EINDHOVEN UNIVERSITY OF TECHNOLOGY

Remote control of a 6R manipulator

Citation for published version (APA):
Serrarens, A. F. A. (1996). *Remote control of a 6R manipulator: an application in using computer-networks and cross-development software for real-time control of dynamic systems*. (DCT rapporten; Vol. 1996.025). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1996

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Remote Control of a 6R Manipulator

A.F.A. Serrarens

Report number: WFW 96.025

Practical Assignment Report

Author:	A.F.A. Serrarens
Institution:	Mechanical Engineering Laboratory, Robotics Dep., Cybernetics Div. Tsukuba Science City, Japan
Mentors:	K. Komoriya, Dr.Eng. (Mechanical Engineering Laboratory) dr. ir. F.E. Veldpaus (Eindhoven University of Technology)
Report number:	WFW 95.025
Date:	February 1996

Remote Control of a 6R Manipulator

An application in using computer-networks and cross-development software for real-time control of dynamic systems

A.F.A. Serrarens

27th February 1996

Contents

List of Figures

List of Tables

Summary

Nowadays, control applications can be very complex. This complexity can be the result of sophisticated and extended control strategies or because of the enormous control and communication overhead of a great number of co-operating systems. These types of applications usually require parallel processing of the real time control code. In this report the distibution of CPU's with Ethernet as communication medium is discussed as a solution for real time parallel processing applications.

This method can be simplified by the use of real time operating software especially developed for real time network communication within a set of CPU nodes of the network. The software used in this research (Wind River Systems' VxWorks 5.2) is implemented on a UNIX Workstation and can be booted by many different commercially available CPU platforms. The advantage of this is that the operating system is standardized for every node in the real time distributed control setup. The setup is easily scalable depending on the requirements of the control task.

This software and the use of Ethernet as communication line is tested by controlling a 6R robot from a (remote) UNIX Workstation communicating with a PC which is connected with the robot controller. This test was succesful and illustrates the use of this software in communication and distribution of CPU's within Ethernet.

The use of existing computer networks as real time control medium is a flexible and powerful way to deal with the continueously growing complexity in control applications. Although the method is promising, extended testing of the CPU processing times in combination with the time delays in Ethernet communication is required in order to gain successful results in any control task.

Acknowledgements

This research could not have been done without the help of a number of people, whom I would like to thank herewith.

First, I would like to thank Prof. ir. Kals (Eindhoven University of Technology) and Dr. Matsuno (Mechanical Engineering Laboratory) who made the arrangements for me to join the Mechanical Engineering Laboratory for a guest research during October 4th untill December 30th 1995.

I will thank Dr. Komoriya (Cybernetics Division, Mechanical Engineering Laboratory) for accepting me as a guest researcher. Furthermore, I would like to thank all the memebers of Cybernetics Division for there useful remarks and ideas.

Special thanks to Mr. Arisumi (Cybernetics Division, Mechanical Engineering Laboratory) for helping me in daily life problems.

I will thank the people of Fanuc Ltd., Japan for giving me the opportunity to take a look in an advanced japanese robot factory.

Furthermore, I will thank all the people of Kanagawa Industrial Technology Research Institute in general and Mrs. Komoriya in particular for showing me a High-Tech prefectural research institute.

I am grateful to dr. ir. Veldpaus (Eindhoven University of Technology) for being my official mentor of this practical assignment.

I will thank the NUFFIC-STIR/JP-foundation for their financial support for this traineeship.

Finally, I would like to thank my family, friends and girlfriend for their interest and attention during my stay in Japan.

Chapter 1

Introduction

This report offers a description of the research on the possibilities and application of controlling a 6R industrial robot through a computer network system. Section 1.1 gives a brief description of the "world of robotics". In Section 1.2 the objectives of this research are given. The outline of this report is described in Section 1.3 .

1.1 Robotics Today

Nowadays, the field of robotics is very broad in applications as well as in design. The boundary, which separates mechanical devices in robots and non-robots is not very clear and is often liable to subjective convictions, for example, in the field of mechanical manipulation. In this area a sequence of programmed tasks are executed by the mechanical device to achieve a final goal: realization of a product. The flexibility in the programming of a mechanical device is probably the most important criterion to draw the mentioned boundary. In more sophisticated robot designs, this flexibility is a natural part of the system and complex design and refined control is given more importance.

Nowadays, robots gain a certain amount of intelligence when equipped with a great number of sensors, for example tactile and vision based sensors. Industrial manipulators are equipped with sensors just to maintain their programmed, repetitive operation while sensors of intelligent robots also detect external circumstances. Use of this sensor information in a set of actions is the main topic in the control of intelligent robots.

The combination of mobilization and intelligence of robots opens a lot of potential applications for robots. Robots can be designed as self operating mobile devices, which interpret their surroundings and react by avoiding collisions, taking the easiest way, etc, simular to human behaviour. This is useful in situations in which it is not possible for humans to do certain activities because of dangerous or inhuman circumstances.

In this study a six degree of freedom robot is used to test a computer network as control medium. It is not the particular robot that is of main interest, but a demonstration of the potentials of using distributed CPU's to manage the control of a mechanical device. In the case of intelligent robots a lot of computations have to be made on a real time basis. The use of networking CPU's as an intertasking and multitasking operating system then can be very useful.

The underlying research of this report is done at the Cybernetics Division of the Robotics Departement of the **Mechanical Engineering Laboratory**, in Tsukuba Science City, Japan.

For three months I have been dealing with cross-development operating software, fundamentals of robotics, networking, programming and being in a complete different culture. At the rather young age of a graduate student, this experience is very impressive. Besides of the fact that I learned a lot of new skills in the broad field of Mechanical Engineering, I also learned a lot of new things about life and about the lives of culturally very different people. I think the latter has been just as important for me as dealing with this practical assignment, as a part of the graduate course in Mechanical Engineering at the Eindhoven University of Technology, The Netherlands.

1.2 Research Objectives

This survey narrates about the use of a computer network system as a medium to control a six D.O.F. manipulator. The manipulator is the RV-E2 Move Master produced by Mutsubishi Electric Corporation. This rather small industrial robot is provided with DC servomotors to actuate the six rotational joints. A hardware controller device, also produced by Mutsubishi, controls the servomotors when the user gives commands which are recognizable for this controller. The commands can be given by a IBM-PC which communicates with the controller by a serial line connection. The system is completed with a so called teaching box. The user can, for example, program initial positions of the robot with this device.

The purpose of this research is not to develop a new controller concept for this robot. This is even impossible, because the controller overhead is included in the controller device. For the user it is important to program a suitable task for a desired operation. Examples of such tasks are the stacking of products on a palette or tray, positioning semi-finished articles at assembly lines or welding of metals. A task is programmed by a list of commands in an IBM-PC which are send to the controller as 8-bit characters. If this PC is connected to the local Ethernetwork then, in principle, it is possible to give the commands from a remote machine, also connected to this network.

However, the problem in this application is that different computer architectures are part of the network. The development software VxWorks provides an easier way to use different types of computer architectures in a real time network application.

Generally, the graphical performance of workstations (UNIX) is much better and faster than that of PC's. If one wants to develop a graphical userinterface it is obvious to use a workstation for this. In the proposed network application an userinterface is developed in the X Windows library, which can be included in C-codes. This userinterface holds the commands the user can give to the controller A kinematic wire-model can simulate the movements of the manipulator.

Figure 1.1 shows the configuration of the hardware elements of this application. Note

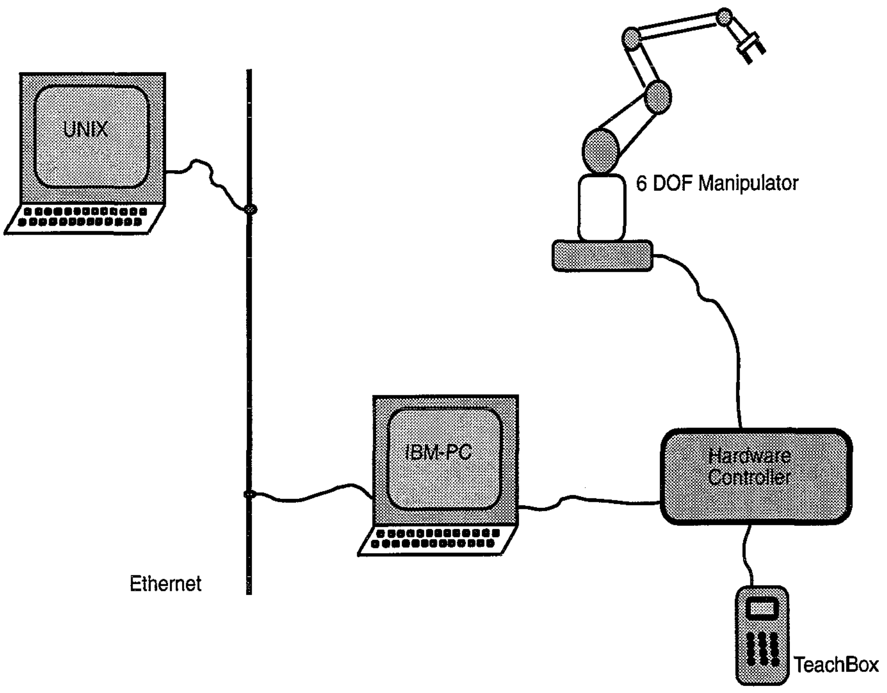


Figure 1.1: **Hardware configuration of network experiment**

that in reality there are many more computers connected to the Ethernetwork and that these machines can be used for real-time control applications. This gives the potential to use extended control strategies that are available in theory but hardly ever used in practice because of their high demands on CPU-time and/or memory. Especially, model-based control strategies are a real burden in real time controll. Another example of using the network as a real time medium is the simultanuous controll of two or more co-operating manipulators. One can easily make up many more examples of using real time network development.

1.3 Outline of this Report

In the experiments a robot with six rotational joints (6R) is used. In Chapter 2 this manipulator is described in more detail. Aspects like forward kinematics, inverse kinematics, dynamics and control are discussed.

The next chapter explaines the background of the network application. The operating software VxWorks is considered together with the use of this software in the ix86

CPU-family. Fragments of programs illustrate the aspects of implementing a real-time controlling-task through the Ethernet.

To ease the use of the industrial manipulator a graphical userinterface is developed. Simulation of the robot movements, kinematics and inverse kinematics are required to calculate the positions and postures of the robot. A wire model shows these movements, positions and postures in a graphical display. This graphical userinterface is discussed in Chapter 4.

To test the network application, a pick-and-place task is implemented in the UNIX workstation. This task, in the form of commands, is send to the PC via the Ethernet and from the PC to the robot (-controller). This test is also discussed in Chapter 4.

At the end-effector of the robot a laser range sensor is attached. With this sensor it is possible to scan simple-shaped objects. A suitable algorithm controls the robot in such a way that it can locate the position and orientation of the object in a predefined area. After that, the robot is able to pick up the object. This experiment is done with the PC as a stand-alone machine. These experiments are also discussed in Chapter 4.

This survey is concluded in Chapter 5. Conclusions and recommendations summarize the preceding discussions.

Chapter 2

Kinematics and Dynamics of the 6R Robot

The robot under consideration is a typical six degree of freedom manipulator that is very common in industrial applications. In Section 2.1 the kinematics is point of discussion. Inverse kinematics is derived in Section 2.2. The dynamic features of the robot are of less importance in this application. Nevertheless, some dynamics and control of industrial manipulators are discussed in Section 2.3. This chapter is concluded in Section 2.4

2.1 Forward Kinematics

The most important part of an industrial robot is its end-effector. After all this part has to perform a desired task (follow a trajectory, pick-up products, etc.). Such a task is realized by mutual co-operation of, in this case, six links. The links are connected with rotational joints. Every joint can change the relative position and orientation of all consecutive joints.

The forward or direct kinematic analysis [1] provides methods to compute the position and orientation of the manipulator's *end-effector* relative to a user-defined *base* as a function of the *joint-variables* and a given set of *link-parameters*. The last two quantities are explained further on in this section.

In this section one kinematic convention is discussed thoroughly: **The Denavit-Hartenberg Convention**. In this convention Cartesian coordinate frames fixed to each of the links are introduced. The position and orientation of a joint, between two links, specifies the spatial transformation between two consecutive coordinate frames.

Generally, a transformation between Cartesian coordinate frames is described by a

[1]A definition of kinematics in general is: the science of motion (position, velocity, acceleration and all higher derivatives with respect to time) that analyzes motion without regard to forces which cause it.

