National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

The Development and Analysis of an Integrated Workcell Controller for Multiple-Robot Synchronisation

Suresh Kabra

A Thesis

in

The Department

of

Mechanical Engineering

Presented in partial fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

July 1992

ISBN 0-315-80951-5

Abstract

The Development and Analysis of an Integrated Workcell Controller for Multiple-Robot Synchronisation

Suresh Kabra

This thesis describes the development of an integrated workcell controller at the Centre for Industrial Control, Concordia University. The controller integrates, conceptually and physically, the control functions required in a multi-device robotic workcell, at a central location. An analysis of the need for open architecture, where new devices can be integrated with minimal disturbance, leads to the use of parallel processing architecture at all levels. The motion device itself may range from a simple conveyor belt to a complex robotic manipulator. As a typical embodiment of this concept, a multi-robot workcell consisting of two Puma industrial robots has been developed. A discrete time model for minimum-time synchronisation between the two robots is developed and implemented. Using this model, the controller guides a tool manipulator to intercept and subsequently synchronise with a workpiece being carried by another robotic arm. Synchronisation is achieved even under on-line path determination and speed variations of the workpiece arm, while ensuring that the acceleration limits of the tool arm are not violated. A real-time efficient algorithm for robotic path planning has also been developed. The technique uses cubic spline formulation to construct smooth paths in task space from a few look ahead knot points. Using arc length parameterisation, the robot end effector can be guided along the constructed path at a regulated speed and with controlled

orientation with respect to the arc being traversed. The performance of the integrated controller, implemented on a network of Transputers, is measured and factors effecting it are analyzed.

Acknowledgements

The author dedicates this thesis to Sri Thakur Raman Bihari and looks forward for his guidance in all future endeavour.

The author expresses sincere gratitude to the thesis supervisor, Prof. R.M.H.Cheng for his invaluable guidance and consistent encouragement through out the course of this research. The author is indebted to Mr. Simon Poon and Dr. Ramesh Rajagopalan for the numerous discussions he had with them on several issues related to this work.

The author expresses his thanks to Mr. Gilles Huard, whose enthusiasm to build and make things work played a significant role in the development of this project. The author would like to thank all the colleagues and staff of the Centre for Industrial Control for making his brief stay at the Centre, a memorable one.

My regards to my parents and other members of the family for their continued support and encouragement all through these years.

Financial assistance for this work was provided through several NSERC grants awarded to Prof. Cheng.

Suresh Kabra

Table of Contents

List of Figures

List of Tables

List of Acronyms

CIC	Centre for Industrial Control.
IWC	Integrated Workcell Controller.
CONCIC	Concordia University Centre for Industrial Control.
IDFS	Interrupt Driven File Server.
SISD	Single Instruction Single Data.
SIMD	Single Instruction Multiple Data.
MIMD	Multiple Instruction Multiple Data.
MISD	Multiple Instructions Single Data.
TRAM	Transputer Module.
GTPC	Global Task Planning Controller.
CPC	Cartesian Path Controller.
JSC	Joint Space Controller.
GMS	Global Message Switch.
EE	End Effector.
PID	Proportional Integral Derivative.

Nomenclature

t	time in seconds.
T_c	Robot controller sampling period.
T_s	Servo controller sampling period.
k	The k^{th} sampling interval of robot controller.
$\mathbf{r}[k]$	End Effector position vector in cartesian space in k^{th} interval.
$\mathbf{q}[k]$	Vector of joint angles in the k^{th} interval.
$\dot{\boldsymbol{r}}[k],\dot{\boldsymbol{q}}[k]$	rate of change of vectors $\mathbf{r}$ and $\mathbf{q}$ respectively.
$e(n)$	Position error in servo loop in n^{th} servo sampling period.
$u(n)$	Motor drive signal from servo controller for sample period n.
n'	Derivative controller sampling interval.
$e(n')$	Position error for derivative control.
K_p,K_i,K_d	PID controller gains for the servo loop.
k_p,k_i,k_d	PID gains as used in the LM629 motion controller.
${}^{d}\theta_k,{}^{a}\theta_k$	Desired and Actual joint angle in the k^{th} interval.
${}^{d}\delta\theta_k,{}^{a}\delta\theta_k$	Desired and Actual increment in θ during the k^{th} interval.
${}^{d}\delta e_k,{}^{a}\delta e_k$	Desired and Actual rotation of a joint in k^{th} interval in terms of encoder counts.
${}^{d}\dot{e}_k,{}^{a}\dot{e}_k$	Desired and actual joint velocity in the k^{th} interval in terms of counts/T_s.
$\mathbf{e}$	Joint position vector in terms of encoder counts.
$\mathbf{R}$	encoder count/radian matrix for the manipulator.

X_o,Y_o,Z_o	Base frame of the manipulator.
X_i,Y_i,Z_i	D-H frame attached on joint #i of the manipulator, where i=1,2,...,6.
τ	Free parameter used in interpolation.
$g(\tau)$	Unknown function in τ being interpolated.
g_i	Value of g at $\tau=\tau_i$.
$f(\tau)$	Piecewise cubic interpolant of g.
i	Subscript for ordering knot points.
$\mathbf{K}_i$	The i^{th} knot point in cartesian space.
τ_i	Value of τ associated with $\mathbf{K}_i$ knot point.
$\tilde{\tau}_k$	Value of τ associated with k^{th} sampling period.
n	The number of look ahead knot points used in spline.
$[\tau_i,\tau_{i+1}]$	The interval $\tau_i \leq \tau \leq \tau_{i+1}$.
$\Delta\tau_i$	Increment in τ i.e. $\tau_{i+1} - \tau_i$.
$\Delta\tau$	1/(n-1).
P_i	A cubic polynomial in τ over interval $[\tau_i,\tau_{i+1}]$.
$\wp_4$	Linear space of cubic polynomials.
$C^{(1)}[a,b]$	Space of continuous functions with continuous derivative over [a,b].
u_i	A free parameter in cubic spline.
r	A coordinate of the task (cartesian) space.
r_i	Value of r at the i^{th} knot point i.e. $\tau = \tau_i$.
$[\tau_i,\tau_{i+1}]r$	The divided difference of r a τ_i,τ_{i+1}.
$r(\tau)$	Cubic spline interpolant for coordinate r over $r_i,...,r_{i+n}$ points.

$r_i(\tau)$	Cubic polynomial segment $r(\tau)$ between $[\tau_i,\tau_{i+1}]$.
q	The number of cubic segments travelled over $r(\tau)$.
Γ_i	3D arc corresponding to $(x_i(\tau),y_i(\tau),z_i(\tau))$.
s	arc length.
δs	Increment in arc length.
$s_i(\tau)$	Arc length function of Γ_i.
$\wp_i(\tau)$	Fourth order polynomial used in $s_i(\tau)$ evaluation.
$W'(\tau)$	Remainder polynomial in $\sqrt{\wp_i(\tau)}$ evaluation.
m	Controller sampling interval when EE moves from Γ_{i-1} to Γ_i.
S[k]	Desired manipulator cartesian speed for k^{th} interval.
$\Delta\ell_k$	Incremental length to be covered by manipulator in k^{th} interval.
$\sigma_{k,i}$	Set point loaded in k^{th} interval and lying on Γ_i arc.
Δ_i	The uncovered length along last segment travelled Γ_{i-1}.
jj	The last interval, counting from m, travelled on Γ_i.
x_o,y_o,z_o	Start position for elliptical ellipse and polar arc.
R_x,R_y	Ellipse radii along X_o and Y_o axis respectively.
$X_W[i],Y_W[i],Z_W[i]$	Incremental displacements of workpiece.
P_z	Pitch of the arc travelled along Z axis.
ξ	Free angle used to generate elliptical/polar arcs.
R_p	Maximum radial distance for the polar arc.
[u p b]	Tangent, principal normal and binormal vectors in a Trihedron.
[n a s]	normal, approach and slide vectors for EE orientation.

Ψ_i	Parametric (vector) form of arc Γ_i.
$\kappa(s)$	Curvature of an arc as a function of arc length.
t_o	Time when trigger for synchronisation is recieved.
$\mathbf{P}_W[k]$	Commanded Workpiece position vector at time $t = kT_c + t_o$.
$\mathbf{P}_{WA}[k]$	Actual Workpiece position vector at time $t = kT_c + t_o$.
$\mathbf{P}_T[k]$	Commanded Tool position vector at time $t = kT_c + t_o$.
$\mathbf{P}_{TA}[k]$	Actual Tool position vector at time $t = kT_c + t_o$.
$\mathbf{V}_W[k], S_W[k]$	Commanded Workpiece velocity/speed at time $t = kT_c + t_o$.
$\mathbf{V}_T[k], S_T[k]$	Commanded Tool velocity/speed at time $t = kT_c + t_o$.
$\mathbf{P}_{WT}[l,j]$	$\mathbf{P}_W[l] - \mathbf{P}_T[j]$.
$S_{W,min}$	Minimum speed of workpiece manipulator in synchronisation model.
$S_{T,min}$	Minimum speed of tool manipulator in synchronisation model.
$\delta l_W[k], \delta l_T[k]$	Incremental distance travelled by workpiece and tool in the k^{th} period.
$\alpha[k], \alpha$	Ratio of tool speed with respect to workpiece in interception phase.
p	Maximum number of intervals allowed for interception phase.
pp	Interval in which synchronisation starts.
ϕ_c	Critical angle for tool manipulator.
$\delta_W\phi[j]$	Change in workpiece direction from one period to next.
$\phi[j]$	Angle between velocity vectors of tool and workpiece manipulators.
$\Omega[j]$	Vector to change direction of tool manipulator in synchronisation.
$[\hat{\mathbf{u}}\ \mathbf{v}\ \mathbf{w}]$	A coordinate frame used in two manipulator synchronisation.
$\mathbf{R}^U_{560}$	Transformation matrix between Puma 560 base and Universal frame.

Chapter 1

Introduction

With a pressing need for increased productivity and uniform quality, the industry can no longer employ inefficient manual assembly techniques and then use subjective inspection criterion to sort for quality. Today, the approach one has to take is to *do it right the first time* by fully utilising the immense power of the sophisticated computer based manufacturing tools available. Earlier this was achieved through *fixed-automation systems*, consisting of special purpose machines designed to perform predetermined functions. However, the inflexibility and the associated high costs of the fixed automation has led to its gradual replacements by industrial robots in the recent years. As industrial robots are flexible as well as repeatable in their operations, their use has become a dominant factor in the productivity, quality, profitability and indeed the competitive strength of an industry.

An industrial robot, as defined by the Robot Institute of America, is a *reprogrammable* multi-functional manipulator designed to move materials, parts, tools or specialised devices, through variable programmed motions for the performance of a variety of tasks. For this, the robot must possess *intelligence*, which is due to the computer algorithms associated with its control and sensing systems. In the studies of robots, a clear distinction must be made between the science of robotics and robotics engineering. While the goal of robotics science is to understand the algorithmic processes underlying the actions of perception and manipulation, robotics engineering deals with

design, construction, control and applications of the robotic machines. This thesis falls in the domain of the latter.

In the conventional industrial set up, a manipulator is associated with a *workcell*, which includes fixtures for part handling and orientation determination, to carry out a definable task. However, *fixtures* obviously contradicts *flexibility*. Even a minor change in operations may lead to extensive retooling and fixture redesign. The natural solution, one which permits maximum freedom in operations, is to use additional manipulators instead of fixtures. The *coordinated control of multiple robots* with overlapping workspace not only reduces the hard automation substantially, but also increases the parallelism in the operations thus increasing productivity.

In general, the multi-manipulator systems can be investigated under the following categories :

(i) Multi-device Controller : Issues related to architecture, design and implementation.

(ii) Collision free path planning for synchronised operations.

(iii) Dynamic modelling and control.

(iv) Workspace Analysis.

(v) Vision and sensing.

This thesis concentrates on the first two issues.

1.1 Objectives and Scope

A multi-manipulator workcell may contain several types of mechanical devices such as industrial robots, conveyor belts, indexing/rotary tables and sensory devices like camera, force sensors etc. The approach of using the dedicated device controllers, as

supplied by the manufacturer, is not necessarily a good solution for the resulting workcell has a distributed non-homogeneous architecture with different hardware schemes and programming environments.

An alternate to the sprawling design is an integrated workcell controller which integrates, conceptually and physically, all the control functions required for complete operations in a multi-device workcell. With such an integrated controller, all the higher level control processors and associated software would reside in a central host while the manipulators and the lower level servo controllers may be placed at convenient locations on the shop floor in a distributed fashion. Communication between the two is established through high speed data pathways.

The primary objective of this thesis is a **detailed analysis of the issues involved in the architecture of such an integrated workcell controller and a typical implementation on a network of processors.** Using parallelism in hardware and software, the developed workcell controller offers a uniform structure through all levels with upgradable computing power. It has a flexible, modular and open architecture which allows one to incorporate new devices into the workcell with minimal disturbance to the on-going operation.

Coordinated control of manipulators can not be achieved without collision free path planning, which constitutes a second objective of the research conducted. Towards this purpose a technique is proposed (and tested) for **on-line path planning in task space for continuous path applications of an industrial robot.** Using this formulation, a manipulator can be made to travel at a regulated speed along smooth cartesian paths

constructed on-line from a few look ahead points. The technique is extended to include end effector orientation control along the 3D arc traced in the cartesian space.

As a typical illustration of the integrated controller application in a multi-robot industrial workcell the path planning algorithm is used in conjunction with a discrete-time synchronisation algorithm to achieve **complete coordination between two industrial robots under the supervision of the integrated controller.** This is the third and the final objective of this thesis. The spatial model used for coordination ensures complete synchronisation between the two robots in a predictable time even under on-line variations in path and speed of the manipulator carrying the workpiece. Further, the technique is extended to generate a smooth path for the tool manipulator, the one attempting to synchronise, such as to minimise the torque demands placed on the joint servos.

At the current stage of development, the integrated workcell controller at the Centre for Industrial Control, Concordia University, uses a cluster of four processors for coordinated control of a Puma 560 and a Puma 260 industrial robots. In the experimental set up, the Puma 260 moves the workpiece along a helical path generated on-line. The speed of travel can be regulated by the user from a dial provided for this purpose. When initiated by the user, by activating an appropriate switch, the Puma 560 moves along a collision free path to synchronise with the workpiece. The speed of travel of the workpiece may be varied while the synchronisation is being attempted.

1.2 Thesis Outline

Chapter 2 discusses several issues of importance in the implementation of the integrated controller such as architecture, processing element etc. Chapter 3 gives a

detailed description of the integrated workcell controller (IWC) at all levels of hierarchy. Chapter 4 describes the implementation of a basic arm controller in the IWC framework. Chapter 5 gives the analysis and experimental results of the path planning algorithm used. The synchronisation model used for two manipulator coordination is presented in Chapter 6. Finally the conclusions of the research conducted and recommendations for future work are presented in Chapter 7.

Chapter 2

Robotic Workcell : An Overview

2.1 Introduction

Any manufacturing operation involves considerable repetitive work which can be delegated to industrial robots. Earlier, it was done through manual operation or the more recent and conventional approach of *fixed automation*. Manual operations, although flexible, lacks consistency thereby leading to low quality. Fixed automation gives the desired consistency but lacks the flexibility. Industrial robots provide the desired solution as they are predictable and repeatable in that they do their assigned task every time. This consistency leads to high quality of output. Also, flexibility in manufacturing is achieved by the reprogrammability of the robot system.

Theoretically, a single robot can perform all assembly operations. However, due to production time constraints and reliability, several robots are usually deployed sequentially, breaking the work load in sub-assembly modules and stages. Each robot is dedicated to a single assembly task leading to the concept of a *robotic workcell.*

2.2 Concept and Applications of a Robotic Workcell

A workcell has been defined by Cheng [2.1][1] as

> *A workcell is a unit of flexible manufacturing environment with a definable task, to which is assigned a specific group of equipment to carry out the task.*

It can, therefore, be treated as an island of automated process, whether the nature of the

[1]Numbers in square brackets represent reference number.

process be machining, fabrication, assembly or packaging. A *robotic workcell* is a workcell utilising one or more industrial robot to execute the assigned task. Work can be transferred between workcells manually or via conveyor belts, transfer lines or using mobile robots such as Automated Guided Vehicles (AGV). Such factory organisation is typical of many process oriented industries such as steel making, moulding and IC manufacturing etc. Flexibility with in a workcell becomes vital as each workcell has to handle many types of parts and process variables are to be controlled in a flexible range.

Recent literature is full of such *single robot* workcell examples. Earlier, Davis et.al. [2.2] of Ford Motor Company used a robotic workcell for linear welding of car bumpers. Bloom [2.3] of IBM used a Puma 560 based workcell for diskette writing operation. More recently, Graf [2.4] of 3M has discussed the implementation of robotic workcell for deburring and finishing applications. The use of a flexible robotic workcell for assembling airframe components has been discussed by Olsen [2.5]. Cheng et.al. [2.6] demonstrated the use of a Puma 560 based workcell for adaptive synchronisation with a conveyor belt for pick and place and assembly operations using camera vision.

However, the tooling and fixtures required in such single robot workcells are not only an overhead on the system cost, but also consume considerable design and engineering effort. One approach is to use additional manipulators instead of fixtures. In such a case manipulators have to be coordinated so that they utilise the other manipulators capabilities, share sensory feedback data and lead to desired part orientation.

The coordination of manipulators with overlapping work spaces not only reduces the hard automation but also allows greater parallelism in operations [2.7]. As an

	Single Manipulator	Coordinated Manipulators	
		Robot #1	Robot #2
1.	Pick A	Pick A	Pick B
2.	Place A in Fixture	Place B in A	Wait for Robot #1
3.	Pick B	-	Place AB in Pallet
4.	Place B in A		
5.	Pick AB		
6.	Place AB in pallet		

example, consider the typical steps required for assembly operations in single and two robot workcells. It can be seen that the total number of steps are reduced from six to five and the sequential number of steps to only three, thus allowing higher parallelism in operations leading to higher productivity.

Several new issues appear when one moves from single robot to multi-robot coordination and control. Apart from system modelling as a closed chain, issues such as sharing of sensory data, collision avoidance, task allocation and planning etc. become of increasing importance. Recent advances from the fields such as image processing, pattern recognition, optimal and adaptive control, force control and artificial intelligence have to be incorporated in the *real-time* operation of the workcell controller. This is a challenging task as many of the advanced algorithms in these fields are computationally too demanding for real time implementation. This also demands the use of *increased parallelism* or *specialised architecture for specialised computations* in the workcell controller design.

Due to the aforementioned reasons, the multi-robot coordination is still in the

research phase with a few reported industrial set-ups. Fisher [2.8] of Dupont describes the use of dual forearm manipulator for nuclear material handling applications. Pfister [2.9] of Babcock and Wilcox has discussed the use of three manipulators for workpiece transfer through a series of inspection stations. Also, the US National Bureau of Standards [2.10] has developed a deburring station consisting of multi robots with AGVs and a number of ancillary devices all controlled by a multi-tasking controller. Agapakis et. al. [2.11] of Automatix Inc. have also discussed the coordination of multiple robotic devices (up to a total of 12 axis).

2.3 Review of Workcell Controller Architectures

Two approaches exist to design a workcell controller capable of coordinating multiple robotic devices. One approach is to use a multi-tasking supervisory control, which in turn controls the motion controller of individual arm (supplied by the robotic manufacturer) through serial or parallel communication port. This is the approach used by Zheng et. al. [2.12] for coordinated control of two Puma robots. In their controller, a VAX-11/750 computer is used to supervise two Puma arm controllers. The arm controller, supplied by the manufacturer consists of a LSI-11 computer and a cluster of microprocessors modules. The resulting software architecture requires the knowledge of three different operating systems i.e. VMS for VAX-11/750, VAL-II for LSI-11 controller and REKCOR, an overriding operating system designed by the authors.

This approach of using arm controllers as supplied by the robotic manufacturer is at best a matter of convenience and expedience and not necessarily a logical solution. This is because most of the robots today are still designed for individual operations. The

resulting architecture of the controller lacks the flexibility of integration with other robotic systems or sensory devices. Some of the drawbacks associated with the currently available robot controllers are :

(i) **Non-Homogeneous Architecture** : Each commercially available manipulator arm comes with its associated controller wl ich has a unique hardware and software architecture (and not necessarily a very advanced one). The controller software is usually conservative in design and often prevents the use of the robotic device to the fullest. The lack of uniformity in hardware architecture and diversity in control algorithms used for p^'h planning and execution (which are closely guarded secrets) makes the task of controlling them from a *central control* in a cooperative manner practically infeasible.

(ii) **Limited communication bandwidth** : Most of the currently available controllers have very limited external communication capabilities. The severe limitation in the bandwidth of the channel (which is generally a RS232 port with a maximum of 19.2K baud rate) increases the real-time response time to beyond the acceptable limits. When working with multiple manipulators, a fast response time is of critical importance in interference zone detection and collision avoidance.

(iii) **Diversity in Programming Languages** : Each robotic controller has its unique programming language e.g. VALII developed by Unimation for Puma series, RAIL by Automatix Inc., AML in IBM products and KAREL by GMFanuc., to name a few. Each language has its unique commands and data structures to specify motion and store cartesian and joint space configurations [2.13]. This

diversity in programming environment poses another challenge to the control engineer who is attempting to work with a mixed family of industrial robots.

Instead of sprawling controller design, a better approach is to design a compact and efficient controller for multi-device workcells where a processor or processor cluster is dedicated to the local control of a separate robot (motion control, start-up and shutdown procedures etc.). This concept of an *integrated workcell controller* is discussed in greater details in the following section. However, to replace an arm controller by a processor cluster requires some investigation on the architecture of a robot controller.

As mentioned before, the real-time operation of a robotic system demands the use of increased parallelism or special architecture in a robot controller. Poon [2.14] has done an exhaustive survey of common controller structures and categorised them in the following three categories :

(i) multi-tasking uniprocessor.

(ii) special processors like DSP, ASICs, Vector and Array processors.

(iii) general purpose multiprocessors

The earliest approach of multi-tasking uniprocessor [2.15] is not common any more. Although sufficient for purely kinematic control of a simple manipulator, a single processor lacks the computing power required to implement more complex control schemes such as computed torque control etc. in real-time. As a result, a number of custom made processors have been designed to provide enhanced real-time computing power. A VLSI Robotic Processor chip has been designed by Olson et. al. [2.16] for Jacobian computation. A single chip VLSI implementation for the direct kinematic

solution has been proposed by Leung et. al. [2.17] to achieve an estimated update time of 10 microseconds. Lanthrop [2.18] has proposed a systolic array architecture for solving Newton-Euler equations for manipulator dynamics. DSP based controller architecture has been reported by Taknashi et. al. [2.19]. Unfortunately, these special architectures rely heavily on the regularity of algorithms to achieve the maximum performance. Hence, Graham [2.20] suggests a low impact of special architectures on robotic applications with the majority of cases handled by general purpose architectures. This is very true in the current stage with a great diversity in robot models available in the market.

The multiprocessor approach, which provides a general purpose architecture, remains to be the most popular. This concept has already been implemented in large systems for a number of years. Poon [2.14] has compared several multiprocessor system on the basis of a performance index ($/mip). The NYMPH multiprocessor system [2.21] used for control of a Stanford Arm with three fingers, is a typical example. It consists of three layers, namely a VAX connected to a Sun station (through an ethernet link), which in turn controls a set of floating point coprocessors through multi-bus.

The upgradable computing power of multiprocessor environment allows its usage in multiple manipulator control. Agapakis et. al [2.11] have used a multiprocessor computer system incorporating several MC68000 CPUs sharing a common bus for coordinated control of multiple robotic devices. Kali [2.22]-[2.23], a low level hardware and software environment for multiple coordinated machines is distributed over multiple processors in a VME bus environment. At the higher level, Kali uses a Robot Control C Library (RCCL) [2.24] for developing control programs in a Unix environment. The

library concept of a robot control language for multiple device synchronisation is also discussed by [2.25]. The parallel processing techniques used in such multiprocessor architectures will be taken up in further details in section 2.6.

More recently, a hierarchical neurocontroller architecture has been suggested for manipulator control [2.26]-[2.27]. The capability of a neural network to learn and perform massively parallel processing results in an enhanced robotic control system capable of adapting to environment changes. Also, the implementation of a holonic manipulator, i.e. a manipulator characterised by autonomous controllers built in its mechanical structure, has been investigated [2.28]. High reliability and drastic decrease in wiring are some advantages of such an architecture. The design of a micro-robot holonic manipulator, which may have 100 to 1000 CPUs in its structure, will add new dimensions to the field of robotics.

2.4 Objectives and Requirements of an Integrated Workcell

The need for a integrated controller for coordinated control of multiple robotic devices was discussed in the last section. In this section, this concept is developed further and given a firm footing by identifying the goals and design criterion which should be met by such a controller.

2.4.1 Goals

For multiple devices coordination, Cheng [2.1] introduced the concept of an **Integrated Workcell Controller (IWC)**, whose objectives are

To have the capabilities required to control and coordinate multiple industrial robots and other motion devices to share an overlapping workspace and perform

useful synchronised operations on a common workpiece in an industrial environment.

It is proposed that the control functions of the several robotic manipulators and other mechanical devices that make up a given workcell be integrated conceptually and physically within one single location. The hierarchical structure of this integrated controller consists of a host processor plus a number of resident processors or processor clusters, all sharing the same user interface and working under the same software environment. Each cluster of processor is designated to specific type of robot. A control software, tailored for the particular robot, resides in each cluster and communicates with other such clusters using a *common protocol.* This leads to a *flexible* and *open* architecture of IWC where one can add new kinds of devices such as different types of robots, sensors etc. and incorporate them into the system by writing corresponding *device drivers*, with minimal disturbance to the existing system.

This concept of the *device drivers* is wide spread in the commonly used professional software packages like AUTOCAD etc., where it is used to interface such standard devices as printers, plotters etc. If it is desired to change a device, all that needs to be done is to instruct the control software to install the suitable driver software. In a similar manner, in the framework of IWC, the host computer provides the control software with drivers for robotic devices installed as and when required for the application at hand. However, this analogy between a driver for computer peripheral and a driver for a robotic manipulator can not be extended further. The driver for a manipulator control, which requires joint servo control with sensor feedback and a

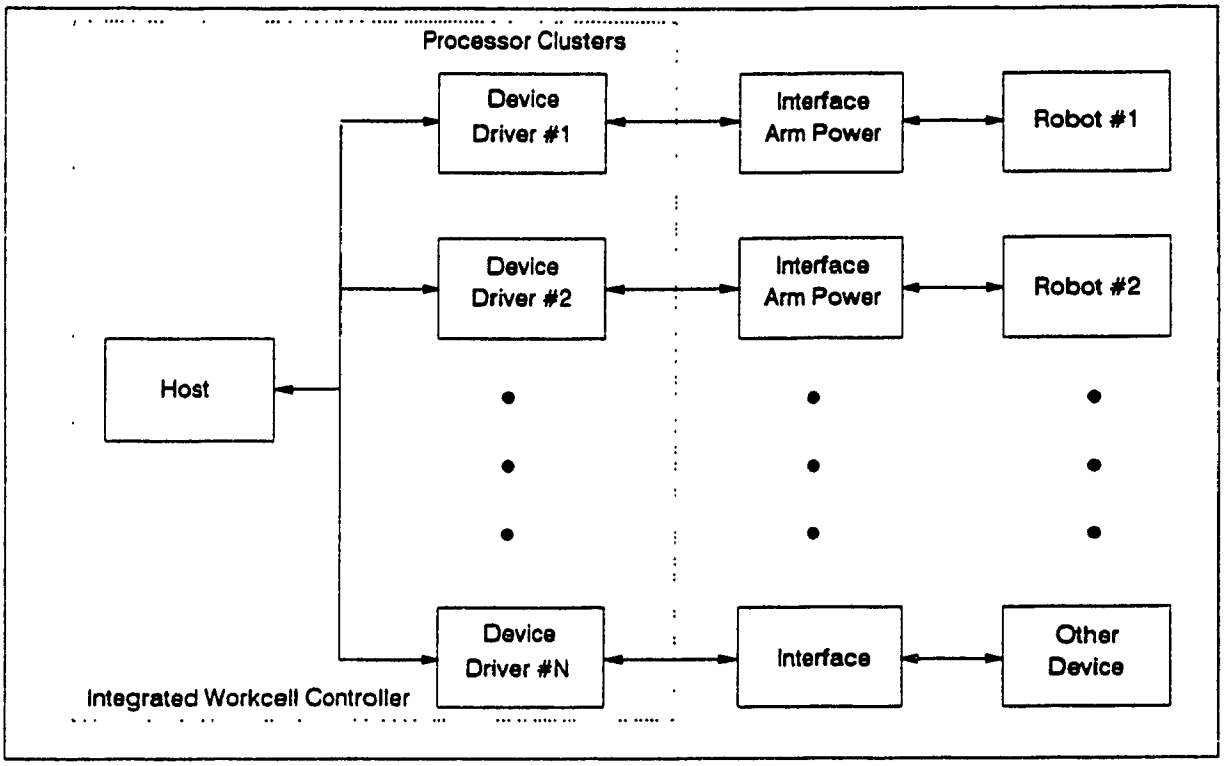

Figure 2.1 Integrated Workcell Controller architecture

sophisticated software based on arm kinematics or dynamics, is much more complicated than a driver for a regular peripheral.

The resulting architecture of the workcell is shown in Figure 2.1 in its most general form. The cluster of processors for each robotic device thus replaces a conventional robot controller. Several such clusters can be housed within the host, which is placed at a central location and exchange information with each other using well defined protocol. The actual manipulators, with associated hardware like servo controllers, amplifiers and power supplies etc. can be located in the vicinity of the workpiece. Shielded cables with high noise immunity are used to connect the two. This leads to an *exceedingly compact* integrated controller as compared to the sprawling interconnections discussed before.

2.4.2 Design Criterion

In order for the workcell to meet the stated objectives, its architecture should satisfy certain design criterion as outlined earlier by Simon [2.14]. These are

(i) **High precision floating point operations** : The controller must have the capability to perform floating point operations with high accuracy. This is essential as any robot control algorithm, using kinematics or dynamics, requires such operations as evaluation of transcendental functions and matrix inversion calculations. A low accuracy of floating point operations will lead to error accumulation culminating in a deviation from the desired path. 32 bit is the minimum acceptable floating point datawidth, although 64 bit is preferable.

(ii) **Upgradable computing power** : More and more complex algorithms for robot control, incorporating image processing, sensor feedback, adaptive control etc., are being developed. The implementation of these for *real time* operation will place an ever increasing demand on the computational power of the controller. Hence, it is essential that the controller has an *extensible* architecture which allows one to upgrade the computing power in an easy manner. It is this capability of the IWC which shall determine its worth for future research and applications.

(iii) **Modularity** : For easy management, it is required that the system functions in IWC be distributed over modules with well defined external interfaces. In fact, the concept of device drivers discussed earlier, is possible only if the controller has a modular architecture. High reliability can be obtained by using an *active* and *standby* pair for critical modules. System can be reconfigured easily by

adding/replacing the appropriate modules.

(iv) **Sensor interface** : The importance of sensor feedback in intelligent control of robotic manipulator is clearly demonstrated by the operation of the human arm. Even a simple task of holding a pencil between tips of the two index fingers requires vision, touch and force sensing. Thus, a straightforward way to interface sensors becomes an essential criterion of the IWC architecture. Fur ., it is desirable that such an interface be commercially available for more commonly used sensors such as camera vision to reduce the development time.

(v) **Uniformity** : The uniformity in hardware and software is required for the ease of installation and maintenance in the field. This must be assured by using same family of processing elements and related peripherals in all modules. Also, the programming language used and the software environment should be the same at all levels.

(vi) **Software environment** : It should be possible for the user to develop new applications in a high level language such as C. Further, appropriate tools for debugging support should be available to aid in software development.

As a consequence of the above listed criterion, parallelism in hardware and software becomes an essential feature of the controller structure. Only an architecture possessing *parallel processing capabilities at all levels*, from user task execution to the parallel control of the joint servos of a particular manipulator, can satisfy the stated objective.

From the above listed design criterion, one can draw the following list of features which must be available in the processor used in the IWC.

(i) The processor should be capable of operating in true hardware parallelism with respect to one another and with respect to the host.

(ii) The processor should have multi-tasking capabilities to allow parallelism in software.

(iii) The processor should have floating point capabilities preferably on-chip or through an available math coprocessor which can be interfaced.

(iv) The processors should be capable of communicating with one another and also with the host, whenever required.

(v) They must be sufficiently compact in physical dimensions so as to be accommodated within the host.

(vi) They should have reasonably low cost.

(vii) High level language support should be available for programming.

To conclude this section, the advantages offered by the IWC concept, earlier mentioned by Cheng [2.1], are summarised below :

(i) It offers a tidy system and economy in space.

(ii) It offers uniformity in interface and software design, avoiding differences in communication protocols between robots.

(iii) It affords modularity in design once sufficient device clusters or drivers are available for different robot types.

(iv) The architecture speeds up inter-device communication.

(v) It offers system expendability and flexibility, and complete system control by the user.

(vi) It offers ease of maintenance and diagnostics.

(vii) As a research and development tool, this design facilitates easy implementation of future robotic control methodology by simply changing the control algorithm in the cluster software.

2.5 Development Background of the Integrated Workcell Controller

The first configuration of the CONCIC (*CON*cordia University *C*entre for *I*ndustrial *C*ontrol) Workcell was developed in 1986 [2.29]. It consisted of an IBM-PC connected to a Puma motion controller, provided by the vendor, via two serial links. Using a stationary binary camera, the CONCIC workcell scans a workpiece travelling on a conveyor belt. Using an efficient image processing algorithm [2.30]-[2.31], the IBM-PC computes the position and orientation of the workpiece and charts out an appropriate path for the robot end effector to move upstream and meet the travelling workpiece in the correct orientation. The PC would then direct the actual robot motion by supplying the new desired position to the Puma arm controller every 28msec, through a serial link. The performance of this configuration was satisfactory for simple pick and place operations. However, much of the code was written in assembler and the system required considerable effort in maintenance and enhancement.

Optimised path planning was targeted for the second phase by incorporating microprocessors which are able to function in parallel with the host computer. Using a MC68000 based PC68K board, occupying one slot of the PC bus, most of the control software was redeveloped using the C language. The result was an upgraded system performance, with the robotic path planning being optimised into two straight line

segments taking minimum time for rendezvous [2.6].

The third phase of the CONCIC workcell development lead to further optimisation in path planning using elliptical segments. The entire path between the starting position of the robot and its rendezvous with the workpiece was split into one elliptical segment and one parabolic segment [2.14]. This leads to lesser demands on the motor acceleration leading to less stringent motor rating at the specification stage of the robot. As additional enhancement, a data acquisition system was added to bring out the joint encoder feedback signals and improve the accuracy of the workpiece rendezvous. The system block diagram at this stage is shown in Figure 2.2.

The hardware architecture also went through a radical change in this phase. As the supplier of the PC68K board never delivered the promised multi-copy of the MC68000 processor within the same PC bus, an alternate parallel processing architecture had to be adopted. For reasons discussed in section 2.8, Inmos transputer T800 was selected as the new processing element. This processor is discussed in greater details in section 2.8.1. Thus, at this stage, the CONCIC workcell had four processing elements, as shown in Figure 2.2, with functions as follows :

(i) IBM-PC : Primary and incremental image analysis.

(ii) Root Transputer : Cartesian path feedback using the joint counter feedback circuitry.

(iii) Slave Transputer : Spatial path planning for the Puma manipulator.

(iv) Puma Robot Controller : Actual control of the Puma 560 arm.

To provide true parallelism between the host PC and the transputer, an

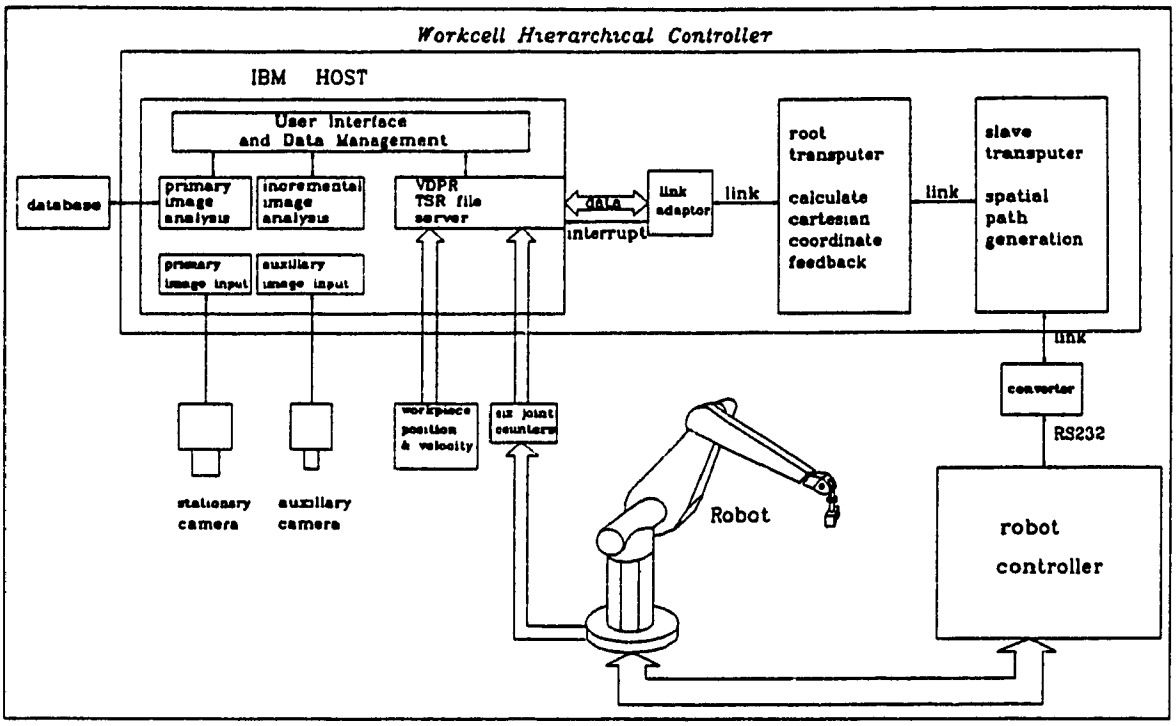

Figure 2.2 Task Distribution in Multiprocessor Workcell

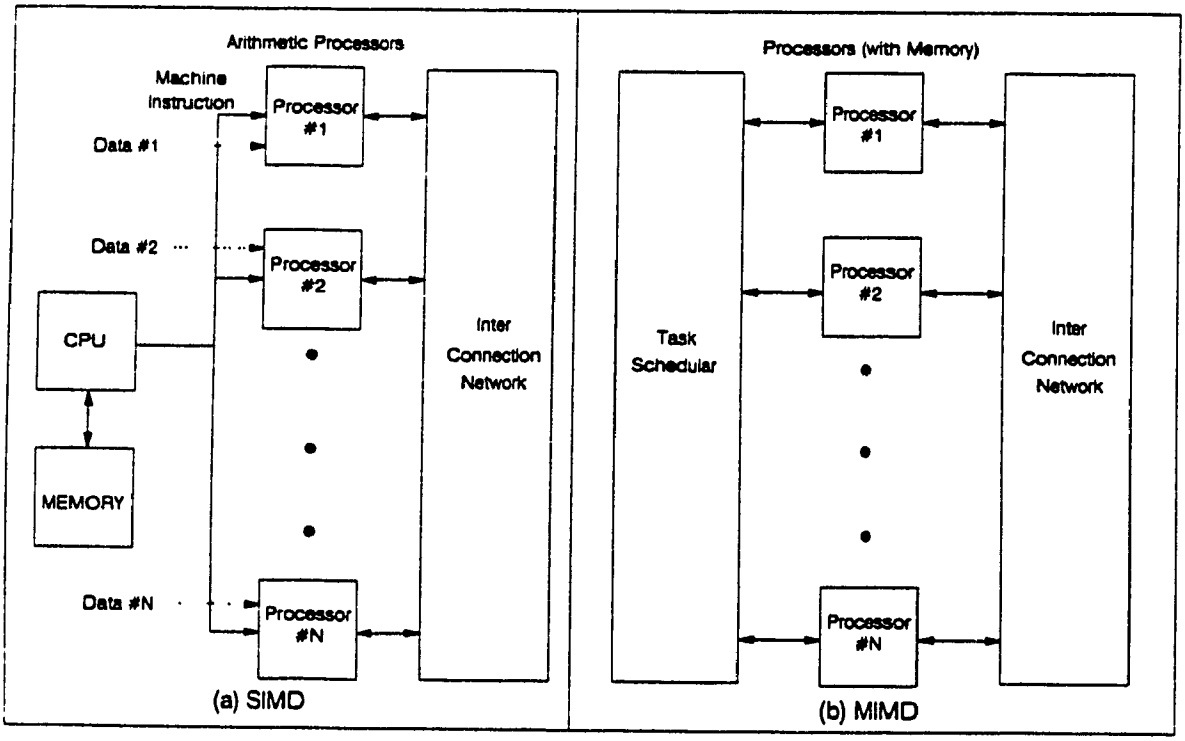

Figure 2.3 SIMD and MIMD architectures

interrupt driven file server (IDFS) was also implemented [2.32]. The IDFS takes the form of a resident program in DOS, which is *waken up* by the transputer request. After carrying out its tasks to provide the necessary peripheral functions, the file server returns to its *sleeping state* allowing the host program to resume execution of its control functions.

At this stage it was realised that limitations imposed by the vendor supplied robotic controller, highlighted earlier in section 2.3, were too severe to allow the extension of the CONCIC workcell for multi-arm coordinated control. Hence, it was imperative to abandon the Puma arm controller and to develop indigenous one where the *transputer was in direct control of the joint servos.* Such a controller would also provide an excellent test bed to implement and test different manipulator control schemes. As a first step in this direction, a parallel interface for the transputer serial link was developed by Poon and Huard [2.33] and upgraded by author for better performance.

At the current stage, the transputer based robot controller, developed *in-house* by the author, has been used to interface two Puma robot arms to the workcell. Using the transputer technology, an integrated workcell controller has been implemented and successfully used to perform *coordinated control of multiple robots.* The integrated controller satisfies the design criterions discussed before. The issues related to this phase of development are discussed in greater details in the subsequent chapters of this thesis.

2.6 Parallel Processing Architectures : An overview

As was indicated in section 2.4, parallel architecture is the best way to achieve the needed computational power required in the Integrated Workcell Controller. This section provides a brief review of the architectural alternatives available towards this purpose.

There are several ways to classify the many types of parallel computing systems proposed and available to the user. The most widely used classification, based on parallelism in ›oth instructions and data streams [2.20], is as follows :

(i) SISD : Single Instruction Stream/Single Data Stream

This is the conventional Von Neuman sequential uniprocessor architecture and is not parallel at all.

(ii) SIMD : Single Instruction Stream/Multiple Data Stream

Figure 2.3(a), on page 21, shows an SIMD architecture where a central processor broadcasts instructions to the arithmetic processors, which execute in parallel on separate data items. SIMD architecture is most suitable for vector and matrix operations, where there is a built in parallelism.

(iii) MIMD : Multiple Instruction/Multiple Data

In this configuration, as shown in Figure 2.3(b), each processor fetches and executes its own instructions. As each processor may execute a different program on the associated data set, special hardware and software is required to ensure coordination between them. This type of parallel architecture is appropriate when there is not enough inherent concurrency in the algorithm but where programs can be decomposed into independent sections, which can be executed in parallel and do not require frequent data exchange between each other.

(iv) MISD : Multiple Instruction/Single Data

The MISD is the least common architecture with not many applications. A possible use is simultaneous use of a set of algorithms to a common data set for

pattern recognition.

However, just the presence of multiple processor in any of the discussed configuration is not enough to ensure *concurrent processing* by these processors. One can have a multiprocessor system running a purely sequential algorithm. In such a case, the system performance will actually degrade when compared to a uniprocessor system as most of the time a processor will be waiting for some other processor to finish the previous step.

A typical computation consists of an algorithm applied on a large dataset, called as the *domain*, running on its *member* processing elements [2.34]. Concurrent processing is characterised by the following features :

(i) The problem domain can be decomposed into parts or *grains*, which can be executed in parallel. Each grain is assigned to one member.

(ii) Each member may run a conventional (sequential) algorithm. However, there are two key differences. First each member operates only on a part of domain and not on full domain. Secondly, the boundary conditions in the algorithm require communication with other members.

(iii) The speed-up is dependent on the *grain-size* and the communication overhead incurred for data exchange between grains.

The grain size can be used as another criterion for parallel architecture classification. The *small grain size* machines have memories upwards of about 1000 bytes per member. An example is the Connection Machine CM-1 which consists of 65536 very small nodes, each having a one-bit arithmetic unit and 4K bits of storage. As it is not possible to store any significant program at each node in such a mode, these machines are

used for SIMD applications. In contrast, *large grain size* machines provide at least 100K bytes of memory at each node and are used in MIMD applications. An example is the CRAY-XMP supercomputer which has only four identical processing unit sharing a common memory.

In addition to the grain size, the *topology* of the interconnection between the members also plays a critical role. As mentioned earlier, the time spent in interprocess communication is a principal factor determining the net speed up. Some of the popular network architectures are shown in Figure 2.4. The *bus architecture*, shown in Figure 2.4(a) on the following page, with *shared global memory* forms one end of the spectrum requiring as less as 200 ns to transfer a byte between processors. Although *complete interconnectivity* can also be achieved in this scheme, the resulting design is *tightly coupled* with following disadvantages :

(i) To avoid bus contention, arbitration logic for memory access has to be implemented in hardware. The increasing design complexity reduces the flexibility for system expansion.

(ii) Multiprocessor bussing imposes severe board layout problems, high tracking densities, multilayer PCBs and corresponding high costs.

(iii) Any alleviation of the hardware and tracking complexities results in an hierarchical system involving master/slave dependencies. This reduces the options for task assignment.

(iv) The user is responsible for code to map the programs on the appropriate processor. This leads to a substantial software overhead.

