PARALLEL PROCESSING AND THE DYNAMICS AND KINEMATICS OF A THREE DEGREE OF FREEDOM PLANAR MANIPULATOR.

Ahmed Helmy

B.Sc. Alexandria University 1980

Department of Mechanical Engineering

McGill University, Montréal

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Masters in Engineering

January 1994

PARALLEL PROCESSING AND THE DYNAMICS OF A PLANAR MANIPULATOR.

Ahmed Helmy

B.Sc. Alexandria University 1980

Department of Mechanical Engineering
McGill University, Montréal

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Masters in Engineering

March 1994

Abstract

The application of a parallel processing system (hardware and software) in the solution of the kinematics and dynamics to control a three degree of freedom parallel manipulator is the subject of this thesis.

Parallel processing systems are introduced and analysed for this application. A parallel microcomputer system is used and a new method of direct kinematics for displacement analysis is implemented. The selected microcomputer system is integrated in an IBM 286 compatible personal computer.

Finally, a parallel software program is implemented, which allows for efficient control of three actuators driving the three degree of freedom planar manipulator. Hence, introducing a new processing method for the control of this manipulator.

Résumé

L'application d'un système à processus parallel (materiel et logiciel) dans la dynamique el la cynématique pour le contrôl d'un manipulateur a trois degré de liberté représente l'objet de cette thèse.

Les systèmes à processus parallel sont introduit et analysé pour cette application. Un système parallel de microordinateur est utilisé et une nouvelle méthode de dynamique direct pour l'analyse des déplacements est implementé. Le système de microordinateur est integré dans un IBM 286 compatible.

Un logiciel parallel qui permet un contrôl éfficace des méchanismes qui communiquent le mouvement au manipulateur à trois degré de liberté est implementé. Il represente une nouvelle méthode de contrôl du manipulateur.

Acknowledgements

I would like to express my deepest gratitude and appreciation to Professor P. Z. Murray and Professor L. J. Vroomen, for their excellent guidance and their help in developing my abilities of clear, rational thought and disciplined research.

I would like to express my thanks to all my fellow students in the robotics group at MCRCIM with whom I spent a very good time. Also I would like to express my warm thanks to Sophia Kounias, MCRCIM staff and computer systems administration for numerous discussions on various topics related to the subject of this research work.

I am also grateful to my wife, Lila Madour for her understanding, patience, encouragement and moral support throughout the course of my graduate study and for translating the abstract of this thesis into French.

Finally, I acknowledge the partial financial support from the Department of Mechanical Engineering through teaching assistantships. Furthermore, this research was made possible by NSRC (Natural Science and Engineering Research Council) Grant No. A4219 and partial funding from IRIS (Institute for Robotics and Intelligent Systems).

List of symbols

r: number of movable rigid bodies.

m: number of one-degree of-freedom joints.

n: number of degrees of freedom of the system.

$\mathbf{a}_i$: The vector connecting the ith joint O_i to one of its neighboring joints.

C_i: the center of mass of the ith link.

$\mathbf{o}_i$: The position vector of O_i, in the X-Y frame.

$\mathbf{c}_i$: The position vector of C_i, in the X-Y frame.

ω_i: Scalar denotes the angular velocity of the ith link.

$\dot{\omega}_i$: Scalar denotes the angular acceleration of the ith link.

$\dot{\mathbf{c}}_i$: 3-D velocity vector of the center of mass of the ith body.

$\ddot{\mathbf{c}}_i$: 3-D acceleration vector of the center of mass of the ith body.

m_i: mass of the ith body.

$\mathbf{I}_i$: 3×3 inertia tensor of the ith body about its center of mass.

$\mathbf{f}_i^*$: 3-D vector of the inertia force of the ith body at its mass center.

$\mathbf{n}_i^*$: 3-D vector of the inertia moment of the ith body about its center of mass.

$\mathbf{w}_i^*$: 6-D vector of the inertia wrench of the ith body.

$\boldsymbol{\tau}_i^f$: a 3-D vector friction torque acting on the ith joint.

$\mathbf{w}_i^f$: 6-D vector of the friction wrench of the ith body.

$\mathbf{w}_i^g$: 6-D vector of the gravity wrench of the ith body.

$\boldsymbol{\tau}_i$: a 3-D vector generalized deriving torque of the ith actuated joint. $\boldsymbol{\tau}_i$ is a torque if the ith joint is revolute and a force if the ith joint is prismatic.

τ^a : a 3-D vector generalized driving torque of the whole manipulator.

Ω : is the *twist-constraint matrix*, which is of dimension 18×21 and configuration-dependent.

O : is a 3×3 zero matrix,

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Parallel processing

There appears to be an unwritten rule which states that whenever a new program is debugged and running, someone always asks how it can be made to run faster or process more data. This demand for more processing power stems from systems designers having to meet ever higher performance requirements. Advances in technology enables sensors to acquire more data and algorithms get more sophisticated which demand higher performance computers.

The conventional approach in the quest for higher performance has been to push the speed of sequential, or Von Neumann, architectures by using more advanced technology.

An alternative to sequential processors is to develop parallel architectures with data and algorithms being distributed over many processors. This approach offers orders of magnitude improvement in usable processor complexity by multiple replication of simply connected processors and memory elements and utilizing existing Very Large Scale Integration (VLSI) technology.

The transputer (chapter 2) and its programming language, **occam** (chapter 3), have been designed specifically to support multi-processor systems. Any number of transputers can form a network to build multi-processor systems. The performance is scalable giving a

substantial increase in performance as more processors are added to the network.

Assuming that biological systems got it right a few million years ago by evolving highly parallel nervous systems then highly parallel Multiple Instruction Multiple Data (MIMD) architectures will have significant advantages over other architectures. MIMD machines, such as transputer systems, have the potential of high performance with the flexibility to suit a wide range of applications. Compared to single processor machines, both programming and control is more complicated in MIMD architectures. In the following chapters, we describe the transputer and its programming environment and show how it can be applied to the computation of the dynamics and kinematics necessary to control a Three Degree of Freedom Parallel Manipulator.

1.1.1 Parallel processing architectures

Processors can be connected together in many ways to produce a wide variety of parallel processing architectures. Similarly, a variety of methods, or paradigms, have evolved to program these machines. Transputers can be used in many different topologies and it is helpful to consider the different classes of architectures. Parallel architectures are commonly grouped according to Flynn's taxonomy [16], shown in figure 1.1, as:

1) Multiple Instruction-stream Single Data-stream (MISD): several processors simultaneously execute different instruction streams on a single data stream. A pipeline system is a typical example of a MISD architecture.
2) Single Instruction-stream Multiple Data-stream (SIMD): several processors simultaneously execute the same instruction on multiple data streams, such as Array Processors.
3) Multiple Instruction-stream Multiple Data-stream (MIMD): in this case, each processor may be simultaneously performing different instructions on different data.

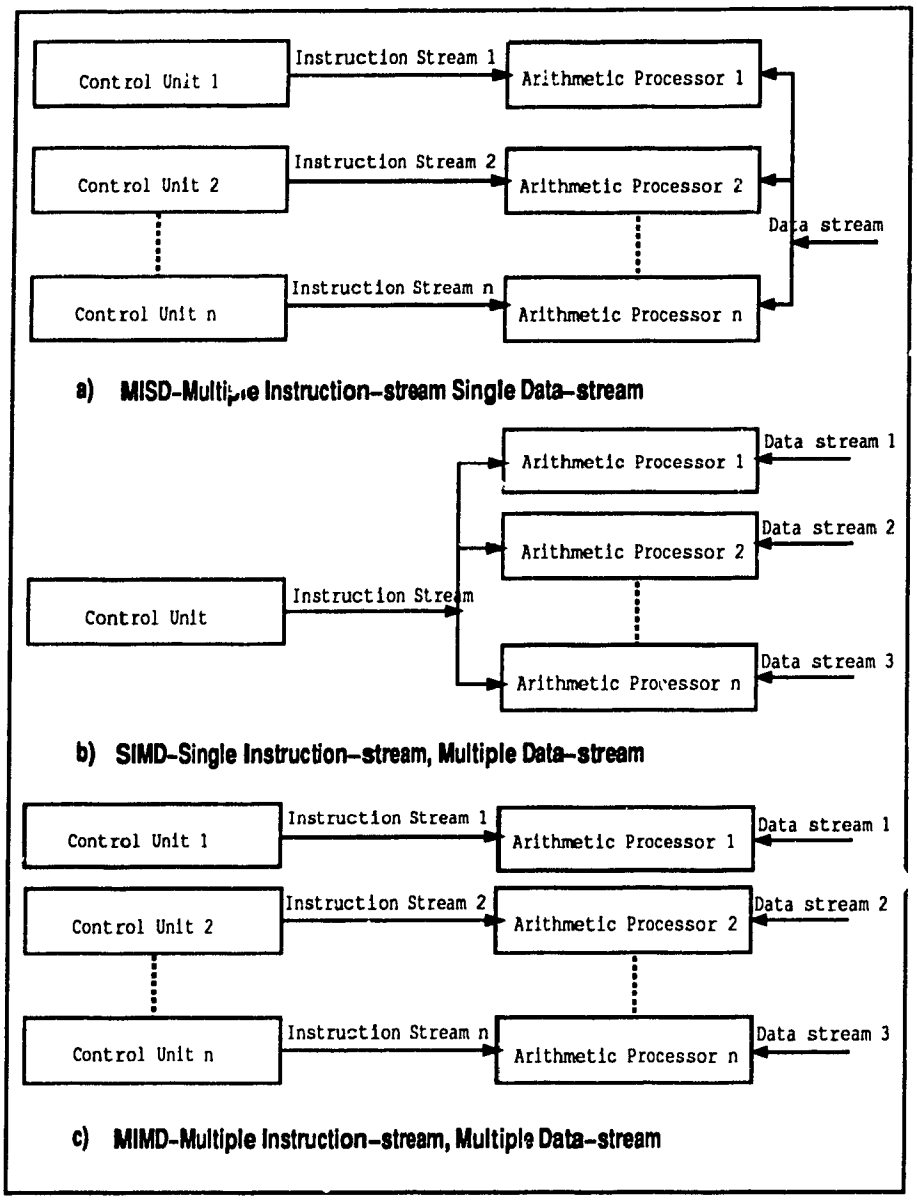


Figure 1.1: Flynn's classification

MISD machines:

MISD or pipeline machines are based on the partitioning of a complete algorithm into tasks that each can be executed sequentially. Such a machine consists of elementary processors each assigned to a task that is executed in parallel with other tasks in the other processors. Data is passed into a pipeline of processing stages. Within each stage the same operation is

performed on every element of the data with different stages operating on the data sequentially. Once the pipeline has been filled, each cycle produces a new result irrespective of the number of stages in the chain. The machine operates at the speed of the slowest processor and therefore effective partitioning of the algorithm is essential, as the slowest processor forms a bottleneck. MISD machines have the advantage of having regular interconnects between stages and are relatively easy to program.

SIMD machines:

The SIMD architecture consists of an array of identical processing elements (PEs), all operating under the control of a single control unit. All PEs execute the same program in synchronism (lock-step) but on different data. The PEs are usually connected in a two dimensional array, each with its own local memory and with a nearest neighbor connection network. In the majority of SIMD machines the PEs are 1-bit arithmetic units (ALUs) with data paths being 1-bit wide. This feature makes them eminently suitable for implementation where a simple cell can be replicated many times.

An array of transputers can be connected in a manner similar to SIMD. In other words transputers can be connected to form a two-dimensional network where each transputer executes a program block rather than a single instruction, with synchronization occurring at the end of each block. The term Single Program Multiple Data (SPrMD) has been coined for this mode of operation.

MIMD machines:

A MIMD machine is characterized by a set of independent processors, executing different programs and communicating with each other. The processors invariably have some local memory, also have either access to common shared memory or access to each other's memory, as shown in figure 1.2. The interconnection network may be fixed or non-fixed (switched).

MIMD processors offer the promise of highly flexible architectures. Until recently, they have been difficult to program efficiently and are often used in a pseudo SIMD mode

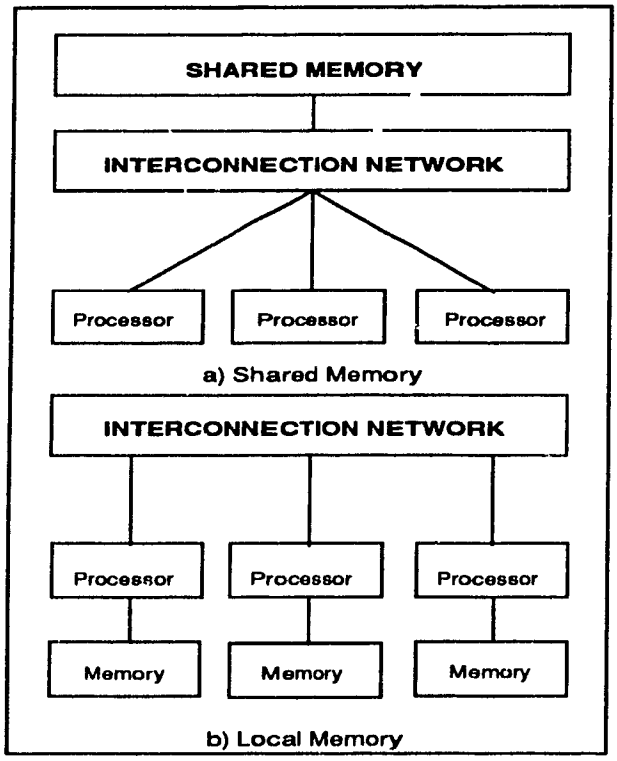


Figure 1.2: Memory topologies

(SPrMD) where all processors execute the same program. Transputers can be used in local memory MIMD architecture machines as shown in figure 1.2(b).

More on architectures:

Although Flynn's classification is useful, it is not exhaustive. Systolic architectures are special-purpose architectures consisting of simple processors or cells locally and regularly interconnected. Data streams flow through these cells in such a way that they interact at each connection. They can use both pipeline and parallel concepts. Again, an analogy with biology, where the word systol describes the heart contraction rhythm to pump blood, systolic arrays pump data to give a regular data flow in the network. Systolic arrays have the same disadvantage as pipeline processors in that they have to be clocked at the speed of the slowest processor.

A closely related structure, the wavefront processor, is more relevant to transputer

applications [7]. The wavefront processor has the same connectivity as a systolic array but is a data-flow-driven machine, therefore does not require a synchronous clock.

Important characteristics of a parallel machine are the size and number of processors and the communication bandwidth between them. Machines with tens or hundreds of relatively large processors are classified as coarse grained, whereas machines with a thousand or more small and simpler processors are classified as fine grained.

Fine grained processors are potentially faster because the larger number of processors allows a higher degree of parallelism. The degree of parallelism that can be achieved is very much dependent on the problem and the algorithms for solving that problem. For example, an image of 1000 by 1000 pixels would map easily onto a million-processor machine if one were available. This is obviously an extreme example but there are very few problems which cannot exploit a high degree of parallelism.

Low level operations conveniently map onto fixed networks of processors connected in regular square or rectangular arrays but there may be problems in transferring data to and from processors in the centre of the array. Higher-level operations may be more efficiently implemented on tree or pyramid structures. Reconfigurable networks appear to have advantages for inputting and outputting data and are capable of efficient execution of a wider range of algorithms. MIMD processors with flexible interconnections appear to have advantages over the fixed topology of SIMD array processors.

The main issue in choosing between SIMD or MIMD architectures is programmability. The programming language **occam** appears to solve the difficulties in programming and controlling MIMD machines and, given that a MIMD machine can execute in a SIMD mode but not vice versa, then MIMD machines are worthy of further consideration. It is interesting to note that more advanced biological nervous systems have evolved into richly connected MIMD processors with a degree of reconfigurability.

1.1.2 Transputer-based architectures

As outlined previously, MIMD machines have a potentially wider range of applications than SIMD machines as SIMD operation is a subset of the MIMD mode. Until the advent of the transputer, MIMD machines were limited to a relatively small number of processors by the difficulties in programming and synchronizing such machines. The transputer and **occam**, which explicitly controls concurrency, was designed to overcome these limitations.

There are three major classes of applications for transputers: accelerator boards for PCs and workstations, embedded systems, and general purpose computers. The first transputer products were mostly accelerator boards to boost the performance of existing computers, as add-in boards [38]. The second class is the use of transputers in embedded systems such as laser printer controllers. Image processing [21] and space-borne applications [13] are typical examples of embedded systems.

1.1.3 Transputer networks

Transputers have four links (links are described in chapter 2) allowing several interconnection topologies to be implemented (see figure 1.3). Many existing algorithms can be mapped into a pipeline or series of pipelines. An n-stage transputer constructed pipeline has 2n free (or unused) links and therefore does not make optimum use of the transputer communication facilities and the disadvantages of MISD machines apply. Trees are suitable for hierarchical processes such as data reduction with data being passed up, or sorting with data being passed down.

Two-dimensional arrays are suitable for array structured data. The 2-D array topology is the same as the SIMD array processors, fully utilizing the four transputer communication links. Mapping techniques developed for SIMD arrays apply. Pipelines and tree architectures can be mapped onto such an array. There are two potential problems associated with large 2D arrays: first, in transferring data in and out of the array, and second in irregular communications.

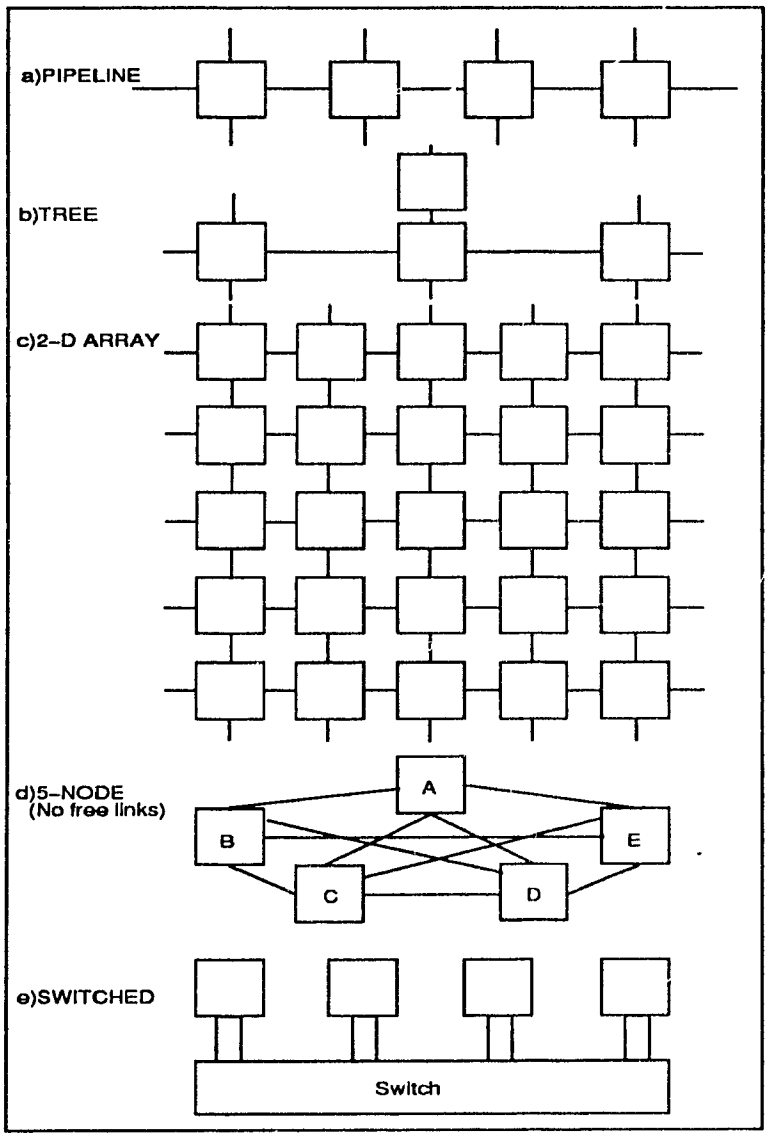


Figure 1.3: Transputer networks

I.e., if the processor in the top left corner of the 2-D array shown in figure 1.3.c, is required to communicate with the processor at the bottom right at the same time as the top right processor is communicating with the bottom left processor, then all data or messages get routed through the centre processors causing a bottleneck. Communication kernels based on message routing or packet switching techniques have been developed for more general communication, at the expense of bandwidth reduction and increased latency. A node of totally connected transputers can overcome this problem but, with only four links per transputer,

the maximum node-size is five transputers (e.g., to make all node transputers connected together, in figure 1.3.d, if another transputer is added to the node, transputer A cannot be linked to that extra transputer as all nodes are totally connected). Data input and output is limited by the ratio of links connected to the external world to the number of internal connections. For a square array of size $\mathbf{n}$, the number of processors is $\mathbf{n^2}$ with $\mathbf{4n}$ links connected externally and $\mathbf{2n(n-1)}$ links connected internally. As the size of the array increases, the overhead in transferring data to the centre increases by a ratio of $\mathbf{n : (n^2 - n)/2}$. To overcome problems inherent in fixed networks and to provide maximum flexibility, a reconfigurable topology can be used.

1.1.4 Reconfigurable networks

The granularity, and hence, the ratio of communication time to compute time, is an important issue in concurrent processors (concurrency is described in chapter 3). The use of reconfigurable networks allows a reduction in communication time by reducing the number of intermediate processors that messages or data have to travel through. A reconfigurable network can be used for many applications and is particularly useful for large multi-processor development systems. Reconfiguration also provides some degree of fault tolerance as faulty processors can be bypassed or switched out of the network.

In case the configuration is implemented by a software controlled switch, the topology can be changed statically, quasi-statically or dynamically. In the case of static switching, the topology is fixed before programs are loaded and the application is run with that switch setting. The network configuration may be extracted by the Transputer Development Set (TDS) configuration utility (described in chapter 3). Quasi-static switching is used when all processors can be synchronized at predetermined points in the program. At the synchronization point, all communication ceases and all processors wait for the link connections to be changed. Once the connections have been switched computation resumes. This is particularly applicable to image processing where the network may be configured to allow inputing images at the maximum data rates, for example, configured into multiple parallel

pipelines with the maximum connectivity (since the two-dimensional array middle processors have no free links to the outside world). Once the data is completely loaded into the system, it can be reconfigured into a two-dimensional array for low-level operations which can be executed in a SPrMD mode. After all low-level operations have been computed by all processors, the array can be configured into e.g., a tree, for high-level MIMD processing. In a dynamic switching mode, any connection between any two processors can be changed at any time, provided that no communication is taking place on that connection. Dynamic switching allows new programming methods to be used, such as, dynamic load balancing and transputer networks which are control machines, can be used as data-driven, data flow or as reduction machines.

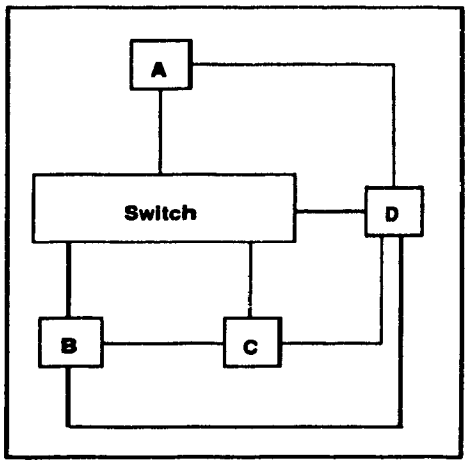


Figure 1.4: Dynamic switching

There will be some overhead and inefficiencies to be overcome with dynamic switching. For example, consider the network shown in figure 1.4 where processor A is sending data to two processors, B and C, through one link via a switch, with the switch being controlled by processor D. At some instant, A will be sending data to B. If C becomes idle and requires more data then the link needs to switched from A-B to A-C. Processor C has to signal D that it requires data. D has to signal A and B that the link is going to be changed and then wait until A and B stop communicating. D then changes the link and signals A and C that the new connection is made and that both processors can commence communication. If switching is frequent then D will have a heavy work load and may become a bottleneck in

the system.

If the ratio of computing to communication time is large, then the communication overhead is negligible and the configuration is irrelevant, regardless of which topology is used and reconfiguration is not necessary. Nevertheless, a switchable network can aid program development and debugging and may give some degree of fault tolerance [20].

1.2 Kinematics and dynamics of a planar 3-DOF Parallel Manipulator

Based on their kinematic architecture, industrial manipulators fall into two categories: serial and parallel. Serial manipulators have an open-chain kinematic architecture, all of whose joints are actuated. On the contrary, parallel manipulators have a closed-chain kinematic architecture, some of their joints being unactuated. Serial manipulators, in general, have the advantages of simpler kinematic and dynamic models, larger work-spaces, higher dexterities, etc $\cdots$, and have been studied extensively [42], [6], [10]. However, because of their cantilever-beam-like architecture, serial manipulators inherently suffer from some drawbacks, such as low mechanical stiffness which leads to lower operation accuracy, or dynamic characteristics, and lower loading capacity. These disadvantages can be overcome by designing manipulators with closed kinematic loops, namely, parallel manipulators.

The two major problems in kinematics and dynamics of serial manipulators, namely, the *forward* and *inverse* analysis, appear in parallel manipulators as well. In case of serial manipulators, forward kinematics is a straight forward problem, which can be solved recursively and on line because the relative motions of all joints are independent and available. However, the inverse kinematics of serial manipulators although well-developed now [1], [39], remains a rather challenging research topic. The inverse kinematics of parallel manipulators is straight forward, similar to serial manipulators, because the problem can be solved independently with each individual kinematic loop and hence, the methodology of inverse kinematics of serial manipulators can be applied directly. However, the forward kinematics

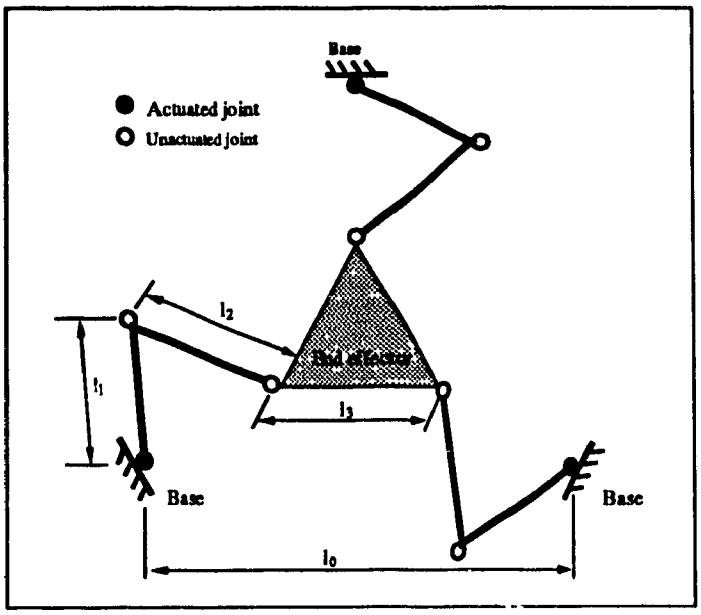


Figure 1.5: A planar parallel manipulator

of parallel manipulators is much more complex than that of serial manipulators because of the presence of unactuated joints whose relative motions are dependent and not available. The motions of unactuated joints cannot be found from individual loops of the manipulator but from a set of simultaneous nonlinear equations involving all the loops of the manipulator. The problem of direct kinematics is an active topic of current research. Moreover, for both manipulator control and simulation, as far as dynamics is concerned, on-line forward kinematics is inevitable (as the dynamics solution depends on forward kinematics data). The importance of developing efficient methods for analyzing dynamics and kinematics for parallel manipulators thus becomes apparent.

1.3 Summary

Based on the foregoing two sections, we present in this thesis a method for solving the dynamics and kinematics as applied to the parallel manipulator shown in figure 1.5. This manipulator was introduced by Hunt [24], and could be considered as a planar example of the well known Stewart platform [49]. In recent years, because of their typical kinematic

architecture related to parallel manipulators, Stewart platforms have attracted the attention of many researchers. Yang and Lee [52] investigated the kinematic feasibility of this type of manipulators. Fichter [15] built several Stewart platforms and proposed their inverse kinematics and inverse dynamics models as well as a discussion on singularities. Later, intensive studies on inverse kinematics and inverse dynamics were presented by Do and Yang [12], Lee and Chao [33]. Merelet [40] and Behi [5] briefly discussed in their papers the nonlinear displacement problem of Stewart platforms. Their approaches are rather straightforward, i.e., solving all simultaneous constraint equations numerically. This type of approach is believed to be inefficient and does not provide a way of controlling the solution branches. In this thesis we apply the method proposed by Ma and Angeles [37] in which they remove one of the manipulator links, so that only one of the three kinematic loops remains. Consequently, the nonlinear constraint equations which must be solved for the determination of the unactuated joint displacements reduce to one. The remaining constraint equations are solved in closed form. Based on the rigidity condition of the removed link, the resulting distance equation contains only one variable and hence, can be solved efficiently (chapter 4).

1.4 Organization of the Thesis

The thesis is organized as follows:

Chapter 1 introduces parallel processing concept, highlighting parallel processing architectures, the transputer as a parallel building block, and the previous research (theoretical) presented by the researchers in the solution of the 3-DOF planar parallel manipulator kinematics and dynamics.

Chapter 2 describes briefly the internal architecture of the hardware selected and implemented in the current work, to present the motivation for its selection and to show its adequacy for this work. Moreover, this chapter presents the reader the motivation for many of the desgin decisions taken in the course of the implementation discussed later. This chapter concludes by explaining the main features of the processor selected by the author for the

physical application part of this thesis.

Chapter 3 starts with a historic background on the development of the transputer special programming language (**occam**), then proceeds with an overview of the important aspects of the transputer program development environment used and **occam** programming language. Moreover, this chapter explains the reasons for many of the program design decisions taken during the software implementation phase of this study and shows towards the end the physical configuration at the software implemented.

Chapter 4 describes the historic development of the mathematical algorithms in the solution of the kinematics and dynamics of the 3-DOF planar manipulator. Then, the solution developed by Ma is presented [36]. Ma's solution is adopted by the author in the implementation of the manipulator's prototype controller phase of this thesis work.

Chapter 5 deals with the control system integration. It explains in detail the control system that is implemented by the author from inception in both hardware and software. Furthermore, this chapter gives explanations of the integration decisions.

In the hardware section of this chapter the hardware parts selection, the relevancy of their selection, how they were integrated and the specifications of the whole system after integration are thoroughly discussed.

In the software section, a thorough description of the control scheme design is given. Furthermore, a thorough description is given of the software that is developed to provide real time interactive control of the manipulator utilizing the capabilities of the hardware. Comparative examples of actual parallel **occam** code used and FORTRAN code are given to demonstrate the parallel concept implementation in the program. Moreover, some examples of actual sequential **occam** code used are discussed to show the explicit nature of sequential processing in the language used. The examples also provide the motivation for the selection of the parallel and sequential parts in the program.

Finally, an overview of the control program's technical considerations is provided. These include:

- The timing constant (the cycle time) which control the control program's performance.
- The specifications to provide the program its required run time inputs (the manipulator's three actuator angles).
- The specifications of the program's outputs (the manipulator's three actuator torque values).

Chapter 6, is concerned with the performance aspect of the controller. Comparative benchmark test results are presented to show the hardware capabilities and potentials. Furthermore, a simulation of the control algorithm was developed on the basis of the main program to prove that it operates and the results of that simulation are presented and discussed.

At this point the objectives set for this study are completed.

Chapter 7 concludes the thesis by highlighting the significant contributions of the current work, discussing the limitations of the present approach, and pointing out the directions for future research.

Chapter 2

The Transputer Architecture

The introduction of the transputer has been described by Barton [4] as *the most significant event in concurrent computing.*

The list of current transputer based applications is growing almost daily and includes:

- Industrial controllers - robotics, manufacturing processes automation.
- Telecommunications - switching and node controllers.
- Image processing, pattern recognition and artificial intelligence.
- Biomedical applications - body scanners.
- Computer vision which includes computer graphic simulation.

In this chapter, a brief description of the internal transputer architecture is presented.

2.1 General description

The transputer is a single chip microcomputer with its own 32 bit processor, local memory and links for connecting one transputer to another and communicating with the outside world. Furthermore, each transputer contains special interface circuitry which can be used to adapt it to a particular application. For example, to use the transputer as a disk controller, see figure. 2 1.

