares Methods to Resolved-Rate and Resolved-Acceleration Control of Manipulators", Journal of Dynamical Systems, Measurement and Control, Vol.110, January of 1988.
- [12] Golub, G.H., Klema, V.C., Stewart, G.W., "Rank Degeneracy and Least Squares Problems", Department of Computer Science, Stanford University, Technical Report STAN-CS-76-559, August 1976.
- [13] Maciejewski, A.A., Klein, C.A., "Numerical Filtering for the Operation of Robotic Manipulators through Kinematically Singular Configurations", Journal Robotics Systems, Vol.5, 1988.
- [14] Chiaverini, S., "Estimate of the Two Smallest Singular Values of the Jacobian Matrix: Application to Damped Least-Squares Inverse Kinematics", Journal of Robotic Systems, Vol.10, N<sup>o</sup>8, 1993.
- [15] Luh, J.Y.S., Walker, M.W., Paul, R.P.C., "Resolved-Acceleration Control of Mechanical Manipulators", IEEE Transactions on Automatic Control, Vol.AC-25, 1980.
- [16] Dote, Y., "Servo Motor and Motion Control using Digital Signal Processors", Texas Instruments and Prentice-Hall, 1990.
- [17] Herdershot Jr., J.R., and Miller, T.J.E., "Design of Brushless Permanent-Magnet Motors", Magna Physics Publishing and Clarendon Press, Oxford, 1995.
- [18] Crowder, R.M., "Electric Drives and their Controls", Clarendon Press, Oxford, 1995.
- [19] Tamagawa Seiki Co. Ltd, "SmartSyn Brushless Resolvers, General Catalog", Tamagawa Seiki, Japan, 2005.
- [20] ABB Robotics, "S4-IRB1400 Product Manual" - M94A, ABB Flexible Automation, 1994.
- [21] Hanselman, D.C., "Techniques for Improving Resolver-to-Digital Conversion Accuracy", IEEE Transactions on Industrial Electronics, Vol.38, No. 6, December 1991.

- [22] Boyes, G., "Synchro and Resolver Conversion", Analog Devices Inc. (Norwood, MA), 1980.
- [23] Analog Devices, "AD2S80A Resolver to Digital Converter - Data Sheet", Data Conversion Manual, 1995.
- [24] Goldstein, H., "Classical Mechanics", 2ª Edição, Adison-Wesley, 1980.
- [25] Paul, Shimano and Mayer, "Differential Kinematic Control for Simple Manipulators", IEEE Trans. SMC Vol.11, n.6 Junho de 1981.
- [26] ABB Robotics, "IRC5 documentation CD", ABB Robotics, 2005
- [27] Deboor, C., "A Practical Guide to Splines", Springer, New-York, 1979
- [28] Rogers, D., Adams, J.A., "Mathematical Elements for Computer Graphics", McGraw-Hill, 1976.
- [29] Ogata, K., "Modern Control Engineering" Prentice-Hall Inc., 1970.
- [30] Halsall F., "Data Communications, Computer Networks and Open Systems", Third Edition, Addison-Wesley, 1992.
- [31] Kusiak A., "Modelling and Design of Flexible Manufacturing Systems", Elsevier Science Publishers, 1986.
- [32] Ou-Yang C. and Lin JS., "The Development of a Hybrid Hierarchical/ Heterarchical Shop Floor Control System Applying Bidding Method in Job Dispatching", Robotics and Computer-Integrated Manufacturing, 1998;14(3):199-217.
- [33] Waldner JB., "CIM, Principles of Computer Integrated Manufacturing", John Wiley & Sons, 1992.
- [34] Liang GR., "A Hybrid Model of Hierarchical Control Architecture in Automated Manufacturing Systems", in Advances in Factories of the Future, CIM and Robotics, Elsevier Science Publishers, 1993:277-286.
- [35] Baker AD., "Complete Manufacturing Control Using a Contract Net: A Simulation Study", Proceedings of the IEEE International Conference on Computer Integrated Manufacturing, 1988: 100-9.
- [36] Shaw M., "A Distributed Scheduling Method for Computer Integrated Manufacturing; the use of Local Area Networks in Cellular Systems", International Journal on Production Research, 1987;25(9):1285-1303.
- [37] Zhang Y., Kameda H. and Shimizu K., "Adaptive Bidding Load Balance Algorithms in Heterogeneous Distributed Systems", Proceedings of the IEEE Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1994:250-254.
- [38] RAP, Service Protocol Definition, ABB Flexible Automation, 1996.
- [39] Remote Connection Manual for the NX100 Controller, Motoman Robotics, 2005
- [40] Siciliano B., Villani L., *Robot Force Control*, Kluwer Academic Publishers International Series in Engineering and Computer Science, Boston, MA, 1999
- [41] Pires, JN, and Sá da Costa JMG, "A Real Time System for Position/Force Control of Mechanical Manipulators", Proceedings of the 7th International Machine Design Conference, Ankara, Turkey, 1996.
- [42] De Schutter, J., and Van Brussel H., "Compliant Robot Motion I. A Formalism for Specifying Compliant Motion Tasks", The International Journal of Robotics Research, August de 1988.
- [43] De Schutter, J. and Van Brussel, H., "Compliant Robot Motion II. A Control Approach Based on External Control Loops", The International Journal of Robotics Research, August, 1988.
- [44] Craig JJ, and M.H Raibert MH, "A Systematic Method of Hybrid Position/Force Control of a Manipulator", IEEE Computer Software Applications Conference, November, 1979.
- [45] Nilsson K., "Industrial Robot Programming", Ph.D. Thesis, Department of Automatic Control, Lund Institute of Technology, May of 1996.

- [46] Hogan N., "Impedance Control: An Approach to Manipulation: Part I-Theory, Part II-Implementation, Part III-Applications", ASME Journal of Dynamic Systems, Measurement, and Control", March, 1985.
- [47] Khatib O., "A unified Approach for Motion and Force Control of Robotic Manipulators: The Operational Space Formulation", IEEE Journal of Robotics and Automation, February 1987.
- [48] Volpe R. and Khosla P., "A theoretical and Experimental Investigation of Explicit Force Control Strategies for Manipulators", IEEE Transactions on Automatic Control, November, 1993.
- [49] Volpe R. and Khosla P., "An Analysis of Manipulator Force Control Strategies Applied to an Experimentally Derived Model", IEEE/RSJ International Conference on Intelligent Robots and Systems, Raleigh, July, 1992.
- [50] Volpe R. and Khosla P., "Computational Considerations in the Implementation of Force Control Strategies", Journal of Intelligent and Robotic Systems, 9-1994
- [51] Volpe R. and Khosla P., "On the Equivalence of Second Order Impedance Control and Proportional Gain Explicit Force Control", to appear in The International Journal of Robotics Research, 1994.
- [52] Siciliano B., "Parallel Force/Position Control of Robot Manipulators", Proceedings of the 7th International Symposium of Robotics Research, Springer-Verlag, London, UK, 1996:79-89.
- [53] Chiaverini, S., "Force/Position Regulation of Compliant Robot Manipulators", IEEE Transactions on Automatic Control, Março de 1994.
- [54] Pires, JN, "MATJR3PCI", Users Manual of the JR3PCI Matlab Toolbox, <http://robotics.dem.uc.pt/norberto/jr3pci/>, 2001.
- [55] Pires, JN, "Using Matlab to Interface Industrial Robotic & Automation Equipment", IEEE Robotics and Automation Magazine, September 2000.
- [56] JR3 Force/Torque Sensor Users Manual, JR3 Inc. Woodland, California, 2001.
- [57] JR3 PCI Web Site, <http://robotics.dem.uc.pt/norberto/jr3pci/>, 2001.