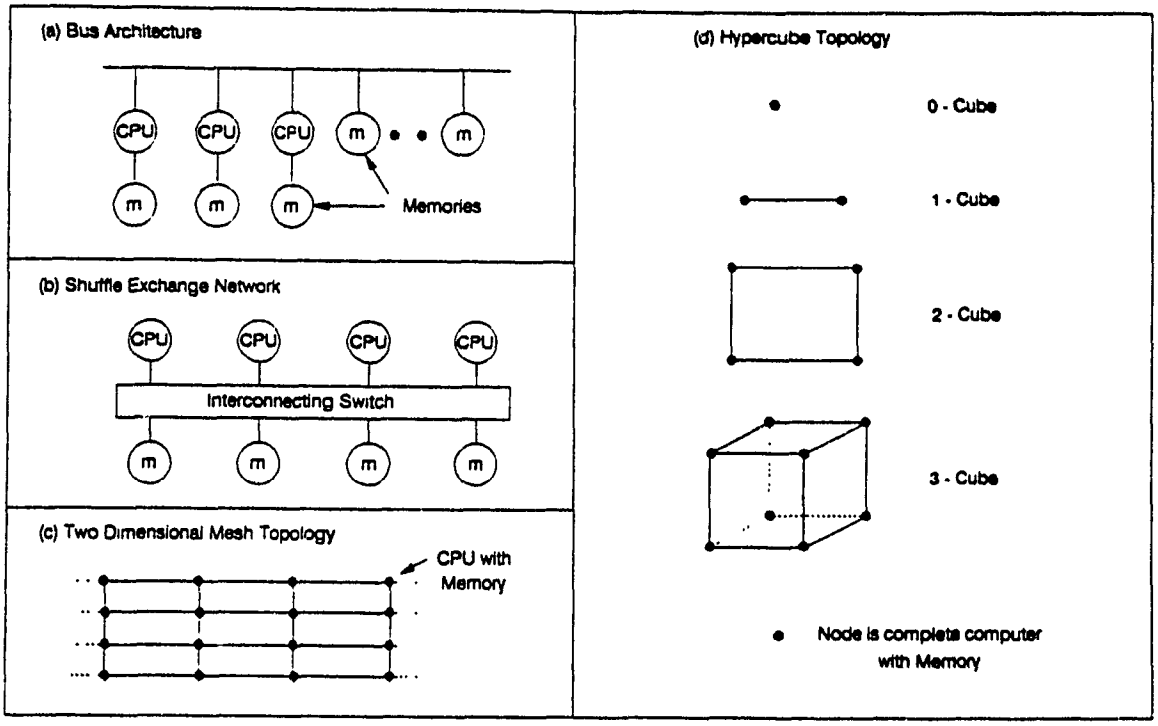

Figure 2.4 Four popular network topology for concurrent computers

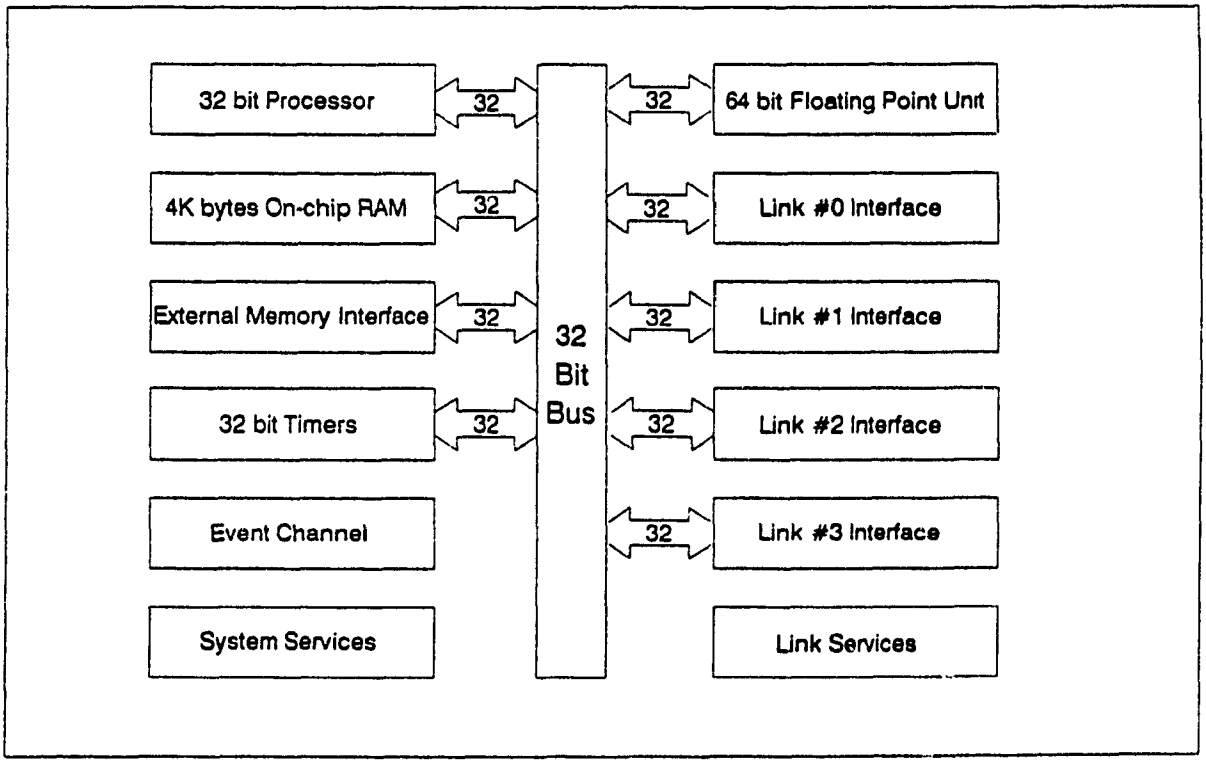

Figure 2.5 T-800 Transputer block diagram

A VME multiprocessor system is an example of a bus oriented parallel architecture. Instead of using a bus, shared memory can also be implemented using a switch interconnecting processing units to memory units as shown in Figure 2.4(b). However, in such a shuffle exchange the interprocessor communication time increases in proportion to log N, where N is the number of nodes [2.34].

As an alternate, one can use the *distributed memory* architectures as shown in Fig 2.4 (c)-(d). In pure distributed memory machines, each member includes local memory to the exclusion of shared global memory. These machines use *message passing* model for concurrent computations. However, the full interconnectivity of shared memory systems may not be available. The messages can only be directly exchanged between pairs of nodes sharing an hardware connection. Other connections are established using intermediate nodes for forwarding the message.

A final feature in the parallel computers classification is the nature of the node itself. Although a *general purpose* processor is used as a node commonly, one may build nodes optimised for specific applications such as digital signal processing, image processing etc. One can also have a MIMD node within a MIMD architecture.

2.7 IWC : A Large-Grain MIMD Distributed Memory Application

After considerable analysis of the IWC design specifications, it was concluded that the task of the multiple manipulator controller falls into a MIMD structure at all levels. This is so because at the higher level of multiple arm synchronisation, parallel processes run for manipulator path planning requiring limited inter-process communication for position/velocity feedback of participating arms. At the lower level of servo control for

a manipulator, the kinematics or inverse dynamics algorithm propagate in a particular direction, i.e. from manipulator base to the end effector or vice versa, thus excluding SIMD schemes. As the program complexity is substantial at all the levels, a large grain MIMD architecture is desirable for the IWC.

Further, the full interconnectivity between processors, which may be the only reason to adopt shared memory scheme, is not essential for IWC applications. This is so at global level because one expects only neighbouring robotic devices, with overlapping workspace, to coordinate. Thus, direct connection between adjacent processors suffices. At the lower level of manipulator control the control algorithm, as typified by the parallel Newton-Euler algorithm for inverse dynamics [2.35], requires communication between processors for adjacent joints only.

Although a general purpose node suffices for IWC applications, nodes optimised for image processing and DSP will provide easy sensor interface. To summarize, one can conclude that the task of multiple device synchronisation requires a **large grain MIMD with distributed memory** architecture of the integrated controller built from a general purpose processing element. Having identified broadly the nature of the IWC architecture, we look for a suitable processor element, the desired characteristics of which have already been outlined in section 2.4.2, to implement the same.

2.8 Transputer : An effective element for parallel architectures

The INMOS **Transputer** [2.36] is a microcontroller with its own local memory and with links for connecting one transputer to another transputer. A transputer can be used in a single processor system or in a network to build high performance concurrent

systems. In a network, the transputers communicate through the links resulting in a point to point communication.

The transputer architecture *defines a family* of programmable VLSI components. A typical member of the family is a single chip containing processor, memory and four communication links. In addition, each transputer product contains special circuitry and interfaces such as floating point unit (FPU), graphics processor etc., adapting it to a particular use. We discuss one of the device of this family T800 transputer in greater details.

2.8.1 IMS T800 Transputer

The T800 transputer [2.36] is an appropriate processor for the general purpose node in the IWC architecture. Figure 2.5 shows a block diagram of the T800. It has a 32 bit CPU with an 64 bit FPU on-chip. The FPU runs concurrently with the CPU, giving a sustained rate of 2.2 million floating point operations per second (MFlops), at a clock speed of 20 MHz. There is 4Kbytes on-chip RAM for high speed processing. Four serial links are available for communication with other processors. At a link speed of 20Mbits/sec, a bidirectional data transfer rate of 2.4 Mbytes/sec per link is achieved. The CPU can address a linear address space of 4Gbytes.

The instruction set of the transputer is simple and efficient. About 70% of executed instructions are encoded in a single byte giving a peak rate of 20 million instructions per second (MIPS). The T800 has *multi-tasking* capabilities. The software model of the processor consists of several processes running in parallel. Typically, a process is a sequence of instructions which starts, performs a number of operations and

is either suspended for reasons mentioned below or terminates to completion [2.37]. The T800 supports two levels of priority. Priority 1 (NON URGENT) processes are executed whenever there are no Priority 0 (URGENT) processes ready to run.

The processor has a microcoded scheduler which enables any number of processes to share the processor time. **This removes the need for software kernel**. At any time, a process may be :

ACTIVE - Being Executed

- On a waiting list to be executed.

INACTIVE - Waiting for an input.

- Waiting for an output.

- Waiting until a specified time.

The schedular operates in such a way that the **inactive process do not consume any processor time**. The URGENT process scheduling is non-preemptive i.e. the process will continue to execute till in the active state. In contrast, the NON-URGENT processes are periodically time sliced to provide an even distribution of processor time between them. Each time slice lasts for about 1msec.

Communication between processes is achieved by means of channels. A channel between two processes on the same transputer is implemented by a single word in memory. a channel between processes executing on different transputers is implemented through the serial links discussed before. Process communication is point-to-point, synchronised and unbuffered. As a result, the channels need no process queue, no message queue and no message buffers. This leads to a *communicating sequential process*

[2.38] software model for the transputer.

The T800 transputer also has two 32-bit timers, which can be used for a incorporating delays etc. One timer is accessible only to URGENT processes and is incremented every microsecond. The other is accessible only to NON URGENT processes and is incremented every 64 microseconds.

An Event channel is also provided for interrupting the transputer. If, one and only one, URGENT process is waiting for an input on this channel, then the interrupt latency (from when the channel becomes ready to the when the process starts executing) is typically less than 1 microsecond. The reader is referred to T800 data sheet [2.36] for further details on these features.

To summarise, one can say that the T800 transputer has all the features required for building a large grain size MIMD machine suitable for IWC applications.

2.8.2 Development Environment

This section discusses other available transputer family of products, which are useful from the following point of view

- Lead to considerable saving in development time.
- Add features to upgrade the IWC in accordance to the design criterion established earlier in section 2.4.2.

2.8.2.1 Hardware Support Tools

The following transputer hardware products are of interest in the IWC implementation.

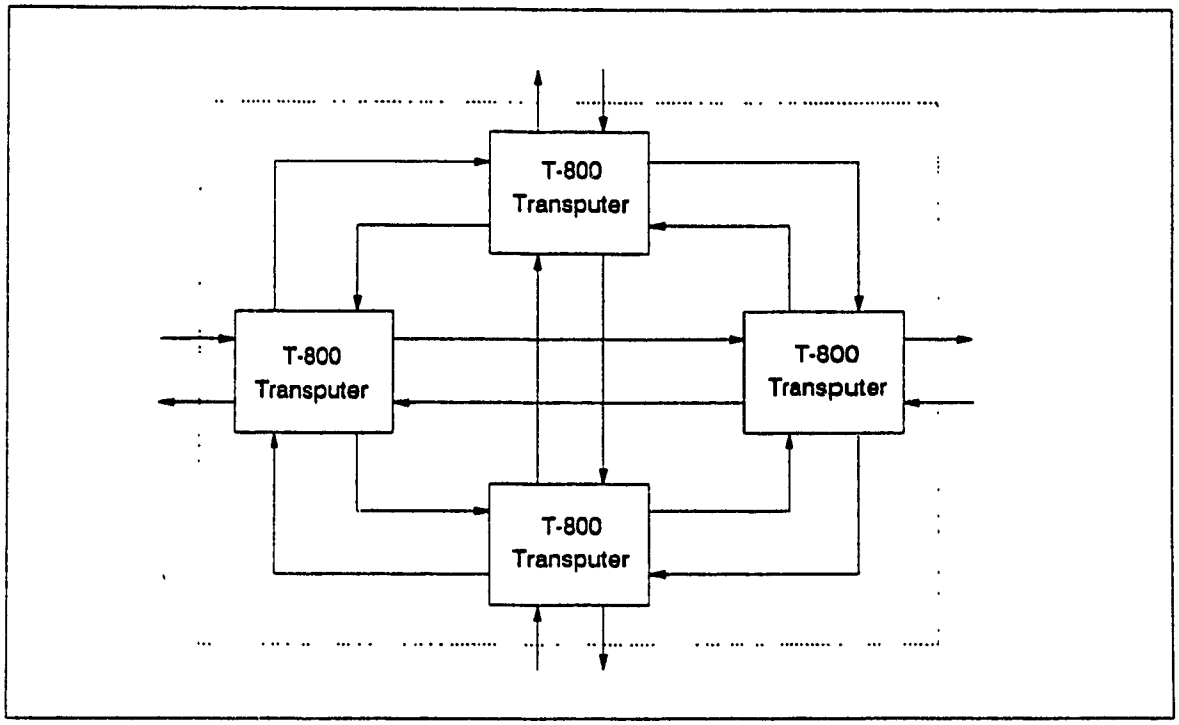

Figure 2.6 A node of four transputers

2.8.2.1.1 Transputer Module

The *Tr*ansputer *M*odules (TRAMS) are board level transputer products with simple and standardised interfaces [2.39]. Each TRAM has a processor interfaced to 1-8 Mbytes of RAM, four serial links for interprocessor communication and subsystem control circuitry. It is exceedingly compact and roughly the size of a credit card. A number of TRAMs can be interconnected to form multi transputer network. Figure 2.6 shows a scheme for building a four transputer node with four communication links. This pattern can be repeated to build MIMD nodes of increasing computational capacities.

A TRAM with T800 transputer interfaced to 1-2 Mbytes of memory suffices for a general purpose node of IWC. Special nodes optimised for image processing, sensor interface etc. can be implemented using the following *application specific* TRAMs,

available off the shelf.

(i) Framegrabber TRAM for real-time image capture and processing.

(ii) High Resolution Graphics TRAM, which supports upto 1024 x 1024 8 bit pixels, for enhanced graphics capabilities.

(iii) A to D Converter TRAM with 16 bit resolution and 200 KHz sampling rate for sensor interface.

Some limitations of the TRAM architecture are :

(i) The transputer data/address bus is not accessible for peripheral interface.

(ii) The Event channels is not accessible and can not be used to interrupt the processor.

2.8.2.1.2 Motherboard

A number of TRAMs are seated on a transputer motherboard to form a multi-transputer network. The motherboard is then plugged onto the host computer bus. Motherboards are available for IBM PC, PS/2, VME bus and SUN computer systems. One such motherboard, TM-B08 for IBM PC can accommodate 10 TRAMs and is discussed in greater details in section 3.2.1. As a convention, the transputer in slot 0, which is connected to the host is called the *root* transputer. Only the root transputer has the privilege to communicate with the host. Other transputer requests for host processor are routed through the root. Using DMA techniques, a data transfer rate of 200 Kbytes to 300 Kbytes between the root transputer and the host can be established. Multiple motherboards can be interconnected in a straightforward manner to form large networks, as discussed in [2.39].

2.8.2.1.3 Peripheral Interfacing

Apart from accessing the host peripherals, three methods are available for connecting peripherals directly on the transputer. These are :

(i) By using special peripheral TRAMs. TRAMs are available for hard disk controller, RS232, RS422, SCSI and GPIB interfaces.

(ii) By employing IMSC011 [2.37] link adaptor. This device converts between the serial transputer link and an eight bit parallel interface. The S/P interface (section 3.2.4) used in the IWC uses this technique.

(iii) Memory mapping the peripheral onto the transputer bus. This is *not possible* in TRAM architecture.

2.8.2.2 Software Development Tools

A number of software tools are available for developing applications on transputers. We start by considering the programming languages.

2.8.2.2.1 Programming Languages

To implement the parallelism possible in an algorithm, many concurrent languages have been developed over the last few years. Noticeably among them is the OCCAM programming language [2.40], which can be called as the *assembly language for transputers*. The OCCAM language bears a special relationship with the transputer architecture. Parallel computer systems can be developed in OCCAM and then implemented using transputers as *hardware OCCAM processes*. However, OCCAM provides limited support for pointer operations, data structures and dynamic allocation. Also, OCCAM requires the use of a Transputer Development System for software

development.

Other than OCCAM, parallel versions of many conventional languages have been developed such as Parallel C, Parallel Fortran and Parallel Pascal. The compilers for these are available from several vendors such as Logical Systems, Inmos and 3L etc. Taking into consideration the popularity of C language, its portability and the available software support, the Parallel C from 3L has been adopted as the programming language of the IWC. Parallel C has all the essential features of ANSI C alongwith the following special function libraries to support concurrent programming :

(i) *Channels <channel.h>* : The basic communication primitives, *chan_in_message and chan_out_message*, for transferring a message across a channel are provided. The channel may be between two processes running on the same processor or between two tasks running on different transputers. Variations of basic primitive are provided for a byte transfer, word transfer or an arbitrary message length.

(ii) *Threads <thread.h>* : A set of functions such as *thread_create, thread_stop* etc. are provided to create new processes at run time. Threads, which are Parallel C equivalent of Unix process, are discussed in greater detail in section 2.8.3.1.

(iii) *Semaphores <sema.h>* : A set of functions such as *sema_wait, sema_signal* etc., are provided to create, initialise and manipulate semaphores. Semaphores are used to synchronize the activity of several concurrently executing threads for accessing common data structures such as to avoid deadlocks [2.37].

(iv) *Alt <alt.h>* : The *alt* (for alternative) functions allow a process to input from a group of channels, whichever becomes ready to communicate first. It provides a

convenient means of handling external and interval events that must be handled by assembly level interrupt programming in conventional processors.

More functions for implementing *monitors, signals* etc., are available and the reader is referred to [2.41] for further details.

2.8.2.2.2 Debugging Tools

The availability of a powerful debugger can not be over stressed for projects requiring significant software development. TBUG [2.42] is a window based interactive source level debugger available for Parallel C. Using TBUG one can interact with running transputer programs, which may contain several concurrent threads/tasks, by inserting break points, analyzing/modifying variable or memory etc. However, a faulty concurrent program with ill defined timing dependencies may produce unexpected results under TBUG, where such time relations get changed, making fault identification extremely difficult.

A last observation regarding the prices of the transputer products is in order. With increasing applications the prices of transputer products have been falling steadily over the last few years. As an example, a package consisting of a TMB08 motherboard for PC with five T800-20 (20 MHz clock) transputers, having upto six megabytes of memory, along with the Parallel C compiler for software development was available for as low as $4800, in the last quarter of 1990.

2.8.3 Developing Concurrent Applications on Transputers

This section briefly illustrates the techniques used in developing concurrent applications in Parallel C on a network of transputers. As an example, we consider the

```
#include <thread.h>
#include <chan.h>

#define        STACKSIZE   1024

CHAN chan,consumer_finished;

void    producer()     /* To generate 10 values */
{
        int i;
        for (i=0;i<10;i++)
                chan_out_word(i,&chan);
}

void    consumer()     /* To consume 10 values */
{
        int i,val;
        for (i=0;i<10;i++)
                chan_in_word(&val,&chan);

        chan_out_word(1, &consumer_finished);
}

main()
{
        int dummy;

        chan_init(&chan);              /* Initialise channels */
        chan_init(&consumer_finished);

        thread_create(producer, STACKSIZE, 0);
        thread_create(consumer, STACKSIZE, 0);

        /* Wait for all threads to terminate  before exiting */
        chan_in_word(&dummy, &consumer_finished);
}
```

Multi-Thread Implementation of Producer-Consumer problem

producer-consumer problem [2.37]. One program, called the producer, generates information which is to be used by the consumer program. With multi-tasking capabilities

of the transputer, the producer-consumer problem can be implemented in two ways.

2.8.3.1 Multi-Thread Implementation : Parallelism on one processor

As mentioned before, Parallel C allows one to develop multi-thread applications. This means that a task (discussed below) can contain any number of concurrent threads, running on the same processor, each of which is independently executing the code of the task. Although all the threads created by a task share their code and data memory, each can still operate independently because each thread has its own stack and instruction pointer.

The box on the previous page shows an implementation of producer-consumer problem in Parallel C using channels for synchronisation. Recall that a channel in this case is just a shared memory location where the producer leaves its result and the consumer retrieves it in a synchronised fashion.

2.8.3.2 Multi-Task Implementation : Parallelism on Transputer Networks

If the producer and consumer programs are computationally intensive, processing time can be reduced substantially by mapping them on different transputers as independent tasks. A task in Parallel C has its own code and data areas separate from all other tasks, even when several of them are placed on the same processor. As there is no shared memory between tasks, they only communicate via channels.

For implementation as a multi-tasking application, a *configuration file* to describe hardware and software configuration needs to be defined. A configuration language and a software utility called configurer are provided in Parallel C towards this purpose. For a two transputer implementation of producer-consumer, as shown in Figure 2.7, the

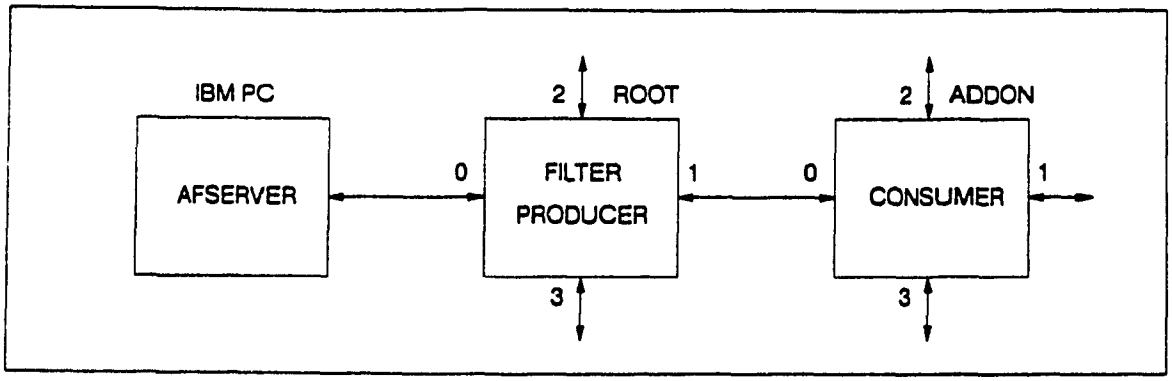

Figure 2.7 Two Transputer implementation of Producer-Consumer problem.

```
/* Producer */
#include <chan.h>

main(int argc, char *argv[], char
*envp[], CHAN *in_port[], int ins,
CHAN *out_ports[], int outs)

{
int i;

for (i=0;i<10;i++)
        chan_out_word(i, out_ports[1]);

}
```

```
/* Consumer */
#include <chan.h>

main(int argc, char &argv[], char
*envp[], CHAN *in_ports[], int ins,
CHAN *out_ports[], int outs)

{
int i,val;

for(i=0;i<10;i++)
        chan_in_word(&val, in_ports[0]);

}
```

Multi-Task Implementation of Producer-Consumer problem

required configuration file is as shown on the following page.

The *afserver* and the *filter* tasks are vendor supplied which run on the host and cater to the transputer request for host services such as file I/O, peripheral access etc.

The modified code for the two programs for multi-task application is shown in the text box above. The channel used in this case is mapped over the physical link connecting the two processors.

```
!       Producerconsumer.cfg

processor     host          !The PC
processor     root
processor     addon

wire    root[0]-host[0]     !Link connection between root Link 0 and host
wire    root[1]-addon[0]    !Link connection between Producer and Consumer

task    afserver ins=1 outs=1
task    filter    ins=2 outs=2 data=10K
task    producer ins=2 outs=2
task    consumer ins=1 outs=1

place   afserver host       !afserver runs on the PC
place   filter    root      !filter runs on the root
place   producer root       !producer also runs on the root
place   consumer addon      !consumer runs on addon

                            !Establish two-way connections between tasks
connect ? afserver[0] filter[0]
connect ? filter[0] afserver[0]
connect ? filter[1] producer[0]
connect ? producer[0] filter[1]
connect ? producer[1] consumer[0]
connect ? consumer[0] producer[1]
```

Configuration File for Two-Transputer Implementation

2.8.4 Performance of Transputers in Real-Time Control

Apart from robotics, the use of transputers in other real-time applications is becoming increasingly popular. Asher et.al. [2.43] have used transputers for high performance vector control of AC induction motors. Thompsom et. al. [2.44] have built a controller for gas-turbine engines using transputers and incorporating hardware/software redundancy for fault tolerance.

In the field of robotics, Whitcomb et. al. [2.45] have built a transputer based test

bed for manipulator control algorithms. However, the set up is not cost effective as a dedicated transputer is used for each joint. Sharkey et.al. [2.46] have used a pipeline of six T800 for variable structure control and inverse dynamics of Puma 560. Hashimoto et.al. [2.35] have used parallelism in Newton-Euler algorithm for inverse dynamics to implemented the computed torque control for the first three links of Puma 560. Rajagopalan [2.47] and Xiao et. al. [2.48] have also investigated the control of Puma 560 manipulator using transputers.

The effective timings for purely sequential implementation of kinematic algorithms, obtained by the author on the designed workcell controller, with T800-20 transputer, are as follows :

Table 2.1 Actual Timings for Kinematic Algorithms obtained from experimental set-up.

Operation	Time Required
Evaluating Joint Angle for one joint from the actual encoder count.	256 microseconds
Forward Kinematics of Puma 560	5.824 milliseconds
Inverse Kinematics of Puma 560	1.344 milliseconds

Real-time expensive parts of the code, where majority of the processor time is spent, have been identified as follows :

(i) Evaluating joint angle from encoder counts requires the use of a set of decision equations as discussed in section 4.5. It also requires considerable interprocessor communication between transputer and servo controller.

(ii) With multiprocessor architecture, time is spent in preparing messages in accordance with the communication protocol adopted in IWC and discussed later

in section 3.2.3.3.

(iii) Safety checks have to be included in forward kinematics to ensure that the joint angles, as evaluated from the encoder counts, are in valid range.

(iv) Safety checks have to be included in the inverse kinematics to ensure that the solution is in permissible range.

2.9 Summary

The advantages offered by an integrated workcell controller for multiple robotic devices synchronisation over the conventional schemes have been discussed in this chapter. Apart from offering a uniform hardware and software architecture at all levels, the concept of IWC leads to an open, compact, modular and highly flexible workcell controller. An analysis of the objectives and design criterion lead to a large grain size MIMD machine with local memory architecture for IWC. Although any processor or processor cluster with inter-processor communication capabilities can be used to implement a node in IWC, the T800 transputer with its unique architecture of four serial links and sufficient local memory is found to be highly suitable for implementing the workcell controller.

Chapter 3

Integrated Workcell Controller:

Architecture and Implementation

3.1 Overview

As the primary goal of this thesis, an Integrated Workcell Controller (IWC) satisfying the design criterion outlined in the last chapter, has been developed by the author at the Centre for Industrial Control (CIC), Concordia University. This chapter gives a comprehensive description of the hierarchical architecture of the CIC-IWC, henceforth referred to as IWC only, and its implementation using transputer technology.

Apart from high operational precision, fast response time, upgradable computing power and sensor interface, the proposed architecture is highly flexible and modular. Hiding the peculiarities of arm servos, geometry and other specific characteristics, it provides the user with a uniform set of well defined commands for well defined responses. The modular architecture of the IWC allows one to add new kinds of manipulators to the existing set up and incorporate them into the system by writing corresponding *software drivers*, with minimal disturbance to the existing system.

3.2 Control Hierarchy

The hierarchal structure of the IWC, as shown in Figure 3.1, consists of four layers. At the top is the IBM PC-AT, which acts as the host to the transputer network and as the OPERATOR console. Using a transputer link, it is connected to the second layer in hierarchy, the Global Task Planning Controller (GTPC) which performs the task

of synchronisation between several robots.

As described in the following section, in multi-arm coordination environment the operation of an individual robot can be split into a number of phases. As the GTPC has the global picture, it directs transitions between such phases by communicating with individual robot's Manipulator Controller, through a global message switch. The Manipulator Controller forms the third layer in the IWC architecture. Depending on the execution phase, the Cartesian Path Controller (CPC) generates the desired cartesian path (in association with other manipulator controllers) and gives suitable anchor points (in task space) to the Joint Space Controller (JSC) of the corresponding robot. The Joint Space Controller performs interpolation in the joint space and loads the desired set point in the motion controllers for joint servos every sampling interval.

However for extremely fast real-time response required in situations like collision avoidance, a scheme has been developed to close the *inter-robot loop* at the lowest level of Joint Space Controller. This is made possible by giving an *open architecture* to the Joint Space Controller, wherein it can communicate directly with the CPC of the robots in near physical vicinity, as indicated in Figure 3.1.

The motion controllers and the associated interface circuitry form the innermost control loop and the lowest layer in system design. In the following sections, we consider the function of each level and its implementation using transputer technology in greater details. However, any reference to transputer may be equally replaced with a cluster of processors with four outgoing links as discussed in section 2.8.2.1.1.The inter-processor communication protocol, specifically formulated for IWC, will also be discussed.

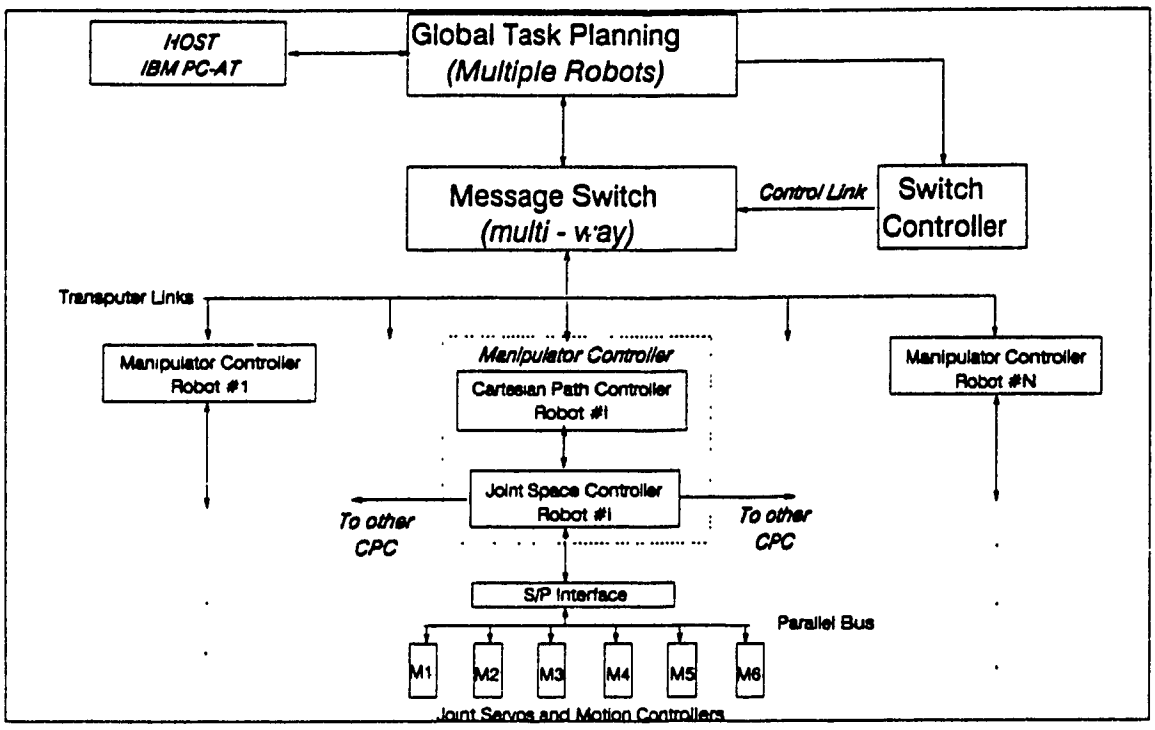

Figure 3.1 CIC Integrated Workcell Architecture

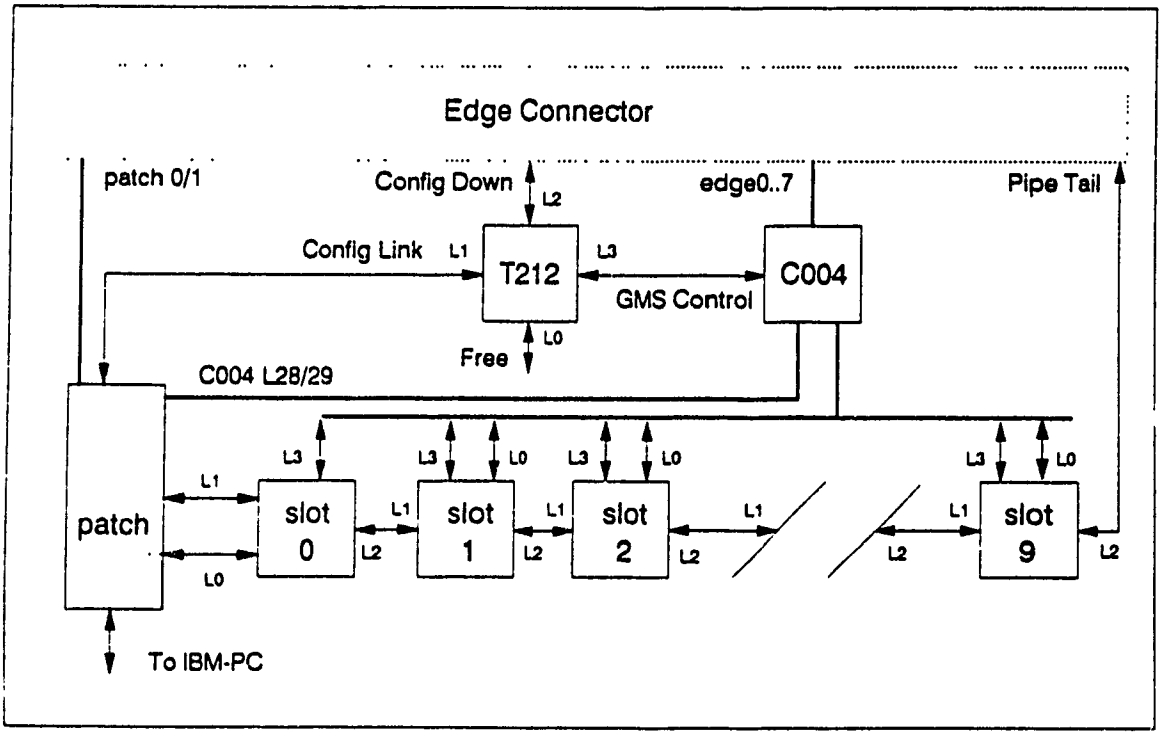

Figure 3.2 The Default Transputer Interconnections in TMB08.

3.2.1 Operator Console and Host

The IBM-PC serves the dual purpose of Operator Console and Host for the lower layers of IWC. Using the PC as the host offers certain advantages. First, the processing power of the PC has and will increase substantially in the near future, accompanied by considerable decrease in cost. PC remains to be the most popular form of local intelligence in the industrial scenario and hence a wide variety of transducers, data acquisition systems and vision equipment are available as add-on PC peripherals. Further, the often quoted drawbacks of PC i.e. its limited graphics and multitasking capabilities are disappearing rapidly with the availability of windows for multitasking and add on compute engines for graphics processing.

The Transtech TMB08 PC Transputer Motherboard [3.1] occupies one slot of the PC and supports all the lower layers of the IWC. The TMB08 is a full length PC hosted Transputer Motherboard with space to plug up to ten Transputer Modules (refer section 2.8.2.1.1). The architecture of the motherboard supports the following:

- Build computing system consisting of any mix of TRAMS (size and type).
- Configure the transputers in any desired topology.
- Link a number of motherboards together.

The important features of the motherboard architecture can be outlined as :

- The TRAM modules on the TMB08 are connected in a pipeline, using two links from each module as shown in Figure 3.2.
- The remaining links from each TRAM module can be configured by the user, either by direct wiring or through software programmable link switch

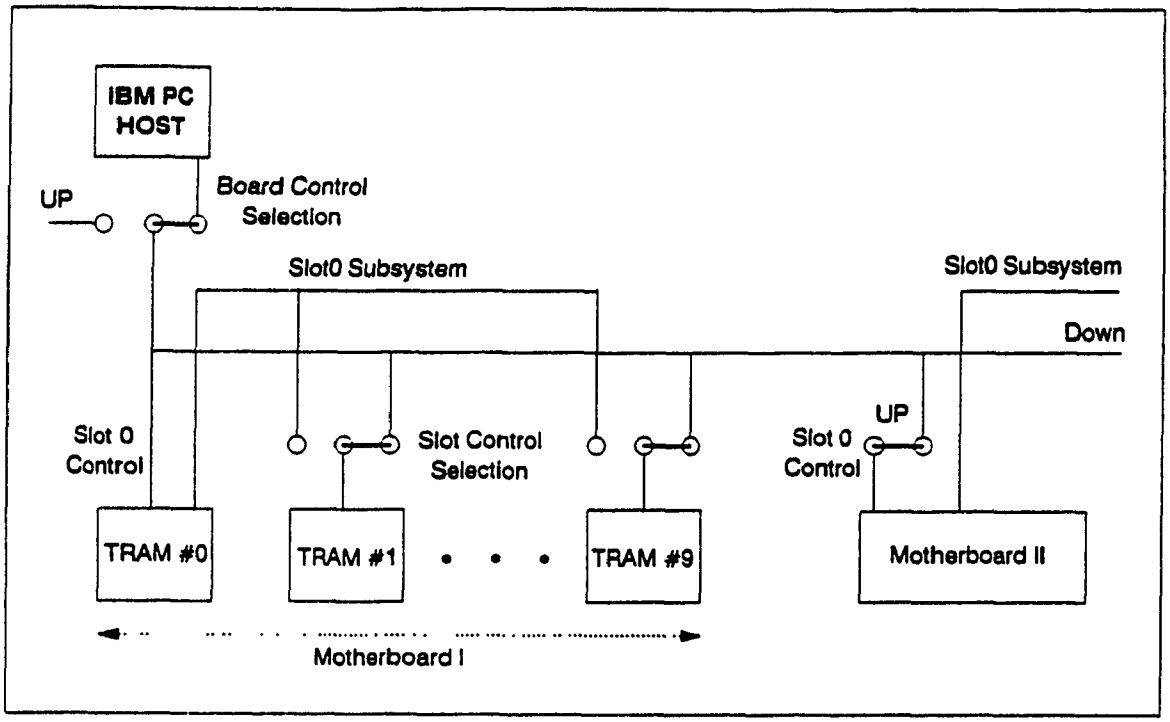

Figure 3.3 Subsystem Control in the IWC

IMS C004 present on board, for realising different network topology. The link switch is controlled by the on-board T212 transputer.

- The TRAM should be reset before a user program can be downloaded and executed by it. In the IWC, reset and other control signals for all TRAM subsystems are supported by the host PC as shown in Figure 3.3.

The set-up configuration for the TMB08 in the IWC is :

- Board Address : #150
- PC DMA channel used : #1
- PC Interrupt used : #3

The user must ensure that no other PC peripheral uses the same DMA/Interrupt channel. If however there is a clash with another add-in card, it is possible to change them as

described in [3.1].

The various tools of the parallel C programming environment (compiler, linker, debugger etc.) reside in the host PC and are invoked from it under the control of DOS operating system. The DOS provides standard I/O, file I/O and access to other PC peripherals for the Parallel C program executing on the transputer network. The special interface task called *afserver*, mentioned earlier in section 2.8.3.2, is provided by the vendor for this purpose, which runs on the host PC. This task provides access to the host computer for tasks running in the transputer system using one link of the root transputer. One major drawback of the current implementation of the 3L transputer file server is that it utilises a polling mechanism to detect transputer request for services thereby destroying the independent processing power of the PC and reducing it to the mere status of a server. However, this problems has been resolved earlier by the development of a Interrupt Driven Transputer File Server [2.32], in which it takes the form of a Terminate and Stay Resident program in the DOS and is waken-up by the transputer request. Though the current IWC is not making use of this enhanced file server, it is expected to do so in the future and free PC to perform other functions like providing a graphical interface to the user. The host computer interface of the transputer i.e. PC - TMB08 hardware interface, is a separate issue and not covered here as it is transparent to the end user. The interested reader is referred to [3.1].

3.2.2 Global Task Planning Controller

As mentioned earlier, the Global Task Planning Controller (GTPC) forms the second layer of hierarchy in the IWC architecture. This block performs the functions

listed below.

3.2.2.1 Cartesian Control of several manipulators

The GTPC carries a *global image* of the work environment and its relation to the job being executed. The complexity of this block cannot be underestimated as it spans over all aspects of robotics such as task definition and partition for multiple manipulators, collision and obstacle avoidance, interference zone detection, video & sensor based servoing etc. However, one essential aspect of GTPC function is to coordinate the global picture by switching the *phase of operation* of the robots constituting the workcell. It has been realised that in multi-arm synchronisation it is possible to split the operation of individual manipulator into certain phases of operations as described below.

(i) **Set-up phase** : A phase in which problem dependent and arm dependent code is downloaded (individually or broadcasted) to the individual arm controllers. Subsequently, each manipulator *initialises* itself, runs diagnostics and takes a pre-defined *HOME* position.

(ii) **Execution of predefined (non interfering) path** : In this phase each arm executes a certain pre-defined preparatory trajectory which lies totally in the non-interfering zone, so that it can be executed independently. Such a trajectory may imply picking a tool/workpiece from a pre-defined location and reaching a certain point before the commanded time.

(iii) **Coordination with neighbours** : This phase marks the manipulators entry into a common workspace with its neighbours. After switching the associated manipulators into this phase, the GTPC may delegate one among them as *local*

master and direct the others to *close the loop locally* with the new designated master at the Cartesian Path Controller level. It is important to note that the neighbour need not be a manipulator always but may be any motion device like conveyor belt etc.

(iv) **Termination and safe return to Nest** : After having received acknowledgements from the local masters, the GTPC takes charge and charts out non-colliding return path for each manipulator to its native position called *Nest*. The GTPC updates the desired data on the host and terminates its program, after having supervised proper termination of individual controller programs.

As is clear, the complexity of the GTPC block directly depends upon the number of participating manipulators. Hence, it may be mapped to a single transputer or to a cluster of them with a private message switch.

3.2.2.2 Global Message Switch Control

It is clear that the proposed architecture of the IWC leads to significant inter-processor communication. The management of the **Global Message Switch (GMS)** is an integral functions of the GTPC. Some observations on the implementation of GMS in transputer framework are in order.

The GMS is implemented in a straightforward manner using the on-board **Electronic Link Switch IMSC004** [2.36]. The IMSC004 acts as a crossbar switch. It can be connected to 29 transputer links, and can be programmed to switch any connected link to any other connected link in real-time. The reader should not be alarmed by the figure of 29 as a possible limitation as generic networks can be build by cascading many

IMSC004. It should be noted that the **complete connectivity of up to 32 Transputer** devices is attained by using just four such link switches. With respect to TMB08 motherboard, the links connected to IMSC004, as shown in Figure 3.2, are :

(i) Link 0 of all TRAM slots (except ROOT#0 Link #0 which goes to the PC host).

(ii) Link 3 of all TRAM slots.

(iii) Eight links from edge connectors to communicate with other motherboards (down-stream or upstream).

(iv) Two spare links from patch area.

The C004 can be programmed (in real-time) by the on-board **Switch Controller** T212 configuration processor. The switch controller is in turn programmed by using one of the links of GTPC cluster, of which it is a constituent part. In the current implementation the NCS Configuration Software [3.1] is used to program the link switch. In this scheme, the Link #1 of the root transputer is connected to the Link #1 of the Switch Controller through the path area. The desired link configuration is read directly from a MSDOS text file called *connect file* and downloaded to the link switch through this connection.

3.2.2.3 Multiplexing Requests for Host Services

This important function of the GTPC results from the fact that the transputer file server residing in the PC cannot be shared between multiple transputer tasks. This is a direct outcome of the transputer *point to point* communication architecture. As a result, only one transputer task can make use of host services which includes C standard I/O functions like *printf, scanf* etc. This is a strong limitation specially in the design phase

where such statements provide a strong support, and sometimes the only means, for debugging. The only solution is to implement some sort of a *multiplexer task*, which can accept requests from several processors and route them to the PC. Though initially the author attempted to write such a task, later it was provided by the vendor in its revised version of the compiler. The task *filemux* [2.41], which comes as an integral part of the 3L parallel C (version 2.1), allows several transputer tasks to share a single file server running on the host and is mapped on one of the processors of the GTPC cluster.

To conclude, the GTPC establishes a two-way channel between the host and the lower layers of IWC and coordinates the manipulators to perform some productive task. To perform this it makes use of TMB08 - PC hardware interface, software interface and the on-board link switch with switch controller. It carries the global picture of the work environment and delegates control jobs to other manipulator controllers.

3.2.3 Manipulator Controller

As the name suggests, the function of this level is the complete cartesian/joint space control of a manipulator in order to obtain the end effect desired by the GTPC. In Figure 3.1 it has been split into two processing blocks which are :

(i) **Cartesian Path Controller (CPC).**

(ii) **Joint Space Controller (JSC)**

The above two work in synchronisation to control the arm to follow the desired cartesian path with any added limitation, like constant speed etc. as specified by the GTPC for the particular phase of operation. The complexity of these depends on the geometry of the arm and desired response time. Experience has shown that for a general

6 DOF Puma type arm each of the above mentioned can be mapped onto a single transputer without compromising with the system sampling period. However for a more complicated arm, like redundant robots, it may be desirable to map the above onto a cluster of transputers.

As mentioned before, in the set up phase specific task code and arm dependent data are downloaded in the manipulator controller. The arm dependent data consists of link lengths, joint resolutions, gear ratios etc. These are stored in the host as corresponding header files e.g. *puma260.h*. The functions of the above mentioned controllers will now be briefly explained.

3.2.3.1 Cartesian Path Controller

The CPC generates the cartesian path to be pursued by the End Effector (EE) of the manipulator. This path may be pre-determined from the phase of operation or may be generated by using cartesian space interpolation techniques, discussed in Chapter 5, on sensor identified selective points.

After having determined the desired cartesian path the CPC communicates a suitable number of *anchor points* (in world coordinates) to the JSC for constructing smooth joint trajectories using appropriate interpolation techniques. Also, at regular intervals, the CPC interrogates the JSC for the actual position of the EE. The CPC may update (if commanded so) the GTPC with this information or, if acting as a local master, may take independent decision to direct the JSC of the *neighbouring* manipulators. How this is made possible shall become clear in the following section.

As is seen from Figure 3.1, two links of the CPC transputer (or transputer cluster)

are hardwired towards the corresponding JSC and GMS respectively. This leaves the CPC cluster with two free links. These are hardwired to the JSCs of the two neighbouring manipulators for *closing the loop locally* in times of synchronisation. If one robot has more than two neighbours or is synchronising with "not-a-neighbour", then it can communicate with the CPC of the particular manipulator through the GMS.

3.2.3.2 Joint Space Controller

The Joint Space Controller (JSC) controls and coordinates the multiple axis of the particular manipulator in the joint space. Its architecture is very much dependent upon the nature of the actual motion controllers working under its supervision and the hardware interface with them. However the whole purpose of the JSC is

> *To hide the peculiarities of the manipulator characteristics and the joint servos, from the higher layers such that the latter are presented with a uniform set of device independent commands with well defined actions.*

It is this feature of providing uniform interface across all types of manipulators which provides the IWC with its capability of integrating new types of manipulators as *add on arm* without disturbing the existing setup. Also it provides a much desired partition between the hardware specifics (manipulator servo related) and the control algorithm selected. Thus to add on a new manipulator type, one can independently decide on the servo controller types and their interface to the transputer, implement the existing commands and then add it to the IWC with minimal disturbance.

The JSC receives a number of anchor points from the CPC, transforms them into joint space using inverse kinematic formulation and using appropriate interpolation

techniques, generates smooth joint trajectories passing through them. Every sampling period JSC updates the *set point* of the individual servo controllers, which drive the corresponding servo to the target position. Depending on the complexity of the algorithm used for determining the joint space trajectory, which may vary from simple trapezoidal profile currently implemented to generating optimal spline functions with suitable criterion to observe the servo constraints, the JSC may be mapped to a single transputer or a cluster of them.

3.2.3.2.1 Open Architecture

Last but not the least important feature of the JSC is its so called **open architecture**. The term open architecture has been used to denote the capability of the JSC to communicate directly with the neighbouring CPCs for assuring fast response in real-time coordination. To appreciate this concept the reader should foresee the potential limitation of the message switching scheme of the IWC. As the system complexity grows, sending the "end effector position update" from one JSC to the neighbouring JSC by routing and forwarding it all the way up to the Global Message Switch and back may take considerable time and lead to a possible limitation on the overall sampling period. So a method had to be devised which allowed the control loop to be closed between the neighbouring manipulators, at a lower level, such that information may be exchanged at a reasonably fast rate to assure coordination of the participating arms.

To implement this, the JSC's design must permit interrogation by more than one processor and assure proper response to them. This is made possible by using the function *alt_wait_vec(n,in_ports)* of Parallel C. Briefly, this function blocks the execution of the

```
CHAN *in_ports[n], *out_ports[n];

/* in_ports and out_ports are array of communication channels */
/* Initialise arrays with link addresses */

port = alt_wait_vec(n,in_ports);

/* Use port as index for all further communication */
chan_in_message(&command,in_ports[port]);

/* Process the request */
/* Output the response to the processor which sent the command */

chan_out_message(&response,out_ports[port]);
```

calling task until a message is received from any one the communication channels listed in its argument list. The function returns the index, in the array in_ports[ ], of the channel attempting to communicate. In the IWC, the JSC is hard-wired to three other transputers using transputers links. Out of these, one is its CPC and two are the CPCs of the neighbouring manipulators. The construct shown above determines the processor trying to communicate and assures correct replies. It is important to note that no processor time is consumed while waiting and any other thread can continue execution in the JSC.