(4x4) homogeneous transformation matrix A, i.e.

$$A = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \vec{n} & \vec{o} & \vec{a} & \vec{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

This matrix can be divided into two main parts, i.e.

$$T = \begin{bmatrix} R_{(3x3)} & \vdots & P_{(3x1)} \\ O_{(1x3)} & \vdots & 1 \end{bmatrix} \tag{2.2}$$

in which $R_{(3x3)} = \begin{bmatrix} \vec{n} & \vec{o} & \vec{a} \end{bmatrix}$ represents the rotational component and $P_{(3x1)} = \vec{p}$ the translational component. Denavit and Hartenberg apply this general transformation between any two coordinate frames to write

$$T_n = A_1 \cdot A_2 \cdot \ldots \cdot A_n \tag{2.3}$$

where T_n defines the position and orientation of a coordinate frame fixed to link n (end-effector) with respect to a coordinate frame fixed to the user-defined base. A_i defines the transformation of a coordinate frame $\mathcal{T}_{i-1}$, fixed to link $i-1$, to the coordinate frame $\mathcal{T}_i$, fixed to link i. This transformationis described by (see Appendix A):

$$A_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

This transformation is a function of the four **Denavit-Hartenberg parameters** (D-H-parameters) θ_i, d_i, a_i and α_i. For rotational joints, θ_i is called a *joint-variable* or *generalized joint coordinate*. The remaining three parameters are called the *link-parameters*.

Figure 2.1 illustrates the definitions of these parameters. With the convention of fixing link frames to links as in figure 2.1, the following definitions of the D-H parameters are valid:

θ_i = the angle between X_{i-1} and X_i measured about Z_i
d_i = the distance from X_{i-1} to X_i measured along Z_i
a_i = the distance from Z_{i-1} to Z_i measured along X_{i-1}
α_i = the angle between Z_{i-1} and Z_i measured about X_{i-1}

For the Move Master robot the parameters can be found in table 2.1

Figure 2.2 depicts the kinematic model of the Move Master robot. This model is illustrative for the parameters in table 2.1

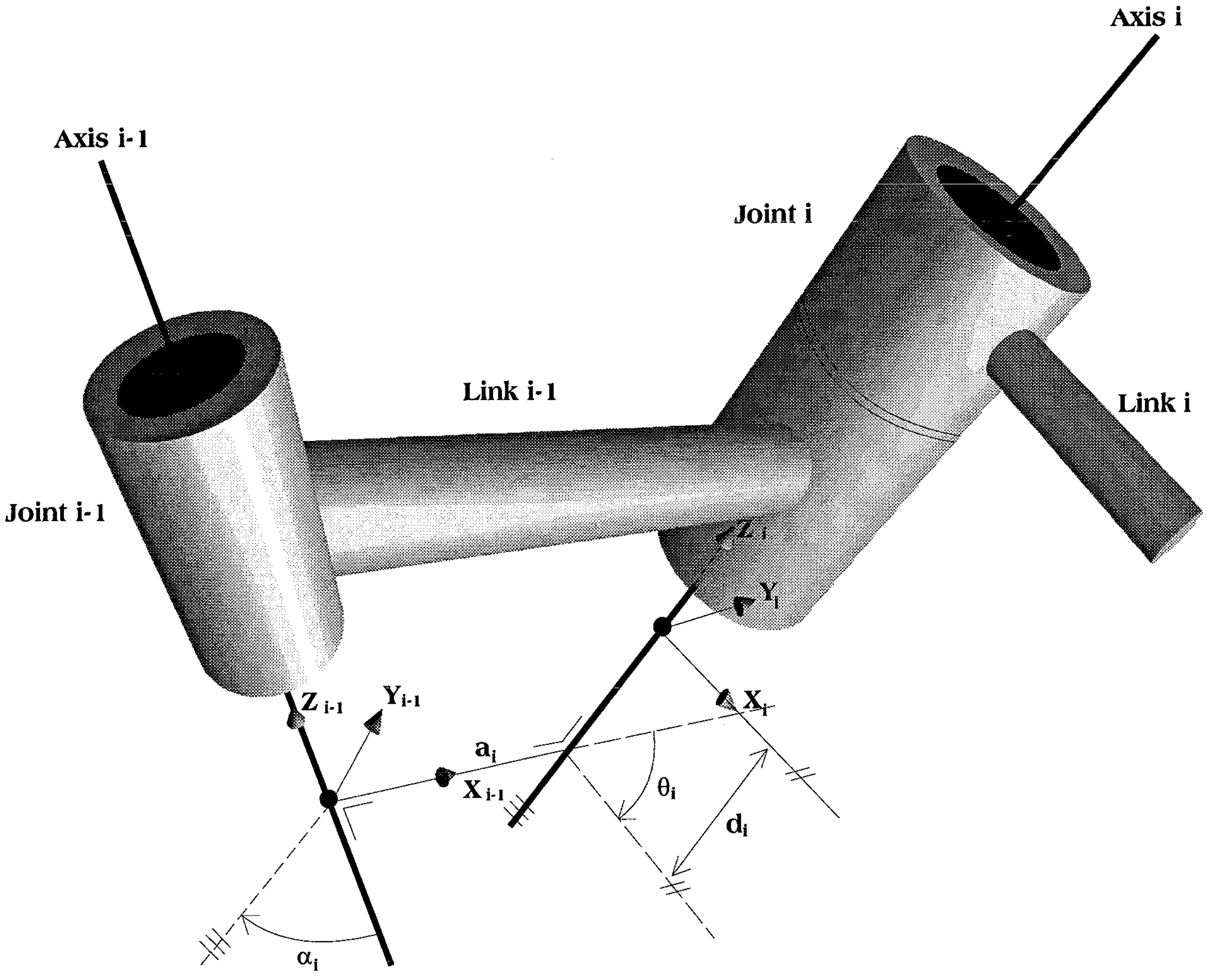


Figure 2.1: **Two consecutive links and their D-H parameters**

Table 2.1: **D-H parameters of the 6R Move Master Robot**

i	θ_i	d_i	a_i	α_i
1	θ_1	L_1	0	0
2	$\theta_2 - 90^o$	0	L_2	-90^o
3	$\theta_3 - 90^o$	0	L_3	0
4	θ_4	L_5	L_4	-90^o
5	$\theta_5 + 180^o$	0	0	$+90^o$
6	θ_6	L_6	0	-90^o

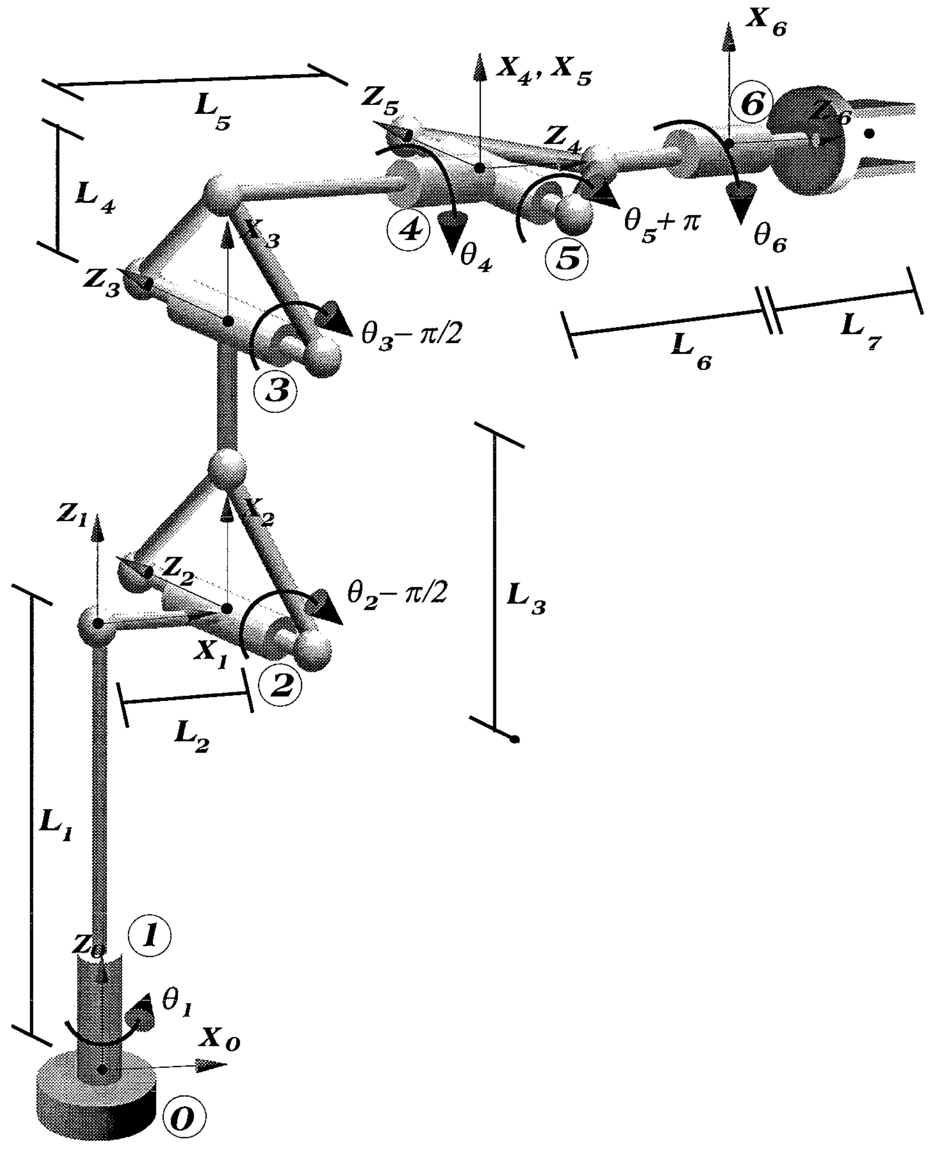


Figure 2.2: **Kinematic model of the Move Master Robot**

2.2 Inverse Kinematics

In the previous section a method to compute the position and orientation of the end-effector relative to the base, given the joint angles, was discussed. In this section the inverse problem is observed: Given the desired position and orientation of the end-effector relative to the base, how to compute the set of joint angles which will achieve this desired result? This problem of **inverse kinematics**, is more important in task programming of a robot than **forward kinematics**. Unfortunately, it is also more difficult to solve.

In the case of forward kinematics, the state of the manipulator is given in the joint space by the DH-parameters. Inverse kinematics requires the state of the manipulator given in the Cartesian space description of the end-effector (see Figure 2.3 The position of the end-effector relative to the base is presented by x_e, y_e and z_e and its orientation relative to the base by α_e, β_e and γ_e.

Generally, it is impossible to solve the inverse kinematics in a closed form because of the strong non-linearities in the transformations. On the other hand, if the kinematic lay-out of a manipulator is rather simple it is often possible to solve the inverse kinematics in closed form. To solve the inverse kinematics of the Move Master one has to deal with

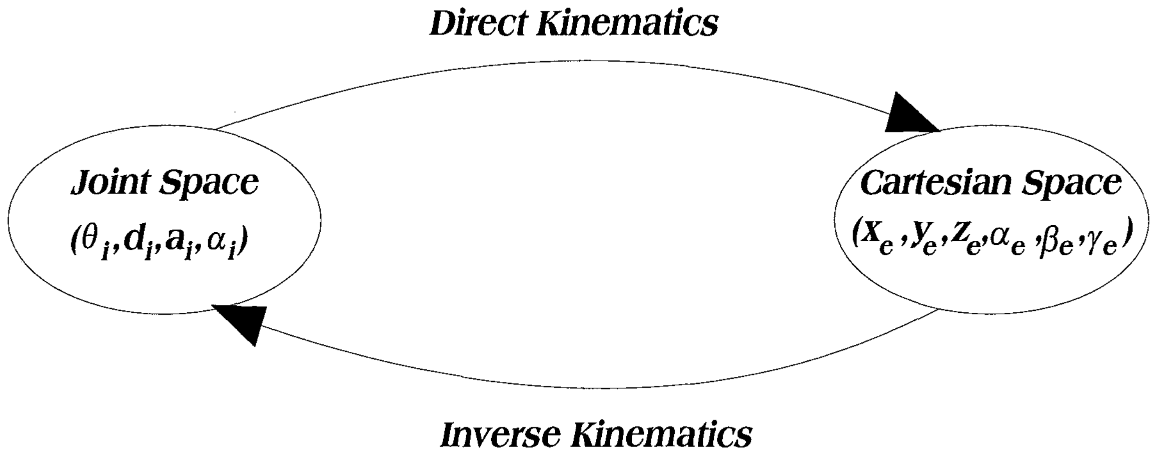

Direct Kinematics
Joint Space
$(\theta_i, d_i, a_i, \alpha_i)$
Cartesian Space
$(x_e, y_e, z_e, \alpha_e, \beta_e, \gamma_e)$
Inverse Kinematics

Figure 2.3: **The two fields in the science of kinematics**

the fact that, in general, there is no unique solution: there can exist multiple solutions, an inifinite number of solutions or no solution at all.