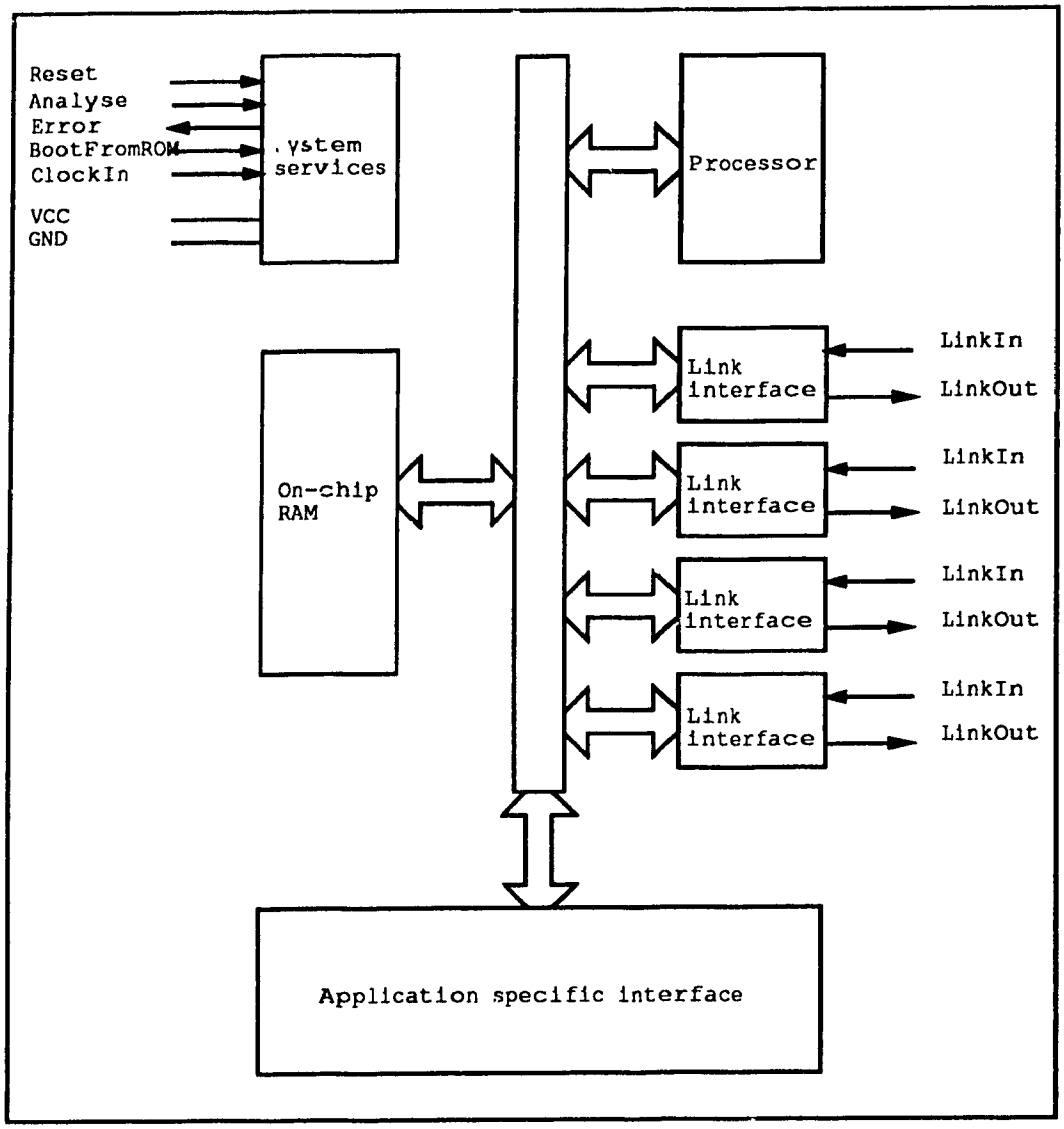


Figure 2.1: Transputer architecture

The 32 bit CPU transputer can have up to 4 Kbytes of on-chip memory for local fast access [9] [47]. The most powerful transputer available to date (T9000) is able to deliver a sustained performance of more than 70 Million program Instructions Per Second (MIPS) and more than 15 Million FLoating point Operations Per Second (MegaFLOPS) [46]. Moreover, the transputer architecture has the advantage of allowing one to increase the total system performance with the addition of more transputers to an existing transputer network.

2.1.1 Transputer support for concurrency

The processor provides efficient support for the **occam** model of concurrency and communication. It has a microcoded scheduler, which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel. The processor does not need to support the dynamic allocation of storage since the **occam** compiler is able to perform the allocation of memory to concurrent processes.

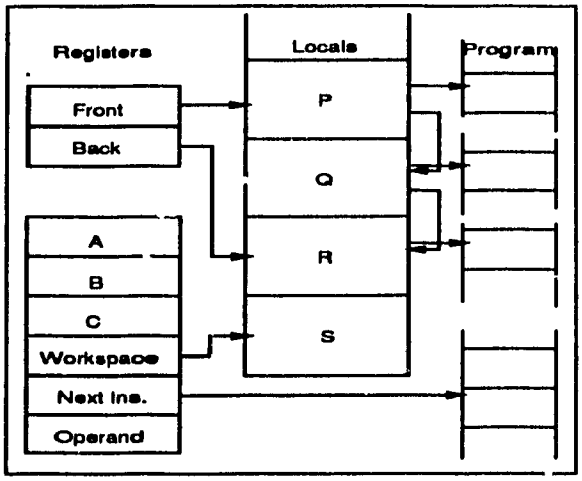


Figure 2.2: Linked process list

At any time, a concurrent process can be:

- **Active**: being executed or on a list waiting to execute.
- **Inactive**: ready for input, ready to output, or waiting until a specified time.

The scheduler operates in such a way that inactive processes do not use any processor time. The active processes waiting to be executed are held in a list. This is a linked list of process workspaces implemented by using two registers one of which points to the first process in the list, the other to the last. In figure 2.2, while process S is executing, P, Q and R are active awaiting execution.

A process is executed until it is unable to proceed and waits for an input from another process, to output, or for a timer signal. Whenever a process is unable to proceed, its

instruction pointer is saved in its workspace and the next process is taken from the list. Since it is not necessary to save the evaluation stack on a rescheduling operation, the actual process switch time is very small.

The processor provides a number of special operations to support the process model. These include *start process* and *end process.*

During the execution of a parallel construct, *start process* instructions are used to create the necessary concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed.

The concurrent termination of a parallel construct is assured by the use of the *end process* instruction. This uses a workspace location as a counter of the components of the parallel construct which still have to terminate. Initially, the counter is initialized to the number of components before the processes are started. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is set to zero and the component continues.

2.1.2 Transputer communication

Communication between processes is achieved by means of channels. **Occam** communication procedures are point-to-point, synchronized and unbuffered. As a result, a channel does not need a process queue, a message queue, or a message buffer.

A channel between two processes being executed on the same transputer is implemented by a single word in memory and this channel is called "internal channel". However, a channel between processes executing on different transputers is implemented by point-to-point links and this type of channel is called "external channel". The processor provides a number of operations to support message passing, the most important being *input message* and *output message* [47].

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. This means that the same instruction sequence can be used for both soft and hard channels, allowing a process to be written and compiled without knowledge of where its channels are connected.

As it will be shown later in chapter 3, the communication procedure takes place when both the inputting and the outputting processes are ready. Consequently, the process which first becomes ready must wait until the second one is also ready.

A process performs an input or an output by loading the evaluation stack with a pointer to a message, the address of a channel and a count of the number of bytes to be transferred, then executes an *input message* or an *output message* instruction.

Transputer communication links

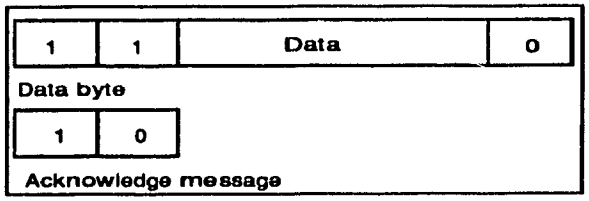


Figure 2.3: Link data and acknowledge formats

Communication between two transputers is established by connecting (hardwire) a "link" interface connection on one transputer using two uni-directional signal wires along which data is transmitted serially to the corresponding link interface on the other transputer. The two wires provide two **occam** channels, one in each direction. The presence of those two channels require a simple protocol to multiplex data and control its flow. Messages are transmitted as a sequence of bytes, each message is acknowledged before the next one is transmitted. A byte of data is transmitted in the following sequence; a one (high) start bit, one (high) bit, eight bits of data and a zero (low) stop bit. The receiving transputer sends an acknowledgement indicating both that a process has received the data byte and it is able to receive another byte, see figure 2.3.

The protocol permits an acknowledgement to be generated by the receiver transputer

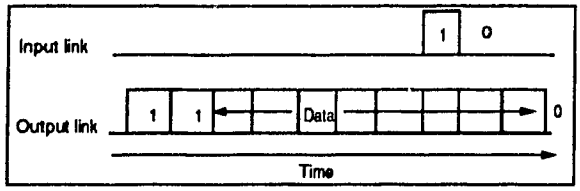


Figure 2.4: Overlapped link acknowledge

as soon as it identifies a data packet. In this way, the acknowledgement can be received by the transmitting transputer before the whole data packet has been transmitted. Consequently the transmitter can transmit the next data packet immediately. Some transputers do not implement this overlapping and achieve a data rate of 0.8 Mbytes/sec, using the links to transfer data in one direction. However, by the overlapping method along with the availability of buffering in the link hardware, the rate can be increased to 1.8 Mbytes/sec in one direction, and 2.4 Mbytes/sec when the link carries data in both directions. Figure 2.4 shows the signals that would be carried on the two wires when a data packet signal is in the same time frame with an acknowledgement.

2.1.3 Transputer timer

The transputer has an internal clock which generates a pulse every microsecond. The current value of the processor clock can be read by executing a *read timer* instruction by any process.

A process can perform a *timer input* instruction, in which case it will become ready to execute after a specified time (indicated by the *timer input* instruction) has elapsed since the *timer input* instruction requires a time to be specified. If this time is in the 'past' (i.e. *ClockReg* **AFTER** *Specified Time*) then the instruction has no effect. On the other hand if the time is in the 'future' (i.e. *SpecifiedTime* **AFTER** *ClockReg* or *SpecifiedTime* = *ClockReg*) then the process is de-scheduled. As soon as the specified time is reached, the process is scheduled again.

2.1.4 The IMS T425 features and overview

For the application described in this thesis, the author of this thesis used the IMS T425 microcomputer. The choice of this type of transputer was due to the fact that it was, at the time of the implementation, the fastest available transputer.

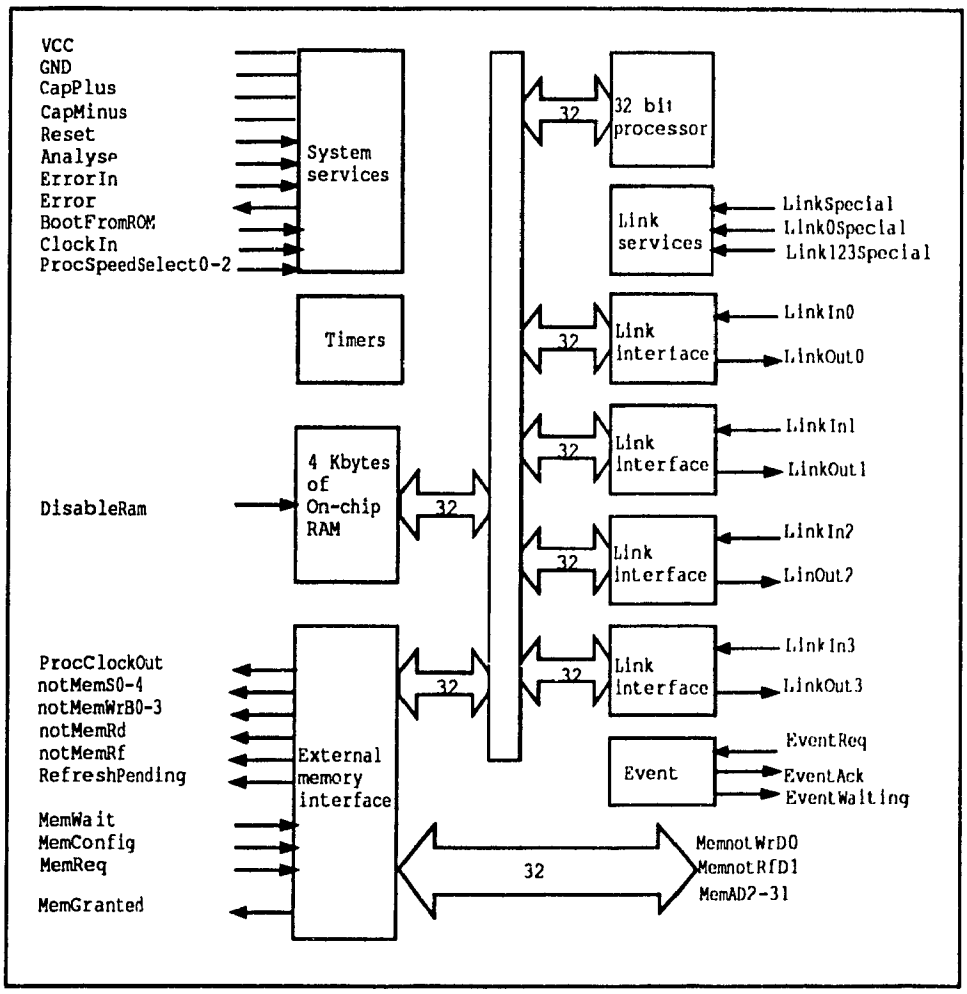


Figure 2.5: IMS T425 block diagram

Features

- 32 bit bus.
- 33 ns internal cycle time.
- 30 MIPS (peak) instruction rate.

- 4 Kbytes on-chip static RAM.
- 120 Mbytes/sec sustained data rate to internal memory.
- 4 Gbytes directly addressable external memory.
- 40 Mbytes/sec sustained data rate to external memory.
- 630 ns response to interrupts.
- Four INMOS serial links that can run on 5, 10 or 20 Mbits/sec operation speed.
- High performance graphics support with block move instructions.
- Boot from ROM or communication links.

The IMS T425 transputer (figure 2.5) has a configurable 32 bit wide memory interface. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems. The instruction set achieves efficient implementation of high level languages and provides direct support for the **occam** model of concurrency when using either a single transputer or a network. Procedure calls, process switching, and typical interrupt latency are in the sub-microsecond range.

The standard INMOS communication links allow networks of transputers to be constructed by direct point to point connections with no external logic. Each link can transfer data in two directions (through each of its two wires) at up to 2.4 Mbytes/sec. However, only two links are available for this implementation. The T425 two remaining links are used by the transputer motherboard to communicate with the host PC.

Chapter 3

Introduction to the Transputer Language "Occam"

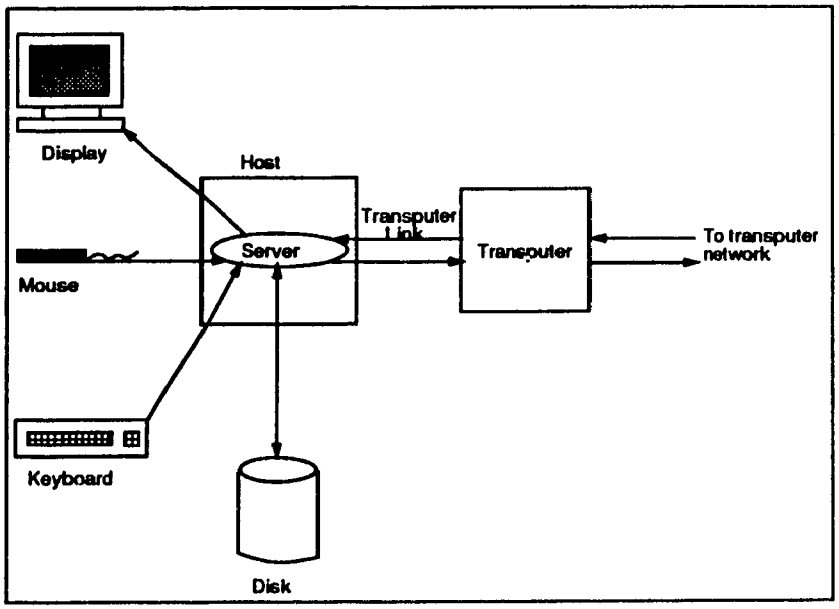


Figure 3.1: Host-server model: the host is connected to the transputer network by a single link

During a program development process it is necessary that some operating system facilities be available, such as access to a disk filing system, input devices and terminals, and that there should be facilities to run text editors, assemblers, high-level language compilers and

debuggers. One way to provide these facilities is to use a host, running a server program that communicates with the transputer system. Porting the embedded system to the host requires a server program to provide the communication between the host and the embedded system. The server model should be user friendly to allow developers to continue to use the facilities of a familiar operating system with its command language and utility programs.

Figure 3.1 demonstrates the transputer host-server model. The first transputer host server model introduced was the:**INMOS** Transputer Development System (TDS). The TDS included apart from a server, an integrated **occam** programming environment which includes a compiler, an editor, a linker, and a configurer. The server ran on a PC host, communicating with an INMOS B004 board containing a T414-15 with 2 Mbytes of RAM interfaced by a link adaptor to the PC bus.

A successor to the TDS server is a more general purpose server called the 'iserver'. This is also available with a number of stand alone tools such as compilers, editors, linkers, etc···. It can run on any computer system providing that the transputer link can be interfaced with it. The server supports a protocol through the link to provide access to the screen, keyboard and the host file system. This protocol is used by the various tools that run on the transputer. Whenever the editor, for example, needs to read a file it calls a set of **occam** routines which send a request to the server. The server locates the file by calling the host operating system and sends it through the link to the editor. The editor then uses other **occam** routines which interact with the server to display characters on the host's screen and to take input from the keyboard.

The principal language provided with the TDS is **occam**, a high-level language that was specially designed for the transputer. Many INMOS documents describe **occam** as the best possible language for programming the transputer. For several years **occam** was the only one available with the TDS.

The name **occam** is derived from William of Occam (or Ockam,*c.* 1270-1349), an English scholar and philosopher. It was he who originated Occam's Razor, which states in its most familiar form, 'Entities are not to be multiplied without reason' [25]. The maximum

performance gained was never actually stated in this form by Occam, but he did say, 'it is vain to do with more what can be done with fewer'.

Occam 2 (version 2) provides most of the features that one can expect from a high-level language. However, many programmers used to Pascal or C will find **occam** different; there are no recursions, structures, records (except in input and output), dynamic memory allocation or user-defined types. On the other hand, **occam** provides access to some of the transputer's facilities in a very clear and simple way, in particular, it supports multiple concurrent processes, multiple processors, inter-process communications and process scheduling.

Although INMOS has been a very strong supporter of **occam**, many programmers have preferred to continue with the more widely used programming languages, such as FORTRAN and C. This is due to the difficulty that some programmers find in working with **occam** and TDS environment and mainly because of the lack of the existing of **occam** software libraries. However, standard FORTRAN and C libraries contain little if any support for parallel programming.

Two main ways of adding parallel processing support to conventional languages have found favor with software developers:

- Using the language unchanged and adding parallel programming support through run-time libraries.
- Altering the semantics of the language to add parallel programming constructs.

Moreover, a compiler for a stand alone transputer system must support the server protocol, multi-tasking, internal and external channels, timers, and the distribution of code over a network of transputers. The reader may find a brief description of **occam** in the following sections which is not intended to be a full tutorial on **occam**. INMOS provides a detailed tutorial on **occam** [43].

3.1 Concurrency

By performing certain operations according to a set of instructions, a traditional single processor computer executes theses instructions one at a time (sequentially). Even the simple operation such as the addition of 2 + 2 is a model of the real world, except when it is performed by or for a mathematician who is interested in the pure properties of numbers. Far more frequently this operation is a model for the act of adding two pounds, dollars, apples or airplanes, to an existing stock of two. Certainly the major applications of computers, such as accounting, banking, process control and even word processing, are explicitly modeling objects, events and activities in the real world.

The world which we live in is inherently concurrent. At the scale of human affairs, the world can be described as a union of time and space. Events are mapped in time and space. It is possible for two events to occur in the same place one after the other in time (sequentially), and equally possible events can occur in different places at the same time (concurrently, or in parallel).

3.2 Occam

Occam is a transputer programming language that allows the programmer to define a program as a collection of parallel tasks. An **occam** program specifies each task, and task requirements without specifying the order of performing individual tasks. The **occam** model of computation is based on communicating processes [23]. Processes have distinct memory locations. In addition, a mild degree of memory sharing is allowed in case two or more concurrent processes are required to read the same memory location at the same time. However, if any process writes to a specific memory location, no concurrent process can read from that memory location at the same time. Messages are passed between processes via channels. A process sending a message, or a process attempting to receive a message, will wait until the transaction is complete. Thus messages inherently act to synchronize processes.

As well as explicit support for concurrent programming on a single transputer, the

occam language allows allocation of processes to processors (configuration) to allow the user to compile and configure a program that will run on a network of transputers. This chapter contains an outline description of the transputer language (**occam** 2) which we used for the software part of this implementation study. If the reader is interested to learn more about **occam** 2 he is encouraged to read [43] and [28].

3.2.1 Processes

An **occam** program consists of one or more concurrently executing processes. Each process performs a sequence of actions, and then terminates. Each action may be an assignment (an assignment changes the value of a variable), an input (an input receives a value from a channel) or an output (an output sends a value to a channel).

If no input is available from the channel, the process will wait. Similarly, if the process at the other end of the channel is unable to receive the output, the sending process will wait. Thus input and output provide both data transfer between processes and synchronization of processes. Both the sending and receiving process must be ready before the data transfer takes place.

The syntax of the assignment process is the := operator:

variable := expression

Input, is designated by the ? operator:

channel ? variable

and output by the ! operator:

channel ! variable

There are two special processes. The **SKIP** process starts to execute, does nothing and then terminates. Thus it is equivalent to a **NOP** in assembly language programming. The **STOP** process starts to execute, does nothing but never terminates thus effectively stops the processor from looking for another instruction. We will see in the following sections how these special processes are used.

3.2.2 Process sequences

Most of the conventional languages assume that statements will be executed one after the other in sequence. However, in **occam** this is not necessarily so, and the sequential nature of a process must be stated explicitly by the **SEQ** construct:

```
INT a:
CHAN OF INT chan1, chan2:
SEQ
  chan1 ? a
  a := a * 5
  chan2 ! a
```

The above program fragment (a declaration in this example) declares a to be a variable, of type **INT**. It consists of a sequence of three processes that input a value from channel 1 into a (from some other process), multiplies it by the constant 5, and sends the output to another process through channel 2. The declarations have the scope of the immediately following construct (in **occam** a scope of a construct is identified by its indentation and can be a sequence or parallel construct), and are indented to the same depth. Indentation by two spaces is used to show the scope of the sequential construct; in all cases **occam** uses indentation to show a grouping, where other languages might use **begin.....end or** {.....}.

3.2.3 Parallel processes

Occam processes may also be executed concurrently or "in parallel". This is denoted by the use of the **PAR** construct.

```
INT a, b:
CHAN OF INT chan1:
PAR
  chan1 ! a
```

```
chan1 ? b
```

This has the effect of copying the value of **a** into **b** and is equivalent to the following assignment statement:

```
- declarations
SEQ
  b := a
```

The lexical order of the processes within the **PAR** construct is not important since the processes will start in an arbitrary order. Thus the following code fragment:

```
- - declarations
PAR
  chan1 ? b
  chan1 ! a
```

is exactly equivalent to the above.

The **PAR** construct is a process that will terminate when all of its component processes terminate. Thus a **SKIP** process may be added to the **PAR** construct with essentially no effect. However, adding a **STOP** process would mean that the **PAR** construct would never terminate.

Occam says nothing about which of the processes in the **PAR** construct will be executed first, or which will get a larger share of the processor time. When it is necessary to give one process priority over another, then the **PRI PAR** construct must be used as follows:

```
PRI PAR
  process_a
  process_b
```

This **PRI PAR** construct is limited to two way processes (two way processes are any two processes or constructs that are independent of each other, in the above example

process_a is an independent process of process_b). Whenever the first process can execute it will, if both process are ready to execute simultaneously; the second process will only start to execute after the first has completed or is waiting for an input or output.

The Occam 2 **PRI PAR** is limited to two components (two constructs with the same indentation, and can be SEQ, PAR or PRI PAR constructs, two simple statements or any two combinations of the aforementioned), which map directly on to the high- and low-priority processes of the underlying hardware. If we want one set of processes to execute at high priority and another set at low, then **PAR** constructs must be nested in the **PRI PAR** as follows:

```
PRI PAR
  PAR
    high_1
    high_2
  PAR
    low_1
    low_2
```

In transputer implementation of Occam 2, a high-priority process will not be time sliced but will execute until completion or waiting for input or output. Only when all high priority processes are unable to execute will the low-priority processes get their time slices. Because of the limitations of this mechanism, **PRI PAR** can only be used at the outermost level of the program, not within any enclosing **PAR** constructs.

3.2.4 Arrays of processes

The **FOR** construct creates an array of processes which can operate in sequence or in parallel.

```
SEQ input = 1 FOR 3
  chan[input] ? buffer[input]
```

creates an array of three sequentially executing processes, each input from a channel in the array **chan** into a corresponding cell of the array **buffer**. This is equivalent to:

```
SEQ
  chan[1] ? buffer[1]
  chan[2] ? buffer[2]
  chan[3] ? buffer[3]
```

Thus the **SEQ ..... FOR** construct acts in a similar fashion to the **FOR** loop in Pascal or other high-level languages.

The **FOR** construct can also be used with **PAR**, such that:

```
PAR input = 1 FOR 3
  chan[input] ? buffer[input]
```

creates an array of three concurrently executing processes.

The loop limits in **PAR.....FOR.....** must be constant integers, as **occam** 2 does not allow the dynamic creation of processes; the total number of processes must be known at compile time.

3.2.5 Channel protocols

A channel transfers data between two concurrent processes. The format and type of this data is specified by the channel protocol. The channel protocol is specified when the channel is declared. Each input and output must be compatible with the protocol of the channel used. Channel protocols enable the compiler to check the usage of the channels.

An **occam** 2 channel is declared with a **PROTOCOL** as in the following examples:

```
CHAN OF BYTE a:
CHAN OF INT a:
CHAN OF REAL64 data:
```

The above protocols declare that the channels will be used only for input or output of a single variable of the appropriate type.

A protocol can also be of array type:

```
CHAN OF [20]INT data:
```

which will receive or transmit streams of **20 INTs.**

When the size of the array to be transmitted is unknown, it is possible to declare a counted array protocol which consists of an integer describing the size of that array, followed by the number of that array elements:

```
CHAN OF INT::[]INT counted.chan :
  INT vector :
```

When data is transmitted, the first word must be the number of array elements to be sent.

```
- - declarations
SEQ
  counted.chan ! 2 :: vector
```

The process in the above program fragment will send a count of 2, followed by the first two elements of the array vector. Similarly, the count is the first data item to be received. Thus, the following program fragment will replace the number of data elements received in the variable **itemcount**, and the number of elements in the array **datain.**

```
INT itemcount
[20] INT datain :
SEQ
  counted.chan ? itemcount :: datain
```

A protocol can be either a sequence of variables of the same or different types. This is achieved through the declaration of a sequential protocol, for example:

```
PROTOCOL DataPacket IS BYTE; REAL32; REAL32; INT:
```

is a protocol declaration in which the elements are separated by semicolons. The protocol

thus declared can be used in the channel initialization:

```
CHAN OF DataPacket InStream :
```

which can only be used for the input and output of streams of the sequence:

BYTE, REAL32, REAL32, INT.

3.2.6 Timers

A timer in **occam** is treated as a channel. It is possible to declare it singly:

```
TIMER working :
```

or as an array:

```
TIMER [20]intervals :
```

Each timer can be read as if it is a channel returning a single integer value:

```
INT start, end :
SEQ
  working ? start
  intervals[2] ? end
```

This process will read the value of the timer appropriate to the priority of the process in which the timer input occurs. However, when comparing times read in this way, it must be remembered that the number of timer pulses is read as an unsigned **INT**, with a number of bits equal to the word length of the transputer. Thus it is possible that the timer will roll over during any timing interval. **Occam** for this reason provides an **AFTER** operator which causes a process to wait until the timer reaches a particular value:

```
working ? AFTER timeout
```

where timeout contains the value of the timer which will be waited for. Thus, to suspend execution for, e.g. 1000 pulses one can use the following code:

```
TIMER s:
```

```
VAL INT wait IS 1000 :
INT now :
SEQ
  s ? now
  s ? AFTER now PLUS wait
```

where **PLUS** denotes unsigned addition. If the wait is required to be in seconds it must be computed using the number of pulses per second of that particular timer.

3.2.7 Placement

Occam contains features that allow specifying the position that variables occupy in the processor's memory, and also on which processor a particular process will be executed. This is know as placement.

The hardware memory map of the transputer is *byte*-addressed, with signed addresses running from **MinInt** to **MaxInt**. In the **occam** memory map, addresses are unsigned *words*, running from 0 to the top of the address space. One can use the **PLACE** keyword to assign a memory address to a variable.

```
INT i :
PLACE i AT 28 :
```

The above code fragment will place the integer variable **i** in the hardware memory address **0x80000070** on a 32-bit processor, or **0x8038** on a 16-bit processor. This is useful in allocating variables to the on-chip fast **RAM**. The disadvantage of this method is it tends to compromise the security of **occam**, as two processes can place variables at the same address, and can access them with no control.

This type of placement is most useful in associating channels with hardware links. The link control words lie in the bottom eight words of the memory map,

Control Word	**Occam** address	Hardware address	
		16-bit	32-bit
Link0Out	0	0x8000	0x80000000
Link1Out	1	0x8002	0x80000004
Link2Out	2	0x8004	0x80000008
Link3Out	3	0x8006	0x8000000c
Link0In	4	0x8008	0x80000010
Link1In	5	0x800A	0x80000014
Link2In	6	0x800C	0x80000018
Link3In	7	0x800E	0x8000001C

Table 3.1: **Occam** and hardware addresses of link control words

as shown in Table 3.1. For example:

```
CHAN OF INT chanin, chanout :
PLACE chanin AT 7:
PLACE chanout AT 3:
```

This set of instructions will place the channels **chanin** and **chanout** on hardware link 3 input and output respectively. This is the actual input and output set up implemented by the author in the control program he wrote for the physical application phase of this thesis.

Placement of variables can also be used to gain access to memory-mapped peripherals. However, this is considered unsafe as more than one process can gain access to the peripheral in an uncontrolled manner. **Occam** standard gives no guarantee that a variable name appearing is an expression will only be accessed once. It is therefore much safer to treat memory-mapped devices as **PORT**s, which are extensions of the **occam** channel concept to locations in memory. If a **PORT** is declared and placed at an address, it is read and written using channel input and output:

```
PORT OF INT memloc :
PLACE memloc at #10000000 :
INT i :
SEQ
```

```
memloc ? i
```

This code will read the contents of the specified memory location into **i**. It guarantees that only one read access will be made to **memloc**.

Chapter 4

Kinematics and Dynamics of the manipulator

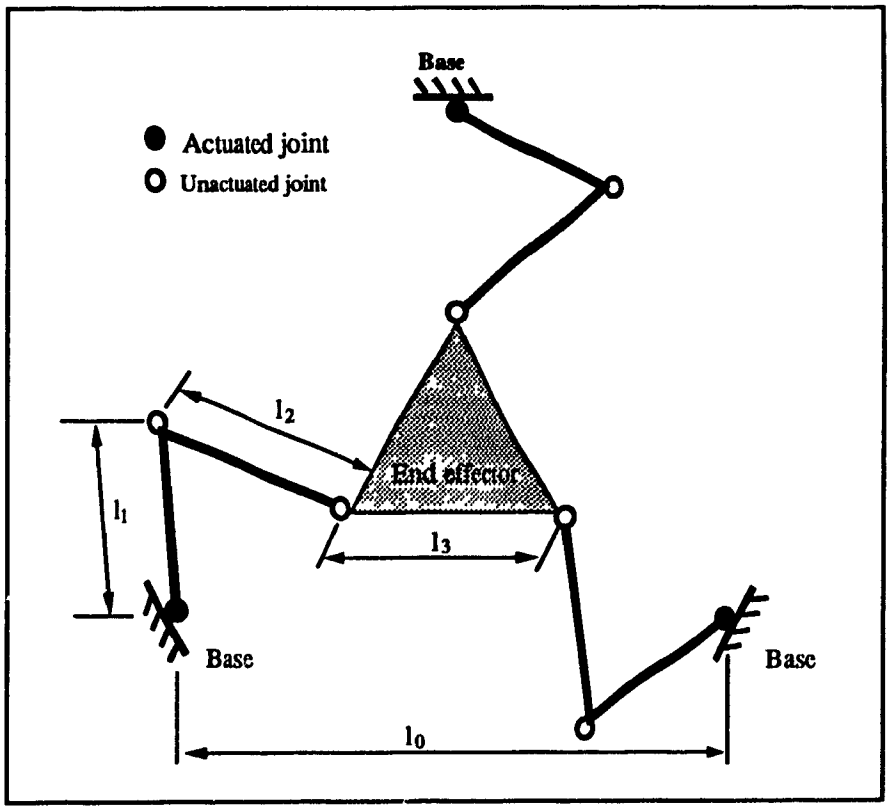


Figure 4.1: A 3-DOF Planar Parallel Manipulator

A typical architecture of the manipulator which is the subject of our study is shown in figure 4.1. It is composed of seven movable links, i.e., $r = 7$, and nine revolute joints, i.e., $m = 9$. The motions of all links are limited in one plane parallel to the plane of the manipulators base. According to the Chebyshev-Grübler-Kutzbach formula [22], it can be easily verified that the degree-of-freedom of this manipulator is three, namely,

$$n = 3r - 2m = 3 \tag{4.1}$$

As :

r: number of movable rigid bodies.

m: number of one-degree-of-freedom joints.

n: number of degrees of freedom of the system.