3.2.3.3 Inter-processor Communication Protocol

The need for a compact communication protocol for information exchange between the CPC and the JSC is self evident. The communication has to be two-way essentially. Though mostly motion related commands flow from top (CPC) and acknowledgements from the bottom (JSC), some *interrupt kind critical conditions* may demand the reversal of the information flow. Considerable time and effort was spent in the design of the communication protocol between the CPC and JSC. At one hand it should not be too

general so that the message formation and processing at the two ends may become too time consuming and on the other it should not be too restrictive so as to pose severe limitations as the system grows. Apart from providing a uniform manipulator interface across the IWC, the protocol should also provide a means for accessing the peripheral devices of the lowest layer i.e. particular motion controllers, A/D converters etc. Although in the end product only JSC is supposed to communicate with these peripherals, however an access to them from higher layers is invariably required during the development phase and for debugging and fault identification. Needless to say that the protocol should be real-time efficient.

The designed protocol consists of splitting the message into two fields. These are

(i) **Message Header**

(ii) **Message Data Fields**

Every message *must* have a message header which *precedes* the (if any) message data fields. A message header may or may not have any associated data fields depending upon the context. Further if the context of the header demands data communication then the data fields *must* follow the header with no other message in between. A data field with no corresponding message header is *illegal* and (in general) will lead to unpredictable behaviour.

As mentioned in section 2.8, communication with respect to transputers is essentially asynchronous both at the process level and the bit level. Thus, a message data field may follow its header after any interval of time. However, the process at the receiving transputer will be made *inactive* and blocked from proceeding ahead for that

duration of time. At present, the author sees no situations where a intentional time delay between the header and its data field may be required.

A last observation before the message sub fields are considered in greater details is that no error checking codes have been incorporated in the communication protocol. The architecture of the IWC makes it immune to the conventional types of data corruption errors, as the inter-processor communication channels in our case are pcb tracks with their inherent high reliability. Further as all the transputers are physically on the same motherboard, the inter-processor distance is bare minimum. Reference [3.2]-[3.3] discusses some techniques for interconnecting transputer links and recovering from link failures.

3.2.3.3.1 Message Header

The message header is of one word length (32 bits) and is further split into four byte long subfields as shown in Figure 3.4. Immaterial of the internal distinction, the header should always be transmitted as a single unit. The four subfields are as follows :

(i) **cmd_type** (LS Byte) : This field forms the least significant byte of the header word. cmd_type (short for command_type) tells the receiving transputer (JSC) that the intended command is for which device. The command may be for the JSC itself or for another intelligent peripheral downstream, like the motion controller. A list of such commands implemented in the IWC is given in Appendix A.

Commands like **MVEL_6JT, INVKIN, CALIBRATE,** etc. are of a generic nature and indicate specific actions to be performed by the JSC like move to a desired position in velocity mode, return inverse kinematic solution and

calibrate the manipulator respectively. It is the responsibility of the JSC to consider the characteristics of the particular manipulator under its control to bring about the desired effect and thereby provide a uniform interface to the layers above.

If the command is for a intelligent peripheral downstream, as in the case of **CMD_LMIO** (command for motion controller), the JSC acts as a *store and forward buffer*. This prevents the malfunctioning of a lower peripheral device blocking the execution of the CPC. For in simple *receive and forward* scheme, if the JSC fails to download a part of the data field (which may be transmitted from the CPC as multiple chan_out statements) to a faulty peripheral, then the CPC will be halted at the next attempt to chan_out a data item. In the store and forward scheme, the JSC receives the complete message (including the data field) from the CPC before forwarding the same to the peripheral device. Thus CPC is immediately relieved and does not get stuck due to a downstream error. The actual command of the downstream peripheral is found in the lm_cmd subfield, which will be discussed in a short while.

(ii) **channel_no** : The more significant byte of the lower byte pair of the header constitutes this field. For a general 6 DOF arm, this field determines the particular joint(s) for which the accompanying command is intended. At present this field has been formulated with a 6 or less DOF manipulator in mind. The channel_no (channel here implies manipulator joint) byte is actually bit organised as shown in Figure 3.4. BIT0 (least significant bit) to BIT5 of this field correspond to Joint

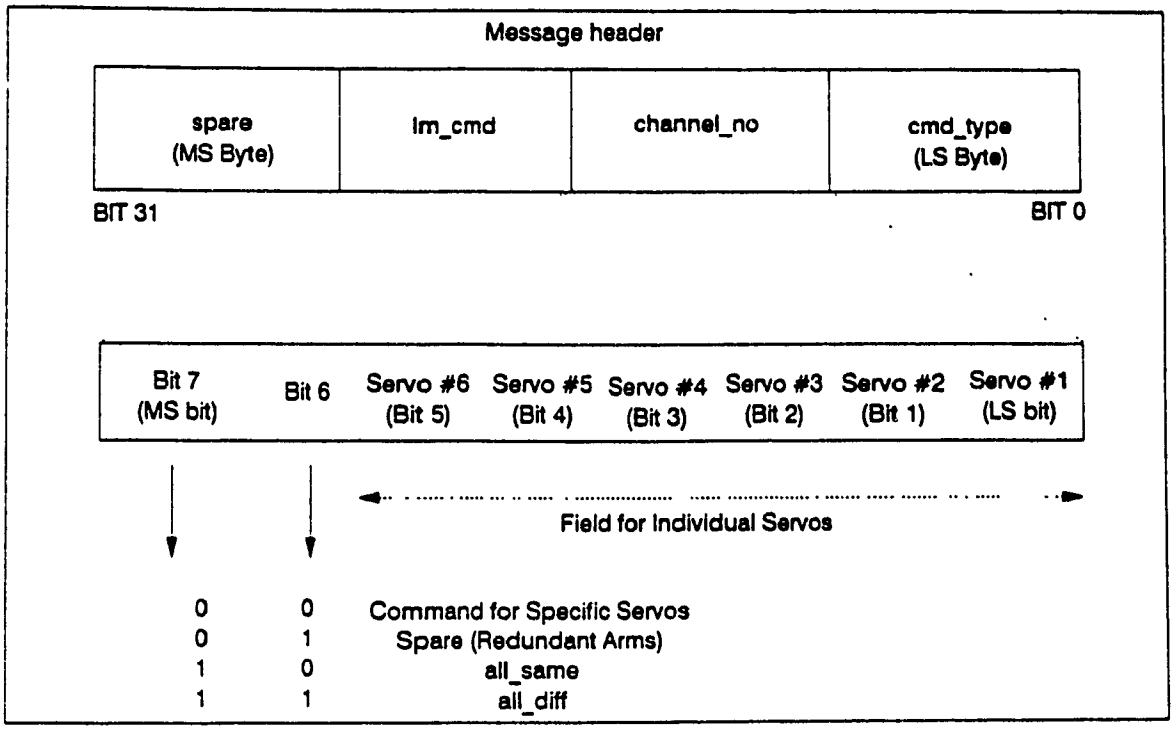

Figure 3.4 Message Header and bitfields of channel_no.

1 (base end) to Joint 6 (EE end) of the arm, in the respective order. If the particular bit is set (=1) in the channel_no field, it implies that the command is valid for the corresponding joint. Thus if the CPC wants the joint angles of the first three joints, it will send **Get_Angle** command (in cmd_type field) with 00000111 bit pattern in the channel_no field. The same command for the last three joints (wrist) will have 00111000 in its channel_no field. Such a scheme drastically reduces the CPC-JSC communication, for otherwise independent message will have to be sent for each channel.

BIT6 and BIT7 are used to further reduce the communication overhead. Table 3.1 shows the significance of these bits. The first combination is the most often used, however **ABRUPT_STOP** (stop all

Table 3.1 Significance of BIT7 and BIT6.

BIT7	BIT6	Description
0	0	Command for specific joint(s). The JSC scans the remaining bits of the channel_no subfield and determines the specific joints for which the command is intended, in the manner described before.
0	1	Spare at present. Can be used as an extension for **Redundant** manipulators.
1	0	Same command for all channels with **same data field.** The command is meant for all channels of the specific manipulator and the data following is to be used for all the channels. This condition is represented in software by an **all_same** flag.
1	1	Same command for all channels, but with **different data fields.** The accompanying command is for all the channels of the manipulator but the data items are different and will be loaded in the order of channel #1 to channel #6 respectively. The flag **all_diff** indicates this condition.

joints with maximum deceleration) and **LFIL** (load PID filter coefficients) are conditions where all_same and all_diff combinations are used respectively. The all_diff condition can be alternatively communicated by setting BIT0 to BIT5. However, that will require testing of six independent bits as compared to two and thus be more time consuming. In the current implementation, the JSC tests for the all_same condition first as it is most often used for safety related actions like sudden stop etc.

(iii) **lm_cmd** : The lower byte of the upper byte pair constitutes this subfield. This field is used in conjunction with the cmd_type field. When the cmd_type field indicates that the command is for an intelligent slave device downstream, as in the case of CMD_LMIO, then this field has the actual command for the particular

slave device. In the current implementation it carries the actual LM629 motion controller commands (discussed in 4.4.1) to be forwarded.

(iv) **spare** : The upper byte of the upper byte pair of the message header constitutes this subfield. It is kept free at present to support system growth with time. If in the future, support is require for a 16-bit peripheral, then this field may be merged with the 8 bit lm_cmd subfield for supporting 16-bit commands.

3.2.3.3.2 Message Data Fields

As mentioned earlier, depending on the context a message may or may not have any accompanying data fields. Number of data words is implicitly decided by the command in the message header and can be determined by the CPC (for transmission) and JSC (for reception) independently. In the current implementation a buffer, called mot_dat (for *mot*or_*dat*a), is maintained at each end and used in data transmission. The buffer mot_dat is an array of 6x4 dimension, with one row used for storing data for one servo channel. A word (4 bytes) of data per channel is the maximum required for any command in the current stage and a chan_out_word/chan_in_word combination is used to transmit and receive data for a particular servo channel.

3.2.4 S/P Interface and Servo Controllers

The serial/parallel (S/P) interface, the servo controllers and other related circuitry constitute the innermost control loop in the IWC architecture. As mentioned before, a general purpose interface between a serial digital device and a 8-bit parallel digital device has been earlier designed by Simon & Gilles [2.33] and upgraded by the author for better performance. For complete details reader is referred to [2.33] and only some essential

features of the interface circuitry are discussed here. An Inmos IMS C011 link adaptor [2.36] forms the heart of the S/P interface by converting bi-directional serial link data into parallel data streams. The IMSC011 is operated in peripheral interface mode (mode 1). In this mode it provides eight bit parallel input interface, eight bit parallel output interface and full handshaking signals for both input and output. On the serial side the IMSC011 is connected to the JSC through one of the transputer links. On the parallel side the IMSC011 is interfaced to the peripheral devices through programmable logic arrays (PLAs).

The details of the PLA implementation are well discussed in [2.33] and will not be dealt with here. However attention is drawn towards certain advantages in using PLAs for such an interface over the regular TTL interface circuits. It is well known that the PLAs provide a higher gate density and thus lead to compact circuits. Also, their propagation delay is considerably lower than the regular TTL circuits and they can be readily reprogrammed to accommodate modifications. However the most important reason for the current design is that with PLAs it is possible to design an interface which could select and write data to multiple devices at the same time. This capability is essential for synchronising multiple axis control where it is desired that all the servos start the motion simultaneously. In case of IWC, all the joint servos are pre-loaded with their specific trajectories and signalled to start motion by writing a single START command simultaneously to all of them.

In the current implementation, S/P interface has been used to interface six LM629 motion controllers, two AD7828 A/D converters [3.4] and a number of general purpose

digital input/output latches to the JSC transputer. Their addresses are given in Appendix B. The reader should notice the address ALL_SERVO (0x0f) which is common to all motion controllers. Thus a START command written on this address will start motion on all the joint servos (programmed with a valid trajectory) simultaneously. The A/D converter is mainly used for reading potentiometer voltages to determine the absolute position of the arm at any time. It can also be used for reading other analog signal from the environment, like the desired speed of the manipulator etc. The digital latches can be used for varied purposes. The digital inputs can be used to receive triggers from the environment to start synchronisation with a particular arm. Digital outputs may be used to set/reset other subordinate devices.

A typical write operation for a peripheral device from the JSC end looks like the following.

---

chan_out_byte(WR_CMD(DAT)_CMD | (channel << 4), LinkxOutput);
chan_out_byte(command or data, LinkxOutput);
chan_out_byte(RESET, LinkxOutput);

The first chan_out byte is for the interfacing PLA. Here *channel* carries the address of the peripheral device being accessed and *Linkx* (where x is between 0 and 3) denotes the transputer link being used for communication between JSC and S/P interface. It tells the PLA that wether the byte following is a command or data and for which slave device. As a result, the PLA selects the proper port (command port or data port) of the desired device. The second chan_out command contains the actual byte to be transmitted to the slave device. Here the PLA acts like a transparent buffer. The third chan_out

command is used to reset the PLA. The PLA deselects the slave device and returns to a known initial state. A similar scheme is used for reading data from the peripheral device.

Several general purpose shell functions like **wr_cmd, rd_dat_byte, rd_port, wr_port** etc. which are used for communication between the JSC and the peripheral device have been developed. These can be called by any program to perform peripheral I/O.

With this we conclude our discussion of the control hierarchy in the IWC architecture. The actual implementation of the proposed architecture for two-manipulator control will be further taken up in Chapter 6.

3.3 Summary

A detailed discussion of the Integrated Workcell Controller architecture, as implemented at CIC, has been presented in this chapter. As can be seen, the concept of a IWC, a robotic workcell capable of coordinating multiple industrial robots and other motion control devices from a central intelligence, is a industrially viable one. Although transputer technology has been used in the current design, the proposed design can be implemented using any processor type with capabilities to communicate with four or more neighbouring processor clusters. It is noted that with transputer products, the central control station is **very compact** in the form of a transputer board plugged into a IBM PC and can be located at any convenient place on the shop floor. Transputer links with differential drivers are used to connect it to different robots, which will be located close to the workpiece. With the decreasing price and increasing computational power of the transputer related products, the system has a wide horizon for future growth.

Chapter 4

Joint Space Controller : Implementation for PUMA Type Arm

4.1 Overview

The purpose of the Joint Space Controller (JSC) and its position in the control hierarchy of the IWC were discussed in the preceding chapter. Briefly, the JSC hides the specific characteristics of the arm and its interface to the IWC by supporting a set of device independent commands. Thus for the higher levels, a Puma type arm, a Cincinnati Milacron or a Yamaha Z-1 deburring robot, all appear like a generic manipulator supporting a uniform set of commands. Not only does this scheme make the implementation of a global control strategy much simpler but also the coordination between arms with different characteristics is greatly simplified thus allowing a local master at CPC level to coordinate a family of robots with mixed characteristics.

As a typical design example, this chapter discusses the implementation of the JSC for a Puma arm in detail. It is expected that this discussion will provide enough guidelines along which a JSC for a different type of arm can be developed in the future and integrated in the workcell.

4.2 Control Approaches to Robots

Before considering the implementation of the JSC, an insight of the requirements and architecture of a robot controller is required. For the current discussion, we restrict our attention to the permanent magnet DC servo motor actuated robot arms. Two alterative

approach exist in the control of DC motor actuated robot arm [4.1]. One approach is to apply to each joint the necessary torque by manipulation of the motor current. The desired torque is evaluated by using the *dynamic model* of the manipulator, which incorporates friction, gravitational forces and dynamic torques due to moment of inertia. The torque/force required to manipulate the workpiece is also included. This control scheme is essential for applications requiring a specific torque such as in deburring, metal cutting etc. However, not only is the evaluation of dynamic model computationally demanding but also an error in it can lead to disastrous results. This may happen if e.g. the estimated value of inertia is larger than the actual. Then, the torque applied will be larger than required leading to high acceleration. This may lead to damage to a workpiece since the manipulator velocity at the target position is non-zero, leading to a collision.

The alternative approach is to control the velocity of the robot arm by manipulating the voltage of the servo motors. The scheme is computationally much simpler and safer when compared to the torque control. The variations in the moment of inertia affect the time constant of the response and do not lead to any catastrophic consequences. The arm reaches the target position in stipulated time and at a low speed. In the worst case, the fuse of the voltage amplifier may blow if the arm encounters a rigid obstacle. However, as the control over the torque/force exerted is lost, this approach is not suitable for applications requiring a constant torque/force. It should be noted that either velocity or torque may be controlled, not both of them.

For multi-arm synchronisation, which is the end goal of IWC, torque control becomes computationally very demanding. For safe operation, a dynamic model of high

accuracy is required for each participating arm. Hence, it was decided to use the voltage control scheme as it is simpler and safer to implement for the beginning stages.

4.2.1 Architecture of a Robot Controller

The hierarchical structure typical of many robot systems is shown in Figure 4.1. At the top of the system is the robot supervisory computer and at the base are the servo controllers, typically microprocessors, one for each axis of motion. The supervisory computer performs the following functions :

(i) Trajectory planning and interpolation in world coordinates.

(ii) Coordinate system transformation from world to joint coordinates. Specifically, it loads the servo controllers with the target *set point* every T_c seconds. T_c is called the controller sampling period and is typically of the order of 30 milliseconds.

(iii) If any high level programming language is supported by the robot controller, then it also contains the compiler, interpreter and other related software.

The functions of the servo controllers can be briefly listed as follows :

(i) It receives the servo set point every T_c time period from the supervisory computer.

(ii) It controls the servo loop with a sampling rate of T_s seconds, i.e. it reads the position feedback from shaft encoder, evaluates the position error and generates a drive signal using the control algorithm prog.ammed. T_s is typically in the sub millisecond range.

(iii) Output the drive voltage to the actuator using Pulse Width Modulation (PWM) or Digital Analog Conversion (DAC) techniques.

The techniques used in *interpolator*, which form the heart of robot control system,

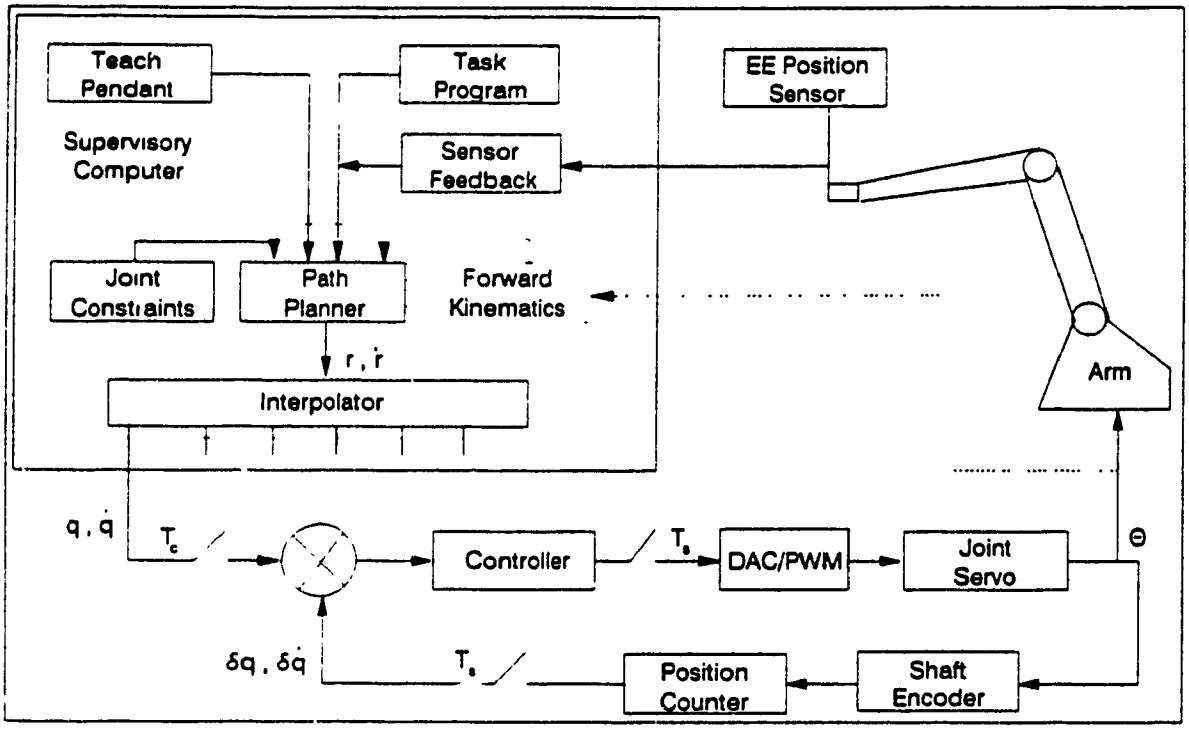

Figure 4.1 Architecture of a robot controller [4.1]

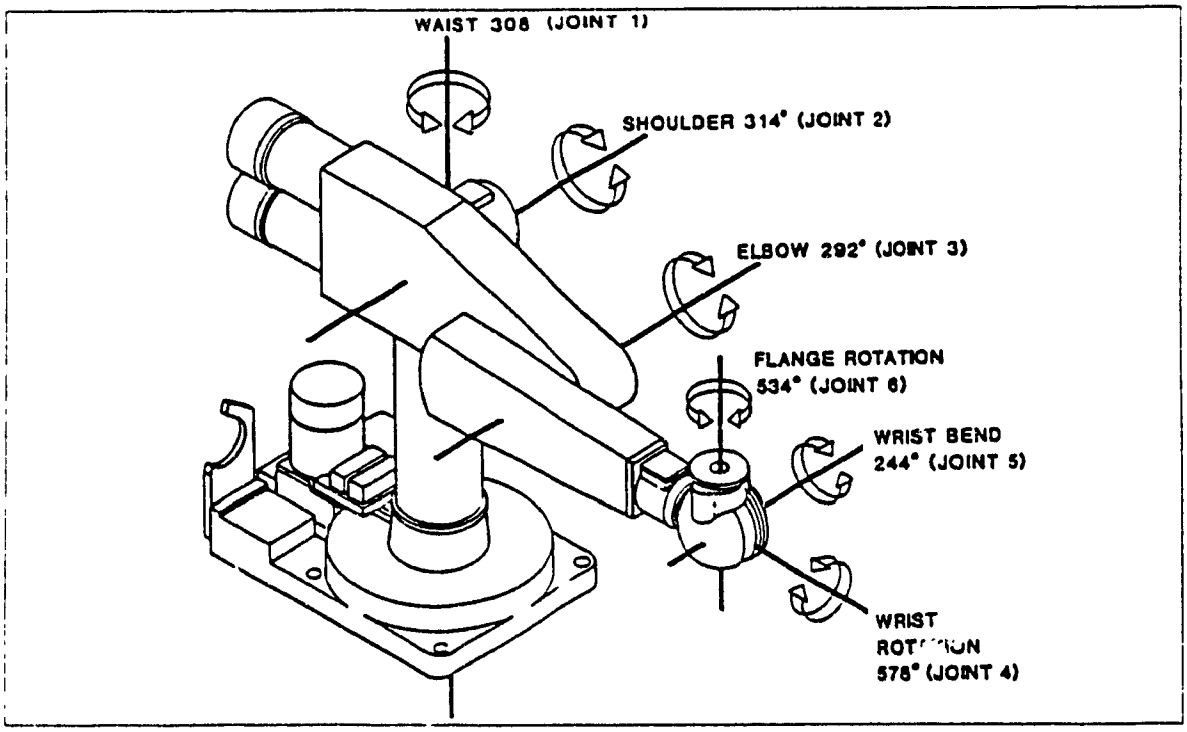

Figure 4.2 Puma 260 robot arm [4.12]

are discussed in section 4.3. It is important to note that the end effector position feedback using direct kinematics, indicated in Figure 4.1 by the dotted line, is practically not possible. This is because $T_s \ll T_c$. In general, the difference in their order of magnitude may be anywhere between 30-100 times. Thus, by the time interpolator calculates the new set point for the joint servos, which takes a major part of the T_c period, the end effector (EE) position feedback from direct kinematics is no more valid (assuming of course that the arm is in motion) leading to high position errors which increase with each iteration and make the scheme unstable.

Thus, effectively from interpolator point of view the system is open loop i.e. it assumes that all the joint controllers will drive their respective servos to the target set points with no errors. Though ideally one can overcome this limitation by reducing T_c, but in practice several factors like joint resolution, computational complexity of the algorithm etc. impose a lower limit on T_c. This is discussed in detail in section 4.7. To the author's surprise, this open loop control from interpolator is not discussed anywhere except for a subtle hint in [4.1], and he learnt it the hard way of repeated trials to incorporate the feedback with no success. For any scheme involving EE position feedback, one has to develop an independent sensor interface and use it to modify the points handed over to the interpolator from the task program as shown in Figure 4.1.

4.2.2 Puma 260/560 Robot System

The Puma is one of the most widely used robot in both industry and academia and a Puma 260 [4.2]-[4.3] and a Puma 560 [4.4] arms have been interfaced to the IWC for experimental study. Except for the differences in physical dimensions and payload

capacity, the construction of both the arms is on similar lines. Both are 6-DOF manipulators with revolute joints using permanent magnet DC motors as joint actuators. In general we shall refer to both of them as a Puma type arm. A Puma 260 arm is shown in Figure 4.2.

Primarily, the Puma robot arm works under the supervision of VAL controller [4.5]. The architecture of the VAL controller is shown in Figure 4.3. Essentially, an LSI-11 computer is used for high level control. The LSI-11 interprets the VAL commands and sends out set points to each of the joint servo loop every 28 msec, which is the controller sampling period T_c. A 6502 microprocessor controls each servo loop and samples the joint position every 0.876 msec, which is the servo sampling period T_s. The VAL controller uses a Proportional-Derivative (PD)/ Proportional-Integral-Derivative (PID) filter in servo loop. When in motion, the robot is under PD control. Integral control is added when the robot is stationary to maintain positional accuracy. This constitutes a brief description of the servo-level part of the Puma controller. The reader is referred to [2.14], [4.6]-[4.7] for further details.

Several attempts have been made earlier to control the Puma arm without the VAL controller [4.8]-[4.11]. The approach has been mainly to interface an external computer to the Q-BUS or DRV-11J serial/parallel interface cards. However unless one is in direct and total control of the joint servos, as attempted by Kazanzides et. al. [4.10], in all other schemes one has to live with some or the other limitations of the VAL architecture. The situation becomes hopeless when one compares the order of difference between the processing power of the transputer technology with that of the VAL controller. Hence it

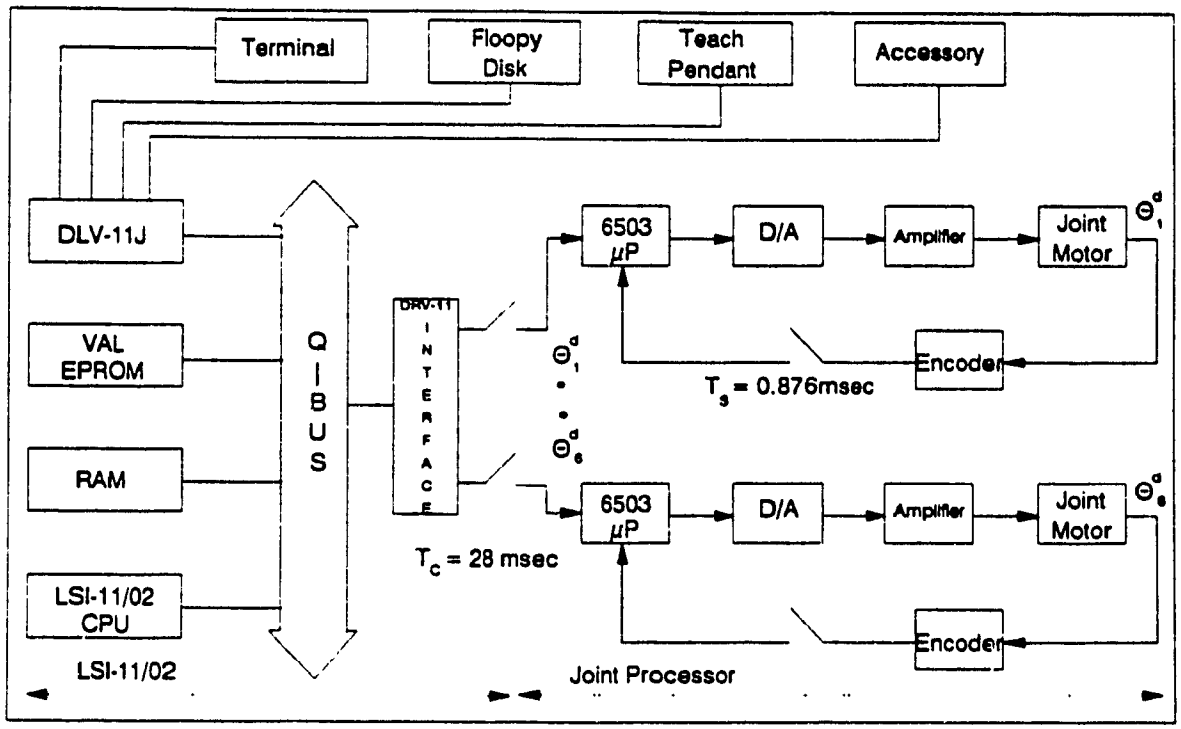

Figure 4.3 Architecture of VAL controller

was imperative that the Puma JSC in IWC is in complete control of the joint servos with no dependence on VAL. Actually, the VAL controllers associated with the robot arms were stripped of the servo amplifiers and power supplies for use in the IWC.

The kinematic analysis of the Puma type arm, using D-H frame assignment, has been widely studied [4.12]-[4.17] and is not discussed in this thesis. Parallelism in kinematic formulation of robot manipulators has been discussed by [4.18]-[4.19]. Forward and inverse kinematic formulation from [4.12] have been used in current work. The coupling in the wrist joints is discussed in [4.20]-[4.21]. The arm related data for Puma 260 and Puma 560 are given in Appendix C and Appendix D respectively.

In this chapter we restrict our discussion to Puma 260 interface to the IWC. Similar results for Puma 560 are included in Appendix E.

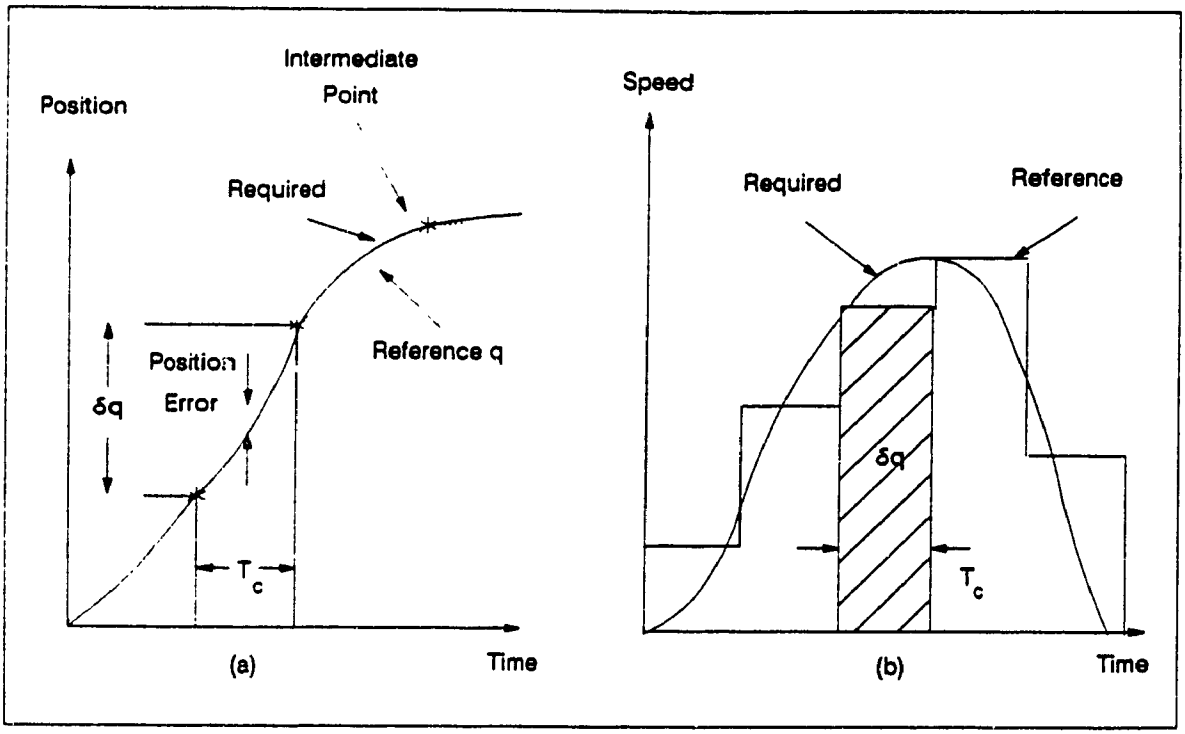

Figure 4.4 Absolute Interpolator

4.3 Basic Structure of an Interpolator

The term *interpolator* has been used by [4.1] to signify the algorithm used to determine the servo controller set points (joint space) from the desired cartesian position of the EE, as provided by the path planning algorithm. Though the term is widely used in CNC systems, its use in robotic systems is not very common. This came as a surprise to the author as the architecture of the interpolator plays a critical role in the performance of the robot system, as will become clear in a short while.

As shown in Figure 4.1, the inputs to the interpolator are the spatial position and/or velocity vectors as calculated in the path planing stage. The interpolator transforms these vectors into desired joint positions and/or speed, which are subsequently used as reference set points in the lower level servo controllers. The transformation, which in the

least requires the solution of inverse kinematic problem, is performed in each sampling interval T_c. An interpolator can be constructed in two ways, as discussed below.

4.3.1 Absolute Interpolator

In an absolute interpolator, using the inverse kinematic algorithm, the position vector **r** is transformed into the absolute joint angles **q**. At each sampling interval k, the **q(k)** calculated is sent as new reference position to the individual servo loops. It is **assumed** that at time $t=(k+1)T_c$, the joint will reach the absolute position **q(k)**. If the servo controllers require a velocity reference, it is generated by taking

$$\dot{q}(k) = \frac{\delta q}{T_c} \qquad (4.1)$$

where

$$\delta q = q(k) - q(k-1)$$

With absolute interpolator, there is no reference position error at the intermediate points, as shown in Figure 4.4(a). However, an error is induced between the intermediate set points because of constant speed profile over the sampling period. This error is dependent on the magnitude of δq and increases with longer sampling interval T_c or higher speed, as shown in Fig 4.4(b). Additional errors accrue because of truncations and approximation of transcendental functions. However, since the calculation of the next reference position is independent, the **errors do not accumulate from one iteration to next**. Needless to say that the implementation of the absolute interpolator requires an explicit and unique solution to the inverse kinematic problem. This is not a problem for Puma type arm as such a solution exists.

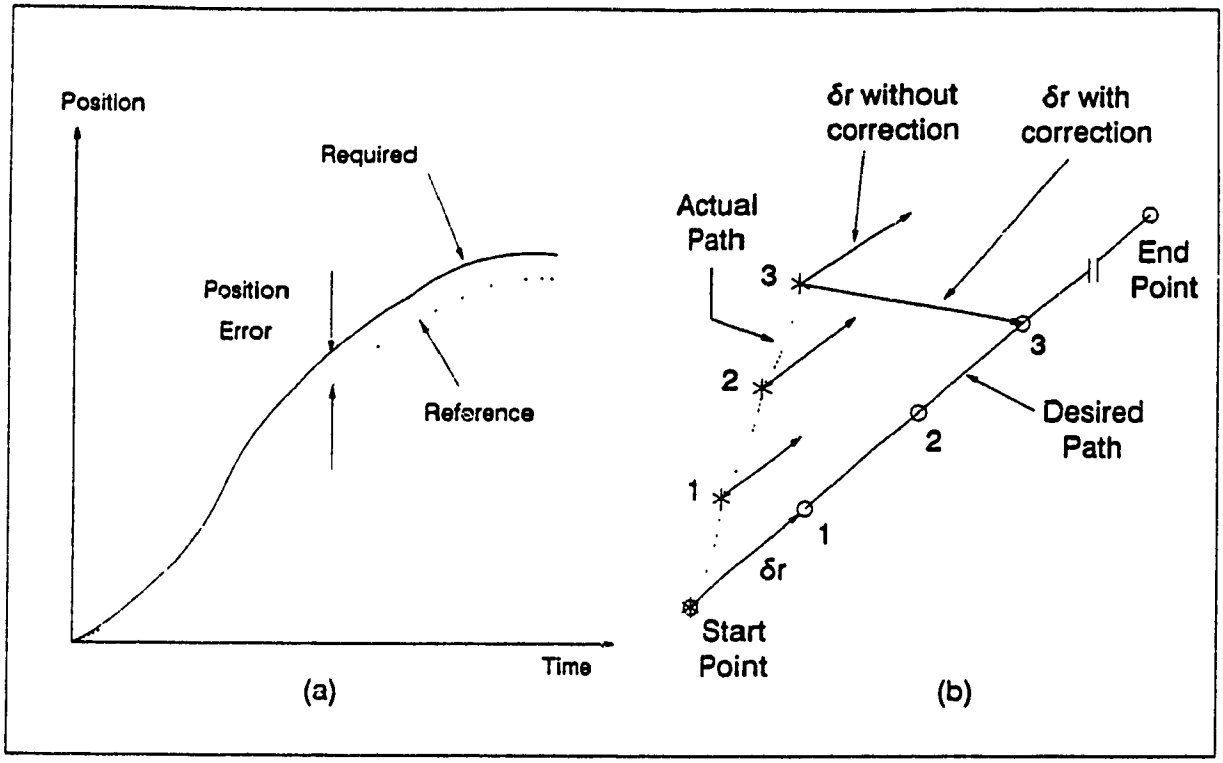

Figure 4.5 Incremental Interpolator

4.3.2 Incremental Interpolator

The incremental interpolator transforms the spatial velocity vector into the corresponding joint speeds using the manipulator Jacobian. A stair case approximation of the speed is used as reference to the servo controllers. Again, the reference speed remains constant for T_c period. If the servo controller requires a position reference, it is obtained by digital integration of speed i.e.,

$$q(k) = \sum_{j=1}^{k} \delta q(j) \qquad (4.2)$$

where

$$\delta q(j) = \dot{q}(j) T_c$$

The digital integration leads to an error between the required and reference speed, as shown in Figure 4.5(a), for each joint. This error leads to an **accumulative error in path following**. Consider a straight line path as shown in Figure 4.5(b). The $\dot{r}[k]$ is calculated by the path planning algorithm and represents the desired direction of travel. However, a position error is induced in the first cycle due to the reasons mentioned before. As a result, in the next cycle $\dot{r}[k+1]$ represents a direction parallel to the original path but not leading to the desired end point. This path following error is accumulative and increases with subsequent cycles. This error can be eliminated only by inserting cycles of absolute interpolation to converge the path back to the required path.

Since position errors are accumulative, the incremental interpolator is more sensitive to computational errors. Further, the scheme in this general form can be used only for a 6-DOF manipulator for which the jacobian is 6x6 and invertible. Also, the arm configurations which make the jacobian singular can lead to dangerously high values for joint speeds and necessary software checks have to be added to prevent this condition.

In both absolute and incremental interpolators, the reduction in spatial increment $\delta\mathbf{r}$ will lead to higher positional accuracy. This can be done by two means : reducing manipulator speed or reducing the sampling period T_c. It should be noted that additional path-following errors are caused by the position errors in the servo control loops and by mechanical inaccuracies.

Due to the problems associated with incremental interpolator, the JSC for Puma arms in the IWC uses absolute interpolation technique.

4.4 Servo Control Loop

Before going into the details of the implementation of the absolute interpolator, a discussion of the servo control loop is in order. The performance of the servo loop is of critical importance to the manipulator control. This is more so due to the open-loop operation of the interpolator as mentioned before. Thus, unless an external sensor system is used, any position/velocity error in the servo loop remains uncorrected.

Like any digital control system, the performance of the servo control loop improves with a decrease in sampling period T_s. T_s has to be in sub millisecond for low overshoot and fast response. However, a low T_s restricts the choice of the control algorithms which can be used in the servo controller. Typically, a tuned PID controller assures a low T_s and acceptable performance. To save on the development time, an off-the-shelf servo controller providing the listed features is used in the IWC and described below.

4.4.1 LM629 Motion Controller

The LM629 Precision Motion Controller [4.22] from National Semiconductor is a dedicated processor designed for use with a variety of DC and brushless DC servo motors, with quadrature incremental position feedback. The chip has a very comprehensive high level command set, through which the host (transputer) can program it to perform specific motion control actions. LM629 provides an 8-bit PWM output which is amplified and fed to the motor terminals. Figure 4.6 gives the block diagram of a LM629 based DC motor control system.

LM629 has two modes of operation i.e. **position mode** and **velocity mode**. In

position mode the host processor (JSC) loads the desired final position of the motor shaft, in terms of number of encoder counts (henceforth called counts only) it should travel, in a 32-bit position register. Also the maximum velocity of travel (in counts/sample period) and desired acceleration (in counts/sample/sample) are loaded into 32 bit velocity register and acceleration register respectively. The LM629 then generates a **trapezoidal velocity profile** by accelerating at the commanded rate until maximum programmed velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. Figure 4.7(b) shows a typical trapezoidal velocity profile generated by LM629. Figure 4.7(a) shows the corresponding position profile which is **linear with parabolic blends**. Figure 4.7(c) shows the corresponding acceleration profile which is of on/off type. At any time during the move the maximum velocity and/or the target position may be changed, and the LM629 will cause the motor to accelerate or decelerate accordingly. Figure 4.7(d) shows an example of a trajectory when the velocity and position are changed at different times during the move. In the velocity mode, the motor accelerates to the desired velocity at the specified acceleration rate and then maintains the specified velocity until commanded to stop. The velocity is maintained by advancing the desired position at a specified rate.

As mentioned before, all trajectory parameters like position, velocity and acceleration are 32 bit values. Position is signed value to determine the direction of motion. Acceleration and velocity are specified as 16-bit positive-only integer having 16-bit fractions. The integer portion of velocity specifies the number of encoder counts the motor shall travel in one sampling period. The fractional portion designates additional

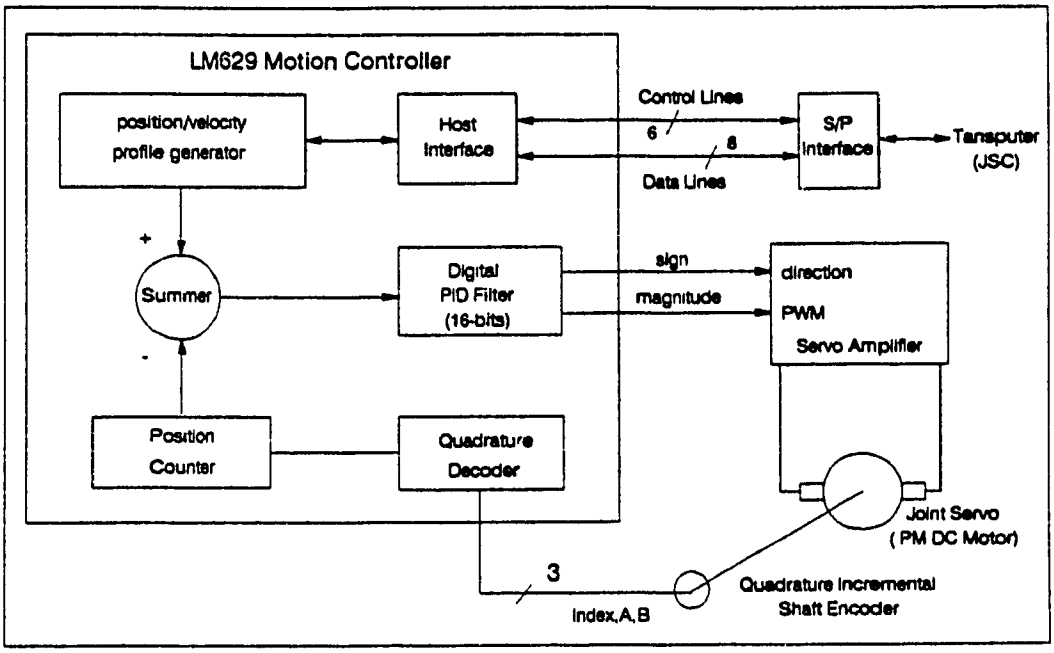

Figure 4.6 LM629 based DC motor control system

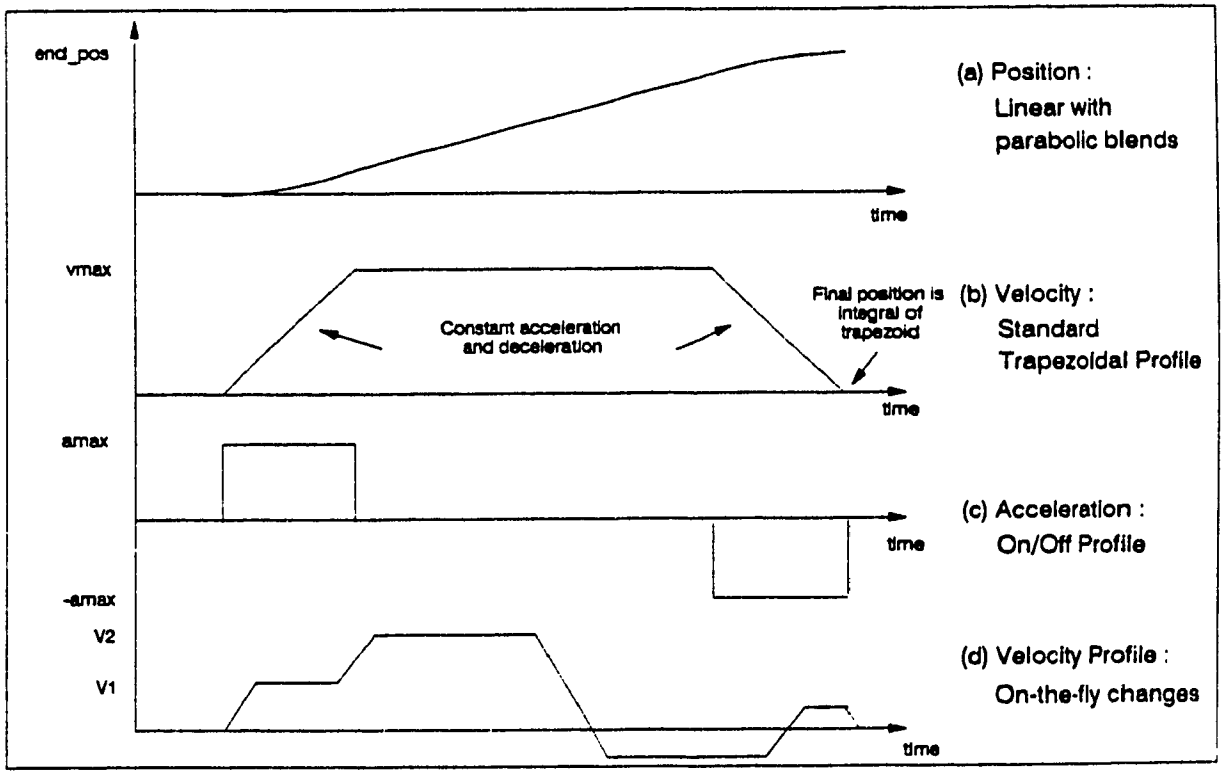

Figure 4.7 LM629 trajectory generation

fractional counts per sampling period. Although, the position resolution of the motion control system is restricted to integer counts, the fractional counts provide increased resolution for average velocity. Acceleration is treated in the same manner.

As shown in Figure 4.6, the LM629 has a digital **Proportional Integral Derivative (PID)** filter to compensate the control loop. The motor is held at the desired position by applying a restoring force proportional to the error, plus the integral of the error and the derivative of the error. The following discrete-time equation illustrates the control performed by the LM629

$$u(n) = K_p e(n) + K_i T_s \sum_{n=0}^{N} e(n) + \frac{K_d[e(n') - e(n'-1)]}{T_d} \qquad (4.3)$$

where u(n) is the motor control signal output at the sample time n, e(n) is the position error in terms of encoder counts at sample time n. K_p, K_i, and K_d are the filter gains. n' indicates sampling at the derivative sampling rate of T_d. T_d can be chosen to be (1..255) times T_s. At a input clock of 8MHz, used in the current design, T_s is **256 microseconds.**

In actual operation, three 16-bit registers are provided in the chip for programming the filter gains called the k_p, k_d and k_i registers. In terms of these register values, Eq.(4.3) can be written as

$$u(n) = k_p e(n) + k_i \sum_{n=0}^{N} e(n) + k_d[e(n') - e(n'-1)] \qquad (4.4)$$

The 16-bit filter coefficients are respectively multiplied by the 16-bit error, as in the above equation, to produce 32-bit product. However, as the chip provides only an 8-bit drive signal, u(n), only selective bytes from the 32-bit products of each term are added. This is discussed in detail in [4.23]. As a result, the coefficients used in the LM629 are

scaled in the following manner :

$$k_p = 256xK_p \tag{4.5a}$$

$$k_d = 256xK_d/T_d \tag{4.5b}$$

$$k_i = 65536xK_iT_s \tag{4.5c}$$

Some useful hints for programming the LM629 are given in [4.24].