The occurence of multiple solutions simply results from the fact that the end-effector can reach a specified position with more than one configuration of the linkages. In that case the manipulator is said to be **redundant**. This is illustrated in Figure 2.4 for a three link planar arm. Then the control algorithm has to choose between possible solutions, using criteria such as the avoidance of collisions with external obstacles. Another criterion could be the minimization of the required movements of each joint. In practical situations there might be many more criteria to choose between possible solutions.

An example of an infinite number of solutions can be found in the Move Master. If $\theta_5 = 0$ then $\theta_6 = -\theta_4$ results in the same position and orientation of the end-effector

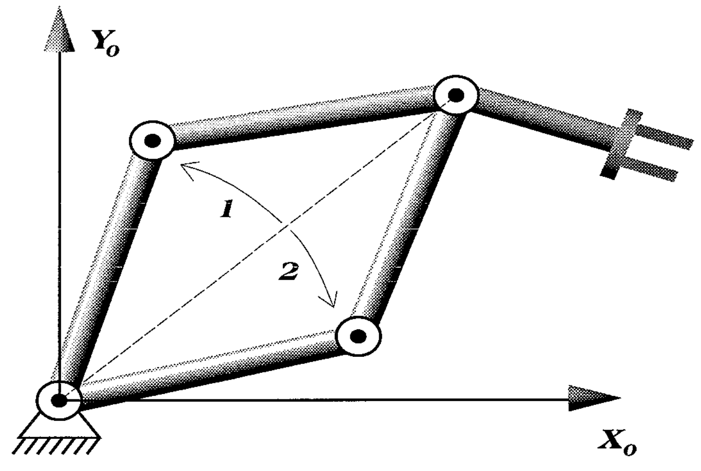


Figure 2.4: **Redundant solution of a three link arm**

relative to the base for each value of θ_4. In this case, the manipulator is said to have a **degeneracy**. If not restricted, control over the two degrees of freedom is lost.

Situations in which no solutions can be found occur if the prescribed orientation and position do not lie in the manipulator's workspace.

Generally, solutions can be obtained by geometrical, algebraical or numerical methods or by mixtures of these methods. In Appendix B, a purely geometrical solution for the inverse kinematics of the Move Master is presented. Pieper [11] describes an algebraic method to solve the inverse kinematics of 6 D.O.F. manipulators with three consecutive axes intersecting in a point. This method could also be applied to the Move Master, since the axes of the last three joint frames intersect at the origin of the 5th joint frame. Manocha et al. [7] describe a mixture of algebraical and numerical methods to solve the inverse kinematics of general six revolute joint manipulators. Also their method could be applied to the Move Master.

2.3 Dynamics and Control

In the field of dynamics of manipulators, globally, two formulations are used: the Newton-Euler and the Lagrange formulation. Because it is out of the scope of this research to discuss the dynamics of industrial manipulators in detail, a brief discussion of the formulations together with applications in manipulator control is presented.

The science of dynamics deals with the motion of bodies under the influence of applied forces. For manipulators, the dynamics are described in terms of changes of the arm configuration as a function of time caused by torques exerted by the joint actuators. Why are dynamics important in industrial manipulators? One can try to control the actuators such that the end-effector traces a desired trajectory. Stepper-motors or DC motors can be used to realize desired joint angles. In this case feedback control of joint angles and joint velocities is enough. However, this will result in the end-effector to lag behind the desired trajectory during acceleration and to overshoot when decelerating, especially for high speed operations. Many industrial manipulators operate at reduced speed to overcome this problem. This also holds for the Move Master. When higher speed is required, a more sophisticated control strategy is necessary. At this point there is a need for model based control.

This requires a dynamic model of the manipulator. Such a model can predict the behaviour of the robot arm as a result of given joint torques. Implementing this model, the control scheme has a feedforward control part (dynamic prediction of the positions, velocities and accelerations) and a feedback control part to deal with all interactions that have not been taken into account by the model.

There are two parts in manipulators dynamics: **forward dynamics** and **inverse dynamics**. Forward dynamics concerns the problem of computing the manipulator motions as a result of applied joint torques. Inverse dynamics is related to computing the required set of joint torques given the positions, velocities and accelerations of the joints as a function of time. In task planning, the inverse dynamics play a crucial role.

The dynamic model of a manipulator with n links and n actuated joints, obtained by either the Newton-Eulerian (see [5]) or the Lagrangian (see [18]) formulation, has the following general form:

$$\tau = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) + F(\Theta, \dot{\Theta}) \qquad (2.5)$$

where τ is the $nx1$ vector of applied joint torques, $M(\Theta)$ is the nxn **mass matrix,** $V(\Theta, \dot{\Theta})$ is the $nx1$ vector of centrifugal and Coriolis terms, $G(\Theta)$ is the $nx1$ vector of gravity terms, and $F(\Theta, \dot{\Theta})$ is the $nx1$ vector that represents other interactions, for instance due to friction, damping and flexibility of the links. The last term is extremely hard to model and is often not included in the dynamic model. While controlling the manipulator, these unmodelled effects have to be overcome by feedback control.

The variables in the model (2.5) are the joint parameters Θ (joint space). However, the desired motion of the end-effector usually is given in Cartesian space so there is a need for transformation of (2.5) from joint to Cartesian space. There are two ways to implement these transformations into a control scheme. The first one is to use inverse kinematics, which results in a **joint based control scheme** and the second one is to use forward kinematics, which results in a **Cartesian based control scheme**. Both methods require the so called Jacobian J, a nxn matrix of partial derivatives $\frac{\partial x_i}{\partial \theta_i}$. The Jacobian relates velocities in Cartesian space (x_i) and joint space (θ_i) with each other. Torques in Cartesian space are interpreted as the set of forces and torques applied at the end-effector resulting in the desired motion of this manipulator part. The relations between velocities and torques in the two spaces are

$$\begin{aligned} \dot{\mathcal{X}} &= J(\Theta)\dot{\Theta} && (2.6) \\ \mathcal{F} &= J^{-T}(\Theta)\,\tau && (2.7) \end{aligned}$$

in which $\dot{\mathcal{X}}$ is the $nx1$ vector of end-effector velocities (translational and rotational), $\dot{\Theta}$ is the $nx1$ vector of angular joint velocities, $\mathcal{F}$ the $nx1$ vector of applied forces and torques at the end-effector and τ is the $nx1$ vector of applied joint torques.

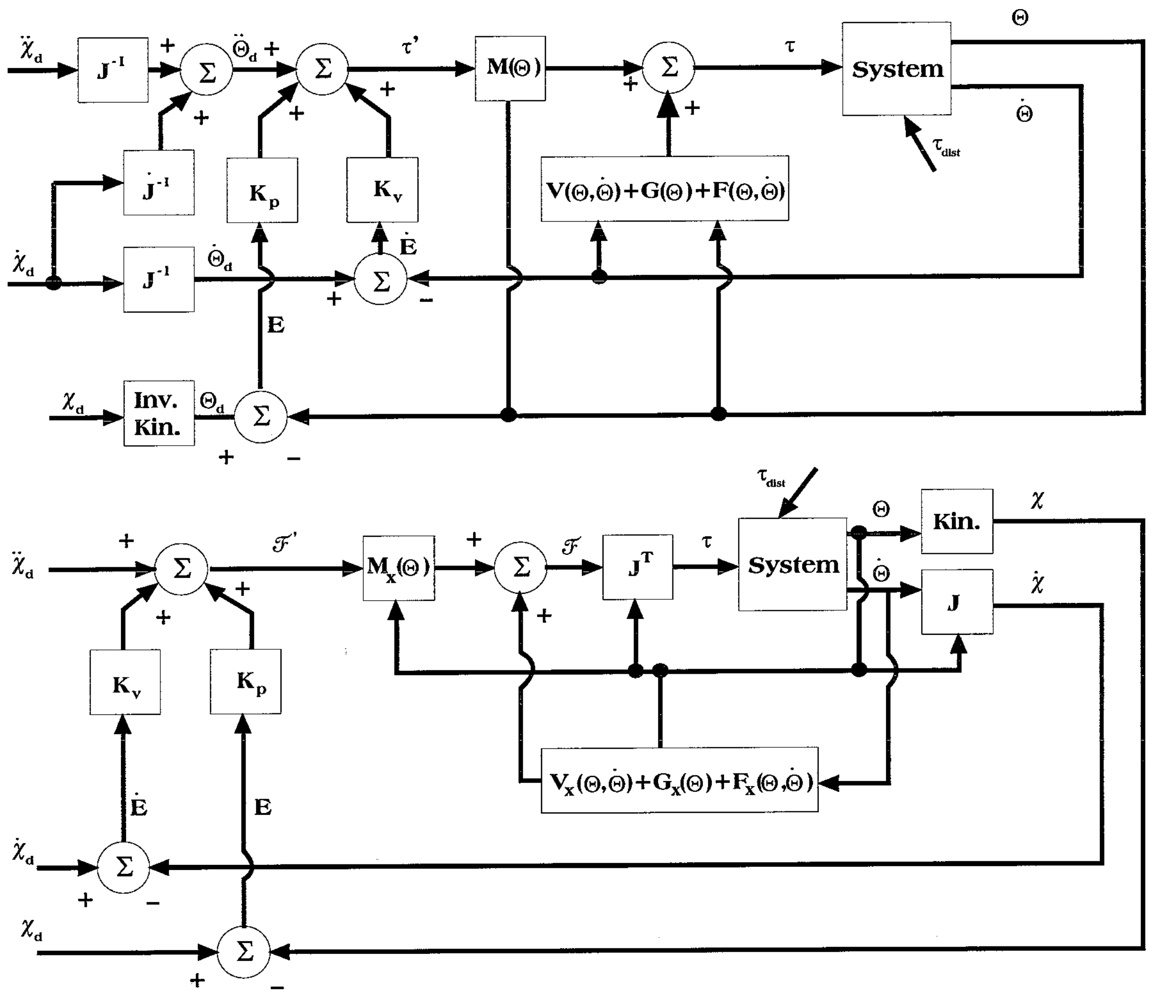


Figure 2.5: **Joint space and Cartesian space control schemes**

The two control schemes are depicted in Figure 2.5. In these schemes the feedback part and the feedforward part are included. In Joint space control, the error in joint angles and velocities is corrected by the feedback gains K_p and K_v respectively. For Cartesian space control the feedback control "pushes" the errors in the end-effector positions and velocities to zero by equivalent gains. The control strategy of the feedforward control part applied in both cases is trying to remove all non-inertial forces and torques. In the ideal situation the only part that remaines would be a simple Newton's law:

$$\tau = M(\Theta)\ddot{\Theta} \tag{2.8}$$

Unfortunately, there are a number of errors, which have to be overcome by the feedback control. The cause of these errors can be unmodelled aspects (joint friction and resonances, for example), errors in manipulator parameters, external disturbances (τ_{dist}) like an unknown load at the end-effector, errors in measurements, differences in the calculated actuator torques and the actual actuator torques, etc.

The system error equation of the proposed controlled system can be written as

$$\ddot{E} + K_v\dot{E} + K_pE = M^{-1}\tau_{dist} \tag{2.9}$$

Since $M(\Theta)$ is not, in general, diagonal the right hand side is a coupled system, that is, if a disturbance on any joint occurs, all other joints will have errors too.

Implementations of such extended control laws demands for large CPU-power. Inverse kinematics, kinematics, Jacobians, inverse Jacobians, dynamic models, adaptation mechanisms, feedback control laws, etc, have to be computed real time. A distribution of CPU-power is inevitable in such cases.

2.4 Conclusions

This chapter discussed a number of fundamental aspects of robots in general and industrial manipulators in particular. Summarizing, the following conclusions can be drawn,

- Forward kinematics is a tool to calculate the end-effector position and orientation as a function of the joint angles. This study is useful in kinematic simulations, but less applicable in task planning of robotic manipulators.

- Inverse kinematics is not a tool in itself. It only defines the problem of calculating the joint angles as a function of a desired set of end-effector positions and orientations. Generally, the inverse kinematics cannot be solved in closed form. In certain cases, especially if the manipulator lay-out is simple, it is possible to compute the inverse kinematics in a closed form, either analytically, geometrically or by combinations of these methods. However, in many cases numerical algorithms have to be developed to solve the problem, which results in less accuracy and, in certain cases, numerical stability problems.