The three links connected to the base are considered as input links, while the one with three joints is the end effector, which undergoes a 3-DOF planar motion. Moreover, the manipulator is driven by three motors which are located at the three fixed joints connecting the input links to the base. Hence, these three joints, the *actuated joints*, are independent joints, while the others, the *unactuated joints*, are dependent joints. This means that once the variables associated with the former are assigned, those associated with the latter are fixed.

This manipulator, could be considered a typical example of planar multi-loop mechanical systems. It was first introduced by Hunt [24] and afterwards attracted the attention of many researchers. Yang and Lee [52] investigated the kinematic feasibility of this type of manipulators based on fundamental mechanism theory. An optimum design method of this type of manipulators was presented by Gosselin and Angeles [18]. Later, studies on direct kinematics and on inverse dynamics of parallel manipulators similar to our manipulator were reported by Do and Yang [12], Lee and Chao [33], Lee and Shah [31] [32] and Stoughton and Kokkinis [29]. The approach for direct kinematics presented in the above-mentioned

works was rather straightforward by solving all simultaneous constraint equations numerically. This type of approach was inefficient for real-time computations and does not provide any means of controlling the solution branches, which may lead to physically infeasible results as indicated by Waldron [50]. In the work by Ma and Angeles [37] an efficient method of direct kinematics calculation was introduced. In this method, one of the manipulator links is removed, so that only one of the three kinematic loops remain. Thus, the nonlinear constraint equations which must be solved for the determination of the unactuated joint displacements are reduced to one. This remaining constraint equation could be solved in a closed form. Based on the rigidity condition of the removed link, the removed constraint is then recovered by introducing a scalar distance constraint. By applying a technique of four-bar linkage-performance evaluation, the resulting distance constraint contains only one variable and so could be solved very efficiently. The solution which lies in a physically unreachable branch can be avoided.

The inverse dynamics of our manipulator has been studied by Do and Yang [12], Lee and Chao [33], Williams and Reinholtz [51], but the direct dynamics of this manipulator was not published until Ma [36] introduced a method in which the equations of motion of the manipulator were formulated in terms of the actuated joint coordinates using the natural orthogonal complement, (Natural orthogonal complement was introduced by Angeles and Lee [2], Angeles and Ma [3]). The dynamics model was derived in the form of the Euler-Lagrange equations.

4.1 Direct kinematics analysis

4.1.1 Joint Coordinates

For the kinematic modeling, the seven movable links are numbered from 1 to 7, as indicated in figure 4.2. Each joint is assigned a joint coordinate, denoted by q_1, q_2, $\cdots$, q_9, as indicated in the same figure. In the figure, point P is a reference point defined at the centroid of the end-effector. In this case the vectors of the independent and dependent joint coordinates are

given by:

$$\mathbf{q}^a = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}, \qquad \mathbf{q}^u = \begin{bmatrix} q_4 \\ \vdots \\ q_9 \end{bmatrix}. \tag{4.2}$$

$\mathbf{q}^a$ consists of independent generalized coordinates, which are associated with the actuated joints, and $\mathbf{q}^u$ is a 6-dimensional vector of dependent generalized coordinates associated with the unactuated joints.

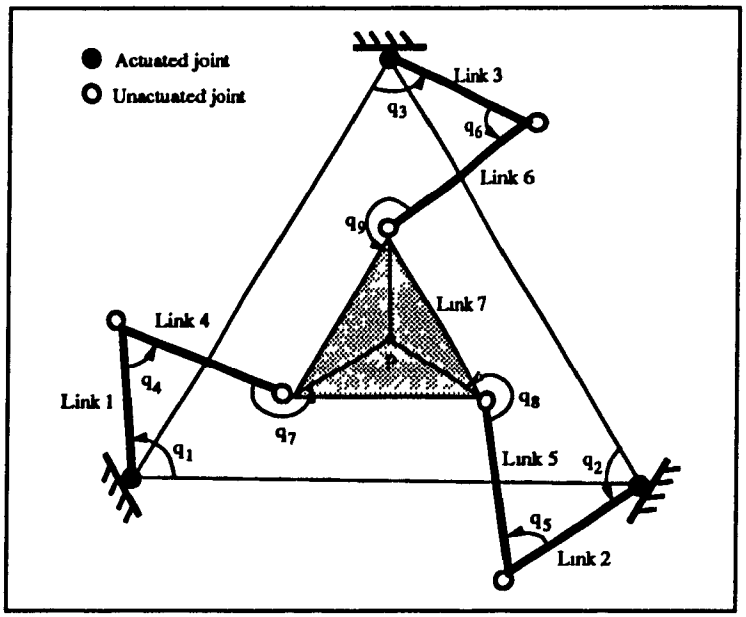


Figure 4.2: Joint coordinates of the 3-DOF Planar Manipulator.

It was assumed that the architecture of the manipulator has symmetric geometry, (the three fixed joints are located at the three apexes of an equilateral triangle with edges of length l_0; the three input links have the same length l_1; the other three links following the input ones have length l_2, while the end effector is an equilateral triangle with edges of length l_3.), as shown in figure 4.1.

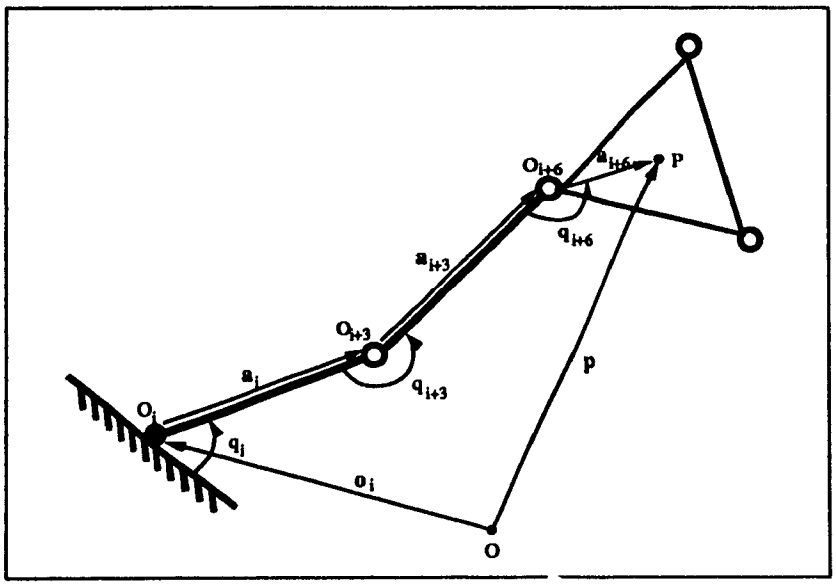


Figure 4.3: Position vectors

4.1.2 Displacement analysis

Since the manipulator under study contains nine joints, it requires nine *generalized coordinates*, grouped in a 9-dimensional vector $\mathbf{q}$, to represent the kinematic relationship between individual links. Each generalized coordinate represents the rotational displacement of a joint. Because of the presence of closed kinematic loops, some generalized coordinates depend on the others. Hence, $\mathbf{q}$ can be partitioned as:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}^a \\ \mathbf{q}^u \end{bmatrix} \tag{4.3}$$

These generalized coordinates are subjected to kinematic constraints which can be described by a set of holonomic *constraint equations* of the general form:

$$\phi(\mathbf{q}^a, \mathbf{q}^u) = \mathbf{0}. \tag{4.4}$$

This system was found to have six nonlinear scalar constraint equations as the number of independent equations should be equal to the dimension of $\mathbf{q}^u$, i.e. six, which were used to solve for the six dependent joint coordinates. A set of intermediate variables, ψ_1, ψ_2 and

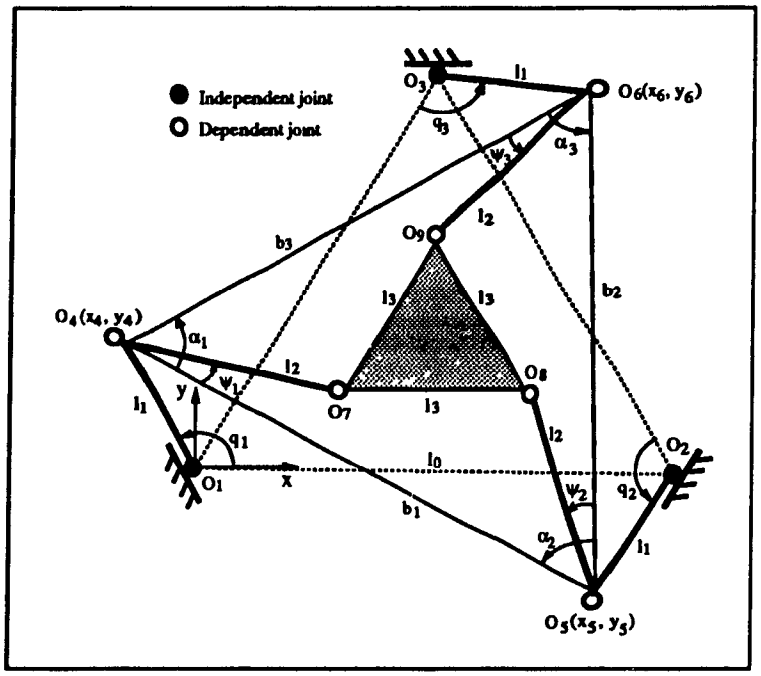


Figure 4.4: The dimensional notation of our 3-DOF Parallel Manipulator

ψ_3, were introduced and shown in figure 4.4. In that figure, O_i denotes the ith joint, for $i = 1, 2, \cdots, 9$. By the use of these three variables, three independent constraint equations were formulated which represent *input-output equations* of the three *RRRR* planar four-bar linkages, $O_4O_7O_8O_5$, $O_5O_8O_9O_6$, and $O_6O_9O_7O_4$, respectively. The positions of joints 4, 5 and 6 are fixed if the three independent joint coordinates q_1, q_2 and q_3 are given. The rest of the manipulator was considered as a structure consisting of three coupled four-bar planar linkages whose input angles are ψ_1, ψ_2 and ψ_3, respectively. Using the notation shown in figure 4.4, the three constraint equations were formulated :

$$\phi_1 = k_{11} - k_{12}\cos\psi_1 + k_{13}\cos(\alpha_2 - \psi_2) + \cos(\psi_1 + \alpha_2 - \psi_2) = 0, \qquad (4.5a)$$

$$\phi_2 = k_{21} - k_{22}\cos\psi_2 + k_{23}\cos(\alpha_3 - \psi_3) + \cos(\psi_2 + \alpha_3 - \psi_3) = 0, \qquad (4.5b)$$

$$\phi_3 = k_{31} - k_{32}\cos\psi_3 + k_{33}\cos(\alpha_1 - \psi_1) + \cos(\psi_3 + \alpha_1 - \psi_1) = 0, \qquad (4.5c)$$

where

$$k_{i1} = \frac{b_i^2 + 2l_2^2 - l_3^2}{2l_2^2}, \quad k_{i2} = k_{i3} = \frac{b_i}{l_2}, \quad \text{for } i = 1,\ 2,\ 3; \tag{4.6}$$

$$\alpha_1 = \cos^{-1}\left[\frac{(x_5 - x_4)(x_6 - x_4) + (y_5 - y_4)(y_6 - y_4)}{b_3 b_1}\right], \tag{4.7}$$

$$\alpha_2 = \cos^{-1}\left[\frac{(x_6 - x_5)(x_4 - x_5) + (y_6 - y_5)(y_4 - y_5)}{b_1 b_2}\right], \tag{4.8}$$

$$\alpha_3 = \cos^{-1}\left[\frac{(x_4 - x_6)(x_5 - x_6) + (y_4 - y_6)(y_5 - y_6)}{b_2 b_3}\right]; \tag{4.9}$$

$$b_1 = \sqrt{(x_5 - x_4)^2 + (y_5 - y_4)^2}, \tag{4.10}$$

$$b_2 = \sqrt{(x_6 - x_5)^2 + (y_6 - y_5)^2}, \tag{4.11}$$

$$b_3 = \sqrt{(x_4 - x_6)^2 + (y_4 - y_6)^2}. \tag{4.12}$$

In the equations above, x_i and y_i are the Cartesian coordinates of joint O_i, for $i = 4,\ 5,\ 6$, which we could calculate. For example, if the Cartesian x-y frame is set as shown in figure 4.4, we get,

$$x_4 = l_1 \cos q_1, \qquad y_4 = l_1 \sin q_1, \tag{4.13}$$

$$x_5 = -\frac{l_1}{2}(\cos q_2 - \sqrt{3}\sin q_2) + l_0, \quad y_5 = \frac{l_1}{2}(\sqrt{3}\cos q_2 - \sin q_2), \tag{4.14}$$

$$x_6 = -\frac{l_1}{2}(\cos q_3 + \sqrt{3}\sin q_3) + \frac{l_0}{2}, \quad y_6 = -\frac{l_1}{2}(\cos q_3 - \sqrt{3}\sin q_3) + \frac{\sqrt{3}}{2}l_0. \tag{4.15}$$

It was found that once ψ_1, ψ_2 and ψ_3 were computed, the dependent joint coordinates $q_4, q_5, \cdots, q_9$ could be found as well and also the position of the end effector. In such a method only three nonlinear equations are involved. The Newton-Raphson method was utilized to solve them numerically (the Newton-Raphson method is defined in appendix A) [1]. By numerically solving eq. (4.5), which involves the inversion of a 3×3 Jacobian matrix at each iteration, reduces the computational cost to approximately $1/8$ of that required for inverting the 6×6 Jacobian involved when solving the original constraint system. Although this improves the computational efficiency substantially, the complexity involved is still high.

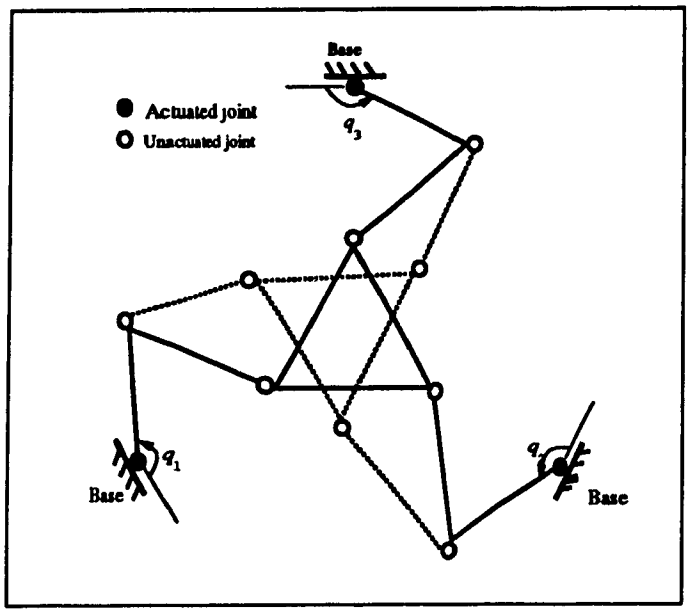


Figure 4.5: Example of configuration solution branches.

Moreover, in solving the 6×6 system, the branches of solutions are still uncontrollable because the numerical procedure works without regard to the position of the end effector. An example of branches of solutions to the direct displacement problem is shown in figure 4.5.

From that figure, we see that the end effector has two different configurations for a set of given input displacements q_1, q_2 and q_3, and the number of solutions of eq. (4.4) can be up to sixty-four because eq. (4.4) can be formulated as a set of six simultaneous quadratic scalar equations. Of course, most of the sixty-four solutions are not physically feasible for a given manipulator, because they belong to different configuration branches, while a motion from one branch-configuration to another may be impossible unless the mechanism is disassembled. Thus, it is important and practical to find a solution lying in a desired branch — *control of solution branches.* General numerical methods do not have the capability of controlling solution branches.

However, García de Jalón et al. [17], proposed a model of constraint equations using *natural coordinates*, which are defined as the Cartesian coordinates of one point of each joint axis and a unit vector parallel to that axis. Although the constraint equations based on such

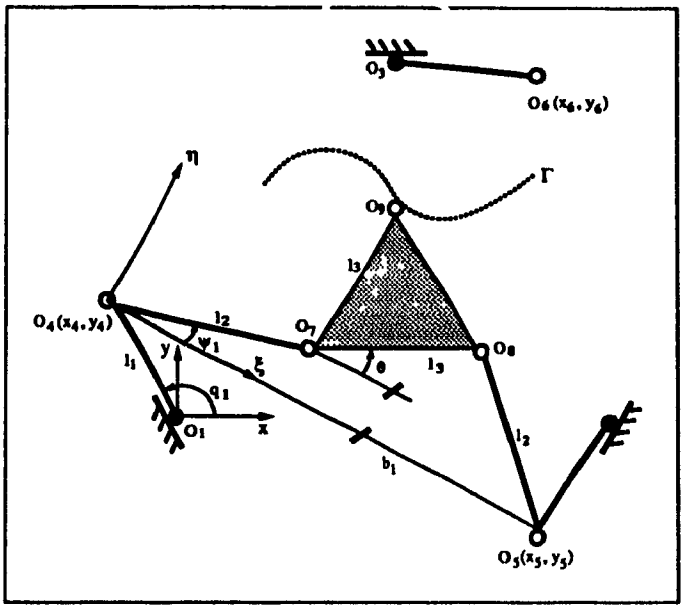


Figure 4.6: The system after the virtual cut

coordinates are simpler, the numbers of equations and coordinates increase, and so does the dimension of the dynamics equations involved. Hence, solving this larger system numerically will be costlier.

Ma [36], has introduced another method, which applies the technique of linkage-performance evaluation introduced by Ma and Angeles [35]. This method assumes virtually removing one link, figure. 4.6. This means eliminating one of the kinematic constraints, resulting in reducing the original system kinematic constraints to one. Which will make eq. (4.5a) the only effective equation. Then we would have the intermediate angle, θ, indicated in figure 4.6, as:

$$\phi_1 = k'_{11} - k'_{12}\cos\psi_1 - k'_{13}\cos\theta + \cos(\psi_1 - \theta) = 0 \tag{4.16}$$

where

$$k'_{11} = \frac{b_1^2 + l_3^2}{2l_2l_3}, \qquad k'_{12} = \frac{b_1}{l_3}, \qquad k'_{13} = \frac{b_1}{l_2}. \tag{4.17}$$

Notice that variable ψ_2 disappears in the new constraint equation, and angle θ can be

solved from this equation in closed form. As well as the degree of freedom of the manipulator increases from 0 to 1, and joint 9 describes a trajectory, Γ(shown in figure 4.6), which is the *coupler curve* of the four-bar linkage, discussed by Nolle [41], $O_4O_7O_8O_5$.

A coordinate frame, ξ-η, (figure 4.6), was defined and an angle ψ_1 was considered as the input angle of the four-bar linkage. Then the coupler curve was defined as :

$$\Gamma = \{\mathbf{o}_9(\psi_1) : \ \psi_1 \in [\psi_a, \ \psi_b]\} \tag{4.18}$$

in the local coordinate frame. where $[\psi_a, \ \psi_b]$ is the *mobility range* of angle ψ_1, while $\mathbf{o}_9$ is the position vector (figure 4.3) of the 9th joint O_9:

$$\mathbf{o}_9(\psi_1) \equiv \begin{bmatrix} \xi_9 \\ \eta_9 \end{bmatrix} = \begin{bmatrix} l_2 \cos\psi_1 + l_3 \cos(\theta + \pi/3) \\ l_2 \sin\psi_1 + l_3 \sin(\theta + \pi/3) \end{bmatrix}. \tag{4.19}$$

And angle θ computed from eq. (4.16), in terms of angle ψ_1, as

$$\theta = 2 \arctan\left(\frac{b + K\sqrt{b^2 - ac}}{a}\right), \tag{4.20}$$

where

$$a = -k'_{11} + (k'_{12} + 1)\cos\psi_1 - k'_{13}, \tag{4.21}$$

$$b = \sin\psi_1, \tag{4.22}$$

$$c = -k'_{11} + (k'_{12} - 1)\cos\psi_1 + k'_{13}. \tag{4.23}$$

Parameter K, termed the *branching index* (by Ma and Angeles [35]) in eq. (4.20), is equal to either $+1$ or -1, depending on which configuration branch the linkage is in. figure4.7(a) shows two branch curves of a linkage. In this case, there are two output configurations for one given input angle ψ_1, one corresponding to the branching index of $+1$ and the other to -1. Figure 4.7(b), also shows the situation when the two branching curves merge into one, which will cause a part of the curve to have the branching index $+1$ and the

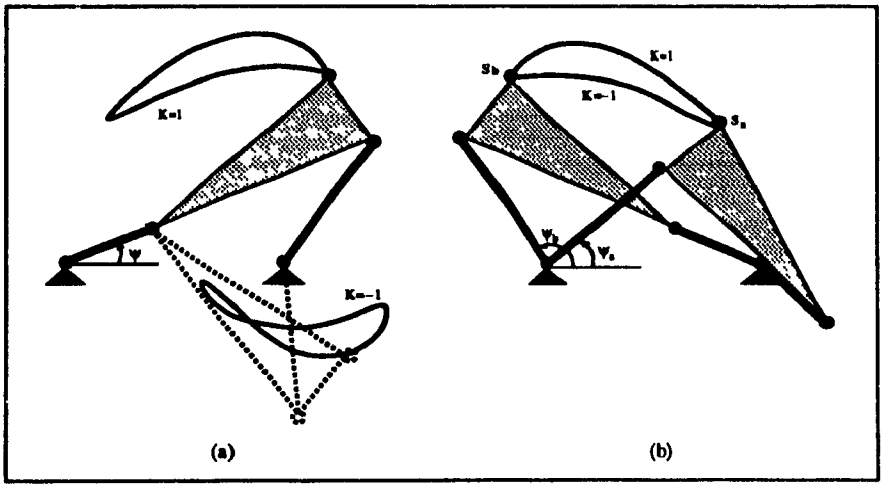


Figure 4.7: The coupler curve of a four-bar linkage branches

other -1. In such a case, if the solution of ψ_1 is equal to either ψ_a or ψ_b, (either of the two bounds of the mobility range of ψ_1), we would have a singular configuration, as the output motion of the linkage is uncertain. Two singular points S_a and S_b are shown in figure 4.7(b). We can find the other singular points of the other loops by removing other links. (More on this manipulator singularity can be found in Gosselin and Angeles [19]).

To recover the removed constraint, we find on the coupler curve a point whose distance to the 6th joint, O_6, is equal to the length of the removed link :

$$\sqrt{(\xi_9 - \xi_6)^2 + (\eta_9 - \eta_6)^2} = l_1 \tag{4.24}$$

where ξ_6 and η_6 are the coordinates of joint 6 in the local frame ξ-η. By solving eq. (4.24) numerically, one can find ψ_1. The numerical solution for ψ_1 can be obtained more efficiently than by solving eq. (4.4) or eq. (4.5) as eq. (4.24) is a single-variable scalar equation.

The aforementioned coupler curve is only conceptual, because one actually does not need to compute the whole coupler curve when solving ψ_1 from eq. (4.24). Instead, only the evaluation of eq. (4.20) is required when solving eq. (4.24), the branch of the solution being controlled by specifying, based on the assembly configuration of the given manipulator, the index K of that equation.

By obtaining ψ_1 and θ, the determination of the coordinates of all moving joints is possible. (Further details can be found in Ma [36]).

4.1.3 Velocity and acceleration analyses

Let :

$\mathbf{a}_i$: The vector connecting the ith joint O_i to one of its neighboring joints.

C_i: the center of mass of the ith link.

$\mathbf{o}_i$: The position vector of O_i, in the X-Y frame.

$\mathbf{c}_i$: The position vector of C_i, in the X-Y frame.

ω_i: Scalar denotes the angular velocity of the ith link.

$\dot{\omega}_i$: Scalar denotes the angular acceleration of the ith link.

One can find the relation between the independent and the dependent angular velocities by noting that the velocity of the 9th joint, $\dot{o}_9$, (figure 4.8), can be expressed in either of the following three different forms:

$$\dot{\mathbf{o}}_9 = \omega_1 \mathbf{E}\mathbf{a}_1 + \omega_4 \mathbf{E}\mathbf{a}_4 + \omega_7 \mathbf{E}\mathbf{a}_7 \tag{4.25}$$

$$\dot{\mathbf{o}}_9 = \omega_2 \mathbf{E}\mathbf{a}_2 + \omega_5 \mathbf{E}\mathbf{a}_5 + \omega_7 \mathbf{E}\mathbf{a}_8 \tag{4.26}$$

$$\dot{\mathbf{o}}_9 = \omega_3 \mathbf{E}\mathbf{a}_3 + \omega_6 \mathbf{E}\mathbf{a}_6 \tag{4.27}$$

where $\mathbf{E}$ is an orthogonal matrix which rotates the vectors 90° counterclockwise, without changing their magnitudes:

$$\mathbf{E} \equiv \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{4.28}$$

From the above equations we could deduce :

$$\mathbf{A}\boldsymbol{\omega}^u = \mathbf{B}\boldsymbol{\omega}^a \tag{4.29}$$

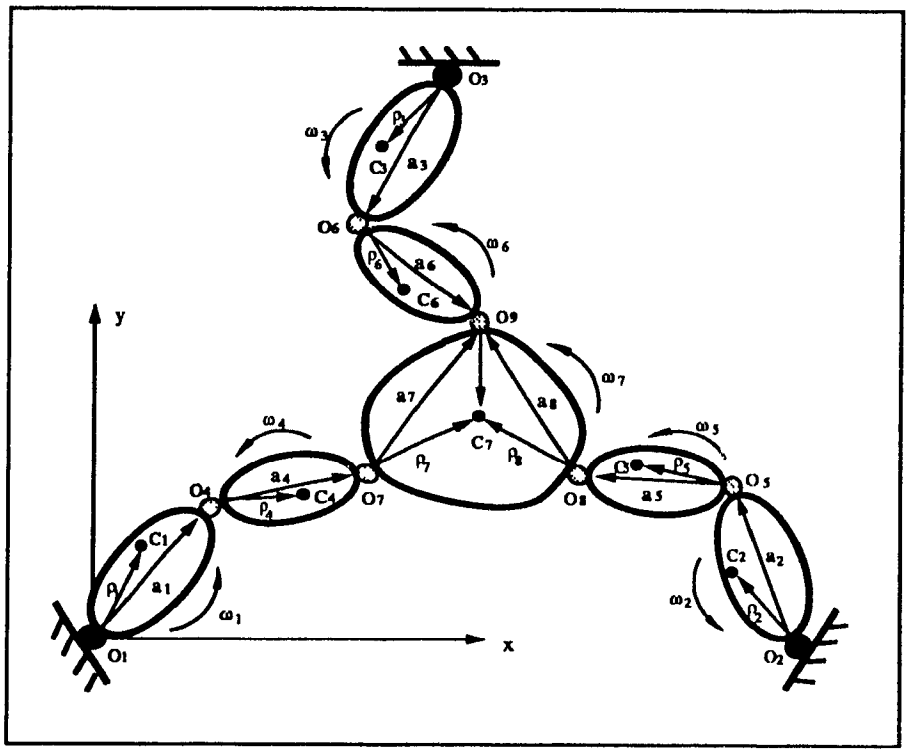


Figure 4.8: Dynamics analysis notations

as:

$$\boldsymbol{\omega}^a \equiv \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}, \qquad \boldsymbol{\omega}^u \equiv \begin{bmatrix} \omega_4 \\ \omega_5 \\ \omega_6 \\ \omega_7 \end{bmatrix}, \tag{4.30}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_4 & \mathbf{0} & -\mathbf{a}_6 & \mathbf{a}_7 \\ \mathbf{0} & \mathbf{a}_5 & -\mathbf{a}_6 & \mathbf{a}_8 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -\mathbf{a}_1 & \mathbf{0} & \mathbf{a}_3 \\ \mathbf{0} & -\mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix}. \tag{4.31}$$

To compute the angular accelerations, $\dot{\omega}_i$, we can differentiate both sides of eq. (4.29) with respect to time:

$$\mathbf{A}\dot{\boldsymbol{\omega}}^u = \dot{\mathbf{B}}\boldsymbol{\omega}^a + \mathbf{B}\dot{\boldsymbol{\omega}}^a - \dot{\mathbf{A}}\boldsymbol{\omega}^u, \tag{4.32}$$

in which

$$\dot{\omega}^a \equiv \begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix}, \qquad \dot{\omega}^u \equiv \begin{bmatrix} \dot{\omega}_4 \\ \dot{\omega}_5 \\ \dot{\omega}_6 \\ \dot{\omega}_7 \end{bmatrix}, \tag{4.33}$$

and

$$\dot{\mathbf{A}} \equiv \frac{d}{dt}\mathbf{A} = \begin{bmatrix} \omega_4 \mathbf{E}\mathbf{a}_4 & \mathbf{0} & -\omega_6 \mathbf{E}\mathbf{a}_6 & \omega_7 \mathbf{E}\mathbf{a}_7 \\ \mathbf{0} & \omega_5 \mathbf{E}\mathbf{a}_5 & -\omega_6 \mathbf{E}\mathbf{a}_6 & \omega_7 \mathbf{E}\mathbf{a}_8 \end{bmatrix}$$
$$= \mathbf{P}\mathbf{A}\mathrm{diag}(\omega_4,\ \omega_5,\ \omega_6,\ \omega_7), \tag{4.34}$$

$$\dot{\mathbf{B}} \equiv \frac{d}{dt}\mathbf{B} = \begin{bmatrix} -\omega_1 \mathbf{E}\mathbf{a}_1 & \mathbf{0} & \omega_3 \mathbf{E}\mathbf{a}_3 \\ \mathbf{0} & -\omega_2 \mathbf{E}\mathbf{a}_2 & \omega_3 \mathbf{E}\mathbf{a}_3 \end{bmatrix}$$
$$= \mathbf{P}\mathbf{B}\mathrm{diag}(\omega_1,\ \omega_2,\ \omega_3). \tag{4.35}$$

In the above equations, $\mathbf{P}$ denotes a 4×4 permutation matrix defined as

$$\mathbf{P} = \begin{bmatrix} \mathbf{E} & \mathbf{O} \\ \mathbf{O} & \mathbf{E} \end{bmatrix} \tag{4.36}$$

and $\mathbf{O}$ is the 2×2 zero matrix. For simplicity of representation, let

$$\mathbf{u}^a \equiv \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \end{bmatrix}, \qquad \mathbf{u}^u \equiv \begin{bmatrix} \omega_4^2 \\ \omega_5^2 \\ \omega_6^2 \\ \omega_7^2 \end{bmatrix}. \tag{4.37}$$

Then, we could rewrite eq. (4.32) :

$$\mathbf{A}\dot{\omega}^u = \mathbf{P}(\mathbf{B}\mathbf{u}^a - \mathbf{A}\mathbf{u}^u) + \mathbf{B}\dot{\omega}^a \tag{4.38}$$

By using the inverted matrix $\mathbf{A}$ when solving eq. (4.29) we could realize that not much additional computation is required.

Once ω^u and $\dot{\omega}^u$ are computed, the remaining part of direct kinematics, i.e., the computation of the center of mass velocity, denoted as $\dot{\mathbf{c}}_i$, and the center of mass acceleration, denoted by $\ddot{\mathbf{c}}_i$, of the ith link, for $i = 1,\ 2,\ \cdots,\ 7$, becomes a straightforward problem.

4.2 Dynamics analysis

4.2.1 Inverse Dynamics

Let:

$\mathbf{c}_i$: 3-dimensional (3-D) position vector of the center of mass of the ith body.

$\dot{\mathbf{c}}_i$: 3-D velocity vector of the center of mass of the ith body.

$\ddot{\mathbf{c}}_i$: 3-D acceleration vector of the center of mass of the ith body.

m_i : mass of the ith body.

$\mathbf{I}_i$: 3×3 inertia tensor of the ith body about its center of mass.

$\mathbf{f}_i^*$: 3-D vector of the inertia force of the ith body at its center of mass.

$\mathbf{n}_i^*$: 3-D vector of the inertia moment of the ith body about its center of mass.

$\mathbf{w}_i^*$: 6-D vector of the inertia wrench of the ith body.

$\boldsymbol{\tau}_i^f$: a 3-D vector friction torque acting on the ith joint.

$\mathbf{w}_i^f$: 6-D vector of the friction wrench of the ith body.

$\mathbf{w}_i^g$: 6-D vector of the gravity wrench of the ith body.

$\boldsymbol{\tau}_i$: a 3-D vector generalized deriving torque of the ith actuated joint. $\boldsymbol{\tau}_i$ is a torque if the ith joint is revolute and a force if the ith joint is prismatic.

$\boldsymbol{\tau}^a$: a 3-D vector generalized driving torque of the whole manipulator.