4.4.2 Iterative Tuning of the PID filter

The tuning of the PID controller, for each joint, is essential for high positional accuracy and a fast response. For a robotic manipulator control, it is desirable to have each servo loop critically damped so as to prevent any position overshoot with reasonably fast response. An analytical technique can be employed by modelling each joint servo independently and then using suitable criterion to find the optimum gains. However, assuming the servo loops to be totally decoupled fails to account for the significant variations in the model parameters due to manipulator dynamics, as the arm takes different configurations during motion.

Hence, an iterative procedure was used for experimental determination of the PID gains. In the technique devised, the optimum gains for a particular servo are identified while the complete arm is in motion so as to include the dynamic effects to the best possible. For this, the same trajectory is loaded in all the servo controllers and executed repeatedly a number of times. As an example, a typical trajectory may be clockwise joint motion through 40 degrees in 5 seconds and return. While this trajectory is being executed, the k_p-k_i gains of the servo controller being tuned are varied, from one iteration to next, in a wide range. During each iteration the cumulative errors in position and

velocity are evaluated for the joint being tuned, using the following

$$ise\_pos[n] = ise\_pos[n-1] + (\text{desired position} - \text{actual position})^2 \quad (4.6a)$$

$$ise\_vel[n] = ise\_vel[n-1] + (\text{desired velocity} - \text{actual velocity})^2 \quad (4.6b)$$

where the desired/actual joint position is measured using the corresponding LM629 commands. Position increment over measured intervals of time is used to evaluate the joint velocity. The actual number of times Eq.(4.6) gets executed in a particular iteration is immaterial as the number is the same from one iteration to next. Further at the end of each iteration, when all the joint trajectories have been completed and the arm is stationary, the steady state error in position is measured for the joint being tuned using

$$ss\_pos = |\text{desired final position} - \text{actual final position}| \quad (4.6c)$$

Using the data recorded, a 3D contour plot is made for each of the error variable. Figure 4.8 shows one such plot, ss_pos vs. k_p-k_i, for Puma 260 joint #1 where k_p,k_i refer to the actual coefficients use in the LM629 controller. For this plot, k_p was varied from 1000 to 2600 in steps of 200 and k_i was varied from 0 to 100 in steps of 20. It can be seen that the steady state position error decreases with increasing value of k_p and decreasing value of k_i. From the plot a smaller region with uniformly low error is selected. For Figure 4.8, a k_p range of 1800-2200 and k_i range of 0-20 gives a uniformly low error of less than 0.0002 radians which corresponds to an error of 1 encoder count at the servo level. Higher value of k_p is not preferred for reasons to be discussed in a short while.

For verification and fine tuning of the parameters, the procedure is repeated for the selected range with a better resolution. Figure 4.9 shows the plot obtained from a second trial for joint #1 when k_p-k_i are varied in the aforementioned ranges with steps of

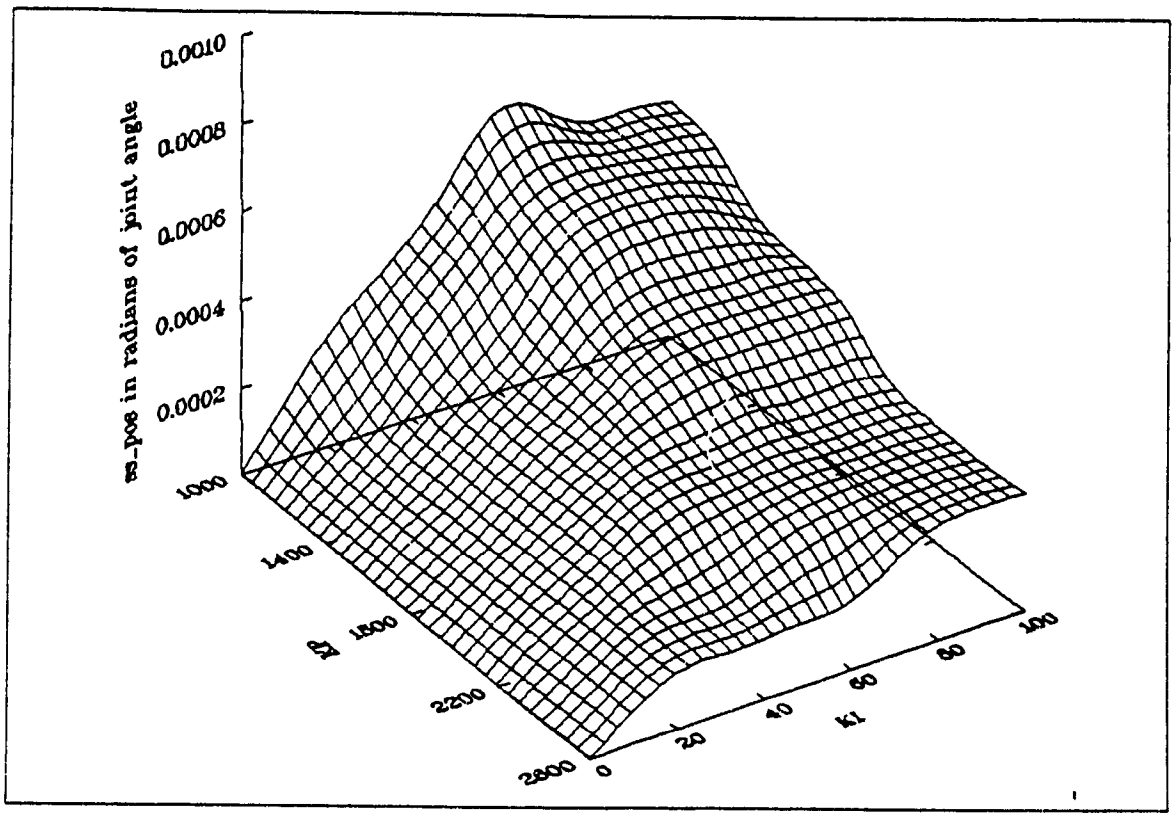

Figure 4.8 Joint #1 steady state error as a function of k_p-k_i.

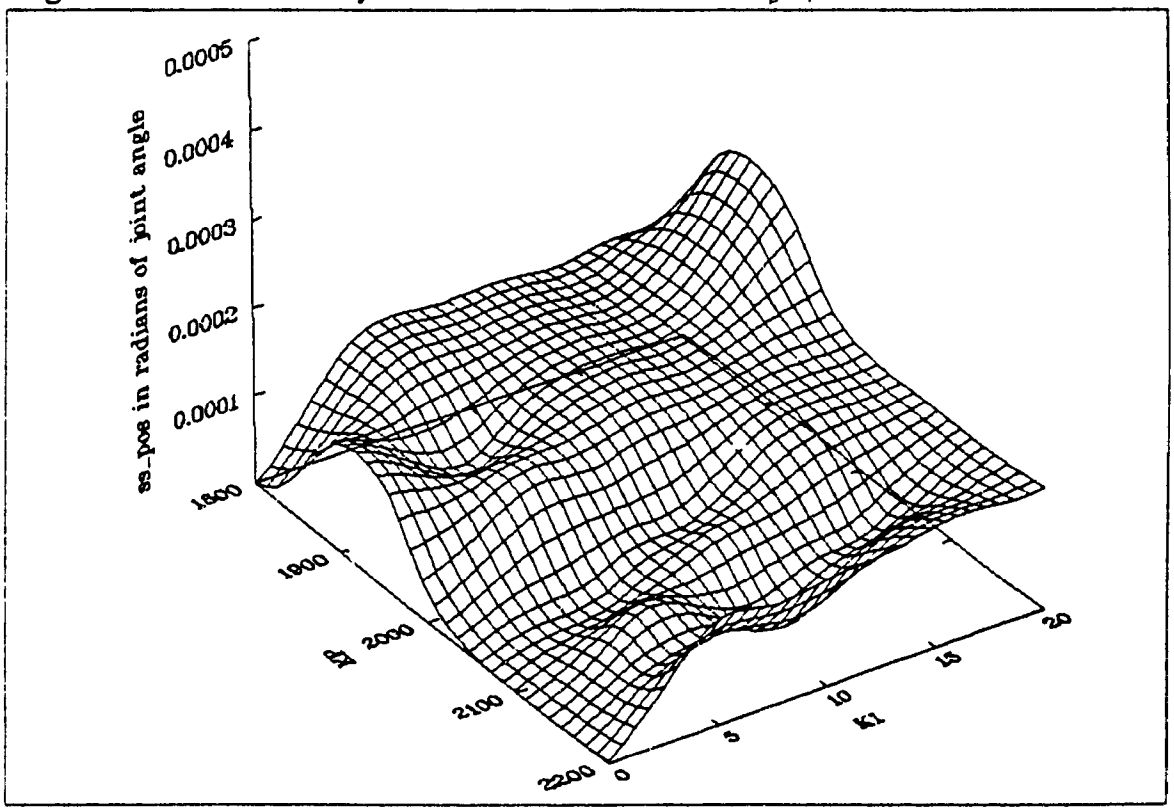

Figure 4.9 PI tuning for Joint #1

40 and 5 respectively. As expected, the steady state error of less than or equal 0.0002 radians is verified. It can be seen that a k_p-k_i pair of 2150-10 corresponds to *valley* with zero error. Hence, k_p of 2150 and k_i = 10 are chosen for joint #1 PID controller. Using Eq.(4.5), these correspond to actual gains of K_p = 8.40 and K_i = 0.60.

This procedure is repeated for finding the optimum k_p and k_i for each joint. The following observations were made during repeated trials :

(i) For all joints, the cumulative position error ise_pos and the cumulative velocity error ise_vel decrease in a monotonic way with increasing value of k_p-k_i. Figure 4.10, which shows the variation in ise_pos for joint #3 of Puma 260, is a representative plot. Hence, steady state error ss_pos becomes the critical criterion for determining optimum k_p-k_i values in the manner discussed before.

(ii) As expected, the servo loop becomes unstable at high values of k_p. This is characterised by high vibration and a distinct cracking sound, as if the gears are cutting into each other due to frequent reversal of motor direction. Hence a higher (than required) value of k_p is not preferable as it may lead to mechanical damage in the long run.

(iii) The performance of the joint servos is marginally different for the two directions of motion (clockwise and anticlockwise). This may be due to asymmetric mechanical assembly leading to different loading in the two directions. This phenomenon can be compensated by selecting different gain for each direction. However, loading new PID gains into the servo controller every time the direction is changed leads to considerable data transfer overhead within a sampling interval

and is not implemented.

Once optimum k_p-k_i gains are selected for all the joints, k_d is tuned so as to minimize cumulative error in velocity ise_vel using a similar technique. The derivative sampling period T_d is taken as 2.048 msec ($8T_s$). This is so taken because, as will be clear in section 4.7, even with highest manipulator speed the joint servos operate with fractional desired velocity of less than 1 count/T_c. Hence a longer T_d is required to make any reasonable estimate of velocity error.

Table 4.1 gives the experimentally determined optimum values of controller gains for Puma 260. As can be seen, the determined values of K_d are negligible. The problems associated with the derivative controller term will be further discussed in section 4.7. Figure 4.11 shows the response of Joint #2 servo when loaded with a trajectory to move 22 encoder counts in 28.16 msec, which for reasons to be discussed later, is taken as the controller sampling period T_c. As will become clear in section 4.7 a displacement of 20-22 counts in one sampling period T_c is about the typical required for any joint servo even for applications requiring high manipulator speed. As T_c = 110 T_s, a displacement of 22 counts corresponds to an **average fractional velocity** of 0.2 count/sample period T_c. It is seen from the figure that with PID gains determined before, the desired position is achieved in one T_c. **The steady state error is plus or minus 1 count.** An **overshoot of ~ 10%** is also observed. Although critical damping is desirable, reducing K_p to reduce overshoot also increases the rise time leading to a failure in trajectory completion in time T_c. This leads to error accumulation as the interpolator evaluates the trajectory for the following sampling interval under the assumption th the

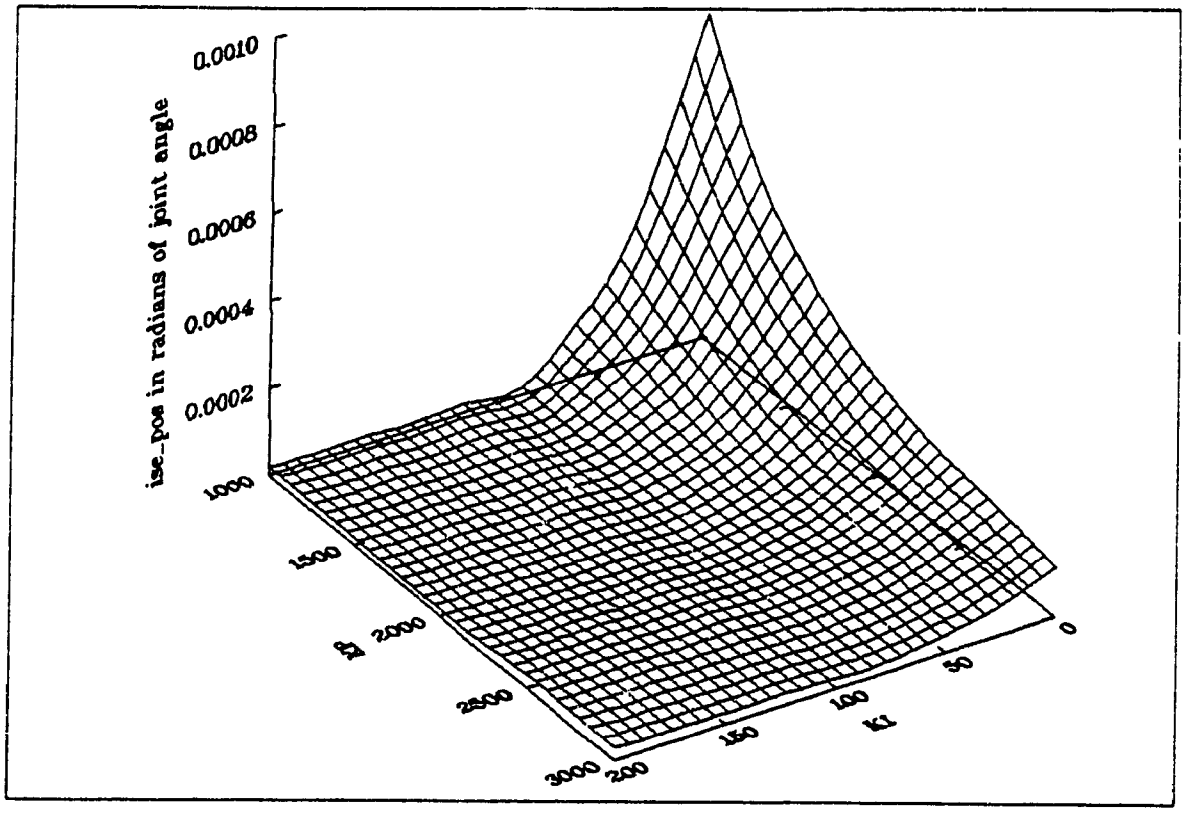

Figure 4.10 Cumulative position error as a function of k_p-k_i.

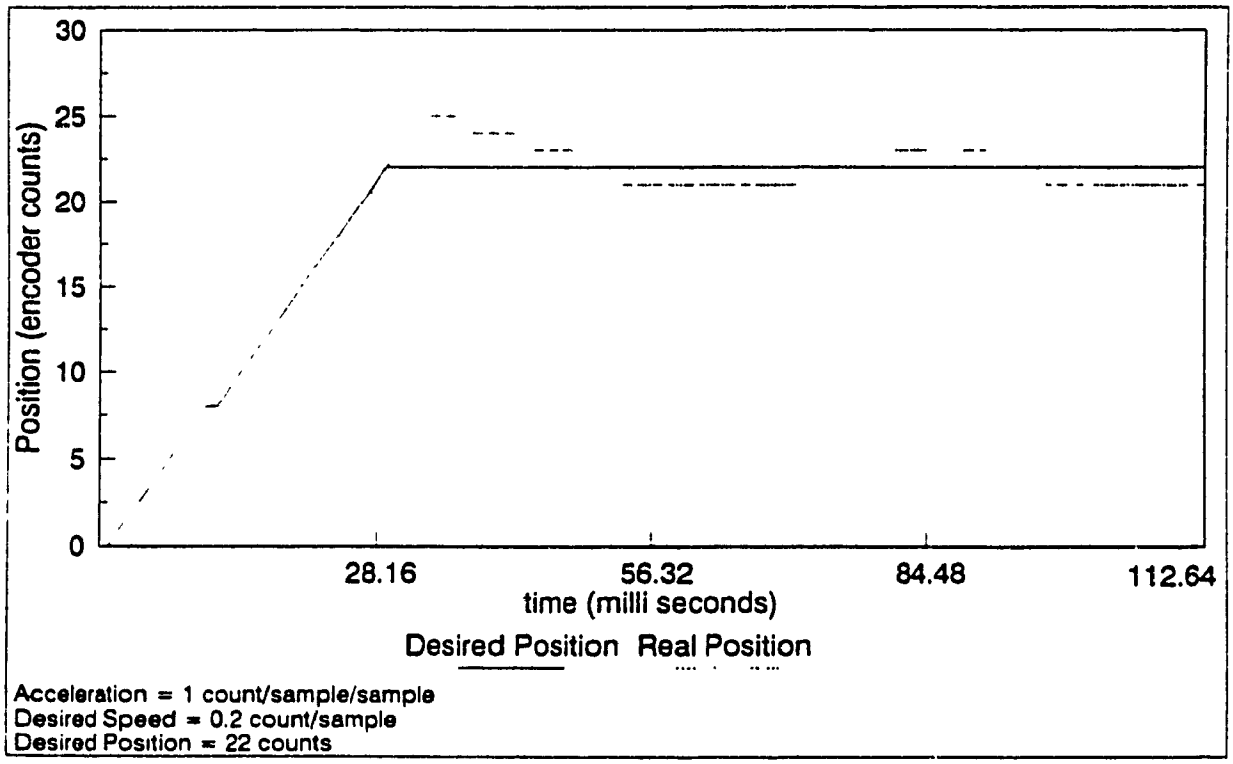

Figure 4.11 Joint #1 response for a typical motion trajectory.

Table 4.1 PID Gains for Puma 260

PUMA Joint	LM629 k_p	Actual K_p	LM629 k_i	Actual K_i	LM629 k_d	Actual K_d
1	2150	8.40	10	0.60	80	0.0003
2	2250	8.80	110	6.60	50	0.0002
3	2050	8.00	110	6.60	60	0.00025
4	1000	3.90	20	1.20	150	0.0006
5	1000	3.90	17	1.00	100	0.0004
6	1000	3.90	25	1.50	260	0.001

last set point has been reached which is really not the case.

4.4.3 Some Limitations of LM629

As is expected, the LM629 architecture also imposes some restrictions on the JSC design. Some of the limitations of LM629 command set and their effects are :

(i) The desired acceleration loaded at initialisation can not be changed on-line. Any change in acceleration register requires a software reset making the drive signal zero. This will make the arm fall freely under gravity. Due to this, a sufficiently high initial acceleration is loaded and used for complete manipulator motion. Otherwise, out of the 110 T_s available (during one T_c), fixed number of intervals could have been allotted for acceleration and deceleration.

(ii) There is no command for initialising the 32-bit register for real position of motor shaft. Due to this, a one-to-one mapping between the real position count and the joint angle can not be established. This is discussed further in the following section.

(iii) There is no command to STOP on observing an index pulse, though a flag

indicating so is provided. Such a command is useful for arm calibration when the joint is locked on an index pulse and the corresponding joint angle is read from a look up table. Now, the transputer has to poll for the index flag and load the LM629 command to STOP motor when the flag is TRUE. Calibration repeatability suffers due to this round about procedure.

4.5 Implementation of Absolute Interpolator

The architecture of an absolute interpolator was discussed in section 4.3.1. As a typical example, this sections describes the implementation of the same for joint #1 of Puma 260 using LM629 servo controller. We start by looking at the way joint angle θ changes over a full rotation of joint #1.

The *atan2(x,y)* function used in the inverse kinematic formulation returns a value θ between $[-\pi,\pi]$. Hence, it is desirable that the joint angles be so evaluated (from shaft position in encoder counts) such that they lie in the range $[-\pi,\pi]$ instead of $[0,2\pi]$. This can be assured in a straightforward manner. Let ${}^{a}\theta_k$ be the actual joint #1 angle at the start of k^{th} sampling interval i.e. $t \in [kT_c, (k+1)T_c]$. Let ${}^{a}\delta\theta_k$ be the actual joint #1 rotation during the k^{th} interval. ${}^{a}\delta\theta_k$ is evaluated by

$$ {}^{a}\delta\theta_k = 0.1345 \times 10^{-3} \, {}^{a}\delta e_k \qquad (4.7) $$

where ${}^{a}\delta e_k$ is the actual shaft rotation in encoder counts and 0.1345×10^{-3} is radian/count ratio for joint #1 (refer Appendix C). Using,

$$ {}^{a}\theta_{k+1} = ({}^{a}\theta_k + {}^{a}\delta\theta_k) + 2\pi \qquad \text{if } ({}^{a}\theta_k + {}^{a}\delta\theta_k) \le -\pi \qquad (4.8) $$

$$ {}^{a}\theta_{k+1} = ({}^{a}\theta_k + {}^{a}\delta\theta_k) - 2\pi \qquad \text{if } ({}^{a}\theta_k + {}^{a}\delta\theta_k) \ge \pi $$

${}^{a}\theta_{k+1}$ can be determined such that it lies in the range $[-\pi,\pi]$.

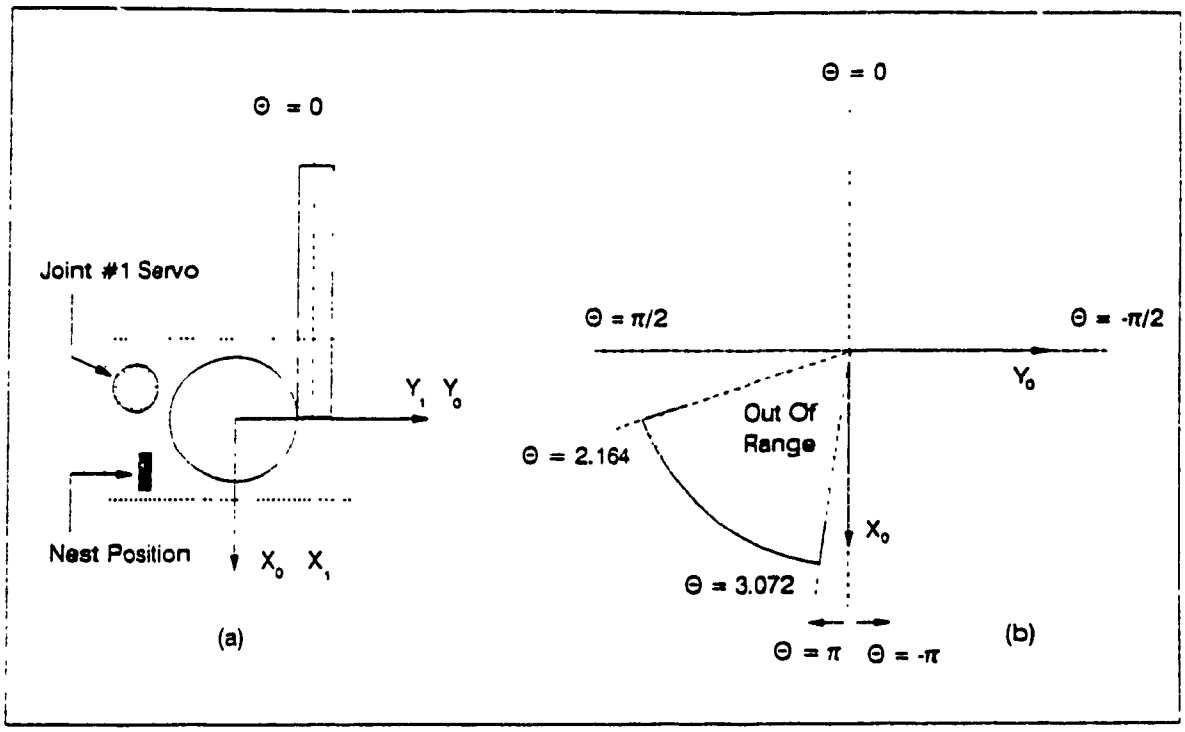

Figure 4.12 Variation in joint angle #1 over full rotation

A direct consequence of using $[-\pi,\pi]$ range is that the joint angle does not change linearly with joint rotation over the full range. Figure 4.12 illustrates this for joint #1. Figure 4.12(a) shows the position of joint #1 and the attached coordinate frames for $\theta = 0$, as seen from the top. The (X_o, Y_o, Z_o) is the base coordinate frame. Recall that for joint #1, θ is the angle of rotation required about Z_o axis to align X_o to X_1 using the right hand rule. Figure 4.12(b) shows the variation in θ as the joint #1 is rotated between the two extreme limits where the arm is not shown for the sake of clarity. The following observations are of importance :

(i) Full range of $[-\pi,\pi]$ is not available. This is true for most of the joints of both Puma 260 and Puma 560. The valid range in the present case is

$-\pi \leq \theta \leq 2.164$

$3.072 \leq \theta \leq \pi$

(ii) As the joint turns anticlockwise from one extreme to other, θ varies in the following manner

$3.072 \rightarrow \pi/-\pi \rightarrow -\pi/2 \rightarrow 0 \rightarrow \pi/2 \rightarrow 2.164$

This abrupt change of joint angle from π to $-\pi$ (and vice versa) is typical of the first three joints of both the robots.

In section 4.3.1, we saw that in the k^{th} interval, the absolute interpolator evaluates ${}^{d}\theta_{k+1}$ and downloads it as the reference position at the start of the $k+1^{th}$ interval. To download ${}^{d}\theta_{k+1}$, it has to be converted in terms of desired shaft position in encoder counts. One way to do this is make $\theta = 0$ correspond to zero value of position counter, and establish a one-to-one correspondence between the joint angle and the position count. Then the absolute position count corresponding to ${}^{d}\theta_{k+1}$ can be downloaded as the next reference and the servo controller is run in *absolute position mode*, where it moves from one absolute position to next. However, as mentioned before, the non-availability of a command to initialise the real position counter prevents the implementation of this mode by using LM629.

The other option is to evaluate ${}^{d}\delta\theta_{k+1}$ and load it as an incremental displacement (relative to the current position) desired in the $k+1^{th}$ interval. However, with $\pi/-\pi$ switch the evaluation of ${}^{d}\delta\theta_{k+1}$ is no more as simple as ${}^{d}\theta_{k+1} - {}^{d}\theta_{k}$ (and not ${}^{a}\theta_{k}$). The correct ${}^{d}\delta\theta_{k+1}$ for joint #1 is given by the following *decision equations.*

$$
\begin{aligned}
{}^{d}\delta\theta_{k+1} &= {}^{d}\theta_{k+1} - {}^{d}\theta_{k} && -\pi \le {}^{d}\theta_{k},\ {}^{d}\theta_{k+1} \le 0 \\
& && 3.072 \le {}^{d}\theta_{k+1},\ {}^{d}\theta_{k} \le \pi \\
& && 0 \le {}^{d}\theta_{k+1}, {}^{d}\theta_{k} \le 2.164 \\
& && 0 \le {}^{d}\theta_{k+1} \le 2\ 164 \text{ and } -\pi \le {}^{d}\theta_{k} \le 0 \\
& && -\pi \le {}^{d}\theta_{k+1} \le 0 \text{ and } 0 \le {}^{d}\theta_{k} \le 2.164 \\
&= {}^{d}\theta_{k+1} - {}^{d}\theta_{k} + 2\pi && 0 \le {}^{d}\theta_{k+1} \le 2.164 \text{ and } 3.072 \le {}^{d}\theta_{k} \le \pi \\
& && -\pi \le {}^{d}\theta_{k+1} \le 0 \text{ and } 2.164 \le {}^{d}\theta_{k} \le \pi \qquad (4.9) \\
&= {}^{d}\theta_{k+1} - {}^{d}\theta_{k} - 2\pi && 3.072 \le {}^{d}\theta_{k+1} \le \pi \text{ and } 0 \le {}^{d}\theta_{k} \le 2.164 \\
& && 2.164 \le {}^{d}\theta_{k+1} \le \pi \text{ and } -\pi \le {}^{d}\theta_{k} \le 0
\end{aligned}
$$

Using count/radian ratio, the desired rotation in encoder counts for joint #1 is given by

$$ {}^{d}\delta e_{k+1} = 7435.72\ {}^{d}\delta\theta_{k+1} \qquad (4.10) $$

For a controller sampling period of 28.16 msec (= 110 T_s), the reference velocity is given as

$$ {}^{d}\dot{e}_{k+1} = {}^{d}\delta e_{k+1}/110 \qquad (4.11) $$

Once ${}^{d}\dot{e}_{k+1}$ and ${}^{d}\delta e_{k+1}$ are determined, the joint servo can be operated in two modes. If operated in *relative position mode* the LM629 will generate a trapezoidal profile, using programmed acceleration and desired velocity, to move the shaft to the new reference position evaluated as

$$ {}^{d}e_{k+1} = {}^{d}e_{k} + {}^{d}\delta e_{k+1} \qquad (4.12) $$

It is important to note that as increment is added to last reference ${}^{d}e_{k}$ (and not the actual position ${}^{a}e_{k}$), the position error does not accumulate. However in position mode the LM629 will attempt to stop the motor on trajectory completion i.e. every T_c time interval.

Instead of position, ${}^{d}\dot{e}_{k+1}$ can be used as a velocity reference using the *absolute velocity mode* of the motion controller. In this mode, refer 4.4.1, the LM629 will attempt to attain and maintain the downloaded velocity till a new reference velocity is loaded. The motor will not be stopped every T_c time period. However, the actual shaft rotation will be

$$ {}^{a}\delta e_{k+1} = {}^{a}\dot{e}_{k+1} T_c \tag{4.13} $$

which may be different from the desired due to error in speed regulation, ${}^{d}\dot{e}_{k+1} - {}^{a}\dot{e}_{k+1}$, and the approximation in Eq. (4.11). Because velocity is used as the reference, the reader should not confuse this approach with incremental interpolation. The velocity reference signal is still determined from ${}^{d}\delta\theta_{k+1}$. The following section presents a comparison between the two modes of operation from experimental results.

The absolute interpolator for first three joints of both Puma 260 and Puma 560 are implemented on similar lines. For wrist joints, some modifications have to be made. Essentially, equations (4.7) and (4.10) are changed to accommodate the coupling between the joints. Also, for the joints which have a wider range (more than 2π) of motion, Eq.(4.8) and (4.9) are modified suitably. Although implemented, the details of the wrist control are not included in this thesis.

4.6 Experimental Results

Although the design of a robot controller is not the central theme of the research undertaken, its performance plays a critical role in achieving the end goal, i.e. multi-manipulator synchronisation. Obviously, no synchronisation algorithm can succeed if the

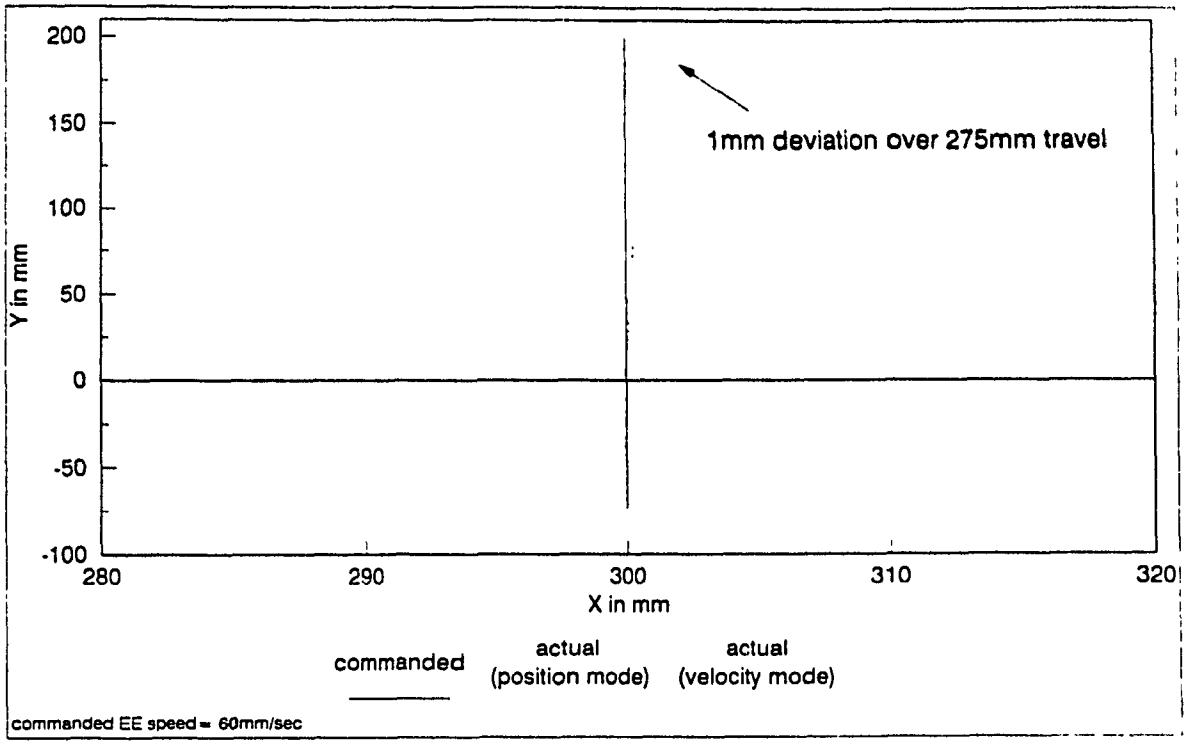

Figure 4.13 Puma 260 motion along a straight line along Y direction.

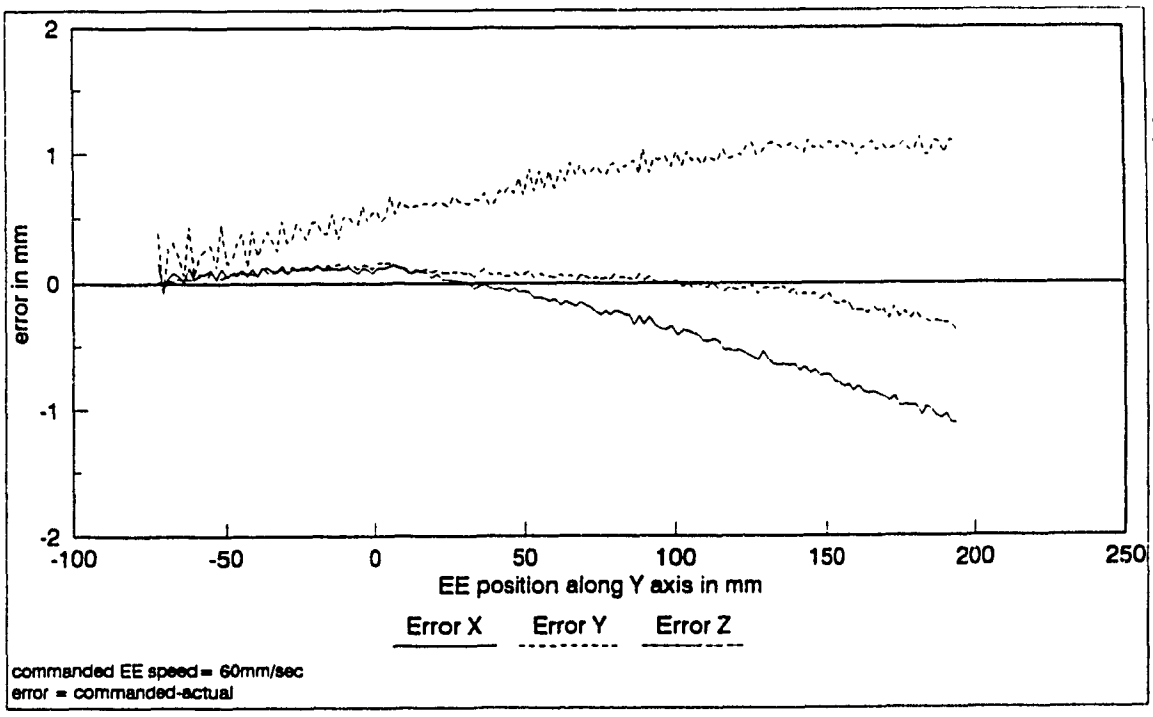

Figure 4.14 Error in straight line motion using velocity mode.

JSC of any participating arm does a bad job leading to large error between desired and actual end effector position.

Figure 4.13 shows the actual path traced by the Puma 260 EE when commanded to move in a straight line along the world Y axis at a speed of 60mm/sec. The orientation of the EE is kept fixed (vertically down). It should be noted that with a constant orientation, straight line motion along Y axis required synchronised control of all the six joints. As can be seen from the plot, **the actual path offsets from the desired path by about 1mm over a total travel of 275mm.** Further, the position mode of absolute interpolator gives only a marginally better performance than the velocity mode. Although the motion in position mode is smooth, still the frequent attempt to stop/start each servo will cause greater mechanical damage in the long run with no great improvement in performance. Hence, it is not the preferred mode of operation. All the experimental results reported in subsequent chapters are taken under velocity mode, which is implemented as the default mode of JSC operation.

Figure 4.14 shows the errors along each coordinate for straight line path discussed above under velocity mode. It is seen that the error along x-y coordinates increases as the EE moves further along the Y axis to a maximum of ≈ 1mm. This increase in error can be attributed to two factors. First, as the EE moves far along the Y axis, link #2 and link #3 are more and more extended. This leads to increased torque exerted on joint #2 due to gravity. The purely kinematic algorithm implemented can not compensate for such effects due to manipulator dynamics. The second factor is due to the simple phenomenon that a vector when rotated by say 1 degree will span twice the arc length if its length is

doubled. Thus, any steady state error in joint #1 servo gets more and more magnified as the EE moves away from the origin of the base coordinate frame.

The speed of the EE at which the observations are made, i.e. 60mm/sec, is about the maximum for majority of continuous path applications like welding etc. As explained in section 4.3, the position accuracy will improve with lower speed. The EE orientation control was found to be sufficient for the application at hand.

4.7 Determination of Controller Sampling Period

It has been mentioned several times before that the controller sampling period T_c is taken to be **28.16 msec**. This section discusses the reasons behind the particular choice. Generally, the computational time required by the interpolator and the higher level path planning is quoted as the single key factor limiting the T_c period. As a result, considerable research has been directed towards the use of parallel processing schemes to reduce the computational time. However, with the increasing speed of processors, this is becoming less of a limitation day by day. As an example, using the T-800 transputer at 20 MHz clock, the interpolator algorithm takes about **12msec**, when executed in a purely serial fashion on the JSC. This involves sequential evaluation of forward kinematics (to get the current arm configuration), inverse kinematics and decision equations for each joint. As a natural choice T_c = 14msec was taken in the first attempt, which to the authors surprise failed to give good results. The reason for the failure becomes clear from Figure 4.15.

The figure shows the desired incremental rotation (in encoder counts) for the first three joints of Puma 260 for a T_c = 28.16 msec, when the arm is moving along the Y axis (between Y = 40-100 mm) in a straight line path as discussed above. These coordinates

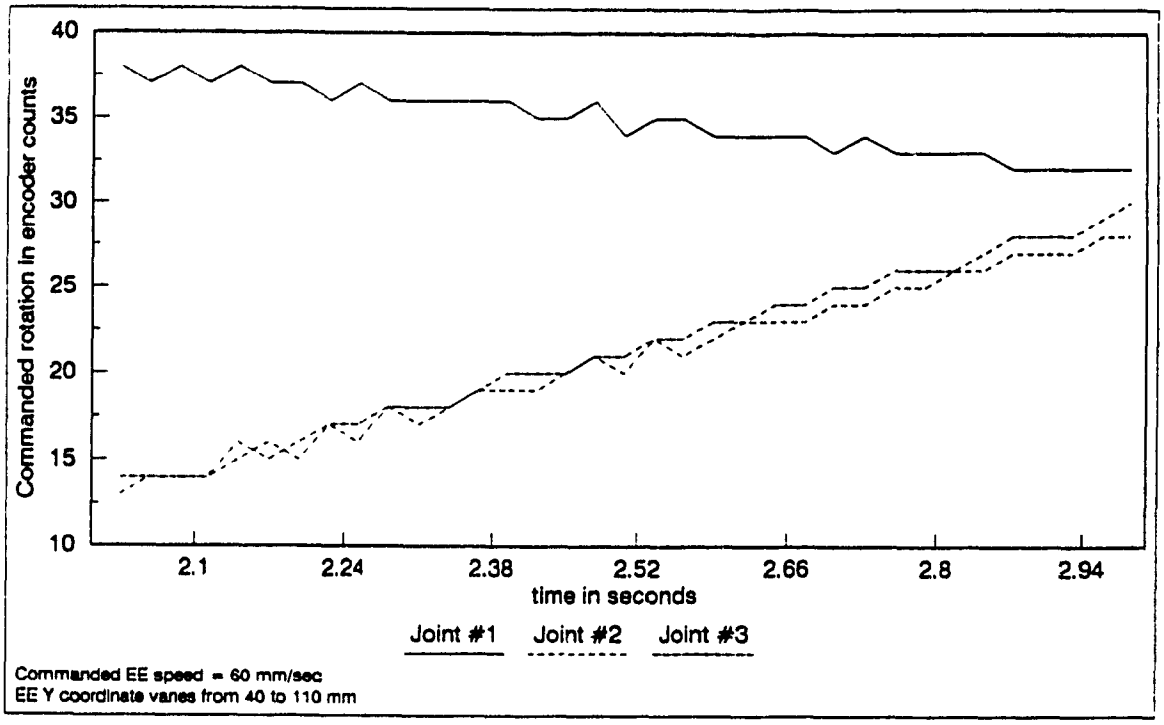

Figure 4.15 Commanded incremental rotation for joints #1-#3.

correspond to the preferred region of operation as it is well inside the *working envelope* of the Puma 260 arm. The commanded speed of the EE is 60mm/sec. It can be seen that inspite of a high desired speed, the commanded rotation is **less than 40 counts.** This is already a small trajectory from servo controller point of view and requires a fine tuning of the PID for proper execution. At lower arm sampling interval, these increments will be even lower. The particular value of 28.16msec (=110 T_s) is chosen as it is an integral multiple of servo controller sampling period T_s.

A lower T_c can be used if the resolution of the joint encoders is increased. If the shaft encoders mounted currently on the joint motors (250 line) are replaced by 1000 line encoders, the desired increments will increase four fold. Then, a T_c = 12msec can be used assuming 30-40 mm/sec as the average speed of operation. To conclude, one can say that

the choice of T_c is influenced by the joint angle resolution, average speed of operation and (to a lesser extent) by the computational time required by the path planning algorithm. Another option is the **on-line tuning of T_c** depending upon the speed of operation. This is not possible with the conventional controller architectures as T_c is determined by a hardware clock. However, as will be shown in 6.3.2.1, the hardware clock is emulated in the IWC by a **timer thread** running at GTPC level. This permits one to tune the T_c on-line by passing a different parameter to the thread for *timer_wait* so as to give best performance for the task being executed.

The small increments lead to low **fractional desired velocity** which can be regulated only in an average sense. An increment of 40 counts in one T_c corresponds to an average speed of ≈ 0.4 count/sample. With an actual resolution of 1count/sample, any derivative control during one T_s looses its significance. Due to the same reasons, a large derivative sampling period T_d of 2.048 msec. is chosen.

4.8 Implementation of Device-Independent Commands

It has been mentioned before that the JSC provides a uniform interface to the higher levels by supporting a set of device independent commands. This section gives some illustrative examples on the implementation of the same.

(i) **MVEL_6JT** : This is the command for operating the arm in velocity mode, as discussed before. The commanding processor (from CPC level) sends the messages shown in the box on the following page. The *EE_des_loc* structure specifies the desired end effector position and orientation. The JSC, using the kinematic algorithms for the particular arm under its supervision, guides the arm

```
cmd_type = MVEL_6JT;
send_header(TOJSC);
chan_out_message(EE_des_loc, out_port[TOJSC]);
```

Communication for MVEL_6JT.

so as to reach the desired location in time T_c. Absolute interpolator with velocity mode is used. Position mode can be selected by using a similar command **MPOS_6JT**.

(ii) **EE_LOC** : As the name suggests, in response to this command the JSC returns the EE position and orientation in the *global coordinate frame*. Forward kinematics algorithm for the particular manipulator is used in determining these. The end effector location is returned in a structure *EE_act_loc*.

(iii) **CALIBRATE** : In response to this command, the JSC calibrates the arm under its control. Calibration procedure differs from manipulator to manipulator. As an example, in Puma 260, which does not have potentiometers, it requires initialising the joint angles when the arm is NEST position and bringing it out of it by particular sequence of motion commands. On the other hand, Puma 560 is calibrated by reading the potentiometer values and determining the joint angles using look up tables. In any case, JSC returns an acknowledgement flag to the commanding processor if the arm calibration is successful.

(iv) **TRAJ_COMP** : In response to this command, the JSC waits till the trajectories loaded in the servos specified in the *channel_no* field are successfully completed and returns an acknowledgement flag when it is so. This is required for point to

point motion required in applications like pick and place etc.

Appendix A gives a list of such commands available in the IWC in the current stage of development.

4.9 Summary

The critical issues involved in the design of a Joint Space Controller, within the framework of IWC architecture, are highlighted in this chapter. From the discussion, the following steps can be identified for integrating a new arm to the workcell :

(i) A suitable servo controller is selected depending on the nature of the joint actuators and the type of control action (position/torque) desired.

(ii) Tuning of the servo loops for optimum performance.

(iii) Using the kinematic formulation of the manipulator, the interpolator algorithm for conversion from task space to joint space is implemented. If computer torque control is used then the dynamic model of the manipulator is used for determining the torque required from the joint servos.

(iv) Once the manipulator has been interfaced, suitable drivers are incorporated in the JSC to support the device independent commands and provide a uniform interface to the higher levels.

Lastly, it should be ensured that the resolution of the feedback sensors is high enough to support the sampling period T_c for the average speed of operation required in the application at hand.

Chapter 5

Path Planning in Cartesian Space

5.1 Review of Path Planning Algorithms

The relation between the *trajectory* and the *path* of a manipulator is [5.1] :

A trajectory is the time sequence of intermediate configurations of the manipulator between the source P_0 and the destination P_f. The space curve traced by the end effector is called the path of the trajectory.

The manipulator path is described in a six-dimensional configuration space. The spatial path traced by the *position* of a point on EE is described in three dimensions while the remaining three dimensions are used for describing the EE *orientation*. This six-dimensional configuration space is generally referred as *task space* in contrast to the *joint space*, which refers to the space spanned by joint angles (or displacements). Typically, the task of the manipulator is defined by a sequence of desired positions through which the EE must pass with specified orientations. These points are called as *knot points*. The knot points may be generated on-line and subjected to real-time modifications. It is the function of the *path planning* algorithm to construct a continuous path from the selective knot points. The interpolator then generates a sequence of time-based *set-points* for the low-level servo controllers to guide the manipulator from initial location to final location using the methods discussed in the last chapter. As expected, the path planner is highly dependent on the type of application. From application point of view, the robotic systems

can be classified into two major categories : *point to point* applications versus *continuous-path* applications [4.1].

In point to point (PTP) applications, the arm moves from one defined location (i.e. position and orientation both) to another and stops. The tool performs the required task and on completion the arm moves to the next point and the cycle is repeated. These stationary locations are referred as *end points*. Typical examples are spot welding, pick and place operations and palletizing operations etc. The path and speed of the EE while moving from one point to the next are of no significance except when considering such constraints as obstacle avoidance and time spent during the passage. In the continuous-path (CP) applications, the tool performs the task while the arm is in motion. Typical applications are arc welding, flame cutting, deburring, spray coating and routing etc. In CP motion, the motion of all joints of the arm have to coordinated such that the desired path is traced at the desired speed.