- Dynamics of manipulators are described with two generally used formulations: Newton-Euler formulation, using Newton's law and the Lagrange formulation using system energy principle. Both methods are equivalent and result in a closed form solution of the manipulators dynamics.

- In control of industrial manipulators, usually no dynamic model is used, because it is computationally very expensive. Simple PID-based feedback controllers have to perform a stable and accurate task execution. A drawback of using this strategy is that the manipulators have to operate at reduced speed.

- Extending the control model with model-based descriptions results in a high demand on computational power, because the dynamic model has to be evaluated real time. Multitasking of computations could be a solution to this problem.

Chapter 3

Network as Real Time Control Medium

This chapter discusses the implementation of control applications through a local Ether-network. For this application the real time software VxWorks is used. An introduction is given in Section 3.1. Section 3.2 describes some features of VxWorks and the development for ix86 CPU targets. The chapter is concluded in Section 3.3.

3.1 Real-time Networking: An Introduction

The applications of network-based computer systems have been very divers and useful. However, they can be extended if distributed real time applications can be realized with the use of such networks.

Suppose you are faced with the task to implement a model-based control algorithm in a stand-alone computer (for example a PC). This machine has to control a strongly non-linear mechanic system. The PC has to compute the non-linear model and the control law and send actuator information to, and read sensor information from the mechanical system. Moreover, it has to display user-defined information on a graphical userinterface. All of this has to be performed at a real time basis. Another task can be to control, let's say, three mechanical devices, which have to co-operate with each other. The control overhead may be simple, but most of the actuators have to be activated simultaneously. Besides, reading and interpreting sensor information has to take place simultaneously. Moreover, the planning of the synchronized tasks has to be stored somewhere and be easily accessible.

These examples illustrate that many modern control applications are distributed or can better be distributed due to physical limitations of stand-alone computer systems. In these cases there is a need for several processors, terminals, I/O-handling and data-handling devices, which synchronously co-operate with each other to manage a real time control task. In other words, there is a need for connected computer systems of which the application can use as many as necessary.

Computer network systems provide this scaled distribution of access to computer power.

UNIX operating systems have become the standard for using computer network systems. UNIX already provides the important features for portable software development such as a graphical interface to the network (X windows for example), network programming via TCP/IP and NFS, and multiple processors and users. The recent IEEE-1003.4 Posix real time extensions seem to make UNIX systems, as network nodes, almost complete. On the other hand, platforms like ix86, SPARC, MIPS, CPU32, i960 and PowerPC all are different nodes in many networks, nowadays. In distributed real time applications it would be helpful to standardize the operating software. This will ease the co-operation between different platforms. The cross-development, real-time operating software VxWorks makes this possible. Even for more platforms than listed above.

3.2 VxWorks for PC Targets: An Application

VxWorks in combination with UNIX forms a complete, integrated development and operating environment. The UNIX system is used for software development and non-time-critical components of an application, while VxWorks is used for testing, debugging and running real time applications. VxWorks can be networked with any operating system that has TCP/IP networking facilities. UNIX workstations can be used to edit, compile, link and store real time code, that will be used in a VxWorks environment afterwards. The basic architecture of a networked [1] VxWorks application is formed by a *host-target pair.* For this research such a combination is a **UNIX Workstation – Pentium-S PC** pair, respectively. Once a hardware connection between these two machines through the Ethernet is made, the target can be booted (using a disk boot-partition, contrary to most other architectures that use a boot-ROM) by VxWorks and configured so that the pair can be established using the Ethernet addresses of these two machines. After that it is possible to run real time code on the VxWorks target. This real time code can be written very economically, since it doesn't have to be linked with the VxWorks system libraries or with each other. While loading object modules (unlinked C-codes), the external symbol references are solved dynamically using the symbol tables already loaded in the VxWorks target by default (during the booting-procedure) or by previously loaded modules. This method prevents the application programs to be unnecessarily large, which is desirable running these programs in real time applications.

The previous section motivates the need for distributed software development. Every hardware part of the application deals with its own real time or non- real time job. Of course, there is also a strong need for communication between jobs, since they have to be synchronized in most cases. This idea of distributed jobs, running simultaneously, is well known as **multitasking and intertask communication.** VxWorks provides an extended set of tools for multitasking and intertasking, regardless of the location or CPU-type of the computers running the different tasks. To use VxWorks in a networked application one can use the UNIX-compatible sockets, which make it possible to communicate between tasks running on different nodes in the Ethernet. A task can be related to one or more

[1]VxWorks can also be used in stand-alone machines as a real time operating system

sockets. The information read from or written to these sockets may come from or go to different CPU's or the same CPU running different tasks. In this way it is easy to collect the information of all distributed tasks in a central CPU which has to combine it to control a mechanical device. Backplane CPU's can also collect this distibuted information to store it in a RAM or NFS for evaluation or use it in a graphical userinterface. These two non-real time processes may take much time and would be a real burden if queued in a real time run. The distributive character of the network can be very useful in these situations.

Another great advantage is the fact that VxWorks is completely UNIX-compatible. In other words, one can use the same UNIX standard in all types of CPU's booted by VxWorks. In the case of a PC-target this has several advantages. In the application of this research, the serial line of the PC is used to give commands to the robot via the controller (see Section 1.2). This serial line is a standard device in the VxWorks development for the PC-target. To have access to devices in an UNIX environment, normally, the following procedure principles are required:

```
devicefd = open("device_name",access_flag,option);
......
read(devicefd,buffer_ptr,nBytes);
......
write(devicefd,buffer_ptr,nBytes);
......
close(devicefd);
```

in which `devicefd` is a handle label for the particular `"device_name"`; `access_flag` can be `O_RDONLY`, `O_WRONLY` or `O_RDWR` corresponding to read only, write only or read/write access; `buffer_ptr` is a pointer to the information buffer to be read or written.

This simple type of device access can also be used for the serial line in the PC. The overhead of writing a difficult piece of code to access the serial port is included in VxWorks and can be handled in a standard UNIX form. In other words, it is possible to simply `write` and `read` character-buffers to and from the serial port of the PC, without any need for access procedures, specific for PC serial lines (RS-232-C).

PC's used for real time applications are equipped with A/D and/or D/A cards. These devices are non-standard and require a driver routine written by the user. After such a driver is written it can be put dynamically in the list of available device drivers. Access to these not standard devices can then be obtained in the same way as described above. Continuing these procedures for every device used in a particular real time control application, makes it possible to develop a complete standardized operating system for this application. This operating system can be easily transformed to another, since every module, whether a device driver or task, is linked with the system at run time. This is desirable, since the operating system kernel doesn't have to be rebuilt every time a real time control application is changed or extended. In other words, the VxWorks operating system is an **open** system in many fields: real time code development, user-defined device access, network programming, kernel development and running applications.

As mentioned before, VxWorks is used to control a 6 D.O.F. robot through a local Ethernet. Because the axes of this robot are controlled by a hardware control device (see Figure 1.1), only commands have to be send through to Ethernet. This makes the control overhead rather limited. Nevertheless, the experiment illustrates many features of the use of VxWorks and the network. Rather than giving an obscure listing of source codes of the Ethernet control application, in Figure 3.1 the steps in such an application are depicted. The hardware elements are emphasized in this illustration to make the steps more obvious to the reader. Creation of sockets (connection to Ethernet to read and write character-

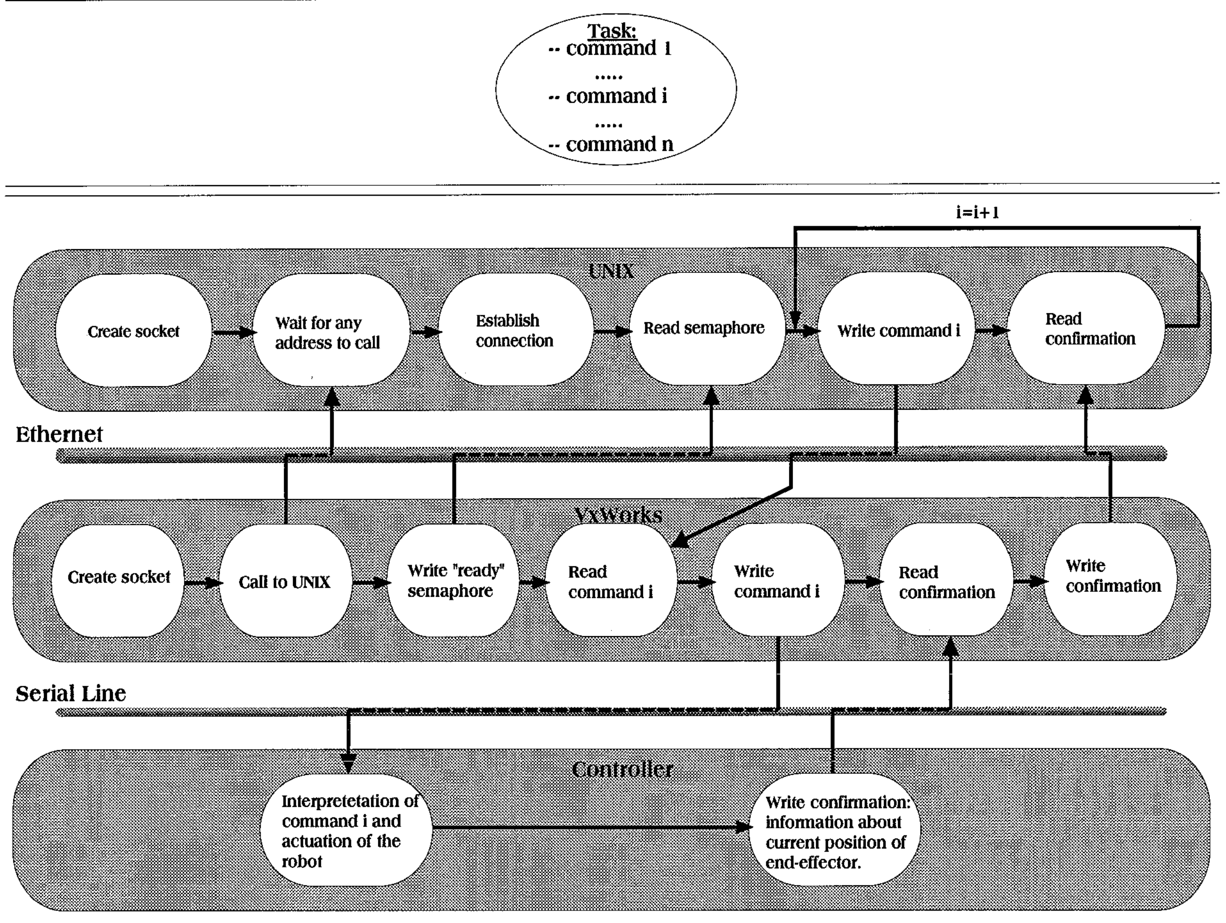


Figure 3.1: **Steps in the Ethernet control of the Move Master**

based data streams) and reading/writing from and to these sockets is executed by TCP/IP network programming conventions. In the described implementation it is obvious that the UNIX Workstation is the leading component in giving control commands. The Workstation is the first to give the command and the last to confirm whether this command is actually executed after which a new command can be given. The VxWorks machine (PC) is only a gate-way for writing commands to the robot. If the control of the axes had to be done by a real time software control algorithm, then this could be distibuted over the UNIX Workstation and VxWorks. For example, the inverse dynamics could be computed by the Workstation and the feedback-control of the axes as well as the I/O with the robot by the PC.

3.3 Conclusions

In this chapter arguments are given to use Ethernet as real time control medium. The UNIX-compatible real time software operating system VxWorks is narrated and its usefulness in such Ethernet applications is emphasized. The general structure of implementing tasks within a host-target pair is also discussed. From this chapter the following conclusions can be drawn.

- Using Ethernet opens the potential to control distibuted or complicated systems at a real time basis.
- VxWorks provides a fully UNIX-compatible real time operating system, which can be used across different CPU platforms.
- VxWorks is a scalable and flexible open operating system. The developer can use as much of it as necessary. Moreover, it can be extended dynamically without rebuilding the operating kernel.
- Standard Internet protocols, like TCP/IP, can be used to program the communication through Ethernet.
- A complete set of tasks running on different or the same CPU's can easily be synchronized by an extended set of multitasking and intertasking tools.
- Non-time-critical components can be set aside from the real time control tasks.