## Software Interfaces

### 3.1 Introduction

This chapter explains the basics of remote procedure calling using robot manipulators and industrial automation systems in general. The underlying idea here is to demonstrate how to set up and explore a basic facility for robot cell commanding and supervision operations, using the available network services. Consequently, a client-server model is adopted where the robot acts like a server exposing to the remote clients its remote services.

The basic idea is simple. For each equipment we need to design and build a server (if it is not yet available) to expose the equipment functionality as remote services. The technology to build the server is highly dependent on the equipment resources and computing facilities, but if possible some kind of RPC (*remote procedure calls*) [1,2] mechanism should be used. Software controls that explore these services should then be available as basic tools to develop remote and distributed applications using the selected equipment.

The OSI (*open systems interconnection*) reference model [1,2] defines the seven basic levels of network communications. The OSI seven layers can be summarized as follows (Figure 3.1):

1. **Physical layer** - Provides electrical, functional, and procedural characteristics to activate, maintain, and deactivate physical links that transparently send the bit stream
2. **Data link layer** - Provides functional and procedural means to transfer data between network entities and eventually correct transmission errors. It also provides mechanisms for activation, maintenance, and deactivation of data link connections, grouping of bits into characters and message frames,

- character and frame synchronization, error control, media access control, and flow control
3. **Network layer** - Provides independence from data transfer technology and relaying and routing considerations; masks peculiarities of data transfer media from higher layers and provides switching and routing functions to establish, maintain, and terminate network layer connections and transfer data between users
  4. **Transport layer** - Provides transparent transfer of data between systems, relieving upper layers from concern with providing reliable and cost effective data transfer; provides also end-to-end control and information interchange with the quality of service needed by the application program; first true end-to-end layer
  5. **Session layer** - Provides mechanisms for organizing and structuring dialogues between application processes; these mechanisms allow for two-way simultaneous or two-way alternate operation, establishment of major and minor synchronization points, and techniques for structuring data exchanges
  6. **Presentation layer** - Provides independence to application processes from differences in data representation, i.e., in syntax; syntax selection and conversion provided by allowing the user to select a "presentation context" with conversion between alternative contexts
  7. **Application layer** - This layer is dedicated to the requirements of application. Consequently, application processes use the service elements provided by the application layer. The elements include library routines that perform inter-process communication, provide common procedures for constructing application protocols and for accessing the services provided by servers that reside on the network

The user/programmer selects the remote procedure calling mechanism to be used with the application. Ideally, the libraries used should isolate the user from the transport selected, hiding the tricky details about how to handle the communication flow.

This chapter considers the various ways to achieve client-server communication, with the objective of commanding remote execution of selected functions. The final objective is to achieve semi-autonomous systems, i.e., highly automated systems that require only minor operator intervention. In many industries, production is closed tracked in many parts of the manufacturing cycle, which is composed by several in-line manufacturing systems that perform the operations necessary to transform the raw materials into a final product. In many cases, if properly designed, those individual manufacturing systems require simple parameterization to execute their tasks. If that parameterization can be commanded remotely by automatic means from where it is available, then the system becomes almost autonomous in the sense that operator intervention is reduced at a minimum and essentially needed only for error and maintenance situations. A system like this will improve efficiency and agility, since it is less dependent on human operators. Also, since those systems are built under distributed frameworks, based on client-

server software architectures that require a collection of functions to implement the system functionality, it is easier to change production by adjusting parameterization (a software task now), which also contributes to agility. Furthermore, since all information about each item produced is available in the manufacturing tracking software, it is logical to use it to command some of the shop floor manufacturing systems, namely the ones that require simple parameterization to work properly. This procedure would take advantage of the available information and computing infrastructure, avoiding unnecessary operator interfaces to command the system. Also, further potential gains in flexibility and productivity are evident.

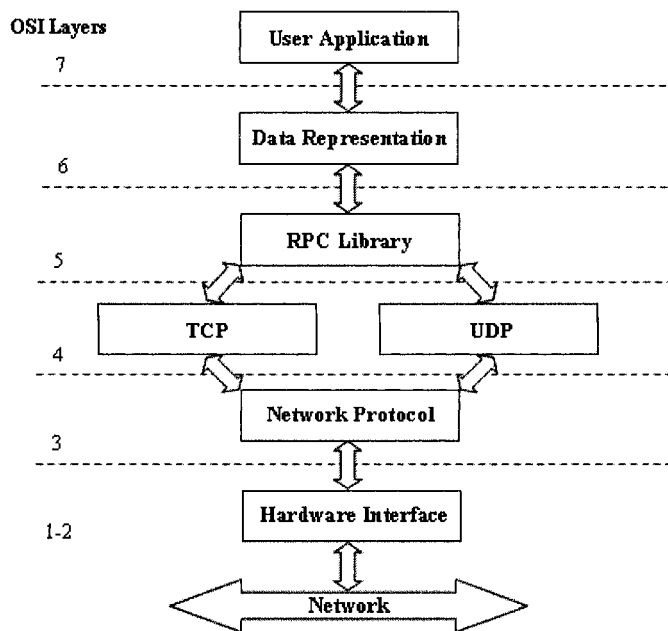


Figure 3.1 OSI reference model, with reference to an RPC library (used in this book)

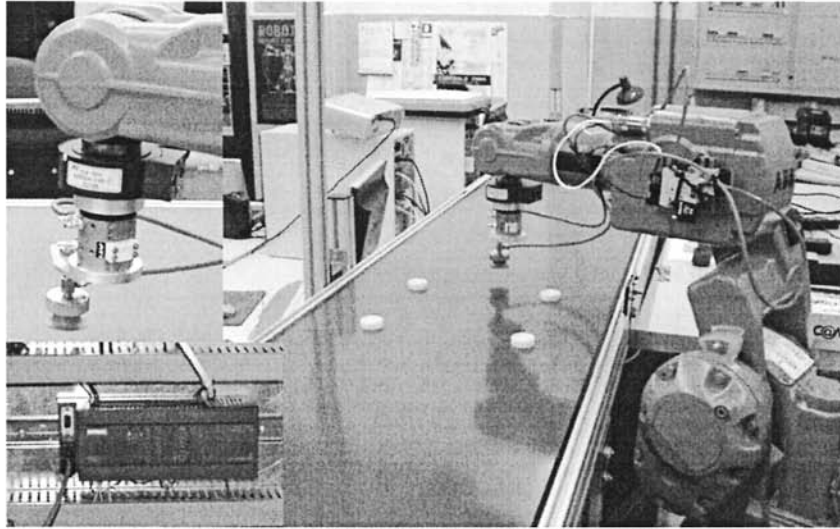
### 3.2 Low Level Interfaces

#### 3.2.1 IO Digital Signals

Probably the simplest way to exchange information between two machines, the first acting as *client* and the other as *server*, is by using IO digital signals. Basically, the *client* and the *server* can “agree” to exchange information using a predefined number of IO digital lines and a simple messaging protocol.