The dynamics is analyzed in a three-D space, whose first dimension is the rotation about the axis perpendicular to the motion plane and the other two are the translations in the motion plane. In this space, the twist of a link, say the ith link, is a 3-D vector :

$$\mathbf{t}_i = \begin{bmatrix} \omega_i \\ \dot{\mathbf{c}}_i \end{bmatrix} \tag{4.39}$$

where ω_i is a scalar representing the angular velocity of the ith link and $\dot{\mathbf{c}}$ is the 2-D velocity vector of the center of mass of the same link. Moreover, the intended mass matrix reduces to a 3×3 diagonal matrix:

$$\mathbf{M}_i = \begin{bmatrix} I_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{bmatrix} \tag{4.40}$$

where m_i and I_i are the mass and the polar moment of inertia about the center of mass of the ith link, respectively. The inertia wrench, $\mathbf{w}_i^*$ is a 6-D vector by definition:

$$\mathbf{w}_i^* \equiv \begin{bmatrix} \mathbf{n}_i^* \\ \mathbf{f}_i^* \end{bmatrix} = -\mathbf{M}_i\dot{\mathbf{t}}_i - \mathbf{\Omega}_i\mathbf{M}_i\mathbf{t}_i \tag{4.41}$$

and $\mathbf{w}_i^*$ of the ith link also reduces to:

$$\mathbf{w}_i^* = -\mathbf{M}_i\dot{\mathbf{t}}_i = \begin{bmatrix} -I_i\dot{\omega}_i \\ -m_i\ddot{\mathbf{c}}_i \end{bmatrix}, \qquad \text{for } i = 1,\ 2,\ \cdots, 7 \tag{4.42}$$

where $\mathbf{\Omega}_i$ are 6×6 matrices is defined as in its general definition:

$$\mathbf{\Omega}_i = \begin{bmatrix} \omega_i \times \mathbf{1} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix}. \tag{4.43}$$

for this manipulator:

- $\mathbf{\Omega}$, termed the *twist-constraint matrix*, which is of dimension 18×21 and configuration-dependent.
- $\mathbf{O}$ denotes the 3×3 zero matrix,
- $\mathbf{1}$ denotes the 3×3 identity matrix.
- $\omega_i \times \mathbf{1}$ denotes the cross-product matrix associated with vector ω_i.

The generalized twist and wrench vectors are defined as :

$$\mathbf{t} = \begin{bmatrix} \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_7 \end{bmatrix}, \qquad \mathbf{w}^* = \begin{bmatrix} \mathbf{w}_1^* \\ \vdots \\ \mathbf{w}_7^* \end{bmatrix}, \tag{4.44}$$

Since the term $\mathbf{\Omega}_i\mathbf{M}_i\mathbf{t}_i$ is always zero for planar motions, as for any holonomic mechanical systems, the following set of twist constraint equations hold [3]:

$$\mathbf{\Omega t} = \mathbf{0}. \tag{4.45}$$

Also it can also be shown that $\mathbf{t}$ is a linear transformation of the vector of independent generalized velocities, $\dot{\mathbf{q}}_a$, i.e.,

$$\mathbf{t} = \mathbf{T}\dot{\mathbf{q}}_a \tag{4.46}$$

From eq. (4.45) and eq. (4.46), we can show that $\mathbf{\Omega T} = \mathbf{0}$ and hence, the 21×3 matrix $\mathbf{T}$ is termed the *natural orthogonal complement* of matrix $\mathbf{\Omega}$. Moreover, $\mathbf{t}$ can also be expressed as a linear transformation of the vector $\dot{\mathbf{q}}$, i.e.,

$$\mathbf{t} = \mathbf{T}\dot{\mathbf{q}} \tag{4.47}$$

From eq. (4.46) and eq. (4.47) with eq. (4.4), we get

$$\mathbf{T} = \mathbf{K}_a - \mathbf{K}_u\mathbf{J}_u^{-1}\mathbf{J}_a \tag{4.48}$$

where $\mathbf{K}_a$ and $\mathbf{K}_u$ are termed *velocity Jacobian matrices*, while $\mathbf{J}_a$ and $\mathbf{J}_u$ are *displacement Jacobian matrices*, defined as

$$\mathbf{K}_a = \frac{\partial \mathbf{t}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}_a}, \qquad \mathbf{K}_u = \frac{\partial \mathbf{t}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}_u}; \tag{4.49}$$

$$\mathbf{J}_a = \frac{\partial \phi(\mathbf{q})}{\partial \mathbf{q}_a}, \qquad \mathbf{J}_u = \frac{\partial \phi(\mathbf{q})}{\partial \mathbf{q}_u}. \tag{4.50}$$

Evaluation of the orthogonal complement matrix by eq. (4.48) is rather costly. Noticing that $\mathbf{T}$ depends on joint displacements only and also the jth column of $\mathbf{T}$ equals $\partial \mathbf{t}/\partial \dot{q}_j$, for $j = 1,\ 2,\ 3$, one can compute $\mathbf{T}$ as follows:

$$\mathbf{T} = \left[\mathbf{t}\Big|_{\dot{q}_1=1}, \mathbf{t}\Big|_{\dot{q}_2=1}, \mathbf{t}\Big|_{\dot{q}_3=1} \right]_{\text{other } \dot{q}_i'\text{s of } \dot{\mathbf{q}}_a \text{ are zero}} \tag{4.51}$$

i.e., the jth column of $\mathbf{T}$ is calculated as the generalized twist of the manipulator assuming that all the actuated joints are locked but the jth one has a motion with unity velocity. For example, the second column of $\mathbf{T}$ is equal to $\mathbf{t}(\mathbf{q}_a, \dot{\mathbf{q}}_a)$ where $\dot{\mathbf{q}}_a$ is set to $[0, 1, 0]^T$ and $\mathbf{q}_a$ should be kept as is. It was found (Ma [37]) that using this method to compute $\mathbf{T}$ is more efficient and easier than using eq. (4.48)

Let π^a and π^* denote the power supplied by the actuators and the power associated with the generalized inertia force, respectively. Then we get:

$$\pi^a = \dot{\mathbf{q}}_a^T \boldsymbol{\tau}^a, \tag{4.52}$$

$$\pi^* = \mathbf{t}^T \mathbf{w}^* = \dot{\mathbf{q}}_a^T \mathbf{T}^T \mathbf{w}^*. \tag{4.53}$$

From the conservation of energy of the whole system, the following equation holds:

$$\pi^a + \pi^* = 0 \tag{4.54}$$

in other words:

$$\dot{\mathbf{q}}_a^T \boldsymbol{\tau}^a = -\dot{\mathbf{q}}_a^T \mathbf{T}^T \mathbf{w}^*. \tag{4.55}$$

By definition, all components of $\dot{\mathbf{q}}_a$ are independent and hence we derive from eq. (4.55) the following:

$$\boldsymbol{\tau}^a = -\mathbf{T}^T \mathbf{w}^* \tag{4.56}$$

which is the dynamics model of the manipulator. In the formulation of this model, gravity forces were not considered. However, gravity-effect can be implicitly included if we consider, when computing $\ddot{\mathbf{c}}_i$ recursively, that the base of the manipulator has an acceleration of $-\mathbf{g}$ where $\mathbf{g}$ is the gravity acceleration vector, a technique introduced by Luh et al [34]. If friction is considered, the power dissipated by friction forces/torques must be included in eq. (4.54), which leads to the following dynamics model:

$$\boldsymbol{\tau}^a = -\mathbf{T}^T(\mathbf{w}^* + \mathbf{w}^f) \tag{4.57}$$

or

$$\boldsymbol{\tau}^a = -\mathbf{T}^T\mathbf{w}^* - \mathbf{R}^T\boldsymbol{\tau}^f \tag{4.58}$$

where $\mathbf{w}^f$ is a 21-dimensional vector composed of all friction wrenches exerting at each link's center of mass, while $\boldsymbol{\tau}^f$ is a 9-dimensional vector composed of all friction torques exerted on each joint. Moreover, $\mathbf{R}$ is the 9×3 *joint-velocity Jacobian matrix*, defined as:

$$\mathbf{R} \equiv \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{q}}_a} = \begin{bmatrix} \mathbf{1} \\ -\mathbf{J}_u^{-1}\mathbf{J}_a \end{bmatrix} \tag{4.59}$$

where $\mathbf{1}$ is the 3×3 identity matrix. Matrix $\mathbf{R}$ can also be calculated in the same way as matrix $\mathbf{T}$, i.e., the jth column of $\mathbf{R}$ is calculated as $\dot{\mathbf{q}}$ assuming that all the actuated joints are locked but the jth one has a motion with velocity unity. Friction forces/torques, as they act on joints, are usually modeled as $\boldsymbol{\tau}^f$ rather than $\mathbf{w}^f$ and hence eq. (4.58) is more useful than eq. (4.57). However, the evaluation of eq. (4.57) is simpler.

If we want to include the generalized gravity wrench vector:

$$\mathbf{w}^g = \begin{bmatrix} \mathbf{w}_1^g \\ \vdots \\ \mathbf{w}_7^g \end{bmatrix}, \tag{4.60}$$

we add the gravity term to eq. (4.58) to become:

$$\boldsymbol{\tau}^a = -\mathbf{T}^T(\mathbf{w}^* + \mathbf{w}^g + \mathbf{w}^f) = -\mathbf{T}^T(\mathbf{w}^* + \mathbf{w}^g) - \mathbf{R}^T\boldsymbol{\tau}^f \tag{4.61}$$

Chapter 5

System integration

This chapter contains a description of the physical work done by the author of this thesis which deals with the transputer's hardware selection, and integration and the software design, implementation and integration to build the prototype controller for the 3-DOF planar manipulator which was discussed earlier.

This study is part of a larger project which is currently pursued in McRCIM (McGill Research Center for Intelligent Machines). The project's main objective is to achieve real time control of a 3-DOF Planar Manipulator (chapter 4). The algorithm to control this type of mechanism is different from serial manipulators, thus the utilization of parallel processing is essential.

A complete diagram of the project is shown in figure 5.1. The project is divided into four sub-projects:

1. Development of a parallel architecture control system to control the manipulators movement through a pre-set trajectory (presented in this chapter).
2. Design and manufacture of a 3-DOF parallel manipulator with revolute joints.
3. Selection, design, and implementation of a closed-loop control circuit for the actuators installed on the manipulator.

4. Design and implementation of an interface to achieve total bidirectional communication between the manipulator and the control transputer board.

The following section contains a description of the actual parallel architecture system (hardware and software), and how it was integrated to control the 3-DOF planar manipulator to complete the first sub-project. The remaining sub-projects are presently under research by Kounias [30] and Felton [14].

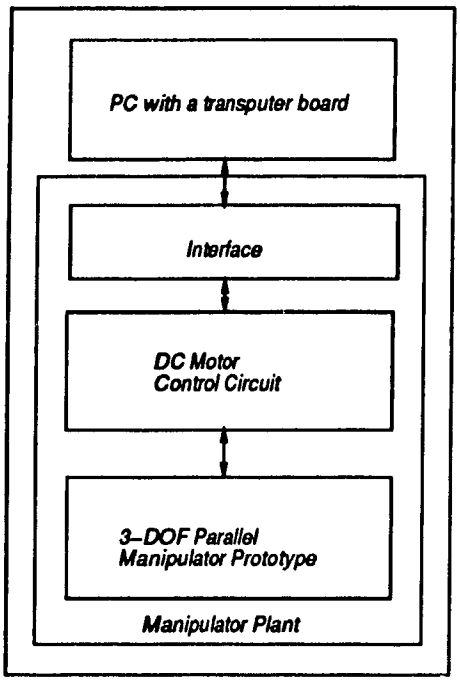


Figure 5.1: Block diagram of the complete project

5.1 Hardware integration

The T425 transputer (chapter 2) needs to be mounted on a suitable printed circuit board (PCB) equipped with power supply input and properly grounded at low impedance with sufficient decoupling, and the memory interface properly designed. This is achieved by the use of transputer module or TRAM. A TRAM consists of one or more transputers mounted on a PCB with memory and sometimes other interface circuitry (figure 5.2).

The TRAM, in turn, is mounted on a motherboard that supplies power and is responsible for clock distribution. Many motherboards also have one or more C004 link switches for reconfiguration of the TRAM-mounted transputer links.

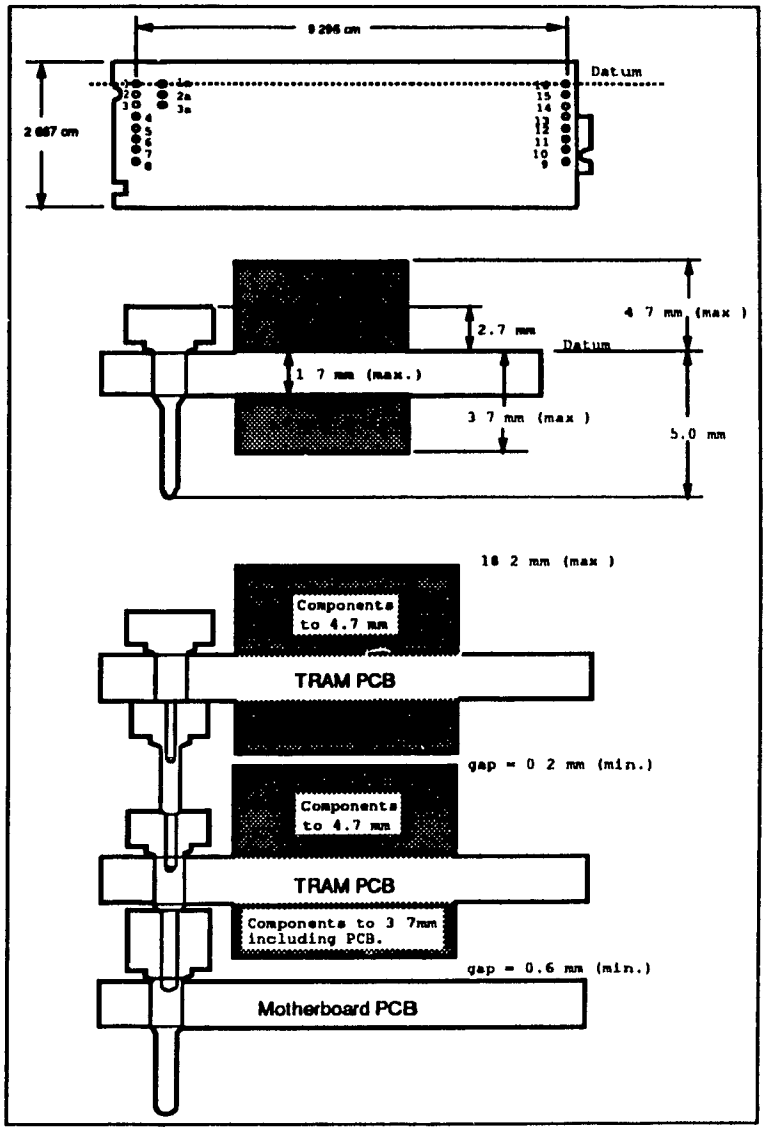


Figure 5.2: TRAM geometry

5.1.1 Module architecture

The standard size of a **TRAM** is a 2.667×9.296 cm (centimeters). with sixteen connections divided in two groups of eight each at one side of the module. There are 16 main pins in the

TRAM; the power supply pin (*Vcc*), the ground pin (*GND*); the reset pin (*RESET*); the analyse pin (*ANALYSE*) and the no error signal pin (*notError*); eight pins for four *links* (two pins per link); and two link speed selection pins (*LinkSpeedA* and *LinkSpeedB*). To select the speed of the links, if both *LinkSpeedA* and *LinkSpeedB* are low the links will operate at 10 Mbps, when both are high the links operate at 20 Mbps. Other combinations are reserved by the manufacturer.

Link2Out	1	16	Link3In
Link2In	2	15	Link3Out
Vcc	3	14	GND
Link1Out	4	13	Link0In
Link1In	5	12	Link0Out
LinkSpeedA	6	11	notError
LinkSpeedB	7	10	Reset
ClockIn	8	9	Analyse

Table 5.1: TRAM pinouts

It is assumed that 5 Mbps will not be used. The *Error* pin of the transputer is inverted at the *notError* **TRAM** output and is driven by an open collector or an open drain circuit. Thus, the *notError* pin of several modules may be connected together through an OR gate. This means that the *ErrorIn* pin is not used on transputers mounted on modules, and should be short circuited with the ground pin. Table 5.1 give the **TRAM** pinouts.

Pin	Signal
1a	SubSystem not Error
2a	SubSystem Reset
3a	SubSystem Analyse

Table 5.2: TRAM subsystem pinouts

The three subsystem control signal pinouts, which are, *SubSystem Reset*, *SubSystem Analyse* and *SubSystem notError*, are available on the **TRAM** to control another group of

transputers or **TRAMS**. Table 5.2 shows these pin signal assignments to the inside positioned pins 1, 2 and 3 in a standard module.

5.1.2 IMS B404 TRAM

A B404 **TRAM** has been utilized in our application. Its memory sizes and cycles are included in the Table 5.3 below. This is a size 1 **TRAM**, figure 5.2, that can contain a transputer and eight memory devices which corresponds to 1 Mbyte of dynamic memory, or 256 Kbytes of static RAM.

Part no.	Transputer	Memory size/cycles	SubSystem	Size
B401	T414-20, T425-25 or T800-25	32K/3 SRAM	no	1
B402	T222-20	8K/2 SRAM	no	1
B403	T414-20, T425-20 or T800-20	1M/3 DRAM	yes	4
B404	T414, T425 or T800	32K/3 SRAM, 2M/4 DRAM	yes	2
B405	T800-20	8M/5 DRAM	yes	8
B410	T801	160K/2 SRAM	no	2
B411	T425-20 or T800-20	1M/3 DRAM	no	1
B416	T222	64K/2 SRAM	no	1
B417	T800	64K/3 SRAM, 4M/4 DRAM	yes	4

Table 5.3: INMOS TRAMs

B008 Motherboard

The IMS B008 is the TRAM motherboard which has been integrated in an IBM PC-AT for this particular application. It has the capacity of ten TRAMs on board. Links 1 and 2 from each TRAM slots are hard wired on the IMS B008, to form a pipeline of processing elements among plugged in TRAMs. The remaining links can be "soft wired" using an INMOS IMS C004 programmable link switch, incorporated on the IMS B008. Figure 5.3 shows the complete block diagram of the B008.

The IMS C004 device is controlled by an IMS T212 16-bit transputer. Configuration data for the IMS C004 is fed into link 1 of the IMS T212 which in-turn passes it to the

IMS C004 on link 3. The same data is also fed through link 2 of the T212 to the 37-pin D-connector (DIN 37) on the edge of the board.

An interface with an IBM PC is available such that a program running on the PC can control the TRAMs mounted on the IMS B008 and passes data to or from them. Data communication can take place by either means of a software which uses polling, or via a Direct Memory Access (DMA) mechanism which gives a higher data flow rate. Different events on the IMS B008 can generate an interrupt signal which can be transmitted to the PC. This eliminates the need for the processor in the PC to continuously poll status registers on the IMS B008. Thus the PC can carry on with other tasks while programs are running on the IMS B008.

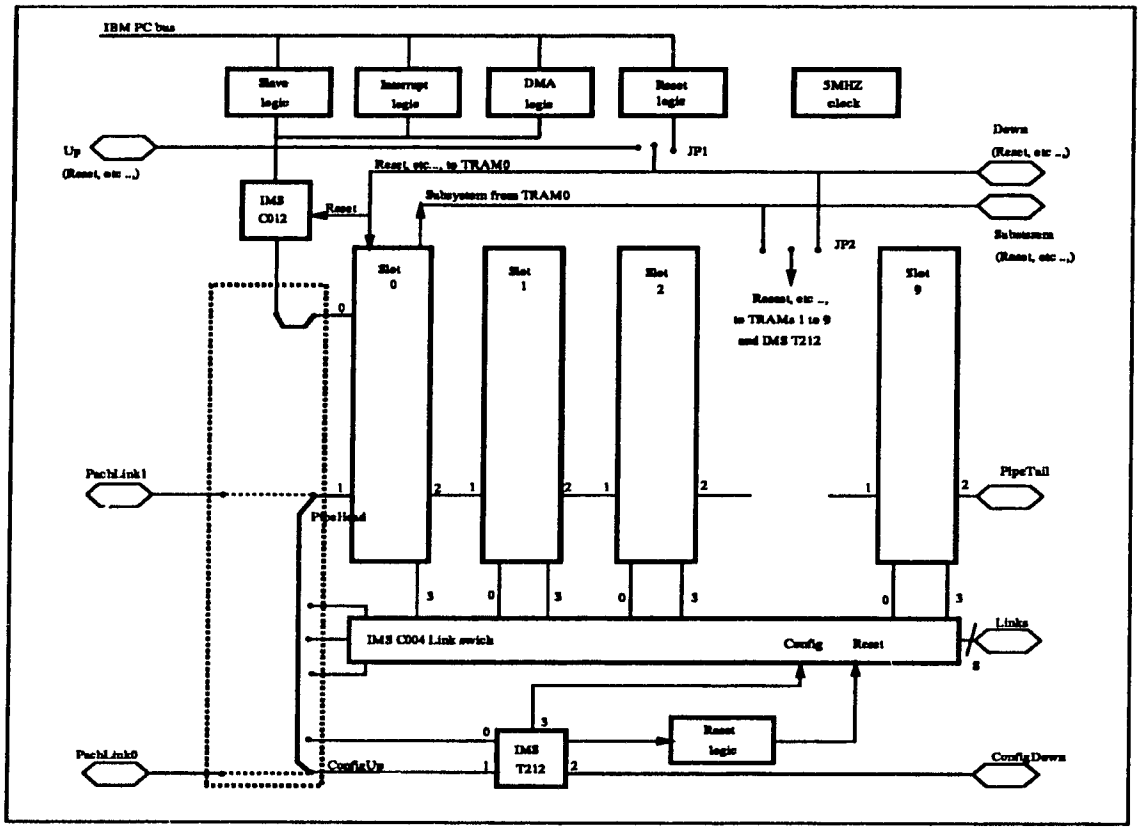


Figure 5.3: B008 block diagram

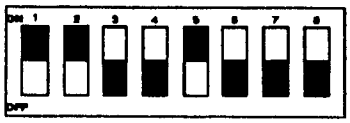


Figure 5.4: B008 switch settings

5.1.3 Integrating the B008 into the host personal computer

The switches and jumpers of the IMS B008 motherboard provide the user with a variety of operating modes. In the following segment, the significance of those settings will be explained. Moreover, the configuration used will be discussed (see figure 5.4 for switch settings utilized).

DMA channel (switches 1, 2)

Direct Memory Access (DMA) channel selection is done according to Table 5.4

SW1	SW2	DMA Channel
ON	ON	0
OFF	ON	1
ON	OFF	DMA disabled
OFF	OFF	3

Table 5.4: The available DMA channel settings

For this particular application DMA channel 3 was selected.

Interrupt channel (switch 3)

The interrupt channel can be selected according to Table 5.5

SW3	Interrupt Channel
ON	3
OFF	5

Table 5.5: The available Interrupt channel settings

during our implementation, the Interrupt channel is set for channel 5.

Board address (switches 4, 5)

Switches 4 and 5 are used to select the base location in the I/O address space at which the IMS B008 appear visible to the PC, or to disable the board from the PC bus all together.

SW4	SW5	Address (hexadecimal)
ON	ON	Not selected
OFF	ON	$150
ON	OFF	$200
OFF	OFF	$300

Table 5.6: The available B008 address settings

Table 5.6 present the address options available. Note that a $ sign indicates a hexadecimal number. Address $150 is selected for our B008.

Link speed selection (switches 6, 7 and 8)

All the IMS C004 and TRAMs must have identical link speeds. The IMS T212, however, can have its link 0 running at different speeds. Table 5.7 shows the possible combinations of link speeds.

SW6	SW7	SW8	T212 Link 0	All Other Links
ON	ON	ON	10 Mbits/s	10 Mbits/s
ON	ON	OFF	5 Mbits/s	10 Mbits/s
ON	OFF	ON	10 Mbits/s	10 Mbits/s
ON	OFF	OFF	20 Mbits/s	10 Mbits/s
OFF	ON	ON	NON-FUNCTIONAL	NON-FUNCTIONAL
OFF	ON	OFF	NON-FUNCTIONAL	NON FUNCTIONAL
OFF	OFF	ON	10 Mbits/s	20 Mbits/s
OFF	OFF	OFF	20 Mbits/s	20 Mbits/s

Table 5.7: The available B008 link speed settings

For this application, link speeds are set to be 20 Mbits/s for all links including the IMS T212 link 0.

Link configuration

One T425 transputer mounted on a **B404** TRAM was utilized. The B404 is connected to the B008 motherboard in slot 0. This connection configuration is dictated by the B008 design specifications [26]. Using slot 0 allows only two (links 3 and 2) of the T425 transputer four links to communicate with the outside world, the other two links are used to communicate with the host PC. The B404 communication links have been configured such that the transputer links are connected to the DIN 37 (peripheral input and output port of the B008 motherboard) as follows:

1) Transputer link 3 is connected to pins 3 and 22 of the DIN 37 for output and input signals respectively.
2) Transputer link 2 is connected to pins 16 and 35 of the DIN 37 for input and output signals respectively.

The B008 enables software link configuration through its on-board IMS C004 link switch. The following program fragment was used to achieve the link configuration specified in above (configuration item number 1):

```
SOFTWIRE
  SLOT 0, LINK3 TO EDGE 0
  .
  .
  .
END
```

The second configuration for link2 has been achieved via hard wire pipe jumpers (special 8-pin plugs) to the pipetail, see figure 5.3.

5.2 Occam software integration strategy

This section explains the software program designed and written by the author of this thesis.

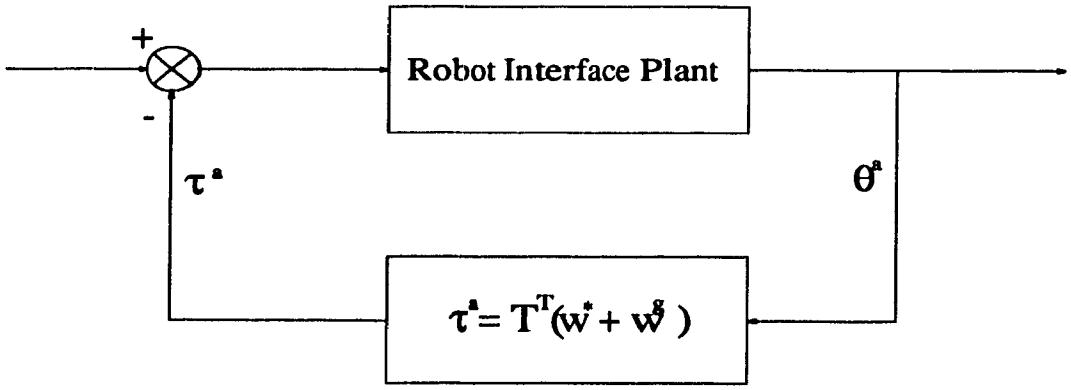


Figure 5.5: Manipulator control scheme

The manoeuvre of the manipulator is described in terms of the coordinates of its three actuated joints using the following equation:

$$q_i(t) = q_i(0) + \Big(q_i(T) - q_i(0)\Big)\left[\frac{t}{T} - \sin\left(2\pi\frac{t}{T}\right)\right], \qquad \text{for } i = 1,\ 2,\ 3. \tag{5.1}$$

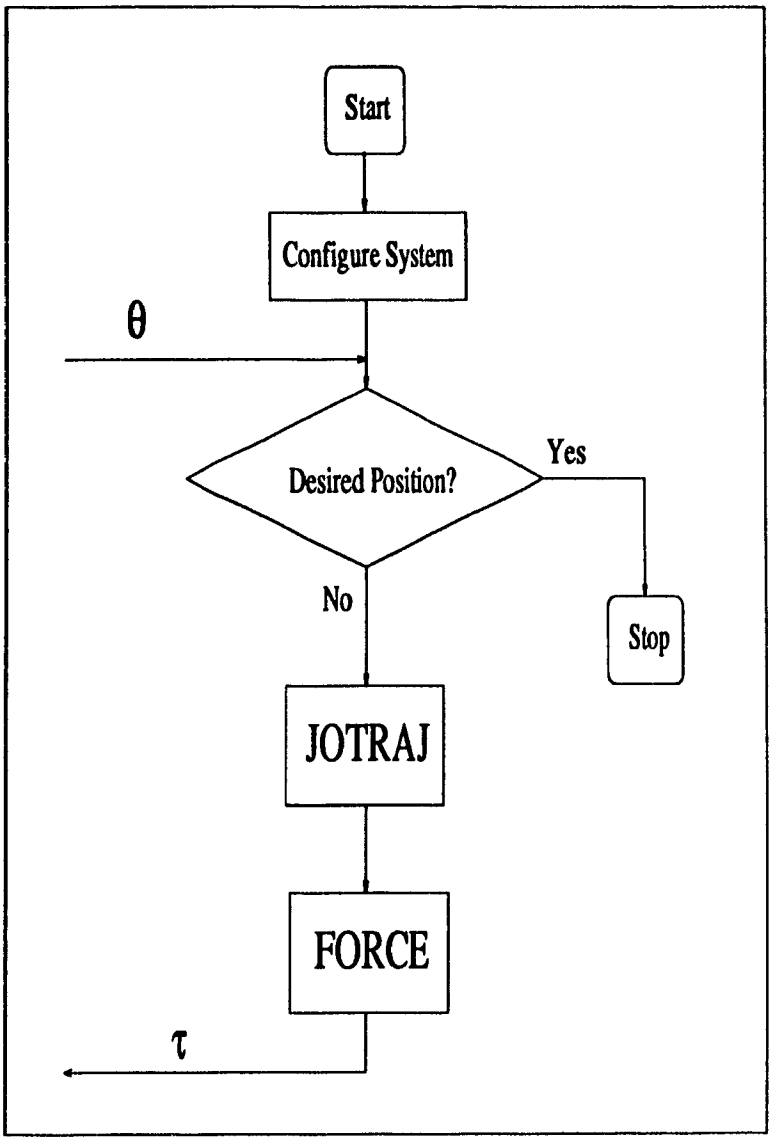


Figure 5.6: Occam program flow chart

where T is the time period of the whole manoeuvre. Using the direct kinematics described in chapter 4, it was possible to obtain the positions, velocities and accelerations of all links for the whole manoeuvre. Substituting the results of the initial condition (it is assumed that the manipulator starts its manoeuvre from rest) into the inverse-dynamics equation:

$$\boldsymbol{\tau}^a = -\mathbf{T}^T(\mathbf{w}^* + \mathbf{w}^g) \tag{5.2}$$

which is used to compute the driving torques needed by the three actuators in order to drive the manipulator through the given motion to move it from rest through the precalculated trajectory. The angles of the manipulator's three actuators are then used to calculate the velocities and accelerations of all links for every trajectory step which are then used for the remainder of the manoeuvre to generate the torques required. The resultant torques are piped from the transputer mounted on the B008 slot 0 via link 3 through the DIN 37 to the outside world.

Based on this configuration, the control algorithm was designed.

Figure 5.5 represents the closed loop control scheme utilized. The prototype supplies adequate torque values to drive the manipulator's three actuators. These values are first changed to voltage by the motor controllers [30] before they are fed to the manipulator. Furthermore, the occam program will keep updating those torque values, based on the motor angle values generated by the motor controllers, throughout the manipulator's manoeuvre.

Figure 5.6 illustrates the structure and the relationship of the two main modules, "JOTRAJ" and "FORCE", of the **occam** program. Starting from the manipulator's initial condition, "JOTRAJ" calculates the position, velocity, and acceleration in joint space, knowing in advance the final position and the required time to finish the manoeuvre. Then these values are fed to the "FORCE" module which in-turn calculates the required torque values to start moving the manipulator from the starting condition; thereafter, these values are sent to the designated output link, thus completing the first program loop. Subsequent program loops will follow the first program loop. At the beginning of each subsequent loop, "JOTRAJ" reads the actuators angle values from the manipulator and compares them with the final actuators angle values. If the actuators angle values at the beginning of each subsequent loop match the actuators angle values at the final position, the program stops; Otherwise, "JOTRAJ" will continue to calculate the velocities and accelerations and feeds them to "FORCE".