Chapter 4

Implementations and Experiments

The previous chapter discussed some features of the VxWorks operating system together with methodologies for implementations in real time applications. This chapter narrates about the implementation of a graphical userinterface and the programming of two tasks. In Section 4.1 a pick-and-place task of an object is discussed. Section 4.2 discusses the userinterface. Section 4.3 describes a second pick-and-place experiment. In this experiment the robot is equipped with a laser range sensor near the end-effector (gripper) for detecting the object. The chapter is concluded in Section 4.4.

4.1 Remote Control of Pick-and-Place task

Many robot tasks in industrial applications are simple pick-and-place tasks. Industrial robots can be quite easily programmed by giving a list of commands to perform such tasks. The list of available commands is very divers and scopes nearly all possible configurations and movements of the manipulator. For example, it is possible to command a particular joint with a specified modification of the joint angle (relative to previous position). This can be executed with a specified speed and interpolation technique. Speed settings modify the acceleration and the maximum speed when moving with a constant speed. Interpolation settings modify the method of moving from position A to B, either linear, circular or smooth interpolation. Such interpolations are generated by the controller itself and are based on parabolic blends near the interpolation points. Figure 4.1 illustrates this interpolation. Timer settings can also be used to delay executions of commands. This is useful when the manipulator is part of a larger co-operating system in which tasks have to be synchornized. The most frequently used command types are those to specify the movements of the center of the end-effector. After all, this part of the manipulator has to execute the tasks, while the other links have to move in that manner making this possible. The controller uses inverse kinematics to compute the joint movements as function of time, taking into account the interpolation method. The user simply provides a short command called "`MP`" with additional numerical values specifying the desired final position of the end-effector. It is obvious that addition of via point specifications results in a movement from A to B simular

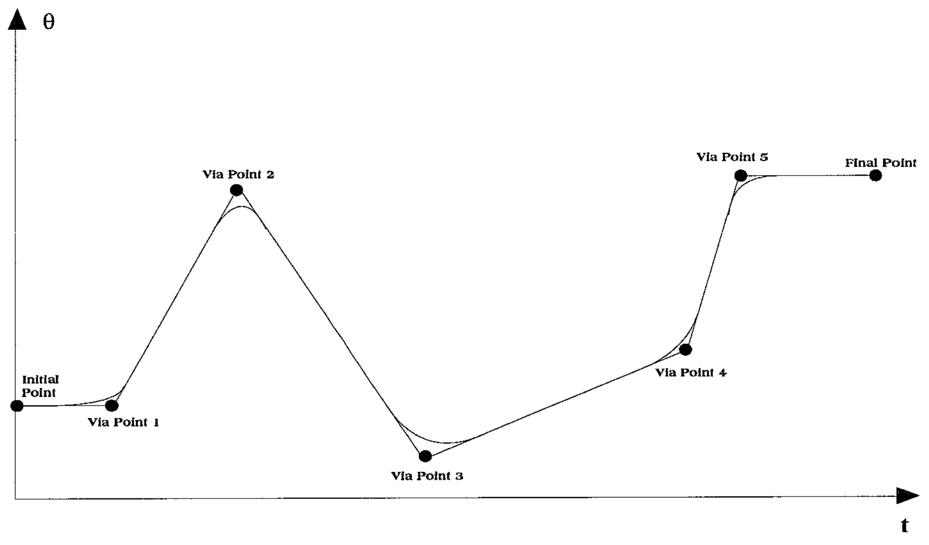


Figure 4.1: **Parabolic blends smooth a path with via points**

to the way illustrated in Figure 4.1. In the considered pick-and-place task this MP command is used to move from one point to another using a linear interpolation method. Figure 4.2 illustrates the implemented pick-and-place task. The task is implemented in the UNIX

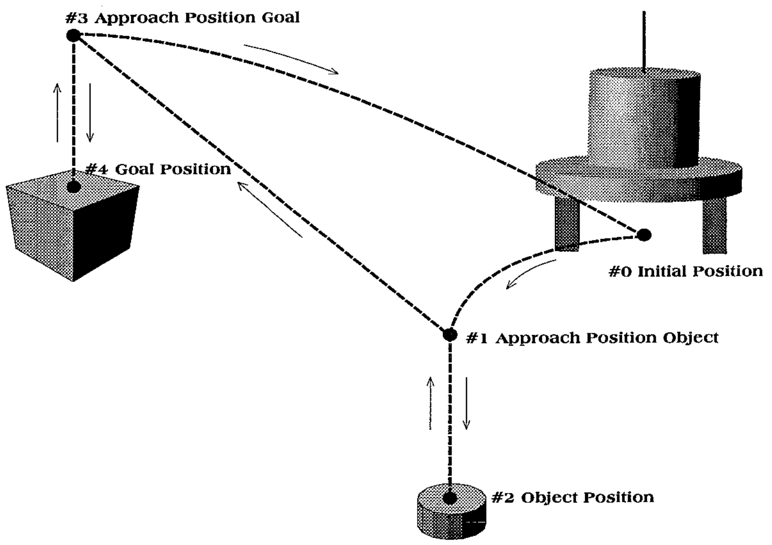


Figure 4.2: **Positons of the Pick-and-Place task**

host which sends the commands one after another to the PC. The PC writes them to the controller. Before a new command can be send, a message from the controller has to be send telling the command list that the previous command is executed and a new command is expected. This semaphore is executed by sending the command “WH” to the controller. This command, an abbreviation of “Where?”, asks for the current position and orientation of the end-effector as well as the three posture flags of the complete manipulator (see

Appendix B). Because this information can only be given if the manipulator is in a (new) fixed position, the command list "knows" that the next command can be given. This task is succesfully executed: the commands from the UNIX host arrived to the controller and the controller could send the acknowledgements back to the UNIX. Evaluating these acknowledgements showed that the positions detected by the joint-sensors were the same as the positions given by the `MP` commands. In Appendix C the source codes of this experiment are given.

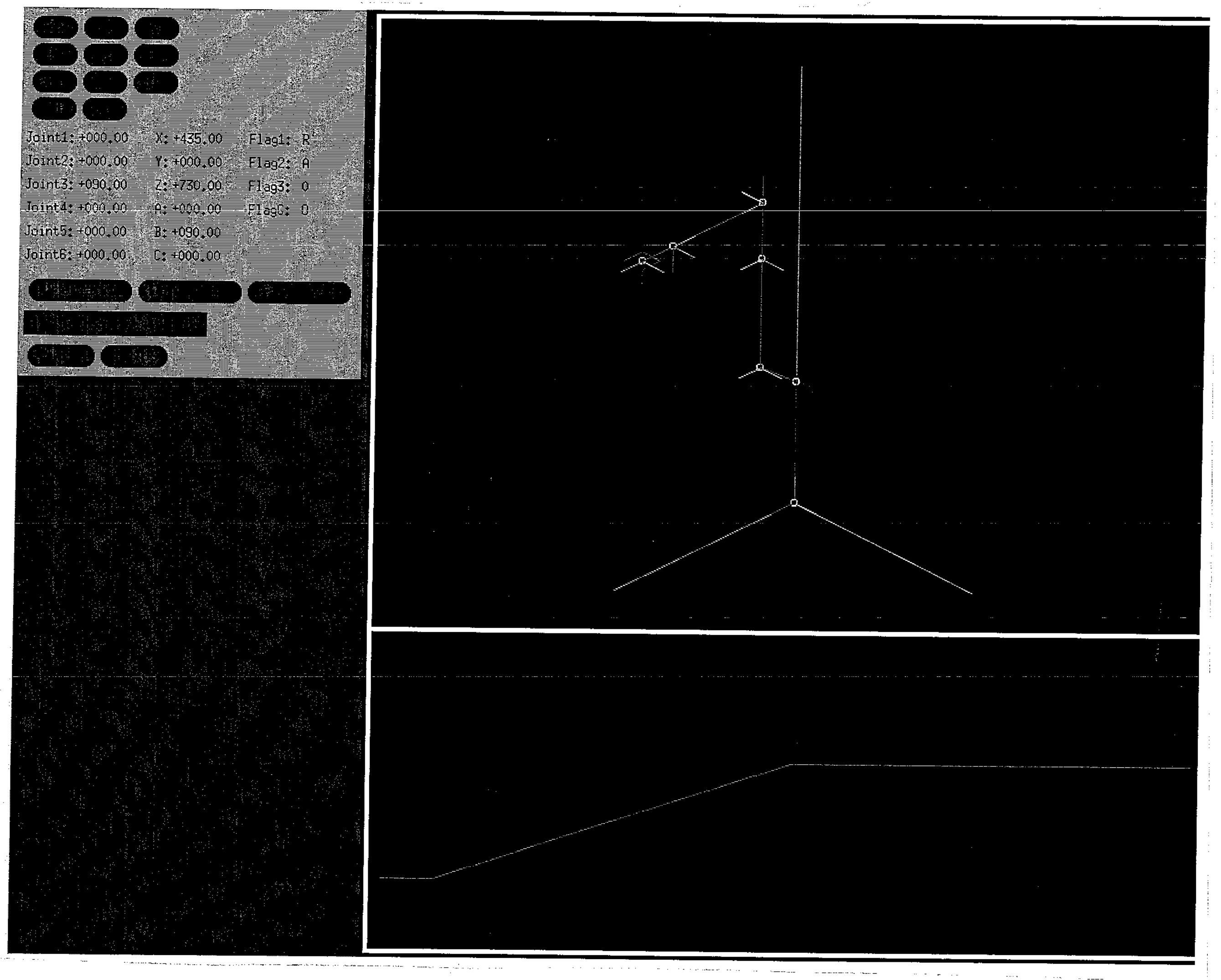
4.2 What's happening?

Before any tasks will be executed in reality, it is wise to test them on a simulator. Moreover, since the robot is controlled remotely through the Ethernet, it is convenient to have update information of the state of the manipulator. These two items can best be shown by a graphical userinterface. Such an interactive tool makes it possible to easily program, simulate and execute tasks together with a constant updating of manipulator states. The design of an userinterface for the Move Master is discussed in this section.

Since displaying just numerical values is not a very clear way of keeping the user up-to-date, a graphically displayed kinematic model of the robot is the best way to show manipulator information. To implement tasks, a surveyable graphical control panel is developed. Other control panels can start the simulation of the task or send the task to the robot. Movements of the robot are shown by adapting the kinematic model, displayed as a wire frame, and sensor information, displayed by numerical values.

This userinterface is developed with the X Window library. This library has a complete set of standard window-components: buttons, canvas-windows, sliders, gauges, etc. This library can be used in the C programming language, which makes coupling with the real time task execution codes (VxWorks) very easy. The most important ingredient of programming with X Window library is the *callback*. This is a piece of code describing the handling of the user's interaction with the userinterface. For example, if the user pushes a function button, the callback code has to deal with this action and make sure that the demand of the user will be executed.

Figure 4.3 illustrates the userinterface as it is currently developed. Many modifications and augmentations still have to be made. The upper left panel holds the command buttons, a numerical value display (joint angles, position and orientation of the end-effector and the posture flags together with a gripper open/close flag). Beneath the numerical value display an `Orientation` button can change the angle of view to the 2D map of the 3D manipulator model, a `Demo` button shows a demonstration of a task in the canvas window and the `Task 1` button executes the task described in the previous section. The last set of buttons can change the angle of every joint seperately by incrementing the joint chosen angle with `PLUS` or `MINUS`. The canvas window at the upper right shows the wire model of the Move Master. Commands given at the control panel will change this wire model. The canvas window at the lower right shows sensor information of the sensor fixed at the gripper of the manipulator. The application of this sensor will be discussed in the next section.



```
Joint1: +000.00   X: +435.00   Flag1: R
Joint2: +000.00   Y: +000.00   Flag2: A
Joint3: +090.00   Z: +730.00   Flag3: 0
Joint4: +000.00   A: +000.00   FlagC: 0
Joint5: +000.00   B: +090.00
Joint6: +000.00   C: +000.00
```

Figure 4.3: **Preliminary design of the userinterface**

4.3 A little intelligence...

The pick-and-place task of Section 4.1 is only succesful if the object is placed at a predefined location. In industrial applications this is usually the case. On the other hand, if the robot has to pick objects from a conveyor belt which does not have frames to hold objects, it might be useful to equip the robot-gripper with some sort of vision-based sensor, like a CCD Camera or a laser range sensor. In this research a laser range sensor is used. This sensor detects differences in height. The working area of the laser is just a square millimeter spot and therefore it is necessary to perform more than one scan in order to localize the (center of) the object. A geometrical algorithm has to control the movements of the end-effector in such a way that the dimensions, position and orientation of the object can be found. After that, grasping and placing the object can be done in the same way as described in

Section 4.1.