Let's illustrate this possibility with an example. Consider the setup represented in Figure 3.2, composed of a robot manipulator equipped with a vacuum suction cup and four fixed pick-place positions defined over a working table.



**Figure 3.2** Simple pick-and-place robotic example

The user should be able to control the robot from a personal computer (PC), commanding it to pick or place a working piece on any of the available four positions. The user should also be able to start the robot, send it to the “*home position*” and get basic monitoring information.

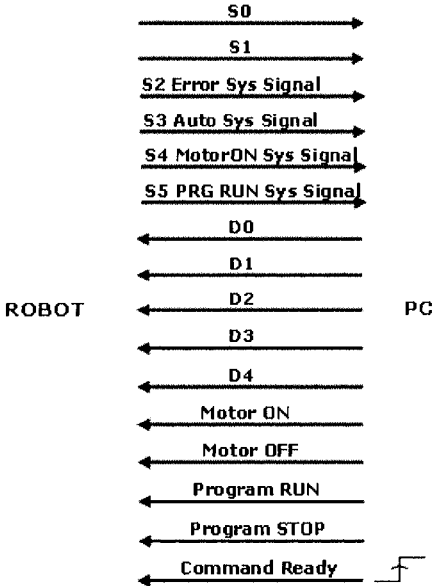
The commands needed for this application are:

<b>Commands</b>	<b>Parameters</b>
Pick piece from position	P1 to P4
Place piece at position	P1 to P4
Program RUN/STOP	...
Motor ON/OFF	...
Go home	...
Start Vacuum	...
Release Vacuum	...
Get Robot Status	...
Acknowledge Error	...

Therefore, considering all the possibilities there are seventeen different commands that require at least five bits (signals). Furthermore, to include the system commands “*Motor ON*”, “*Motor OFF*”, “*Program RUN*”, and “*Program STOP*” four new digital input signals are needed (defined in the robot controller as

SYSTEM INPUTS). These system commands may be necessary for systems that don't support multitasking, and consequently require systems inputs to implement those actions; we plan to implement the server routine as a semistatic independent task, i.e., a task that runs when the system is in automatic mode. Other systems may require to have those commands associated with independent IO lines. For generality we admit here both scenarios. The synchronization signal "*command ready*" is also needed to signal valid commands.

To add a simple handshaking mechanism to be used to get robot status information (like *busy*, *ready*, and *error* status information), and system and program state information, another six digital output signals are needed. Consequently, the following IO digital signals should be used:



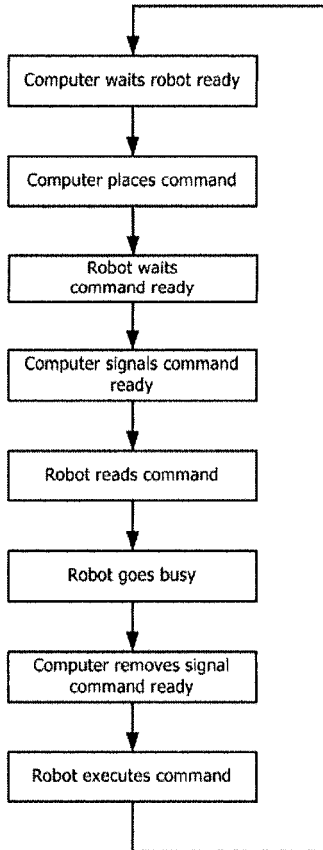
i.e., six (6) robot digital outputs for robot status information and ten (10) robot digital inputs for system command, data communication, and command validation. Consequently, Table 3.1 lists the commands identified for robot command and supervision.

**Table 3.1** Commands adopted for this example – PLC side

Command	Value of D0-D4 (Hex)
Pick from P1	01
Pick from P2	02
Pick from P3	03
Pick from P4	04
Place at P1	05
Place at P2	06

Place at P3	07
Place at P4	08
Go home	09
Start Vacuum	0A
Stop Vacuum	0B
Acknowledge Error	0C
Motor ON	0D
Motor OFF	0E
Program RUN	0F
Program STOP	10
Get robot status	1F

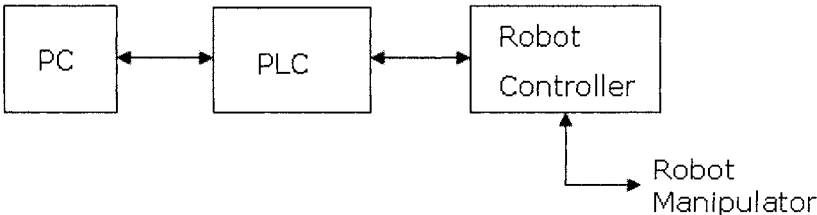
The following procedure should be used to run the presented setup:



with the following exceptions:

- 1. The robot only accepts commands when in *automatic mode*. In *manual mode* or *error state* the *robot ready* signal is never activated.
- 2. When in *manual mode*, the system always returns the *offline state* status.
- 3. On an error situation, the system returns *error state* status and requires the user to issue a *release error* command.

A simple IO board installed on the PC can be used to support the implementation of the ROBOT – PC interface. Nevertheless, in this example, an industrial PLC was used to implement the IO interface with the robot controller, being the communication between the commanding PC and the PLC done through a serial link (RS232C) – see Figure 3.3. The setup (Figure 3.2) is composed of an industrial PLC (*Siemens S7-200 CPU15*) [2], a personal computer running *Windows XP* and an industrial robot manipulator (ABB IRB 140 equipped with the IRC5 robot controller).



**Figure 3.3** Main components of the system: PC (user interface), PLC (Io interface), robot controller, and manipulator

The PLC was designed to operate as a server, offering IO services to the remote computer. Basically, the PLC waits for remote commands, processes them, and returns the status of all the IO signals. The commands have the following format:

$$CMD\ PAR\_1\ PAR\_2\ \dots\ PAR\_N$$

where *CMD* is a code that identifies the command (Table 3.2), and *PAR\_1* to *PAR\_N* are parameters associated with each command.

**Table 3.2** Commands adopted for this example – PC side

Command	Code (decimal)	Parameters
Pick	200	1 to 4
Place	201	1 to 4
Go home	202	...
Start Vacuum	203	...
Stop Vacuum	204	...
Acknowledge Error	205	...
Motor ON	206	...
Motor OFF	207	...

Program RUN	208	...
Program STOP	209	...
Get robot status	500	...

In the following few sections, the developed robot software, the PLC server software, and the PC commanding software will be presented and explained.