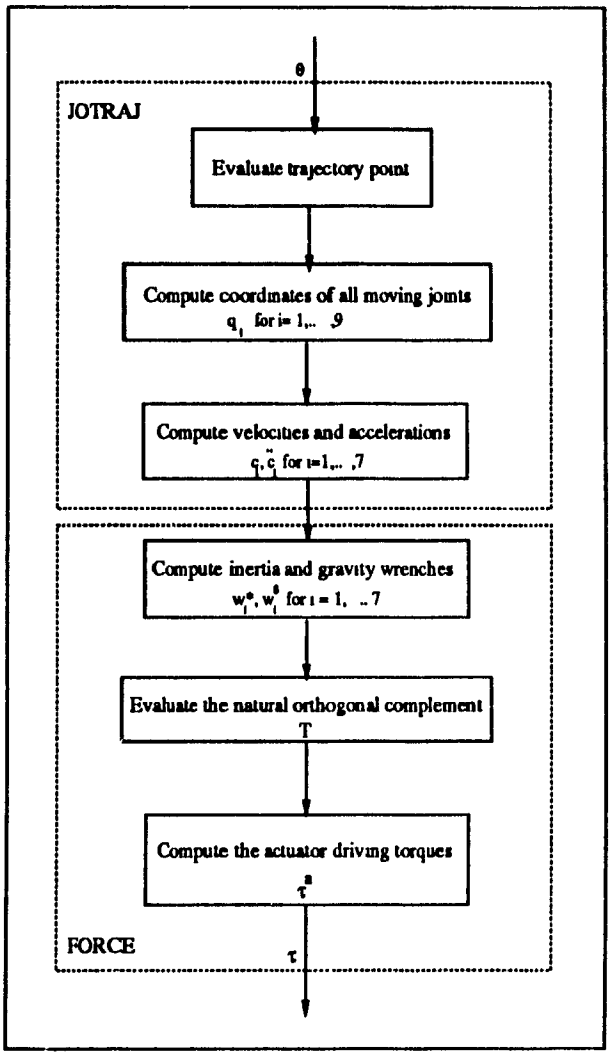


Figure 5.7: JOTRAJ and FORCE inner-structure

Figure 5.7 shows the inner-structure of "JOTRAJ" and "FORCE" modules. The formulae corresponding to each block of the diagram have been discussed in chapter 4, therefore, will not be presented here again.

5.2.1 Software examples

In this section, various comparative examples between actual parallel **occam** code fragments used in the implemented controller program and conventional FORTRAN code fragments to produce the same values will be presented to explain how the **occam** parallel code is

different from the sequential FORTRAN code. Parallel code has the advantage of utilizing the transputer hardware and its **occam** language support for concurrency, thus executing the independent operations involved simultaneously which results in their faster execution. Other actual sequential **occam** code fragments used in the program are presented to show the explicit nature of the **occam** language SEQ construct. In **occam** the sequential nature of a process must be stated explicitly by the SEQ construct. **Occam** PAR and SEQ constructs are discussed in chapter 3.

Parallel **occam** code is used throughout the program to execute in parallel the independent operations as they arise. However, in certain cases where the operations are dependent, **occam** sequential code is used. The full listing of the **occam** program can be found in Appendix B.

Example 1, on occam parallel code versus FORTRAN code:

```
---declarations
VALOF
  SRT:= DSQRT(3.0(REAL64))
  PAR
    SEQ
      A0:= ((L2*CP)+((CTH-(SRT3*STH))*(L3/2(REAL64)))-XQ)
      A1:= A0*A0
    SEQ
      A9:= ((L2*SP)+((STH+(SRT3*CTH))*(L3/2(REAL64)))-YQ)
      A2:= A9*A9
  DPQ := A1+A2-(L2*L2)
RESULT DPQ
```

In FORTRAN:

```
      SRT3=DSQRT(3.D0)
      A1=(L2*CP+(CTH-SRT3*STH)*L3/2-XQ)**2
      A2=(L2*SP+(STH+SRT3*CTH)*L3/2-YQ)**2
      DPQ=A1+A2-L2*L2
      RETURN
```

The above **occam** example function contains a **PAR** construct which calculates the two indented sequences in parallel, thus computes A1 and A2 simultaneously and returns the value of the function DPQ. In FORTRAN, the same equations are executed sequentially. The parallel approach significantly reduces computation time.

Example 2, on occam parallel code versus FORTRAN code:

```
PAR I = 1 FOR 3
  SEQ
    STH[I] := DSIN(THETA[I])
    P[1,I] : L1 *(STH[I])
  SEQ
    CTH[I] := DCOS(THETA[I])
    P[2,I] := L1 * (CTH[I])
```

In FORTRAN:

```
      DO 10  I=1, 3
         STH(I)=DSIN(THETA(I))
         CTH(I)=DCOS(THETA(I))
         P(1,I)=L1*CTH(I)
10       P(2,I)=L1*STH(I)
```

In this example, the FORTRAN DO construct calculates P(1,I) and P(2,I) three execution times to finish the task. However, in **occam** the PAR construct allows the parallel calculation of the two variables in only one execution. This considerably reduces the calculation time.

Example 3, on occam parallel code versus FORTRAN code:

```
PAR
  SEQ I = 0 FOR 1
    A[I,0] := AA[I,3]
    A[I,1] := -AA[I,4]
    A[I,2] := 0.0(REAL64)
    I2 := I + 2(INT)
    A[I2,0] := AA[I,3]
    A[I2,1] := 0.0(REAL64)
    A[I2,2] := -AA[I,5]
    A[I2,3] := RO[I,6] - RO[I,8]
  SEQ I = 0 FOR 1
    B[I,0] := -AA[I,0]
    B[I,1] := AA[I,1]
    B[I,2] := 0.0(REAL64)
    I2 := I + 2(INT)
    B[I2,0) := -AA[I,1]
    B[I2,1] := 0.0(REAL64)
    B[I2,2] := AA[I,2]
```

In FORTRAN:

```
      DO 10  I=1, 2
         A(I,1)=AA(I,4)
```

```
          A(I,2)=-AA(I,5)
          A(I,3)=0
          A(I,4)=RO(I,7)-RO(I,8)
          I2=I+2
          A(I2,1)=AA(I,4)
          A(I2,2)=0
          A(I2,3)=-AA(I,6)
10        A(I2,4)=RO(I,7)-RO(I,9)

       DO 20  I=1, 2
          B(I,1)=-AA(I,1)
          B(I,2)=AA(I,2)
          B(I,3)=0
          I2=I+2
          B(I2,1)=-AA(I,1)
          B(I2,2)=0
20        B(I2,3)=AA(I,3)
```

Both code fragments build two matrices. However, in **occam** the two matrices are constructed in parallel, thus reducing calculation time.

Example 1, on occam sequential code:

```
SEQ
  IF
    (DABS(U[(K-1(INT)),(K-1(INT))]) < (1.0E-40(REAL64)))
      SKIP
```

```
NOT (DABS(U[(K-1(INT)),(K-1(INT))]) < (1.0E-40(REAL64)))
  X[(K-1(INT))] := (B[(K-1(INT))]) / (U[(K-1(INT)),(K-1(INT))])
  SEQ J = 1(INT) FOR (N-1(INT)) -- First loop
      K := N - J
      SUM := 0.0
      SEQ I = (K+1(INT))  FOR N       --Second loop
        SUM := SUM + (U[(K-1(INT)),(I-1(INT))] * X[(I-1(INT))])
      IF
        (DABS(U[(K-1(INT),(K-1(INT)]) < (1.0E-40(REAL64)))
          SKIP
      X[(K-1(INT))]:=(B[(K-1(INT))]-SUM)/U[(K-1(INT)),(K-1(INT))]
```

In FORTRAN

```
      IF (DABS(U(K,K)).LT.1.D-40)  GOTO 30
      X(K)=B(K)/U(K,K)
      DO 20  J=1, N-1
         K=N-J
         SUM=0
         DO 10  I=K+1, N
10          SUM=SUM+U(K,I)*X(I)
      IF (DABS(U(K,K)).LT.1.D-40)  GOTO 30
20    X(K)=(B(K)-SUM)/U(K,K)
      RETURN
30    next program statement
```

In the above **occam** example indented if loops were utilized along with a **SEQ** construct to be able to solve the upper triangle system of a matrix. A **SEQ** construct had to be explicitly stated to execute the dependent mathematical operations sequentially. The **occam**

sequential execution has no advantage over FORTRAN execution, and both have identical performance and results.

Example 2, on occam sequential code:

```
SEQ I = 0(INT) FOR 2(INT)
  CDOT[0,I] := -RO[1,I] * W[I]
  CDOT[1,I] := RO[0,I] * W[I]
  I3 := I + 3(INT)
  CDOT[0,I3] := CDOT[0,I]-((AA[1,I] - RO[1,I])*W[I])-(RO[1,I3]*W[I3])
  CDOT[1,I3] := CDOT[1,I]+((AA[0,I] - RO[0,I])*W[I])+(RO[0,I3]*W[I3])
CDOT[0,6] := CDOT[0,3] -((AA[1,3]-RO[1,3])*W[3])-(RO[1,6]*W[6])
CDOT[1,6] := CDOT[1,3] +((AA[0,3]-RO[0,3])*W[3])+(RO[0,6]*W[6])
SEQ I =0(INT) FOR 6(INT)
  II := I * 3
  T[II,K] := W[I]
  T[(II+1(INT),K] := CDOT[0,I]
  T[(II+2(INT),K] := CDOT[1,I]
W[K] := 0.0(REAL64)
```

IN FORTRAN

```
      DO 10  I=1, 3
         CDOT(1,I)=-RO(2,I)*W(I)
         CDOT(2,I)=RO(1,I)*W(I)
         I3=I+3
         CDOT(1,I3)=CDOT(1,I)-(AA(2,I)-RO(2,I))*W(I)-RO(2,I3)*W(I3)
```

```
            CDOT(2,I3)=CDOT(2,I)+(AA(1,I)-RO(1,I))*W(I)+RO(1,I3)*W(I3)
10        CONTINUE
          CDOT(1,7)=CDOT(1,4)-(AA(2,4)-RO(2,4))*W(4)-RO(2,7)*W(7)
          CDOT(2,7)=CDOT(2,4)+(AA(1,4)-RO(1,4))*W(4)+RO(1,7)*W(7)
          DO 20  I=1, 7
             II=(I-1)*3
             T(II+1,K)=W(I)
             T(II+2,K)=CDOT(1,I)
20           T(II+3,K)=CDOT(2,I)
          W(K)=0
```

In this example a **SEQ** construct was used to compute the velocities of the mass centers of the manipulators three links in base coordinates. The **occam** and FORTRAN codes will produce the same results. However, in **occam** a **SEQ** construct had to be used to perform the dependent mathematical operations sequentially as per **occam** language rules.

5.2.2 Running the control program

The transputer control program, developed by the author, is called prog1.occ and resides now in an IBM-AT compatible PC at the Measurement Laboratory of McGill University.

To run prog1.occ the following steps should be followed:

Step 1:

Edit the file setup.inc using any file editing program and enter the setup values as defined. The file setup.inc should be placed in the same directory as the main program prog1.occ. A typical setup.inc file is presented below:

```
--This file should contain setup and required performance data
--fill in the data adhering to the specified format
```

```
--Time period to define trajectory in seconds

TT := 003.000

--Enter time to start in seconds in max three digit

TS := 000.000

--Enter the time step size in seconds

TSTEP := 000.010

--Enter the end time in seconds

TE := 003.000

--Start angles in degrees in three digit format.
--For actuator #1

TH0[0] := 060.000

--For actuator #2

TH0[1] := 120.000

--For actuator #3

TH0[2] := 090.000
```

```
--End angles in degrees in three digits.

--For actuator #1

THT[0] := 120.000

--For actuator #2

THT[1] := 000.000

--For actuator #3

THT[2] := 120.000
```

Step 2:

Compile the program as per the following steps:

Type:

```
occam prog1.occ t5
```

The result of this compilation (a file called "prog1.t5h") must be linked with the libraries it uses (hostio.lib and dblmath.lib).

Step 3:

To link the program type:

```
ilink prog1.t5h hostio.lib dblmath.lib
```

The linked program will be written to the file prog1.c5h.

Step 4:

Before the program can run it must be made 'bootable'. This is done by using the bootstrap tool iboot. Type:

```
iboot prog1.c5h
```

This will generate the file prog1.b5h.

Step 5:

To run the program it must be loaded onto the transputer board using the host file server tool iserver. To load and run the program type:

```
iserver /sb prog1.b5h
```

The program now is loaded into the transputer board and running. For detailed description on the operation of the various commands required to compile, link and run an **occam** program, refer to [27].

5.2.3 Technical considerations of the control program

The control program is currently set for the following specific operational and input/output specifications. These specifications are set to utilize the available transputer hardware facilities and are compatible to the other phases of the prototype manipulator project of Kounias [30] and Felton [14].

Timing

The control algorithm performance execution requirements are currently set to complete a 3 second manipulator manoeuvre with a slow program cycle rate of 0.01 seconds per cycle. Once the manipulator's dc motors used in the implementation of the motor controller system parameters [30] are known the control systems set cycle rate can be changed to achieve an optimum performance level. The performance execution requirements can be changed by inputing the new values in the input file of the program and by changing the value of the time step (TSTEP). A typical input file example has been presented earlier.

Actuator angles input to the program

The **occam** control program expects to receive the three actuator angles sequentially on link 3 through pin 22 in the DIN 37 at the B008 motherboard. As the T425 transputer, as discussed earlier, only has two free links, it is not possible to divide the three input angle signals equally between the two free links. If, however, one decides to send two signals on one link and the third on the other link, the design of the PC-dc motor controller interface should accommodate for this configuration by implementing a buffering and synchronization technique. Those approaches are inefficient as they will slow down the communication between the dc motor controllers and the transputer board due to the additional processing involved.

The current input angles configuration is set in the control program by the following code segment:

```
-- Reading actuator angles originating from the manipulator
-- Step 1- initializing the input channel
PROTOCOL Theta.in IS REAL32; REAL32; REAL32:
CHAN OF Theta.in RealtimeTheta:
PLACE RealtimeTheta AT 7:
-- Step 2- reading thetas sequentially
RealtimeTheta ? TH[0]; TH[1]; TH[2]
```

The above actual control program segment reads the actuator angle values according to **occam 2** language definition [43] and to the transputer hardware link communication protocol (discussed in chapter 2).

The control program expects to receive the three actuator angle values in REAL32 type. According to **occam** 2 language definition, a value of type REAL32 is represented by a sign bit, an 8 bit exponent and a 23 bit fraction. The value is positive if the sign bit = 0 and negative if the sign bit = 1. A signed real value of type REAL32 is represented according to ANSI/IEEE standard 754-1985. However, in this program only positive angle

values are expected as per the manipulators design characteristics [30].

From the above discussion, the transputer control system expects to receive 3×32 bits representing actuator angles (in the order of: motor 1, motor 2 and motor3 *always*) via a hardwire connection between pin 22 in the B008 motherboard DIN 37 and the motor control interface.

However, the input angles configuration can be changed to parallel input through their perspective input statements in the program at a later stage, if the hardware used has three available input links. For example, if the transputer used has links 1, 2 and 3 free, one can write:

```
--Reading thetas from the manipulator from three links parallel
-- step 1- initializing input channels
PROTOCOL Theta1.in IS REAL32:
PROTOCOL Theta2.in IS REAL32:
PROTOCOL Theta3.in IS REAL32:
CHAN OF Theta1.in RealtimeThetaA:
CHAN OF Theta2.in RealtimeThetaB:
CHAN OF Theta3.in RealtimeThetaC:
PLACE RealtimeThetaA AT 5:
PLACE RealtimeThetaB AT 6:
PLACE RealtimeThetaC AT 7:

-- step 2- Reading the input in parallel
PAR
  RealtimeThetaA ? TH[0]
  RealtimeThetaB ? TH[1]
  RealtimeThetaC ? TH[2]
```

The above example code reads the three input angles in parallel through links 1, 2 and 3. Protocol, channels and placement statements are presented in chapter 3.

Actuator torque values output

The T425 position in slot 0 in the B008 (discussed earlier) leaves only two links (2 and 3) free for data transfer. The control program is set to output the torque values sequentially on the T425 transputer link 3 through pin 3 of the B008 motherboard's DIN 37. This configuration decision has been implemented as it was not possible to divide the three torque values equally between the two available links. One of the alternative possible configurations is to send two torque values over one link and the remaining torque value over the second link. This approach is inefficient as it will lead to manipulator actuator synchronization problems. The other alternative solution is by using a buffering technique in the PC-dc motors interface. This approach will slow down data transfer between the transputer board and the dc motor controllers.

The current output torque configuration is set in the control program by the following program segment:

```
-- Step 1- initializing output channel
PROTOCOL Torque.out IS REAL32; REAL32; REAL32:
CHAN OF Torque.out ResultTorqueMotor:
PLACE ResutTorqueMot AT 3:
-- Step 2- sending the output torque values
ResultTorqueMotor ! TORQUE[0]; TORQUE[1]; TORQUE[2]
```

This actual control program segment sends the three actuator torque values according to **occam** 2 language definition [43] and to the transputer hardware link communication protocol (highlighted in chapter 2).

The control program sends the three torque values in REAL32 type. From **occam** 2 language definition (highlighted earlier), the torque values are sent in a 32 bit format with the least significant bit represent a sign bit, the remaining bits represent an 8 bit exponent and a 23 bit fraction. The torque value is positive if the sign bit = 0 and negative if the sign bit = 1. The sign bit is used in the control program to define the direction of the

torque. If the sign bit is positive, this indicates that this torque should rotate its designated motor clockwise. If the sign bit is negative, this indicates that this torque should rotate its designated motor counterclockwise.

The transputer control system sends sequentially three torque values organized as: motor 1, 2 and 3 *always*. These torque values are represented by 3 $\times$ 32 bits. The transputer control system sends these torque values through pin 3 in the DIN 37 on the B008 motherboard. To the manipulator motor control circuitry. A transputer-motor controllers interface proposal, for the torque output signals, was submitted based on the above configuration by Felton [14].

If the transputer hardware has three free links, the torque values can be sent out of the transputer control system in parallel as per the following program segment example:

The transputer in this example has links 1, 2 and 3 free. One can write:

```
-- Step 1- Output channel initialization
PROTOCOL Torque.outA IS REAL32:
PROTOCOL Torque.outB IS REAL32:
PROTOCOL Torque.outC IS REAL32:
CHAN OF Torque.outA ResultTorqueMotA:
CHAN OF Torque.outB ResultTorqueMotB:
CHAN OF Torque.outC ResultTorqueMotC:
PLACE ResutTorqueMotA AT 1:
PLACE ResutTorqueMotB AT 2:
PLACE ResutTorqueMotC AT 3:
-- Step 2- Sending the torque output out in parallel
PAR
  ResutTorqueMotA ! TORQUE[0]
  ResutTorqueMotB ! TORQUE[1]
  ResutTorqueMotC ! TORQUE[2]
```

The above example code sends the three output torques out of the transputer in parallel

over links 1, 2 and 3. Protocol, channels and placement statements are highlighted in chapter 3.

Chapter 6

Performance

In this chapter transputer performance is compared with an IBM-AT performance based on standard benchmark tests. These performance comparisons are published in [48]. A benchmark is a standard measure of performance that enables one computer to be compared with another. Three benchmark tests have established themselves as the industry standards: the Whetstone benchmark, the Savage benchmark and the Dhrystone benchmark. At the end of this chapter the implemented architecture performance test is demonstrated.

6.1 The Whetstone benchmark

The Whetstone benchmark program [11] is used to compare processor power for scientific applications. Running the program is considered equivalent to executing (approximately) one million "Whetstone" instructions. Performance, as measured by the benchmark, is quoted in "Whetstones per second". In addition to floating-point operations, it includes integer arithmetic, array indexing, procedure calls, conditional jumps, and elementary function evaluations.

Table 6.1 and Table 6.2 illustrate the performance of the transputer T425 used versus the performance of some INTEL processor chips.

System	Thousands of Single-Precision Whetstones per Second
IMS T425-30	1056
INTEL 286/287	300

Table 6.1: Single-Precision Whetstone benchmark results

System	Thousands of Double-Precision Whetstones per Second
IMS T425-30	242
INTEL 286/287	Not available
INTEL 8086-8087	152

Table 6.2: Double-Precision Whetstone benchmark results

6.2 The Savage benchmark

The Savage benchmark is a benchmark of elementary function evaluation only [45]. The Savage benchmark tests both execution speed and accuracy.

System	CPU	FPP	MHz	Language	Time (seconds)	Error (absolute)
IMS T425			30	occam	4.2	1.2E-9
IBM PC-AT	286	287	6.0	turbo pascal	7.4	1.2E-9

Table 6.3: Comparative Savage benchmark results

From table 6.3, it is clear that the time required to complete the Savage test is almost half of that used by 286-287 PC.

6.3 The Dhrystone benchmark

The Dhrystone benchmark [44] is a synthetic benchmark designed to test processor performance on "Systems programs".

System	Dhrystones per Second
IMS T425-30	13400
INTEL 80286-9	1976

Table 6.4: Comparative Dhrystone benchmark results

From table 6.4, one can notice the performance increase when employing the IMS T425.

6.4 The implemented parallel architecture performance experiment

6.4.1 Introduction

The main objective of this experiment is to demonstrate the ability of the control program to produce torque values which correspond to the manipulators three actuators positions to move the manipulator through a given trajectory. The author of this thesis modified the original control program to achieve this objective.

The performance of the control system is governed by two aspects:

1) A fast processor (IMS T425-30 is presented in chapter 2).
2) The use of a parallel programming language compatible with the aforementioned processor (**occam** language highlighted in chapter 3).

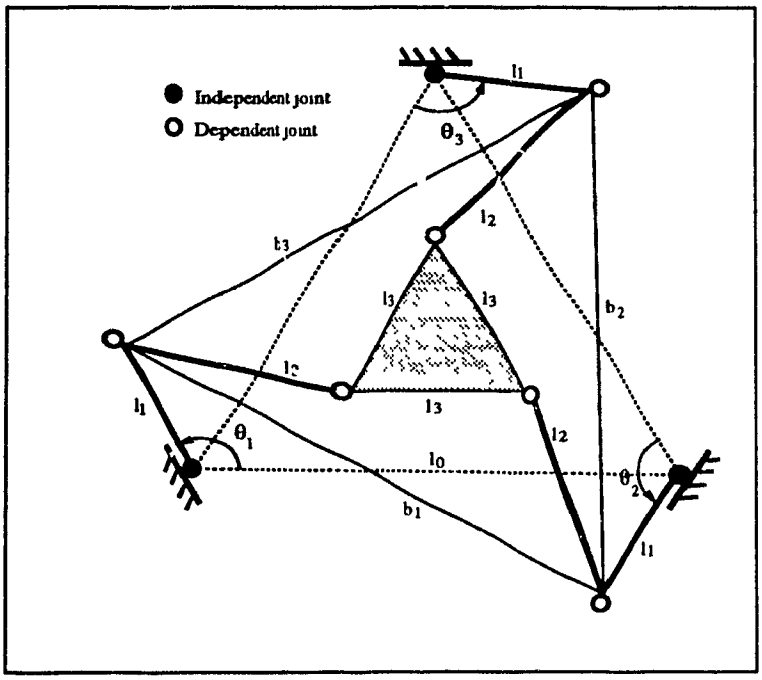


Figure 6.1: The 3-DOF Parallel Manipulator

6.4.2 The simulation test program

To test the program behaviour, the geometric dimensions of the 3-DOF planar manipulator illustrated in figure 6.1 are:

$$l_0 = 0.8\,\text{m}, \qquad l_1 = 0.44\,\text{m}, \qquad l_2 = 0.22\,\text{m}, \qquad l_3 = 0.125\,\text{m}. \tag{6.1}$$

with mass and inertia properties of:

$$m_1 = m_2 = m_3 = 2.765\,\text{kg}, \qquad I_1 = I_2 = I_3 = 0.06\,(\text{kg}\cdot\text{m}^2), \tag{6.2}$$

$$m_4 = m_5 = m_6 = 1.328\,\text{kg}, \qquad I_4 = I_5 = I_6 = 0.0066\,(\text{kg}\cdot\text{m}^2), \tag{6.3}$$

$$m_7 = 5.06\,\text{kg}, \qquad I_7 = 0.0132\,(\text{kg}\cdot\text{m}^2), \tag{6.4}$$

where m_1, m_2 and m_3 are the masses of the three links of length l_1 and I_1, I_2 and I_3 are their moments of inertia. m_4,m_5 and m_6 are the masses of the three links of length l_2 and I_4, I_5 and I_6 are their moments of inertia. It was assumed also that a 3 kg circular object was placed at the center of the end effector, hence, m_7 and I_7 include its mass and its moment of

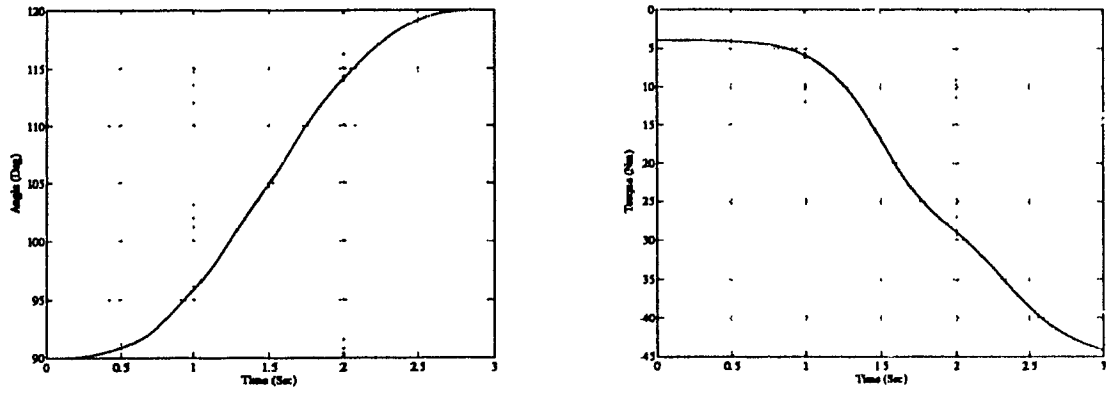
inertia. The mass center of each link is located at its center. The manipulator's manoeuvre is described by the cycloidal equation:

$$q_i(t) = q_i(0) + \Big(q_i(T) - q_i(0)\Big)\left[\frac{t}{T} - \sin\left(2\pi\frac{t}{T}\right)\right], \quad \text{for } i = 1,\ 2,\ 3 \tag{6.5}$$

T is the time for the manipulator to complete the whole manoeuvre. For this test the whole manoeuvre was set to take 3 seconds, with a cycle time of 0.01 seconds (same as the pre-set values of the original control program). Using the direct kinematics described in chapter 4, the positions, velocities and accelerations of all of the links were obtained for the whole manoeuvre. Substituting these results into the inverse dynamics equation:

$$\tau^a = -\mathbf{T}^T(\mathbf{w}^* + \mathbf{w}^g) \tag{6.6}$$

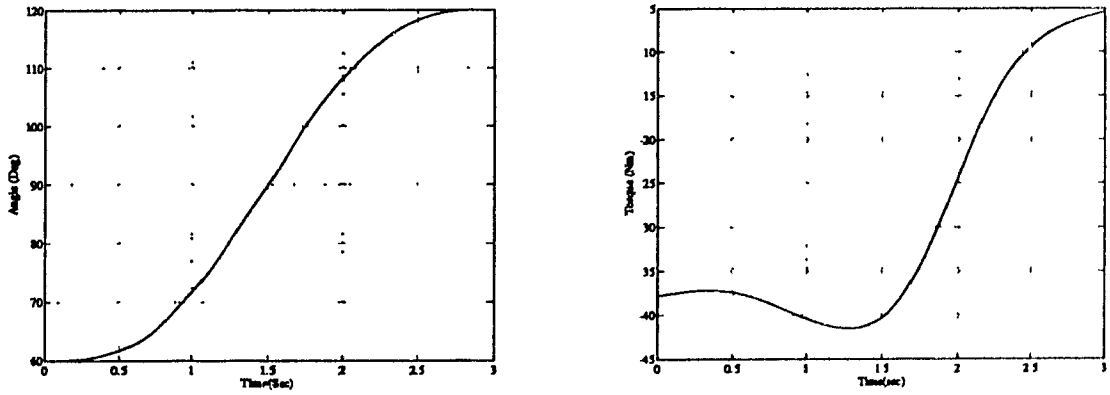
the torque values which the manipulators three actuators should provide throughout the whole manoeuvre were obtained. Direct substitution of trajectory calculation (positions, velocities and accelerations of all links) was used in the computation of the torques throughout the manoeuvre. This setup was used instead of direct measurement due to lack of manipulator real time data feedback (actuator angles) throughout the manoeuvre. The following figures illustrate the actuated joint angles throughout the manoeuvre and the actuators corresponding torques calculated to move the manipulator throughout its manoeuvre (the raw data are listed in Appendix D).



(a) (b)

Figure 6.2: First joint. (a) Angle θ_1 versus time (b)Torque versus time.

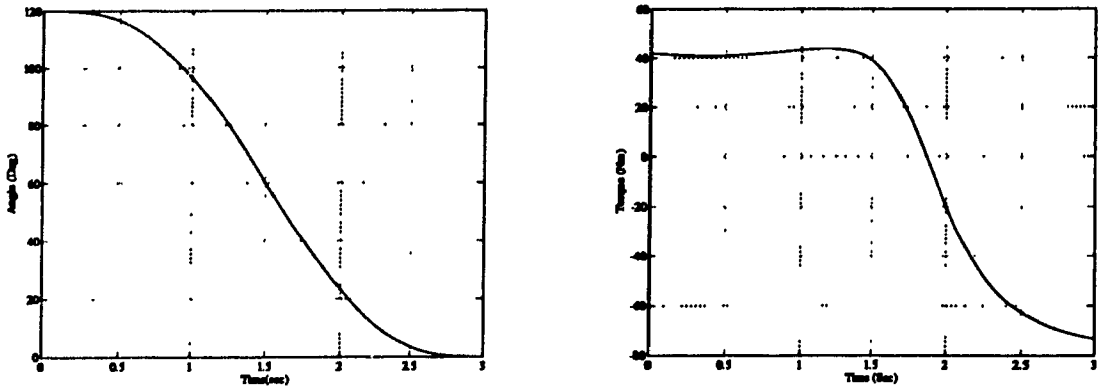
Figure 6.2 illustrates the first joint angle starting from 90° to have a final value of 120° and its required torque during the 3 seconds manoeuver.



(a) (b)

Figure 6.3: Second joint. (a) Angle θ_2 versus time (b)Torque versus time.

In figure 6.3 the second joint angle motion to reach 120° starting from 60° and the required computed torque values.



(a) (b)

Figure 6.4: Third joint. (a) Angle θ_3 versus time (b)Torque versus time.

Figure 6.4 represents the third joint angle and its required torque time histories.

The plots presented in figures 6.2, 6.3 and 6.4 plot the data represented in the above tables versus elapsed time, thus showing the timing results for each of the manipulator's actuated joints torque values. The values are plotted for each program cycle (0.01 seconds per cycle) throughout the 3 second manoeuvre. One can notice the non-identical torque behaviour for the three actuators as well as the smoothness of the torque curves as they develop through time.

Chapter 7

Conclusions

This thesis contains a study of the application of parallel processing in the mechanical analysis of 3-DOF parallel manipulators, to control the movement of that type of manipulators in real time. As an application of this study, a parallel control scheme was developed and implemented to control the motion of a prototype 3-DOF Planar Parallel Manipulator in real time.

The hardware used (transputer) was specially designed to support parallel processing. This processor was integrated in an IBM-AT and used as the hardware controller for the parallel manipulator.

Since the transputer operation requires a special high level programming language (**occam**), a computer program was developed using **occam** to compute the torques required for the parallel manipulator providing the positions of each actuated link are given. This program is considered the software part of the controller.

Using the facilities provided by the transputer (mainly parallel processing), the program can provide more than one output torque value at the same time for the 3-DOF planar manipulator to meet its three actuators requirements.

Both hardware and software were implemented in a modular design to allow for application versatility and future improvements. From the hardware perspective, the B008

transputer motherboard used can carry up to 10 processor TRAMs. In-addition each transputer TRAM allow stacking. On the other-hand, the integrated **occam** program can be used for any similar 3-DOF planar manipulator of different dimensions by changing the design specifications constants. Also, the program is modular and structurally modifiable to facilitate the use of more than one transputer.

Due to the lack of adequate number of transputers, only one transputer is used for this controller scheme implementation. Also, to simplify the interface design sub-project (under development [14]) and to achieve fast transputer-motor controllers communication, only one of the two transputer's free links is used. The utilization of only one link forced the originally calculated and produced torque values in parallel to be sequentially sent out through the 37-pin D connector at the B008 board edge.

7.1 Suggestions for future research

This project concludes the first phase of the main project. In order to obtain real time control, the controller system need to be interfaced with the existing prototype presented by Kounias [30]. An interface proposal was presented by Felton [14] as a first step towards a fully bidirectional interface implementation. That leaves the main project remaining completion steps to be:

1. Design and implement a bidirectional interface to accomplish the desired communication between the B008 transputer motherboard and the prototype 3-DOF Planar Manipulator.

2. Extend the parallel architecture to include three transputers. Thus allocating one transputer to generate the torques for each of the manipulators three actuators. This approach promises a full parallel control of the manipulator in real time.

Bibliography

[1] Angeles, J., *"On the numerical solution of the inverse kinematic problem"*. The Int. J. Robotics Research, Vol. 4(2), pp. 21-37, 1985.

[2] Angeles, J. and Lee, S. K., *" The formulation of dynamical equations of holonomic mechanical systems using a natural orthogonal complement"*, Trans. ASME, Journal of Applied Mechanics, Vol. 55(1), pp. 243-244, 1988.

[3] Angeles, J. and Ma, O., *" Dynamic simulation of n-axis serial robotic manipulators using a natural orthogonal complement"*, The International Journal of Robotics Research, Vol. 7(5), pp. 32–47, 1988.