Extensive work has been done towards the trajectory planning in joint space predominantly for PTP applications. Several arguments used against planning in cartesian task space were [5.1] :

(i) For cartesian space motion, it is required to compute the transformation between cartesian and joint coordinates in real-time. This leads to high computational demand for real-time operation, which hindered the development of such algorithms. However, with the availability of cheap computing power, this not a limitation any more.

(ii) The solution for a inverse kinematic problem may not be unique for a general

purpose industrial robot, i.e. the cartesian location may be achieved by several joint vectors. This is not a problem with wrist-partitioned robots such as the Puma series where a unique solution can be obtained by using suitable indicators for arm configuration.

(iii) The actual physical constraints of the joint servos like maximum torque, limits of the joint rotation or displacement are readily expressed in joint coordinates. Thus trajectory planning can be done in joint space, by taking due consideration of these constraints. Due to nonlinear and coupled robot dynamics, it is too difficult to convert these bounds into their task space equivalents which has to be done experimentally. However, this is not a major issue for industrial environment where the typical working envelope of the manipulator and it payload vary in a small range and are chosen with sufficient margin from limiting values.

As a result, trajectory planning in joint space has received much attention. For PTP applications, sufficient intermediate positions (lift-off, set-down etc.) are selected between the desired end points, transformed into joint space and a joint trajectory for each joint is constructed using lower order spline functions. The spline functions are so generated so that the limits on joint velocity, acceleration and jerk (derivative of acceleration) are observed. Minimum time and obstacle avoidance may be taken as additional constraints. Lin et. al. [5.2] used piecewise cubic polynomials while minimising the travelling time, Luh & Lin [5.3] used cubic and quartic spline with least square error fit, Chang & Lin [5.4] used X-spline and quartic spline, Thompson and Patel [5.5] used B-spline and Anderson [5.6] used quintic polynomials for trajectory generation in joint

space. Of these, X-spline and B-spline require only a few look ahead knot points and are suitable for real-time modification. Optimum trajectory planning in joint space with minimum time criterion is also discussed by [5.11]-[5.15]. More recently, obstacle avoidance and path constraints have also been considered as additional factors by [5.16]-[5.20].

Although suitable for PTP operations, trajectory planning only in joint space has severe limitations for CP applications. Since there is no known functional transform between the joint trajectories and corresponding cartesian path, it is not possible to predetermine the actual path traced by the EE using the above mentioned techniques. It is important to realize that this problem can not be fully resolved by taking more knot points for interpolation in joint space. This is so because the actual arc traced between consecutive knot-points still remains indeterminate. Further, if the cartesian knot points are taken too close such that the distance between them is very less then the knot points themselves will act as the set-points for the servo controllers and any interpolation in joint space becomes redundant. In practice, the distance between knot points is taken large enough so that sufficient number of intermediate set points can be read from the joint space trajectories to permit operation of the manipulator in real-time.

Further, control over cartesian speed of the EE along the path traced is lost when trajectory is planned in joint space. It is so because normalised time is taken as the free independent parameter for generating polynomial trajectories in joint space. This fixes the time interval required to move from one knot position to the next. Very often these time intervals are optimised such as to minimise a particular cost function while a number of

imposed constraints are satisfied. However, as the actual arc length traversed by the EE while going from one knot point to the next is indeterminate, EE speed control is not possible.

In recent years some straight line approximation of cartesian paths have been proposed. The basic work towards straight line cartesian paths came from Paul[5.21], Taylor[5.22] and Whitney[5.23]. All of them attacked the problem of straight line trajectory wherein the position of the EE is linearly interpolated from the source to the destination. Chang et.al.[5.24] extended it to straight line segment approximations for a given curve in cartesian space. Because of linear approximation, there is an inherent error associated with the technique. Further the method is not suitable for real-time path modification. Dubowsky et.al. [5.19] used Bezier spline in cartesian space. Beizer spline has a drawback that it fails to pass through all the knot points. [5.25]-[5.28] have used cubic spline, circular interpolations and blend functions respectively for path generation in cartesian space. However, methods to allow real-time path modifications and/or speed regulation are not discussed. The orientation specification in the cartesian space has been investigated by [5.29]-[5.30]. The technique proposed by Angeles et.al. [5.31] for trajectory planning in CP applications formulates the problem as a non-linear two-point boundary value problem [5.32] makeing it computationaly demanding for real-time application. Gutsche et.al. [5.33] mentioned the use of cubic Hermite spline for cartesian path generation but do not address the issues of speed regulation and orientation determination.

An essential pre-requisite for multiple manipulator synchronization, which is the

topic of discussion in next chapter, is the availability of an efficient and readily implementable path planning algorithm with the capability to accommodate real-time path modifications and assure EE speed regulation. This chapter discusses a novel scheme devised and implemented for synthesising smooth manipulator paths using Cubic Spline Functions. Using selective knot points, smooth spline functions are generated for each of the cartesian coordinates x,y and z. Although discussed for cartesian coordinates, the technique is general and can be used to generate smooth paths in any other coordinate frames used in the task space. The arc length of the generated path is used as the free parameter and the manipulator can be made to travel at the user-desired speed along the generated path.

5.2 Problem Statement

The problem under consideration can be stated as follows :

The desired cartesian path of a manipulator is available as few look ahead knot points, identified on-line, through which the EE must pass as shown in Figure 5.1. The path planning algorithm should construct a continuous curve with smooth curvature passing through these knot points. The speed of traversal along the constructed arc and the orientation of the EE with respect to it should be user controllable in real-time.

5.3 Review of Cubic Spline Functions

Before we proceed to discuss the application of cubic spline function for manipulator path generation, a brief discussion of cubic splines and their properties is in order. For a detailed analysis, [5.34] forms an excellent reference. Many methods have been devised for approximating an unknown curve from some sample values. The

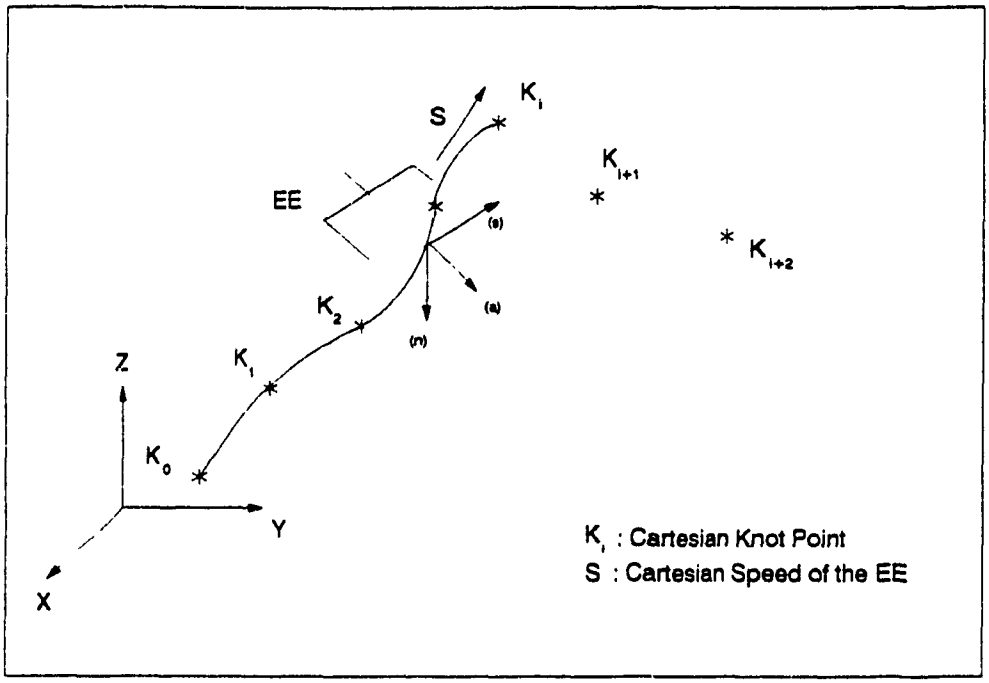

Figure 5.1 Path Planning in Cartesian Space.

simplest technique of piecewise linear approximation is neither very smooth nor very efficient in its approximations. One has to go to piecewise polynomial approximations with higher order pieces for smoother and better approximations. The most popular choice continues to be a piecewise cubic approximating function.

Given the data $g(\tau_1)$,....., $g(\tau_n)$ with $a = \tau_1 < \ldots < \tau_n = b$, the piecewise cubic interpolant function f to g is constructed in the following manner. On each interval $[\tau_i,\tau_{i+1}]$ we make f agree with some cubic polynomial P_i i.e.

$f(\tau) = P_i(\tau)$ for $\tau_i \le \tau \le \tau_{i+1}$ $i = 1,\ldots,n-1$.

where $P_i \in \mathcal{P}_4$, the linear space of cubic polynomials.

The ith polynomial piece P_i is made to satisfy the following conditions

$$P_i(\tau_i) = g(\tau_i) = g_i \qquad (5.1a)$$

$$P_i(\tau_{i+1}) = g(\tau_{i+1}) = g_{i+1} \tag{5.1b}$$

$$P_i'(\tau_i) = u_i \tag{5.1c}$$

$$P_i'(\tau_{i+1}) = u_{i+1} \qquad i = 1,...,n-1 \tag{5.1d}$$

where $u_1,...,u_n$ are free parameters. How these parameters are actually chosen will be discussed in a short while.

As a result, the piecewise cubic function f agrees with g at $\tau_1,...,\tau_n$ and is an element of $C^{(1)}[a,b]$, i.e. it is continuous and has a continuous first derivative on [a,b], regardless of the way the free parameters $u_1,...u_n$ are chosen.

Different piecewise cubic interpolation schemes differ in the choice of the free parameters $u_1,...,u_n$. As an example if we take

$u_i = g'(\tau_i)$ for all i

we get the *cubic hermite interpolation*. For *cubic spline interpolation* one selects the free parameters in such a manner so that f is *twice* continuously differentiable, i.e. it has a continuous curvature. This leads us to the condition

$$P_{i-1}''(\tau_i) = P_i''(\tau_i) \qquad i = 2,...,n-1. \tag{5.2}$$

If we express $P_i(\tau)$ as

$$P_i(\tau) = q_{1,i} + q_{2,i}(\tau-\tau_i) + q_{3,i}(\tau-\tau_i)^2 + q_{4,i}(\tau-\tau_i)^3 \tag{5.3a}$$

then on applying (5.1a)-(5.1d) we get

$$q_{1,i} = P_i(\tau_i) \tag{5.3b}$$

$$q_{2,i} = P_i'(\tau_i) = u_i \tag{5.3c}$$

$$q_{3,i} = P_i''(\tau_i)/2 = ([\tau_i,\tau_{i+1}]g - u_i)/\Delta\tau_i - q_{4,i}\Delta\tau_i \tag{5.3d}$$

$$q_{4,i} = P_i'''(\tau_i)/6 = (u_i + u_{i+1} - 2[\tau_i,\tau_{i+1}]g)/(\Delta\tau_i)^2 \tag{5.3e}$$

where

$\Delta\tau_i = \tau_{i+1} - \tau_i$ and

$[\tau_i,\tau_{i+1}]g = (g_{i+1} - g_i)/(\tau_{i+1} - \tau_i)$ is generally called the divided difference of the function g at the points τ_i, τ_{i+1}.

On applying condition (5.2) on (5.3a) we get

$$2q_{3,i-1} + 6q_{4,i-1}\Delta\tau_{i-1} = 2q_{3,i} \tag{5.4}$$

which on using (5.3d)-(5.3e) leads to

$$2([\tau_{i-1},\tau_i]g - u_{i-1})/\Delta\tau_{i-1} + 4q_{4,i-1}\Delta\tau_{i-1} = 2([\tau_i,\tau_{i+1}]g - u_i)/\Delta\tau_i - 2q_{4,i}\Delta\tau_i \tag{5.5}$$

Rearranging terms we get

$$u_{i-1}\Delta\tau_i + 2u_i(\Delta\tau_{i-1} + \Delta\tau_i) + u_{i+1}\Delta\tau_{i-1} = c_i \tag{5.6a}$$

where

$$c_i = 3(\Delta\tau_i[\tau_{i-1},\tau_i]g + \Delta\tau_{i-1}[\tau_i,\tau_{i+1}]g) \qquad i = 2,...,n-1. \tag{5.6b}$$

It is clear that with suitable choice of u_1 and u_n, (5.6) leads to tridiagonal system of equations with n-2 equations for n-2 unknowns $u_2,...,u_{n-1}$. Further the system has *diagonal dominance* (i.e. coefficient of u_i > coefficient of u_{i-1} + coefficient of u_{i+1}) and hence it can always be solved for exactly one solution without pivoting [5.37].

The different choices for choosing *boundary conditions* i.e. u_1 and u_n lead to further variations in the cubic spline interpolation. Two of the most commonly used conditions are

(i) **Complete cubic spline** : If derivative of the function g is known at τ_1 and τ_n then one can choose

$$u_1 = g'(\tau_1) \tag{5.7a}$$

$$u_n = g'(\tau_n). \tag{5.7b}$$

The resulting spline interpolant is called the *complete cubic spline*. Some interesting properties of the complete cubic spline like *smoothest interpolant property* and the *best approximation property* are discussed in detail in [5.35].

(ii) **Natural cubic spline :** If one has no estimate of the function derivative at the end points, one can choose

$$P_1''(\tau_1) = P_n''(\tau_n) = 0. \tag{5.8}$$

The resulting spline is called the *natural cubic spline*. However, as expected from this arbitrary assignment, this leads to significant error (a function of $|\tau|^2$) near the end points unless $g''(\tau_1)$ and $g''(\tau_n)$ are also zero.

Many other schemes for constructing piecewise cubic splines like Akima's interpolation etc. are discussed in [5.36]. With this background in the theory of cubic splines, we investigate their application for on-line path generation in cartesian space. [5.37] gives some excellent and very compact programs for generating cubic splines for a given set of sample points, which were suitably modified for the current application.

5.4 Cartesian Path Generation using Cubic Splines

For discussion of the proposed path planning algorithm, we shall attack the problem in three steps. These are :

(i) Determination of desired EE position (x-y-z).

(ii) Speed regulation along the arc traced by the EE.

(iii) Determinaton of desired EE orientation [**n a s**].

5.4.1 EE Position (x-y-z) Determination

Let $\mathbf{K}_i$ denote a knot point detected by the sensor interface, i.e.

$$\mathbf{K}_i \equiv (x_i, y_i, z_i) \quad i = 1,2,...$$

When at point $\mathbf{K}_i$ we **look ahead by n knot points** and construct a smooth path through $[\mathbf{K}_i, \mathbf{K}_{i+1}, ..., \mathbf{K}_{i+n}]$. The factors effecting choice of n are discussed in section 5.4.3. It is important to note that $[\mathbf{K}_i, \mathbf{K}_{i+1}, ..., \mathbf{K}_{i+n}]$ have no pre-defined temporal relationship except for the fact that the EE should traverse through them in a specific order, i.e from $\mathbf{K}_i$ to $\mathbf{K}_{i+1}$ and so on. The exact time spent in moving from say $\mathbf{K}_i$ to $\mathbf{K}_{i+1}$ is dependent on the speed of the EE and can not be pre-determined. To construct a smooth curve through $[\mathbf{K}_i, \mathbf{K}_{i+1}, ..., \mathbf{K}_{i+n}]$, we cubic spline each of the cartesian coordinate independently. In the following discussion, r can be equally replaced by x,y or z. To each of the knot point $\mathbf{K}_i$, we assign τ_i as the value of the free parameter τ. Except that the values $[\tau_i, \tau_{i+1}, ..., \tau_{i+n}]$ are so chosen such that

$$0 = \tau_i < \tau_{i+1} < ... < \tau_{i+n} = 1 \tag{5.9}$$

is satisfied, the assignment is otherwise arbitrary. One clear choice is to make them equally spaced by taking

$$\Delta\tau = \tau_{i+1} - \tau_i = 1/(n-1) \qquad \forall i$$

Using the data $r_i, ..., r_{i+n}$, we construct the cubic spline interpolant $r(\tau)$, for coordinate r in the follwing manner.

Let $r_i(\tau)$ be the cubic polynomial segment of the spline interpolant $r(\tau)$ between $[\tau_i, \tau_{i+1}]$. We rewrite (5.3) in the form

$$r_i(\tau) = a_{3r,i}\tau^3 + a_{2r,i}\tau^2 + a_{1r,i}\tau + a_{0r,i} \tag{5.10a}$$

and find an expression for the coeffecients by using the following equations

$$r_i = r_i(\tau_i) = a_{3r,i}\tau_i^3 + a_{2r,i}\tau_i^2 + a_{1r,i}\tau_i + a_{0r,i} \tag{5.11a}$$

$$r_{i+1} = r_i(\tau_{i+1}) = a_{3r,i}\tau^3_{i+1} + a_{2r,i}\tau^2_{i+1} + a_{1r,i}\tau_{i+1} + a_{0r,i} \tag{5.11b}$$

$$r_i''(\tau_i) = 6a_{3r,i}\tau_i + 2a_{2r,i} \tag{5.11c}$$

$$r_i''(\tau_{i+1}) = 6a_{3r,i}\tau_{i+1} + 2a_{2r,i} \tag{5.11d}$$

Equations (5.11c), (5.11d) give

$$a_{3r,i} = (r_i''(\tau_{i+1}) - r_i''(\tau_i))/6\Delta\tau \tag{5.10b}$$

$$a_{2r,i} = (\tau_{i+1}r_i''(\tau_i) - \tau_i r_i''(\tau_{i+1}))/2\Delta\tau \tag{5.10c}$$

Equations (5.11a), (5.11b) along with the identities

$$\tau^3_{i+1} - \tau^3_i = \Delta\tau^3 - 3\tau_{i+1}\tau_i^2 + 3\tau^2_{i+1}\tau_i$$

$$\tau^2_{i+1} - \tau^2_i = \Delta\tau(\tau_{i+1} + \tau_i)$$

give

$$a_{1r,i} = [\ \{r_i''(\tau_i)(\Delta\tau^2 - 3\tau^2_{i+1})\}/6 + \{r_i''(\tau_{i+1})(3\tau_i^2 - \Delta\tau^2)\}/6 + r_{i+1}) - r_i\ ]/\Delta\tau \tag{5.10d}$$

$$a_{0r,i} = [\ \{r_i''(\tau_i)\tau_{i+1}(\tau^2_{i+1} - \Delta\tau^2)\}/6 + \{r_i''(\tau_{i+1})\tau_i(\Delta\tau^2 - \tau_i^2)\}/6 + \tau_{i+1}r_i - \tau_i r_{i+1})\ ]/\Delta\tau \tag{5.10e}$$

To determine $r_i''(\tau_i)$ and $r_i''(\tau_{i+1})$ such that the continuous curvature is assured, we solve the n-2 order tridiagonal system, refer equation (5.6), which can be re-written as

$$(u_{p-1} + 4u_p + u_{p+1})\Delta\tau = c_p \tag{5.12a}$$

where p varies from i+1 to i+n-1 and

$$c_p = 3\Delta\tau([\tau_{p-1},\tau_p]r + [\tau_p,\tau_{p+1}]r) \tag{5.12b}$$

To solve (5.12), we chose u_i and u_{i+n} as follows

$$u_i = r_{i-1}'(\tau_i) \qquad i \neq 1 \qquad (5.13a)$$

$$= (r_2 - r_1)/\Delta\tau \qquad i = 1 \qquad (5.13b)$$

$$u_{i+n} = (r_{i+n} - r_{i+n-1})/\Delta\tau \qquad \forall i \qquad (5.13c)$$

The equations (5.13b), (5.13c) give first order estimates of $r_1'(\tau_1)$ and $r_{i+n}'(\tau_{i+n})$ respectively. Equation (5.13a) assures the desired continuity with $r_{i-1}(\tau)$ polynomial. With reference to (5.7), it is clear that (5.13) leads to the construction of *complete cubic spline* over any interval.

With the above choices of u_i and u_{i+n}, Eq.(5.12) is solved for $u_{i+1},...,u_{i+n-1}$. Having determined $u_i,...,u_{i+n}$, using (5.3d) and (5.3e) in terms of $r_i(\tau)$, we get

$$r_i''(\tau_i) = (6[\tau_i,\tau_{i+1}]r - 2u_{i+1} - 4u_i)/\Delta\tau \qquad (5.14)$$

Similarily $r_i''(\tau_{i+1})$ can be evaluated. Thus using (5.10) the cubic polynomials $r_i(\tau)$, $r_{i+1}(\tau)$...., $r_{i+n}(\tau)$, are known and the complete cubic spline $r(\tau)$ for the coordinate r is determined.

This procedure is repeated for all the coordinates of the task space. In case of cartesian space, a cubic spline interpolant is thus constructed for x,y, and z giving a cubic spline formulation for the 3D arc to be traversed. For moving the arm along this 3D arc, the following conditions are tested.

(i) **C1 : End Point is not detected**

This condition is true when the sensor system detects additional knot points beyond $\mathbf{K}_{i+n}$. In this case the manipulator is moved on the spline constructed by using $[\mathbf{K}_i,\mathbf{K}_{i+1},...,\mathbf{K}_{i+n}]$ only till an intermediate point $\mathbf{K}_{i+q}$, where q<n. Factors effecting choice of q are discussed in section 5.4.3. While this motion is being

executed at the desired speed, the sensor system should accumulate q new knot points $[\mathbf{K}_{i+n+1},...,\mathbf{K}_{i+n+q}]$. A new spline is constructed from the n points $[\mathbf{K}_{i+q},\mathbf{K}_{i+q+1},..,\mathbf{K}_{i+n},...,\mathbf{K}_{i+n+q}]$. Beyond the point $\mathbf{K}_{i+q}$, the motion is directed along the new spline. The cycle repeats itself till C2 is satisfied.

(ii) **C2 : End Point is detected**

This is the condition when a knot point, say $\mathbf{K}_{i+p}$ $1 \le p \le n$, is the end point, i.e. the manipulator is expected to come to a complete stop on reaching $\mathbf{K}_{i+p}$. Then we assign the remaining n-p points as

$$\mathbf{K}_{i+p} = \mathbf{K}_{i+p+1} = ... = \mathbf{K}_{i+n-1} = \mathbf{K}_{i+n}$$

We use the spline constructed from $[\mathbf{K}_{i},...,\mathbf{K}_{i+n}]$, which consists of n-p repeated knot points, to direct the arm till the end point $\mathbf{K}_{i+p}$. Under this condition, (5.25c) gets modified as

$$u_{i+n} = (r_{i+p} - r_{i+p-1})/\Delta\tau \qquad (5.13d)$$

Some remarks on the technique before we conclude the section are :

(i) Except in the spline terminating in the end point, the error induced in path construction due to first order estimate of $r_i'(\tau_{i+n})$, (5.13c), is significantly reduced due to the fact that the EE moves only till the intermediate point $\mathbf{K}_{i+q}$, $q<n$, on the constructed spline. Similarily, error due to estimation of $r_i'(\tau_i)$, (5.13b), is restricted to the starting motion of the manipulator.

(ii) Equation (5.13a) assures continuity of first derivative between consecutive splines. However, as the consecutive splines overlap over $[\mathbf{K}_{i+q},...,\mathbf{K}_{i+n}]$ knot points, the curvature (second derivative) of the resulting path is found to be smooth.

To conclude, the described technique gives a formulation for the cartesian path (position) to be traced by the EE in the form of independent cubic spline functions for each of the coordinate constructed from n look ahead knot points.

5.4.2 Cartesian Speed Regulation

Having determined a formulation for the desired path, we want to regulate the speed of EE as the path is traversed. One should recall the fact that independent parameter τ, used in the spline construction, has no relation with time. Let

$$\Gamma_i \equiv (x_i(\tau), y_i(\tau), z_i(\tau))$$

represent the 3D arc constructed between the knot points $[\mathbf{K}_i, \mathbf{K}_{i+1}]$, where expression for each of the coordinate is of the form (5.10). To control the arm speed, we seek a relation between τ and arc-length along Γ_i. The *arc length function* of Γ_i [5.38] is given as

$$s_i(\tau) = \int_{\hat{\tau}_i}^{\tau} \sqrt{[dx_i(\hat{\tau})]^2 + [dy_i(\hat{\tau})]^2 + [dz_i(\hat{\tau})]^2}\, d\hat{\tau} \qquad \hat{\tau} \in [\tau_i, \tau_{i+1}] \tag{5.15}$$

From (5.10) we get

$$[r_i'(\tau)]^2 = 9a^2_{3r,i}\tau^4 + 12a_{3r,i}a_{2r,i}\tau^3 + (4a^2_{2r,i} + 6a_{3r,i}a_{1r,i})\tau^2 + 4a_{2r,i}a_{1r,i}\tau + a^2_{1r,i} \tag{5.16}$$

Let

$$\wp_i(\tau) = [dx_i(\tau)]^2 + [dy_i(\tau)]^2 + [dz_i(\tau)]^2$$

Using (5.16) for each of the coordinate and summing, we get

$$\wp_i(\tau) = p_{4,i}\tau^4 + p_{3,i}\tau^3 + p_{2,i}\tau^2 + p_{1,i}\tau + p_{0,i} \tag{5.17a}$$

where

$$p_{4,i} = 9(a^2_{3x,i} + a^2_{3y,i} + a^2_{3z,i}) \tag{5.17b}$$

$$p_{3,i} = 12(a_{3x,i}a_{2x,i} + a_{3y,i}a_{2y,i} + a_{3z,i}a_{2z,i}) \tag{5.17c}$$

$$p_{2,i} = 4(a^2_{2x,i} + a^2_{2y,i} + a^2_{2z,i}) + 6(a_{3x,i}a_{1x,i}+a_{3y,i}a_{1y,i}+a_{3z,i}a_{1z,i}) \tag{5.17d}$$

$$p_{1,i} = 4(a_{2x,i}a_{1x,i} + a_{2y,i}a_{1y,i} + a_{2z,i}a_{1z,i}) \tag{5.17e}$$

$$p_{0,i} = a^2_{1x,i} + a^2_{1y,i} + a^2_{1z,i} \tag{5.17f}$$

To find an analytical expresson for $s_i(\tau)$, we need to evaluate the square root of the polynomial $\wp_i(\tau)$. As $\wp_i(\tau)$ need not be a perfect square, we attempt to reduce it in the form, [5.39],

$$\wp_i(\tau) = [b_{0,i}\tau + b_{1,i}\tau + \ldots b_{j,i}\tau^2]^2 + W'(\tau) \tag{5.18}$$

i.e., to the sum of a perfect square and a polynomial, $W'(\tau)$, *whose lowest term is of as high a degree as j+1*. As $0 \leq \tau \leq 1$ in (5.9), it is clear from (5.15) that in the polynomial expression for $s_i(\tau)$, obtained after integration, higher powers of τ will contribute less and less. Thus, depending on the desired accuracy in speed regulation, one can select j and use

$$\sqrt{\wp_i(\tau)} \approx b_{0,i}\tau + b_{1,i}\tau + \ldots b_{j,i}\tau^2 \tag{5.19}$$

to get a closed form expression for $s_i(\tau)$.

In the present case, for the arc Γ_i, $\wp_i(\tau)$ is 4th order polynomial, (5.17a), and hence it is sufficient to take j=2. This has an added advantage that the resulting polynomial for $s_i(\tau)$ is cubic and the roots have a closed form expression. With this choice, comparing coeffecients in (5.18) gives

$$b_{0,i} = \sqrt{p_{0,i}} \tag{5.20a}$$

$$b_{1,i} = p_{1,i}/(2b_{0,i}) \tag{5.20b}$$

$$b_{2,i} = (p_{2,i} - b^2_{1,i})/(2b_{0,i}) \tag{5.20c}$$

From (5.17f), we see that $p_{0,i} > 0$ $\forall i$, except in the degenerate solution of (5.12) with $r_i''(\tau_i) = r_i''(\tau_{i+1}) = \ldots = r_i''(\tau_{i+n}) = 0$ for each of the coordinate. Using (5.19) and (5.20) in (5.15) and integrating we get

$$s_i(\tau) = (b_{2,i}/3)\tau^3 + (b_{1,i}/2)\tau^2 + b_{0,i}\tau - b'_i \quad \tau \in [\tau_i, \tau_{i+1}] \tag{5.21}$$

where $b'_i = (b_{2,i}/3)\tau^3_i + (b_{1,i}/2)\tau^2_i + b_{0,i}\tau_i$

Thus $s_i(\tau_{i+1})$ gives the arc length of the 3D curve Γ_i, which is the path to be pursued between knot points $\mathbf{K}_i$ and $\mathbf{K}_{i+1}$ as shown in Figure 5.2.

Let m_i^{th} be the sampling interval, i.e. $t \in [(m_i-1)T_c, m_iT_c]$, $m_i = 1,2,\ldots$, **when the EE transits from arc Γ_{i-1} to arc Γ_i**. If i=1, i.e. at the start of the motion, then clearly $m_1=1$. As we restrict our attention to the segments Γ_i and Γ_{i+1} we will avoid the subscript i for m for the sake of notational clarity.

To represent the temporal association of τ, the symbol $\tilde{\tau}$ is introduced. While τ_i is the value of τ for the knot point $\mathbf{K}_i$, $\tilde{\tau}_k$ is used to denote the value of τ associated with the set point loaded in the servo controllers in the k^{th} sampling interval. Further, let S[k] be the desired cartesian speed of the EE in the k^{th} interval.

The incremental arc length $\Delta\ell_m$ to be covered in the m^{th} interval, i.e. when the mainpulator is crossing from arc Γ_{i-1} to arc Γ_i is then given by

$$\Delta\ell_m = S[m]T_c \tag{5.22}$$

Depending on the magnitude of $\Delta\ell_m$, two conditions arise. These are

(i) **C3 : $\Delta\ell_m < s_i(\tau_{i+1})$**

This is the useful case. Here we need to identify intermediate points on the arc Γ_i

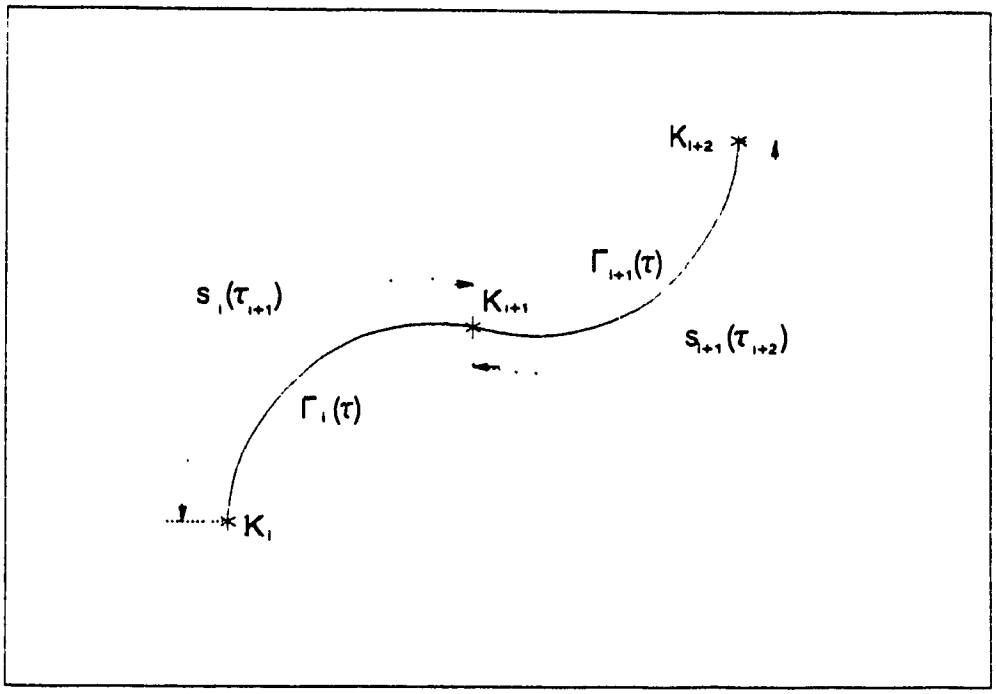

Figure 5.2 Arc length between knot points

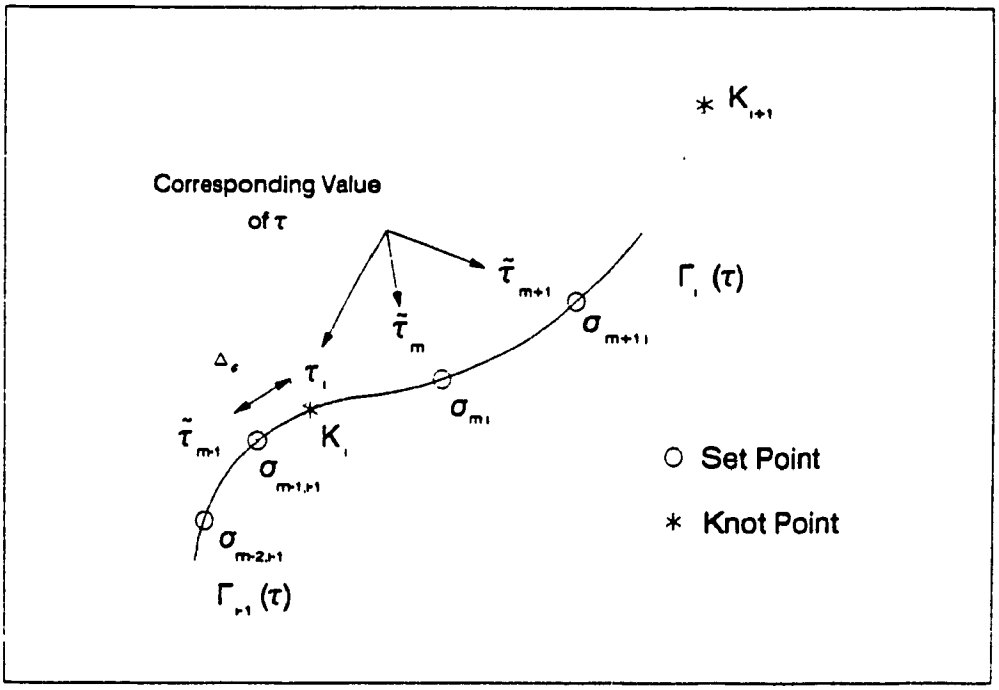

Figure 5.3 Intermediate set points $\sigma_{m+j,i}$ on Γ_i

which can be downloaded into the servo controllers as desired set points in successive sampling intervals as shown in Figure 5.3. To simplify matters, we have assumed that there is only *linear interpolation* in joint space. Discussion for other interpolation techniques in joint space is postponed till the end. The notation used for a servo set point is as follows

$$\sigma_{k,i} \equiv (x_i(\tilde{\tau}_k), y_i(\tilde{\tau}_k), z_i(\tilde{\tau}_k)) \qquad k,i = 1,2,...$$

where

k indicates that $\sigma_{k,i}$ is set point for k^{th} interval.

and i indicates that $\sigma_{j,i}$ lies on the arc Γ_i.

Thus, $\sigma_{m,i}$, $\sigma_{m+1,i}$, $\sigma_{m+2,i}$ ... are consecutive set points identified on Γ_i and corresponding to the $\tilde{\tau}_m$, $\tilde{\tau}_{m+1}$, $\tilde{\tau}_{m+2}$, ... values of the parameter τ. Let Δ_ε represent the uncovered length along Γ_{i-1} when the arm switches from Γ_{i-1} to Γ_i in the m^{th} interval as shown in Figure 5.3. Then

$$\Delta_\varepsilon = s_{i,1}(\tau_i) - s_{i-1}(\tilde{\tau}_{m-1}) \qquad i \neq 1 \qquad (5.23)$$

$$= 0 \qquad i = 1$$

For regulated speed travel one gets the condition, using (5.21), that $\sigma_{m+j,i}{}^{th}$ j = 0,1,2..., set point should satisfy the following equation

$$\frac{b_{2,i}}{3}\tilde{\tau}^3_{m+j} + \frac{b_{1,i}}{2}\tilde{\tau}^2_{m+j} + b_{0,i}\tilde{\tau}_{m+j} - b'_i = \sum_{l=1}^{j} \Delta\ell_{m+l} + (\Delta\ell_m - \Delta_\varepsilon) \qquad (5.24a)$$

where

$$b'_i = (b_{2,i}/3)\tau^3_i + (b_{1,i}/2)\tau^2_i + b_{0,i}\tau_i$$

$$\Delta\ell_{m+l} = S[m+l]T_c$$

The solution of (5.24a) which also satisfies

$$\tilde{\tau}_{m+j} \in [\tau_i, \tau_{i+1}] \tag{5.24b}$$

$$\tilde{\tau}_{m+j} > \tilde{\tau}_{m+j-1} \quad \text{for } j \neq 0 \tag{5.24c}$$

can be used to identify the set point $\sigma_{m+j,i}$. To avoid any confusion we reiterate the fact that symbolic association of $\tilde{\tau}_{m+j}$, with the interval $[\tau_i, \tau_{i+1}]$ is built into the fact that m represents the interval when the manipulator moves from arc Γ_{i-1} to Γ_i. Equation (5.24a) can be rewritten as

$$\tilde{\tau}^3_{m+j} + c_2\tilde{\tau}^2_{m+j} + c_1\tilde{\tau}_{m+j} + c_0 = 0 \tag{5.25a}$$

where

$$c_2 = (3b_{1,i}/2b_{2,i}) \tag{5.25b}$$

$$c_1 = (3b_{0,i}/b_{2,i}) \tag{5.25c}$$

$$c_0 = \Delta_t - \Sigma^j_{l=0}\, \Delta\ell_{m+l} - b'_i \tag{5.25d}$$

To solve (5.25), we look at the sign of the determinant Q^3-R^2 [5.37], where

$$Q = (c^2_2 - 3c_1)/9 \tag{5.26a}$$

$$R = (2c^3_2 - 9c_2c_1 + 27c_0)/54 \tag{5.26b}$$

If Q^3-$R^2 < 0$, (5.25) has only one real root and $\tilde{\tau}_{m+j}$, is given by

$$\tilde{\tau}_{m+j} = -sgn(R)\left[\left(\sqrt{R^2 - Q^3} + |R|\right)^{1/3} + \frac{Q}{\left(\sqrt{R^2 - Q^3} + |R|\right)^{1/3}}\right] - \frac{c_2}{3} \tag{5.27}$$

If Q^3-$R^2 \geq 0$, then (5.25) has three real roots given by

$$r1 = -2\sqrt{Q}\cos[\theta/3] - c_2/3 \tag{5.28a}$$

$$r2 = -2\sqrt{Q}\cos[(\theta+2\pi)/3) - c_2/3 \tag{5.28b}$$

$$r3 = \sqrt[3]{(\theta+4\pi)/3) - c_2/3} \qquad (5.28c)$$

where

$$\theta = \arccos(R/\sqrt{Q^3})$$

The root which satisfies (5.24b) and (5.24c) gives the desired value of $\tilde{\tau}_{m+j}$. The physical significance of (5.24a) ensures that a solution can always be found which satisfies equations (5.24b) and (5.24c). Once $\tilde{\tau}_{m+j}$ is determined, the $\sigma_{m+j,i}^{th}$ set point can be generated by using it in the cubic polynomial expression, (5.10), obtained for each of the coordinate.

As a result of the above mentioned formulation, the manipulator moves along the arc Γ_i in the m^{th}, $m+1^{th}$,..,$m+j^{th}$,... sampling intervals with the desired speed of S[m], S[m+1],..,S[m+j],... till the complete length of Γ_i is covered. This will happen for the particular value of j=jj, when the following becomes true

$$s_i(\tilde{\tau}_{m+jj-1}) + \Delta\ell_{m+jj} > s_i(\tau_{i+1}) \qquad (5.29)$$

At this point, Δ_ϵ is reset to reflect the new error, i.e.

$$\Delta_\epsilon = s_i(\tau_{i+1}) - s_i(\tilde{\tau}_{m+jj-1})$$

and the arm switches from Γ_i to Γ_{i+1}. Only if $\mathbf{K}_{i+1}$ is identified as a set point, i.e. $\tilde{\tau}_{m+jj} = \tau_{i+1}$, Δ_ϵ will be reset to zero. This cycle repeats itself till the end point is reached, when the speed in the last interval is so adjusted such that the terminating knot point becomes the last set point.

(ii) **C4 : $\Delta\ell_m \geq s_i(\tau_{i+1})$**

This is the case when the desired cartesian speed is so high that path planning becomes redundant. Although in case of equality one can in principal

move the arm by using knot points as the set point but for the proposed algorithm to be of any significance, one must reduce the cartesian speed such that C3 hold true.

The propsed algorithm, discussed above in its most general form, considered the worst case when the desired speed S[k] may change in every sampling interval, which may actually be undesirable in real practice. This lead to the limitation of considering only linear interpolation in joint space. If it is assumed that the speed remains constant, say over p intervals, then with an appropriate choice of q (refer condition C1), p look ahead set points (with attached time stamps) can be identified and transformed into joint space. Using them any joint space interpolation technique, which requires p or less anchor points, can be used to generate smooth joint space trajectories. As expected, this will induce some errors in the path traced in cartesian space. Before discussing the extension of the technique for orientation determination, we look at some experimental results.

5.4.3 Implementation and Experimental Results

For experimental verification, the proposed algorithm was implemented on the Integrated Workcell Controller, discussed in Chapter 3, to move the PUMA 260 arm on an on-line generated path with speed regulation. The path planning algorithm forms an integral part of the Cartesian Path Controller (CPC) for the PUMA 260 arm. As the camera/sensor interface is yet not implemented on the IWC, it was simulated through software by developing code in Parallel C to generate cartesian points lying on curves with desired mathematical formulation or to read the knot points from a specified file.

After some initial trials, the value of **n = 4 and q = 1** was found to be optimum. Thus a spline function is generated for each coordinate axis passing through next four knot points. However, the manipulator traverses only along the first cubic segment of this spline to reach the first knot point. While this motion is in progress, a new knot point is received and a new spline is constructed. Beyond the first knot point, manipulator motion is directed along the new spline. The cycle repeats itself till the end point is detected. The independent parameter τ was taken at equal intervals such that $\Delta\tau = 1/3\ \forall i$.

A higher value of n adversely effects the capability to accommodate on-line path modification. Further, the computational time also increases as the order of tridiagonal system increases with n. A lower value of n leads to increased error in the path travelled due to first order derivative estimation in (5.13c). Noting that $\tau_i = 0$ from (5.9), taking n = 4 and q = 1 also simplifies (5.10) to a great extent for the arc being travelled to

$$r_i(\tau) = a_{3r,i}\tau^3 + a_{2r,i}\tau^2 + a_{1r,i}\tau + a_{0r,i} \qquad (5.30a)$$

where

$$a_{3r,i} = (r_i''(\tau_{i+1}) - r_i''(\tau_i))/2 \qquad (5.30b)$$

$$a_{2r,i} = r_i''(\tau_i)/2 \qquad (5.30c)$$

$$a_{1r,i} = 3[r_{i+1} - r_i - (2r_i''(\tau_i) + r_i''(\tau_{i+1}))/54] \qquad (5.30d)$$

$$a_{0r,i} = r_i \qquad (5.30e)$$

A higher value of q will also lead to increased errors due to significant influence of approximation in (5.13c). With these choices of n,l and τ parameters, measurements using transputer timer show that it takes **less than 0.5 milliseconds** to solve (5.24), which

constitutes the real-time critical part of the algorithm, on T800-20 transputer. Further, the implementation makes the realistic assumption that the desired speed remains constant over the arc Γ_i being travelled and may only change when the arm switches from Γ_i to Γ_{i+1}. Results obtained for various curves in 3D space are as follows.

5.4.3.1 Motion along an Elliptical Helix

Figure 5.4 shows the knot points and the cartesian path constructed using cubic spline method for an elliptical helix with the formulation

$$x_i = R_x \cos(\Sigma'_0\, \delta\xi) + x_0 \tag{5.31a}$$

$$y_i = R_y \cos(\Sigma'_0\, \delta\xi) + y_0 \tag{5.31b}$$

$$z_i = P_z(\Sigma'_0\, \delta\xi) + z_0 \tag{5.31c}$$

where

$$\delta\xi = 2\pi/N$$

For Figure 5.4, N = 60, R_x = 60.0 mm, R_y = 100.0 mm and P_z = 1.6 mm giving a pitch of 10 mm along the z axis. As expected, the path constructed using splines is continuous with smooth curvature and passes through the desired knot points. Figure 5.5 shows that actual path traced by the Puma 260 EE with respect to the knot points, where forward kinematics is used to determine the robot EE position. Figure 5.6 shows the speed profile for the desired speed of 40 mm/sec, 80 mm/sec and 140 mm/sec along the helical path. For commanded speed of upto 100mm/sec, the **error in speed regulation is less than 5%**. At higher speed of 140mm/sec, which is too high for most of the CP operations, the error is less than 10%. It should be noted that the positional error in Figure 5.5 is not due to the proposed algorithm. As the IWC Puma 260 controller uses

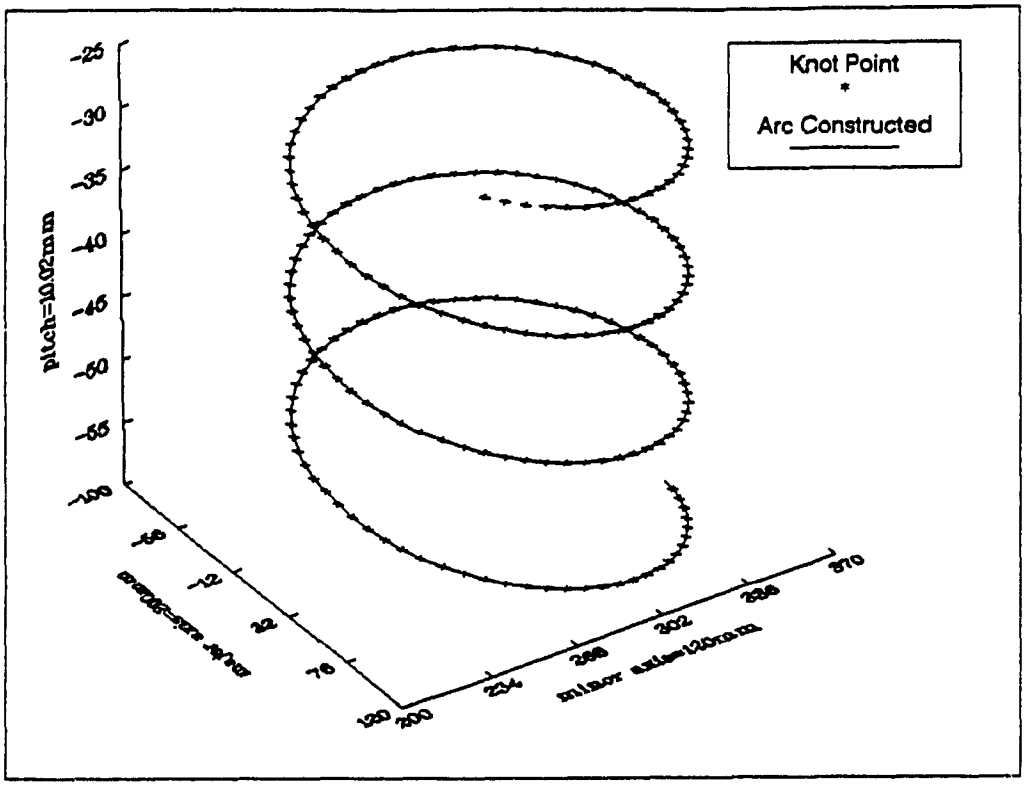

Figure 5.4 Elliptical Helix from knot points.