### 3.2.1.1 Robot Controller Software

In simple terms, the robot software executes the commands defined for the application in Tables 3.1 and 3.2, following the protocol sequence specified above. Consequently, the code has the basic structure depicted in Figure 3.4 where the RAPID programming language (from *ABB Robotics*) was used. For practical reasons the software presented in Figure 3.4 shows only the basic structure of three types of services: *Pick/Place P1*, *Go Home*, and *Start/Stop Vacuum*. It is assumed here that the robot server routine can run as an independent task, which requires a multitasking robot controller.

---

#### MODULE server\_sock

VAR Declaration Here

...

PROC main()

WHILE TRUE DO

SetDO s0,1;

WaitUntil cmd\_rdy = 1;

WaitUntil (command > 0 and command < 15);

SetDO s0 = 0;

WaitUntil command\_ready = 0;

TEST command

CASE 1:  **Pick from P1**

MoveL Offs(p1,0,0,100), v100,fine,tool;

MoveL p1, v50, fine tool;

Vacuum\_ON;

WaitUntil vacuum\_ready=1\Timeout = 2;

MoveL MoveL Offs(p1,0,0,100), v100,z10,tool;

IF timeout=TRUE THEN

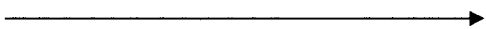
Vacuum\_ON;

SetDO s1, 1;

ELSE

SetDO s1, 0;

ENDIF

CASE 5:  **Place at P1**

MoveL Offs(p1,0,0,100), v100,fine,tool;

MoveL p1, v50, fine tool;

Vacuum\_OFF;

WaitUntil vacuum\_ready=0\Timeout = 2;

MoveL MoveL Offs(p1,0,0,100), v100,z10,tool;

```

IF timeout=TRUE THEN
  SetDO s1, 1;
ELSE
  SetDO s1, 0;
ENDIF
CASE 9: -----> Go home
  MoveJ home, v100,z10,tool;
CASE 10: -----> Start Vacuum
  SetDO doVacuum,1;
  WaitUntil vacuum_ready=1\Timeout = 2;
  IF timeout=TRUE THEN
    SetDO s1, 1;
  ELSE
    SetDO s1, 0;
  ENDIF
CASE 11: -----> Stop Vacuum
  SetDO doVacuum,0;
  WaitUntil vacuum_ready=0\Timeout = 2;
  IF timeout=TRUE THEN
    SetDO s1, 1;
  ELSE
    SetDO s1, 0;
  ENDIF
ENDPROC

```

**Figure 3.4** Application running on the robot controller (RAPID)

The application presented in Figure 3.4 uses the following variables:

- *command\_ready* – this is a digital input signal used to specify that a valid command is ready to be read. This variable is defined as a USER IO SIGNAL in the robot system parameters
- *command* – group of four digital signals (d0, d1, d2 and d3) used to specify the command that should be executed. This variable is defined as a GROUP OF IO SIGNALS in the robot system parameters
- *status* – group of six digital output signals (s0, s1, s2, s3, s4 and s5) used to specify the robot status. This variable is also defined as a GROUP OF IO SIGNALS in the robot system parameters: s0 specifies if the robot is ready (1) or busy (0), s1 specifies if a command was correctly executed (0) or if there was any execution error (1), s2 is associated with the system ERROR OUTPUT ACTION, s3 is associated with the system AUTO OUTPUT action, s4 is associated with the system MOTOR ON OUTPUT action and s5 is associated with the system PROGRAM RUN OUTPUT action. Signals s2 to s5 are defined as SYSTEM OUTPUTS in the robot system parameters
- There are also four extra robot digital IO inputs, associated with the command of system actions *MOTOR ON*, *MOTOR OFF*, *PROGRAM*

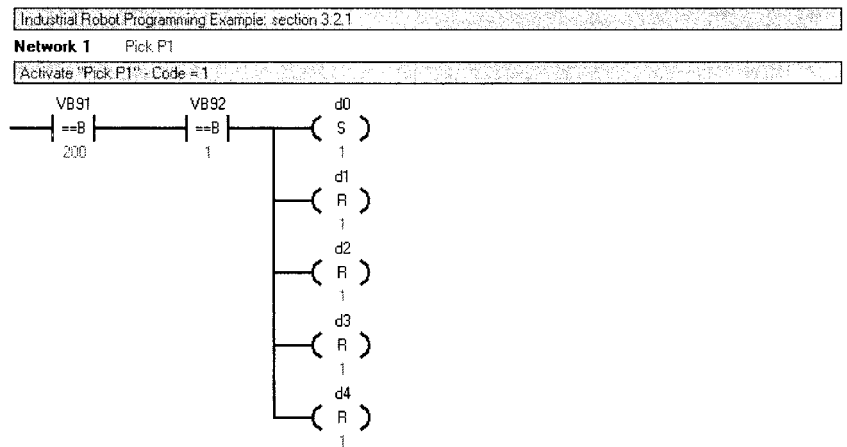
*RUN*, and *PROGRAM STOP*. These signals were named *motor\_on*, *motor\_off*, *program\_run* and *program\_stop*, respectively, and are defined as SYSTEM INPUTS in the robot system parameters.

3.2.1.2 PLC Software

The PLC software was designed to operate as a server. Furthermore, the application is basically composed of a serial port interrupt and service routine that handles the communication with the PC, placing the received string on known memory locations. In this example, the received string is copied to the memory zone that starts with byte 90. Therefore, the following happens when a message is received:

- VB90 – contains the number of bytes received
- VB91 – contains the numeric code associated with that command
- VB92 – contains parameter 1
- ...
- VB92+N – contains parameter N

Note: In this example, the number of possible parameters is limited to 5, i.e., N = 5.



Symbol	Address	Comment
d0	Q0.0	Digital Output Q0.0 [24 Volts] -> to robot
d1	Q0.1	Digital Output Q0.1 [24 Volts] -> to robot
d2	Q0.2	Digital Output Q0.2 [24 Volts] -> to robot
d3	Q0.3	Digital Output Q0.3 [24 Volts] -> to robot
d4	Q0.4	Digital Output Q0.3 [24 Volts] -> to robot

a)

```
Industrial Robot Programming Example: section 3.2.1
Network 1 Pick P1
Activate "Pick P1" - Code = 1
LDB= VB91, 200
AB= VB92, 1
S d0, 1
R d1, 1
R d2, 1
R d3, 1
R d4, 1
```

Figure 3.5 Equation to activate action "Pick P1" using the SIEMENS programming suite for the S7-200 PLC model (Step 7 Micro/Win 32 V4) [3]: a – Ladder view, b – STL view

Furthermore, any PLC action will be triggered by a byte comparison between VB91 (byte carrying the received command numeric code) and the particular numeric code associated with that action, discriminating also the parameters associated with the command. For example, to activate the command "Pick P1" the following command string must be sent to the PLC:

200 1 0 0 0 0

which results in making VB91 = 200 and VB92 = 1.