[4] Barton & Elliot, *"A transputer review"*. Smith Associates, 1988.

[5] Behi, F., *"Kinematic analysis for a six degree-of freedom 3-PRS parallel mechanism"*, IEEE Journal of Robotics and Automation, Vol. 4(5), pp. 561-565, 1988.

[6] Brady, M. et al, *"Robot Motion: Planning and control"*, MIT Press, Cambridge, MA., 1983.

[7] Broomhead, D. S., et al., *"A practical comparison of the systolic and wavefront array processing architectures"*, Proc. IEEE Int. Conf. on Acoustics, Speech and Image Processing, Tampa, March 1985.

[8] Carling, A., *"Parallel Processing, The transputer and Occam"*, Sigma press, Wilmslow, 1988.

[9] Chesney, Miles, *"The transputer spawns a radically new computer"*, Electronics, McGraw Hill Inc. N.Y., N.Y., pp. 43-45, Oct. 7, 1985.

[10] Craig, J. J., *"Introduction to robotics, mechanics and control"*, Addison-Wesly, Redding, 1986.

[11] Curnow H.J., and Wichmann B.A., *"A synthetic benchmark"*, Computer Journal 19(1), pp. 43-49, Feb. 1976.

[12] Do, D. Q. W. and Yang, D. C. H., *"Inverse dynamic analysis of a platform type of robot"*, Journal of Robotic Systems, Vol. 5(3), pp. 210-227, 1988.

[13] Elliot C., Armstrong R., *"Transputer applications: Transputers on board spacecraft"*, Smith Associates Ltd., chapter 11, pp. 237-253, 1991.

[14] Felton, P., *"The design of a transputer interface"*, internal report, McGill University, 1992.

[15] Fichter, E. F., *"A Stewart platform-based manipulator: general theory and practical construction"*, The Int. J. Robotics Research, Vol. 5(2), pp. 157-182, 1986.

[16] Flynn, M. F., *"Some computer organizations and their effectiveness"*, IEEE Trans. Computer C-21, pp. 948–960, 1972.

[17] Garcia de Jalon, J., Unda, J., Avello, A. and Jimenez, J. M., *"Dynamic analysis of three-dimensional mechanisms in "natural" coordinates"*, Trans. ASME, Journal of Mechanisms, Transmission and Automation in Design, Vol. 109, pp. 460–465, 1987.

[18] Gosselin, C. and Angeles, J., *"A new performance index for the kinematic optimization of robotic manipulators"*, Proceedings of ASME 20^{th} Biennial Mechanisms Conference, Sept. 25-28, Kissimmee, Fl., pp. 441–447, 1988.

[19] Gosselin, C. and Angeles, J., *"Singularity analysis of closed-loop kinematic chains"*, IEEE Trans. Robotics and Automation, Vol. 6(3), pp. 281–290, 1990.

[20] Harp, G., *"Transputer Applications: Parallel processing"*, Pitman Publishing, St. Mutaine, pp. 1–10, 1991.

[21] Harp, G., Baker S., Webber H., *"Transputer applications: Image processing"*, Pitman Publishing, Chapter 9, pp. 204-220, 1991.

[22] Hartenberg, R. S. and Denavit, J., *"Kinematic Synthesis of Linkages"*, McGraw-Hill, New York, 1964.

[23] Hoare, C. A. R., *"Communicating sequential processes"*, Prentice Hall, Hemel Hempstead, 1985.

[24] Hunt, K. H., *"Structural kinematics of in-parallel-actuated robot-arms"*, Trans. ASME, Journal of Mechanisms, Transmission and Automation in Design, Vol. 105, pp. 705-712, 1983.

[25] Ian Graham and Tim King, *"The transputer handbook"*, Prentice Hall, Hemel Hempstead, Second edition, 1991.

[26] INMOS *"IMS B008 user guide and reference manual"*, Inmos LTD., technical document, pp. 2-6, 1988.

[27] INMOS *"Occam 2 toolset user manual"*, Inmos LTD., technical document, 1989.

[28] Jones, G. and Goldsmith W., *"Programming in occam 2"*, Prentice Hall, Hemel Hempstead, 1988.

[29] Kokkinis, T. and Stoughton, R., *"Dynamics and control of closed-loop spatial 5-bar linkages"*, ASME paper No. 89AMR-9B-1, pp. 1–7, 1989.

[30] Kounias, S., *Design, manufacture and control of a planar three degree of freedom parallel manipulator*, M. Eng. Thesis, McGill University, Aug. 1993.

[31] Lee, K. M. and Shah, D. K., *"Kinematic analysis of a three-degrees-of-freedom in-parallel actuated manipulator"*, IEEE Journal of Robotics and Automation, Vol. 4(3), pp. 354–360, 1988a.

[32] Lee, K. M. and Shah, D. K., *"Dynamic analysis of a three-degrees-of-freedom in-parallel actuated manipulator"*, IEEE Journal of Robotics and Automation, Vol. 4(3), pp. 361–367. 1988b.

[33] Lee, K. M. and Chao, A., *"On the analysis of a three-degree-of-freedom manipulator"*, International Journal of Robotics and Automation, Vol. 3(2), pp. 90-96, 1988.

[34] Luh, J. Y. S., Walker, M. W. and Paul, R. P. C, *" On-line computational scheme for mechanical manipulators"*, Trans. ASME, J. Dyn. Syst., Meas., and Control, Vol. 102, pp. 103-110, 1980.

[35] Ma, O. and Angeles J., *"Performance evaluation of path-generating planar, spherical and spatial four-bar linkages"*, ASME Journal of Mechanism and Machine Theory, Vol. 23(4), pp. 257-268, 1988.

[36] Ma, O., *"Mechanical Analysis of Parallel Manipulators with Simulation, Design and Control"*. Ph. D. Thesis, Mechanical Engineering Department, McGill University, Montreal, 1991.

[37] Ma, O. and Angeles, J., *"Direct kinematics and dynamics of a planar three-dof parallel manipulator"*. Trans. ASME, J. Mech., Trans. and Automation in Design, volume 3, pp 313-320, 1989.

[38] Manuel, T., *"As the world turns parallel, transputer applications explode"*, Electronics, VNU Business Publication Inc., Hasbrouck Heights, N.J., pp. 110–112, Dec., 1988.

[39] McCarthy, J. M., *"Kinematics of robot manipulators"*, a collection of papers published in The International J. Robotics Research, MIT Press, Cambridge. MA., 1987.

[40] Merlet, J-P., *"Force-feedback control of parallel manipulators"*, Proceedings of IEEE International Conference, Robotics and Automation, IEEE Computer Society Press, Vol. 3, pp. 1484-1489, 1988.

[41] Nolle, H., *"Linkage coupler curve synthesis: A historical review— III. Spatial synthesis and optimization"*, Mechanism and Machine Theory, Vol. 10, pp. 41–55, 1975.

[42] Paul, R. P., *"Robot manipulators: mathematics, programming and control"*, The MIT Press, Cambridge, MA., 1981.

[43] Pountain, D. and May, D., *"A tutorial introduction to occam programming"*, BSP Professional Books. An INMOS LTD. document, BSP Professional Books, Marco, 1988.

[44] Reinhold P. Weicker, *"Dhrystone: a synthetic systems programming benchmark"*, Communications of the ACM, Vol. 27(10), Oct. 1984.

[45] Savage B., *"The Savage benchmark"*, Dr. Dobb's Journal, pp. 120, Sep. 1983.

[46] SGS-THOMSON *"The T9000 transputer products overview manual"*, Inmos LTD., First edition, Consolidated Printers, Berkeley, pp. 55, 1991.

[47] SGS-THOMSON *"The transputer data book"*, Inmos LTD., second edition, Consolidated Printers, Berkeley, 1989.

[48] SGS-THOMSON *"Transputer technical notes"*, Inmos LTD., second edition, Prentice Hall International(U.K.) Ltd., Hemel Hempstead, pp. 205-227, 1989.

[49] Stewart, R. P., *"A platform with six degrees of freedom"*, Proc. of the Institution of Mechanical Engineers, Vol.180(1), pp. 371-386, 1965.

[50] Waldron, K. J., *"Elimination of branch problem in graphical Burmester mechanism synthesis for four finitely separated positions"*, ASME Trans., Journal of Engineering for Industry, Vol. 98, No. 1, pp. 176–182, 1976.

[51] Williams II, R.L. and Reinholtz, C.F., *" Forward dynamic analysis and power requirement comparison of parallel robotic mechanisms"*, Proceedings of ASME Design Technology Conference, 20^{th} Biennial Mechanical Conference, Sept. 25-28, Kissimmee, Florida, pp. 71-78, 1988.

[52] Yang, D. C. and Lee, T. W., *"Feasibility study of a platform type of robotic manipulators from a kinematic view-point"*, Trans. ASME, J. Mech., Trans., and Auto. in Design, 1984, Vol. 106, pp. 191-198.

Appendix A

Schemes of Numerical Methods

A.1

Newton-Raphson Method.

Given an initial guess of variables $\mathbf{x}$, say $\mathbf{x}_0$.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k \tag{A.1}$$

where

$$\Delta\mathbf{x}_k = -\Big[\mathbf{F}(\mathbf{x})\Big]_{\mathbf{x}=\mathbf{x}_k} \tag{A.2}$$

until $\Delta\mathbf{x}_k$ becomes less than a given tolerance.