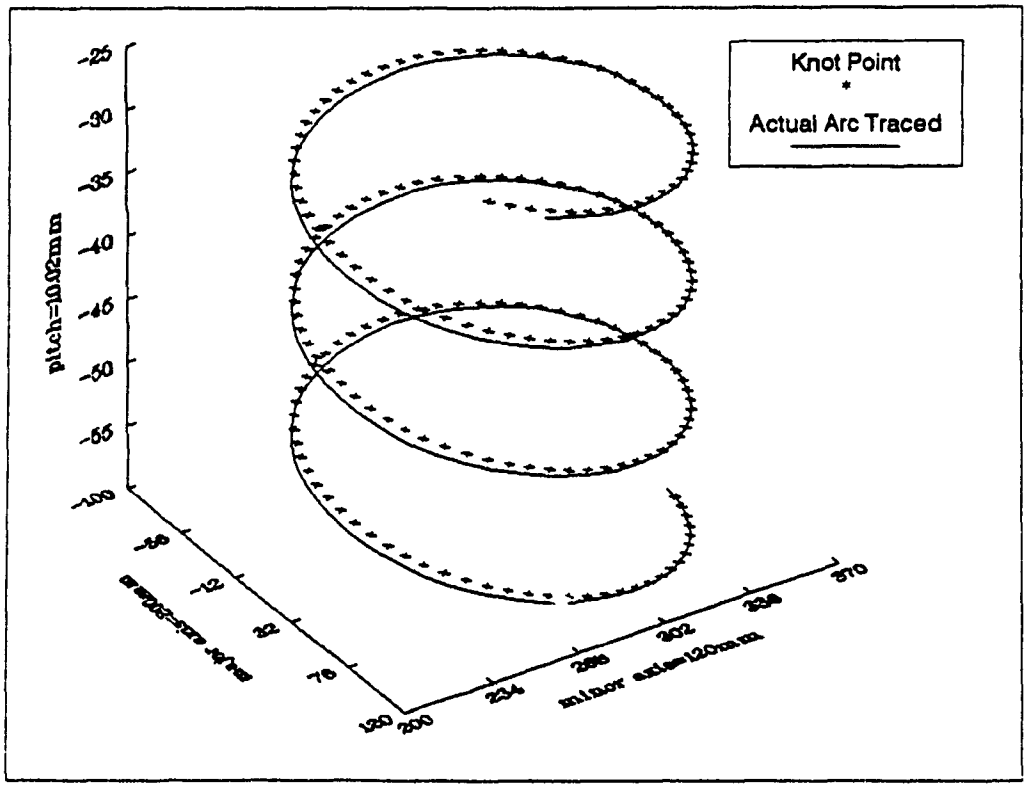

Figure 5.5 Path traced by the EE in relation to knot points.

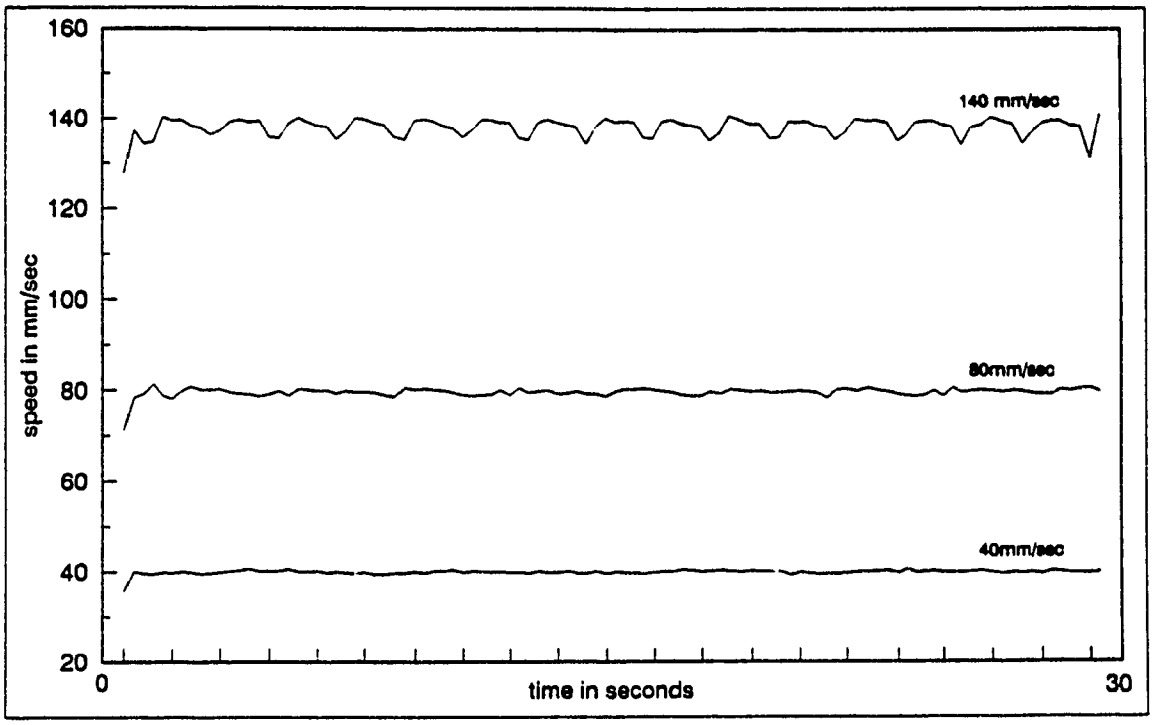

Figure 5.6 Speed Profile for Elliptical Helix

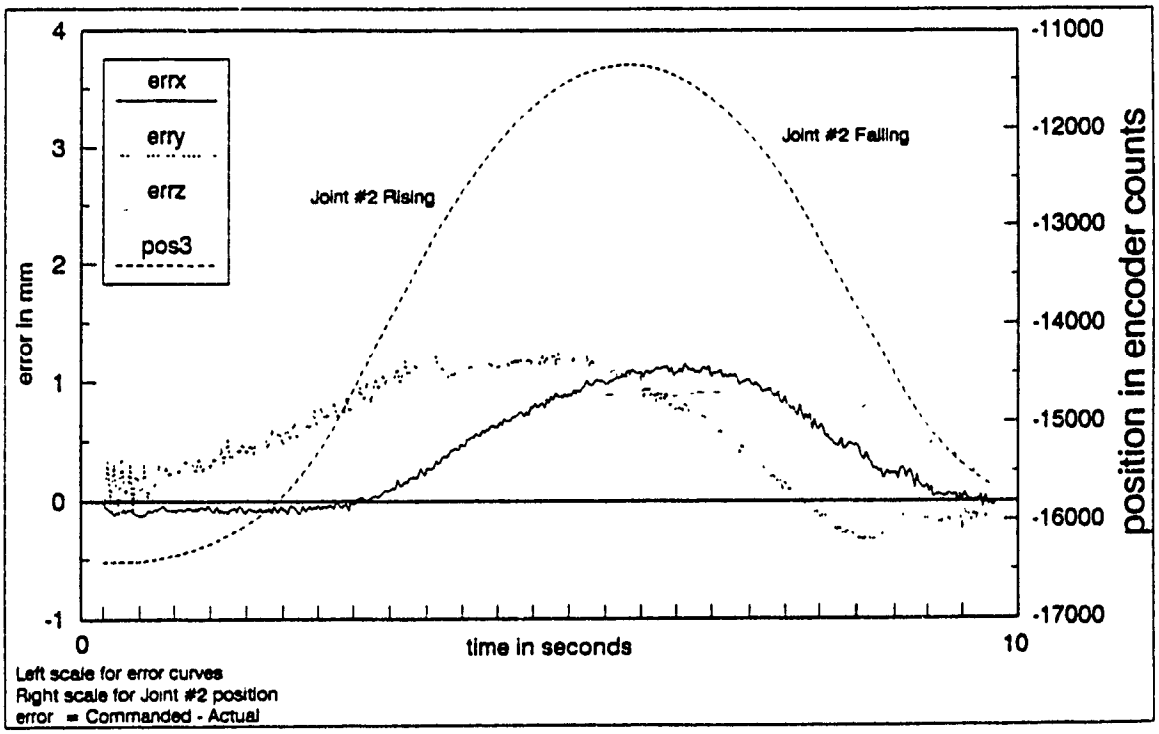

Figure 5.7 Effect of dynamics on accuracy in position

purely kinematic analysis for moving the robot, the effect of the manipulator dynamics is expected to induce positional errors. This was mentioned earlier in section 4.6. Figure 5.7 confirms this observation. It is clear that the error in each coordinate increases when the shoulder, i.e joint #2, lifts against the gravitational force to a maximum of ~ 1mm and reduces when the joint falls. As a second experiment, a curve with frequent changes in curvature was chosen, as follows.

5.4.3.2 Motion along a Polar Arc

Figure 5.8 shows the constructed path using cubic spline method for a 3D arc corresponding to a polar function given by

$$x_i = R_\rho \sin(2\Sigma'_0\, \delta\xi)\cos(\Sigma'_0\, \delta\xi) + x_0 \qquad (5.32a)$$

$$y_i = R_\rho \sin(2\Sigma'_0\, \delta\xi)\sin(\Sigma'_0\, \delta\xi) + y_0 \qquad (5.32b)$$

Z is taken to be

$$z_i = P_z(\Sigma'_0\, \delta\xi) + z_0 \qquad (5.32c)$$

where

$$\delta\xi = 2\pi/N$$

For Figure 5.8, R_ρ = 100mm, P_z = 1.6mm and N = 60. As can be seen, inspite of frequent curvature changes, the constructed path is continuous and smooth. Figure 5.9 show the actual arc traced by the EE for a Puma 260 robot. The positional errors reach maximum (~ 1mm)at the extremes of the arc when the effects of dynamics is significant because of the extended configuration of the arm. Figure 5.10 shows the speed profile for three different speeds. The **error in speed regulation is less than 5%** till reasonably high speed of 60 mm/sec. At higher speed of 100 mm/sec, the error is less than 10% inspite

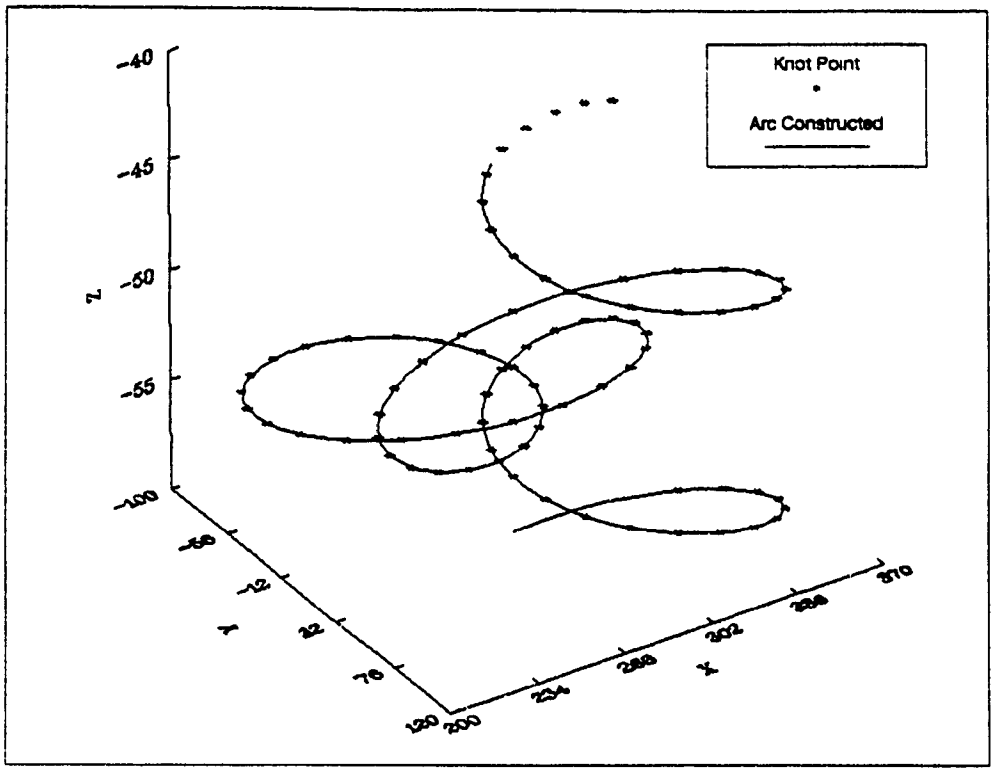

Figure 5.8 Polar Arc from knot points.

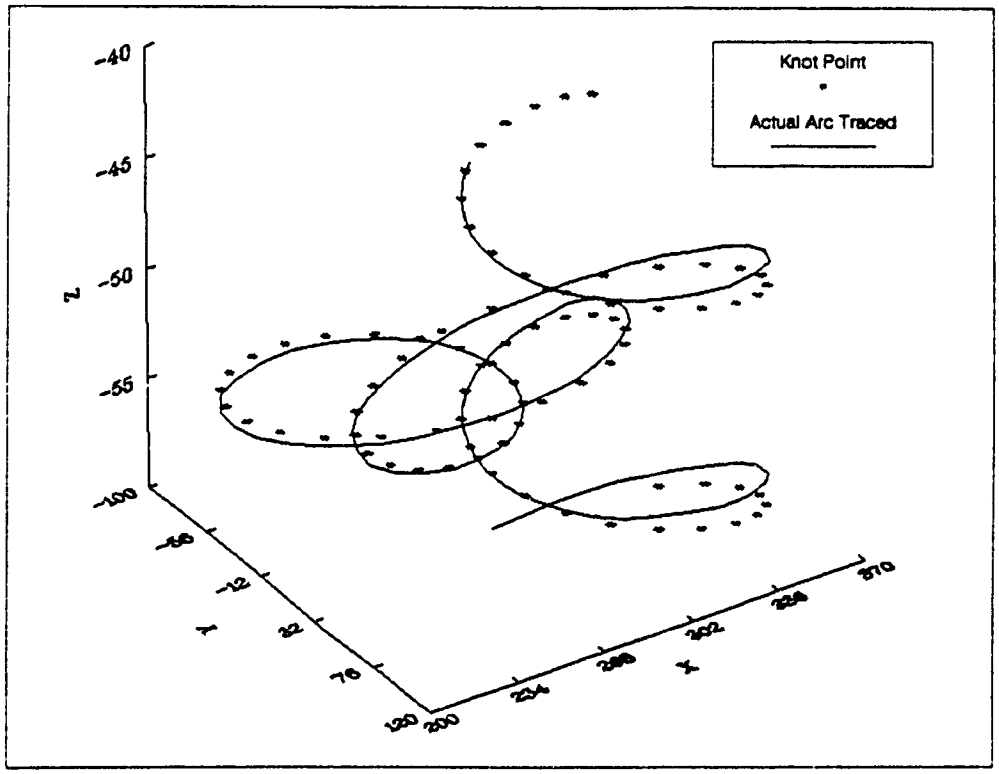

Figure 5.9 Actual path traced for knot points on the polar arc.

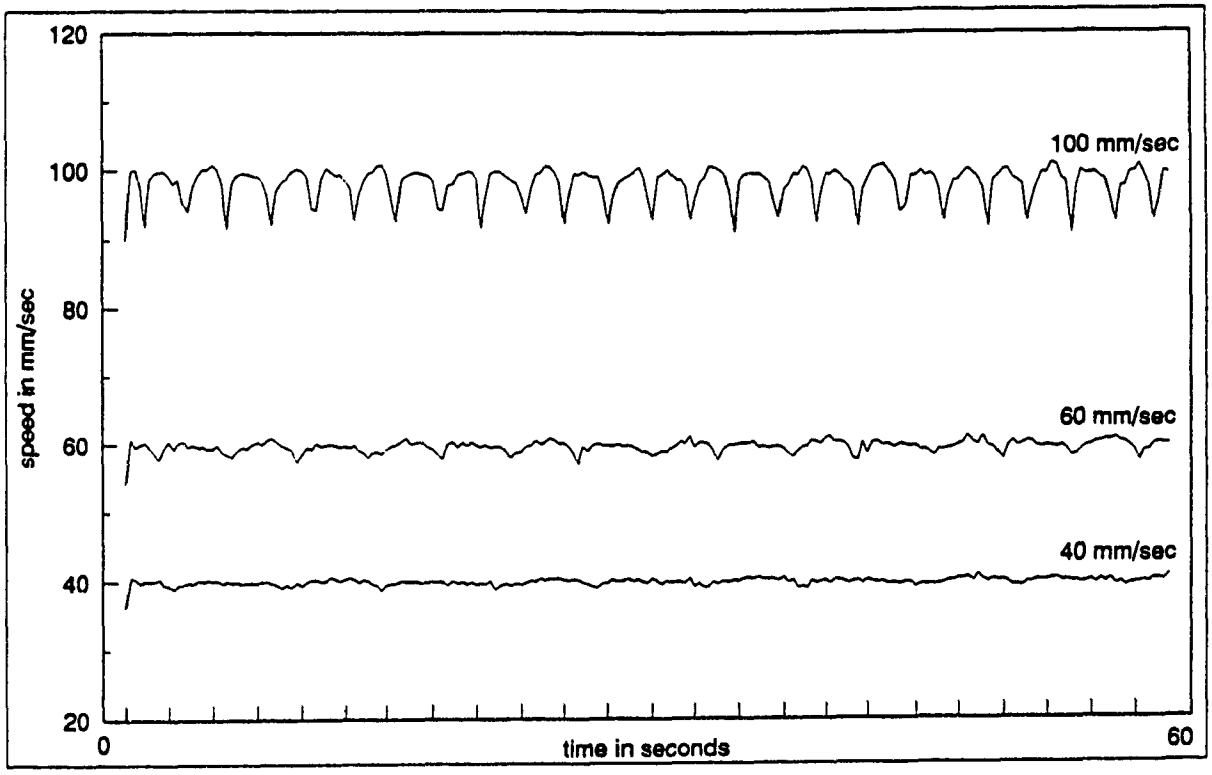

Figure 5.10 Speed Profile along the polar arc

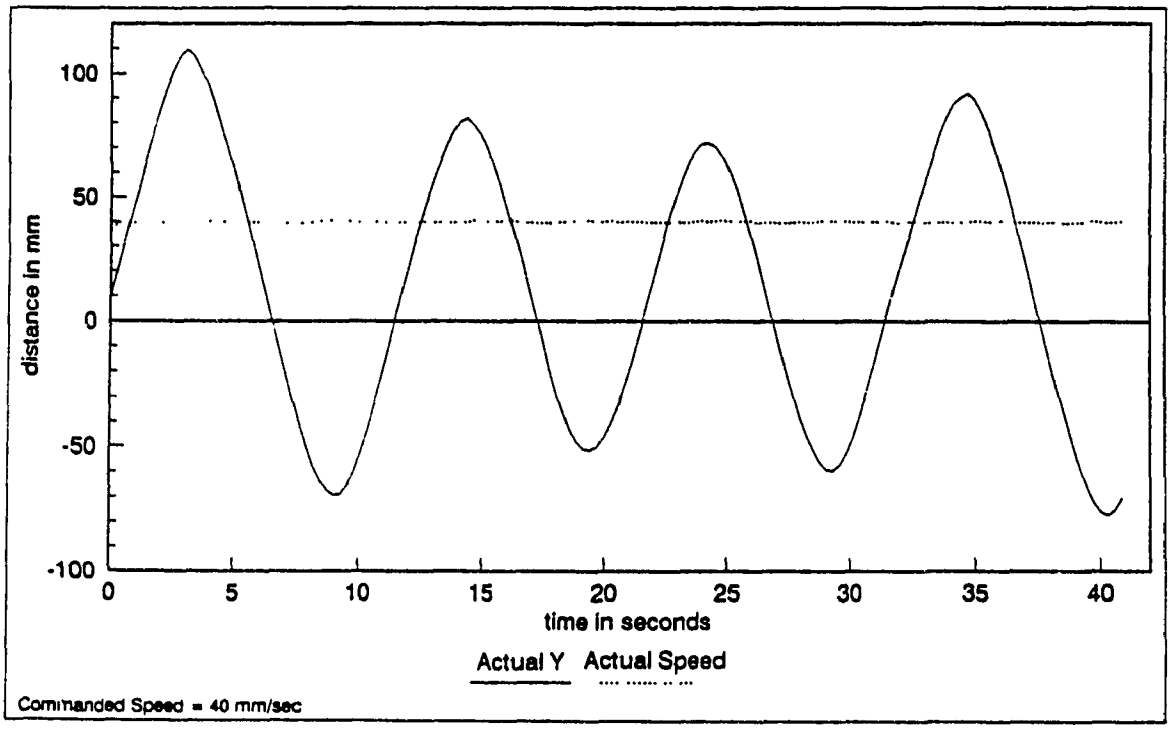

Figure 5.11 Speed profile under on-line path modification

of frequent changes in the velocity direction due to the particular nature of the arc.

5.4.3.3 Speed Regulation for on-line Path Modification

To test the performance of the algorithm under on-line path modification, major axis of the elliptical helix, R_y in (5.31b), was changed in real-time. In implementation, R_y is determined at run time by reading the output volatge of a dial, which is under user control. By moving the dial the knot points generated can be modified to lie on a family of ellipses of decreasing major axis terminating in a circle. Figure 5.11 shows the results of the experiment for a commanded speed of 40 mm/sec. It is clear that inspite of on-line changes in R_y, the manipulator speed is constant with no perceivable deviations. These results firmly establish the success of the algorithm for regulated speed manipulator motion along on-line determined cartesian paths.

5.5 Extension for EE Orientation

In section 5.4, the term cartesian path was restricted to imply the 3D arc in space traced by the EE poistion. However, control of tool position only is not enough for many industrial applications. As an example, in arc welding the welding gun must mantain a desired orientation with respect to the seam being welded. To mantain a desired EE orientation, the *trihedron* of arc Γ, [5.38] as shown in Figure 5.12 has to be determined. The trihedron of a curve C at a point P is the right handed coordinate frame determined by the tangent **u**, principal normal **p** and the binormal vector **b** at the point P. Once **[u p b]** triplet is determined, one can specify normal **n**, approach **a** and slide **s** vectors for EE orientation in trihedrom system and achieve the desired orientation. In this section, we attempt to determine **[u p b]** triplet for the spline constructed path discussed before.

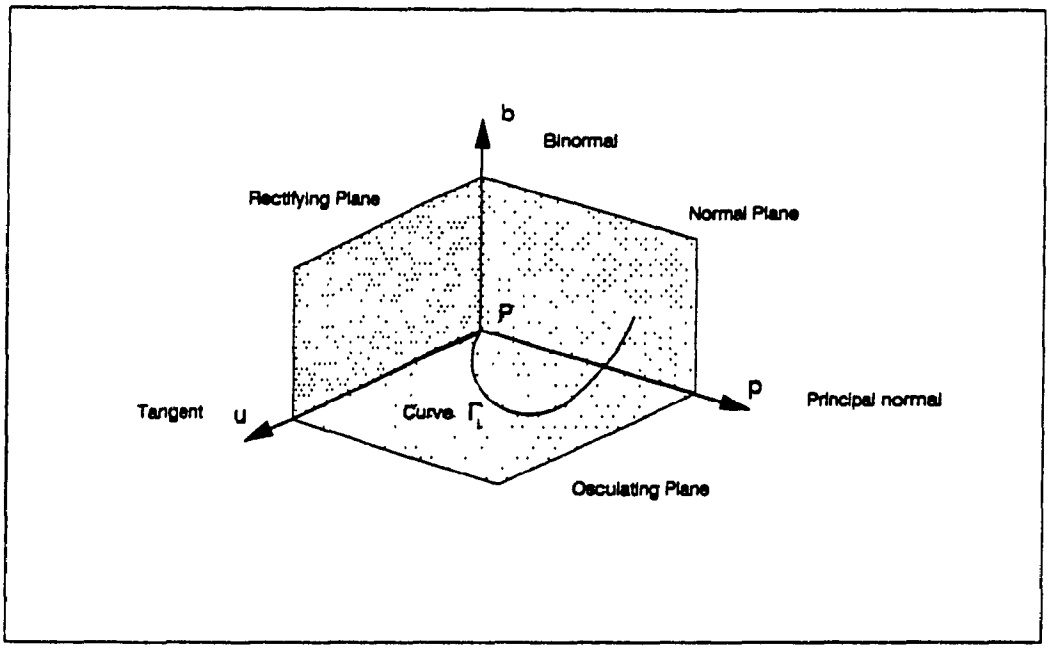

Figure 5.12 Trihedron.

As in the last section, in the discussion below $\tilde{\tau}_k$ denotes the value of τ used for the k^{th} sampling interval. Let the manipulator be on arc Γ_i at k^{th} interval. Ψ_i denotes the vector (parametric) form of arc Γ_i, i.e

$$\Psi_i = [\ x_i(\tau),\ y_i(\tau),\ z_i(\tau)\ ]^T$$

Then the unit tangent vector **u** at $\tau = \tilde{\tau}_k$ is given by

$$u = \frac{\left.\frac{d\Psi_i(\tau)}{d\tau}\right|_{\tau=t_i}}{\left\|\left.\frac{d\Psi_i(\tau)}{d\tau}\right|_{\tau=t_i}\right\|} \tag{5.33}$$

which can be determined easily as $x_i(\tau)$, $y_i(\tau)$ and $z_i(\tau)$ are known and of the form (5.10). However, if arc length s is used as a parameter in parametric representation of Ψ_i, then

u can be expressed as

$$u = \frac{d\Psi_i(s)}{ds} = \frac{d\Psi_i(\tau)}{d\tau}\frac{d\tau}{ds} \tag{5.34}$$

Thus the value of dτ/ds, which is used later, can be determined as

$$\frac{d\tau}{ds}\Big|_{\tau=\hat{t}_i} = \frac{1}{\left\|\frac{d\Psi_i(\tau)}{d\tau}\Big|_{\tau=\hat{t}_i}\right\|} \tag{5.35}$$

For an arc $\Psi_i(s)$, with arc length s as the parameter, the curvature κ(s) is defined as

$$\kappa(s) = |\,\mathbf{u}'(s)\,| = |\,\Psi_i''(s)\,| \tag{5.36}$$

If $\kappa(s) \neq 0$, then the unit principal normal vector **p** is given by

$$\mathbf{p} = \mathbf{u}'(s)/\kappa$$

Though in the present case Γ_i is represented by $\Psi_i(\tau)$, where **τ is not the arc length, p** can be determined as follows. Consider for one coordinate say x, then

$$dx/ds = (dx/d\tau)(d\tau/ds)$$

where, dx/dτ can be determined using (5.10). Using product rule we get

$$\frac{d^2x}{ds^2} = \frac{d}{ds}\left[\frac{dx}{d\tau}\frac{d\tau}{ds}\right] = \frac{d^2x}{d\tau^2}\left(\frac{d\tau}{ds}\right)^2 + \frac{dx}{d\tau}\frac{d^2\tau}{ds^2} \tag{5.37}$$

Thus

$$\frac{d^2x(\tau)}{ds^2}\Big|_{\tau=\hat{t}_i} = \left[\frac{d^2x(\tau)}{d\tau^2}\Big|_{\tau=\hat{t}_i}\right]\left[\frac{d\tau}{ds}\Big|_{\tau=\hat{t}_i}\right]^2 + \left[\frac{dx}{d\tau}\Big|_{\tau=\hat{t}_i}\right]\left[\frac{d^2\tau}{ds^2}\Big|_{\tau=\hat{t}_i}\right]$$

where

$\frac{d^2x(\tau)}{d\tau^2}\Big|_{\tau=\hat{t}_i}$, $\frac{dx(\tau)}{d\tau}\Big|_{\tau=\hat{t}_i}$ *are determined by* (5.10).

$\frac{d\tau}{ds}\Big|_{\tau=\hat{t}_i}$ *is determined by* (5.35) (5.38)

To evaluate $d^2\tau/ds^2$, we proceed as follows. Using (5.21)

$$\delta s = s_i(\tilde{\tau}_k + \delta\tau) - s_i(\tilde{\tau}_k) \tag{5.39}$$

Then

$$\left.\frac{d^2\tau}{ds^2}\right|_{\tau=t_i} = \lim_{\delta\tau,\, \delta s\to 0} \frac{\left.\frac{d\tau}{ds}\right|_{\tau=t_i+\delta\tau} - \left.\frac{d\tau}{ds}\right|_{\tau=t_i}}{\left.\delta s\right|_{\tau=t_i}} \tag{5.40}$$

where

$\left.\frac{d\tau}{ds}\right|_{\tau=t_i+\delta\tau}$, $\left.\frac{d\tau}{ds}\right|_{\tau=t_i}$ *are determined by* (5.35)

δs *is determined by* (5.39)

Thus, using (5.38) $d^2\Psi_i/ds^2$ at $\tau = \tilde{\tau}_k$ is evaluated and by normalizing it the unit vector **p** can be determined.

Once **u** and **p** are known, the unit binormal vector is given by their cross product,

$$\mathbf{b} = \mathbf{u} \times \mathbf{p}$$

and the trihedron coordinate frame is completely known. As an illustration, suppose that it is desired that **a** = -**b** orientation be mantained when the EE moves on Γ_i. Then, the triplet **[u p b]** at $\tau = \tilde{\tau}_k$ and at $\tau = \tilde{\tau}_{k+1}$ is evaluated before moving from $\sigma_{k,i}$ to $\sigma_{k+1,i}$ set point. The desired orientation i.e. **[n s a]** can be evaluated at $\tau = \tilde{\tau}_k$ and $\tau = \tilde{\tau}_{k+1}$ by taking

$$\mathbf{a} = -\mathbf{b},\ \mathbf{s} = \mathbf{p} \text{ and } \mathbf{n} = \mathbf{s} \times \mathbf{a}$$

Using inverse kinematics, incremental displacement for the wrist joints in the k^{th} sampling period can be evaluated.

5.6 Summary

To summarise this chapter, a time efficient algorithm for path planning in cartesian

space for manipulator continuous path applications has been discussed. Using cubic spline formulation, smooth path is constructed from selective knot points. The arm is guided along the constructed path at a controlled speed using arc-length parameterisation. The experiments demonstrate the success of the algorithm in mantaining a high accuracy in manipulator speed under on-line path identification. Extension of the technique to include EE orientation control has also been discussed.

Chapter 6

Multi-Arm Synchronization : Implementation and Results

6.1 Introduction

The design and architecture of the IWC was discussed in a general manner in Chapter 3. Chapter 4 and 5 discussed some of the constituent blocks, namely the JSC and CPC respectively, in greater details. As a typical illustration of an integrated workcell concept, this chapter analyzes the experimental results demonstrating the success of the IWC architecture in multiple manipulator synchronization and control.

Extensive theoretical studies have been done towards path finding in multi-robot systems [6.1]-[6.6]. However, the focus has been mainly on *loosely coordinated motion* when the robots share a common workspace but execute independent tasks. In such a case, failure of one robot's motion does not affect the motion execution of the second robot. Lots of industrial applications like assembly of parts, transfer of tool etc. require *tightly coordinated motion* between the various robots, where the motion of one arm is dependent and to an extent determined on-line by the motion of the coordinating arm.

Alfrod and Belyeu [6.7] used two manipulators to move an object along a predetermined path which lies in a plane parallel to the X-Y plane of each arms base coordinate frame, without considering the kinematic relationships between the cooperating manipulators. The kinematic constraints imposed for two coordinating manipulators have been studied by [6.8]-[6.10]. In most of the cases, a leader/follower arrangement is used

where the motion of follower is obtained from that of the leader. Tao et. al. [6.11] have incorporated compliant control to coordinate two moving industrial robots. The problem of two robots grasping/pushing the same object has also been analyzed by [6.12]-[6.13].

However, in all the literature surveyed the algorithm starts from a *static* scene i.e. an initial alignment between the two cooperating manipulators is assumed to have been reached. The problem of reaching the initial synchronisation between the two robots under *dynamic* conditions, i.e. when the leader arm carrying the workpiece is already in motion, has not been addressed. This issue of initial synchronisation is further complicated if the leader manipulator is subjected to on-line path modification and/or speed modifications so as to avoid obstacles and synchronise with other motion devices downstream on the assembly line. For multi-manipulator synchronisation using the IWC, a typical application identified is to achieve synchronization in the following scenario :

> *The workpiece manipulator is pursuing an independent path at a regulated speed. This path is determined only to the extent of a few look-ahead knot points and the manipulator speed may vary on-line. When triggered from an external source the second manipulator, referred to as the tool, should synchronize with the workpiece so as to traverse the same path (with required offsets) and at the same speed.*

In this chapter, the path of the manipulator is restricted to imply the cartesian position of the EE only. The orientation of the workpiece EE is kept fixed and taken complimentary for the tool EE. In the current implementation, the PUMA 260 robot is taken as the workpiece manipulator and the PUMA 560 robot is taken as the tool manipulator.

6.2 Minimum-Time Model for Two-Arm Synchronization

In the two-arm synchronization problem under consideration, the workpiece path is taken as independent and generated on-line from a few look ahead knot-points, using the techniques discussed in the last chapter. However, a systematic formulation is required for identifying the tool path to ensure synchronization in a pre-determined time interval. Earlier coordination between a manipulator and other motion devices such as conveyor belt, x-y positioning table etc. has been discussed by [6.14]-[6.21]. Specifically, a discrete time model suitable for conveyor belt tracking was developed by Cheng and Poon [6.14]. However in the present case, unlike a conveyor belt, the motion of a workpiece manipulator is not restricted along a fixed direction and thus a synchronization model permitting greater freedom needs to be developed.

Proceeding on lines similar to [6.14], in this section we develop a *minimum-time* synchronization model in which the tool manipulator EE moves along the straight line joining the two end effectors, to achieve interception in minimum time. For analysis, the model can be split into two phases, namely *interception phase* and *synchronization phase.* These are discussed in the following sections.

6.2.1 Interception Phase

Let k = 0 when interception phase starts, i.e. the time t_o when the external trigger is received to start tool motion so as to synchronise with the workpiece. In the discussion below, we use the following notation :

$\mathbf{P}_w[k]$: Workpiece position vector at time $t = kT_c + t_o$.

$\mathbf{P}_t[k]$: Tool position vector at time $t = kT_c + t_o$.

$\mathbf{V}_W[k], S_W[k]$: Workpiece velocity/speed for the k^{th} interval.

$\mathbf{V}_T[k], S_T[k]$: Tool velocity/speed for the k^{th} sampling interval.

$\mathbf{P}_{WT}[l,j] = \mathbf{P}_W[l] - \mathbf{P}_T[j]$.

where, all the vectors are expressed in one universal coordinate frame common between both the manipulators.

In minimum-time synchronization model, under the assumption of non-interfering arm configurations, in the k^{th} interval one directs the tool path along the vector $\mathbf{P}_{WT}[k+1,k]$ i.e. along the straight line joining the *current* tool position and workpiece position *expected* at the end of the interval as shown in Figure 6.1. It is obvious that if the tool is moved along $\mathbf{P}_{WT}[k,k]$, then it will always lag behind the workpiece unless $\mathbf{V}_W[k] = 0$, which can not be assumed. We have

$$\mathbf{P}_{WT}[k+1,k] = \mathbf{P}_W[k] + T_c\mathbf{V}_W[k] - \mathbf{P}_T[k] \tag{6.1}$$

and the line OB in Figure 6.1 can be written in parametric form as follows

$$\frac{x - P_{TX}[k]}{P_{WX}[k] + T_c V_{WX}[k] - P_{TX}[k]} = \frac{y - P_{TY}[k]}{P_{WY}[k] + T_c V_{WY}[k] - P_{TY}[k]} = \frac{z - P_{TZ}[k]}{P_{WZ}[k] + T_c V_{WZ}[k] - P_{TZ}[k]} = C \tag{6.2}$$

where C is determined from the maximum time allowed for interception phase in the manner to be discussed.

If it is desired that the interception phase takes a maximum of p sampling intervals, then for the best case of stationary workpiece i.e. $\mathbf{V}_W[k] = 0\ \forall k$, which is rather trivial, the minimum average speed of the tool along (6.2) should be,

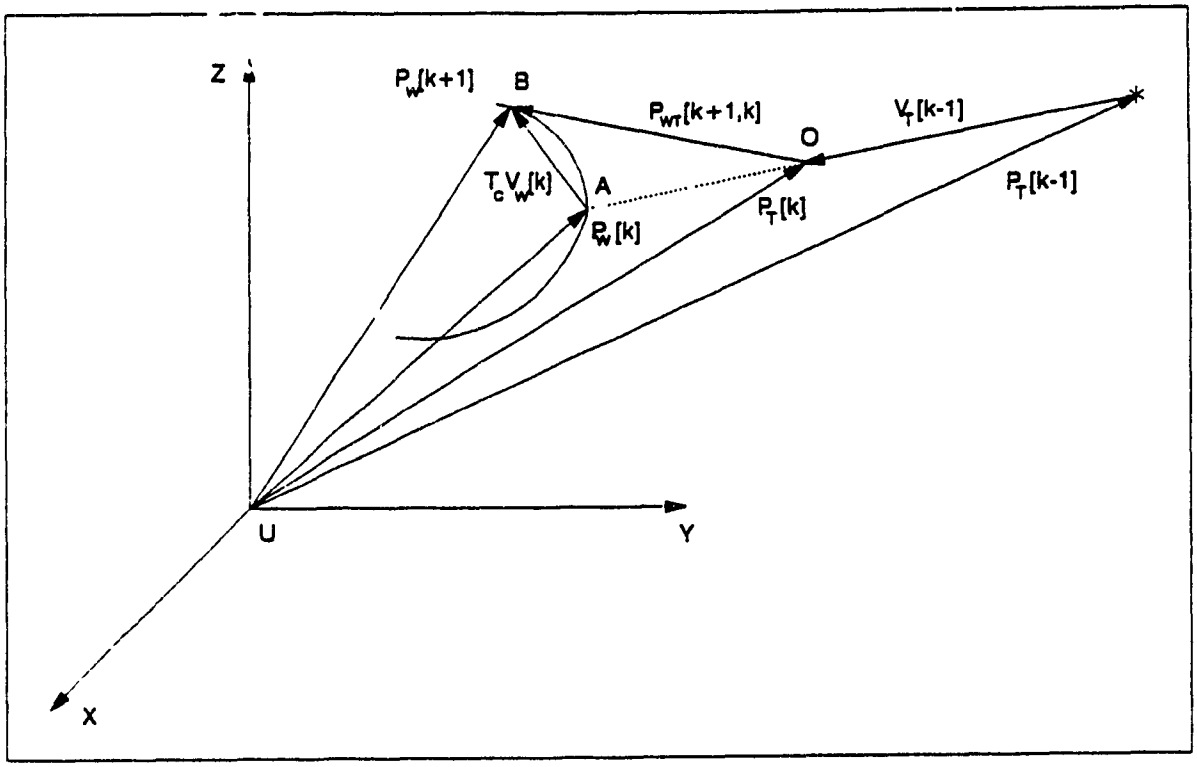

Figure 6.1 Tool path along $\mathbf{P}_{WT}[k+1,k]$

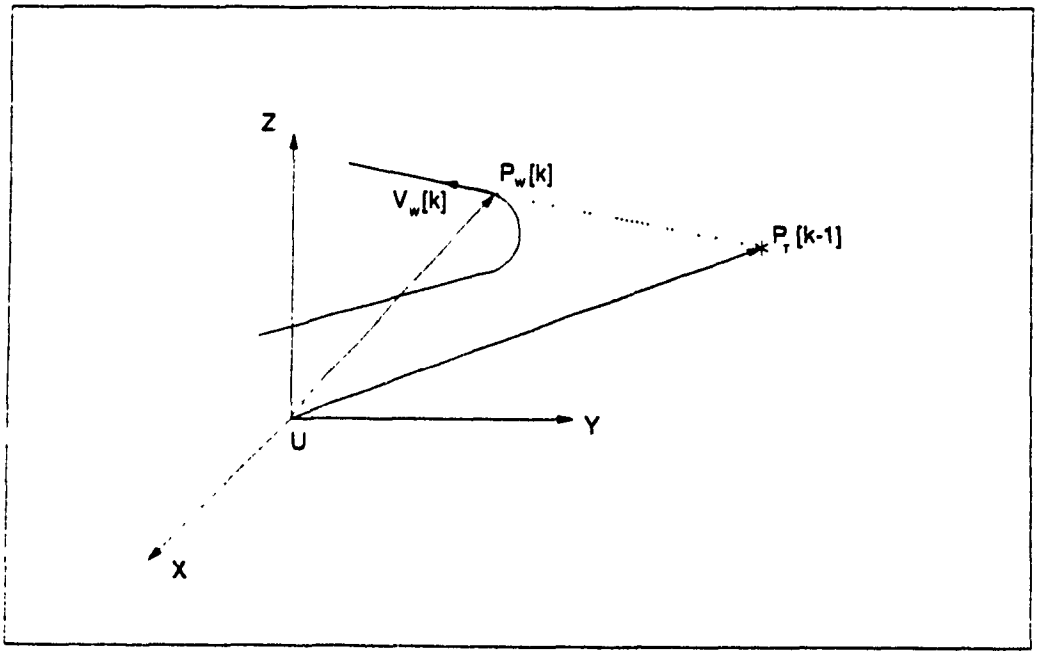

Figure 6.2 $\mathbf{V}_w[k]$ along $\mathbf{P}_{WT}[k,k-1]$

$$S_{T,min} = \|\mathbf{P}_{WT}[1,0]\| /(pT_c) = \|\mathbf{P}_{WT}[0,0]\| /(pT_c)$$

For the worst case when $\mathbf{V}_W[k]$ is along $\mathbf{P}_{WT}[k,k-1]$ for all later times, as shown in Figure 6.2, the speed of the tool manipulator will have to be faster than that of workpiece by a factor $\alpha[k]$, given by

$$\alpha[k] = (S_W[k] + S_{T,min}) / S_W[k] \tag{6.3}$$

Thus if the tool is moved along $\mathbf{P}_{WT}[k+1,k]$ with the speed

$$S_T[k] = \alpha[k]S_W[k]$$

then interception is assured in less than or equal to p intervals for any direction of $\mathbf{V}_W[k]$. In actual practice, as a non-zero minimum speed of the workpiece $S_{W,min}$ always exists, a constant maximum value of α can be chosen as follows

$$\alpha = (S_{W,min} + S_{T,min})/ S_{W,min}$$

Thus, during the interception phase the tool should be moved along $\mathbf{P}_{WI}[k+1,k]$ at a speed of $\alpha S_W[k]$. Let

$$\delta l_W[k] = V_W[k]\, T_c$$

and

$$\delta l_T[k] = \alpha \delta l_W[k]$$

Using the normalised distance as the parameter in (6.2), $\mathbf{P}_I[k+1]$ i.e. the next set point for tool manipulator can be determined by

$$\frac{P_{TX}[k+1] - P_{TX}}{P_{WX}[k] + T_c V_{WX}[k] - P_{TX}[k]} = \frac{P_{TY}[k+1] - P_{TY}}{P_{WY}[k] + T_c V_{WY}[k] - P_{TY}[k]} = \frac{P_{TZ}[k+1] - P_{TZ}}{P_{WZ}[k] + T_c V_{WZ}[k] - P_{TZ}[k]} = \frac{\delta l_T[k]}{\|P_{WI}[k+1,k]\|} \tag{6.4}$$

To prevent tool overshooting, the interception phase is terminated at k=pp when

$$\delta l_T[pp] > \|\mathbf{P}_{WT}[pp+1,pp]\|$$

is satisfied and the synchronisation phase starts.

6.2.2 Synchronisation Phase

In this phase, the aim is to synchronise the tool manipulator so that it moves along the workpiece path, with due offsets, with the same speed. In the j^{th} interval during synchronisation phase i.e. j = pp, pp+1, pp+2,..., we define the following unit vectors, with reference to Figure 6.3.

$$\hat{u}[j] = \frac{P_w[j+1] - P_T[j]}{\|P_w[j+1] - P_T[j]\|}$$
$$\hat{v}[j] = \frac{P_w[j+2] - P_T[j]}{\|P_w[j+2] - P_T[j]\|} \qquad (6.5)$$

To prevent an abrupt change in the direction of tool velocity, we look at the angle between $\mathbf{V}_W[j+1]$ and $\hat{\mathbf{u}}[j]$ given by

$$\phi[j] = \arccos(\ \mathbf{V}_W[j+1]\cdot\hat{\mathbf{u}}[j]\ )$$

Let ϕ_c be the user defined critical angle beyond which the change in direction of the tool manipulator in one sampling interval is not acceptable. Depending on the magnitude of ϕ, two conditions arise as follows :

(i) **C1 : $\phi[j] < \phi_c$**

This is the simpler case in which synchronisation is attained by taking

$$\mathbf{P}_T[j] = \mathbf{P}_W[j] + \mathbf{P}_{offset} \qquad (6.6)$$

for all the subsequent intervals. Here $\mathbf{P}_{offset}$ is a user specified pre-determined

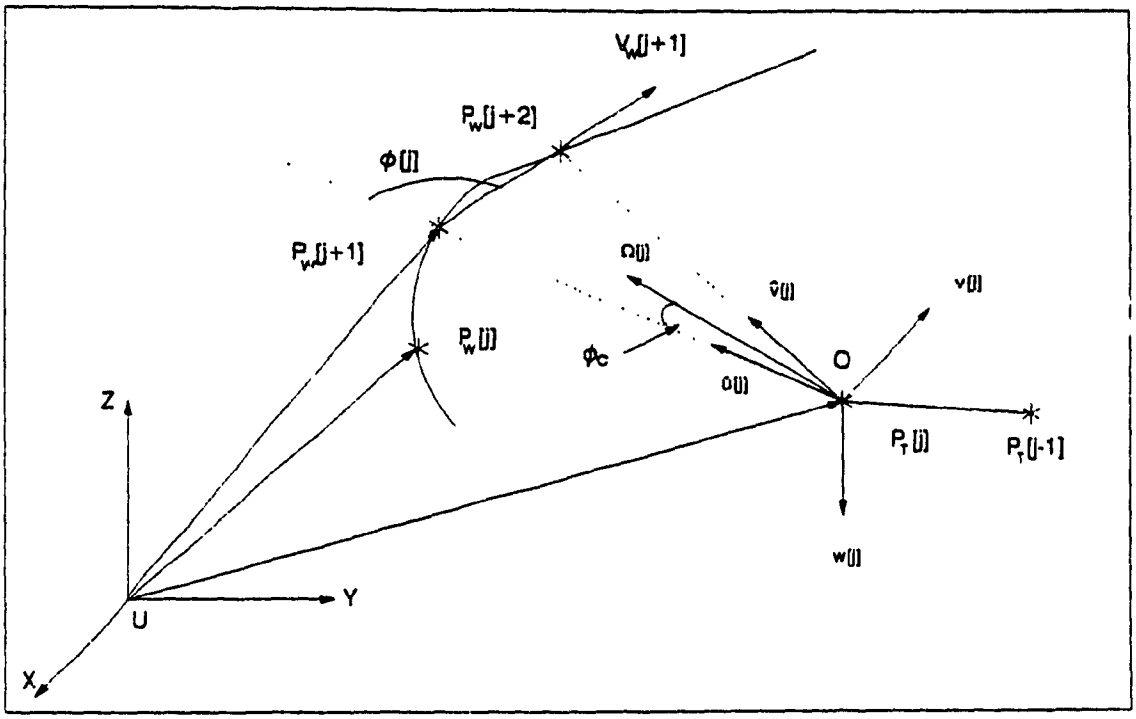

Figure 6.3 $\hat{u}[j]$, $v[j]$ and $w[j]$ coordinate system

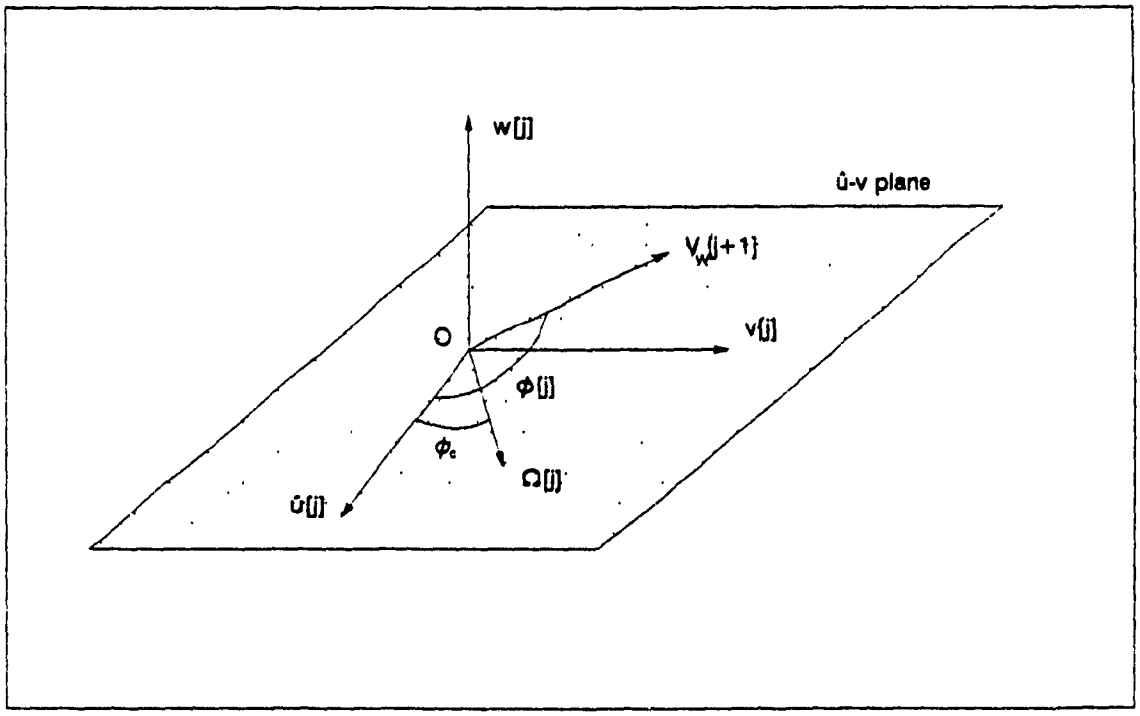

Figure 6.4 Vector $\Omega[j]$

offset vector.