Consequently, the equation necessary to activate the action "Pick P1" is represented in Figure 3.5.

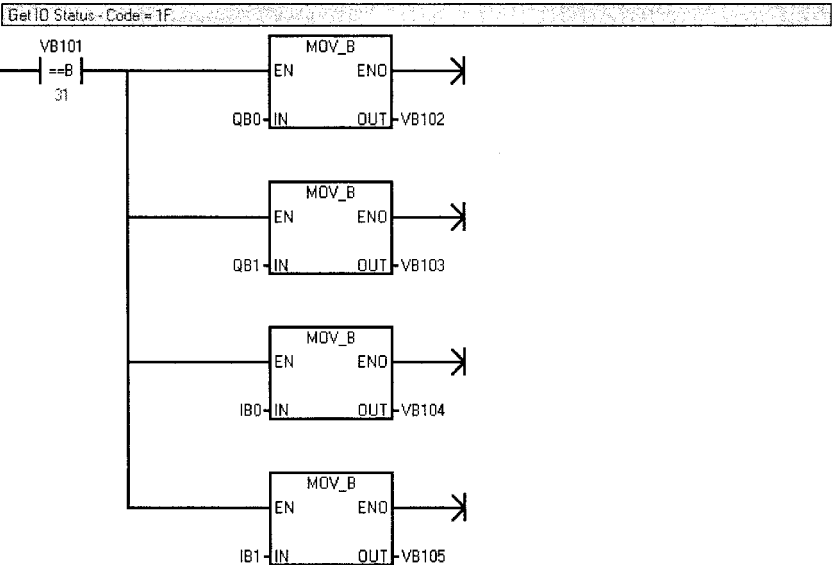


Figure 3.6 Ladder view of the "Get Robot Status" action on the PLC. Bytes VB100 to VB105 constitute an intermediate buffer used by the serial port service routine. Bytes QB0 and QB1 carry the state of all the digital outputs, and bytes IB0 and IB1 carry the state of all the digital inputs.



All the remaining actions are implemented in a similar way. Nevertheless, there is one special action that should return the robot status. This feature is obtained just by packing the actual status of all IO signals and sending it through the serial communication port, as the answer to the monitoring command “*Get Robot Status*” (code 1F) – Figure 3.6.

### 3.2.1.3 PC Software

The software developed to run on the PC provides the user interface to this setup. It is used to send the user selected commands to the PLC and to receive and present to the user the “*status*” information (Figure 3.7).

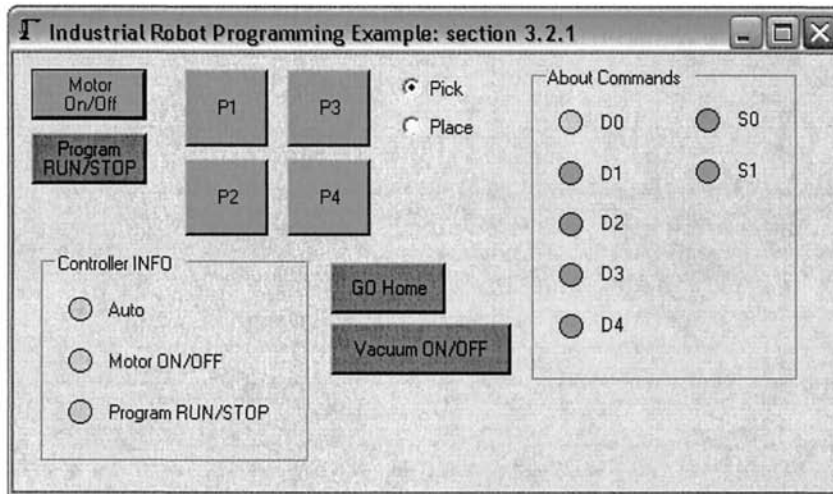


Figure 3.7 PC user interface

This simple application was coded using *Visual Basic .NET2005*. In the following (Figure 3.8) some aspects of the code associated with a few software buttons (actions) are revealed.

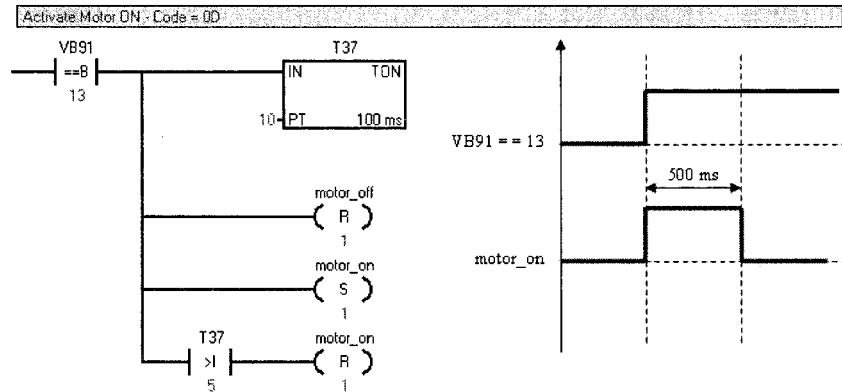
Motor On/Off	<pre> If robot_auto = 1 Then   com.Output = Chr(206) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) Else   com.Output = Chr(207) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) End If </pre>
Program RUN/STOP	<pre> If program_run = 1 Then   com.Output = Chr(208) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) Else   com.Output = Chr(209) + Chr(0) + Chr(0) + Chr(0) + Chr(0) + Chr(0) End If </pre>

```

P1
If pick.Checked = True Then
  com.Output = Chr(200) + Chr(1) + Chr(0) + Chr(0) + Chr(0) + Chr(0)
End If
If place.Checked = True Then
  com.Output = Chr(201) + Chr(1) + Chr(0) + Chr(0) + Chr(0) + Chr(0)
End If
  
```

**Figure 3.8** Some actions available from the PC software

The actions “Motor ON/OFF” and “Program RUN/STOP” are obtained just by introducing a properly temporized *IO PULSE* on the relevant robot system input, which triggers those actions. Consequently, the PLC equation for the above mentioned actions is a simple *IO PULSE* obtained using the *PULSE* function or a *TIMMER* function. Figure 3.9 shows the *ladder* view for the “Motor ON” action and the corresponding timing.



Symbol	Address	Comment
motor_off	Q0.6	Digital Output Q0.6 (24 Volts) -> to robot
motor_on	Q0.5	Digital Output Q0.5 (24 Volts) -> to robot

**Figure 3.9** – Ladder view of the “Motor ON” action on the PLC, including a sketch of the timing of the obtained PULSE

To briefly summarize this section so far, a simple example was presented where a robot is used to pick-and-place objects from four pre-defined positions. An industrial PLC was used to interface the commanding PC with the robot controller. This example demonstrates the utilization of IO digital signals to design a simple communication and data interface for commanding and monitoring applications in industrial environments.