Appendix B

Program listings

B.1

Main program

```
--PROGAM prog1.occ (first go at cracking the robot torque and
--trajectory calculations)
#include "hostio.inc"
PROC occam.program (CHAN OF SP fs, ts, []INT memory)
  #USE "hostio.lib"
  REAL64 B,C,D,E,F,H,P,R,S,T,U,V,W,Y,Z :
  REAL64 M,IM,DD :
  [3]REAL64 TH0 : --when start movement start angles .
  [3]REAL64 THT : --when end movement end angles required
  [3]REAL64 TH :
  [3]REAL64 THDOT : --to have angular velocity variable.
  [3]REAL64 THDDOT :--to have angular acceleration variable.
  [3]REAL64 TORQUE :--toque variable.
  [3]REAL64 THD :
```

```
[3]REAL64 THDD :
[3]REAL64 A : --commmon variables correspond to robot.
[3]REAL64 K :  --commmon variables correspond to robot constants
[3]REAL64 L :  --commmon variables correspond to robot link lengthes
[2,3]REAL64 O :--matrix of coordinates of fixed joints
[3]REAL64 THETA :--angle from X_(i-1) to X_i in the direction of Z_i.
[2,2,9]REAL64 Q :
REAL64 TS, TE, TSTEP:
[7]REAL64 M :--scalar m_i(mass of the i-th link)
[7]REAL64 IM :--moment of inertia about mass center of Link i.
[2,9]REAL64 RO :--vector directed to the mass center of the i-th link
INT IND: --index of initial guess-searching for subroutine POSITN.
[2]REAL64 G : --commmon variables correspond to gravity
#INCLUDE "datain.inc"
#INCLUDE "setup.inc"
DD := (DACOS(-1.0(REAL64)) / (180.0(REAL64)))
PROC JOTRAJ (REAL64 T,TT,[3]REAL64 THO,THETA,THDOT,THDDOT)
              --look for TT declaration

  --compute postions, velocities, and accelerations in joint space
  --with arbitrary time t.
  REAL64 B,C,D,E,F,G,H,L,K,O,P,R,S,T,U,V,W,",Z :
  REAL64 PI,DD,BT:
  [3]REAL64 A, TO, THT:
  SEQ
    PI:= DACOS(-1.00(REAL64))
    DD:=(PI)/ (180.00(REAL64))
    B:= (PI+PI) /TT
    SEQ
```

```
      PAR
        TO[0]:=THO[0] * DD
        A[0]:=((THT[0] -THO[0]) * DD )/ (PI + PI)
        TO[1]:=THO[1] * DD
        A[1]:=((THT[1] -THO[1]) * DD )/ (PI + PI)
        TO[2]:=THO[2] * DD
        A[2]:=((THT[2] -THO[2]) * DD )/ (PI + PI)
      SEQ
        BT:= B*T
        PAR
          THETA[0]:=TO[0]+(A[0] * (BT - DSIN(BT)))
          THDOT[0]:=A[0] * B * (1.0(REAL64)) -DCOS(BT)
          THDDOT[0]:= A[0] * B * B * (DSIN(BT))
          THETA[1]:=TO[1]+(A[1] * (BT - DSIN(BT)))
          THDOT[1]:=A[1] * B * (1.0(REAL64)) -DCOS(BT)
          THDDOT[1]:= A[1] * B * B * (DSIN(BT))
          THETA[2]:=TO[2]+(A[2] * (BT - DSIN(BT)))
          THDOT[2]:=A[2] * B * (1.0(REAL64)) -DCOS(BT)
          THDDOT[2]:= A[2] * B * B * (DSIN(BT))

:
PROC LUDECP ([NA,N]REAL64 A,INT NA, N,[NL,N]REAL64 L,INT NL,[NU,N]REAL64 U,
             INT NU)
  -- Decomposing of a gicen n by n matrix using Crout's method as follows:
  -- A = L * U
  -- Where L and U are lower and upper matrices, respectively
  -- Input:
  --        A --- given N * N matrix
  --        NA --row dimension of array A in the caller
```

```
--        N -- dimension of matrix A
--        NL,NU -- row dimensions of arrays L and U in the Caller
-- outputs:
--        L-- N * N lower triangular matrix
--        U-- N* N  upper triangular matrix
REAL64 SUM :
INT J,K,I.N:
-- INT NA,N,NL,NU,J,K:
--[NA,N]REAL64 A:
--[NL,N]REAL64 L:
--[NU,N]REAL64 U:
SEQ I = 1 FOR N
  SEQ J = 1 FOR N
    L[(I-1(INT)),(J-1(INT))] := A[(I-1(INT)),(J-1(INT))]
SEQ K = 1 FOR N
  SEQ I = K FOR N
    SUM := 0.0
    SEQ J = 1 FOR (K-1(INT))
      SUM := SUM + ( L[(I-1(INT)),(J-1(INT))] * L[(J-1(INT)),(K-1(INT))])
    L[(I-1(INT)),(K-1(INT))]:= L[(I-1(INT)),(K-1(INT))]- SUM
  SEQ J := (K+1(INT)) FOR N
    SUM := 0.0
    SEQ I = 1 FOR (K-1(INT))
      SUM := SUM + (L[(K-1(INT)),(I-1(INT)] * L[(I-1(INT)),(J-1(INT))])
    L[(K-1(INT)),(J-1(INT))]:=(L[(K-1(INT)),
                              (J-1(INT))]-SUM)/(L[(K-1(INT)),(K-1(INT))])
SEQ I = 1 FOR N
  U[(I-1(INT)),(I-1(INT))] := 1.0(REAL64)
  SEQ J = (I + 1(INT)) FOR N
```

```
      U[(I-1(INT)),(J-1(INT))] := L[(I-1(INT)),(J-1(INT))]
  SEQ I = 1 FOR N
    SEQ J = (I+1(INT)) FOR N
      L[(I-1(INT)),(J-1(INT))] := 0.0
      U[(J-1(INT)),(I-1(INT))] := 0.0


:
PROC LINERL ([NL,N]REAL64 L,INT NL,N,[N]REAL64 B, X)
  -- This procedure solves a lower triangle linear system
  -- L(1,1) * X(1)                          =B(1)
  -- L(2,1) * X(1) + L(2,2) * X(2)          =B(2)
  -- .........
  -- L(N,1) * X(1) + L(N,2) * X(2)+...+L(N,N)*X(N)=B(N)
  -- for N-dimensional vector X ,Argument NL is the row dimension of array
  -- L which should be the same as that in the caller
  REAL64 SUM:
  INT J,I:
  I := 1
  SEQ
    IF
      (DABS(L[(I-1(INT)),(I-1(INT))])) < (1.0E-40(REAL64))
        SKIP
      NOT (DABS(L[I,I])) < (1.0E-40(REAL64))
        X[I] := B[I] / L[I,I]
         SEQ I = 2 FOR N
           SUM := 0.0(REAL64)
           SEQ J = 1 FOR (I-1(INT))
             SUM := SUM + (L[(I-1(INT)),(J-1(INT))] * X[(J-1(INT))])
           IF
```

```
            (DABS(L[(I-1(INT)),(I-1(INT))])) < (1.0E-40(REAL64))
              SKIP
            NOT (DABS(L[(I-1(INT)),(I-1(INT))])) < (1.0E-40(REAL64))
              X[I] := (B[I] - SUM ) / L[I,I]


:
PROC LINERU ([NU,N]REAL64 U,INT NU,N,[N]REAL64 B,X)
  -- Solves a upper triangle linear system, i.e.
  --
  -- U(1,1)*X(1) + U(1,2) * X(2) + ... +U(1,N) * X(N) = B(1)
  --               U(2,2) * X(2) + ... +U(2,N) * X(N) = B(2)
  --                                .........
  --                                       U(N,N) * X(N) = B(N)
  --For N-dimensional vector x.argument NU is the raw dimension of array U.
  REAL64 SUM :
  INT K,J,I:
  K := N
  SEQ
    IF
      (DABS(U[(K-1(INT)),(K-1(INT))]) < (1.0E-40(REAL64)))
        SKIP
      NOT (DABS(U[(K-1(INT)),(K-1(INT))]) < (1.0E-40(REAL64)))
        X[(K-1(INT))] := (B[(K-1(INT))]) / (U[(K-1(INT)),(K-1(INT))])
        SEQ J = 1(INT) FOR (N-1(INT)) -- First loop
            K := N - J
            SUM := 0.0
            SEQ I = (K+1(INT))  FOR N      --Second loop
              SUM := SUM + (U[(K-1(INT)),(I-1(INT))] * X[(I-1(INT))])
            IF
```

```
            (DABS(U[(K-1(INT),(K-1(INT)]) < (1.0E-40(REAL64)))
              SKIP
          X[(K-1(INT))]:=(B[(K-1(INT))]-SUM)/U[(K-1(INT)),(K-1(INT))]

:
PROC FDISSP ([3]REAL64 THETA,THDOT,[21]REAL64 WD)
  -- This procedure is used to evaluate the dissipative forces and
  -- torques exerting on each link.
  INT I:
  PAR I = 0(INT) FOR 20(INT)
    WD[I] := 0.0
:
PROC VTOU ([N]REAL64 V,INT N,[N]REAL64 U)
  -- This procedure is to assign Vector V(N) to Vector U(N).
  INT I:
  PAR  I = 0(INT) FOR (N-1(INT))
    U[I] := V[I]
:
PROC AMLV ([NA,N]REAL64 A,INT NA,M,N,[N]REAL64 V,[M]REAL64 U)
  -- This Procedure computes the product of an m by n matrix A
  -- and an m-dimension vector V, i.e.
  -- U = A * V
  INT I,J :
  SEQ I = 1 FOR M
    U[(I-1(INT))] := 0.0(REAL64)
    SEQ J = 1 FOR N
      U[(I-1(INT))]:=U[(I-1(INT))]+(A[(I-1(INT)),(J-1(INT))]*V[(J-1(INT))])
:
PROC QTMLV ([2,2]REAL64 Q,[2]REAL64 V,U)
```

```
  -- This procedure computes the product of the transpose matrix Q'
  --  2 by 2
  -- and a 2-dimension Vector V i.e.
  --  U = Q * V
  INT I :
  PAR I = 0(INT) FOR 1(INT)
    U[I] := (Q[0,I] * V[0]) + ( Q[1,I] * V[1] )
:
PROC QMLV ([2,2]REAL64 Q,[2]REAL64 V,U)
  -- This procedure computes the product of a 2 by 2 matrix Q
  -- and a 2-dimension Vector V i.e.
  --  U = Q * V
  INT I :
  PAR I = 0(INT) FOR 1(INT)
    U[I] := (Q[I,0] * V[0]) + ( Q[I,1] * V[1] )
:
PROC VECTOP ([2]REAL64 U,V,W,BYTE M)
  -- This Procedure calculates two 2-Dimension vectors U & V
  -- As follows :-
  -- if M = '+' then W = U + V
  -- if M = '-' then W = U - V
  IF
    (M := "+")
      PAR I = 0(INT) FOR 1
        W[I] := U[I] + V[I]
    NOT (M := "+")
      PAR I = 0(INT) FOR 1
        W[I] := U[I] - V[I]
:
```

```
PROC EVALUT ([2,9]REAL64 AA,[2,9]REAL64 RO,[21,3]REAL64 T)
  -- This procedure is used to evaluate the Natural Orthogonal Complement
  -- namely, the 21 by 3 matrix T. The evaluation is done as follows :-
  -- the j-th column of T i scomputed as teh 21-dimension generalized twist
  -- assuming that all the actuated joints are locked but the j-th one has
  -- a unit velocity . the reasulting T is represented in the base
  -- coordinate frame.
  -- Input :-
  --      AA ---Translational vectors represented in the i-th coordinates.
  --      RO ---RO(1,i) & RO(2,*) are the components of teh relative
  --      position vector of mass center of teh i-th link in i-th coordinates.
  --      G  ---G(1,k) & G(2,k) are the components of the gravity acceleration
  --      represented in the k-th base coordinate frame.
  -- Output :-
  --      T ---21x3 matrix of the natural orthogonal complement in the base
  --      coordinates.
  -- Internal variables :
  --  WDOT ----7-dimension vector of angular acceleratins in the iTh
  --           coordinates .
  --  CDOT ----CDOT(1,i) & CDOT(2,i) are components of the acceleration of
  --           the mass center of the iTh link in the iTh coordinates.
  --  O(*,i)---Position vector of teh iTh fixed joint in the base frame.
  --  PE(*,i)--Position vector of the iTh moving joint in E frame.
  --  X      ---4 Dimension working vector.
  -- First step forming the 4x4 matrix A and the 4x3 matrix B:
  [4,4]REAL64 A,L,U:
  [4,3]REAL64 B:
  [4]REAL64 X:
  [7]REAL64 W:
```

```
[2,7]REAL64 CDOT:
INT I,I2,I3,II,K:
PAR
  SEQ I = 0 FOR 1
    A[I,0] := AA[I,3]
    A[I,1] := AA[I,4]
    A[I,2] := 0.0(REAL64)
    A[I,3] := RO[I,6] - RO[I,7]
    I2 := I + 2(INT)
    A[I2,0] := AA[I,3]
    A[I2,1] := 0.0(REAL64)
    A[I2,2] := -AA[I,5]
    A[I2,3] := RO[I,6] -RO[I,8]
  SEQ I = 0 FOR 1
    B[I,0] := -AA[[I,0]
    B[I,1] := AA[I,1]
    B[I,2] := 0.0(REAL64)
    I2 := I + 2 (INT)
    B[I2,0] := -AA[I,1]
    B[I2,1] := 0.0(REAL64)
    B[I2,2] := AA[I,2]
-- Decomposition of A using Crout's Rule.
LUDECP(A,4,4,L,4,,U,4)
-- Solving for (absolute) angular velocities W(i) from : A*W = B*THDOT
PAR I = 0 FOR 2
  W[I] := 0.0(REAL64)
SEQ K = 0 FOR 2
  W[K] := 1(REAL64)
  LINERL(L,4,4,B[0,K],X)
```

```
    LINERU(U,4,4,X,W[3 FOR 6])
  --Computation of the velocities of mass centers in base coordinates
    SEQ I = 0(INT) FOR 2(INT)
      CDOT[0,I] := -RO[1,I] * W[I]
      CDOT[1,I] := RO[0,I] * W[I]
      I3 := I + 3(INT)
      CDOT[0,I3] := CDOT[0,I]-((AA[1,I] - RO[1,I])*W[I])-(RO[1,I3]*W[I3])
      CDOT[1,I3] := CDOT[1,I]+((AA[0,I] - RO[0,I])*W[I])+(RO[0,I3]*W[I3])
    CDOT[0,6] := CDOT[0,3] -((AA[1,3]-RO[1,3])*W[3])-(RO[1,6]*W[6])
    CDOT[1,6] := CDOT[1,3] +((AA[0,3]-RO[0,3])*W[3])+(RO[0,6]*W[6])
    SEQ I =0(INT) FOR 6(INT)
      II := I * 3
      T[II,K] := W[I]
      T[(II+1(INT),K] := CDOT[0,I]
      T[(II+2(INT),K] := CDOT[1,I]
    W[K] := 0.0(REAL64)

:
PROC CPANGO (REAL64 K1,K2,K3,PSI,STH,CTH,INT KTH, JROOT)
  --This procedure is used to compute the sine and cosine of the coupler
  --angle "theta" of an arbitrary RRRR planer four-bar linkage in terms of
  --the given input angle "PSI" and the branch index K.
  --Formulation:
  --Solving the Input-coupler function:
  --K1+(K2*COS(PSI))+(K3*COS(TH))-COS(PSI-TH)=0
  --AS
  --   COS(TH)= (1-T**2)/(1+T**2)
  --   SIN(TH)= 2*T / (1+T**2)
  --Where
```

```
--    T=TAN(TH/2) = (B-KTH*SQRT(B**2-A*C))/A
--    A=K1+(K2+1)*COS(PSI)-K3
--    B=SIN(PSI)
--    C=K1+(K2-1)*COS(PSI)+K3
--INPUT:
--     K1,K2,K3   Linkage parameters
--     PSI input angles (in rads.)meaning angle between the fixed and
--     the input links
--     KTH branch index.
--     if KTH = 1 ( or KTH >= 0 ),then  T=(B+DSQRT(B**2-A*C))/A
--     if KTH=-1 (or KTH < 0 ),then T=(B-SQRT(B**2-A*C))/A
--OUTPUT
--     TH1 coupler angel (in rads.) corresponding to KTH.
--     TH2 coupler angle (in rads.) corresponding to -KTH.
--     JROOT output status index: JROOT=0 no real solution.
--                                        =1, solutions are OK.
INT KTH, JROOT:
REALL64 K1, K2, K3, PSI, STH,CTH, KA, KB, A,C,B, T, Q, TT:
JROOT := 1.0(REAL64)
KA := K1 +(K2 *(DCOS(PSI)))
KB := K3 - (DCOS(PSI))
A := KA - KB
C := KA + KB
B := (DSIN(PSI))
IF
  (DABS(A) < (1.0E-12(REAL64)))
    T := C / (B+B)
  NOT (DABS(A) < (1.0E-12(REAL64)))
  --COS(TH1 /2 ) AND COS (TH2/2) ARE ROOTS OF QUADRATIC EQUATION
```

```
    -- A*T**2 - 2 *B*T+C=0
      Q := (B * B) - (A * C)
        IF
          (Q < (0(REAL64))
            JROOT := 0.0(REAL64)--NEED TO MAKE THE PROGRAM NICE TO PRINT
            SKIP -- REMARK,THERE IS NO REAL ROOTS OF THE QUADRATIC EQUATION
                 -- ENVOLVED .
          NOT (Q < (0(REAL64))
            T := (B + ((DSQRT(Q))*(KTH(REAL64)))) / A
  TT := T * T
  CTH := (1 -TT) /(1 + TT)
  STH := (T + T) /(1 + TT)
:
PROC RRRR4B (REAL64 A1,A2,A3,A4,A5,BETA,PSIO,THO)
  --this procedure is used to precompute some basic data, such as, linkage
  --parameters, bounds of mobility range, etc., of a RRRR planer four-bar
  --linkage. The data computed will be put in a common block named DATA4B
  --which may be accessed by other routines for further computations
  --Input :
  --       A1,A2,A3,A4 ===lengthes of the base, input and output links
  --                        respectively. (link diimensions)
  --       A5 ====Distance from the second joint to the coupler point.
  --       BETA ==Angle defining the shape of the coupler (in deg.)
  --       input angle of the initial congfiguration (deg).
  --OUTPUT :(all the following output data are outputted to the main
  --         program so they can be accessed by other procedures
  --         when neccessary).
  --       A ---5 members vector array containing [A1,A2,A3,A4,A5]
  --       K ---3 members vector array containing linkage parameters
```

```
--      PSIB1-- lower bound of the mobility range(in rad.) of input
--              angle.
--      PSIB2-- upper bound of the mobility range(in rad.) of input
--              angle .
--      MFLAG-- if MFLAG=1, the input link is crank, otherwise it is
--              a rocker.
--      INDEX-- Value of the branch index corresponding to PSIO & THO.
REAL64 PI,DD,BT,PSIB1,PSIB2,:
INT MFLAG,INDEX:
BOOL LOG1, LOG2:
[5]REAL64 A:
[3]REAL64 K:
A[0]:= A1
A[1]:= A2
A[2]:= A3
A[3]:= A4
A[4]:= A5
PI := DACOS(-1.0(REEAL64))
DD := PI /(180(REAL64))
BT := BETA * DD
--CALCULATION OF THE LINKAGE PARAMETERS:
PAR
  K[0] :=((A4*A4)-(A1-A2)-(A2*A2)-(A3*A3))/(2 * A2 * A3)
  K[1] := A1/A3
  K[2] := A1/A2
--CALCULATION OF THE BOUNDS OF MOBILITY RANGE OF THE INPUT LINK:
MFLAG :=  0
LOG1 := (A1+A2) < (A3 +A4)
LOG2 := ((A1-A2) * (A1-A2)) > ((A3-A4)*(A3-A4))
```

```
IF
  (LOG1 AND LOG2) --IF THE LINK IS A CRANK THE ROTATION SHOULD BE 360 DEG.
    PSIB1 := 0
    PSIB2 := PI
    MFLAG := 1
    SKIP ---LINK IS A CRANK
  NOT (LOG1 AND LOG2)
    IF
      ((NOT LOG1) AND (LOG2))
        PSIB2 :=((A1*A1)+(A2*A2)-((A3+A4)*(A3+A4)))/(A1 *A2 *2(RREAL64))
        IF
          (DABS(PSIB2) > 1(REAL64))
            SKIP --A1,A2,A3,A4 ARE INCONSISTANT!!!!
        PSIB2 := DACOS(PSIB2)
        PSIB1 := -PSIB2
      NOT ((NOT LOG1) AND (LOG2))
        IF
          (LOG1 AND (NOT LOG2))
            PSIB1 :=((A1*A1)+(A2*A2)-((A3-A4)*(A3-A4)))/(2(REAL64)*A1*A2)
            IF
              (DABS (PBSI1) > 1(REAL64))
                SKIP --A'S ARE NOT CONSISTANT
            PSIB1 := DACOS (PSIB1)
            PSIB2 := (PI*2(REAL64)) -PSIB1
          NOT (LOG1 AND (NOT LOG2))
            PSIB1 :=((A1*A1)+(A2*A2)-((A3-A4)*(A3-A4)))/(2(REAL64)*A1*A2)
            IF
              (DABS(PSIB1) > 1 )
                SKIP --A1,A2,A3,A4 ARE INCONSISTANT!!!!
```

```
                PSIB1 := DACOS(PSIB1)
                PSIB2 :=((A1*A1)+(A2*A2)-((A3+A4)*(A3+A4)))/(A1*A2*2(REAL64))
                IF
                  (DABS(PSIB2) > 1(RREAL64))
                    SKIP --A1,A2,A3,A4 ARE INCONSISTANT!!!!
                PSIB2 :=DACOS(PSIB2)
  PSIB1 := PSIB1 + (1.0E-6(REAL64))
  PSIB2 := PSIB2 - (1.0E-6(REAL64))
  T0 := (TH0*DD)
  CPANG0(K[0],K[1],K[2],(PSI0*DD),ST,CT,1,JROOT)
  T1 := DATAN2(ST,CT)
  CPANG0(K[0],K[1],K[2],(PSI0*DD),ST,CT,-1,JROOT)
  T2 := DATAN2(ST,CT)
  IF
    (DABS(T1-T0) < DABS(T2-T0))
      INDEX := 1
    NOT (DABS(T1-T0) < DABS(T2-T0))
      INDEX := -1
:
REAL64 FUNCTION DPQ(SP,CP,STH,CTH,XQ,YQ)
  -- This is a function defined as
  -- (distance between Q and P)**2 - (given constant D)**2
  -- input:
  --       SP,CP --sine and cosine of the input angle PSI
  --       STH,CTH --sin and cosin of  the coupler angle theta
  --       XQ,YQ --coordinates off the given point Q.
  --       L2---dimension of the input and output links..
  --       L3 --dimension of the coupler link.
  --       D --given constant.
```

```
  REAL64 SRT,A0,A1,A2,A9,DPQ:
  VALOF
    SRT:= DSQRT(3.0(REAL64))
    PAR
      SEQ
        A0:= ((L2*CP)+((CTH-(SRT3*STH))*(L3/2(REAL64)))-XQ)
        A1:= A0*A0
      SEQ
        A9:= ((L2*SP)+((STH+(SRT3*CTH))*(L3/2(REAL64)))-YQ)
        A2:= A9*A9
    DPQ := A1+A2-(L2*L2) --NOTICE HERE THE CONSTANT D=L2**2
  RESULT DPQ
:
PROC CONF(REAL64 XQ,YD,D,TOL,DPSI,INT KTH,REAL64 PSI,STH,CTH)
  -- THIS PROCEDURE IS USED TO FIND THE CONFIGURATION
  -- This subroutine is used to find the configuration of an RRRR four-bar
  --linkage at which the distance of linkage's coupler point, P, to a
  --specified point, Q, is equal to a given constant D.
  --
  --INPUT:
  --    XQ, YQ ---- coordinates of the given point Q.
  --       D   ---- distance between points P and Q.
  --      TOL  ---- tolerance used to control the numerical method.
  --      DPSI ---- step size of input angle PSI while searching for the root.
  --                NOTE: if DPSI=0, then no searching is required.
  --      KTH  ---- branch index K which equals either 1 or -1.
  --      PSI  ---- initial guess of the input angle PSI (in rads.), which is
  --                useful only when DPSI=0.
  --OUTPUT:
```

```
--      PSI   ---- input angle defining the desired configuration (in rads.).
--    CTH,STH ---- cosine and sine of the coupler angle corresponding to PSI.
--
REAL64 G1,G2,P1,P2,SP2,CP2:
INT I, IJ:
L2 := A[1]
L3 := A[2]
DIS := D
--Settting the initial staus in order to search for initial guess:
P1 := PSI
IF
  (DSPI > (1.0E-7(REAL64)))
    P1 := PSIB1
SP1 := DSIN(P1)
CP1 := DCOS(P1)
CPANGO(K[0],K[1],K[2],P1,st1,CT1,KTH,JROOT)
IF
  (JROOT := 0(REAL64))
    SKIP --skip if
G1:= DPQ(SP1,CP1,ST1,CT1,XQ,YQ)
IF
  (DPSI < (1.0E-7(REAL64))
    P2 := P1 + (0.02(REAL64))
    IF
     (P2 > PSIB2)
       P2 := P1 - (0.2(REAL64))
  NOT (DPSI < (1.0E-7(REAL64))
    BOOL going2, going1:
    going2 := TRUE
```

```
WHILE going2
  IJ := 1
  KT := KTH
  SEQ I = 1 FOR 2 -- the I loop
    P2 := PSIB1 + DPSI
    going1 := TRUE
    WHILE going1
      IF
        (P2 > PSIB2)
          going1 := FALSE
      SP2 := DSIN(P2)
      CP2 := DCOS(P2)
      CPANGO(K[0],K[1],K[2],P2,STH,CTH,KTH,JROOT)
      IF
        (JROOT := 0(REAL64))
          SKIP
      G2:=DPQ(SP2,CP2,ST2,CT2,XQ,YQ)
      IF
        ((G1*G2) < 0(REAL64))
          going2 := FALSE
      G1 := G2
      P1 := P2
      SP1 := SP2
      CP1 := CP2
      ST1 := ST2
      CT1 := CT2
    IJ := IJ + 1(INT)
    IF
      NOT ( IJ > 2 )
```

```
            SKIP
          KT := -KTH
          STOP --CONFIG: No solution found in both branches!exit!!!
  BOOL going:
  INT I:
  going := TRUE
  WHILE going
    I := 0
    SP2 := DSIN(P2)
    CP2 := DCOS(P2)
    CPANGO(K[0],K[1],K[2],P2,STH,CTH,KTH,JROOT)
    G2 := DPQ(SP2,CP2,STH,CTH,XQ,YQ)
    IF
      (DABS(G2) < TOL)
        SKIP
      NOT (DABS(G2) < TOL)
        PSI := ((P1*G2)-(P2*G1))/(G2-G1)
        P1 := P2
        P2 := PSI
        G1 := G2
        I := I+1
        IF
          ( I >= 99 )
            going := FALSE
  DD := 180.0(REAL64) / DACOS(-1(REAL64))
  PSI := PSI * DD
  THETA := (STH /CTH) *DD
  G2 := G2
:
```

```
PROC POSITN(INT IND,[3]REAL64 THETA,REAL64 L0,L1,L2,L3,
                                        [2,3]REAL64 O,[2,7]REAL64 P)
 -- This Procedure is used to solve the position problem of the direct
 -- kinematics for the 3-DOF planar manipulator.
 --
 --INPUTS:
 --     IN  ---- If IND=0, required to search for the initial configuration;
 --               if IND=other, the previous configuration is taken as the
 --           initial guess in the iterative procedure. (NOTE: after each
 --           call to this Procedure, IND is increased by 1 automatically)
 --              THETA ---- 3-D vector of independent input angles (in rad).
 --              L0,L1,L2,L3 ---- dimensions of links.
 --       O(*,i) ---- position vector of the i-th fixed joint in base frame.
 --
 --   OUTPUTS:
 --   AA      ---- translational vectors represented in the base frame.
 --   P(*,i) ---- position vector of the i-th moving joint in base frame.
 --
 -- INTERNAL VARIABLES:
 -- PE(*,i)---- position vector of the i-th moving joint in E frame.
 -- X    ----- 2-D working vector.
 REAL64 CA,SA,PI,B,UB1,UB2,th0,CP,SP:
 [2,7]REAL64 PE:
 [3]REAL64 STH,CTH:
 [2,2]REAL64 Q02,Q03,Q:
 [2]REAL64 X:
 PI := DACOS(-1(REAL64))

 -- Rotation matrices [Q_02]_0 and [Q_03]_0:
```

```
CA := 0.5(REAL64)
SA := DSQRT(3(REAL64)) / (2(REAL64))
Q02[0,0] := -CA
Q02[0,1] := -SA
Q02[1,0] := SA
Q02[1,1] := -CA
Q03[0,0] := -CA
Q03[0,1] := SA
Q03[1,0] := -SA
Q03[1,1] := -CA

--Position vectors of the first three moving joints:

INT I:
PAR I = 1 FOR 3
  SEQ
    STH[I-1(INT)] := DSIN(THETA[I-1(INT)])
    P[1,I-1(INT)] : L1 *(STH[I-1(INT)])
  SEQ
    CTH[I-1(INT)] := DCOS(THETA[I-1(INT)])
    P[0,I-1(INT)] := L1 * (CTH[I-1(INT)])
QMLV(Q02,P[0,1],X)
VECTOP(O[0,1],X,P[0,1],'+')
QMLV(Q03,P[0,2],X)
VECTOP(O[0,2],X,P[0,2],'+')

--Determination of PSI and PHI using numerical method:
```

```
UB1 := ((P[0,0)-P[0,1]) * (P[0,0)-P[0,1]))
UB2 := ((P[1,0]-P[1,1]) * (P[1,0]-P[1,1]))
B := DSQRT( UB1 + UB2 )
th0 := 1(REAL64)
RRRR4B(B,L2,L3,L2,L3,(PI/3),PSI0,th0)
Q[0,0] := (P[0,1]-P[0,0])/ B
Q[0,1] := -( (P[1,1]-P[1,0])/ B )
Q[1,0] := -Q[0,1]
Q[1,1] := Q[0,0]
VECTOP(P[0,2],P[0,0],X,'-')
QTMLV(Q,X,PE[0,2])
IF
  (IND := 0 (INT))
    CONFG(PE[0,2],PE[1,2],L2,(1.0E-10(REAL64)),
                              (1.7E-2(REAL64),1,PSI,SPHI,CPHI)
  NOT (IND := 0 (INT))
    CONFG(PE[0,2],PE[1,2],L2,(1.0E-10(REAL64)),
                              (0.0(REAL64)),1,PSI,SPHI,CPHI)
--Increases index of the initial-configuration-searching by one:

IND := IND + 1

--Coordinates of moving joints in E frame which depends on the positions of
--the first and second moving joints:
CP := DCOS(PSI)
SP := DSIN(PSI)
PE[0,3] := L2 * CP
PE[1,3] := L2 * SP
PE[0,4] := PE[0,3] + (L3 * CPHI)
```

```
  PE[1,4] := PE[1,3] + (L3 * SPHI)
  PE[0,5] := PE[0,3] + (L3 * (CPHI - (DSQRT(3.0(REAL64)) * SPHI))/2)
  PE[1,5] := PE[1,3] + (L3 * (SPHI + (DSQRT(3.0(REAL64)) * CPHI))/2)
  PE[0,6] := (PE[0,3] + PE[0,4] + PE[0,5]) / 3(REAL64)
  PE[1,6] := (PE[1,3] + PE[1,4] + PE[1,5]) / 3(REAL64)

  --Coordinates of moving joints in the base frame:

  INT I:
  SEQ I := 3 FOR 6
    QMLV(Q,PE[0,I],X)
    VECTOP(P[0,0],X,P[0,I],'+')
:
PROC KINET0([3]REAL64 THDOT,THDDOT,[2]REAL64 G,[2,9]REAL64 AA,R0,
                          [7]REAL64 W,WDOT,[2,7]REAL64 CDOT,CDDOT)
  -- This  procedure is used to solve the velocity- and acceleration-problem
  -- of the direct kinematics for the 3-DOF planar manipulator. All the output
  -- results are represented in the base coordinate frame.
  -- INPUT:
  --    THDOT  ---- 3-D vector of independent input angular rates (in rad).
  --    THDDOT ---- 3-D vector of independent input angular accelerations
  --                (in rad./s**2).
  --     G   ---- 2-D vector of the gravity acceleration in base coordinates.
  --    AA   ---- translational vectors represented in the base frame.
  --    R0   --- R0(1,i) & R0(2,*) are the components of the relative position
  --             vector of mass center of the i-th link in base coordinates.
  -- OUTPUT:
  --    WDOT ---- 7-D vector of angular accelerations in base coordinates.
  --    CDOT ---- CDOT(1,i) & CDOT(2,i) are the components of the velocity of
```

```
--              the mass center of the i-th link in base coordinates.
--CDDOT ---- CDDOT(1,i) & CDDOT(2,i) are the components of the acceleration
--              of the mass center of the i-th link in base coordinates.
--  INTERNAL VARIABLES:
--     O(*,i)  ---- position vector of the i-th fixed joint in base frame.
--     PE(*,i) ---- position vector of the i-th moving joint in E frame.
--       X     ---- 2-D working vector.
[4,4]REAL64 A,L,U:
[4,3]REAL64 B:
[7]REAL64 WW:
[4]REAL64 X,Y,Z:
INT I,I2,I3:
REAL64 BUD1,BUD2,BUD3,BUD4,BUD5,BUD6,BUD7,BUD8:
PAR
  SEQ I = 0 FOR 1
    A[I,0] := AA[I,3]
    A[I,1] := -AA[I,4]
    A[I,2] := 0.0(REAL64)
    I2 := I + 2(INT)
    A[I2,0] := AA[I,3]
    A[I2,1] := 0.0(REAL64)
    A[I2,2] := -AA[I,5]
    A[I2,3] := RO[I,6] - RO[I,8]
  SEQ I = 0 FOR 1
    B[I,0] := -AA[I,0]
    B[I,1] := AA[I,1]
    B[I,2] := 0.0(REAL64)
    I2 := I + 2(INT)
    B[I2,0) := -AA[I,1]
```

```
    B[I2,1] := 0.0(REAL64)
    B[I2,2] := AA[I,2]

 --Decomposition of A using Crout's rule:

LUDECP(A,4,4,L,4,U,4)

--Solving for (absolute) angular velocities W(i) from: A*W=B*THDOT:

VTOU(THDOT,3,W)
PAR I = 0 FOR 3
  X[I] := (B[I,0]*THDOT[0]) + (B[I,1] * THDOT[1]) + (B[I,2] * THDOT[2])
LINERL(L,4,4,X,Y)
LINERU(U,4,4,Y,W[3])

--Computation of the velocities of mass centers in base coordinates:

SEQ I = 0 FOR 2
  CDOT[0,I] := -RO[1,I] * W[I]
  CDOT[1,I] := RO[0,I] * W[I]
  I3 := I + 3(INT)
  CDOT[0,I3] := CDOT[0,I] - ((AA[1,I]-RO[1,I]) * W[I]) - (RO[1,I3]*W[I3])
  CDOT[1,I3] := CDOT[1,I] + ((AA[0,I]-RO[0,I]) * W[I]) + (RO[0,I3]*W[I3])
CDOT[0,6] := CDOT[0,3] - ((AA[1,3] - RO[1,3]) * W[3]) - (RO[1,6] * W[6])
CDOT[1,6] := CDOT[1,3] + ((AA[0,3] - RO[0,3]) * W[3]) + (RO[0,6] * W[6])

--Computation of (absolute) angular accelerations:

PAR I = 0 FOR 6
```

```
    WW[I] := W[I] * W[I]
  VTOU(THDDOT,3,WDOT)
  AMLV(B,4,4,3,WW[0],X)
  AMLV(B,4,4,3,THDDOT,Y)
  AMLV(A,4,4,4,WW[3],Z)
  Y[0] := -X[1] + Y[0] + Z[1]
  Y[1] := X[0] + Y[1] - Z[0]
  Y[2] := -X[3] + Y[2] + Z[3]
  Y[3] := X[2] + Y[3] - Z[2]
  LINERL(L,4,4,Y,X)
  LINERU(U,4,4,X,WDOT[3])
  --Computation of accelerations of mass centers in base coordinates:
  SEQ I = 0 FOR 2
    CDDOT[0,I] := G[0] - (RO[1,I] * WDOT[I]) - (RO[0,I] * WW[I])
    CDDOT[1,I] := G[1] + (RO[0,I] * WDOT[I]) - (RO[1,I] * WW[I])
    I3 := I + 3(INT)
    BUD1:= CDDOT[1,I]-((AA[0,I]-RO[0,I])*WW[I])-(RO[0,I3]*WW[I3])
    BUD2 := ((AA[1,I]-RO[1,I])*WDOT[I])-(RO[1,I3]*WDOT[I3])
    CDDOT[0,I3] := BUD1 - BUD2
    BUD3 := CDDOT[1,I]-((AA[1,I]-RO[1,I])*WW[I]) - (RO[1,I3]*WW[I3]
    BUD4 := ((AA[0,I]-RO[0,I])*WDOT[I])+(RO[0,I3]*WDOT[I3])
    CDDOT[1,I3] := BUD3 + BUD4
  BUD5 := CDDOT[0,3]-((AA[0,3]-RO[0,3])*WW[3])-(RO[0,6]*WW[6])
  BUD6 := ((AA[1,3]-RO[1,3])*WDOT[3])- (RO[1,6]*WDOT[6])
  CDDOT[0,6] := BUD5 - BUD6
  BUD7 := CDDOT[1,3]-((AA[1,3]-RO[1,3])*WW[3]) - (RO[1,6]*WW[6])
  BUD8 := ((AA[0,3]-RO[0,3])*WDOT[3]) + (RO[0,6]*WDOT[6])
  CDDOT[1,6] := BUD7 + BUD8
:
```

```
#INCLUDE "datain.inc"
PROC  EVALQA([3]REAL64 THETA,[2,9]REAL64 AA,RHO)
  -- INPUT:
  --    THETA ---- 3-D vector of independent input angles (in rad).
  -- OUTPUT:
  --    AA    ---- translational vectors represented in the base frame.
  -- INTERNAL VARIABLES:
  --    P(*,i)  ---- position vector of the i-th moving joint in base frame.
  --    O(*,i)  ---- position vector of the i-th fixed  joint in base frame.
  --    PE(*,i) ---- position vector of the i-th moving joint in E frame.
  --      X     ---- 2-D working vector.
  INT I,I3,I6:
  REAL64 STH,CTH,RHO:
  --Determination of the configuration corresponding to the input THETA:

  POSITN(IND,THETA,L[0],L[1],L[2],L[3],O,P)

  --Calculation of the translational vectors in base coordinate frame:

  SEQ I = 0 FOR 2
    VECTOP(P[0,I],O[0,I],AA[0,I],'-')
    I3 := I + 3(INT)
    VECTOP(P[0,I3],P[0,I],AA[0,I3],'-')
    I6 := I + 6(INT)
    VECTOP(P[0,6],P[0,I6],AA[0,I6],'-')

  --Calculation of the relative position vector of mass center of each link:

  SEQ I = 0 FOR 2
```

```
    STH := (P[1,I]-O[1,I])/L[1]
    CTH := (P[0,I]-O[0,I])/L[1]
    RHO[0,I] := (CTH * RO[0,I]) - (STH * RO[1,I])
    RHO[1,I] := (STH * RO[0,I]) + (CTH * RO[1,I])
  SEQ I = 3 FOR 5
    I3 := I - 3(INT)
    STH := (P[1,I]-P[1,I3])/L[2]
    CTH := (P[0,I]-P[0,I3])/L[2]
    RHO[0,I] := (CTH * RO[0,I]) - (STH * RO[1,I])
    RHO[1,I] := (STH * RO[0,I]) + (CTH * RO[1,I])
  SEQ I = 6 FOR 8
    RHO[0,I] := (P[0,6]- P[0,(I-3(INT))])
    RHO[1,I] := (P[1,6] - P[1,(I-3(INT))])
:
PROC INVDYN([3]REAL64 THETA,THDOT,THDDOT,FORCE)
  --This procedure is used to compute the generalized driving force acting
  --on the three input links in terms of the three independent input joint
  --positions, velocities and accelerations, as the following:
  --
  --              FORCE = T' * M * TDOT + T' * WD
  --
  --   where  T' ---- transpose of the 21x3 orthogonal complement matrix.
  --           M  ---- 21x21 generalized mass matrix in base frame.
  --         TDOT ---- time derivative of the 21-D generalized twist.
  --          WD  ---- 21-D vector-function of the dissipative force.
  REAL64 BUDY  :
  INT I,II,J  :
  [7]REAL64 W,WDOT:
  [2,7]REAL64 CDOT,CDDOT:
```

```
    [21,3]REAL64 TT:
    [21]REAL64 MTDOT:
    --Although writng to a file here will casue a lot delays it is
    --a good idea to write the following kinematic analysis to a file
    --later.
    -- Direct kinematic analysis

    EVALQA(THETA,AA,RHO)
    KINETO(THDOT,THDDOT,G,AA,RHO,W,WDOT,CDOT,CDDOT)
    EVALUT(AA,RHO,TT)

    --Evaluation of the dynamic equations

    SEQ I = 0 FOR 6
      II := I * 3(INT)
      MTDOT[II] := IM[I] * WDOT[I]
      MTDOT[II+1(INT)] := M[I] * CDDOT[0,I]
      MTDOT[II+2(INT)] := M[I] * CDDOT[1,I]
    FDISSP(THETA,THDOT,WD)
    SEQ I = 1 FOR 3
      FORCE[(I-1(INT)] := 0.0(REAL64)
      SEQ J := 1 FOR 21
        BUDY := TT[(J-1(INT),(I-1(INT))]*(MTDOT[(J-1(INT))]-WD[(J-1(INT))])
        FORCE[(I-1(INT)]:=  FORCE[(I-1(INT)] + BUDY

:
PROC EVATDD([3]REAL64 THETA,THDOT,FORCE,THDDOT)
  -- This subroutine is used to compute the angular accelerations of the three
  -- independent input links in terms of the given angular positions and
```

```
-- velocities of the same links as well as the generalized driving force
-- exerting on the three input links.
--  INPUT:
--     THETA  ---- angular positions of the three input links (in rad).
--     THDOT  ---- angular velocities of the three input links (in rad/s).
--     FORCE  ---- generalized driving force acting on the input links.
--  OUTPUT:
--  THDDOT -- angular accelerations of the three input links (in rad/s/s).
REAL64 BILL,BUN:
INT I, II,J,K:
[7]REAL64 W,WDOT:
[2,7]REAL64 CDOT,CDDOT:
[21,3]REAL64 TT:
[21]REAL64 MM,MTDOT:
[3,3]REAL64 H,L:
[3]REAL64 PHI:

--Evaluation of the time derivatives of the generalized twists
--such that the input angular accelerations are set to zero.

PAR I = 0 FOR 2
  THDDOT[I] := 0.0(REAL64)
EVALQA(THETA,AA,RHO)
KINETO(THDOT,THDDOT,G,AA,RHO,W,WDOT,CDOT,CDDOT)

--Evaluation of the 21 by 3 orthogonal complement matrix

EVALUT(AA,RHO,TT)
```

```
--Evaluation of the dynamics equations excluding the inertia terms.

SEQ I = 0 FOR 6
  II := I * 3(INT)
  MTDOT[II] := IM[I] * WDOT[I]
  MTDOT[II+1(INT)] := M[I] * CDDOT[0,I]
  MTDOT[II+2(INT)} := M[I] * CDDOT[1,I]
FDISSP(THETA,THDOT,WD)
SEQ I = 1 FOR 3
  PHI[(I-1(INT))] := FORCE[(I-1(INT))]
  SEQ J = 1 FOR 21
    BILL := (TT[(J-1(INT)),(I-1(INT))]*(WD[(J-1(INT))]-MTDOT[(J-1(INT))])
    PHI[(I-1(INT))]:= PHI[(I-1(INT))]+ BILL

--Evaluation of the generalized inertia matrix H.

SEQ I = 0 FOR 6
  II := I * 3(INT)
  MM[II] := IM[I]
  MM[II+1(INT)] := M[I]
  MM[II+2(INT)] := M[I]
SEQ I = 1 FOR 3
  SEQ J = I FOR 3
    H[(I-1(INT)),(J-1(INT))] := 0.0(REAL64)
    SEQ K = 0 FOR 20
      BUN := (TT[K,(I-1(INT))] * (MM[K] * TT[K,(J-1(INT))]))
      H[(I-1(INT)),(J-1(INT))]:=H[(I-1(INT)),(J-1(INT))] + BUN
    H[(I-1(INT)),(J-1(INT))] :
```

```
    --Solving the linear equations (dynamics model) for the angular
    --accelerations using Cholesky decomposition, namely, H = L'*L.
    SEQ
      L[2,2] := DSQRT(H[2,2])
      PAR
        L[2,1] := H[2,1] / L[2,2]
        L[2,0] := H[2,0] / L[2,2]
      L[1,1] := DSQRT(H[1,1] - (L[2,1]*L[2,1]))
      L[1,0] := (H[1,0] - (L[2,1] * L[2,0])) / L[1,1]
      L[0,0] := DSQRT(H[0,0] - (L[1,0] * L[1,0]) - (L[2,0] * L[2,0]))
      THDDOT[2] := PHI[2] / L[2,2]
      THDDOT[1] := (PHI[1] - (THDDOT[2] * L[2,1])) / L[1,1]
      THDDOT[0]:=(PHI[0]-(THDDOT[1]*L[1,0])-(THDDOT[2] * L[2,0]))/L[0,0]
      THDDOT[0] := THDDOT[0] / L[0,0]
      THDDOT[1] := (THDDOT[1] - (THDDOT[0]*L[1,0]))/L[1,1]
      THDDOT[2] := (THDDOT[2]-(THDDOT[0]*L[2,0])-(THDDOT[1]*L[2,1]))/L[2,2]
:
PROC FORCE(REAL64 T,[3]REAL64 TORQUE)
  --This procedure provides the generalized input tauque to each input link
  --as a function of time.
  [3]REAL64 THETA,THDOT,THDDOT:
  JOTRAJ(T,tt,th0,tht,THETA,THDOT,THDDOT)
  INVDYN(THETA,THDOT,THDDOT,TORQUE)
:
PROC RUNKT(REAL64 T0,T1,[3]REAL64 THETA,THDOT,INT IN)
  --This procedure solves the first-order differential equations of motion
  --for 6-dimensional state-variable vector [THETA',THDOT']' where ' stands
  --for transpose. The integral technique used is the 4th order Runge-Kutta
  --method with the given step size of T1-T0.
```

```
--INPUT:
--          T0  ---- initial time when the initial conditions are given.
--          T1  ---- final time when the output is desired.
--       THETA ---- initial values of the first three components of state
--                  variables (joint angles).
--       THDOT ---- initial values of the last three components of state
--                   variables (joint rates).
--         IN  ---- counter of calls. IN must be set to zero in the first
--                call to this subroutine, while it could be any value
--                   in the subsequential calls.
--    OUTPUT:
--       THETA ---- final values of the first three components of the state
--                   variables (joint angles).
--         THDOT ---- final values of the last three components of the state
--                   variables (joint rates).
REAL64 H:
[3]REAL64 THDDOT, TAU,X,XDOT:
[4,3]REAL64 K,KD:
H := T1 -T0
FORCE(T,TAU)
PAR I = 0 FOR 2
  X[I] := THETA[I]
  XDOT[I]:= THDOT[I]
IF
  (IN <> 0(INT))
    EVATDD(X,XDOT,TAU,THDDOT)
PAR I = 0 FOR 2
  K[0,I] := ( H * XDOT[I])
  KD[0,I] := ( H * THDDOT[I])
```

```
-- Computation of K2's:

T := T0 +(H/2(REAL64))
FORCE(T,TAU)
PAR FOR I = 0 FOR 2
  X[I] := THETA[I] + (K[0,I]/(2(REAL64)))
  XDOT[I] := THDOT[I] +(KD[0,I]/(2(REAL64)))
EVATDD(X,XDOT,TAU,THDDOT)
PAR I =0 FOR 2
  K[1,I] := (H * XDOT[I])
  KD[1,I] := (H * THDDOT[I])

--Computation of K3's:

T := T0 +(H/2(REAL64))
FORCE(T,TAU)
PAR I = 0 FOR 2
  X[I] := THETA[I] + (K[1,I]/(2(REAL64)))
  XDOT[I] := THDOT[I] + (KD [1,I]/(2(REAL64)))
EVATDD(X,XDOT,TAU,THDDOT)
PAR I = 0 FOR 2
  K[2,I] := (H * XDOT[I])
  KD[2,I] := H * THDDOT[I]

--Computation of K4's:

T := T0 + H
FORCE(T,TAU)
```

```
  PAR I = 0 FOR 2
    X[I] := THETA[I] + K[2,I]
    XDOT[I] := THDOT[I] + KD[2,I]
  EVATDD(X,XDOT,TAU,THDDOT)
  PAR I = 0 FOR 2
    K[3,I] := H * XDOT[I]
    KD[3,I] := H * THDDOT[I]

  --Evaluation of the state variables at time T1:

  PAR I = 0 FOR 2
    THETA[I] := THETA[I]+((K[0,I]+(K(1,I]*2)+(K[2,I]*2)+K[3,I])/(6(REAL64)))
    THDOT[I]:= THDOT[I]+((KD[0,I]+(KD[1,I]*2)+KD[2,I]*2)+KD[3,I])/(6(REAL64)))
  -- icreasing the index

  IN := IN + 1(INT)
:
PROC FOWDYN(REAL64 TS,TE,STEP,[3]REAL64 THETA,THDOT,THDDO,INT IN)
  --This subroutine is used to simulate the motion of the 3-DOF planar
  --manipulator for the three supplied independent driving torques
  --(generalized forces) acting on the three input links from given initial
  --angles and angular velocities of the three input links.
  --
  --   INPUT:
  --          TS  ---- initial time associated with the initial conditions.
  --          TE  ---- final time at which the output result is desired.
  --        STEP ---- time step of integration.
  --       THETA ---- initial values of the first three components of state
  --                    variables (joint angles).
```

```
    --        THDOT ---- initial values of the last three components of state
    --                     variables (joint rates).
    --          IN  ---- counter of calls. IN must be set to zero in the first
    --                   call to this subroutine, while it could be any value
    --                     in the subsequential calls.
    --
    --  OUTPUT:
    --        THETA ---- input angles obtained by simulation (in rad).
    --      THDOT ---- angular velocities obtained by simulation (in rad/s).
    --      THDDOT ---- angular accelerations obtained by simulation (in rad/ss)
    [3]REAL64 TAU,TDD:
    REAL64 T0,T1
    INT I:
    T0 := TS
    T1 := (TS + STEP)
    WHILE T1 <  TE
      RUNKT(T0,T1,THETA,THDOT,IN)
      FORCE(T1,TAU)
      EVATDD(THETA,THDOT,TAU,THDDOT)
      T0 := T1
      T1 := T1 + STEP
    PAR I = 0 FOR 2
      TDD[I] := THDDOT[I]
:
#INCLUDE "setup.inc"
--CHAN OF REAL64 ResutTorqueMot1 :
--CHAN OF REAL64 ResutTorqueMot2 :
--CHAN OF REAL64 ResutTorqueMot3 :
--CHAN OF REAL64 ResultMot1Theta :
```

```
--CHAN OF REAL64 ResultMot2Theta :
--CHAN OF REAL64 ResultMot3Theta :
PROTOCOL Torque.out IS REAL64; REAL64; REAL64:
CHAN OF Torque.out ResultTorqueMotor :
-- sending torques via link 3  transputer at slot 0 out to manipulator
PLACE ResutTorqueMot AT 3 :
--reading thetas from manipulator remove comment marks if needed
--PROTOCOL Theta.in IS REAL64; REAL64; REAL64:
--CHAN OF Theta.in RealtimeTheta :
--PLACE RealtimeTheta AT 7 :
--RealtimeTheta ? TH[0]; TH[1]; TH[2]

--Computation of the initial conditions:
JOTRAJ(TS,TT,TH0,THT,THETA,THDOT,THDDOT)
--calculation of torques and thetas at a given time t
IN := 0(INT)
T := TS
WHILE T < (TE + (TSTEP/(10(REAL64))))
  FOWDYN(T0,T,TSTEP,THETA,THDOT,THDDOT,IN)
  JOTRAJ(T,TT,TH0,THT,TH,THD,THDD)
  FORCE(T,TORQUE)
  ResultTorqueMotor ! TORQUE[0]; TORQUE[1]; TORQUE[2]
  --ResultMot1Theta ! TH[0]
  --ResultMot2Theta ! TH[1]
  --ResultMot3Theta ! TH[2]
  --ResutTorqueMot1 ! TORQUE[0]
  --ResutTorqueMot2 ! TORQUE[1]
  --ResutTorqueMot3 ! TORQUE[2]
  T := T + TSTEP
```

:

B.2

Data input file

```
--This file should contain setup and required performance data
--fill in the data adhering to the specified format
--Time period to define trajectory in seconds

TT :=

--Enter time to start in seconds in max three digit format no decimal points.

TS :=

--Enter the time step size

TSTEP :=

--Enter the end time

TE :=

--Start angles in degrees three digits only allowed, no decimal points allowed
```

```
--For actuator #1

THO[0] :=

--For actuator #2

THO[1] :=

--For actuator #3

THO[2] :=

--End angles in degrees three digits only allowed, no decimal points allowed

--For actuator #1

THT[0] :=

--For actuator #2

THT[1] :=

--For actuator #3

THT[2] :=
```

Appendix C

Program subroutines overview

The following is a brief description of the program subroutines which construct its main building blocks, a detailed program listing is provided in Appendix B. For full description of the program's mathematical solution, please refer to chapter 4.

C.1

LUDECP

Decomposing of a given n by n matrix using Crout's method by A = L× U, where L and U are lower and upper matrices, respectively

C.2

LINERL

This procedure solves a lower triangular linear system

```
L(1,1) * X(1)                          =B(1)
L(2,1) * X(1) + L(2,2) * X(2)          =B(2)
.........
L(N,1) * X(1) + L(N,2) * X(2)+...+L(N,N)*X(N)=B(N)
```

for N-dimensional vector X. Argument NL is the row dimension of array L which should be the same as that in the caller procedure.

C.3

LINER

Solves a upper triangular linear system, i.e.

```
U(1,1)*X(1) + U(1,2) * X(2) + ... +U(1,N) * X(N) = B(1)
              U(2,2) * X(2) + ... +U(2,N) * X(N) = B(2)
                                  ..........
                                         U(N,N) * X(N) = B(N)
```

For N-dimensional vector X. Argument NU is the row dimension of array U.

C.4

FDISSP

This procedure is used to evaluate the dissipative forces and torques exerting on each link.

C.5

VTOU

This procedure is to assign Vector V(N) to Vector U(N).

C.6

AMLV

This procedure computes the product of an m by n matrix A and an m-dimension vector V, i.e. U = A × V.

C.7

QTMLV

This procedure computes the product of the transpose matrix Q' (2 by 2) and a vector of 2-dimensions V i.e. U = Q × V.

C.8

QMLV

This procedure computes the product of a 2 by 2 matrix Q and a vector of 2-dimensions V i.e. U = Q × V

C.9

VECTOP

This procedure calculates two 2-Dimension vectors U and V as follows:-

if M = '+' then W= U + V

if M = '−' then W= U − V

C.10

EVALUT

This procedure is used to evaluate the Natural Orthogonal Complement namely, the 21 by 3 matrix T. The evaluation is done as follows:

The j-th column of T is computed as the 21-dimension generalized twist assuming that all the actuated joints are locked but the j-th one has a unit velocity . The resulting T is represented in the base coordinate frame.

C.11

CPANG0

This procedure is used to compute the sine and cosine of the coupler angle "theta" of an arbitrary RRRR planar four-bar linkage in terms of the given input angle "PSI" and the branch index K.

Formulation; by solving the Input-coupler function:

```
K1+(K2*COS(PSI))+(K3*COS(TH))-COS(PSI-TH)=0
```

as

```
COS(TH)= (1-T**2)/(1+T**2)
```

```
SIN(TH)= 2 * T / (1+T**2)
```

where

```
T=TAN(TH/2) = (B-KTH * SQRT(B**2-A * C))/A
```

```
A=K1+(K2+1) * COS(PSI)-K3
```

```
B=SIN(PSI)
```

```
C=K1+(K2-1) * COS(PSI)+K3
```

C.12

RRRR4B

This procedure is used to precompute some basic data, such as, linkage parameters, bounds of mobility range, etc., of a RRRR planar four-bar linkage. The data computed may be accessed by other routines for further computations.