The experiment is done with an arbitrary rectangular object. This object has to be localized somewhere in an area known by the searching algorithm. Figure 4.4 depicts this searching problem. Since the shape of the object is rather elementary, the searching

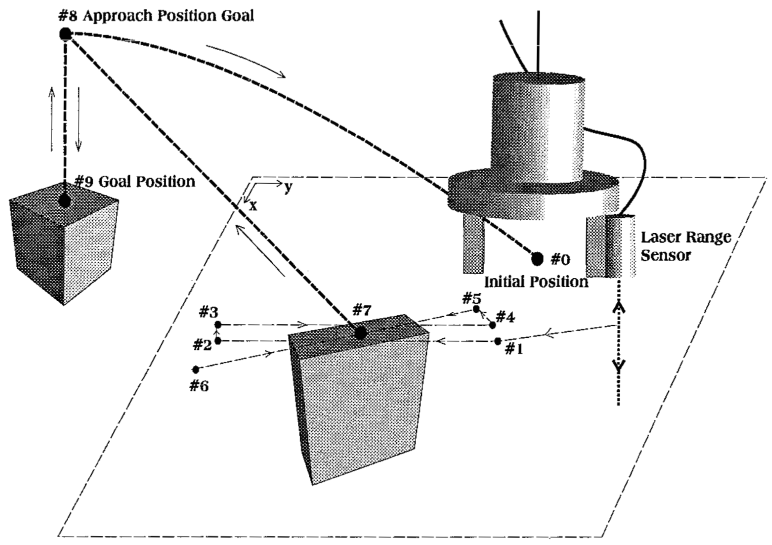


Figure 4.4: **The search-and-grasp problem**

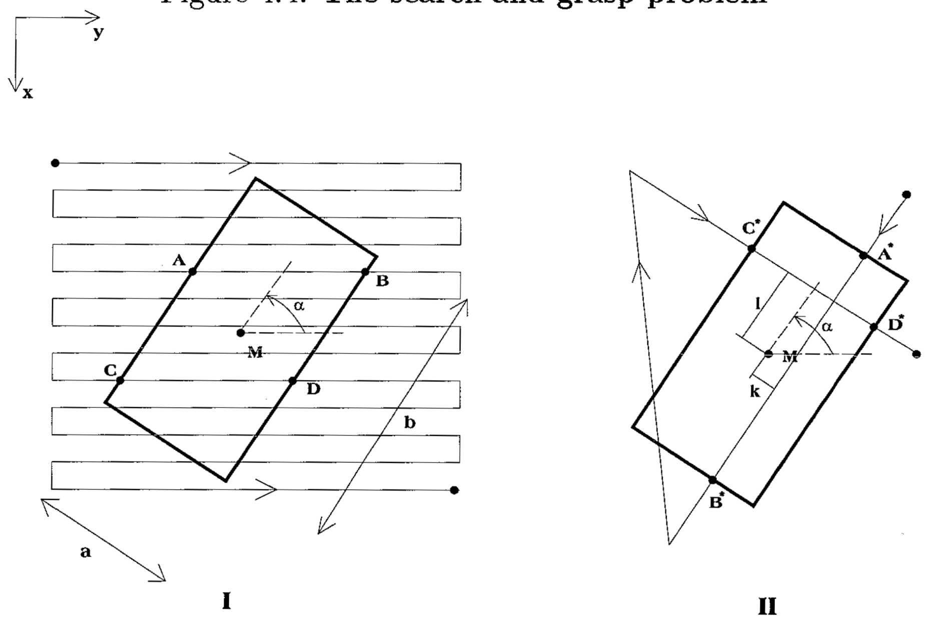


Figure 4.5: **Two phases in precise scanning**

problem can be executed with a straightforward algorithm. First the sensor has to do a rough scan within the predefined area. During this scan he will roughly detect the object somewhere. In that neighbourhood, the sensor has to scan more precisely. With this last scan it is possible to calculate the position and orientation of the object. The two step procedure is illustrated in Figure 4.5: firstly, multiple scans along the y-direction of the predefined area (Figure 4.4), which will result in knowledge of the orientation of the rectangular object; secondly two more scans across the object, parallel with the main sides of the rectangular, which results in knowledge of the position of the center point **M** and the dimensions of the rectangular.

Scan I results in four points of intersection: A, B, C and D. If the lines lines AC and BD. are parallel, they are points of two parallel sides of the rectangular. From this, the orientation α in the xy plane, can be computed as

$$\sin(\alpha) = \frac{|C_x - A_x|}{\sqrt{(C_x - A_x)^2 + (C_y - A_y)^2}} \quad \text{for } 0^o \leq \alpha \leq 180^o \tag{4.1}$$

In Scan II, a scan in the direction of α and a scan perpendicular to this direction are made. This results in four points of intersection. Combining the coordinates of these points results in the following equations

$$\begin{aligned} a &= |C^*D^*| \\ b &= |A^*B^*| \\ B_x^* - \frac{1}{2}b\sin(\alpha) &= M_x + k\cos(\alpha) \\ B_y^* + \frac{1}{2}b\cos(\alpha) &= M_y + k\sin(\alpha) \\ A_x^* + \frac{1}{2}a\cos(\alpha) &= M_x - l\sin(\alpha) \\ A_y^* + \frac{1}{2}a\sin(\alpha) &= M_y + l\cos(\alpha) \end{aligned} \tag{4.2}$$

from which the unknown parameters M_x, M_y, k and l can be obtained by

$$\begin{bmatrix} 1 & 0 & \cos(\alpha) & 0 \\ 0 & 1 & \sin(\alpha) & 0 \\ 1 & 0 & 0 & -\sin(\alpha) \\ 0 & 1 & 0 & \cos(\alpha) \end{bmatrix} \begin{bmatrix} M_x \\ M_y \\ k \\ l \end{bmatrix} = \begin{bmatrix} B_x^* - \frac{1}{2}b\sin(\alpha) \\ B_y^* + \frac{1}{2}b\cos(\alpha) \\ A_x^* + \frac{1}{2}a\cos(\alpha) \\ A_y^* + \frac{1}{2}a\sin(\alpha) \end{bmatrix} = ORIENT \cdot \tilde{u} = \tilde{n}$$

$$\Rightarrow \tilde{u} = ORIENT^{-1} \cdot \tilde{n} \tag{4.3}$$

The inverse of $ORIENT$ can be computed in closed form, which results in a numerically stable solution of $\tilde{u}$. Implementation of this scheme in a task list results in a succesful grasping of the object after scanning. The algorithm is tested in the stand-alone PC. In the future it will be tested and augmented in the remote control procedure, discussed in earlier parts of this report. The sensor information can be send real time through the Ethernet after which it can be displayed on the userinterface (see Section 4.2). Due to limited time for this research, I could not finish the implementation of the search algorithm and test the grasping of the arbitrary placed object with the Ethernet control. Since the sensor information is tested to be send through the Ethernet and the procedure to give commands through the Ethernet is already tested (see Section 4.1), this procedure should be able to perform successfully.

The pick-and-place task of Section 4.1 will be provided with a little more intelligence such that the robot can find the object by itself without having to place the object at a defined location. This procedure can be fastened and augmented to cases in which the object is moving by using a camera instead of a laser range sensor. Such an experiment is described in [14]. If the complete robot is given an extra degree of freedom (as will be the case in the future), the possibilities will be even more extensive.

4.4 Conclusions

In this chapter some experiments and implementations of Ethernet control and the grasping of objects are described. Although the available time made it impossible to finish the experiments and implementions, the obtained results are satisfactory and emphasize the potential of future research. From this chapter the following conclusions can be drawn.

- Using VxWorks as an operating system for the PC-target, executing a pick-and-place task with the 6R manipulator through the Ethernet, is quite successful.
- An userinterface, written with the X Windows library, is implemented on the UNIX-host. Commands can be simulated and send to the robot by this user-panel.
- The manipulator can grasp arbitrary placed rectangular objects with unknown dimensions if a laser range sensor is attached near the gripper. A geometrical algorithm controls the movements of the end-effector.
- Grasping objects can be advanced by using a camera near the gripper. Also moving objects can be grasped, especially if the robot is mobilized.

Chapter 5

Conclusions and Recommendations

In this chapter conclusions are drawn with respect to the research objectives as described in Section 1.2. Moreover, some recommendations for future investigation are given.

- The inclusion of the main kinematic and dynamic aspects in a model is straightforward but results in complicated formulations. Real time model-based control demands for distributed CPU-power.
- Networking, cross-development software and standard UNIX-systems provide the opportunity to use the (local or global) computer network system as a medium for real time control.
- Real time and non- real time jobs can be simultaneously distributed over different CPU platforms.
- Experiments showed that the Ethernet pair: UNIX-PC results in a succesful execution of a pick-and-place task of the 6R manipulator remotely controlled by the UNIX Workstation.
- A graphical userinterface is developed in the UNIX-host. Commands can be given from this control panel and the movements of the robot can be illustrated by a kinematic wire model .
- The robot gains a limited amount of intelligence, when equipped with a laser range sensor. Rectangular shaped, arbitrary placed objects can be picked up after a number of scans, under control of a geometrical algorithm.

The remote control idea is illustrated in this report. The pick-and-place experiment through the net was succesful. The use of Ethernet was rather simple: only commands and small batches of numerical information had to be transported through the net. Further investigation should be done to the use of the network in more time-critical control tasks, with a constant transport of control and sensor data. Because of the non-deterministic behaviour of the Ethernet transport (collision detection and retry procedures) and (small) time-delays in transport, this might give problems, which have to be solved by extended testing and measuring.

The vision-based sensoring of a static or moving target object can be extended by using a camera near the gripper and by mobilization of the robot. Together with the use of network controll, this requires further research.

Bibliography

[1] Move Master user Manual, Mutsubishi Electric Corp., Model RV-E2.

[2] Wind River Systems, VxWorks 5.2, Programmer's Guide

[3] Wind River Systems, VxWorks 5.2, Reference Manual

[4] Bloomer, J., "Power Programming with RPC," *O'Reilly & Associates, Inc.*, 1991.

[5] Craig, J.J., "Introduction to Robotics, Mechanics & Control," *Addison-Wesley Publishing Company*, 1986.

[6] Kernighan, B.W. and Ritchie, D.M., "The C programming Language," *Prentice-Hall, Englewood Cliffs, NJ*, 1987.

[7] Manocha, D. and Canny, J.F., "Efficient Inverse Kinematics of General 6R Manipulators," *IEEE Trans. Robotics Automation*, Vol. 10(5):648-657, 1994.

[8] McKerrow, P.J., "Introduction to Robotics," *Addison-Wesley Publishing Company*, 1991.

[9] Nicolson, E.J., "Standardizing I/O for Mechatronic Systems (SIOMS) using Real Time UNIX Device Drivers," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, Vol.4:3489-3494, 1994.

[10] Paul, R.P., "Robot Manipulators: Mathematics, Programming, and Control," *The MIT Press, Cambridge, MA*, 1981.

[11] Pieper, D., "The Kinematics of Manipulators Under Computer Control," Ph.D. Thesis, Stanford University, 1968.

[12] Pardo-Castellote, G. and Schneider, S., "The Network Data Delivery Service: Real-Time Data Connectivity for Distributed Control Applications," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, Vol.4:2870-2876, 1994.

[13] Skowronski, J.M., "Control Dynamics of Robotic Manipulators," *Academic Press, Inc.*, 1986.

[14] Smith, C.E. and Papanikolopoulos, N.P., "Grasping of Static and Moving Objects Using a Vision-Based Control Approach", *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, Vol.1:329-334, 1995.

[15] Stone, H.W., "Kinematic Modeling, Identification, and Control of Robotic Manipulators," *Kluwer Academic Publishers*, 1987.

[16] Williams, T., "Fiber network supports distributed real-time systems," *Computer Design*, 29(17):60-62, Sept. 1990.

[17] Williams, T., "Real-time Unix develops multiprocessing muscle," *Computer Design*, 30(5):26-30, March 1991.

[18] Wells, D.A., "Theory and Problems of Lagrangian Dynamics," *Schaum*, 1967.

[19] Young, D.A. and Pew, J.A., "The X window system Programming and Applications with Xt OPEN LOOK Edition," *Prentice Hall*, 1992.