### 3.2.2 Using Fieldbuses

A fieldbus [4] is an industrial network used for distributed control, i.e., to use with systems in which the control function is distributed among the several components of the system. In fact, actual industrial components like sensors, actuators, drive systems, programmable controllers, etc., are equipped with powerful computing systems that enable the system designer to transfer part of the control software, associated with acquisition, control, and actuation tasks, to those systems, distributing in this way the overall control function. Consequently, the available fieldbuses were developed to provide a reliable platform to transmit IO data (digital and analog) between industrial PLCs and peripheral equipment, like sensors and actuators but also to established a low-level network with other PLCs and microprocessor-based programmable devices. Consequently, fieldbuses are mainly seen by users as a way to have remote IOs, i.e., a way to access remote sensors and actuators using a two-wire network, avoiding in this way a huge amount of cables and analog transmissions on the field (process) level. Furthermore, fieldbuses are also a reliable and convenient way to make application-oriented, low-level networks. There are several technical specifications available in the market, maintained by international and generally non-profit organizations, supported by the big majority of hardware manufacturers. Three of the most popular specifications will be covered here: *Profibus*, *CAN* and *DeviceNet* [4].

#### 3.2.2.1 *Profibus (Process FieldBus)*

*Profibus* is probably the most popular type of fieldbus with more than 15 million installed devices as of 2006. It was developed in 1989 as a deliverable of a German research project, whose consortium was composed by several companies and research institutions.

Based on the real-time capable token-bus principle, *Profibus* handles multi-master and master-slave communications, allowing transfer rates up to 500 Kbits/s. *Profibus* is based on standards (the application, data, and physical layers are all standard) and enables reliable communication that distinguishes between confirmed and unconfirmed services allowing process communication, broadcast and real-time. Since *Profibus* is a master-slave pooling network with the ability to upload/download configuration data, it allows process synchronization of multiple devices on the network.

#### 3.2.2.2 *CAN (Controller Area Network)*

*CAN* is a fast serial bus that was designed to provide an efficient, reliable, and very economical link between sensors and actuators. *CAN* uses a twisted pair cable to communicate at speeds up to 1Mbit/s with up to 40 devices. Originally developed to simplify the wiring in automobiles, its use has spread to machine and factory automation products. For example, SDS (*Smart Distribution System*) was developed by *Bosch* for networking most of the distributed electrical devices

throughout an automobile, initially for eliminating the large and expensive wiring harnesses at *Mercedes* (car manufacturer from Germany).

*CAN* provides standardized communication objects for process data, service data, network management, synchronization, time-stamping, and emergency messages. It is the basis of several sensor buses, such as *DeviceNet* (*Allen-Bradley*), *SDS* (*Smart Distribution System*) from *Honeywell* or *CAL* (*Can Application Layer*) from “*CAN in Automation Group*” (a group of about 300 international users and manufacturers). *CANOpen* is a family of profiles based on *CAN* which was developed within the “*CAN in Automation Group*”. The extensive error detection and correction features of *CAN* may easily withstand the harsh physical and electrical environment presented by a car.

### 3.2.2.3 *DeviceNet*

*DeviceNet* is an extension of *CAN* adapted for critical factory networking purposes. At the next level are the “*control*” networks, which include *ControlNet*, developed by *Allen-Bradley* and also utilized by *Honeywell*, overlapping with some of the functionality provided by *Profibus-FMS* (*FieldBus Message Specification*). *Profibus-FMS* uses the same physical layer as *Profibus DP* (*Decentralized Peripheral*) but allows multi-master, asynchronous, peer-to-peer communication. *FMS* and *DP* can operate simultaneously on the same network. *ControlNet* was conceived as the ultimate high-level fieldbus network and was designed to meet several high performance automation and process control criteria. Of primary importance is the ability to communicate with each other being 100% deterministic, while achieving faster response than traditional master/slave poll/strobe networks.

Furthermore, *DeviceNet* is a simple, open networking solution that reduces the cost and time required to wire and install industrial automation devices, while providing interchangeability of components from multiple vendors. *DeviceNet* is a cost-effective solution for low-level industrial device networking and an effective way to provide access to the intelligence present in those devices. A *DeviceNet* network lets the user/programmer connect devices directly to shop floor controllers without hard-wiring each device into an I/O module. It is also used to:

- Reduce wiring and installation cost
- Reduce start-up time
- Significantly reduce downtime and the total cost of ownership with the aid of diagnostics, Auto Device Replacement, and other time- and cost-saving features
- Support standard and safety applications on the same wire
- Benefit from an open network
- Control, configure, and collect data on a single network

Consequently, using a fieldbus is not significantly different if compared to regular IO, since the same logic of encoding commands and parameters is used, utilizing

the IO signals/bits like a data bus. Nevertheless, fieldbuses use high bit rates over a reduced number of wires (normally a twisted-pair cable), which is an enormous advantage for industrial utilization since it allows a considerable reduction in the number of wires within the system. Other than that, since a fieldbus can accommodate a big number of remote IOs, it is easier to implement a messaging protocol to handle the necessary commands and related parameters, events, and monitoring tasks. In fact, many of the fieldbus consortiums developed their own protocols and consequently the user can choose between his own protocol, or the one available from the specific technology adopted.

Currently there is a debate about using *Ethernet* with predictable timing (deterministic and robust) for “*fieldbus type*” operations, i.e., penetrating deep into the factory network hierarchy, down to the I/O level. This is justified by the fact that *Ethernet* is a network commonly available on the shop floor and used for many operations between controllers and computers. A decade ago, no serious design engineer would have suggested using *Ethernet* for networking shop floor devices.

*Ethernet*, the technology for office automation, was developed more than three decades ago as a high-speed serial data-transfer network. It has become a worldwide standard and is now the most widely used *Local Area Network* (LAN). More than 85% of all installed network connections in the world are *Ethernet*. But it was deliberately ignored for industrial applications, and for good reasons: Its lack of determinism and robustness made it feeble and not suitable for the shop floor. Nevertheless, with time and research things changed, and today the scene is considerably different. In fact, over the past few years there have been many enhancements to the *Ethernet* standard, especially in areas of determinism, speed, and message prioritization. So there is no longer any reason why *Ethernet* cannot be used to build deterministic fieldbus networks that are cost-effective and open. And since *Ethernet* is already the network choice for business computing, its presence at the control level will facilitate the integration of low-level data with high-level applications.

Another good reason why manufacturers are looking at *Ethernet* is the coming explosion of shop floor data traffic. As smart sensors and various devices on the shop floor consume the available bandwidth over the next few years, manufacturing plant information generated by PLCs and control systems is expected to increase from 10 to 30 times the current level. *Ethernet*, with its Internet-friendly TCP/IP protocol, is ideally positioned. It is popular, sinking in price and being propelled by utter market demand.

Nevertheless, this scenario makes some of the PLC manufacturers uncomfortable. Even the recently arrived fieldbus systems are beginning to feel threatened by *Ethernet*. Furthermore, the *DeviceNet*, *Profibus* and *Foundation Fieldbus* protocols are all available or in development as application layers for *Ethernet*. And most PLCs now offer *Ethernet* as a standard networking option in addition to their fieldbus of choice. *High Speed Ethernet* (HSE) is a 100 Mbit/s *Ethernet* standard that uses the same protocol and objects as *Foundation Fieldbus H1*, on TCP/IP.