C.13

CONF

This subroutine is used to find the configuration of an RRRR four-bar linkage.

C.14

POSITN

This procedure is used to solve the position problem of the direct kinematics for the 3-DOF planar manipulator.

C.15

KINET0

This procedure is used to solve the velocity- and acceleration-problem of the direct kinematics for the 3-DOF planar manipulator. All the output results are represented in the base coordinate frame.

C.16

INVDYN

This procedure is used to compute the generalized driving force acting on the three input links in terms of the three independent input joint positions, velocities and accelerations, as the following:

FORCE = T' × M × TDOT + T' × WD

where

T': is the transpose of the 21x3 orthogonal complement matrix.

M: is the 21x21 generalized mass matrix in base frame.

TDOT: is the time derivative of the 21-D generalized twist.

WD: is a 21-D vector-function of the dissipative force.

C.17

EVATDD

This procedure is used to compute the angular accelerations of the three independent input links in terms of the given angular positions and velocities of the same links as well as the generalized driving force exerting on the three input links.

C.18

RUNKT

This procedure solves the first-order differential equations of motion for 6-dimensional state-variable vector [THETA',THDOT']' where ' stands for transpose. The integral technique used is the 4th order Runge-Kutta method with a given time step size fixed (input by user).

Appendix D

Peroformance Angle and Torque data

D.1

Output angle data

Time (Sec.)	First joint angle (Deg.)	Second joint angle (Deg.)	Third joint angle (Deg.)
0.000	90.0000	60.0000	120.0000
0.010	90.0000	60.0000	120.0000
0.020	90.0001	60.0001	119.9998
0.030	90.0002	60.0004	119.9992
0.040	90.0005	60.0009	119.9981
0.050	90.0009	60.0018	119.9963
0.060	90.0016	60.0032	119.9937
0.070	90.0025	60.0050	119.9900
0.080	90.0037	60.0075	119.9850
0.090	90.0053	60.0106	119.9787
0.100	90 0073	60.0146	119 9708
0.110	90.0097	60.0194	119.9612
0.120	90.0126	60.0252	119.9496
0.130	90.0160	60.0320	119.9360
0.140	90.0200	60.0399	119.9201
0.150	90.0246	60.0491	119.9018
0.160	90.0298	60.0596	119.8809
0.170	90.0357	60.0714	119.8572
0.180	90.0423	60.0847	119.8307
0.190	90.0497	60.0995	119.8010
0.200	90.0580	60.1160	119.7681
0.210	90.0671	60.1341	119.7318
0.220	90 0770	60.1540	119 6919
0.230	90.0879	60.1758	119.6483
0.240	90.0998	60.1996	119.6008
0.250	90.1127	60.2254	119.5493
0.260	90.1266	60.2532	119.4936
0.270	90.1416	60.2832	119.4335
0.280	90.1577	60.3155	119.3690
0.290	90.1750	60.3501	119.2998
0.300	90.1935	60.3871	119.2259
0.310	90.2133	60.4265	119.1470
0.320	90.2342	60.4685	119 0630
0.330	90.2565	60.5130	118.9739
0.340	90.2801	60.5603	118.8794
0.350	90.3051	60.6103	118.7795
0.360	90.3315	60.6631	118 6739
0.370	90.3594	60.7187	118 5626
0.380	90.3886	60.7773	118.4454
0.390	90.4194	60.8389	118.3223
0.400	90.4517	60.9035	118 1930
0.410	90.4856	60.9712	118.0575
0.420	90.5211	61.0421	117.9157
0.430	90.5581	61.1163	117.7674
0.440	90.5968	61.1937	117.6126
0.450	90.6372	61.2745	117.4511
0.460	90.6793	61.3586	117.2828
0.470	90.7231	61.4462	117 1076
0.480	90.7686	61.5373	116 9255
0.490	90.8159	61.6319	116.7363

Table D.1: Joint 1, 2 and 3 output angle data

Time (Sec.)	First joint angle (Deg.)	Second joint angle (Deg.)	Third joint angle (Deg.)
0.500	90.8650	61.7301	116.5399
0.510	90.9159	61.8319	116.3362
0.520	90.9687	61.9374	116.1252
0.530	91.0233	62.0466	115.9068
0.540	91.0798	62.1595	115.6809
0.550	91.1381	62.2763	115.4474
0.560	91.1984	62.3969	115.2063
0.570	91.2606	62.5213	114.9574
0.580	91.3248	62.6496	114.7008
0.590	91.3909	62.7819	114.4363
0.600	91.4590	62.9181	114.1638
0.610	91.5291	63.0583	113.8835
0.620	91.6012	63.2025	113.5951
0.630	91.6754	63.3507	113.2986
0.640	91.7515	63.5030	112.9940
0.650	91.8297	63.6594	112.6812
0.660	91.9099	63.8198	112.3603
0.670	91.9922	63.9844	112.0311
0.680	92.0766	64.1531	111.6937
0.690	92.1630	64.3260	111.3480
0.700	92.2515	64.5030	110.9940
0.710	92.3421	64.6842	110.6316
0.720	92.4348	64.8695	110.2609
0.730	92.5295	65.0591	109.8818
0.740	92.6264	65.2528	109.4944
0.750	92.7254	65.4507	109.0986
0.760	92.8264	65.6528	108.6944
0.770	92.9295	65.8591	108.2818
0.780	93.0348	66.0695	107.8609
0.790	93.1421	66.2842	107.4316
0.800	93.2515	66.5030	106.9940
0.810	93.3630	66.7260	106.5480
0.820	93.4766	66.9531	106.0937
0.830	93 5922	67.1844	105.6311
0.840	93.7099	67.4198	105.1603
0.850	93.8297	67.6594	104.6812
0.860	93.9515	67.9030	104.1940
0.870	94.0754	68.1507	103.6986
0.880	94.2012	68.4025	103.1951
0.890	94.3291	68.6583	102.6835
0.900	94.4590	68.9181	102.1638
0.910	94.5909	69.1819	101.6363
0.920	94.7248	69.4496	101.1008
0.930	94.8606	69.7213	100.5574
0.940	94.9984	69.9969	100.0063
0.950	95.1381	70.2763	99.4474
0.960	95.2798	70.5595	98.8809
0.970	95.4233	70.8466	98.3068
0.980	95.5687	71.1374	97.7252
0.990	95.7159	71.4319	97.1362

Table D.2: Joint 1, 2 and 3 output angle data (continued)

Time (Sec.)	First joint angle (Deg.)	Second joint angle (Deg.)	Third joint angle (Deg.)
1.000	95.8650	71.7301	96.5399
1.010	96.0159	72.0319	95.9363
1.020	96.1686	72.3373	95.3255
1.030	96.3231	72.6462	94.7076
1.040	96.4793	72.9586	94.0828
1.050	96.6372	73.2745	93.4511
1.060	96.7968	73.5937	92.8126
1.070	96.9581	73.9163	92.1674
1.080	97.1211	74.2421	91.5157
1.090	97.2856	74.5712	90.8575
1.100	97.4517	74.9035	90.1930
1.110	97.6194	75.2389	89.5223
1.120	97.7886	75.5773	88.8454
1.130	97.9594	75.9187	88.1626
1.140	98.1315	76.2631	87.4739
1.150	98.3051	76.6103	86.7795
1.160	98.4801	76.9603	86.0794
1.170	98.6565	77.3130	85.3739
1.180	98.8342	77.6685	84.6630
1.190	99.0133	78.0265	83.9470
1.200	99.1935	78.3871	83.2259
1.210	99.3750	78.7501	82.4998
1.220	99.5577	79.1155	81.7690
1.230	99.7416	79.4832	81.0335
1.240	99.9266	79.8532	80.2936
1.250	100.1127	80.2254	79.5493
1.260	100.2998	80.5996	78.8008
1.270	100.4879	80.9758	78.0483
1.280	100.6770	81.3540	77.2919
1.290	100.8671	81.7341	76.5318
1.300	101.0580	82.1160	75.7681
1.310	101.2497	82.4995	75 0010
1.320	101.4423	82.8847	74.2307
1.330	101.6357	83.2714	73.4572
1.340	101.8298	83.6596	72.6809
1.350	102.0246	84.0491	71.9018
1.360	102.2200	84.4399	71.1201
1.370	102.4160	84.8320	70.3360
1.380	102.6126	85.2252	69.5496
1.390	102.8097	85.6194	68.7612
1.400	103.0073	86.0146	67.9708
1.410	103.2053	86.4106	67.1787
1.420	103.4037	86.8075	66.3850
1.430	103.6025	87.2050	65.5900
1.440	103.8016	87.6032	64.7937
1.450	104.0009	88.0018	63.9963
1.460	104.2005	88.4009	63.1981
1.470	104.4002	88.8004	62.3992
1.480	104.6001	89.2001	61.5998
1.490	104.8000	89.6000	60.8000

Table D.3: Joint 1, 2 and 3 output angle data (continued)

Time (Sec.)	First joint angle (Deg.)	Second joint angle (Deg.)	Third joint angle (Deg.)
1.500	105.0000	90.0000	60.0000
1.510	105.2000	90.4000	59.2000
1.520	105.3999	90.7999	58.4002
1.530	105.5998	91.1996	57.6008
1.540	105.7995	91.5991	56.8019
1.550	105.9991	91.9982	56.0037
1.560	106.1984	92.3968	55.2063
1.570	106.3975	92.7950	54.4100
1.580	106.5963	93.1925	53.6150
1.590	106.7947	93.5894	52.8213
1.600	106.9927	93.9854	52.0292
1.610	107.1903	94.3806	51.2388
1.620	107.3874	94.7748	50.4504
1.630	107.5840	95.1680	49.6640
1.640	107.7800	95.5601	48.8799
1.650	107.9754	95.9509	48.0982
1.660	108.1702	96.3404	47.3191
1.670	108.3643	96.7286	46.5428
1.680	108.5577	97.1153	45.7693
1.690	108.7503	97.5005	44.9990
1.700	108.9420	97.8840	44.2319
1.710	109.1329	98.2659	43.4682
1.720	109.3230	98.6460	42.7081
1.730	109.5121	99.0242	41.9517
1.740	109.7002	99.4004	41.1992
1.750	109.8873	99.7746	40.4507
1.760	110.0734	100.1468	39.7064
1.770	110.2584	100.5168	38.9665
1.780	110.4423	100.8845	38.2310
1.790	110.6250	101.2499	37.5002
1.800	110.8065	101.6129	36.7741
1.810	110.9867	101.9735	36.0530
1.820	111.1658	102.3315	35.3370
1.830	111.3435	102.6870	34.6261
1.840	111.5199	103.0397	33.9206
1 850	111.6949	103.3897	33.2205
1.860	111.8685	103.7369	32.5261
1.870	112.0406	104.0813	31.8374
1.880	112.2114	104.4227	31.1546
1.890	112.3806	104.7611	30.4777
1.900	112.5483	105.0965	29.8070
1.910	112.7144	105.4288	29.1425
1.920	112.8789	105.7579	28.4843
1.930	113.0419	106.0837	27.8326
1.940	113.2032	106.4063	27.1874
1.950	113.3628	106.7255	26.5489
1.960	113.5207	107.0414	25.9172
1.970	113.6769	107.3538	25.2924
1.980	113.8314	107.6627	24.6745
1.990	113.9841	107.9681	24.0637

Table D.4: Joint 1, 2 and 3 output angle data (continued)

Time (Sec.)	First joint angle (Deg.)	Second joint angle (Deg.)	Third joint angle (Deg.)
2.000	114.1350	108.2699	23.4601
2.010	114.2841	108.5681	22.8638
2.020	114.4313	108.8626	22.2748
2.030	114.5767	109.1534	21.6932
2.040	114.7202	109.4405	21.1191
2.050	114.8619	109.7237	20.5526
2.060	115.0016	110.0031	19.9937
2.070	115.1394	110.2787	19.4426
2.080	115.2752	110.5504	18.8992
2.090	115.4091	110.8181	18.3637
2.100	115.5410	111.0819	17.8362
2.110	115.6709	111.3417	17.3165
2.120	115.7988	111.5975	16 8049
2.130	115.9246	111.8493	16.3014
2.140	116.0485	112.0970	15.8060
2.150	116.1703	112.3406	15 3188
2.160	116.2901	112.5802	14 8397
2.170	116.4078	112.8156	14.3689
2.180	116.5234	113.0469	13.9063
2.190	116.6370	113.2740	13.4520
2.200	116.7485	113.4970	13.0060
2.210	116.8579	113.7158	12.5684
2.220	116.9652	113.9305	12.1391
2.230	117.0705	114.1409	11.7182
2.240	117.1736	114.3472	11.3056
2.250	117.2746	114.5493	10.9014
2.260	117.3736	114.7472	10.5056
2.270	117.4705	114.9409	10.1182
2.280	117.5652	115.1305	9.7391
2.290	117.6579	115.3158	9.3684
2.300	117.7485	115.4970	9.0060
2.310	117.8370	115.6740	8.6520
2.320	117.9234	115.8469	8.3063
2.330	118.0078	116.0156	7.9689
2.340	118.0901	116.1802	7.6397
2.350	118.1703	116.3406	7.3188
2.360	118.2485	116.4970	7.0060
2.370	118.3246	116.6493	6 7014
2.380	118.3988	116.7975	6.4049
2.390	118.4709	116.9417	6.1165
2.400	118.5410	117.0819	5.8362
2.410	118.6091	117.2181	5.5637
2.420	118.6752	117.3504	5.2992
2.430	118.7394	117.4787	5.0426
2.440	118.8016	117.6031	4.7937
2.450	118.8619	117.7237	4.5526
2.460	118.9202	117.8405	4.3191
2.470	118.9767	117.9534	4.0932
2.480	119.0313	118.0626	3 8748
2.490	119.0841	118.1681	3.6638

Table D.5: Joint 1, 2 and 3 output angle data (continued)

Time (Sec.)	First joint angle (Deg.)	Second joint angle (Deg.)	Third joint angle (Deg.)
2.500	119.1350	118.2699	3.4601
2.510	119.1841	118.3681	3.2637
2.520	119.2314	118.4627	3.0745
2.530	119.2769	118.5538	2.8924
2.540	119.3207	118.6414	2.7172
2.550	119.3628	118.7255	2.5489
2.560	119.4032	118.8063	2.3874
2.570	119.4419	118.8837	2.2326
2.580	119.4789	118.9579	2.0843
2.590	119.5144	119.0288	1.9425
2.600	119.5483	119.0965	1.8070
2.610	119.5806	119.1611	1.6777
2.620	119.6114	119.2227	1.5546
2.630	119.6406	119.2813	1.4374
2.640	119.6685	119.3369	1.3261
2.650	119.6949	119.3897	1.2205
2.660	119.7199	119.4397	1.1206
2.670	119.7435	119.4870	1.0261
2.680	119.7658	119.5315	0.9370
2.690	119.7867	119.5735	0.8530
2.700	119.8065	119.6129	0.7741
2.710	119.8250	119.6499	0.7002
2.720	119.8423	119.6845	0.6310
2.730	119.8584	119.7168	0.5665
2.740	119.8734	119.7468	0.5064
2.750	119.8873	119.7746	0.4507
2.760	119.9002	119.8004	0.3992
2.770	119.9121	119.8242	0.3517
2.780	119.9230	119.8460	0.3081
2.790	119.9329	119.8659	0.2682
2.800	119.9420	119.8840	0.2319
2.810	119.9503	119.9005	0.1990
2.820	119.9577	119.9153	0.1693
2.830	119.9643	119.9286	0.1428
2.840	119.9702	119.9404	0.1191
2.850	119.9754	119.9509	0.0982
2.860	119.9800	119.9601	0.0799
2.870	119.9840	119.9680	0.0640
2.880	119.9874	119.9748	0.0504
2.890	119.9903	119.9806	0.0388
2.900	119.9927	119.9854	0.0292
2.910	119.9947	119.9894	0.0213
2.920	119.9963	119.9925	0.0150
2.930	119.9975	119.9950	0.0100
2.940	119.9984	119.9968	0.0063
2.950	119.9991	119.9982	0.0037
2.960	119.9995	119.9991	0.0019
2.970	119.9998	119.9996	0.0008
2.980	119.9999	119.9999	0.0002
2.990	120.0000	120.0000	0.0000
3.000	120.0000	120.0000	0.0000

Table D.6: Joint 1, 2 and 3 output angle data (continued)

D.2

Output torque data

Time (Sec.)	First joint torque (Nm)	Second joint torque (Nm)	Third joint torque (Nm)
0.000	-3.9739	-37.8192	41.8956
0.010	-3.9729	-37.7904	41.8531
0.020	-3.9719	-37.7616	41.8106
0.030	-3.9710	-37.7329	41.7682
0.040	-3.9701	-37.7044	41.7261
0.050	-3.9692	-37.6762	41.6842
0.060	-3.9684	-37.6482	41.6426
0.070	-3.9677	-37.6206	41.6014
0.080	-3.9670	-37.5933	41.5607
0.090	-3.9665	-37.5665	41.5205
0.100	-3.9660	-37.5402	41.4809
0.110	-3.9657	-37.5144	41.4420
0.120	-3.9655	-37.4892	41.4037
0.130	-3.9653	-37.4647	41.3663
0.140	-3.9654	-37.4409	41.3296
0.150	-3.9655	-37.4179	41.2938
0.160	-3.9658	-37.3957	41.2589
0.170	-3.9663	-37.3743	41.2251
0.180	-3.9669	-37.3539	41.1922
0.190	-3.9676	-37.3344	41.1605
0.200	-3.9685	-37.3159	41.1299
0.210	-3.9696	-37.2984	41.1005
0.220	-3.9709	-37.2821	41.0723
0.230	-3.9723	-37.2669	41.0454
0.240	-3.9739	-37.2529	41.0199
0.250	-3.9756	-37.2401	40.9957
0.260	-3.9776	-37.2285	40.9730
0.270	-3.9797	-37.2183	40.9517
0.280	-3.9821	-37.2094	40.9318
0.290	-3.9846	-37.2019	40.9136
0.300	-3.9873	-37.1959	40.8968
0.310	-3.9902	-37.1913	40.8817
0.320	-3.9933	-37.1881	40.8682
0.330	-3.9966	-37.1865	40.8564
0.340	-4.0001	-37.1865	40.8463
0.350	-4.0038	-37.1880	40.8378
0.360	-4.0078	-37.1911	40.8311
0.370	-4.0119	-37.1959	40.8262
0.380	-4.0163	-37.2023	40.8230
0.390	-4.0209	-37.2104	40.8216
0.400	-4.0258	-37.2202	40.8221
0.410	-4.0309	-37.2317	40.8243
0.420	-4.0362	-37.2449	40.8284
0.430	-4.0419	-37.2599	40.8343
0.440	-4.0477	-37.2766	40.8421
0.450	-4.0539	-37.2951	40.8517
0.460	-4.0604	-37.3154	40.8631
0.470	-4.0671	-37.3375	40.8764
0.480	-4.0742	-37.3613	40.8915
0.490	-4.0816	-37.3870	40.9085

Table D.7: Joint 1, 2 and 3 output torque data

Time (Sec.)	First joint torque (Nm)	Second joint torque (Nm)	Third joint torque (Nm)
0.500	-4.0894	-37.4144	40.9273
0.510	-4.0976	-37.4436	40.9479
0.520	-4.1061	-37.4746	40.9703
0.530	-4.1150	-37.5074	40.9945
0.540	-4.1244	-37.5420	41.0205
0.550	-4.1342	-37.5783	41.0482
0.560	-4.1445	-37.6164	41.0776
0.570	-4.1553	-37.6561	41.1087
0.580	-4.1666	-37.6976	41.1415
0.590	-4.1785	-37.7408	41.1759
0.600	-4.1910	-37.7857	41.2118
0.610	-4.2042	-37.8322	41.2494
0.620	-4.2180	-37.8803	41.2884
0.630	-4.2325	-37.9300	41 3290
0.640	-4.2478	-37.9813	41.3709
0.650	-4.2639	-38.0341	41.4143
0.660	-4.2808	-38.0884	41.4590
0.670	-4.2987	-38.1441	41.5049
0.680	-4.3175	-38.2012	41.5521
0.690	-4.3373	-38.2597	41.6004
0.700	-4.3581	-38.3195	41 6499
0.710	-4 3801	-38.3806	41.7004
0.720	-4.4033	-38.4429	41.7518
0.730	-4.4278	-38.5063	41.8042
0.740	-4.4536	-38.5709	41.8575
0.750	-4.4808	-38.6365	41.9115
0.760	-4.5094	-38.7031	41.9662
0.770	-4.5397	-38.7706	42.0215
0.780	-4.5715	-38.8391	42.0774
0.790	-4.6051	-38.9083	42.1338
0.800	-4.6406	-38.9783	42.1906
0.810	-4.6779	-39.0489	42.2477
0.820	-4.7172	-39.1201	42.3050
0.830	-4.7586	-39.1919	42.3625
0.840	-4.8023	-39.2641	42.4200
0.850	-4.8482	-39.3367	42.4775
0.860	-4.8965	-39.4096	42 5349
0.870	-4.9474	-39.4827	42.5921
0.880	-5.0008	-39.5559	42.6491
0.890	-5.0570	-39.6292	42.7056
0.900	-5.1161	-39.7025	42.7617
0.910	-5.1781	-39.7756	42.8172
0.920	-5.2432	-39.8486	42.8720
0.930	-5.3115	-39.9212	42.9261
0.940	-5.3830	-39.9935	42.9794
0.950	-5.4581	-40.0653	43.0317
0.960	-5.5367	-40.1365	43.0830
0.970	-5.6189	-40.2071	43.1332
0.980	-5.7050	-40.2769	43.1821
0.990	-5.7949	-40.3459	43.2297

Table D.8: Joint 1, 2 and 3 output torque data (continued)

Time (Sec.)	First joint torque (Nm)	Second joint torque (Nm)	Third joint torque (Nm)
1.000	-5.8889	-40.4139	43.2758
1.010	-5.9871	-40.4809	43.3205
1.020	-6.0895	-40.5467	43.3635
1.030	-6.1963	-40.6113	43.4047
1.040	-6.3076	-40.6745	43.4441
1.050	-6.4235	-40.7362	43.4816
1.060	-6.5441	-40.7964	43.5169
1.070	-6.6695	-40.8550	43.5501
1.080	-6.7998	-40.9118	43.5810
1.090	-6.9352	-40.9667	43.6094
1.100	-7.0757	-41.0196	43.6353
1.110	-7.2213	-41.0705	43.6584
1.120	-7.3723	-41.1191	43.6787
1.130	-7.5286	-41.1655	43.6960
1.140	-7.6903	-41.2094	43.7102
1.150	-7.8574	-41.2508	43.7209
1.160	-8.0301	-41.2896	43.7282
1.170	-8.2084	-41.3257	43.7318
1.180	-8.3923	-41.3589	43.7314
1.190	-8.5819	-41.3891	43.7268
1.200	-8.7771	-41.4163	43.7178
1.210	-8.9780	-41.4403	43.7042
1.220	-9.1846	-41.4610	43.6855
1.230	-9.3969	-41.4784	43.6616
1.240	-9.6148	-41.4922	43.6320
1.250	-9.8383	-41.5025	43.5965
1.260	-10.0675	-41.5090	43.5545
1.270	-10.3021	-41.5117	43.5057
1.280	-10.5423	-41.5105	43.4496
1.290	-10.7878	-41.5053	43.3857
1.300	-11.0386	-41.4959	43.3134
1.310	-11.2947	-41.4824	43.2321
1.320	-11.5558	-41.4645	43.1412
1.330	-11.8219	-41.4422	43.0401
1.340	-12.0929	-41.4153	42.9280
1.350	-12.3685	-41.3838	42.8040
1.360	-12.6486	-41.3476	42.6675
1.370	-12.9330	-41.3066	42.5174
1.380	-13.2216	-41.2606	42.3529
1.390	-13.5141	-41.2096	42.1730
1.400	-13.8103	-41.1534	41.9766
1.410	-14.1100	-41.0919	41.7627
1.420	-14.4129	-41.0251	41.5301
1.430	-14.7187	-40.9527	41.2777
1.440	-15.0273	-40.8747	41.0042
1.450	-15.3383	-40.7910	40.7084
1.460	-15.6514	-40.7013	40.3891
1.470	-15.9663	-40.6055	40.0449
1.480	-16.2827	-40.5035	39.6745
1.490	-16.6002	-40.3951	39.2767

Table D.9: Joint 1, 2 and 3 output torque data (continued)

Time (Sec.)	First joint torque (Nm)	Second joint torque (Nm)	Third joint torque (Nm)
1.500	-16.9187	-40.2801	38.8501
1.510	-17.2376	-40.1583	38.3935
1.520	-17.5567	-40.0296	37.9055
1.530	-17.8757	-39.8937	37.3849
1.540	-18.1941	-39.7504	36 8307
1.550	-18.5116	-39.5995	36 2416
1.560	-18.8279	-39.4408	35.6166
1.570	-19.1427	-39.2740	34.9549
1.580	-19.4555	-39.0990	34.2555
1.590	-19.7661	-38.9155	33.5177
1.600	-20.0742	-38.7232	32.7409
1.610	-20.3794	-38.5221	31.9247
1.620	-20.6814	-38.3117	31.0688
1.630	-20.9800	-38.0921	30.1729
1.640	-21.2748	-37.8630	29.2370
1.650	-21.5657	-37.6242	28.2614
1.660	-21.8524	-37.3757	27.2463
1.670	-22.1346	-37.1173	26.1923
1.680	-22.4123	-36.8489	25.1000
1.690	-22.6852	-36.5705	23.9703
1.700	-22.9533	-36.2822	22.8042
1.710	-23.2163	-35.9838	21.6028
1.720	-23.4742	-35.6756	20.3676
1.730	-23.7270	-35.3576	19.1000
1.740	-23.9746	-35.0299	17.8016
1.750	-24.2169	-34.6927	16.4741
1.760	-24.4541	-34.3463	15.1195
1.770	-24.6862	-33.9908	13.7396
1.780	-24.9132	-33.6267	12.3365
1.790	-25.1352	-33.2543	10.9124
1.800	-25.3523	-32.8738	9.4693
1.810	-25.5646	-32.4857	8.0096
1.820	-25.7724	-32.0905	6.5353
1.830	-25.9758	-31.6886	5.0489
1.840	-26.1749	-31.2804	3.5524
1.850	-26.3700	-30.8666	2.0482
1.860	-26.5613	-30.4475	0.5385
1.870	-26.7490	-30.0237	-0.9747
1.880	-26.9334	-29.5958	-2.4891
1.890	-27.1146	-29.1643	-4.0028
1.900	-27.2931	-28.7298	-5.5137
1.910	-27.4690	-28.2927	-7.0199
1.920	-27.6425	-27.8537	-8.5197
1.930	-27.8140	-27.4133	-10.0112
1.940	-27.9838	-26.9721	-11.4929
1.950	-28.1520	-26.5305	-12.9631
1.960	-28.3190	-26.0890	-14.4204
1.970	-28.4851	-25.6483	-15.8636
1.980	-28.6504	-25.2086	-17 2912
1.990	-28.8152	-24.7706	-18.7021

Table D.10: Joint 1, 2 and 3 output torque data (continued)

Time (Sec.)	First joint torque (Nm)	Second joint torque (Nm)	Third joint torque (Nm)
2.000	-28.9798	-24.3346	-20.0953
2.010	-29.1443	-23.9011	-21.4699
2.020	-29.3091	-23.4705	-22.8248
2.030	-29.4743	-23.0431	-24.1594
2.040	-29.6402	-22.6193	-25.4730
2.050	-29.8068	-22.1995	-26.7649
2.060	-29.9745	-21.7839	-28.0346
2.070	-30.1433	-21.3728	-29.2817
2.080	-30.3133	-20.9666	-30.5058
2.090	-30.4848	-20.5654	-31.7066
2.100	-30.6578	-20.1694	-32.8839
2.110	-30.8325	-19.7790	-34.0374
2.120	-31.0089	-19.3941	-35.1672
2.130	-31.1871	-19 0151	-36.2730
2.140	-31.3671	-18.6420	-37.3550
2.150	-31.5490	-18.2750	-38.4131
2.160	-31.7329	-17.9142	-39.4475
2.170	-31.9187	-17.5597	-40.4583
2.180	-32.1064	-17.2114	-41.4456
2.190	-32.2960	-16.8696	-42.4097
2.200	-32.4876	-16.5342	-43.3508
2.210	-32.6810	-16.2053	-44.2691
2.220	-32.8762	-15.8829	-45.1650
2.230	-33.0731	-15.5669	-46.0387
2.240	-33.2717	-15.2575	-46.8906
2.250	-33.4718	-14.9545	-47.7211
2.260	-33.6735	-14.6580	-48.5304
2.270	-33.8765	-14.3679	-49.3190
2.280	-34.0808	-14.0842	-50.0873
2.290	-34.2862	-13.8069	-50.8357
2.300	-34.4927	-13.5358	-51.5645
2.310	-34.7000	-13.2710	-52.2742
2.320	-34.9080	-13.0123	-52.9652
2.330	-35.1166	-12.7598	-53.6379
2.340	-35.3256	-12.5132	-54.2928
2.350	-35.5350	-12.2727	-54.9302
2.360	-35.7444	-12.0380	-55.5505
2.370	-35.9537	-11.8091	-56.1542
2.380	-36.1628	-11.5860	-56.7417
2.390	-36.3716	-11.3685	-57.3133
2.400	-36.5797	-11.1565	-57.8695
2.410	-36.7872	-10.9500	-58.4107
2.420	-36.9937	-10.7489	-58.9371
2.430	-37.1991	-10.5531	-59.4493
2.440	-37.4032	-10.3625	-59.9475
2.450	-37.6060	-10.1770	-60.4321
2.460	-37.8072	-9.9966	-60.9035
2.470	-38.0066	-9.8211	-61.3619
2.480	-38.2041	-9.6504	-61.8078
2.490	-38.3995	-9.4845	-62.2413

Table D.11: Joint 1, 2 and 3 output torque data (continued)

Time (Sec.)	First joint torque (Nm)	Second joint torque (Nm)	Third joint torque (Nm)
2.500	-38.5927	-9.3233	-62.6629
2.510	-38.7835	-9.1666	-63 0728
2.520	-38.9718	-9.0145	-63.4713
2.530	-39.1575	-8.8668	-63.8587
2.540	-39.3404	-8.7234	-64.2351
2.550	-39.5204	-8.5843	-64.6010
2.560	-39.6973	-8.4493	-64.9565
2.570	-39.8712	-8.3184	-65.3019
2.580	-40.0417	-8.1916	-65.6374
2.590	-40.2090	-8.0686	-65 9632
2.600	-40.3728	-7.9495	-66.2796
2.610	-40.5331	-7.8342	-66.5867
2.620	-40.6898	-7.7225	-66.8848
2.630	-40.8429	-7.6145	-67.1740
2.640	-40.9923	-7.5100	-67.4545
2.650	-41.1380	-7.4089	-67.7266
2.660	-41.2798	-7.3112	-67.9905
2.670	-41.4179	-7.2168	-68.2462
2.680	-41.5521	-7.1257	-68.4940
2.690	-41.6825	-7.0376	-68.7341
2.700	-41.8090	-6.9527	-68.9666
2.710	-41.9317	-6.8707	-69.1917
2.720	-42.0506	-6.7917	-69 4096
2.730	-42.1656	-6.7155	-69 6205
2.740	-42.2769	-6.6420	-69.8245
2.750	-42.3845	-6.5712	-70.0219
2.760	-42.4884	-6.5030	-70.2128
2.770	-42.5888	-6.4374	-70.3973
2.780	-42.6855	-6.3741	-70.5758
2.790	-42.7789	-6.3132	-70.7483
2.800	-42.8688	-6.2545	-70.9151
2.810	-42.9555	-6.1980	-71.0764
2.820	-43.0389	-6.1436	-71.2323
2.830	-43.1194	-6.0912	-71.3832
2.840	-43.1968	-6.0407	-71.5291
2.850	-43.2715	-5.9920	-71.6704
2.860	-43.3434	-5.9449	-71.8073
2.870	-43.4128	-5.8995	-71.9400
2.880	-43.4797	-5.8556	-72.0687
2.890	-43.5444	-5.8131	-72.1938
2.900	-43.6070	-5.7719	-72.3154
2.910	-43.6677	-5.7318	-72.4338
2.920	-43.7266	-5.6929	-72.5494
2.930	-43.7838	-5.6549	-72.6624
2.940	-43.8397	-5.6177	-72.7731
2.950	-43.8944	-5.5814	-72.8818
2.960	-43.9480	-5.5456	-72.9889
2.970	-44.0008	-5.5103	-73.0945
2.980	-44.0529	-5.4754	-73.1991
2.990	-44.1046	-5.4408	-73.3031
3.000	-44.1561	-5.4062	-73.4066

Table D.12: Joint 1, 2 and 3 output torque data (continued)