Appendix A

Frame Transformation Matrix

Consider the two arbitrary Cartesian coordinate frames $\mathcal{T}_1$ and $\mathcal{T}_2$ in Figure A.1. The

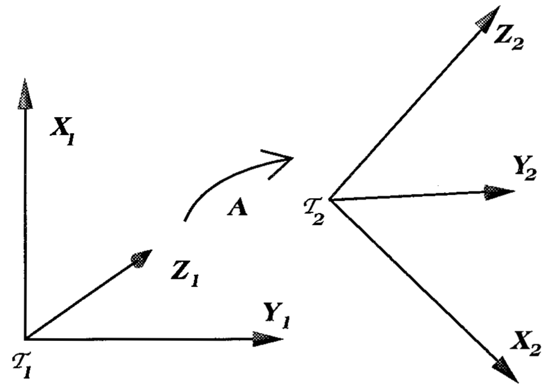


Figure A.1: **Transformation of an arbitrary frame to an other**

matrix A transforms frame $\mathcal{T}_1$ into frame $\mathcal{T}_2$. This transformation can be decomposed into a combination of *primitive* transformations (see Paul [10]):

$$\begin{aligned}
\text{Trans}(x,0,0) &\Rightarrow \text{Translate } x \text{ units along the } X \text{ axis}\\
\text{Trans}(0,y,0) &\Rightarrow \text{Translate } y \text{ units along the } Y\text{axis}\\
\text{Trans}(0,0,z) &\Rightarrow \text{Translate } z \text{ units along the } Z\text{axis}\\
\text{Rot}(x,\theta) &\Rightarrow \text{Rotate } \theta \text{ degrees about the } X \text{ axis}\\
\text{Rot}(y,\theta) &\Rightarrow \text{Rotate } \theta \text{ degrees about the } Y\text{axis}\\
\text{Rot}(z,\theta) &\Rightarrow \text{Rotate } \theta \text{ degrees about the } Z\text{axis}
\end{aligned}$$

These six primitive homogeneous transformations matrices are

$$\text{Trans}(x,0,0) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
\mathrm{Trans}(0,y,0) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\mathrm{Trans}(0,0,z) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\mathrm{Rot}(x,\theta) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\mathrm{Rot}(y,\theta) &= \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\mathrm{Rot}(z,\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{A.1}$$

Applying (A.1), the Denavit-Hartenberg link coordinate frames, $\mathcal{T}_i$ for $i = 1, \ldots, n$, are specified such that the *forward transformation matrices* A_i are prescribed by

$$A_i = \mathrm{Rot}(z,\theta_i)\,\mathrm{Trans}(0,0,d_i)\,\mathrm{Trans}(a_i,0,0)\,\mathrm{Rot}(x,\alpha_i) \tag{A.2}$$

which results in

$$A_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

Appendix B

Inverse Kinematics of the 6R Move Master

The user prescribes a set of positions, orientations and three posture-flags for the end-effector relative to user-defined base frame. The problem is to find the six joint angles which correspond to this prescribed information, summarized in the vector $[x_e, y_e, z_e, \alpha_e, \beta_e, \gamma_e, flag_1, flag_2, flag_3]^T$. The set of non-linear equations to solve are:

$$^0T_e = \begin{bmatrix} ^0R_e & ^0P_e \\ 0_{(1x3)} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_e \\ r_{21} & r_{22} & r_{23} & y_e \\ r_{31} & r_{32} & r_{33} & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^0T_1(\theta_1)\, {}^1T_2(\theta_2)\, {}^2T_3(\theta_3)\, {}^3T_4(\theta_4)\, {}^4T_5(\theta_5)\, {}^5T_6(\theta_6)\, {}^6T_e$$

in which

$$^0R_e = \begin{bmatrix} \cos\alpha_e \cos\beta_e & \cos\alpha_e \sin\beta_e \sin\gamma_e - \sin\alpha_e \cos\gamma_e & \cos\alpha_e \sin\beta_e \cos\gamma_e + \sin\alpha_e \sin\gamma_e \\ \sin\alpha_e \cos\beta_e & \sin\alpha_e \sin\beta_e \sin\gamma_e + \cos\alpha_e \cos\gamma_e & \sin\alpha_e \sin\beta_e \cos\gamma_e - \cos\alpha_e \sin\gamma_e \\ -\sin\beta_e & \cos\beta_e \sin\gamma_e & \cos\beta_e \cos\gamma_e \end{bmatrix} \tag{B.1}$$

and $^{i-1}T_i$ are consecutive link frame transformations (Appendix A).

Figure B.1 illustrates the three flags. The position of joint five can be computed by

$$^0P_5 \;=\; ^0P_e + {}^0R_e\, {}^eP_5 = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} + \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -(L_6 + L_7) \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} ^0P_5{}^x \\ ^0P_5{}^y \\ ^0P_5{}^z \end{bmatrix} = \begin{bmatrix} x_e - (L_6 + L_7)\, r_{13} \\ y_e - (L_6 + L_7)\, r_{23} \\ z_e - (L_6 + L_7)\, r_{33} \end{bmatrix} \tag{B.2}$$

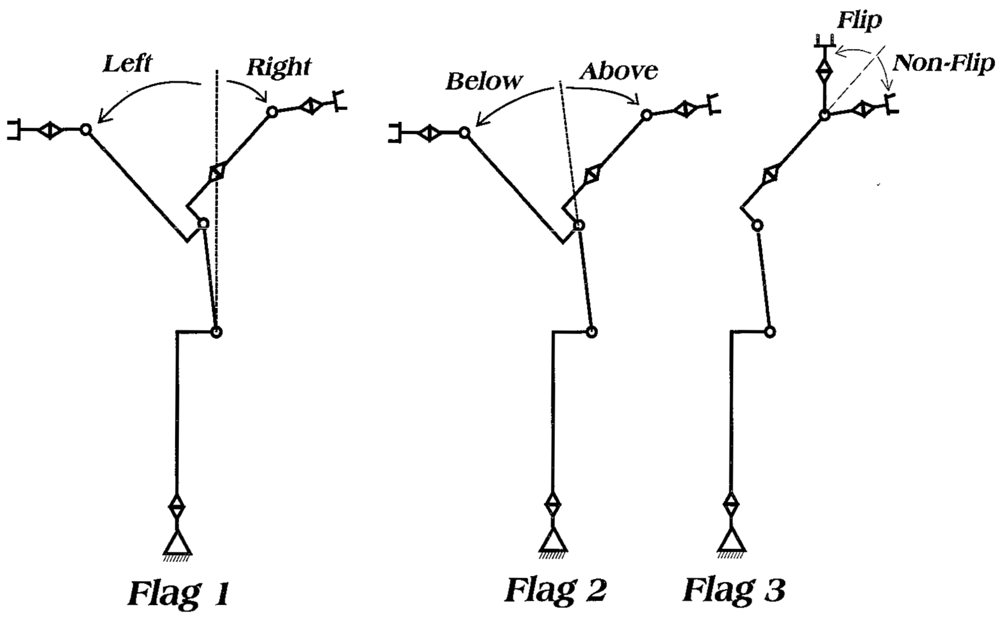


Figure B.1: **Illustration of the three posture flags**

From Figure B.2 it is obvious that θ_1 can be calculated as

$$\theta_1 = \text{Atan2}({}^0P_5{}^y, {}^0P_5{}^x) \text{ if flag 1 is Right} \tag{B.3}$$
$$\theta_1 = \text{Atan2}(-{}^0P_5{}^y, -{}^0P_5{}^x) \text{ if flag 1 is Left} \tag{B.4}$$

For the solution of θ_3 the following relations are valid (see Figure B.3) :

$$\begin{aligned}
{}^1P_5{}^x &= {}^0P_5{}^x \cos(\theta_1) + {}^0P_5{}^y \sin(\theta_1) \\
k_1 &= {}^1P_5{}^x - L_2 \\
k_2 &= {}^0P_5{}^z - L_1 \\
k_3 &= \sqrt{L_4 + L_5} \\
\alpha &= \text{Atan}(L_4, L_5) \\
cos(\theta_3{}^*) &= \frac{(k_1{}^2 + k_2{}^2 - L_3{}^2 - k_3{}^2)}{2\ L_3\ k_3} \\
sin(\theta_3{}^*) &= \sqrt{1 - cos^2(\theta_3{}^*)} \\
\theta_3{}^* &= \text{Atan2}(sin(\theta_3{}^*), \cos(\theta_3{}^*)) \text{ if flag 2 is Above} \\
\theta_3{}^* &= \text{Atan2}(-sin(\theta_3{}^*), \cos(\theta_3{}^*)) \text{ if flag 2 is Below} \\
\theta_3 &= \theta_3{}^* + \alpha
\end{aligned} \tag{B.5}$$

For the solution of θ_2 the following relations are valid (see Figure B.4) :

$$\beta = \text{Atan2}(k_2, k_1)$$

$$\gamma = \mathrm{Acos}\Big(\frac{{k_1}^2 + {k_2}^2 + {L_3}^2 - {k_3}^2}{2\, L_3 \sqrt{{k_1}^2 + {k_2}^2}}\Big)$$

$$\theta_2 = -\beta - \gamma \text{ if } {\theta_3}^* > 0 \qquad \text{(B.6)}$$

$$\theta_2 = -\beta + \gamma \text{ if } {\theta_3}^* < 0 \qquad \text{(B.7)}$$

The last three joint angles 4,5 and 6 can be solved easily if the corresponding three frames are moved to the origin of the 5th joint frame. Kinematically, this situation is identical to the real situation. The advantage of this is that the last three links and joints can be seen as an so called **Euler Wrist**, which rotates the end-effector by **Z-Y-Z Euler angles** relative to the third frame. The corresponding rotation matrix can be described by:

$${}^3R_6 = \begin{bmatrix} \cos\theta_4\,\cos\theta_5\,\cos\theta_6 - \sin\theta_4\,\sin\theta_6 & -\cos\theta_4\,\cos\theta_5\,\sin\theta_6 - \sin\theta_4\,\cos\theta_6 & \cos\theta_4\,\sin\theta_5 \\ \sin\theta_4\,\cos\theta_5\,\cos\theta_6 + \cos\theta_4\,\sin\theta_6 & -\sin\theta_4\,\cos\theta_5\,\sin\theta_6 + \cos\theta_4\,\cos\theta_6 & \sin\theta_4\,\sin\theta_5 \\ -\sin\theta_5\,\cos\theta_6 & \sin\theta_5\,\sin\theta_6 & \cos\theta_5 \end{bmatrix} \qquad \text{(B.8)}$$

${}^3R_6$ is know by:

$${}^3T_e = ({}^0T_1(\theta_1)\,{}^1T_2(\theta_2)\,{}^2T_3(\theta_3))^{-1}\,{}^0T_e = \begin{bmatrix} {}^3R_e & {}^3P_e \\ 0_{(1x3)} & 1 \end{bmatrix}$$

in which

$${}^3R_e = {}^3R_6 = \begin{bmatrix} {}^3{n_6}^x & {}^3{o_6}^x & {}^3{a_6}^x \\ {}^3{n_6}^y & {}^3{o_6}^y & {}^3{a_6}^y \\ {}^3{n_6}^z & {}^3{o_6}^z & {}^3{a_6}^z \end{bmatrix} \qquad \text{(B.9)}$$

Considering (B.8) the solutions of the last three joint angles can be computed as follows:

if ${}^3{a_6}^x \neq 0$ and ${}^3{a_6}^y \neq 0$

$$\theta_4 = \mathrm{Atan2}({}^3{a_6}^y, {}^3{a_6}^x) \qquad \text{(B.10)}$$

$$\theta_5 = \mathrm{Atan2}(\sqrt{({}^3{n_6}^z)^2 + ({}^3{o_6}^z)^2}, {}^3{a_6}^z) \qquad \text{(B.11)}$$

$$\theta_6 = \mathrm{Atan2}({}^3{o_6}^z, -{}^3{n_6}^z) \qquad \text{(B.12)}$$

choose: $0^o \leq \theta_5 \leq 180^o$
if ${}^3{a_6}^x = 0$ and ${}^3{a_6}^y = 0$

$$\theta_4 = 0 \qquad \text{(B.13)}$$

$$\theta_5 = 0 \qquad \text{(B.14)}$$

$$\theta_6 = \mathrm{Atan2}(-{}^3{o_6}^x, {}^3{n_6}^x) \qquad \text{(B.15)}$$

$\theta_5 = 180^o$ is also a case, but since this angle is out of the physical range of the Move Master, this situation doesn't have to be considered.