The new generation of *Ethernet* is called *Gigabit Ethernet*, which is capable of 1 Gbits/sec. This will bridge the gap between the necessity of industrially hardened wiring capability and the growing need for process data via business LANs and the Internet. Most companies cannot afford to have a *DeviceNet* or *Profibus* specialist on their technical staff. Even if a company could afford such a person, it is unlikely that fieldbus would be their specialty. However, almost every company has a network administrator who is well versed and specialized in the *Ethernet* protocol, making *Ethernet* even more attractive for industrial control.

In this book, *Ethernet* and TCP/IP network protocols are used extensively for several types of tasks:

1. To command distributed systems from remote computers
2. To supervise and monitor operation of the manufacturing systems
3. To exchange data, configuration setup, etc., with peripheral devices (sensors and actuators, for example)
4. To monitor and supervise operation of the remote systems, including controllers, sensors, actuator modules, etc
5. To program peripheral devices (sensors and actuators) and/or adjust their behavior
6. To receive events (asynchronous calls) from peripheral devices with data, warnings, or errors

### 3.3 Data Protocols and Connections

The challenges posed by any robotic manufacturing system are similar and independent of the particular application under study. Consequently, the software architecture [5-7] presented in this book was designed to be used with generic robotic manufacturing cells that may include several types of equipment like robot manipulators, mobile robots, PLCs (programmable controllers), CNC machines, vision systems and several types of sensors, *etc.* Usually these systems use different programming languages, even when the manufacturer is the same. It is then very difficult to make adjustments to the cell functionality, or adapt it to new requirements posed by the introduction of a new product or by changes introduced in existing products. Several research and technical efforts have been carried out to overcome these problems. Many of those efforts point to solutions that consider the development of general programming languages that could be used with any equipment, relying on individual interpreters to generate the specific code for any equipment.

Nevertheless, recent research works show that it is desirable to have a flexible environment and still program each machine using its own language. The reason is simple: a general syntax means introducing generalizations and simplifications that tend to limit the potentiality of the equipment. Consequently, some parameterization is not used, special non-grouped functions are not used, and the

generated code takes always a uniform structure which may not be the best for all machines.

The idea presented here is rather different, being an alternative to the solutions presented in the literature, and also for the software products truly distributed available on the market. The basic idea is to define for each individual machine a collection of software functions that expose all its basic operational features. That objective requires local processing capabilities, availability of communication channels, and support for the standard technologies used when implementing the services installed on the individual machines. Since the vast majority of the current robotics and automation (R&A) equipment meets these requirements fully, this is not a serious limitation. Also, the above-mentioned services are to be offered through a local network, on a distributed software framework based on the client-server model. Furthermore, using those services from the remote client computer to build controlling and inspection applications can be performed from any platform (*UNIX, Linux, Win32-DCOM, etc.*), using standard programming languages (*C, C++, C#, Visual Basic, etc.*).

Several approaches can be used and are currently available from various robot manufacturers, with specific details and implementations. Nevertheless, the following objectives are pursued by any of the above-mentioned software architectures:

1. Be able to represent the robot manipulator's motion based on the kinematic and dynamic models, but also based on real-time data coming from the real robot. That can be done using available mathematical and graphical software packages, like *Matlab* for example. This latest objective clearly indicates the need to access robot motion and status information in real-time from the mathematical package
2. Be able to develop applications to explore remotely the entire installation (robot and welding application, for example) using standard programming languages (*C, C++, C#, Visual Basic, etc.*)
3. Be able to integrate and explore intelligent sensors used to obtain information from the process under control
4. Enable users to explore the advanced programming capabilities of actual robot controllers, namely the local programming capabilities, based on a dedicated programming language complemented by extensive libraries of functions, and the optimized manipulation capabilities based on trajectory planning software that takes advantages of optimized kinematic and dynamic models
5. Enable users to build flexible manufacturing cells, which leads to the ability to explore the available industrial data network, and to distribute software to the various components of the system, as well as the capacity to build remote software applications to control and monitor industrial manufacturing cells
6. Develop advanced *Human Machine Interface* (HMI) solutions to operate with industrial systems, hiding from the users all the tricky details about

implementation, allowing them to focus on the operational details, *i.e.*, to focus on how systems work and how they can be explored efficiently

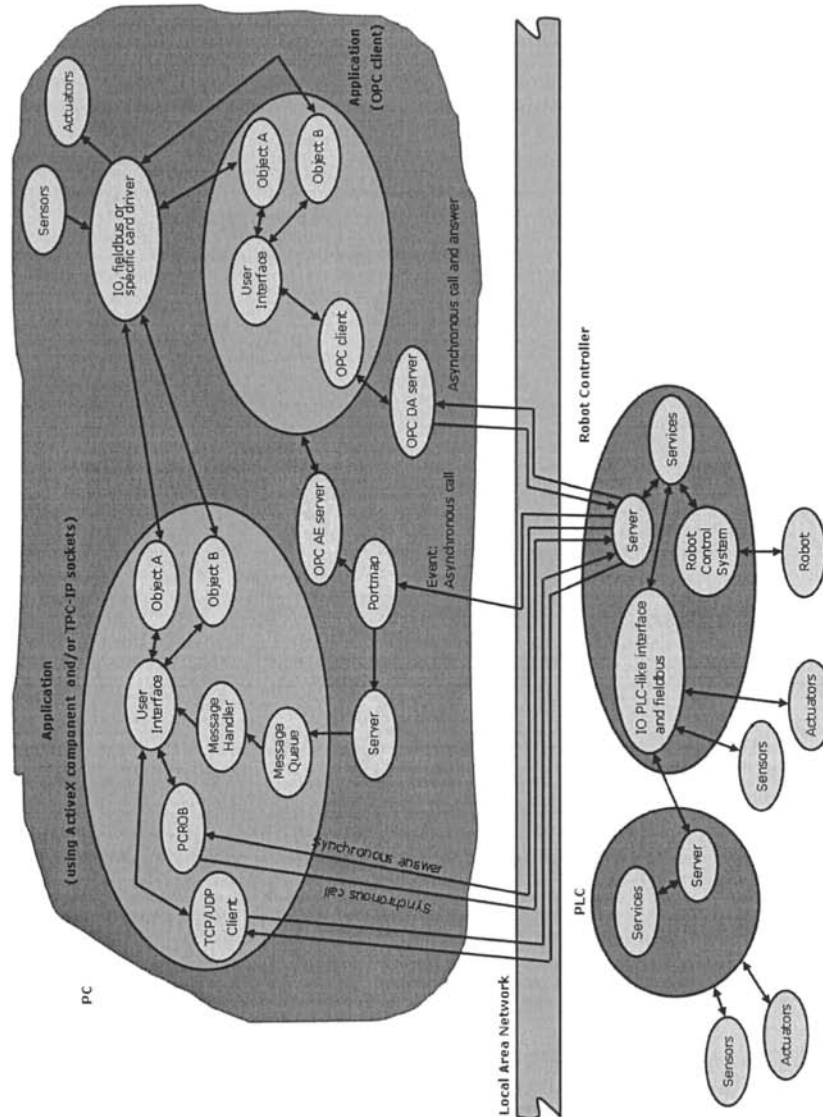
7. Provide ways that could allow developers to focus on the important things about the application they are building: the control algorithm, program functionality, and HMI. All the details related to communications, sensor integration, *etc.*, should be hidden from the user