(ii) **C2 : $\phi[j] > \phi_c$**

In this case, the direction of tool manipulator has to be changed gradually until C1 is satisfied. A right-handed coordinate frame can be attached at point O in Figure 6.3 with unit vectors $\hat{\mathbf{u}}[j]$, $\mathbf{v}[j]$ and $\mathbf{w}[j]$, where

$$\begin{aligned} w[j] &= \hat{u}[j] \; X \; \hat{v}[j] \\ v[j] &= w[j] \; X \; \hat{u}[j] \end{aligned} \tag{6.7}$$

such that the $\hat{u}$-v plane contains the vector $\mathbf{V}_w[j+1]$. Let N_j be the integer such that

$$(N_j-1)\phi_c \le \phi[j] < N_j\phi_c$$

We attempt to turn the direction of motion of the tool manipulator in the $\hat{u}$-v plane by an angle no more than ϕ_c for N_j subsequent intervals to achieve synchronisation. Let

$$L[j] = \|\mathbf{P}_{WI}[j+1,j]\|$$

Then we seek the vector $\Omega[j]$ in $\hat{u}$-v-w frame which makes an angle ϕ_c with $\hat{\mathbf{u}}$ and has a magnitude of $L[j]/N_j$ as shown in Figure 6.4. The magnitude of $\Omega[j]$ is so chosen so as to prevent overshooting of tool position, which may need sudden reversal of tool direction at some later time. $\Omega[j]$ in the $\hat{u}$-v-w frame is given by

$$\Omega[j] = (L[j]/N_j)\cos\phi_c \; \hat{\mathbf{u}}[j] + (L[j]/N_j)\sin\phi_c \; \mathbf{v}[j] \tag{6.8}$$

Using $\hat{\mathbf{u}}[j]$ and $\mathbf{v}[j]$ from (6.6)-(6.7) in the above, the next set point for the tool manipulator $\mathbf{P}_T[j+1]$ is determined in the universal frame. It is clear that under the assumption that the curvature of workpiece path changes gradually, i.e.

$$\delta_w\phi[j] = \arccos(\mathbf{V}_w[j+1]\cdot\mathbf{V}_w[j+2]) < \phi_c \qquad \forall j$$

then $\phi[j]$ will decrease continuously till condition C1 is satisfied. From that point onwards, (6.6) is used to guide both the manipulators and synchronization is attained.

6.3 Implementation of IWC for Two-arm control

Before considering the experimental results in two-arm synchronization, we look at the implementation specifics of the IWC for control of two manipulators. This will also give the discussion of Chapter 3 a firm footing.

6.3.1 Hardware Architecture : Transputer Network Implementation

The real-time computational demands of the synchronization problem, as outlined before, is not very high. As a result to use independent processors for GTPC and CPC (for each robot) leads to a significant waste of computing power. Hence in actual implementation, a GTPC cluster of two transputers was used to perform the functions of operator interaction and cartesian path planning for both the manipulators.

The resulting transputer network configuration consisting of six processors is shown in Figure 6.5 where the symbolic name given to each processor is indicated outside the block. In the same figure, the task mapped on the specific processor is labelled inside the block. It is important to note that although the IWC architecture demands only four transputers, i.e. two JSCs and GTPC cluster of two, two extra processors are required to provide debugging support during the development cycle. These are processors ROOT and IOSERV in Figure 6.5. The reason for this has been discussed in Section 3.2.2.3. Apart from task CPCTWIN, which interacts with the operator, such an arrangement provides two free *debugging* links. These can be connected to two different tasks allowing the use

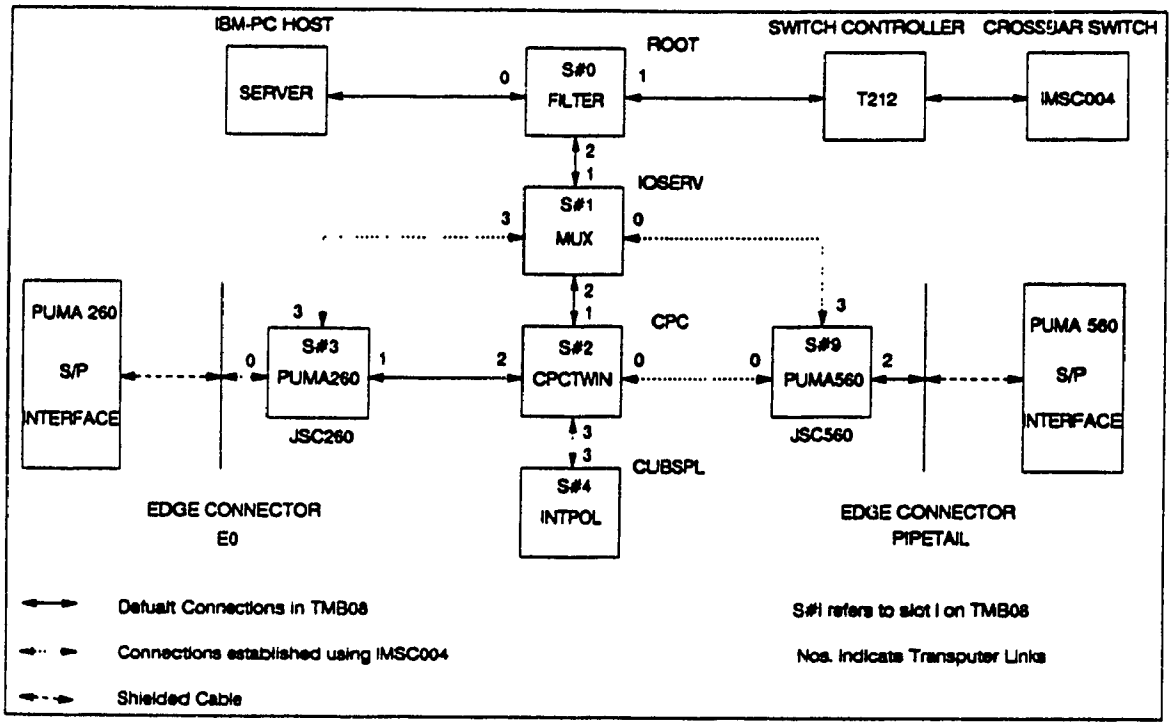

Figure 6.5 Transputer network for Two-Arm implementation

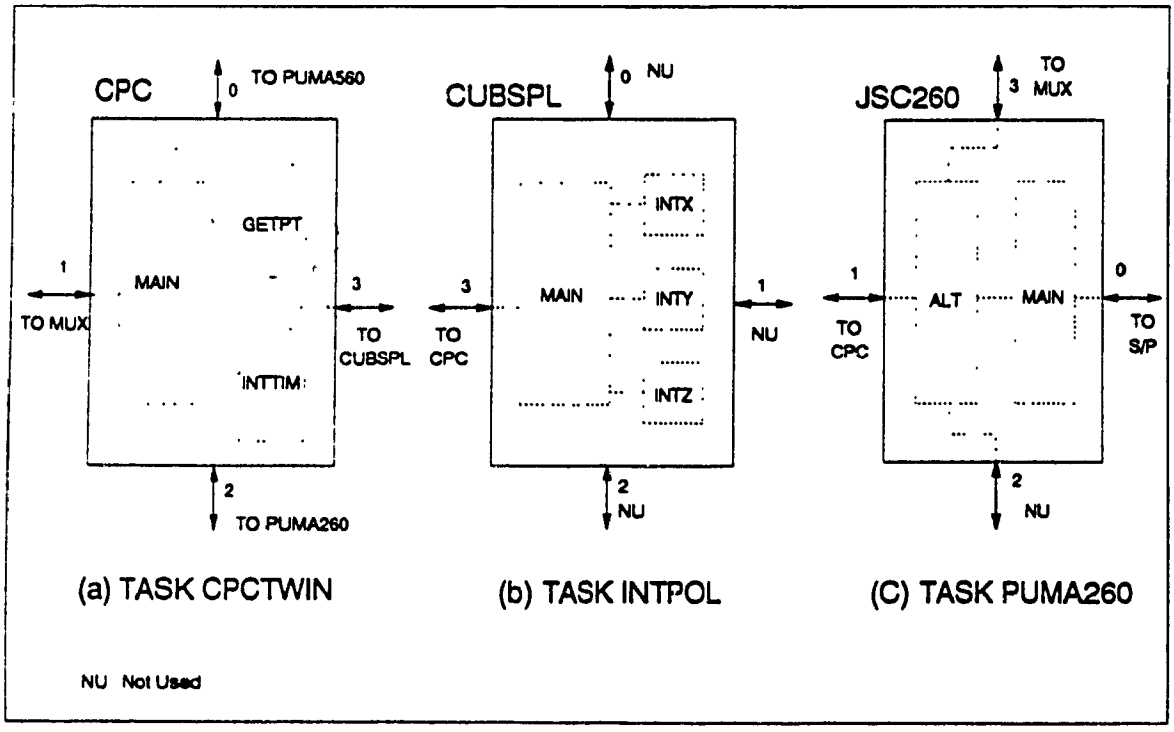

Figure 6.6 Internal Structure of TASKs

Table 6.1 Link Interconnections for Two-Arm implementation

Link Interconnection			Communicating Tasks
No.	Processor From	Processor To	End Point Tasks
1	ROOT #0	PC	FILTER - SERVER
2	ROOT #1	T212	GMS Control Link
3	ROOT #2	IOSERV #1	SERVER - MUX
4	IOSERV #2	CPC #1	CPCTWIN - SERVER (OPERATOR interaction)
5	IOSERV #0	JSC560 #3	SERVER - PUMA560 (Debugging Link)
6	IOSERV #3	JSC260 #3	SERVER - PUMA260 (Debugging Link)
7	CPC #2	JSC260 #1	CPCTWIN - JSC260
8	CPC #0	JSC560 #0	CPCTWIN - JSC560
9	CPC #3	CUBSPL #3	CPCTWIN - INTPOL
10	JSC260 #0	E0 (Edge Connector)	PUMA260 - S/P INTERFACE
11	JSC560 #2	PIPE TAIL (Edge Connector)	PUMA560 - S/P INTERFACE

of Parallel C run-time library (which includes I/O functions like *printf* etc.) in these tasks. This provides the essential debugging support required in the development phase. However, in the end product the only task requiring the I/O support for operator interaction, i.e. CPCTWIN, can be mapped on ROOT and resulting network shall require only four processors. The link connections required for establishing transputer network shown in Figure 6.5 are given in Table 6.1. The table also gives the tasks communicating through the various links. The link connections 1,2,3,4,7 and 11 are default connections on the motherboard TMB08 (refer section 3.2.2.2). Connections 5,6,8,9 and 10 are

established by programming the GMS (IMS C004 Crossbar Switch) using the *ncs* utility described in the same section. Appendix F gives the configuration file and the ncs *connect* file for the implemented network. In the next section, we consider the parallelism built in the IWC software architecture by analyzing each task in greater details.

6.3.2 Software Architecture

The software of the IWC consists of different tasks which are mapped onto specific processors and execute in parallel (refer 2.8.3.1). The mapping and the link configuration between various tasks has been described in the last section. This section will describe the function of each task in detail.

6.3.2.1 Task CPCTWIN

This task runs on the main processor, i.e. processor CPC, of the GTPC cluster. Although in the current implementation, the GTPC and the individual CPC of each robot have been merged in hardware as one GTPC cluster, logical division has been maintained by writing modular software in the form of threads executing in parallel (refer 2.8.3.1). If in the future, the GTPC cluster is split into segregated processing blocks, the individual threads can be ported on the new processor with minimal changes. Recall that communication channels between various threads on the same processor are implemented using memory locations, as discussed in 2.8.3.1. CPCTWIN task consists of the following threads, as shown in Figure 6.6(a).

(i) **Thread INTTIM** : This is the software timer thread. It has been mentioned before in section 2.8.2.1.1, that capability to interrupt the transputer is lost in TRAM architecture. The INTTIM (*int*erval *tim*er) thread, which runs on URGENT

priority, emulates the function of a hardware clock which would interrupt the processor every T_c seconds and hence determine the controller sampling period. Using the transputer timer associated with an URGENT thread, INTTIM waits for T_c time interval and then tries to communicate to the MAIN thread discussed below. If the communication is successful in one timer tick, which corresponds to 1 microsecond, the thread loop backs into another wait state otherwise a suitable error message is given to the operator. It is noteworthy to mention that such a thread can be implemented because waiting on timer does not consume any processor time. Also, it should be noted that the value of T_c can be **tuned at run-time** for best performance. Logically, it can be considered as an integral part of the GTPC software.

(ii) **Thread MAIN** : It is the MAIN thread running on the CPC which interacts with the operator. After initialisation, on operator command it creates INTTIM on run-time and communicates with it every T_c seconds. During this interval, MAIN determines the path to be pursued by the tool manipulator and receives the same information for workpiece from the thread GETPT. On successful communication from INTTIM, it communicates this information i.e. the next cartesian set point, to the JSC of the corresponding arm. Logically it can be considered as an hybrid thread, a part of which will be mapped to the GTPC and another part to the indivdual arm's CPC software in a segregated architecture.

(iii) **Thread GETPT** : The GETPT thread, in conjunction with Task INTPOL, determines the workpiece path by generating the look ahead knot points, which

it communicates to the task INTPOL. One should recall that in the synchronization problem, as outlined before, the path of the workpiece is known only to the extent of few look-ahead points. The cubic spline technique, mentioned in the last chapter, is used to generate a smooth path from these knot points. Specifically, after receiving the user desired speed of the workpiece, GETPT determines the distance to be travelled in the current sampling period and communicates it to INTPOL. In return, it receives the coordinates of the next set point for the workpiece, which it forwards to MAIN. Logically it can be considered as an integral part of workpiece CPC software.

6.3.2.2 Task INTPOL

This task is mapped onto the CUBSPL processor of the GTPC cluster. It communicates with the GETPT thread (on processor CPC), from which it receives the next knot point for the workpiece path. Internally, the MAIN thread of the task INTPOL creates three independent threads, INTX, INTY and INTZ as shown in Figure 6.6(b). All the three threads execute the same function *onedim* except that the data on which they execute corresponds to the coordinate in the thread name. The function *onedim* generates the spline on the supplied data points using the technique described in the last chapter. It is easy to see that a new interpolation technique can be implemented by modifying the function *onedim* only.

The MAIN thread of the task INTPOL collects coefficients of the spline functions for each coordinate from individual threads and determines equation (5.21). It then receives the incremental length to be moved on the current spline from GETPT, solves

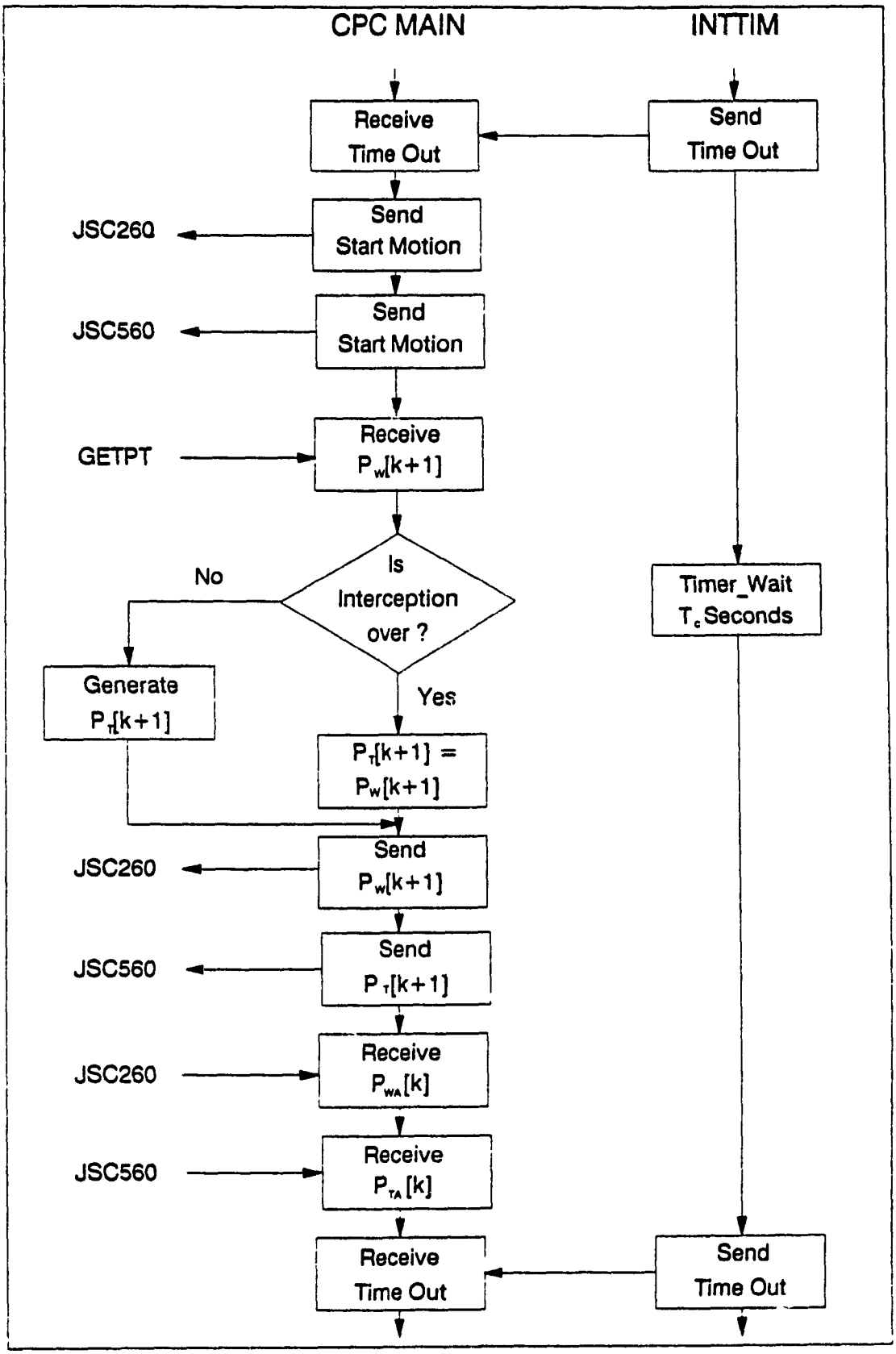

Figure 6.7 Communication with CPC MAIN thread.

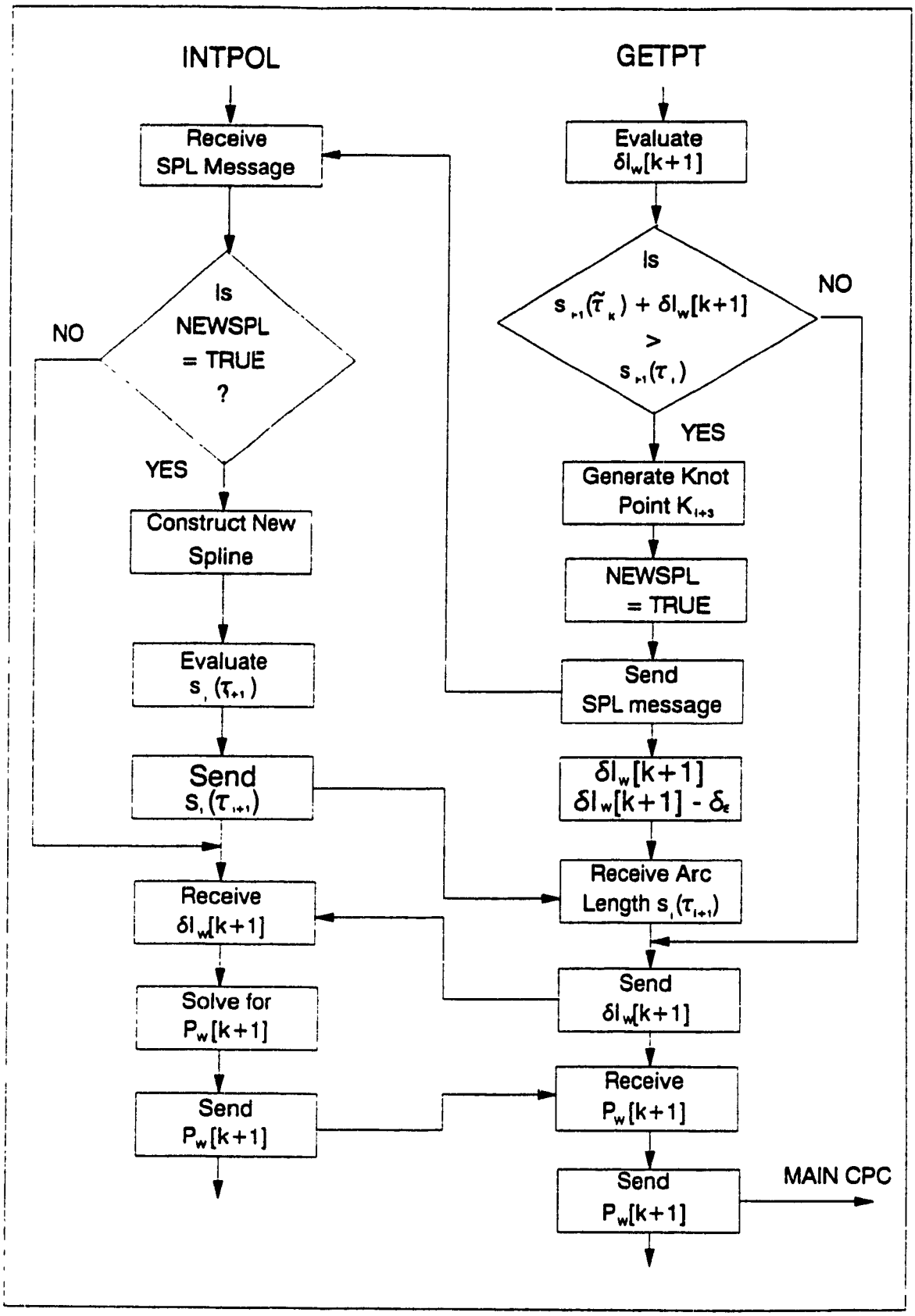

Figure 6.8 Communication between TASK INTPOL and GETPT thread.

equation (5.24) and returns the corresponding set point to GETPT. Logically, INTPOLis an integral part of the workpiece CPC software.

6.3.2.3 Task PUMA260

Task PUMA260 runs on the JSC260 processor. As discussed in detail in Chapter 4, the purpose of this task is to hide PUMA 260 robot arm characteristics by supporting the device-independent commands from the higher levels. It has a single MAIN thread. Though in the current implementation only CPCTWIN communicates with PUMA260, the latter uses the *alt_wait_vec* command, as shown in Figure 6.6(c), to determine the communicating port as discussed in Section 3.2.3.2.1.

6.3.2.4 Task PUMA560

Task PUMA560 runs on the JSC560 process. Functionally it is similar to task PUMA260 discussed above. As the two arms have different characteristics, the code of the two tasks is obviously different.

The tasks **FILTER** and **MUX** are vendor supplied and have been discussed in Section 3.2.2.3.

6.4 Inter-process Communication for Two-Arm Synchronisation

Figure 6.7 and 6.8 show the typical communication between various threads/tasks during the k-1th interval with the workpiece moving from knot point K_{i-1} to K_i. The notation $\mathbf{P}_{WA}[k]$ and $\mathbf{P}_{TA}[k]$ in the flowchart of CPC MAIN thread, is used to denote the actual position of workpiece and tool manipulator at the end of the k-1th interval as determined from forward kinematics. The *SEND* and *RECEIVE* communication primitives are implemented in a straight forward manner by using the communication channels of

```
cmd_type = MVEL_6JT;
send_header(TO260);
chan_out_message(sizeof(EE_DLOC),&ee260,out_p[TO260];
```

where the structure EE_DLOC is defined with following fields :

```
typedef struct des_loc {
        double  position[3];          /* EE position field */
        double  orientation[3][3];    /* EE orientation field */
        double  time;                 /* Time interval for completion */
} EE_DLOC;
```

Code for SEND $P_w[k+1]$

```
cmd_type = EE_LOC;
send_header(TO560);
chan_in_message(sizeof(EE_PLOC),&ee_p560,in_p[TO560]);
```

where the structure EE_PLOC is defined with following fields :

```
typedef struct present_loc {
        double  x,y,z;          /* Position of EE */
        double  o,a,t;          /* Orientation of EE in OAT angles */
} EE_PLOC;
```

Code for RECEIVE $P_{1A}[k]$

PARALLEL C. Some typical examples are shown above.

6.5 Experimental Results

The algorithm discussed in section 6.2 has been implemented on the IWC and successfully tested for synchronization of a PUMA 260 and a PUMA 560 manipulator arms. As mentioned earlier the PUMA 260, which is taken as the workpiece manipulator, moves along a cartesian path constructed on-line from a few look ahead points, using the technique discussed in Chapter 5. When the user triggers (by activating an appropriate switch) the tool manipulator, i.e. PUMA 560, moves so as to synchronize with the

workpiece. While the tool arm is attempting to synchronize, the speed of the workpiece arm can be changed at random by the user, through an appropriate dial, within a broad range. This range for the current set-up is

$$S_{W,min} = 30mm/sec \leq S_W \leq 80mm/sec = S_{W,max}$$

To avoid arm collision, non-interfering arm configuration have been used. Specifically, PUMA 260 traverses its path in elbow-up configuration while PUMA 560 maintains an elbow-down configuration. Due to the same reason, an additional offset is added along the Z direction for the interception phase, which is gradually removed in the beginning few cycles after synchronisation is achieved. The PUMA 260 base frame is taken as the common universal frame for both the robots. The transformation matrix between the PUMA 560 base frame and this universal frame is given by

$$R_{560}^{U} = \begin{bmatrix} 1 & 0 & 0 & 940 \\ 0 & 1 & 0 & 314.5 \\ 0 & 0 & 1 & -344.17 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The starting end effector position in universal frame for the respective robots, as determined using forward kinematics, is

$$\mathbf{P}_{260} = [350.0mm,\ 10.0mm,\ 0.0mm]^T$$

$$\mathbf{P}_{560} = [940.0mm,\ 746.3mm,\ 245.15mm]^T$$

giving an initial distance of ≈ 975mm between the tool and the workpiece. The orientation of PUMA 260 EE is kept vertically down while that of the PUMA 560 EE is maintained vertically up. For the various tool paths considered, 1.15m is an upper bound

on the starting distance between the end effectors of two manipulators, i.e. $\|P_{WT}[1,0]\|$, immaterial of the instant when the user triggers to synchronize. After synchronisation is achieved, an offset of 180mm is maintained along the Z axis to avoid interference between the grippers of the two manipulators.

With this experimental set up, it is found that the PUMA 560 arm is significantly extended when the synchronisation phase starts. This has the affect that significantly large changes in the direction of tool speed can be obtained with mild variations in the joint servo speeds allowing one to take a large value for φ_c. Thus for the current implementation, condition C1 (refer 6.2.2) is taken as the default. With a minimum workpiece speed of 30mm/sec, and a maximum starting dustance of 1.15m between the workpiece and the tool, the value of α is taken as 1.3 to ensure complete synchronisation of the two manipulators in less than 30 seconds.

6.5.1 Workpiece Path Along a Polar Arc

Figure 6.9 shows the workpiece and the tool path for the case when the knot points for workpiece manipulator lie along the polar function discussed earlier in section 5.4.3.2. For sake of convenience, the equations are repeated here

$$x_w[i] = R_p \sin(2\Sigma'_0\, \delta\xi)\cos(\Sigma'_0\, \delta\xi) + x_0$$

$$y_w[i] = R_p \sin(2\Sigma'_0\, \delta\xi)\sin(\Sigma'_0\, \delta\xi) + y_0$$

Z is taken to be

$$z_w[i] = P_z(\Sigma'_0\, \delta\xi) + z_0$$

where

$$\delta\xi = 2\pi/N$$

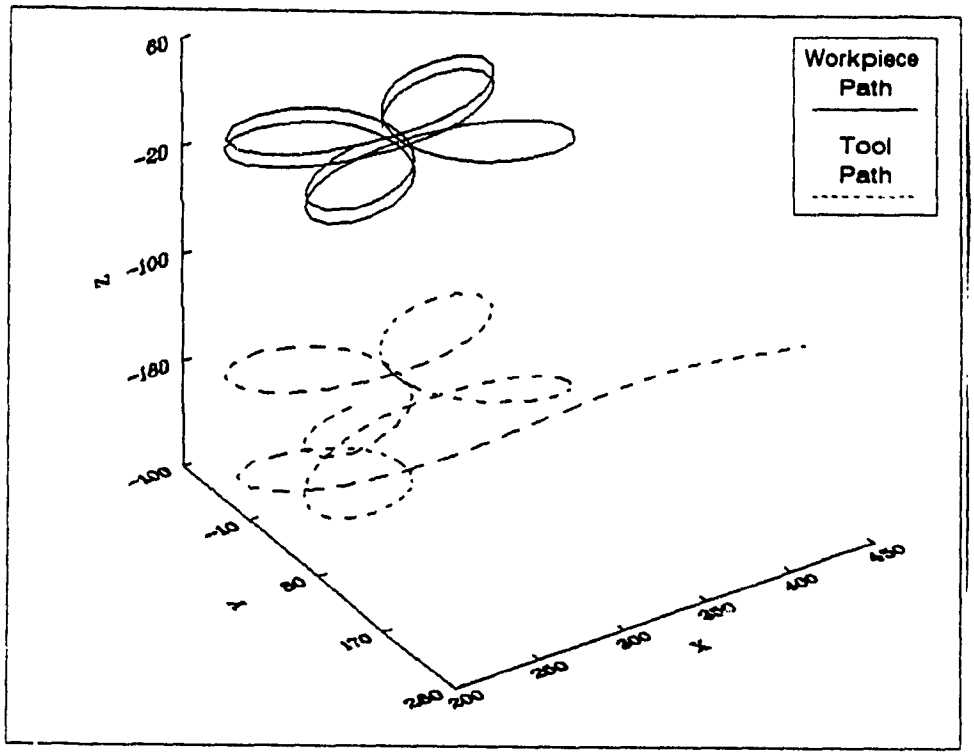

Figure 6.9 Synchronisation along a polar arc

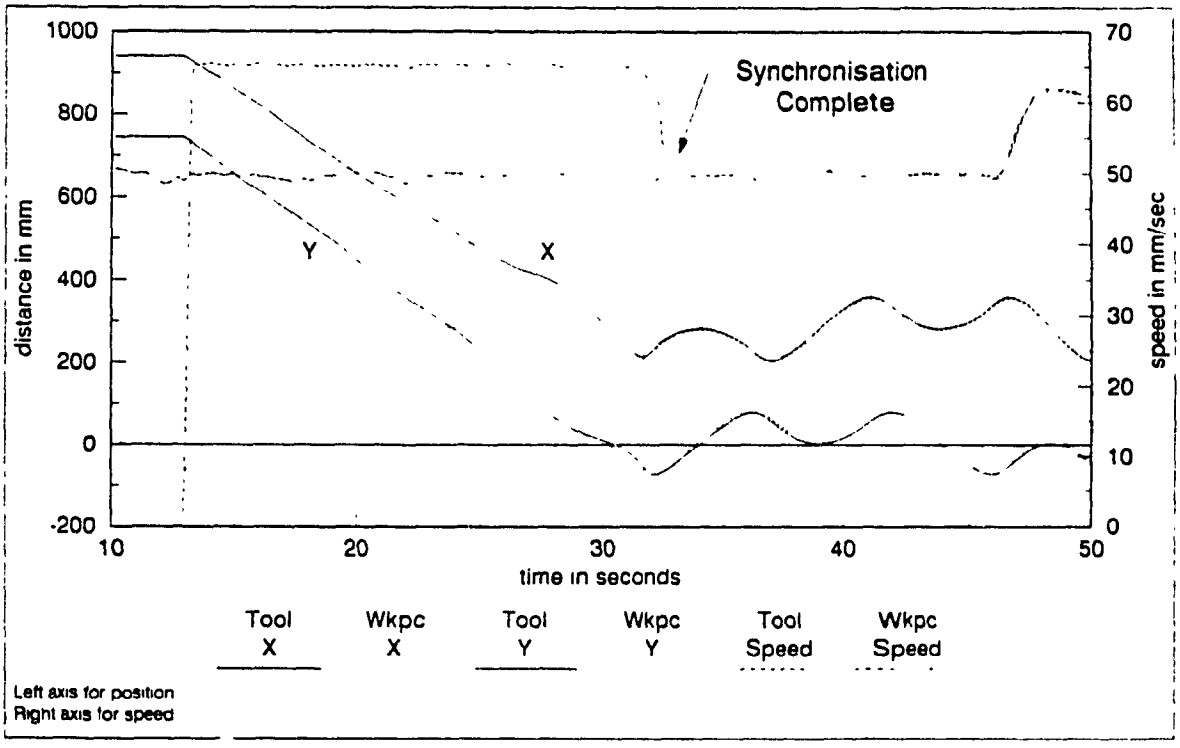

Figure 6.10 Speed profile for synchronisation along a polar arc

For Figure 6.9, R_p = 100mm, P_z = 1.6mm and N = 60. For sake of clarity, only the last part of the tool path, i.e. a little time before and after synchronisation is achieved, is shown. It is clear that inspite of frequent changes in workpiece direction, the straight line approach in interception phase leads to smooth tool path. Synchronisation is achieved without any abrupt changes in the direction of the tool velocity. After achieving synchronisation the additional offset of 70mm along the Z axis is covered by the tool manipulator in the subsequent few sampling periods Figure 6.10 shows the X-Y coordinates and the speed of the EE for both the manipulators. The external trigger for start of interception is given at t≈13 seconds and complete synchronisation is achieved in about 20 seconds, at t≈33 seconds when the tool speed drops from 65mm/sec (i.e. 1.3x50mm/sec) to the workpiece speed of 50mm/sec. The figure also demonstrates that the synchronisation is maintained when the workpiece speed is changed at random at t≈46 seconds. As expected, the x-y coordinates of the tool manipulator vary smoothly resulting in a smooth tool path.

6.5.2 Workpiece Path Along an Elliptical helix

Figure 6.11 shows the workpiece and the tool path for the case when the knot points for workpiece manipulator lie along an elliptical helix as discussed earlier in section 5.4.3.1. For sake of convenience, the equations are repeated here

$$x_w[i] = R_x\cos(\Sigma'_0\,\delta\xi) + x_0$$

$$y_w[i] = R_y\cos(\Sigma'_0\,\delta\xi) + y_0$$

$$z_w[i] = P_z(\Sigma'_0\,\delta\xi) + z_0$$

where

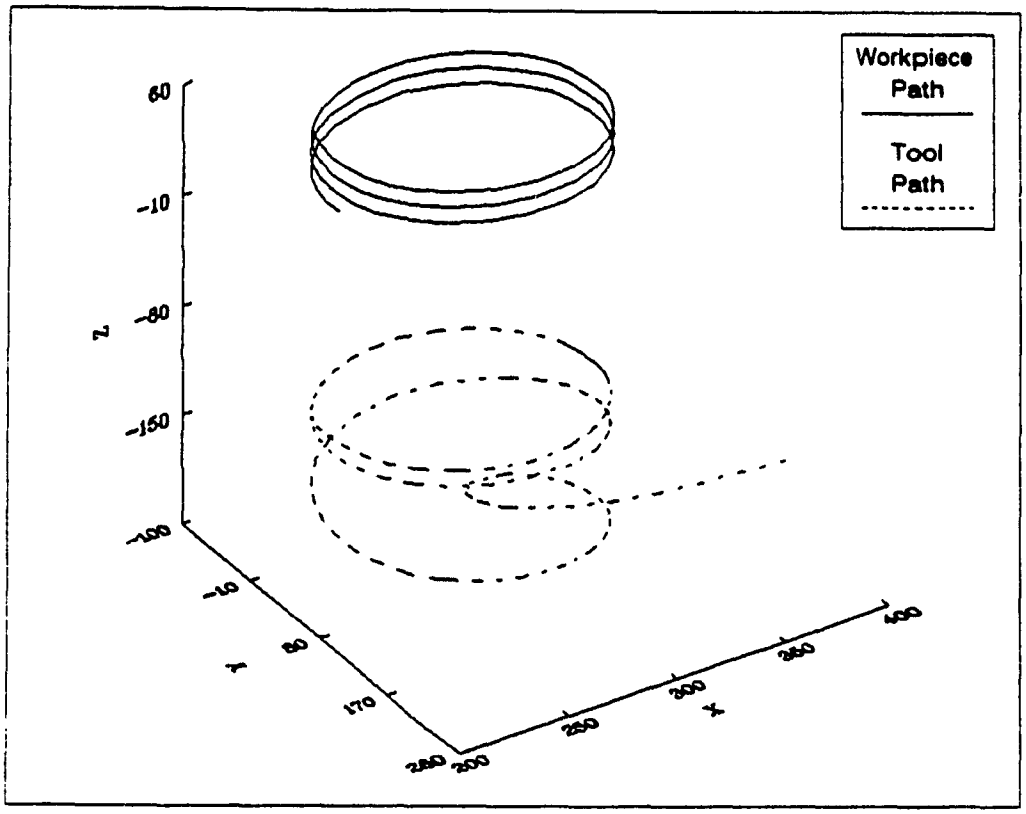

Figure 6.11 Synchronisation along an Elliptical Helix.

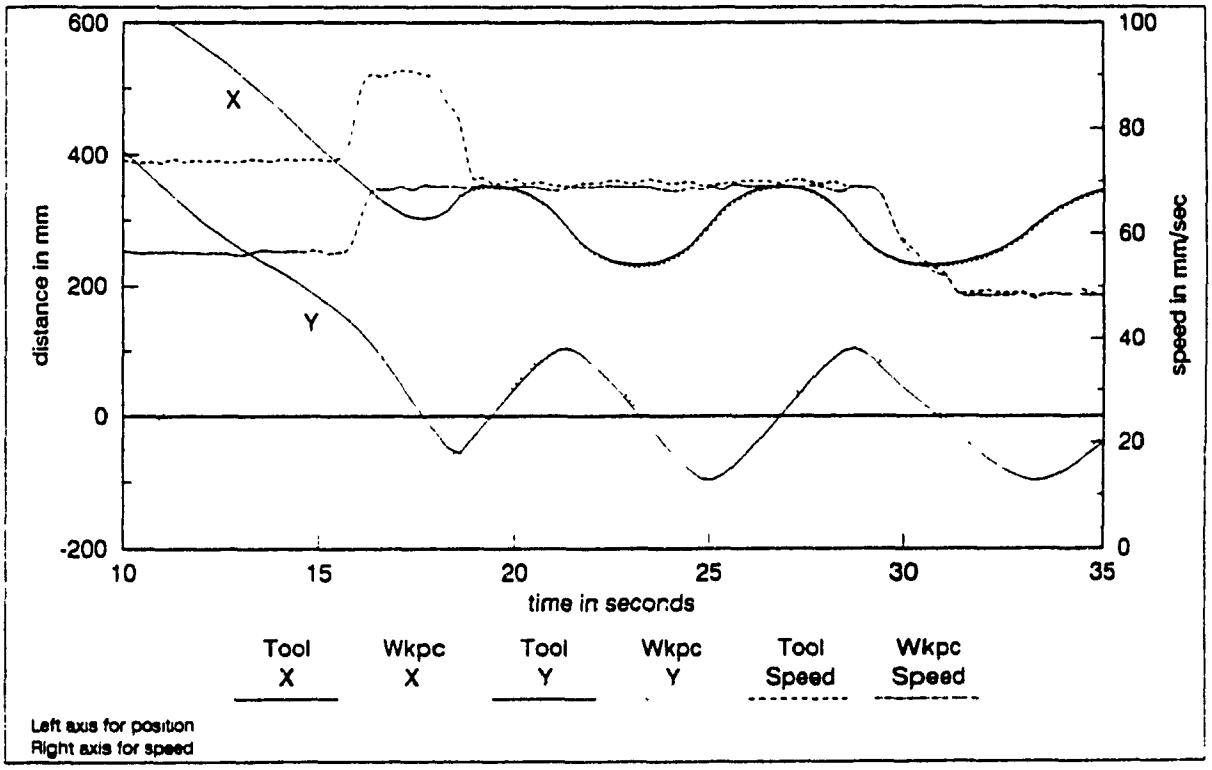

Figure 6.12 Speed profile for synchronisation along an Elliptical Helix.

$\delta\xi = 2\pi/N$

For Figure 6.11, N = 60, R_x = 60.0 mm, R_y = 100.0 mm and P_z = 1.6 mm giving a pitch of 10 mm along the Z axis. Again, for sake of clarity, only the last part of the tool path, i.e. a little time before and after synchronisation is achieved, is shown. Though the change in the tool direction is more pronounced in this case than the last one, it is still smooth enough and no jerk in the manipulator motion is observed. Figure 6.12 shows the speed profile and x-y coordinates for this case. It is seen that synchronisation is achieved inspite of sudden change in workpiece speed close to the anticipated termination of the interception phase at t=16 seconds. Further, the synchronisation is maintained with variation in workpiece speed at t=30 seconds.

In both Figure 6.10 and Figure 6.12, an error of ≈ 4mm is seen along the Y axis after the synchronisation is attained. It should be noted that this error is not due to the synchronisation algorithm. Though initially it was thought to be due to error in Y-component of the position vector of $\mathbf{R}^{I}_{560}$ matrix, attempts to correct it by marginally varying the Y-component failed. It is the authors belief that this error is the net result of error in rotation and position vector of $\mathbf{R}^{U}_{560}$ matrix and steady state error in the joint #1 servos of the PUMA 560 and PUMA 260 manipulators.

6.6 Summary

To summarise this chapter, implementation of an integrated controller on a network of parallel processors, for a typical workcell application requiring coordinated control of two robots has been described. The developed software consists of communicating tasks/threads mapped on virtual processing blocks. Depending on the

computational demands, a virtual processing block is then physically mapped onto a cluster of one or more transputers. This makes the implementation of IWC highly modular rendering itself to easy reconfiguration for future applications. A discrete time model for minimum-time synchronisation between two arms has been developed. The experimental results demonstrate the success of the IWC to achieve complete synchronisation between workpiece and tool manipulators in predicted time even with on-line path determination and speed modification of the workpiece arm.

Chapter 7

Conclusions and Recommendations for Future Work

This thesis has demonstrated that the use of an integrated workcell controller, for multiple device coordination in an industrial workcell, offers several advantages to the control engineer. The integration of all control functions at a central location makes the system extremely compact as compared to the distributed scheme resulting from the use of several independent robot controllers.

The actual implementation of the IWC, using transputer technology, for a typical workcell consisting of two manipulators **firmly establishes the practical feasability and industrial viability of the integrated controller concept.** The transputer based hierarchical controller provides a uniform hardware and software architecture through all levels. The upgradable computing power, resulting from its parallel architecture, allows one to add new kind of motion/force control devices to the workcell in a simple manner.

At the current stage of development, the IWC is capable of performing the following real-time operations :

(i) Kinematic control of a 6 DOF Puma type industrial robot.

(ii) On-line path planning in task space for continuous path robotic applications.

(iii) Two manipulator synchronisation so as to allow parallel operations by the robotic arms on a common workpiece.

Further, the architecture of the transputer controller satisifies all the design criterion outlined in section 2.4. Hence it can be concluded that **all the objectives**

established for this thesis (section 1.1) have been successfully met.

The **resolution of the servo shaft encoders** can be identified as the key factor effecting the accuracy of the robotic arm. The error in end effector position and orientation can be drastically reduced by replacing the 200-250 line shaft encoders by 1000 line encoders. Use of analog tachometers to implement derivative control in the servo loop (refer section 4.4.2) will substantially improve the manipulator speed regulation.

As explained in section 4.7, with better encoder resolution the controller sampling period T_c **can be reduced** to the order of 10 - 12 msec. This would lead to a further improvement in the positional accuracy. Further, the software implementation of the controller clock (thread INTTIM) allows one to **tune the T_c on-line**, depending on the arm speed, for best performance.

As mentioned in section 6.5, refined measurement of the spatial displacement between the two Puma arms would remove the offset errors observed after synchronisation. With the implementation of the EE orientation control (section 5.5) in the path planning task, the **workcell can be used to perform parallel operations using two robots along complex arcs or surfaces in three dimensional space.**

As a research and development tool, the integrated controller lays a foundation upon which new algorithms related to almost all aspect of robotic engineering can be implemented and tested. Increased computing power can be obtained by using the more recent T-9000 transputer [7.1], which offers six communication links leading to better interconnectivity between the processor clusters.

Beginning from the lowest level, the accuracy in manipulator control can be improved by including the effects of manipulator dynamics. The applications of the workcell can be extended to such operations as deburring, grinding etc. by implementing the computed torque control technique with force/torque feedback. To increase the workcell flexibility, work can be pursued to develop device drivers for different type of manipulators and other motion devices so as to build a *library* of such utilities over a period of time. The steps invloved for developing a device driver for a new kind of device have already been discussed in section 4.9.

At the path planning level, camera vision can be used to identify obstruction free paths in a workspace having fixed or moving obstacles. This may be achieved by using a multi-transputer node, with image processing capabilities, which would identify non-interfering arm configurations for participating manipulators to work along a common path in the shared workspace.

At the global level, maximum productivity can be achieved by mounting fixed manipulators on mobile robots so as to remove the workspace limitation altogether. Then a database of *synchronisation models* can be developed for coordination between particular devices. Thus, a PUMA manipulator, mounted on a mobile robot, may be used for pick and place operations on any conveyor belt on the shop floor while a SCARA arm may be used for synchronisation with a X-Y table for assembly operations at any desired location in plant.

Reference

Chapter 2

[2.1] Cheng, R.M.H., *"Research in multi-arm robotic workcell and implementation with Transputer technology"*, Invited Keynote Paper, Proc. CAI Symposium '90, Hong Kong Institute of Engineers, Hong Kong 1990.

[2.2] Davis, C.R., Crawley, L.R., *"A robotic workcell for linear welding/thermovision scanning of thermoplastic bumpers"*, Conf. Proc. of Robots 8, vol.1, June 1984, pg. 1-16 to 1-26.

[2.3] Bloom, J.H., *"Robotic diskette writer system"*, Conf. Proc. of Robots 8 , vol.1, June 1984, pg. 3-14 to 3-29.

[2.4] Graf, T., *"Practical methods for robotic deburring and finishing applications"*, Conf. Proc. of Robots 12 and Vision '88, vol.2, June 1988, pg. 16-21 to 16-38.

[2.5] Olsen, H.B., *"A flexible robotic workcell for the assembly of airframe components"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.2, 1990, pg. 1278-1284.

[2.6] Cheng, R.M.H., Poon, S.C.L., Montor, T., *"Adaptive synchronisation control of a manipulator working in an intelligent workcell"*, IEEE Trans. on Industrial Electronics, vol.37, No.2, April 1990, pg. 119-126.

[2.7] Hussaini, S.S., David, E.J., *"Multiple manipulators and robotic workcell coordination"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.2, 1986, pg. 1236-1241.

[2.8] Fisher, J.J., *"Design of a dual forearm modular robot"*, Conf. Proc. of Robots 12 and Vision '88, vol.1, June 1988, pg. 2-25 to 2-36.

[2.9] Pfister, G.W., *"Robot programming considerations for most computer control of a multirobot workcell"*, Conf. Proc. of Robots 12 and Vision '88, vol.1, June 1988, pg. 4-119 to 4-125.

[2.10] Norcross, R.J., *"A control structure for multi-tasking workstations"*, Proc. of the IEEE Conf. on Robotics and Automation, 1988, pg. 1133-1135.

[2.11] Agapakis, J.E., Katz, J.M., Pieper, D.L., *"Programming and control of multiple robotic devices in coordinated mation"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.1, 1990, pg. 362-367.