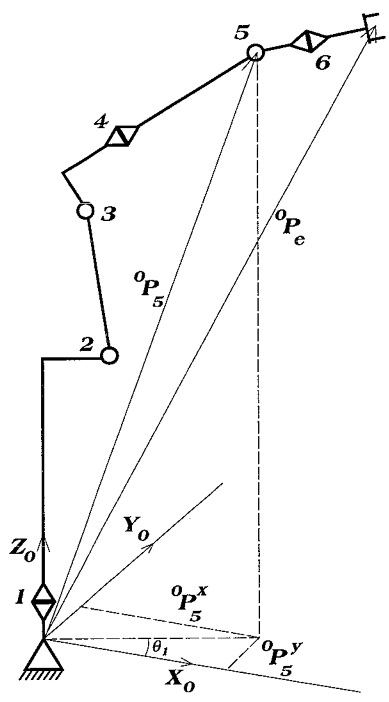


Figure B.2: **Geometrical solution of θ_1**

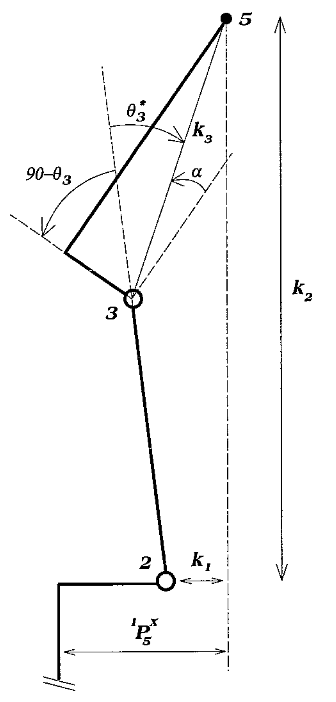


Figure B.3: **Geometrical solution of θ_3**

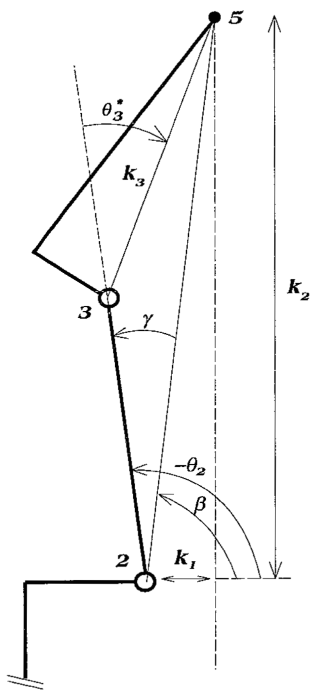


Figure B.4: **Geometrical solution of θ_2**

Appendix C

Source Code Ethernet Control

```
/* task1_host.c: pick and place task for Move Master robot. This program runs
on UNIX and communicates with its vxWorks counterpart: task1_target.c
*/


#include "task.h"

#define REC_SIZE   (1)
#define STDIO_FILENO (0)


main()
{
        struct  sockaddr_in     hostAddr;       /* address of server */
        struct  sockaddr_in     targetAddr;     /* address of client */
        int     sockAddrSize = sizeof(hostAddr);
        int     fds, fda, i;
        int     clen, nRead;
        char    character, READY[17];
        char    whatsinbuffer[64];

        /* set up local address of inet domain socket */

        memset(&hostAddr, 0, sockAddrSize);
        hostAddr.sin_family = PF_INET;
        hostAddr.sin_port = htons(SERVER_PORT_NUM);
        hostAddr.sin_addr.s_addr = htonl(INADDR_ANY);

        /* grab the internet domain socket */

        if ((fds = socket(PF_INET, SOCK_STREAM, 0)) == -1)
        {
                perror("socket");
                close(fds);
        }

        /* bind socket to hostAddr */

if (bind(fds, (struct sockaddr*)&hostAddr, sockAddrSize) == -1)
        {
                perror("bind");
                close(fds);
        }

/* put your ear on the binding and listen for a connection request */
```

```
if (listen(fds, MAX_QUEUED_CONNECTIONS) == -1)
{
        perror("listen");
        close(fds);
}

/* when the binding is ringing: answer and accept a connection */

if((fda = accept(fds, (struct sockaddr*)&targetAddr,
                &sockAddrSize)) == -1)
{
        perror("accept");
        close(fda);
}
else
        printf("Request for connection with target accepted\n");

if ((nRead = read(fda, (char *)&READY, sizeof(READY))) == -1)
{
        perror("read");
        close(fda);
        close(fds);
        exit(1);
}

printf("target said: %s\n",READY);

/*******************start transmitting commands**********************/

char command2[] = "wh\r";
write (fda, (char *)&command2, sizeof(command2));i = 0;
if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
while(character != '\r') {
if( read (fda, &character, REC_SIZE))
whatsinbuffer[i++]=character;
}
printf("Reading size of buffer: %d\n", i);
printf("Reading contents of buffer: %s\n", whatsinbuffer);

/*******************next command*********************************/
/*   gripper open */

char command1[] = "go\r";
write (fda, (char *)&command1, sizeof(command1));

/******************confirmation*******************************/

write (fda, (char *)&command2, sizeof(command2));i = 0;
if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
while(character != '\r') {
if( read (fda, &character, REC_SIZE))
whatsinbuffer[i++]=character;
}
printf("Reading size of buffer: %d\n", i);
printf("Reading contents of buffer: %s\n", whatsinbuffer);

/****************next command ********************************/
/* goto position #0 */

char command3[] = "mp 500,0,500,180,0,90\r";
write (fda, (char *)&command3, sizeof(command3));

/******************confirmation********************************/
```

```
write (fda, (char *)&command2, sizeof(command2)); i = 0;
if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
while(character != '\r') {
if( read (fda, &character, REC_SIZE))
whatsinbuffer[i++]=character;
}
printf("Reading size of buffer: %d\n", i);
printf("Reading contents of buffer: %s\n", whatsinbuffer);

/****************next command ***********************************/
/*  goto position #2 */

char command4[] = "mp 600,0,145,180,0,90\r";
write (fda, (char *)&command4, sizeof(command4));

/******************confirmation**********************************/

write (fda, (char *)&command2, sizeof(command2)); i = 0;
if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
while(character != '\r') {
if( read (fda, &character, REC_SIZE))
whatsinbuffer[i++]=character;
}
printf("Reading size of buffer: %d\n", i);
printf("Reading contents of buffer: %s\n", whatsinbuffer);

/****************next command ***********************************/
/*  gripper close -> grasp object */

char command0[] = "gc\r";
write (fda, (char *)&command0, sizeof(command0));

/******************confirmation**********************************/

write (fda, (char *)&command2, sizeof(command2));i = 0;
if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
while(character != '\r') {
if( read (fda, &character, REC_SIZE))
whatsinbuffer[i++]=character;
}
printf("Reading size of buffer: %d\n", i);
printf("Reading contents of buffer: %s\n", whatsinbuffer);

/****************next command ***********************************/
/* goto position #3  */

char command5[] = "mp 480,-250,300,180,0,0\r";
write (fda, (char *)&command5, sizeof(command5));

/******************confirmation**********************************/

write (fda, (char *)&command2, sizeof(command2)); i = 0;
if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
while(character != '\r') {
if( read (fda, &character, REC_SIZE))
whatsinbuffer[i++]=character;
}
printf("Reading size of buffer: %d\n", i);
printf("Reading contents of buffer: %s\n", whatsinbuffer);

/****************next command ***********************************/
/*  goto position #4  */

char command6[] = "mp 480,-250,220,180,0,0\r";
write (fda, (char *)&command6, sizeof(command6));
```

```
        /*******************confirmation*********************************/

        write (fda, (char *)&command2, sizeof(command2)); i = 0;
        if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
        while(character != '\r') {
        if( read (fda, &character, REC_SIZE))
        whatsinbuffer[i++]=character;
        }
        printf("Reading size of buffer: %d\n", i);
        printf("Reading contents of buffer: %s\n", whatsinbuffer);

        /****************next command*********************************/
        /* gripper open -> place object */

        write (fda, (char *)&command1, sizeof(command1));

        /*******************confirmation*********************************/

        write (fda, (char *)&command2, sizeof(command2));i = 0;
        if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
        while(character != '\r') {
        if( read (fda, &character, REC_SIZE))
        whatsinbuffer[i++]=character;
        }
        printf("Reading size of buffer: %d\n", i);
        printf("Reading contents of buffer: %s\n", whatsinbuffer);

        /****************next command **********************************/
        /* goto position #0 */

        char command7[] = "mp 500,0,500,180,0,90\r";
        write (fda, (char *)&command7, sizeof(command7));

        /*******************confirmation*********************************/

        write (fda, (char *)&command2, sizeof(command2)); i = 0;
        if(read (fda, &character, REC_SIZE)) whatsinbuffer[i++]=character;
        while(character != '\r') {
        if( read (fda, &character, REC_SIZE))
        whatsinbuffer[i++]=character;
        }
        printf("Reading size of buffer: %d\n", i);
        printf("Reading contents of buffer: %s\n", whatsinbuffer);

        /*******************end of task**********************************/

        char commande[] = "ed\r";
        write (fda, (char *)&commande, sizeof(commande));
        printf("send end-of-task command\n");

        close(fds);
        close(fda);
}


==========================================================================
/* task1_target.c: pick and place task for Move Master robot. This program runs
on VxWorks and communicates with its UNIX counterpart: task1_host.c
*/

#include "vxWorks.h"
#include "fcntl.h"
#include "tyLib.h"
#include "ioLib.h"
#include "netinet/tcp.h"
#include "sys/socket.h"
#include "in.h"
#include "stdio.h"

#define SERVER_PORT_NUM (5001)
#define MSG_SIZE (64)
#define MAX_QUEUED_CONNECTIONS (4)
#define SERVER_INET_ADDR "150.29.102.37" #define MAX_DATA 2
#define SERIAL "/tyCo/0"

STATUS TASK1(void)
{
        int     fds, i;
        struct  sockaddr_in  hostSockAddr;
        char    command[64], where[64], character;
        char    ready[17] = "ready to receive";
        int     SOCK_ADDR_SIZE = sizeof(struct sockaddr_in);
        int     sFd;
        int     optval;
        int     nRecv, data;

        /* create socket */

        if ((sFd = socket (PF_INET, SOCK_STREAM, 0)) == -1)
        {
                perror("socket");
                close(sFd);
        }

        /* set socket options */

        optval = 1;

        setsockopt(sFd, IPPROTO_TCP, TCP_NODELAY, &optval, sizeof(optval));

        /* build server socket address */

        memset (&hostSockAddr, 0, SOCK_ADDR_SIZE);
        hostSockAddr.sin_family = PF_INET;
        hostSockAddr.sin_port = htons (SERVER_PORT_NUM);
        hostSockAddr.sin_addr.s_addr = inet_addr (SERVER_INET_ADDR);

        /* connect to server */

        if (connect (sFd, (struct sockaddr*)&hostSockAddr,
        SOCK_ADDR_SIZE) == -1)
        {
                perror("connect");
                close(sFd);
        }

        printf("connected to server\n");

        /* write a message to the server that we want to receive a command */

        if (write (sFd, (char *)&ready, sizeof(ready)) == -1)
```

```
{
        perror("write to host");
        close(sFd);
}

printf("Request for receiving commands from the host\n");
fds = open(SERIAL, O_RDWR, 0);

while (command[0] != 'e' && command[1] != 'd') {
        i = 0;
        if (read(sFd, &character, 1))
        {
                command[i] = character;
                if (write (fds, (char *)&character, 1) == -1)
                {
                        perror("write to serial line");
                        close(fds);close(sFd);
                        exit(1);
                }
        }
        while(1) {
                i++;
                if (read(sFd, &character, 1) == -1)
                {
                        perror("read from host");
                        close(fds);close(sFd);
                        exit(1);
                }
                if (write (fds, (char *)&character, 1) == -1)
                {
                        perror("write to serial line");
                        close(fds);close(sFd);
                        exit(1);
                }
                command[i] = character;
                if (character == '\r') break;
        }

        printf("COMMAND writing to %s: %s\n",SERIAL,command);

        if (command[0] == 'w' && command[1] == 'h' )
        {
                i = 0;
                if (read(fds, &character, 1))
                {
                        where[i] = character;
                        if (write (sFd, (char *)&character, 1) == -1)
                        {
                                perror("write to host");
                                close(fds);close(sFd);
                                exit(1);
                        }
                }
                while(1) {
                        i++;
                        if (read(fds, &character, 1) == -1)
                        {
                                perror("read from host");
                                close(fds);close(sFd);
                                exit(1);
                        }
                        if (write (sFd, (char *)&character, 1) == -1)
                        {
                                perror("write to serial line");
                                close(fds);close(sFd);
```

```
                                        exit(1);
                                }
                                where[i] = character;
                                if (character == '\r') break;
                        }
                }
        }
        close(sFd);
        close(fds);
        exit(1);
}
```