Taking into consideration these objectives, the following programming models are required:

1. **Client-server model:** There should be server code running on each cell equipment, namely on the robot controllers and coordinating PLCs, that could receive calls from the remote client computers, execute the commands and return the results
2. **Remote procedure calls:** This is the most usual method used to implement communications between a client and a server on a distributed environment. The client makes a call to a non-local function and the selected RPC mechanism configures the call so that the proper computer, server program and function are addressed, adding the necessary network headers. The server program, running on the server machine, receives the call, executes the selected function, and returns the results obtained to the client computer
3. **IPC socket connections:** Another approach is to use TCP or UDP sockets to make the interprocess (IPC) and intersystem communication, defining a messaging mechanism to send commands and obtain process data
4. **Data sharing:** Most of the services require data sharing, files and databases between the client and the server. Consequently, the mechanism provided by the RPC technology to implement data sharing must be used

Another important thing to consider is the need to interface intelligent sensors with the system. The most easy and portable way to do that is to build software components that implement the methods, properties and data structures necessary to configure and use the sensor. Consequently, a technology to implement software components is also needed. The basic architecture presented in Figure 3.10 details all these requirements.





**Figure 3.10** Software architecture used (depicting several possibilities: using software components, using RPC sockets, using TCP/IP sockets and OPC – OLE for Process Control)

Sockets provide point-to-point, two-way communication between two processes. Sockets are very versatile and are a basic component of interprocess and intersystem communication. A socket is an end point of communication to which a name can be bound. It has a type and one or more associated processes.

Sockets exist in communication domains (*families*). A socket domain is an abstraction that provides an addressing structure and a set of protocols. Sockets connect only with sockets in the same domain. Several domains are identified and can be used to communicate between processes on a single system, like other forms of IPC.

Sockets can also be used to communicate between processes on different systems. The socket address space between connected systems is called the Internet domain, and in that case the communication uses the TCP/IP Internet protocol suite.

Socket types define the communication properties visible to the application. Processes communicate only between sockets of the same type. There are several types of socket:

*Stream socket* - provides two-way, sequenced, reliable, and unduplicated flow of data with no record boundaries. Stream sockets operate much like a telephone conversation. The socket type is *SOCK\_STREAM*, which, in the Internet domain, uses *Transmission Control Protocol* (TCP).

*Datagram socket* - supports a two-way flow of messages, not necessarily sequenced (messages can appear in a different order), and unreliable flow of data with record boundaries. Datagram sockets operate much like passing letters back and forth in the mail. The socket type is *SOCK\_DGRAM*, which, in the Internet domain, uses *User Datagram Protocol* (UDP).

*Sequential packet socket* - provides a two-way, sequenced, reliable, connection, for datagrams of a fixed maximum length. The socket type is *SOCK\_SEQPACKET*. No protocol for this type has been implemented for any protocol family.

*Raw socket* - provides access to the underlying communication protocols. These sockets are usually datagram-oriented, but their exact characteristics depend on the interface provided by the protocol.

In this book, we use *stream sockets* (for TCP client-server connections) and *datagram sockets* (for UDP client-server connections). Figure 3.11 shows the code used to open a socket on a TCP client application.

---

```

Private Shared Function C_Sock(ByVal server As String, ByVal port As Integer) As Socket
    Dim s As Socket = Nothing
    Dim hostEntry As System.Net.IPEndPoint = Nothing
    Dim address As IPAddress
    address = IPAddress.Parse(server)
    Dim endPoint As New IPEndPoint(address, Integer.Parse(port))
    Dim tempSocket As New
    Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
    Try
        tempSocket.Connect(endPoint)
        If tempSocket.Connected Then
            s = tempSocket
        End If
        Catch e As Exception
            Return s
        End Try
    Return s
End Function

```

---

**Figure 3.11** Code used to open a TCP socket connection (using *Visual Basic .NET 2005*)

Admitting that there's a TCP socket server running on the robot controller, as an independent task (process), which receives remote commands through the open socket, executes them, and returns the correspondent results, Figure 3.12 shows what a simple "motor\_on" command should look like.

---

```

server_name = ip.Text
server_port = port.Text
s = ConnectSocket(server_name, server_port)
If s Is Nothing Then
    ans_robot.Text() = "Error connecting to robot, master."
Else
    Dim bytesSent As [Byte]() = Nothing
    bytesSent = ascii.GetBytes("motor_on")
    s.Send(bytesSent, bytesSent.Length, 0)
    bytes = s.Receive(bytesReceived, bytesReceived.Length, 0)
    ans_robot.Text() = Encoding.ASCII.GetString(bytesReceived, 0, bytes)
    s.Close()
    If Encoding.ASCII.GetString(bytesReceived, 0, bytes) = "0" Then
        ans_robot.Text() = "Motor on, master."
        cstate.Text() = "Motors ON"
    Else
        ans_robot.Text() = "Error executing, master."
    End If
End If

```

---

**Figure 3.12** Sample code used to command the action "motor\_on" with TCP sockets (using *Visual Basic .NET 2005*)

This code will be used later in this book with several examples that explore the utilization of *stream* and *datagram* sockets to command industrial robotic applications.

### 3.3.1 RPC – Remote Procedure Calls

A *remote procedure call* (RPC) is a facility that a software application can use to request a service from a program located in another computer of the network without having to understand network details. (A *procedure call* is also sometimes known as a *function call* or a *subroutine call*.) RPC uses the well known client-server model. The requesting program is the *client* and the service-providing program is the *server*. Like a regular or *local procedure call*, an RPC is a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure are returned. However, the use of *lightweight processes*, or threads that share the same address space, allows multiple RPCs to be performed concurrently.

When the software statements that use RPCs are compiled into an executable program, a *stub* is included in the compiled code that acts as the representative of the remote procedure code. When the software is executed and the procedure call is issued, the *stub* receives the request and forwards it to a client runtime program in the local computer. The client runtime program knows how to address the remote computer and server application, and sends the message across the network that requests the remote procedure. Similarly, the server includes a runtime program and *stub* that interface with the remote procedure itself. Results are returned the same way.

There are several RPC models and implementations. A popular model and implementation is the *Open Software Foundation's Distributed Computing Environment* (DCE). *The Institute of Electrical and Electronics Engineers* (IEEE) defines RPC in its *ISO Remote Procedure Call Specification*, ISO/IEC CD 11578 N6561, ISO/IEC, November 1991.

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

RPC makes the client/server model of computing more powerful and easier to program. When combined with the *ONC RPCGEN* protocol compiler, clients transparently make remote calls through a local procedure interface.