[2.12] Zheng, Y.F., Luh, J.Y.S., Jia, P.F., *"A real-time distributed computer system for coordinated-motion control of two industrial robots"*, Proc. of the IEEE Conf. on Robotics and Automation, 1987, pg. 1236-1241.

[2.13] Tsai, Chi-Keng, *"Multiple robot coordination and programming"*, Proc. of the IEEE Conf. on Robotics and Automation, vol. 2, 1991, pg. 978-985.

[2.14] Poon, S.C.L., *"The development and analysis of an multiprocessor-based intelligent robotic workcell"*, M.Eng. Thesis, Dept. of Mechanical Engineering, Concordia University, 1989, pg. 38-46, pg. 190-191, pg. 33-34, pg. 87-108.

[2.15] Leahy, M.B. Jr., Saridis, G.N., *"The RAL hierarchical control system"*, Proc. of the IEEE Conf. on Robotics and Automation, 1986, pg. 407-411.

[2.16] Olson, D.E., Olson, K.W., Chao, H.H., *"Systolic architectures for computation of the Jacobian for robot manipulators"*, Computer Architectures for Robotics and

Automation, J. Graham Ed., Gordon nad Breach, 1987, New York.

[2.17] Leung, S.S., Shanblatt, M.A., *"Real time DKS on a single chip"*, IEEE Journal of Robotics and Automation, vol. RA-3, no.4, Aug. 1987, pg. 281-290.

[2.18] Lanthrop, R.H., *"Parallelism in manipulator dynamics"*, Int. Journal of Robotics Research, vol.4, no.2, Summer 1985, pg. 80-102.

[2.19] Takanashi, N., Ikeda, T., Tagawa, N., *"A high sample rate robot control system using a DSP based numerical calculation engine"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.2, 1989, pg. 1168-1173.

[2.20] Graham, J.H., *"Special computer architectures for robotics : tutorial and survey"*, IEEE Trans. on Robotics and Automation, vol.5, No.5, October 1989, pg. 543-554.

[2.21] Chen, J.B., et. al., *"NYMPH : A multiprocessor for manipulation applications"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.3, 1986, pg. 1731-1736.

[2.22] Backes, P., Hayati, S., Hayward, V., Tso, K., *"The KALI multi-arm robot programming and control environment"*, Proc. of the NASA Conf. on Space Telerobotics, 1989, pg. 1-10.

[2.23] Nilakantan, A., Hayward, V., *"The Synchronisation of multiple manipulators in KALI"*, Robotics and Autonomous Systems 5, 1989, North-Holland, pg. 345-358.

[2.24] Lloyd, J., Parker, M., McClain, R., *"Extending the RCCL programming environment to multiple robots and processors"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.1, 1988, pg. 465-469.

[2.25] Guptill, R., Stahura, P., *"Multiple robotic devices : position specification and coordination"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.3, 1987,

pg. 1655-1659.

[2.26] Rubelo, L.C., Avula, X.J.R., *"Hierarchical neurocontroller architecture for intelligent robotic manipulation"*, Proc. of the IEEE Conf. on Robotics and Automation, vol.3, 1991, pg. 2656-2661.

[2.27] Guez, A., Bar-Kana, I., *"Two-degree-of-freedom robot neurocontroller"*, Proc. of the 29th Conf. on Decision and Control, 1990, pg. 3260-3264.

[2.28] Hirose. M., *"Development of a holonic manipulator and its control"*, Proc. of the 29th Conf. on Decision and Control, 1990, pg. 91-96.

[2.29] Cheng, R.M.H., Montor, T., *"Synchronisation control of an industrial robotic manipulator using camera vision"*, Proc. Int. Conf. on Intelligent Autonomous Systems, 1986, pg. 157-161.

[2.30] Cheng, R.M.H., Lequoc, S., Anthanasoulias, D., *"An on-line system for locating the position of the centroid of a flat object"*, Proc. of the 9th Canadian Congress of Applied Mechanics, Saskatoon, 1983.

[2.31] Cheng, R.M.H., Montor T., *"Application of a low cost vision system to automatic assembly"*, Proc. IFAC Low Cost Automation, 1986.

[2.32] Cheng, R.M.H., Poon, S.C.L., Rajagopalan, R., *"Interrupt driven transputer file server"*, Proc. of the 4th Conf. of the North American Transputer Users Group, Ithaca, 1990, pg. 119-130.

[2.33] Poon, S.C.L., Huard, G., *"Design of a general purpose interface between a serial device and 8 bit parallel device"*, CIC Report #0028, Centre for Industrial Control, Concordia Un' ersity, Montreal.

[2.34] Fox, G., *"Solving problems on concurrent processors : Vol 1"*, Prentice Hall, New Jersey, 1988, pg. 1-16, 22-24.

[2.35] Hashimoto, K., Ohashi, K., Kimura, H., *"An implementation of parallel algorithm for real-time model based control on a network of microprocessors"*, The Int. Journal of Robotics Research, vol.9, No.6, Dec.1990, pg. 37-47.

[2.36] Inmos Ltd., *"The Transputer data book"*, Second Edition, 1989, pg. 5-47, 189-261, 479-503, 503-529.

[2.37] Tanenbaum, Andrew S., *"Operating systems : design and implementation"*, Prentice Hall, New Jersey, 1987, pg. 45-88.

[2.38] Hoare, C.A.R., *"Communicating sequential processes"*, Communications of the ACM 21, 1978, pg. 666.

[2.39] Inmos Ltd., *"The Transputer applications notebook : systems and performance"*, First Edition, 1989, pg. 114-130, 92-113.

[2.40] Inmos Ltd., *"OCCAM 2 reference manual"*, Prentice Hall Int., 1988.

[2.41] 3L Ltd., *"Parallel C user guide version 2.2.2"*, 1991.

[2.42] 3L Ltd., *"TBUG user guide version 1.0.22"*, 1990.

[2.43] Asher, G.M., Summer, M., *"Parallelism and the transputer for real-time high performance of AC induction motors"*, IEE Proc., Vol.137, No.4, 1990, pg. 179-188.

[2.44] Thompson, H.A., Fleming, P.J., *"Fault-tolerant transputer based controller configurations for gas-turbine engines"*, IEE Proc., Vol.137, No.4, 1990, pg. 253-260.

[2.45] Whitcomb, L.L., Koditschek, D.E., *"Transputers at work : Real-Time distributed robot control"*, Proc. of the 4th Conf. of the North American Transputer Users Group, Ithaca, 1990, pg. 107-118.

[2.46] Sharkey, P.M., Daniel, R.W., Elosegui, P., *"Transputer based real time robot control"*, Proc. of the 20th Conf. on Decision and Control, 1990, pg. 1161-1162.

[2.47] Rajagopalan, R., *"Parallel computation of kinematics and dynamics of robot arms and mobile robots employing transputers"*, CIC Report #0023, Centre for Industrial Control, Concordia University.

[2.48] Xiao, J.W., Cheng, R.M.H., *"Efficient parallel computation of robot inverse dynamics"*, Proc. of the 2nd National Applied Mechanisms and Robotics Conf., Cincinnati, 1991.

Chapter 3

[3.1] Transtech reference: TMB08MAN0690, *"Transtech TMB08 PC Transputer Motherboard user guide"*, Transtech Parallel Systems Corporation, 1990.

[3.2] Inmos Technical Note 18, *"Connecting Inmos links"*, The Transputer Applications Notebook, Systems and Performance, Inmos Ltd., 1989, pg. 30-51.

[3.3] Inmos Technical Note 01, *"Extraordinary use of transputer links"*, The Transputer Applications Notebook, Systems and Performance, Inmos Ltd., 1989, pg.157-166.

[3.4] Analog Devices Datasheet, *"LC^{2}MOS high speed 8-channel 8-bit ADC 7828"*, Data Conversion Products Databook, Analog Devices, 1988, pg. 3-305 to 3-316.

Chapter 4

[4.1] Koren, Y., *"Robotics for engineers"*, McGraw-Hill Book Company, 1985, pg. 128-161, pg. 23-25.

[4.2] Unimation, *"Puma robot 200 series equipment manual P/N 398R1A"*, Unimation Inc., 1983.

[4.3] Unimation, *"Puma robot 200 series programming manual : User's guide to VAL P/N 398R2A"*, Version 260 V13G, Unimation Inc., 1983.

[4.4] Unimation, *"Puma robot 550/560 equipment and programming manual 398H"*, Unimation Inc., 1982.

[4.5] Unimation, *"Puma Mark II robot : 500 series equipment manual for VAL II and VAL PLUS operating systems 398U1"*, Unimation Inc., 1985.

[4.6] Nagy, P.V., *"The Puma 560 industrial robot: inside out"*, Proc. of the Robots 12 and Vision '88 Conf., 1988, pg.467-479.

[4.7] Nagy, P.V., Lim K.B., *"The electrical wiring and signal sensing in a Puma 560 industrial robot"*, Technical Report TR-ME-005-CON-86, Dept. of Mech. and Prod. Eng., National Univ. of Singapore, 1986.

[4.8] Goldenberg, A.A., Chan, L., *"An approach to real-time control of robots in task space. Application to control of PUMA 560 without VAL-II"*, IEEE Trans. on Industrial Electronics, vol.35, No.2, May 1988, pg. 231-238.

[4.9] Goldenberg, A.A., Melidy, A., *"A method for Puma-560 operation without VAL"*, Proc. of Robots 9, 1985, pg. 18.61-18.79.

[4.10] Kazanzides, P., Wasti, H., Wolovich, W.A., *"A multiprocessor system for real-time*

robotic control : design and implementation", IEEE Conf. on Robotics and Automation, 1987, pg. 1903-1908.

[4.11] Valvanis, K.P., Leahy, M.B., Jr., Saridis, G.N., *"On the real-time control of VAX 11/750 computer controlled Puma-600 robot arm"*, RPI, RAL Tech. Rep. 42, 1985.

[4.12] Fu, K.S., Gonzalez, R.C., Lee, C.S.G., *"Robotics : Control, Sensing, Vision and Intelligence"*, McGraw-Hill International Editions, 1987, pg. 12-81, 62-63, 156-159.

[4.13] Elgazzar, S., *"Efficient kinematic transformations for the Puma 560 robot"*, IEEE Journal of Robotics and Automation, vol. RA-1, No.3, Sept. 1985, pg. 142-151.

[4.14] Lloyd, J., Hayward, V., *"Kinematics of common industrial robots"*, Robotics 4, 1988, 169-191.

[4.15] Bazerghi, A., Goldenberg, A.A., Apkarian, J., *"An exact kinematic model for Puma 600 manipulator"*, IEEE Trans. on Systems, Man and Cybernatics, vol. SMC-14, No.3, May/June 1984, pg. 483-487.

[4.16] Orin, D.E., Schrader, W.W., *"Efficient computation of the jacobian for robot manipulators"*, The Int. Journal of Robotics Research, vol.3, No.4, Winter 1984, pg. 66-75.

[4.17] Lai, C.Z.Jim, Yang, D., *"An efficient motion control algorithm for robots with wrist singularities"*, IEEE Trans. on Robotics and Automation, vol.6, No.1, Feb. 1990, pg. 113-117.

[4.18] Zhang, H., Paul, R.P., *"A parallel inverse kinematic solution for robot*

manipulators based on multiprocessing and linear extrapolation", IEEE Trans. on Robotics and Automation, vol.7, No.5, Oct.1991, pg. 660-669.

[4.19] Zomaya, A. Y., et. al., *"Fast robot kinematic modelling via transputer networks"*, Parallel Processing and Artificial Intelligence, Ed. Mike Reeve, John Wiley & Sons, 1989, pg. 193-213.

[4.20] Rajagopalan, R., *"A computer-aided methodology for the calibration of industrial robots employing optical metrology"*, M.Eng. Thesis, Dept. of Mechanical Engineering, Concordia University, 1986, pg. 105-108.

[4.21] Lloyd, J., *"Implementation of a robot control development environment"*, M.Eng. Thesis, Dept. of Electrical Engineering, McGill University, 1985.

[4.22] National Semiconductor Datasheet, *"LM628/LM629 precision motion controller"*, National Semiconductor Corporation, March 1989.

[4.23] David Dale, *"LM628/LM629 user guide"*, Application Note 706, National Semiconductor Corporation, August 1990, pg. 19-20.

[4.24] Steven Hunt, *"LM628 programming guide"*, Application Note 693, National Semiconductor Corporation, April 1990.

Chapter 5

[5.1] Brady M., et.al., *"Robot motion : planning and control"*, Cambridge : MIT Press, 1982, pg. 27, pg. 222-223.

[5.2] Lin C.S., Chang P.R., Luh J.Y.S., *"Formulation and optimization of cubic polynomial joint trajectories for industrial robots"*, IEEE Trans. on Automatic Control, Vol.AC-28, No.12, Dec. 1983, pg. 1066-1073.

[5.3] J.Y.S. Luh, Lin C.S., *"Approximate joint trajectories for control of industrial robots along cartesian paths"*, IEEE Trans. Syst., Man, Cybern., vol.SMC-14, No.3, May/June 1984, pg. 444-450

[5.4] Lin C.S., Chang P.R., *"Joint trajectories of mechanical manipulators for cartesian path approximation"*, IEEE Trans. Syst., Man, Cybern., vol.SMC-13, No.6, Nov./Dec. 1983, pg. 1094-1102.

[5.5] Thompson S.E., Patel R.V., *"Formulation of joint trajectories for industrial robots using B-Splines"*, IEEE Trans. on Ind. Electronics, vol.IE-34, No.2, May 1987, pg. 192-199.

[5.6] Anderson R.L., *"Aggressive trajectory generator for a robot ping-pomg player"*, IEEE Conf. on Robotics & Automation, 1988, pg. 188-193.

[5.7] Mujtaba M.S., *"Discussion of trajectory calculation method"*, Artificial Intelligence Lab., Stanford University, CA.

[5.8] Chand S., Doty K.L., *"On-Line polynomial trajectories for robot manipulators"*, The Int. Journal of Robotics Research, vol.4, No.2, Summer 1985, pg. 38-48.

[5.9] Castain R.H., Paul R.P., *"Polynomial robotic trajectories : A new approach"*, TR-EE 82-37, Purdue University, Dec. 1982.

[5.10] Shahinpoor M., Abdel-Rahman S.Z., *"Generalised algorithm for computing the 4-3-4 N-axis robotic trajectory"*, Journal of Robotic Systems, Vol.1, No. 4, 1984, pg.395-420.

[5.11] Luh J.Y.S., Lin C.S., *"Optimum path planning for mechanical manipulators"*, Trans. of the ASME, vol.102, 1981, pg. 142-151.

[5.12] Sahar G., Hollerbach J.M., *"Planning of minimum time trajectories for robot arms"*, The International Journal of Robotics Research, vol.5, No.3, Fall 1986, pg. 90-100.

[5.13] Pfeiffer F., Johanni R., *"A concept for manipulator trajectory planning"*, IEEE Journal of Robotics and Automation, vol.RA-3, No.2, April 1987, pg.115-123.

[5.14] Seegar H.G., Paul R.P., *"Optimising robot motion along a pre-defined path"*, IEEE Int. Conf. on Robotics and Automation, 1985, pg. 765-770

[5.15] Shiller Z., Dubowsky S., *"On the optimal control of robotic manipulators with actuator and end-effector constraints"*, IEEE Int. Conf. on Robotics and Automation, 1985, pg. 614-620.

[5.16] Kondo K., *"Motion Planning with 6 DOF by multistage bidirectional heuristic free-space enumeration"*, IEEE Conf. on Control and Automation, vol.7, No.3, 1991, pg. 267-277.

[5.17] Jun S., Shin K.G., *"Shortest path planning in discretised workspace using dominance relation"*, IEEE Trans. on Control and Automation, vol.7,No.3, 1991, pg. 342-350.

[5.18] Shin K.G., Mckay N.D., *"Minimum-time control of robotic manipulators with geometric path constraints"*, IEEE Trans. on Automatic Control, vol.AC-30, No.6, June 1985, pg. 531-541.

[5.19] Dubowsky S., Norris M.A., Shiller Z., *"Time optimal trajectory planning for robotic manipulators with obstacle avoidance : A CAD approach"*, IEEE Conf. on Robotics and Automation, 1986, pg. 1906-1912.

[5.20] Lorenz-Perez T., *"Automatic Planning for manipulator transfer movements"*, IEEE Trans. Syst., Man, Cybern., vol.SMC-11, 1981, pg. 681-698.

[5.21] Paul R.P., *"Manipulator cartesian path control"*, IEEE Trans. Syst., Man, Cybern., vol.SMC-9, No.11, Nov. 1979, pg. 702-711.

[5.22] Taylor R.H., *"Planning and execution of straight line manipulator trajectories"*, IBM Journal of Research and Development, vol.23, July 1979, pg. 424-436.

[5.23] Whitney D.E., *"Resolved motion rate control of manipulators and human postheses"*, IEEE Trans. on Man-Machine Systems, vol.10, No.2, June 1969, pg.47-53.

[5.24] Chang Y.H., Lee T.T., Liu C.H., *"On-line cartesian path trajectory for robot manipulators"*, IEEE Conf. on Robotics and Automation,vol.1,1988,pg. 62-67.

[5.25] Goldenberg A.A., Lawrence D.L., *"End Effector path generation"*, Sensors and Controls for Automated Manufacturing and Robotics, 1984, pg.271-280.

[5.26] Bruhm H., Ersu E., *"Cartesian path planning by general polynomial interpolation"*, Second IASTED Int. Symp. Robotics and Automation, 1983, pg. 156-159.

[5.27] Choi Y.K., Bien Z., Youn M.J., *"A path planning algorithm for positioning manipulator end effector in cartesian space using circular interpolation"*, 15th Int. Symp. on Indutrial Robots, 1985, 1031-1040.

[5.28] Lloyd J., Hayward V., *"Real-time trajectory generation using blend functions"*, IEEE Conf. on Robotics and Automation, 1991, pg. 784-789.

[5.29] Orlandea N., Berenyi T., *"Dynamic continuous path synthesis of indutrial robots*

using ADAMS computer program", Trans. of the ASME, vol.103, July 1981, pg. 602-607.

[5.30] Angeles J., Akhras R., *"Cartesian trajectory planning for 3-DOF spherical wrists"*, IEEE Conf. on Robotics and Automation, vol.1, 1988, pg. 68-74.

[5.31] Angeles J., Rojas A., Lopez-Cajun C.S., *"Trajectory planning in robotics continuous-path applications"*, IEEE Journal of Robotics and Automation, vol.4, No.4, August 1988, pg. 380-385.

[5.32] Angeles J., Alvarado M.A., Monsouri A.R., *"A spline-based method of solution of nonlinear two-point boundary-value problems"*, Comp. Maths. with Appls., vol.12A, No.9, 1986, pg.969-985.

[5.33] Gutsche R., Stahs T., Wahl F.M., *"Path generation with a universal 3d sensor"*, IEEE Conf. on Robotics and Automation, 1991, pg. 838-843.

[5.34] Carl de Boor, *"A Practical guide to spline"*, Applied Mathematical Sciences Vol.27, Springer-Verlag, pg. 49-52.

[5.35] ibid., pg. 63-69.

[5.36] ibid., pg. 53-56.

[5.37] Press W.H., Flannery B.P., Teukolsky S.A., Vetterling W.T., *"Numerical recipes in C"*, Cambridge University Press, 1990, pg. 47-48, pg. 94-98, 156-157.

[5.38] Kreyszig E., *"Advanced engineering mathematics"*, John Wiley & Sons, Sixth Edition, 1988, pg. 469-470, 477-480.

[5.39] Fine H.B., *"College algebra"*, Dover Publications, 1961, pg. 262-264.

Chapter 6

[6.1] Lorenz-Perez T., Wesley M., *"An algorithm for planning collision free paths among polyhedra obstacles"*, Communications of ACM, vol. 22, No.10, 1979, pg. 560-570.

[6.2] Freund E., Hoyer H., *"Pathfinding in multi-robot systems : solutions and applications"*, IEEE Conf. on Robotics & Automation, 1986, pg. 103-111.

[6.3] Fortune S., Wilfong G., Yap C., *"Coordinated mation of two robot arms"*, IEEE Conf. on Robotics & Automation, 1986, pg. 1216-1223.

[6.4] Freund E., Hoyer H., *"Real-time pathfinding in multirobot systems including obstacle avoidance"*, The Int. Journal of Robotics Research, vol.7, No.1, Feb. 1988, pg. 42-70.

[6.5] Freund E., Hoyer H., *"Collision avoidance for industrial robots with arbitrary motion"*, Int. Journal of Robotic Systems, vol.I, No.4, 1984, pg. 317-329.

[6.6] Chien Y.P., Koivo A.J., Lee B.H., *"On-line generation of collision free trajectories for multiple robots"*, IEEE Conf. on Robotics & Automation, 1988, pg. 209-214.

[6.7] Alford, A., Belyeu, M., *"Coordinated control of two robot arms"*, IEEE Conf. on Robotics, March 13-15, 1984, pg. 468-473

[6.8] Mason, M.T., *"Compliance and force control for computer controlled manipulators"*, IEEE Trans. on Systems, Man and Cybernetics, vol. SMC-11, No.6, June 1981, 418-432.

[6.9] Luh, J.Y.S., Zheng, Y.F., *"Constrained relations between two coordinated*

industrial robots for motion control", Int. Journal of Robotic Systems, vol.6, No.3, Fall 1987, pg. 60-70.

[6.10] Lim, J., Chyung, D.H., *"On a control scheme for two cooperating arms"*, Proc. of the 24th Conf. on Decision and Control, 1985, pg. 334-337.

[6.11] Tao, J.M., Luh, J.Y.S., Zheng, Y.F., *"Compliant coordination control of two moving industrial robots"*, IEEE Trans. on Robotics and Automation, vol.6, No.3, June 1990, pg. 322-330.

[6.12] Bobrow, J.E., McCarthy, J.M., Chu, V.K., *"Minimum-Time trajectories for two robots holding the same workpiece"*, Proc. of the 29th Conf. on Decision and Control, 1990, pg. 3102-3107.

[6.13] Yun, Xiaoping, *"Coordination of two-arm pushing"*, IEEE Conf. on Robotics & Automation, 1991, pg. 182-187.

[6.14] Cheng R.M.H, Poon S.C.L., Montor T., *"Adaptive synchronization control of a manipulator operating in an intelligent workcell"*, IEEE Trans. on Industrial Electronics, vol. 37, No. 2, April 1990, pg. 119-126.

[6.15] Cheng R.M.H, Poon S.C.L., *"Architecture of an intelligent robotic workcell for synchronization control"*, Proc. IASTED Conf. on Robotics Automation, 1988, pg. 155-160.

[6.16] Cheng R.M.H., Montor T., *"Synchronization control of indutrial robotic manipulator using camera vision"*, Proc. Int. Conf. Intelligent Autonomous Syst., 1986, pg. 157-161.

[6.17] Poon S.C.L, *"The development and analysis of an multi-processor based intelligent*

robotic workcell", M.Eng. Thesis, Dept. Mech. Eng., Concordia University, Montreal, Canada, 1989.

[6.18] Hemami, A., Ranjbaran, F., Cheng, R.M.H., *"A case study of two-robot arm workcell material handling"*, Journal of Robotic Systems, 8(1), 1991, pg. 21-37.

[6.19] Park, T.H., Lee, B.H., *"An approach to robot motion analysis and planning for conveyor tracking"*, IEEE Conf. on Robotics & Automation, 1991, pg. 70-75.

[6.20] Ahmad, S., Luo, S., *"Coordinated motion control of multiple robotic devices for welding and redundancy coordination through constrained optimisation in cartesian space"*, IEEE Conf. on Robotics & Automation, 1988, pg. 963-968.

[6.21] Jouaneh, M.K., Wang, Z., Dornfeld, D.A., *"Trajectory planning for coordinated motion of a robot and a positioning table: Part I-Path Specification"*, IEEE Trans. on Robotics and Automation, vol.6, No.6, Dec. 1990, pg. 735-744.

Chapter 7

[7.1] Inmos Ltd., *"The T9000 transputer products overview manual"*, First edition, 1991, pg. 7-15

Appendix A

Repertoire of Device Independent Commands

(i) Commands for a 6 DOF Robot Control

Command	Purpose
MPOS_6JT	- Reach the desired EE location using absolute interpolator in position mode.
MVEL_6JT	- Reach the desired location using absolute interpolator in velocity mode.
EE_LOC	- Return the End Effector position and orientation in world coordinates.
INVK6JT	- Return the Inverse Kinematics solution for the target position.

(ii) Commands for 3 DOF robot control (for testing new algorithms)

MPOS_3JT	- Move only the major joints in position mode.
MVEL_3JT	- Move only the major joints in velocity mode.
WRIST_POS	- Return the wrist position in world coordinates.

(iii) General Commands

GET_ANGLE	- Return the joint angle(s) for the indicated joints.
START	- Start motion on the indicated servo(s).
ABRUPT_STOP	- Stop one or more joint servos with maximum deceleration.
SMOOTH_STOP	- Stop one or more joint servos at the programmed

deceleration.

MOTOR_OFF - Deactivate one or more joint servos to make the joint free.

TRAJ_COMP - Wait till the trajectory loaded in the indicated joint(s) completes.

LFIL - Load PID filter gains for the indicated channel(s).

CALIBRATE - Calibrate the manipulator.

POTCAL - Calibrate the joint potentiometers (if any).

TUNE_PID - Measure servo performance for pid tuning.

REST - Take the arm to the HOME position.

HAND - Open/close the end effector.

NEST - Arm in NEST position, initialise joint angles.

QUIT - Quit the JSC program

(iv) Commands for accessing downstream peripherals

CMD_LMIO - Command is for LM629 motion controller.

GET_AD - Read a specified A/D channel.

ON40V - Enable high power supply (40V typically) to the arm.

RD40V - Is the high power supply to the arm ON?

(v) Commands for Debugging Support

RD_COUNT - Read the actual/desired position and velocity from the servo controller of the indicated channels (in terms of encoder counts).

INIDPOS - Initialise the desired position counters *dpos_cnt* maintained

in software and used in DRY simulation run.

GET_DPOS - Read the target angle as reflected in the software maintained *dpos_cnt* counters.

Appendix B

Device Addresses

Symbol	Address	Device
SERVO1	0x01	Servo Controller Joint #1 (Base).
SERVO2	0x02	Servo Controller Joint #2.
SERVO3	0x03	Servo Controller Joint #3.
SERVO4	0x04	Servo Controller Joint #4.
SERVO5	0x05	Servo Controller Joint #5.
SERVO6	0x06	Servo Controller Joint #6.
ALL_SERVO	0x0f	Servo Controllers Joint #1 to #6.
AD1_ADD	0x0c	A/D Converter #1 (Eight Channels).
AD2_ADD	0x0d	A/D Converter #2 (Eight Channels).
HND_PORT	0x07	Gripper Controller.
PORT_40V	0x0a	40 Volt status port.
PORT_SYNCH	0x09	Port for synchronisation trigger.

Appendix C

PUMA 260 Robot Arm Related Data

Table C.1 Puma 260 link parameters

Joint	α_i	a_i	d_i	Joint Range	NEST ANGLE
1	-90	0	0	$-180 \le \theta_1 \le 124$ $176 \le \theta_1 \le 180$	179.83
2	0	203.2	125.7	$-180 \le \theta_2 \le 66$ $113 \le \theta_2 \le 180$	157.31
3	90	0	0	$-180 \le \theta_3 \le -124$ $-56 \le \theta_3 \le 180$	-14.64
4	-90	0	203.2	$-223 \le \theta_4 \le 355$	90.11
5	90	0	0	$-122 \le \theta_5 \le 122$	89.74
6	0	0	55.88	$-222 \le \theta_6 \le 312$	50.83

Note : All dimensions in millimeters and angles in degrees

Note : Minimum average velocity is obtained by moving 1 count/65536 T_s.

Table C.2 Puma 260 Servo Resolution

Joint	Encoder Lines	Gear Ratio	Joint Angle Resolution (degree)	Instantaneous Velocity Reolution (degree/sec)	Minimum Average Velocity (degree/sec)
1	250	46.7200	7.706e-3	30.10	4.593e-4
2	250	69.9733	5.145e-3	20.10	3.067e-4
3	250	42.9867	8.376e-3	32.72	4.990e-4
4	200	43.5111	10.341e-3	40.39	6.164e-4
5	200	39.3846	11.424e-3	44.62	6.800e-4
6	200	31.7692	14.163e-3	55.32	8.442e-4

The joint angle to encoder count conversion is given by

$$\mathbf{e} = \mathbf{R}\theta$$

where θ is the joint angle vector in radians. **R** for Puma 260 is given by

$$R = \begin{bmatrix} 7435.72 & 0 & 0 & 0 & 0 & 0 \\ 0 & 11136.60 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6841.55 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5540.0 & 0 & 0 \\ 0 & 0 & 0 & 24.24 & 5014.60 & 0 \\ 0 & 0 & 0 & 0 & 45.76 & 4044.98 \end{bmatrix} \qquad \text{(C.1)}$$

The encoder count to joint angle conversion is given by

$$\theta = \mathbf{R}^{-1}\mathbf{e}$$

where $\mathbf{R}^{-1}$ is as follows

$$R^{-1} = \begin{bmatrix} 0.1345 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0898 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1462 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1805 & 0 & 0 \\ 0 & 0 & 0 & 0.0009 & 0.1994 & 0 \\ 0 & 0 & 0 & 0 & 0.0023 & 0.2472 \end{bmatrix} x10^{-3} \qquad \text{(C.2)}$$

Appendix D

PUMA 560 Robot Arm Related Data

Table D.1 Puma 560 link paramemters.

Joint	α_i	a_i	d_i	Joint Range	Angle at READY
1	-90	0	0	$-160 \le \theta_1 \le 160$	0
2	0	431.80	149.09	$-180 \le \theta_2 \le 43$ $137 \le \theta_2 \le 180$	-90
3	90	-20.32	0	$-180 \le \theta_3 \le -128$ $-52 \le \theta_3 \le 180$	90
4	-90	0	433.07	$-110 \le \theta_4 \le 170$	0
5	90	0	0	$-100 \le \theta_5 \le 100$	0
6	0	0	56.25	$-266 \le \theta_5 \le 266$	0

Note : All dimensions in millimeter and all angles in degrees.

Note : Minimum average velocity is taken as 1count/65536T_s.

Table D.2 Puma 560 Servo Resolution

Joint	Encoder Lines	Gear Ratio	Angle Resolution (degree)	Instantaneous Velocity Resolution (degree/sec)	Minimum Average Velocity (degree/sec)
1	250	62.6111	5.752e-3	22.47	3.428e-4
2	200	107.815	4.171e-3	16.29	2.486e-4
3	250	53.7063	6.703e-3	26.18	3.995e-4
4	250	76.0364	4.732e-3	18.48	2.820e-4
5	250	71.9231	5.007e-3	19.56	2.984e-4
6	250	76.6864	4.692e-3	18.33	2.796e-4

The joint angle to encoder count conversion is given by

$$\mathbf{e} = \mathbf{R}\theta$$

where θ is the joint angle vector in radians. **R** for Puma 260 is given by

$$R = \begin{bmatrix} 9964.87 & 0 & 0 & 0 & 0 & 0 \\ 0 & 13727.43 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8547.62 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12101.57 & 0 & 0 \\ 0 & 0 & 0 & 159.15 & 11446.92 & 0 \\ 0 & 0 & 0 & 159.26 & 2203.69 & 12205.02 \end{bmatrix} \quad \text{(D.1)}$$

The encoder count to joint angle conversion is given by

$$\theta = \mathbf{R}^{-1}\mathbf{e}$$

where $\mathbf{R}^{-1}$ is as follows

$$R^{-1} = \begin{bmatrix} 0.1004 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0728 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1170 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0826 & 0 & 0 \\ 0 & 0 & 0 & -0.0011 & 0.0874 & 0 \\ 0 & 0 & 0 & -0.0009 & -0.0158 & 0.0819 \end{bmatrix} x10^{-3} \quad \text{(D.2)}$$

Appendix E

PID Tuning and Calibration of Puma 560

(i) *PID Tuning*

The typical plots for steady state error as a function of k_p-k_i are shown in Figure E.1 and E.2 on the following page. The optimum PID gains identified, using procedure discussed in section 4.4.2 are as follows :

Table E.1 PID Gains for Puma 560

Puma Joint	LM629 k_p	Actual K_p	LM629 k_i	Actual K_i	LM629 k_d	Actual K_d
1	4200	16.40	10	0.60	100	8.0e-4
2	4100	16.01	170	10.13	100	8.0e-4
3	3700	14.45	175	10.43	120	9.6e-4
4	2800	10.94	10	0.60	250	2.0e-3
5	2700	10.55	70	4.17	200	1.6e-3
6	2800	10.94	10	0.60	360	2.9e-3

(ii) *PUMA 560 Calibration*

The Puma 560 robot arm is calibrated using the potentiometers mounted on the motor shaft. However, before this can be done, data has to be generated for potentiometer calibration. A utility called *POTCAL* has been developed for this purpose. Starting from READY position, which can be reached by limping the arm and moving the joints to get the correct potentiometer voltage (Table E.2), POTCAL moves each joint through its full range and records the potentiometer voltage on each index pulse observed. This data is

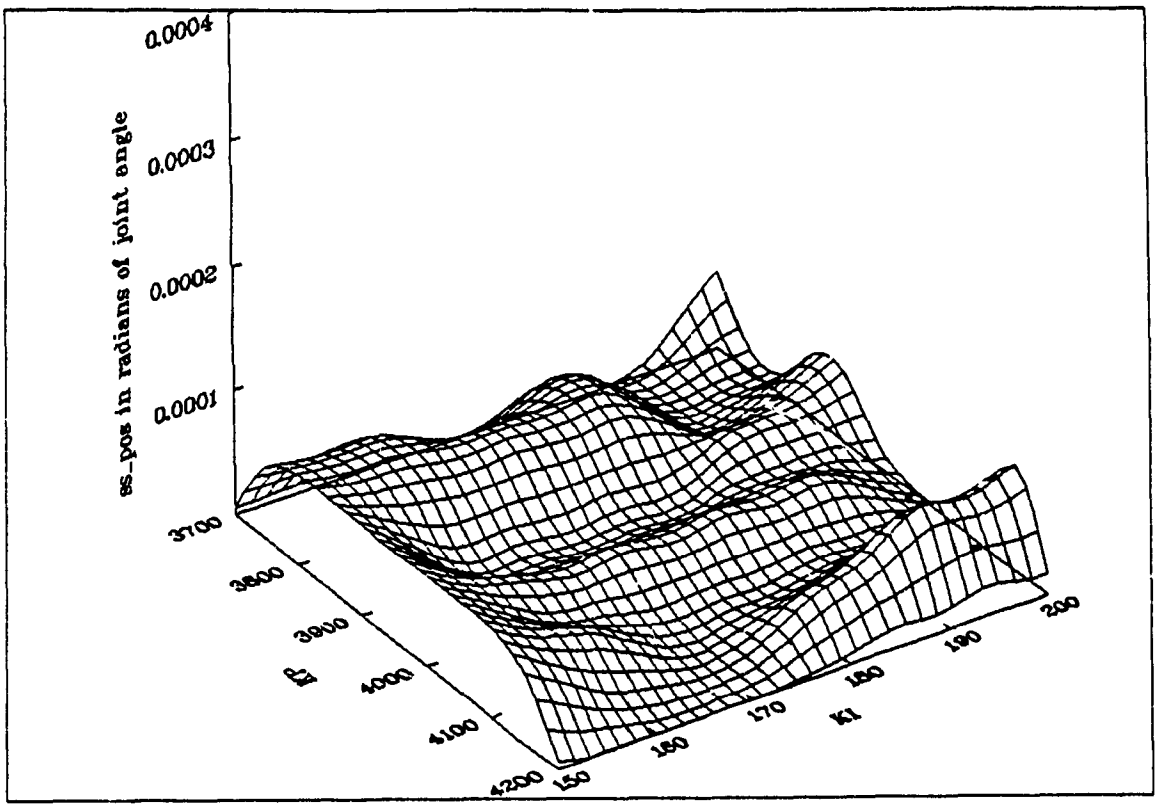

Figure E.1 Steady State Error in Joint #2 vs k_p - k_i.

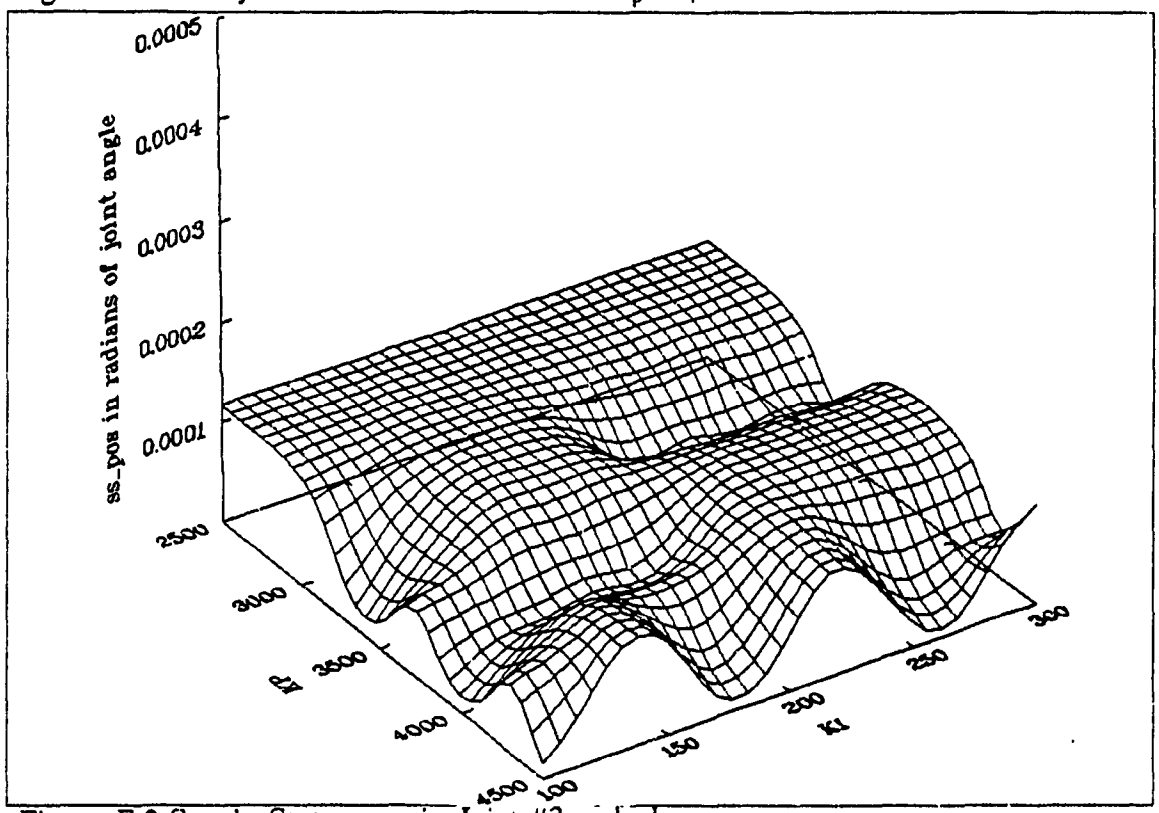

Figure E.2 Steady State error in Joint #3 vs k_p-k_i.

Table E.2 Safe operation pot voltages for Puma 560

Puma 560 Joint	Voltage at READY	Voltage Limit Minimum	Voltage Limit Maximum
1	2.655	1.40	3.90
2	2.659	0.88	4.40
3	2.673	1.70	3.65
4	2.655	1.60	4.30
5	2.704	1.80	3.60
6	2.674	0.20	4.40

then stored in a file called *poti.val*, where i stands for the joint number. These files are used as look up tables for subsequent manipulator caliberation.

Figure E.3 and E.4 show the typical potentiometer caliberation data obtained using POTCAL for joint #2 and joint #3 respectively. The -180/+180 degree transition has been discussed earlier in section 4.5.

(iii) *Safety Limits*

Using the potentiometer voltages, a *safety shutdown* circuit has been implemented for Puma 560 arm. Using voltage comparators and a PAL device, the high power to the arm is automatically shut down if the potentiometer voltage of any joint goes out of the range specified in the following table. The design is on the conservative side in a way that the actual limit for each joint are slightly less than those mentioned in Table D.1.

These limits can be always be changed by the user changing a few resistors.

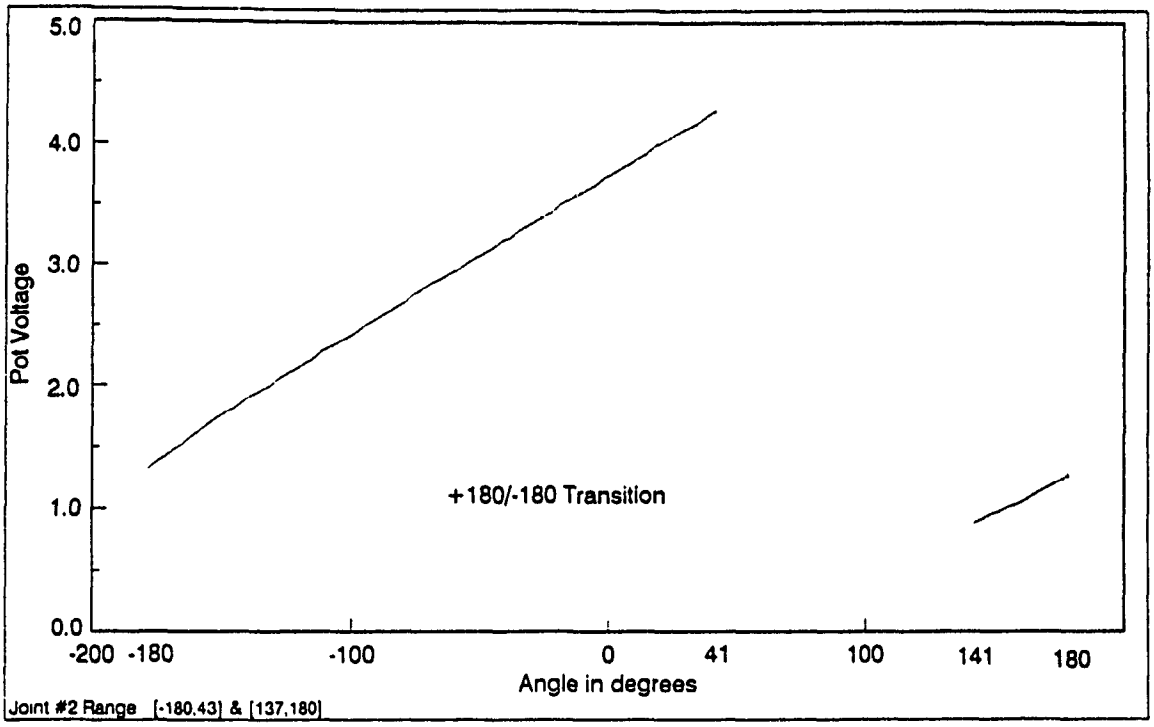

Figure E.3 Potentiometer Calibration : Joint #2

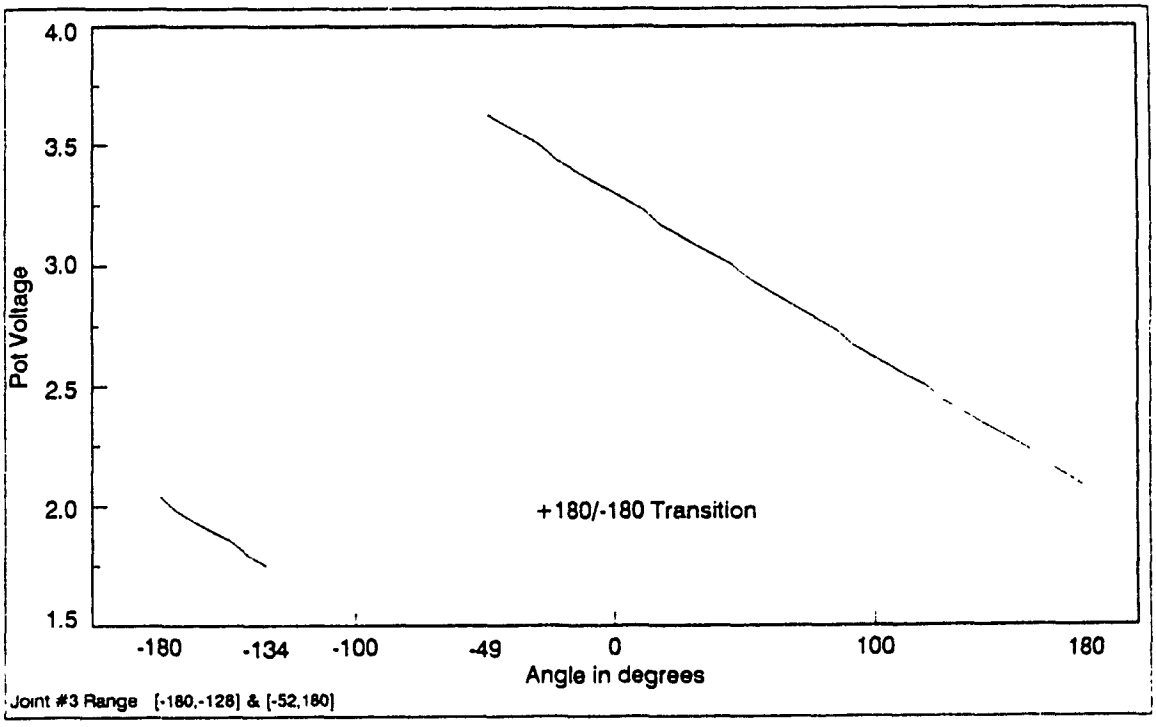

Figure E.4 Potentiometer Calibration : Joint #3

Appendix F

Configuration File

```
!
! TWIN.CFG          Configuration file for CIC-IWC
!
! Hardware
processor host              !IBM-PC
processor root              !Root transputer running filter
processor ioserv            !Additional transputer for mux
processor cpc               !CPC processor for task cpctwin
processor jsc260            !Joint Space Controller Puma 260
processor jsc560            !Joint Space Controller Puma 560
processor cubspl            !Processor for task intpol
!
! Connection between MUX and other tasks for debugging and I/O
!
wire ? host[0] root[0]          !Link connection between IBM-PC and root
wire ? root[2] ioserv[1]        !Link connection between filter and mux
wire ? ioserv[0] jsc560[3]      !Link connection between mux and puma560 (debugging)
wire ? ioserv[3] jsc260[3] !Link connection between mux and puma260 (debugging)
wire ? ioserv[2] cpc[1]         !Link connection between mux and cpctwin
!
! Connection between CPCTWIN and other tasks for control
!
wire ? cpc[0] jsc560[0]         !Link connection between cpctwin and puma560
wire ? cpc[2] jsc260[1]         !Link connection between cpctwin and puma260
wire ? cpc[3] cubspl[3]         !Link connection between cpctwin and intpol
!
! Links used for communication with S/P Interface
!
! jsc260[0] - edge connector - S/P Puma 260
! jsc560[1] - edge connector - S/P Puma 560
!
! Task declarations indicating channel I/O
!
task afserver ins=1 outs=1
task filter ins=2 outs=2
task multiplexer file=filemux ins=4 outs=4
task cpctwin ins=5 outs=5
```

```
task pums260 ins=3 outs=3
task puma560 ins=3 outs=3
task intpol ins=1 outs=1
!
! Assign tasks to physical processors
!
place afserver host
place filter root
place multiplexer ioserv
place cpctwin cpc
place puma260 jsc260
place puma560 jsc560
place intpol cubspl
!
! Set up communication between tasks
!
connect ? filter[0] afserver[0]
connect ? afserver[0] filter[0]

connect ? filter[1] multiplexer[0]
connect ? multiplexer[0] filter[0]

connect ? multiplexer[1] cpctwin[1]
connect ? cpctwin[1] multiplexer[1]

connect ? multiplexer[2] puma260[1]
connect ? puma260[1] multiplexer[2]

connect ? multiplexer[3] puma560[1]
connect ? puma560[1] multiplexer[3]

connect ? cpctwin[2] puma260[2]
connect ? puma260[2] cpctwin[2]

connect ? cpctwin[3] puma560[2]
connect ? puma560[2] cpctwin[3]

connect ? cpctwin[4] intpol[0]
connect ? intpol[0] cpctwin[4]
```

The corresponding *ncs connect file* is

```
s1 0 t s9 3.
s1 3 t s3 3.
s2 0 t s9 0.
s2 3 t s4 3.
s3 0 t e0 .
```